5-2023

Aerodynamic Implications of a Bio-Inspired Rotating Empennage Design for Control of a Fighter Aircraft

Christian R. Bolander
Utah State University

AERODYNAMIC IMPLICATIONS OF A BIO-INSPIRED ROTATING EMPENNAGE DESIGN FOR CONTROL OF A FIGHTER AIRCRAFT

by

Christian R. Bolander

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Aerospace Engineering

Approved:

Douglas F. Hunsaker, Ph.D. Major Professor	James Joo, Ph.D. Committee Member
Tianye He, Ph.D. Committee Member	Matthew Harris, Ph.D. Committee Member
Stephen Whitmore, Ph.D. Committee Member	D. Richard Cutler, Ph.D. Vice Provost for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2023

ABSTRACT

Aerodynamic Implications of a Bio-Inspired Rotating Empennage Design for Control of a Fighter Aircraft

by

Christian R. Bolander, Doctor of Philosophy

Utah State University, 2023

Major Professor: Douglas F. Hunsaker, Ph.D.
Department: Mechanical and Aerospace Engineering

Future tactical aircraft will likely demonstrate improvements in efficiency, weight, and control by implementing bio-inspired control systems. This work explores the aerodynamic implications of a novel control effector for a fighter aircraft inspired by the function of, and the degrees of freedom available in, a bird's tail. Specifically, the control effector in this work is introduced into a fighter aircraft by removing the vertical tail and allowing the horizontal tail surfaces to rotate about the centerline of the aircraft. The geometry of the baseline fighter aircraft and its bio-inspired variant are first defined using open-source geometric data and estimates from published drawings of the baseline aircraft. To analyze the aerodynamic forces and moments acting on each aircraft, two aerodynamic models are constructed, one for each aircraft, based on a linearized model, augmented with certain non-linear effects. The coefficients in each aerodynamic model are then calculated using a numerical lifting-line algorithm and the aerodynamic effects of the rotating tail control effector are characterized. As a result of constructing these models the bio-inspired aircraft was shown to exhibit trade-offs with longitudinal and lateral control. In an effort to better understand these trade-offs, a trim analysis is performed for two static trim conditions: the steady, coordinated turn and steady-heading sideslip. This analysis shows that the bio-inspired aircraft has no appreciable reduction in trim envelope when compared to the baseline aircraft and has a larger trim envelope in steady-heading sideslip. Additionally, the static control authority available to each aircraft is compared to identify the longitudinal and lateral controlling moment trade-offs. The control authority of the bio-inspired aircraft is larger than the baseline aircraft, except when large pitching and yawing moments are coupled. Finally, a linearized state-feedback controller is developed using linear quadratic regulation and applied in simulation to each aircraft in the presence of a wind gust and the robustness of the controller is determined by sweeping through approximately 1300 gust cases. With the linearized feedback controller, the bio-inspired aircraft was shown to reject gust disturbances in all of the cases studied.

(491 pages)

PUBLIC ABSTRACT

Aerodynamic Implications of a Bio-Inspired Rotating Empennage Design for Control of a Fighter Aircraft

Christian R. Bolander

This dissertation presents an analysis of the aerodynamics for an aircraft using a novel, bio-inspired control system. The control system is a rotating tail, that is inspired by the way in which birds use their tail to control their flight. An aerodynamic model for a baseline aircraft and a bio-inspired variant are created by referencing well-known relationships for the aerodynamics of flight, which are then used to analyze the available flight envelope at which each aircraft can reach two different equilibrium states. An analysis of the total aerodynamic control authority of each aircraft is also included along with a preliminary control system to bring the aircraft back to equilibrium when influenced by a wind gust. These studies indicate some of the benefits and trade-offs of using this bio-inspired rotating tail design.

"If ye labor with all your might, I will consecrate that spot that it shall be made holy."
- Doctrine and Covenants 124:44

ACKNOWLEDGMENTS

This work has been an effort that would not be possible with just my capabilities alone. I would like to, first and foremost, thank my wonderful wife, Beth, and our children, Emma, James, and Kezia. They have provided untold support to me during this time, and I'm sure that James will always assume that everyone works on a dissertation at some point in their life.

I am also incredibly grateful for the support of my advisor, Dr. Doug Hunsaker. He has recognized and acknowledged my efforts throughout the journey. Those acknowledgements have been lodestones to me, pointing me in the right direction and helping me to improve and progress. I am also grateful for Dr. James Joo and his support throughout this project, both through funding from the Air Force Research Lab as well as his guidance in improvements that could be made to the project. My current supervisor, Dr. Thom Fronk, has also been an great support to me throughout the final hours of my dissertation writing, and I am grateful for his thoughtfulness and patience with the effort that it has required. My wonderful parents, siblings, colleagues, and friends; I am grateful for each of you and know that you share in any sense of accomplishment that I feel in presenting this work to the world.

Finally, I am grateful to God, and I would be remiss not to include Him here. My faith has been a bedrock foundation for me throughout this process, and I acknowledge that His help has brought me to where I am today.

Christian R. Bolander

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In their work investigating the challenges and opportunities in tailless aircraft stability and control, Bowlus et al. outlined the aerospace research and engineering goals of the office of the Deputy Director of Research and Engineering (DDR&E) in the Department of Defense (DoD) [1]. These goals, as written in 1997, were to "reduce engineering, manufacturing, and development costs, reduce production costs, reduce operation and support costs, reduce airframe weight, improve aircraft lift to drag ratio, and improve aircraft agility." The vision of the Directorate of Defense Research and Engineering for Research and Technology (DDR&E(R&T)) at the time of writing this work more broadly speaks of creating "far-reaching technology innovations and set them on a trajectory to create U.S. military technical advantage [2]." Certainly there exists significant overlap between these goals and the benefits of developing tailless aircraft technologies, such as those investigated by Bowlus et al., would serve to provide technical advantage to the U.S. military today [1].

The work by Bowlus et al. focused on achieving these types of "far-reaching technology innovations" through the development of tailless "flying wing" designs [1]. Their work outlined some of the challenges with tailless aircraft designs, which include: the generation of yaw control power, multi-axis instabilities, optimization of multiple control surfaces for any given axis, and nonlinearity and coupling between control surfaces. From these challenges, we see that one of the primary problems facing the development of tailless aircraft is that of identifying robust, adequately-powered control systems that can provide sufficient stability to a tailless aircraft. Various control systems were explored by Bowlus et al. that demonstrated the ability to address these challenges; however, each of the systems was reliant on coupled supplementary control systems to satisfy control requirements [1].

While many of the conventional approaches to tailless designs are becoming increasingly complex, birds have demonstrated incredible control authority through variable flight conditions without a vertical control surface. Many bird species, such as the albatross shown in Fig. 1.1, have large, high-aspect-ratio wings with a small tail similar to a flying-wing design. Other species, such as the swallow-tailed kite in Fig. 1.2, have large tails that are used throughout flight in a manner consistent with an active control system. Many UAVs employing tail control systems inspired by birds such as the kite are able to produce similar levels of lateral stability and control as a traditional empennage while providing drag reduction [3]. Further research into the area of bio-inspired flight control systems such as these will likely be one of the ways in which the goals of the DDR&E(R&T) and the Department of Defense are achieved in coming years.

Fig. 1.1: An albatross in flight closely represents a flying wing design [4]. Photograph: Max Allen/Alamy.

In 1996, as part of the development of next-generation fighter aircraft, the Department of Defense sought proposals that outlined novel control systems for tailless aircraft. Both Boeing [6] and Lockheed Martin [7] submitted preliminary results as part of their proposals for this Innovative Control Effectors (ICE) contract. Roetman et al. [6] studied a control design that removed the vertical stabilizer while allowing the horizontal tail to rotate for lateral control similar to the swallow-tailed kite in Fig. 1.2. In spite of the intent of this

Fig. 1.2: The swallow-tailed kite uses its tail to control itself while gliding [5]. Photograph: Andy Morffew.

tail control system, the rotating horizontal stabilizer examined by Roetman et al. [6] was limited to analysis at a specific dihedral angle, effectively representing a V-tail. Part of this limitation was attributed to the fact that rotating the tail introduced multiple trim solutions to a given flight condition, which their trim algorithms were not prepared to consider effectively. Nevertheless, they found that this control system was "effective throughout the flight envelope of interest" and appeared "to be a viable concept being nearly as effective as the baseline [aircraft]", in addition to providing potential weight reduction and reduced aerodynamic complexity [6]. Nevertheless, Roetman et al. spent the majority of their analysis on their "rotating" horizontal tail and showed that the handling qualities were favorable when compared to the other designs they analyzed [6].

Although simply a V-tail in practice, the value in a rotating tail control system such as that proposed by Roetman et al. [6] is largely in its simplicity. As an extension of the work begun by Roetman et al. [6], this dissertation proposal will propose research on a Bio-Inspired Rotating Empennage (BIRE) control system. In addition, we will present a consistent nomenclature for bird tail morphology that will be used throughout a review of the literature. The literature will be segmented into two parts, the first of which will focus on the control offered to birds by rotating their tails. The second section of literature will present results outlining the aerodynamic implications of using a rotating tail control system on an aircraft. The information in each of these sections will be tied to traditional flight mechanics relationships for stability and control that are presented by Phillips [8].

Once the current understanding of the effect on aerodynamics and control is established from the literature, the analysis in this work is developed as follows. First, the geometry of a baseline fighter aircraft is determined using publicly-available data and scaled estimations from published figures. The necessary geometric modifications required to represent the BIRE can then be made on this baseline model. Following the geometry definition, a model describing the aerodynamic forces and moments acting on each aircraft is given. These models are based upon a traditional linearized coefficient model and then augmented with certain non-linear effects based on familiarity with the aircraft aerodynamics and understanding from analytical studies. The coefficients in each model will then be evaluated using data from a numerical lifting-line method.

With the aerodynamic model defined for each aircraft, several studies are performed to better understand the aerodynamic and control implications of the BIRE. The first of these studies is a static trim analysis that presents data indicating the effect of implementing the rotating empennage on the trim envelope of the aircraft. Data from this analysis can be compared to the trim envelope of the baseline aircraft for comparison. This static trim analysis also allows for a preliminary understanding of the risk of tail strike posed by the BIRE when landing in a crosswind.

The static control authority of the baseline aircraft and BIRE are then compared using an attainable moment set analysis. This study looks at the aerodynamic moments that can be produced by the tail while maintaining control about another axis (in this case the pitch axis). In particular, this study indicates the intuitive trade-offs embraced by implementing a rotating tail as a control effector.

Finally, a preliminary control law is presented for both the baseline aircraft and the BIRE. The control law is developed for a linearized, rigid body system using state-feedback and a linear quadratic regulator and is intended to stabilize the aircraft in the presence of a gust disturbance. Simulations of each aircraft employing the state-feedback control law are shown alongside a time-domain robustness study to examine the effectiveness of the control in a variety of gusting conditions.

This dissertation lays a foundation of tools that can be used to further explore the implications of a rotating tail design. Additional research will allow many of the questions that are currently outstanding in the literature review to be better understood. Further, these tools and analyses can stand as a foundation for future research efforts into rotating tail designs. By examining the BIRE as is presented in this work, the concerns highlighted by Bowlus et al. [1] can be addressed and future work on this subject can be enhanced.

CHAPTER 2

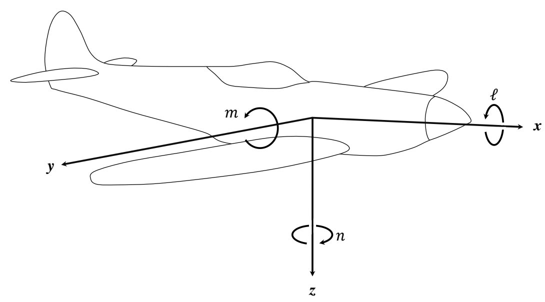
LITERATURE REVIEW

As outlined in the Introduction, this chapter examines what is known about the role of the avian tail in controlling a birds' flight and what has been learned about controlling aircraft using a rotating tail. The literature in each of these sections is placed in the context of known flight mechanics relationships based on simplified models. Using these relationships, intuition can be built to help understand the expected nature of applying a rotating tail control effector to an aircraft. Before investigating these pieces of literature, a consistent nomenclature must be established for both the flight mechanics relationships as well as avian tail morphology.

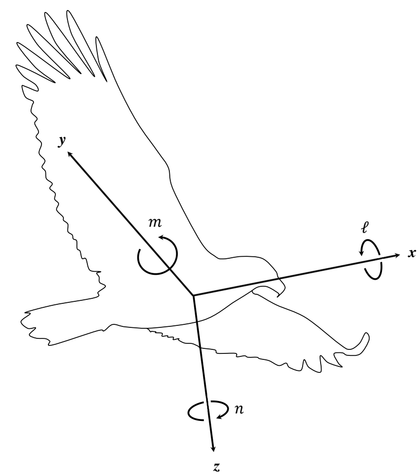
Throughout this dissertation, the word "control" will be used in a variety of ways. The "control system" of an aircraft can generally be described as a combination of the desired state of an aircraft, any measurements of the aircraft states, a control law, and the means of actuation of the system. In addition to the description of the control system, this literature review will frequently refer to longitudinal or lateral control. In this context, control refers to the aerodynamic forces and moments being manipulated by the aircraft to direct its flight and will be referenced as "aerodynamic control". An effort will be made to distinguish between these interconnected definitions referencing control throughout this work.

2.1 Longitudinal and Lateral Degrees of Freedom

The aerodynamic forces and moments acting on an aircraft or a bird can be defined within the body-fixed, six degree-of-freedom reference system shown in Fig. 2.1. These six degrees of freedom can be sub-divided into two categories: *longitudinal* and *lateral* degrees of freedom. The longitudinal degrees of freedom are those within the x-z plane of Fig. 2.1 and the three remaining degrees are called the lateral degrees of freedom.



(a)



(b)

Fig. 2.1: The body-fixed, six degree-of-freedom coordinate system applied to an aircraft (a) and a bird (b).

The total aerodynamic force acting on an aircraft, such as the one in Fig. 2.1, can be separated into components along each of the body-fixed axes. These forces are named after the body-fixed axis along which they lie: the body-fixed X force, the body-fixed Y force, and the body-fixed Z force. Likewise, the total aerodynamic moment acting on the aircraft can be separated into right-hand moments acting about each axis. These are the rolling moment ℓ, the pitching moment m, and the yawing moment n about the x-, y-, and z-axes, respectively. According to our earlier definition, the aerodynamic force components along the x- and z-axes and the aerodynamic moment component about the y-axis are the longitudinal forces and moment. The only remaining force component lies along the y-axis, while the remaining moments are about the x- and z-axes; these constitute the lateral force and moments.

The body-fixed system shown in Fig. 2.1 is not the only reference system that is commonly defined for a body in flight. It is sometimes helpful to use a coordinate system aligned with the atmospheric wind, called the wind coordinate system. The wind coordinate system is similar to the body-fixed system, the exception being that the wind x-axis is parallel with, but in the opposite direction to, the wind. The longitudinal forces in the wind system are referred to as the lift and drag forces (L and D), while the lateral force is referred to as the side force (S).

In a symmetric aircraft in level flight, the longitudinal and lateral forces and moments are very nearly decoupled. In addition, the control surfaces on the wing and tail are primarily responsible for one of the three aerodynamic moments. That is, the elevators on the horizontal tail primarily generates a pitching moment, the ailerons on the main wing primarily generates a rolling moment, and the rudder on the vertical tail primarily generates a yawing moment. The rotating tail of the BIRE will produce substantial coupling between imposed pitching and yawing moments that is not present in the control-surface combination of a traditional aircraft. A planar horizontal tail will produce nearly a pure pitching moment; however, as the tail rotates about the empennage, a combination of pitching and yawing moments will occur. This coupling between the lateral and longitudinal forces and moments will be the subject of much of the discussion of the effects of tail rotation in both birds and aircraft.

Before covering the nomenclature and degrees of freedom that will be used when discussing avian tail morphology, we will briefly review longitudinal and lateral aerodynamic control and stability. We will cover the ways in which stability and aerodynamic control effectiveness is measured and present analytical relationships between aircraft geometry, aerodynamic properties, and stability and aerodynamic control. These relationships are not exact, but will give intuitive insight into the effects that the tail degrees of freedom will have on longitudinal and lateral stability and aerodynamic control.

Longitudinal Stability and Aerodynamic Control

In traditional aircraft, the area of the horizontal tail is directly related to the pitch stability and control effectiveness [9,10]. A linear aerodynamic model can be used to provide a relationship between tail properties and the pitch stability of a traditional aircraft below stall as

$$C_{m,\alpha} \propto -\mathcal{V}\, C_{L_h,\alpha} \tag{2.1}$$

where $C_{m,\alpha}$ is the pitch stability, $\mathcal{V}$ is the tail volume coefficient, and $C_{L_h,\alpha}$ is the lift slope of the tail. The tail volume coefficient is a measure of tail "volume" to wing "volume" and is defined as

$$\mathcal{V} \equiv \frac{S_h l_h}{S_w \bar{c}_w} \tag{2.2}$$

where l_h is the distance from the center of gravity to the aerodynamic center of the tail, $\bar{c}_w$ is the mean chord of the main wing, S_h is the horizontal area of the tail, and S_w is the horizontal area of the main wing. An aircraft is longitudinally stable in pitch if $C_{m,\alpha} < 0$ and therefore any increase in the tail volume coefficient will have a stabilizing effect on the aircraft [9].

The previous paragraph has covered the role of the tail in providing *static* longitudinal stability, but the tail also has an effect on the *dynamic* longitudinal stability of an aircraft as well. Longitudinal motion consists of two dynamic modes, called the *short-period* and *long-period* or *phugoid* modes. These modes are associated with two pairs of oscillatory eigenvalues and will be stable when the real part of those eigenvalue pairs is negative. When these dynamic modes are stable, any disturbances to the aircraft will be damped out naturally by the aircraft.

The short-period dynamic mode of an aircraft is characterized by rapid changes in angle of attack and altitude [11]. By assuming that the longitudinal and lateral degrees of freedom are decoupled and that the angle of attack and pitch rate are decoupled from the other longitudinal states, a closed-form solution can be produced for the dimensionless eigenvalues of the short period mode [12]. A relationship between the properties of the tail and the real part of the dimensionless eigenvalue associated with the short-period mode is given by

$$\mathrm{Re}\left(\lambda_{\mathrm{sp}}\right) \propto \frac{1}{I_{yy_b}} C_{m,\bar{q}} \tag{2.3}$$

where I_{yy_b} is an inertial term and $C_{m,\bar{q}}$ is the pitch damping derivative.

Using the linear aerodynamic model, the relationship between the pitch damping derivative and the properties of the tail can be written as

$$C_{m,\bar{q}} \propto -\mathcal{V}\,\frac{l_h}{\bar{c}_w} C_{L_h,\alpha} \tag{2.4}$$

and will always be negative with an aft tail. From Eqs. (2.3) and (2.4), we can see that increasing the tail volume coefficient, moving the aerodynamic center of the tail further aft, or increasing the tail lift slope will have a stabilizing effect on the short-period mode of the aircraft.

The long-period or phugoid dynamic mode of an aircraft is characterized by slow changes in airspeed, altitude, and elevation angle with very little change in angle of attack [11]. Under the assumption that the change in angle of attack is zero and assuming further that the forward velocity and elevation angle are decoupled from the other longitudinal terms, a closed-form solution for the dimensionless eigenvalues of the phugoid can be produced [13]. The relationship between the real part of the dimensionless phugoid eigenvalue and the tail properties is given by [13]

$$\mathrm{Re}\left(\lambda_{\mathrm{lp}}\right) \propto \frac{1}{I_{yy_b}} \frac{C_{m,\bar{q}}}{C_{m,\alpha} + C_{m,\bar{q}}} \tag{2.5}$$

Equation (2.5) shows that increasing the magnitude of the pitch damping derivative or increasing the pitch stability will providing a stabilizing contribution to the phugoid mode of an aircraft.

An aircraft, or any body in flight, can measure its longitudinal aerodynamic control by its ability to both trim and maneuver in the longitudinal plane across relevant flight conditions [14]. In terms of a traditional aircraft, this means that an aircraft with greater longitudinal aerodynamic control authority will produce more pitching moment per degree of elevator deflection than another. Using the same linear aerodynamic model as before, the relationship between tail properties and the pitch control derivative of a traditional aircraft below stall is

$$C_{m,\delta_e} \propto \mathcal{V} \left(\frac{\overline{c}_h}{l_h} C_{m_h,\delta_e} - C_{L_h,\alpha} \right) \tag{2.6}$$

where C_{m,δ_e} is the pitch control derivative, $\overline{c}_h$ is the mean chord of the tail, and C_{m_h,δ_e} is the pitch control derivative of the tail alone.

A positive deflection of the elevator moves the trailing-edge of the horizontal tail downward. This acts to increase the lift on the trailing-edge of the tail, which produces a negative pitching moment. Therefore, the pitch control derivative of the tail for a conventional aircraft will generally be negative [15]. This same logic applies to the pitch control derivative of the aircraft as a whole. From Eq. (2.6), any negative decrement to the pitch control derivative should act to increase the longitudinal aerodynamic control authority or effectiveness. Since $C_{m_h,\delta_e} < 0$ and $C_{L_h,\alpha} > 0$ in general, an increase in the magnitude of either of these coefficients, an increase in tail volume coefficient, or an increase in the mean chord of the tail will increase the longitudinal aerodynamic control effectiveness of the aircraft [16].

Lateral Stability and Aerodynamic Control

The lateral degrees of freedom are characterized by two aerodynamic moments, the rolling moment and the yawing moment. Therefore, instead of simply referring to lateral stability and control, the distinction will be made between roll stability and control and yaw stability and control. The relationship between the properties of the tail and roll stability will be addressed before moving to relationships concerning roll control. This will be followed by the relationships between tail properties and yaw stability, before finally outlining the relationships concerning yaw control.

Static roll stability is fundamentally different than both pitch stability and yaw stability, since rolling motions are directly related to the orientation of the aircraft through the bank angle ϕ rather than the wind through the aerodynamic angles, α and β. However, when an aircraft is oriented at some bank angle, any degree of sideslip will affect the rolling moment. Thus roll stability is described by the gradient $C_{\ell,\beta}$.

Using the same linear aerodynamic model used to analyze the effects of the tail on pitch, the relationship between the properties of the tail and the roll stability of a traditional aircraft below stall is given by

$$C_{\ell,\beta} \propto -\mathcal{V}_v \frac{1}{l_v} \left(h_v - b_h\right) C_{L_v,\alpha} \tag{2.7}$$

where S_v is the area of the vertical tail, h_v distance in the negative body-fixed z-direction between the aerodynamic center of the vertical tail and the center of gravity, $C_{L_v,\alpha}$ is the lift slope of the vertical tail, and $\mathcal{V}_v$ is called the *vertical tail volume coefficient*, which is defined as

$$\mathcal{V}_v = \frac{S_v l_v}{S_w b_w} \tag{2.8}$$

This ratio of the vertical tail "volume" to the wing "volume" is analogous to the tail volume coefficient defined in Eq. (2.2). Note that Eq. (2.7) assumes that the horizontal tail is positioned below the vertical tail. An aircraft is stable in roll if $C_{\ell,\beta} < 0$ [17]. Thus, with the vertical tail above the center of gravity, $h_v > 0$, and the vertical tail provides a stabilizing contribution to the roll stability. On the other hand, the horizontal tail is destabilizing if located below the vertical tail and stabilizing otherwise.

The lateral motion of an aircraft consists of three dynamic modes: the *roll* mode, the *spiral* mode, and the *Dutch roll* mode. Of these three modes, both the roll and spiral mode generally consist of purely real eigenvalues, while the Dutch roll mode is generally a complex eigenvalue pair. As before, so long as the real portion of the eigenvalue associated with each of these modes is negative, the aircraft system will be dynamically stable and any disturbances to the system will be damped out with time. The roll is primarily related to dynamic stability about the roll axis, while roll and yaw are integrated into both the spiral and Dutch roll modes. Therefore, both of these modes will be discussed after static yaw stability is introduced.

The roll mode of an aircraft is generally characterized in terms of the time required for a rolling motion to approach a constant rate [18]. This rolling motion is generally initiated by the ailerons. An estimate for the real eigenvalue corresponding to the roll mode is [19]

$$\lambda_{\mathrm{r}} \propto \frac{1}{I_{xx_b}} C_{\ell,\overline{p}} \tag{2.9}$$

where I_{xx_b} is an inertial term and $C_{\ell,\overline{p}}$ is the roll damping derivative.

The roll damping derivative is primarily influenced by the main wing, rather than the tail. An estimate of the contribution of the wing to the roll damping derivative can be made using lifting-line theory as developed by Prandtl [20]. An extension of this estimate can be used to find the relationship between the roll damping derivative and the properties of the tail as

$$C_{\ell,\overline{p}} \propto -b_h C_{L_h,\alpha} - b_v C_{L_v,\alpha} \tag{2.10}$$

Considering both Eq. (2.9) and Eq. (2.10), an increase of either the span or lift slope of both portions of the tail will result in an increase in the magnitude of the roll damping derivative and will provide a stabilizing contribution to the roll mode of an aircraft.

As the aileron is primarily responsible for producing rolling moments, the main wing contributes most substantially to the roll control derivative of an aircraft. This traditional roll control derivative is defined as C_{ℓ,δ_a} and it will be negative when a positive aileron deflection deflects the trailing-edge of the right aileron downward and the trailing-edge of the left aileron upward. In the absence of differential tail deflections patterned after the ailerons, the only other contribution from the tail to the aerodynamic control of the rolling moment is induced by rudder deflections on the vertical tail. While primarily responsible for creating yawing moments, the offset in the body-fixed z-direction of the rudder from the center of gravity creates a moment arm for any force generated by the rudder. Thus, the rudder will provide a substantial rolling moment when it is deflected.

From the linear aerodynamic model, the relationship between the properties of the tail and the roll control derivative *induced by the rudder* in a traditional aircraft below stall is given by

$$C_{\ell,\delta_r} \propto \mathcal{V}_v \frac{1}{l_v} \left(h_v - b_h\right) C_{L_v,\alpha} \tag{2.11}$$

for a horizontal tail mounted below the vertical tail. To better understand the implications of Eq. (2.11), we will consider an aircraft in a banked turn to the left. A turn to the left requires a negative rolling moment and a negative yawing moment according to the body-fixed system given in Fig. 2.1. These moments can be generated by a positive aileron deflection and a positive rudder deflection. Therefore, the roll control derivative predicted by Eq. (2.11) will support the roll control derivative of the main wing only if its value is also negative. From Eq. (2.11) we can see that this would require the quantity $h_v - b_h$ to be negative, corresponding to a relatively large horizontal tail and a relatively small vertical tail. In the context of turning flight with a traditional aircraft, therefore, the induced control derivative given in Eq. (2.11) will counteract the roll control derivative of the aircraft.

In an aircraft, the vertical tail provides the majority of the lateral stability, much like the presence of the horizontal tail provides the largest contribution to longitudinal stability. The same linear aerodynamic model used previously can be used to provide a relationship between the properties of the tail and the yaw stability of a traditional aircraft below stall as [21]

$$C_{n,\beta} \propto \mathcal{V}_v C_{L_v,\alpha} + \mathcal{V}\frac{b_h}{l_h}\left(C_{D_h,\beta} + \tan\Lambda_h C_{D_h}\right) \tag{2.12}$$

where C_{D_h} is the drag on the horizontal tail, $C_{D_h,\beta}$ is its derivative with respect to sideslip, and Λ_h is the sweep angle of the horizontal tail. Due to the definition of the sideslip angle, an aircraft is statically stable in yaw if $C_{n,\beta} > 0$ [21]. Thus, an increase in either tail volume coefficient, the tail drag or its derivative, the horizontal tail sweep angle, or the lift slope of the vertical tail will provide a stabilizing contribution to the yaw stability of a traditional aircraft.

As stated previously, the lateral dynamic motion has two modes related to stability about the yaw axis. The spiral mode is characterized by a change in heading or direction of travel [18]. By assuming the sideslip, roll, and yaw accelerations are small, the lateral equations of motion can be solved for a single first-order differential equation with respect to the bank angle [22]. From this analysis, a relationship between the properties of the tail and the dimensionless eigenvalue of the spiral model can be established as

$$\lambda_{\mathrm{s}} \propto \frac{C_{\ell,\beta}C_{n,\overline{r}} - C_{\ell,\overline{r}}C_{n,\beta}}{C_{\ell,\beta}C_{n,\overline{p}} - C_{\ell,\overline{p}}C_{n,\beta}} \tag{2.13}$$

In Eq. (2.13), we note the presence of the roll stability, yaw stability, and roll-damping derivatives. Also included are two cross-damping terms, $C_{\ell,\overline{r}}$ and $C_{n,\overline{p}}$, along with the yaw-damping derivative, $C_{n,\overline{r}}$.

The relationship between the tail properties and the yaw cross-damping term $C_{n,\overline{p}}$ can be estimated using the same process as $C_{\ell,\overline{p}}$ as [20]

$$C_{n,\overline{p}} \propto -\left(1 - \frac{1}{R_{A_h}}C_{L_h,\alpha}\right) b_h C_{L_h} - \left(1 - \frac{1}{R_{A_v}}C_{L_v,\alpha}\right) b_v C_{L_v} \tag{2.14}$$

where R_{A_h} is the aspect ratio of the horizontal tail and R_{A_v} is the aspect ratio of the vertical tail. Similarly, the relationship between the tail and the roll cross-damping term $C_{\ell,\overline{r}}$ can be estimated as [20]

$$C_{\ell,\overline{r}} \propto \mathcal{V}_v h_v C_{L_v,\beta} + \left(1 - \frac{1}{R_{A_h}}C_{L_h,\alpha}\right) b_h C_{L_h} \tag{2.15}$$

Finally, the relationship between the yaw-damping derivative and the properties of the tail can be estimated by [20]

$$C_{n,\overline{r}} \propto \mathcal{V}_v h_v C_{L_v,\beta} \tag{2.16}$$

The relationship between these damping terms and the stability of the spiral mode is more complex than was revealed by examining the longitudinal modes and the roll mode. Each of the damping terms are inter-related through the tail volume ratio and the lift produced by the tail. In general, a vertical tail mounted above the horizontal tail should have $C_{\ell,\overline{r}} > 0$ and $C_{n,\overline{r}} < 0$. The analysis that follows will reveal that stabilizing the spiral mode is coupled significantly with stabilization of the Dutch roll mode.

The Dutch roll mode is characterized by an oscillatory combination of roll, yaw, and sideslip [18]. An approximation for the real part of the eigenvalue for this dynamic mode is included in Phillips [23]

$$\mathrm{Re}\left(\lambda\right)_{\mathrm{Dr}} \propto C_{Y,\beta} + C_{n,\bar{r}} - R_{D_c} + R_{D_p} \tag{2.17}$$

where

$$R_{D_c} \propto \frac{C_{\ell,\bar{r}} C_{n,\bar{p}}}{C_{\ell,\bar{p}}} \tag{2.18}$$

$$R_{D_p} \propto \frac{C_{\ell,\bar{r}} C_{n,\beta} - C_{\ell,\beta} C_{n,\bar{r}}}{C_{\ell,\bar{p}}\left(C_{n,\beta} + C_{Y,\beta} C_{n,\bar{r}}\right)} - \frac{R_{D_s}}{C_{\ell,\bar{p}}} \tag{2.19}$$

and

$$R_{D_s} \propto -\frac{C_{\ell,\beta}(1 - C_{Y,\bar{r}}) C_{n,\bar{p}} + C_{Y,\beta} C_{\ell,\bar{r}} C_{n,\bar{p}}}{C_{\ell,\bar{p}}} \tag{2.20}$$

For a traditional, statically stable aircraft, $C_{Y,\beta} < 0$, $C_{Y,\bar{r}} > 0$, $C_{\ell,\beta} < 0$, $C_{\ell,\bar{r}} > 0$, and $C_{n,\beta} > 0$. Furthermore, the roll-damping and the yaw-damping derivative are negative according to Eqs. (2.10) and (2.16). Thus, $C_{Y,\beta}$ and $C_{n,\bar{r}}$ should provide stabilizing contributions to the Dutch roll mode. The other terms in Eq. (2.17) are more complex and must be examined on a case-by-case basis, just like the stability of the spiral mode. Specifically, increasing the magnitude of one of the stability derivatives will likely also change several of the damping derivatives and provide difficulty in completely stabilizing both modes simultaneously [23].

Finally, using the linear aerodynamic model as before, the relationship between the properties of the tail and the yaw control derivative in a traditional aircraft below stall is given by [21]

$$C_{n,\delta_r} \propto -\mathcal{V}_v \left(C_{L_v,\alpha} - \frac{\bar{c}_v}{l_v} C_{m_v,\delta_r} \right) \tag{2.21}$$

Since the aerodynamic derivative C_{m_v,δ_r} is always negative, the yaw control derivative is always negative for an aft rudder with a positive rudder deflection defined to the left [21]. Also, increasing the vertical tail volume coefficient, the lift slope of the vertical tail, or the moment derivative of the rudder will increase the yaw control authority available to a traditional aircraft below stall.

2.2 Avian Tail Morphology

The nomenclature adopted to discuss avian tail morphology across the studies mentioned in the forthcoming literature review can vary substantially. To ensure that consistency is retained in both morphology definitions and in the relationship between changes in tail morphology and aerodynamics, a common nomenclature will be defined here. Figure 2.2 shows each of the tail morphing capabilities that will be highlighted from the literature: tail spread, tail incidence, and tail rotation. While this dissertation focuses on a control system that incorporates tail rotation and incidence only, variations in tail spread are generally included in many UAV designs that incorporate a rotating tail. Additionally, birds often couple tail rotation with both tail spread and tail incidence and, thus, decoupling these tail degrees of freedom in our treatment of the literature would neglect some key characteristics of rotating the tail during flight.

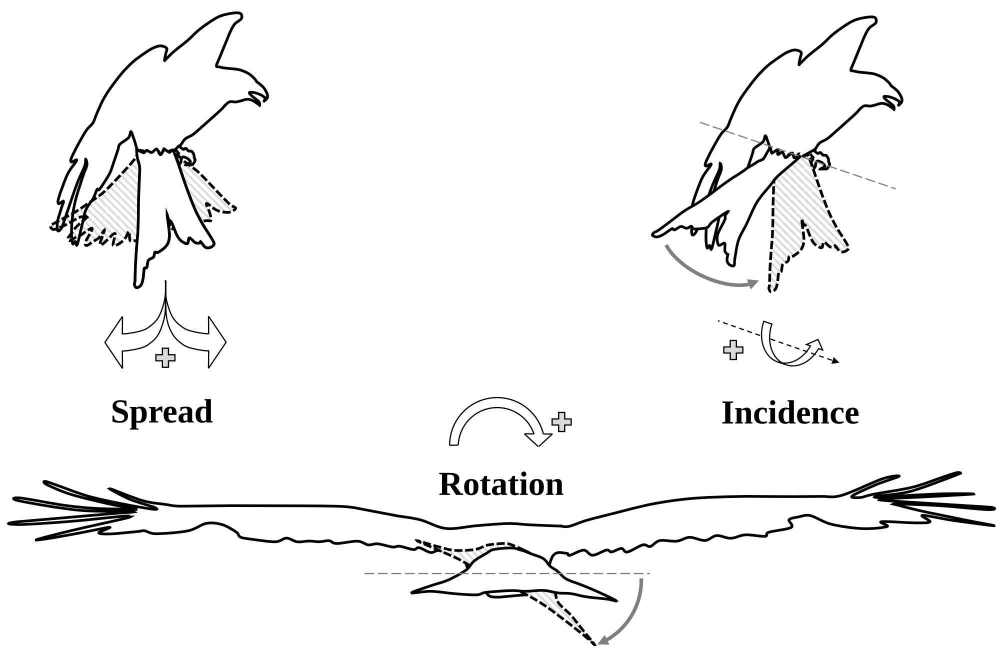


Fig. 2.2: Nomenclature and sign conventions for the avian tail morphing mechanisms studied in this work.

Included with the tail morphology nomenclature in Fig. 2.2 is a representation of an increase in each tail morphing degree of freedom. From Fig. 2.2, we see that an increase in the spread angle corresponds to an increase in the area of the tail. Likewise, an increase in tail incidence corresponds to downward deflection of the tail about its base, while an increase in tail rotation corresponds to a solid-body, clockwise rotation of the tail when viewed from behind the bird.

By manipulating the spread of their tails, birds are able to control the area of their tail mid-flight. We can see by examining Eqs. (2.1)–(2.21) that the horizontal and vertical components of the tail have a substantial impact on stability and control in a traditional aircraft. Changes in tail incidence are often considered synonymous to the elevator deflection of a traditional aircraft, acting to change the camber of the tail and thereby change the zero-lift angle of attack of the tail without substantially affecting the lift slope. Again, the relationships established in Section 2.1 tell us that manipulating the camber line, and therefore properties such as the lift and moment acting on the tail, will have an impact on several of the stability, damping, and control characteristics of a traditional aircraft.

Finally, tail rotation represents a way for birds to control the ratio of vertical to horizontal tail area throughout their flight. Within the context of the relationships given in Section 2.1, we see that tail rotation allows for the longitudinal and lateral degrees of freedom to be coupled by trading off the tail volume coefficients $\mathcal{V}$ and $\mathcal{V}_v$. Tail rotation as defined here is different than tail twist, which is not a solid-body motion, but rather a spanwise variation in tail geometry [24]. Since the amount of literature that details the effects of tail twist in birds is rather limited, such studies will be omitted from this review; however, they can be found in the work by Harvey, Gamble, and Bolander et al. [3].

2.3 The Role of the Tail in Avian Flight Control

As a precursor to this portion of the literature, I make special mention to the reader that the BIRE design is purposely labeled a bio-*inspired* design, rather than a bio-*mimetic* design. Understanding the benefits offered to an aircraft by incorporating such a design does not necessarily translate into an understanding of how birds use their tail in flight. That information can only be obtained by studying the birds themselves; and, even then, the results will likely vary substantially among different species.

It is also important to mention that there are numerous instances in which birds have been observed to couple wing and tail morphing to control their flight [3, 25–33]. Since the BIRE design primarily uses the tail to generate forces and moments, the literature presented here will focus solely on the role of the tail in flight control. The reader is referred to the review presented by Harvey, Gamble, and Bolander et al. [3] for more details on wing-tail coupling in avian flight control.

2.3.1 Longitudinal Stability and Control

Where appropriate throughout this section, we will use the relationships in Eqs. (2.1)–(2.6) to identify how changes in tail morphology may affect the aerodynamics of an avian tail. In this way, we can determine how birds may use their tail configuration to provide longitudinal stability and control during flight. This insight can then be used when analyzing the BIRE control system to provide intuition into potential trade-offs between longitudinal and lateral stability and control.

Tail Spread

Studies that have focused on the aerodynamic effects of tail spread on longitudinal stability and control have concluded that tail spread acts to manipulate pitch stability and can be used in conjunction with tail incidence to increase pitch control effectiveness. Qualitatively, Hankin [34] observed that doves keep their tail habitually spread during gliding flight, from which he concluded that the tail provides pitch stability just like the horizontal tail of a traditional aircraft. Thomas and Taylor [35] also noted in photographs of the Gyrfalcon by Dunne [36] that tail spread seemed to be used in conjunction with wing sweep to maintain stability when carrying prey. Since increasing tail spread will directly increase the tail volume ratio $\mathcal{V}$, we can predict this behavior from the relationship in Eq. (2.1). Changing stability during flight would be helpful when transitioning from a steady-state flight condition to a flight phase requiring maneuverability and control.

Several researchers have shown that tail spread can be used in conjunction with tail incidence to increase pitch control effectiveness. For example, Gillies et al. [25] and Carruthers et al. [29, 30] observed that the steppe eagle coupled tail spread with negative tail incidence angles while landing. The tail morphing happened in-phase with changes in the angle of attack of the wings, likely indicates that the tail was being used to counteract the large pitching moments created by the wings. This same phenomena was also observed in the African vulture by Thomas [26]. We see that increasing the tail volume ratio will similarly increase the pitch control effectiveness of a traditional aircraft as shown in Eq. (2.6). These examples indicate that controlling pitch effectiveness using tail spread may be an effective way to transition from level flight to more vertical flight phases, which is encountered in maneuvers such as perching or vertical take-off and landing scenarios.

Lastly, many researchers have hypothesized that tail spread allows birds to manipulate drag and provide weight support during flight. Pennycuick [37] noted from wind tunnel tests that pigeons adjust their tail spread depending on their speed, adopting a large spread angle during slow glides and reducing tail spread as their glide speed increased. Many others have noted this variation in tail spread with glide speed and have concluded that it is likely used to balance drag generation with weight support requirements [28, 35, 37–45].

Tail Incidence

Research into the effects of tail incidence on longitudinal stability and control have indicated that birds likely use this degree of freedom in a similar manner to a traditional aircraft elevator. That is, tail incidence does not correlate with substantial changes in longitudinal stability; rather, it is used to produce controlling moments in pitch and also to control lift support in conjunction with the wings. Observations by Pennycuick and Webbe [46] indicated that very small tail deflections appeared to be used by the northern fulmar for corrective purposes in pitch. Similar conclusions have been drawn by Raspet [47] and Pennycuick [48], the latter of which observed this behavior in kites, such as the one shown in Fig. 1.2.

Changes in tail incidence or deflection have been observed to be coordinated with the spreading of the tail, especially during maneuvers that required a large degree of pitch control, such as perching and turning [26, 32, 49]. For example, both Thomas [26] and

Carruthers et al. [29] noted that the steppe eagle and African vulture spread and held their tails at negative incidence angles while perching. Equation (2.6) indicates that this would increase the pitch control derivative of the bird, thereby increasing the total pitching moment produced for a given tail incidence angle. Likewise, Gillies et al. [25] found that, while gathering data from a steppe eagle in flight, one of the most consistent movements they observed was increased tail spread and negative tail incidence when initiating a banked turn. The authors hypothesized that this motion is used to create a nose-down pitching motion to start the turn, analogous to that used by paragliders to decrease their elevation angle and increase their turning rate. Combining tail spread and tail incidence in this manner could be helpful in aircraft requiring large amounts of pitch control during certain maneuvers without increasing drag throughout the flight envelope with a larger tail.

Finally, just like with tail spread, several studies have shown that the tail incidence needed for birds to trim changes from negative to positive as their flight speed decreased [44,45]. In the final phase of the steppe eagle's perching maneuver, Gillies et al. [25] observed that the tail adopted a positive incidence angle. This was hypothesized to be a means of weight support offered by the tail at low speeds [50] and correlates with the changes in spread angle noted in Section 2.3.1. As shown by the studies in this section, the tail incidence likely serves the same purpose to birds as the elevator of a traditional aircraft. We have noted that pitch control effectiveness can be increased through spread, which allows for larger pitching moments to be generated by changes in incidence. Nevertheless, there is currently no quantitative evidence comparing the pitch control effectiveness of avian tail incidence to a traditional elevator [3].

Tail Rotation

The author found no studies evaluating the effects of tail rotation on longitudinal stability and control in avian flight. As mentioned previously, tail rotation is a means by which the tail volume coefficients can be coupled, providing a tradeoff between longitudinal and lateral aerodynamic properties [51]. Most studies have focused on the effect of tail rotation on lateral stability and control; therefore, Gamble, Harvey, and Bolander et al. [3]

note that this is an area of study that needs to be further explored in future work on bio-inspired flight control.

2.3.2 Lateral Stability and Control

One of the primary concerns with tailless aircraft is the lack of lateral stability and control without a vertical tail. This section will lend insight into how birds provide that stability and control. Specifically, we will be able to evaluate the effectiveness of a birds' tail in providing lateral stability and control through the use of tail rotation.

Tail Spread and Incidence

Studies on avian aerodynamic control have indicated that tail spread and incidence are most effective at contributing to lateral stability and control when coupled with tail rotation. Changes in incidence of a level tail make substantial changes in only the longitudinal degrees of freedom. The only work that has been done regarding lateral stability and control has focused on the effect of tail spread on the static and dynamic yaw stability.

Analytical and numerical studies by Sachs [52–54] indicate that the relative inertia of some birds may be small enough that the sweep of their wings and tail may provide sufficient dynamic and static yaw stability for their flight. Referring to Eq. (2.12), we see that increasing the tail volume coefficient $\mathcal{V}$ through spread will provide a stabilizing contribution to the yaw stability. In terms of lateral dynamic stability, the resulting effect on the stability of the spiral and Dutch roll modes is unclear. Depending on the relative magnitudes of $C_{\ell,\beta}$, $C_{\ell,\bar{r}}$, $C_{n,\bar{r}}$, $C_{n,\bar{p}}$, and $C_{\ell,\bar{p}}$, increasing the magnitude of the yaw stability derivative could provide a stabilizing contribution to the spiral mode and to the Dutch roll mode through R_{D_p}. The ability to manipulate lateral static and dynamic stability through a longitudinal control effector such as tail spread could serve to mitigate some of the instabilities created by removing the vertical tail.

Tail Rotation

Rotating their tail during flight has been hypothesized to allow birds to: augment their yaw stability, counteract dynamic lateral instabilities, produce lateral moment combinations required to initiate maneuvers like a banked turn, and control adverse yaw. Analytical work by Thomas [26] showed that birds could use tail rotation to augment their yaw stability. The analytical relationship in Eq. (2.12) shows that increasing the vertical tail volume coefficient would provide a stabilizing contribution. This would occur in spite of the decrease in the horizontal tail volume coefficient, since the effects of the lift slope on the vertical portion of the tail likely outweigh the effects of drag-induced yaw stability.

Observing many banked turns performed by the steppe eagle led Gillies et al. [25] to conclude that tail rotation was used to counteract spiral instability. They found that the steppe eagle consistently held its tail at a higher rotation angle than the bank of its turn, which would provide a yawing moment away from the direction of the turn. Thus, holding the tail at this over-banked rotation angle would counteract the additional yawing moment produced by an unstable spiral mode. As there has been relatively little investigation into this hypothesis in the biological community [3], additional research is required to understand the implications of tail rotation on lateral dynamic stability control.

Several researchers have noted that a combination of tail rotation, tail spread, and tail incidence are used to perform banking turns. For example, we have already described the banked turns of the steppe eagle studied by Gillies et al. [25]. Similarly, Oehme [55] noted that a the banked turn of a drongo was coordinated using a spread and twisted tail. Oehme [32] also recorded that a rightward-banked turn by the hen harrier was initiated by negative tail rotation. To stop the turn, the hen harrier used a rapid positive tail rotation [32], which aligns with observations by Gillies et al. [25] of the steppe eagle performing the same rotation when completing its banked turn. Though not equivalent to a rudder deflection, by referring to Eqs. (2.21) and (2.11), we see that increasing the vertical tail volume coefficient results in an increase in the yaw and roll control derivatives. Thus, we would expect that a similar relationship holds with tail rotations combined with tail spread and incidence.

Adverse yaw typically refers to the tendency of an aircraft in a banked turn to yaw in the direction opposite to the desired turning direction. In the presence of adverse yaw, a banked turn to the right will often result in the aircraft yawing to the left. In an aircraft, adverse yaw is typically counteracted by deflecting the rudder to produce a counteracting yawing moment. Research by Thomas [26] into bird flight and control led him to hypothesize that tail rotations could be used to counteract adverse yaw. Supporting his hypothesis, Gillies et al. [25] observed many transient tail rotations throughout the banked turn of the steppe eagle. The authors found that, despite their transient nature, at least some of these tail rotations were consistent with counteracting transient adverse yaw effects. Studying pigeons, Warrick et al. [56] found that tail twist could be used to counteract adverse yaw only in high-speed flight. In slow flight, they found that the demands of weight support did not allow the tail to be twisted and thus could not be used to counteract adverse yaw.

The results discussed to this point indicate that there is still much to be learned about how the tail degrees of freedom available to birds allow them to control their flight [3]. Fortunately, results from bird-scale UAVs and other aircraft allow some of these relationships to be defined in the context of traditional aircraft flight mechanics. Though future work will be required to understand how the tail degrees of freedom are used by birds in flight, the studies that follow will allow us to understand potential risks and benefits of employing the BIRE control system.

2.4 Control of Aircraft Using a Rotating Tail

Several of the studies included in this section of literature were referred to briefly at the beginning of this chapter. Their importance in providing context into the BIRE indicates that they provide substantial evidence of the benefits and risks of using a rotating tail to control flight. In this section, we will specifically mention many of these benefits and risks, while supporting the insight gleaned from avian flight control in Section 2.3. Thus, this section outlines what has been gleaned from aircraft about the use of a rotating tail to control flight, while the section previous gives insight into benefits that we may yet be able to leverage in future designs.

2.4.1 Longitudinal Stability and Control

We have previously discussed in Section 2.3.1 the key findings relating to longitudinal stability and control for bird flight. First, research by several authors showed tail spread allows birds to manipulate their static pitch stability, provide weight support, and increase pitch control effectiveness. Research suggests that tail incidence was primarily a method of generating pitching moments during flight and can likely be used in a manner equivalent to the elevator of a standard aircraft. Finally, although tail rotation provides a coupling between the longitudinal and lateral degrees of freedom, there are currently no studies available that examine the effect of tail rotation on longitudinal stability and control. In the following section, we will analyze the implications to longitudinal stability and control of using a rotating tail control system on UAVs and aircraft.

Tail Spread and Incidence

The function of a traditional elevator control system on aircraft is well understood and was described in Section 2.1. Therefore, we will investigate the effect on longitudinal stability and control of tail spread and tail incidence together in this section, with the emphasis placed on tail spread. The literature in this section shows that aircraft employing both tail spread and incidence are able to manipulate pitch stability and pitch effectiveness, in addition to providing lift support to the aircraft at low speeds.

Hummel [57] presented arguably the most extensive wind tunnel tests investigating avian-inspired tail control that have been performed to date. He created wooden wing-tail models for a variety of wing-and-tail geometries with the tail mounted directly to the trailing-edge of the main wing. Wind tunnel tests on these models revealed that increasing the spread on a given tail model increased both the static pitch stability and pitch control effectiveness of the tail across several angles of attack [57]. Hummel attributed this result to the formation of a leading-edge vortex on the highly-swept tail, thereby increasing the lift generation of the tail. The Lishawk UAV, studied by Ajanic et al. [58], and the bionic morphing tail design by Zheng et al. [59] both found the same relationship between tail spread and pitch stability and control effectiveness. As discussed previously, increasing the spread of the tail directly increases the tail volume coefficient, which increases both $C_{m,\alpha}$ and C_{m,δ_e} as shown in the relationships in Eq. (2.1) and (2.6), respectively.

In Sections 2.3.1 and 2.3.2, we showed several studies that highlighted the use of tail spread and incidence in birds at low speeds to provide additional weight support. Results from the Lishawk showed that flight at slow speeds required both tail spread and incidence to provide the proper lift support to maintain steady, level flight while maintaining minimum power required [58]. This result supports the idea that a combination of tail spread and incidence not only provides lift support, but also can be used to reduce drag, as mentioned by many researchers [28, 35, 37–45].

Tail Rotation

While there were no studies on the effects of tail rotation on longitudinal stability and control for avian flight, there have been several studies on UAVs or aircraft that indicate relationships between tail rotation and longitudinal stability and control. These studies show tail rotations vary the static and dynamic longitudinal stability, pitch control effectiveness, and allow for lift and drag manipulation on the tail.

A bird-inspired rotating tail design, similar to the BIRE control system, was simulated using a 6 degree-of-freedom model by Bras et al. [60] to determine its flight dynamics and control properties. The simulations showed that in longitudinally-trimmed flight, the pitch stability decreased with rotation angle. With approximately 65° of tail rotation, the aircraft became longitudinally unstable $C_{m,\alpha} > 0$, indicating that substantial tail rotation was required to destabilize the aircraft [60].

Bras et al. [60] also examined the effects of tail rotation on the dynamic stability of the aircraft. A state-space analysis showed that the short-period mode decreased in magnitude from $\mathrm{Re}\,(\lambda)_{\mathrm{sp}} \approx -1$ to $\mathrm{Re}\,(\lambda)_{\mathrm{sp}} \approx -0.3$ at 90° tail rotation. Examining Eqs. (2.3) and (2.4), we see that $C_{m,\bar{q}}$ will decrease in magnitude as the horizontal tail volume coefficient decreases with tail rotation. Therefore, the decrease in magnitude of the real part of the short-period eigenvalue is predicted by these analytical relationships.

When examining the result of tail rotation on the stability of the phugoid mode, Bras et al. [60] found that the phugoid mode stabilized with tail rotation, eventually degenerating into two real, stable roots. We first note that, in the results given on static pitch stability by the authors, $C_{m,\alpha}$ became less negative with tail rotation. In the analysis of the short-period mode, we also found that $C_{m,\bar{q}}$ would decrease in magnitude due to a decrease in tail volume coefficient. According to those relationships with increasing tail rotation, we see in Eq. (2.5) that the denominator will trend more positive while the numerator remains negative. Thus, the increase in phugoid stability with tail rotation can also be expected according to Eq. (2.5).

Results generated by Parga et al. [61], Hummel [57], and Bras et al. [60] show that both the lift and drag can be manipulated using tail rotation. For example, the UAV design investigated by Parga et al. [61] found that, with positive tail incidence, tail rotation reduced the lift and drag on the tail regardless of rotation direction. This is consistent with what would be expected from the trade-off between horizontal and vertail tail volume coefficients with tail rotation. Hummel's [57] results expand on those of Parga et al. [61] by noting that this result is dependent on the lateral orientation of the aircraft. When his models had positive sideslip, he noted that a negatively twisted tail negatively incremented the lift. Conversely, positive twist at the same sideslip angle created a positive increment in the lift on the tail.

Finally, the simulations performed by Bras et al. [60] showed that the elevator deflection required to trim the aircraft in steady level flight increased modestly from $2°$ to $3°$ with tail rotations from $0°$ to $\pm 75°$. At each tail rotation, the trim angle of attack remained constant. With tail rotations beyond $\pm 75°$, however, the elevator deflection required to trim rapidly became negative, settling at approximately $-1°$ with a tail rotation of $\pm 90°$. This indicates that there is a substantial range of tail rotations for which tail rotation has a very small effect on the longitudinal trim properties of the aircraft.

Though these examples have expanded on the relationship between tail rotation and longitudinal stability and control when compared to what we know from avian flight, this is still a relatively unexplored area of research [3]. This identifies an important contribution that this dissertation can make to the literature addressing the feasibility of control using a rotating tail.

2.4.2 Lateral Control and Stability

In Section 2.3.2, we learned that tail spread and incidence were most effective at contributing to lateral stability and control when used in conjunction with tail rotation. The work by Sachs noted that the lower inertia of birds may allow them to provide sufficient static stability with only a level tail and the sweep of their wings [52–54]; however, when dealing with larger UAVs and aircraft, this certainly will not be the case. When tail spread and incidence were used in conjunction with tail rotation, research shown that a rotated tail could be used to generate rolling and yawing moments and provide yaw stability.

We have established that tail spread and incidence are the most impactful to lateral control and stability when used in conjunction with tail rotation. Indeed, the only explicitly measured effects of tail spread and incidence are taken in conjunction with tail rotation. Therefore, we will forgo a discussion on tail spread and incidence and instead directly address the effects of tail rotation (coupled with tail spread and incidence) on lateral control and stability.

Tail Rotation

The studies mentioned in this section reveal that tail rotation is able to contribute to lateral stability and control by increasing the roll control effectiveness, stabilizing the dynamic roll mode, and augmenting both yaw stability and control effectiveness. Research has also confirmed that employing a rotating tail on an aircraft allows the aircraft to produce roll and yaw moment combinations for lateral maneuvers and control adverse yaw as discussed in Section 2.3.2.

Both the wind tunnel experiments of Hummel [57] and the simulations performed by Bras et al. [60] indicate that tail rotation has no effect on the static roll stability of an aircraft. The relationship in Eq. (2.7) suggests that increasing the vertical tail volume coefficient through tail rotation would increase the static roll stability of an aircraft. However, we remind the reader that Eq. (2.7) was derived assuming a single vertical tail surface. Solid-body tail rotations like the ones investigated here produce an equal vertical tail volume coefficient above and below the aircraft. Therefore, it is likely that the generated rolling moments cancel one another with the aircraft in sideslip, as suggested by the results above. Roll stability is generally dominated by the lift on the main wing, so a lack of added roll stability from a rotating tail does not represent a substantial concern for a rotating empennage control system.

The dynamic analysis performed Bras et al. [60] showed that the dynamic roll mode became increasingly stable with tail rotation, its magnitude doubling over the range of 90° tail rotations. Examining Eq. (2.9), we can see that an increase in roll damping coefficient $C_{\ell,\overline{p}}$ increases the stability of the roll mode. Although unclear how the relationship in Eq. (2.10) would be affected by a rotating tail, one possible explanation would be that the rotation of the tail out of the downwash of the wing would increase the lift slope of the tail and thereby increase the damping it experiences in roll.

Hummel [57] and Parga et al. [61] both found that tail rotations increased the roll control effectiveness of the aircraft they studied. Data from the wind tunnel models tested by Hummel [57] showed that a positive, planar tail rotation would provide a negative rolling moment. This rolling moment could then be counteracted or augmented by negative or positive incidence angles, respectively. Parga et al. [61] also showed that a rotating tail was able to produce rolling moments, though these moments were substantially smaller than the rolling moments produced by traditional ailerons.

This is to be expected, since the rolling moment produced by ailerons is substantially increased with the large lifting force acting on the main wing. It is likely that the increase in roll control effectiveness is not entirely described by an increase in vertical tail volume coefficient as shown in Eq. (2.11). While increasing the tail volume coefficient would certainly contribute to generated moments, due to the symmetric nature of the tail about the roll axis, it is likely the increase in rolling moment production from tail rotations is a result of downwash effects.

Intuitively, yaw stability has been shown to increase with tail rotation angle. Both Hummel [57] and Parga [61] found that rotating the tail increased the yaw stability independent of the direction of rotation. Further, simulations from Bras et al. [60] showed that, with approximately 50° of tail rotation, a planar rotating tail design was shown to have the same level of yaw stability as the baseline aircraft they modeled, which had a vertical tail. Referring to Eq. (2.12), we can see that any increase in the vertical tail volume coefficient should stabilize an aircraft in yaw.

Investigations into the spiral and Dutch roll modes of Bras et al.'s [60] aircraft designs revealed that increased tail rotation caused both modes to become increasingly unstable. We know from their previous results that the roll stability was constant with tail rotation and it is likely that the roll damping and roll cross-damping terms are each negative. Therefore, referring to Eq. (2.13), we can focus on the second terms in the numerator and denominator (since the change in $C_{\ell,\beta} \approx 0$). Thus, an increase in static yaw stability would drive the real part of the dimensionless spiral mode eigenvalue more positive, as noted in the data produced by Bras et al. [60].

Bras et al. [60] found that the Dutch roll mode was oscillatory for small tail rotation angles before gradually degenerating into two unstable real roots. Using the same analysis method as with the spiral mode, and assuming that $C_{n,\bar{p}} < 0$, we see from Eqs. (2.17) and (2.18) that R_{D_c} is destabilizing. We also note that $R_{D_s} < 0$ from Eq. (2.20), and therefore R_{D_p} in Eq. (2.19) is also destabilizing to the real part of the dimensionless Dutch roll eigenvalue. Again, an increase in static yaw would drive the Dutch roll mode to instability, as shown in the simulations by Bras et al. [60]. The instability of both the spiral and Dutch roll modes with increased tail rotation represents one potential concern with the BIRE control system. Fortunately, instability in these modes may be controlled out by active damping from BIRE rotation and elevator deflections.

The results from Parga et al. [61], Hummel [57], and Bras et al. [60] indicate that a rotating tail is able to provide an increase in yaw control effectiveness. Both Hummel and Parga et al. found that tail rotations created yawing moments on the aircraft and that the direction and magnitude of those moments were directly controlled by tail incidence [57,61]. The dependence of the yawing moment on tail incidence was quickly identified by Parga et al. [61] to be a potential difficulty, as that single control mechanism was responsible for both longitudinal and lateral force and moment control. The wind tunnel tests performed by Hummel [57] showed the same coupling when tail rotation was introduced into the horizontal tails of his models.

Bras et al. [60] further investigated this potential issue by comparing the yawing moment produced by a longitudinally-trimmed rotating tail configuration to a traditional rudder configuration that was likewise longitudinally-trimmed. Their results showed that large rotation angles were required to trim the aircraft and produce a yawing moment with a planar rotating tail. In fact, it required nearly 90° of tail rotation to produce the same yawing moment as 5° of rudder deflection on the baseline aircraft, indicating a much lower yaw control derivative on the rotating tail design. Interestingly, Parga et al. [61] believed that their rotating V-tail design had the potential to produce a yaw control effectiveness that was comparable to a traditional rudder., but they did not show any proof of this idea.

Lastly, like with birds, the results from Parga et al. showed that tail rotation was helpful in completing lateral maneuvers and controlling out adverse yaw. For example, the rotating V-tail UAV design developed by Parga et al. was found to produce proverse yaw when trimmed [61]. The authors also found that negative tail rotation was beneficial when initiating banked turns to the right, which is consistent with the results from Oehme and Gillies et al. [25,32].

CHAPTER 3

DESCRIPTION OF THE BASELINE AIRCRAFT AND ITS BIRE VARIANT

A control system similar to the BIRE could potentially be applied to nearly any aircraft with a traditional empennage including fighter, transport, passenger, and general aviation aircraft. However, in this work the BIRE control system will be applied to a fighter-type aircraft. The application of the BIRE control system to a fighter aircraft is motivated by an interest in the effects of the control system when applied to a marginally stable configuration. Additionally, removing the vertical tail has a substantial effect on the weight of the aircraft. Though the reduction in weight may be mitigated by the weight of the control system itself, the benefits to maneuverability and range will likely justify even a slight increase in weight.

Before a thorough aerodynamic analysis can be performed, a description of the geometry of the aircraft is required along with the flight conditions at which the analysis will be performed. In this chapter, the relevant geometric and inertial properties of the chosen aircraft are given. Applying the BIRE control system to the baseline aircraft makes substantial changes to these properties and these are recorded in this chapter as well. After outlining the characteristics of each aircraft, the flight envelope and corresponding points of analysis for both aircraft are described.

3.1 Description of the Baseline Aircraft

The baseline fighter aircraft chosen for this analysis is a single engine, supersonic, tactical aircraft, similar to the F-16 Fighting Falcon. Publicly-released measurements of the lifting surfaces of the F-16 are given in the works of Fox and Forrest [62] and Butcher [63]. Using these measurements along with the various geometric and aerodynamic definitions from flight mechanics, a simple model of the F-16 can be developed for use with the aerodynamic tools referenced in Chapter 4.

Table 3.1: Lifting surface geometry data used to model the baseline aircraft.

	Parameter	**Fox and Forrest [62]**	**Butcher [63]**
Main Wing	Planform Area, S_w, [ft^2]	300*	—
	Span, b_w [ft]	30*	—
	Aspect Ratio, R_{A_w}	3*	—
	Taper Ratio, R_{T_w}	0.2275	—
	Mean Aerodynamic Chord, $\bar{c}_w$ [ft]	11.32*	—
	Leading-Edge Sweep, Λ_{LE_w} [deg]	40	—
	Trailing-Edge Sweep, Λ_{TE_w} [deg]	0	—
	Airfoil Section	NACA 64A204	—
Horizontal Tail	Planform Area, S_h, [ft^2]	63.675	63.7
	Semispan, $b_h/2$ [ft]	5.803	5.801
	Root Chord, c_{r_h} [ft]	—	7.983
	Tip Chord, c_{t_h} [ft]	—	3.117
	Aspect Ratio, R_{A_h}	2.116	2.1
	Mean Aerodynamic Chord, $\bar{c}_h$ [ft]	5.906	—
	Leading-Edge Sweep, Λ_{LE_h} [deg]	40	40
	Trailing-Edge Sweep, Λ_{TE_h} [deg]	0	0
	Dihedral, Γ_h [deg]	-10	–
	Airfoil Section (Root/Tip)	Biconvex 6/3.5%	
Vertical Tail	Planform Area, S_v, [ft^2]	54.675	—
	Exposed Span, b_v [ft]	8.416	—
	Aspect Ratio (Theoretical), R_{A_v}	1.294	—
	Mean Aerodynamic Chord, $\bar{c}_v$ [ft]	6.838	—
	Leading-Edge Sweep, Λ_{LE_v}, [deg]	47.5	—
	Airfoil Section (Root/Tip)	Biconvex 5.3/3%	—

* Confirmed by Nguyen et al. [64]

Calculated Geometric Properties

The information given in Table 3.1 is enough to completely characterize the planform geometry of the main wing and horizontal tail. To do so, the aspect ratio and taper ratio of a tapered wing can be defined, respectively, as [24]

$$R_A \equiv \frac{b^2}{S} \tag{3.1}$$

and

$$R_T \equiv \frac{c_t}{c_r} \tag{3.2}$$

where b is the span of the surface, S is planform area, c_t is the chord length at the tip of the lifting surface and c_r is the chord length at the root. The planform area is the area of a trapezoid and is given by

$$S = \frac{c_r + c_t}{2} b \tag{3.3}$$

In particular, Eq. (3.2) can be solved for c_t and substituted into Eq. (3.3) to solve for the root chord given the span and taper ratio. Without the taper ratio of the vertical tail, however, additional analysis must be performed to characterize its planform.

The taper ratio can be found using the aspect ratio, span, and the mean aerodynamic chord, $\bar{c}$. The mean aerodynamic chord is defined as [65]

$$\bar{c} = \frac{2}{S} \int_0^{\frac{b}{2}} c(z)^2 \, dz \tag{3.4}$$

and represents the chord length with an equal moment about its aerodynamic center as the entire aircraft has about its aerodynamic center [66]. To calculate the mean aerodynamic chord using Eq. (3.4), the chord distribution of a wing with constant taper, $c(z)$, is required. Given in terms of the span, aspect ratio, and taper ratio, the chord distribution of a wing is [24]

$$c(z) = \frac{2b}{R_A\,(1 + R_T)} \left[1 - (1 - R_T) \left| \frac{2z}{b} \right| \right] \tag{3.5}$$

Eqs. (3.1)–(3.5) can be combined to solve for the taper ratio, given the aspect ratio, mean aerodynamic chord, and span.

Substituting Eq. (3.5) into Eq. (3.4) and simplifying, the integral becomes

$$\bar{c} = -\frac{4b(R_T^3 - 1)}{3R_A(1 - R_T)(1 + R_T)^2}$$

This can be rearranged to form a cubic of the taper ratio, given by

$$(4b - 3\bar{c}R_A)R_T^3 - 3\bar{c}R_A R_T^2 + 3\bar{c}R_A R_T + (3\bar{c}R_A - 4b) = 0 \tag{3.6}$$

Finding the roots of this equation gives three candidates for the value of the taper ratio: these are

$$R_T = (1, 2.290, 0.437) \tag{3.7}$$

Tapered wings are nearly always given a taper ratio that is less than one, corresponding to a larger root chord than tip chord. Under this constraint, the only value that corresponds to a tapered wing with root chord larger than tip chord is $R_T = 0.437$. At this point, the root and tip chord of the vertical tail can be calculated in the same manner as previously outlined.

The aerodynamic model used here requires that the quarter-chord sweep angle of each wing be known. Referring to Fig. 3.1, and letting m represent the chord fraction as measured from the leading-edge, the sweep angle at chord fraction m can be calculated as

$$\tan \Lambda_m = \tan \Lambda_{\mathrm{LE}} + \frac{2m}{b} c_r \left(R_T - 1\right) \tag{3.8}$$

where $0 \leq m \leq 1$. This relationship can be used to solve for the quarter-chord sweep angle, $\Lambda_{c/4}$, of each wing.

Finally, the mean geometric chord for each wing surface will need to be known for an analysis performed later in this work. The mean geometric chord is defined as [24]

$$\bar{c}_g \equiv \frac{S}{b} = \frac{1}{b} \int\limits_{z=-b/2}^{b/2} c(z)\, dz \tag{3.9}$$

For wings with linear taper, the integral in Eq. (3.9) becomes

$$\bar{c}_g = \frac{1 + R_T}{2} c_r \tag{3.10}$$

Scaled Geometric Properties

The planform geometries of each lifting surface can be described completely with the methodology above; however, modeling the aircraft in a low-order aerodynamic tool re-

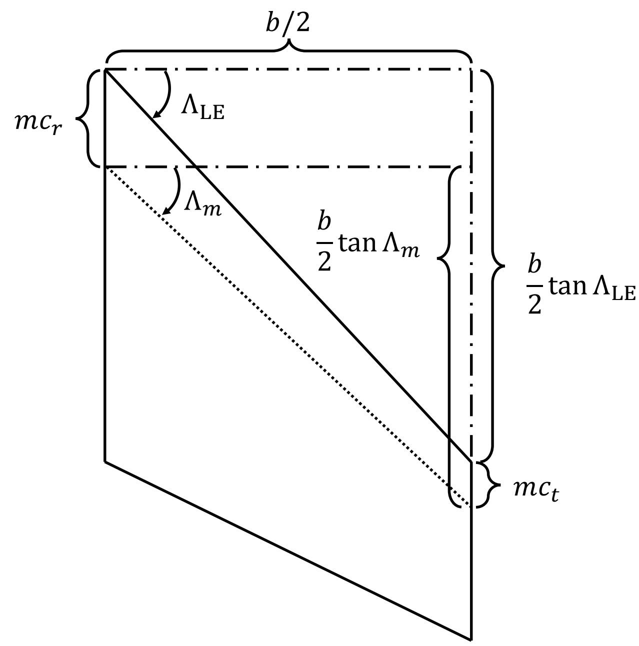


Fig. 3.1: Diagram used to solve for the quarter-chord sweep angle of a wing.

quires that each lifting surface extend to the centerline of the aircraft. Since the fuselage contributes to the lift generated by the aircraft [67], approximating fuselage effects by extending lifting surfaces through the fuselage is an acceptable modeling assumption. Therefore, the measurements from the root chord of the horizontal and vertical tails must be approximated by referring to scaled drawings included in the work of Fox and Forrest [62].

Using these scaled drawings, the spanwise distance from the root of the horizontal tail to the centerline of the aircraft is approximated to be 3.40 ft. The vertical distance from the root of the vertical tail to the centerline of the vehicle is likewise approximated at 2.07 ft. In an effort to avoid substantially over-predicting the aerodynamic effect of these surfaces, the leading-edge sweep of the lifting surface was set to zero degrees over the fuselage section of the aircraft.

In the works of both Fox and Forrest [62] as well as Nguyen et al. [64], the axial location of the center of gravity of the aircraft is given in terms of a percentage (35%) of the mean aerodynamic chord of the main wing. Since no information is given to the contrary, it is assumed that the center of gravity lies on the fuselage centerline of the aircraft; that is, there is no displacement in the body-fixed y- or z- directions. The axial distance from the center of gravity can be calculated by first finding the distance from the root leading-edge to the leading edge of the mean aerodynamic chord section. This relationship is defined as

$$l_{\mathrm{cg}_w} = c_{r_w} + \bar{c}_w \,(0.35 - 1)$$

The distance from the center of gravity to the quarter-chord of the main wing at the root chord is

$$x_{\mathrm{cg}_w} = l_{\mathrm{cg}_w} - 0.25c_{r_w} = 0.75c_{r_w} + (0.35 - 1)\bar{c}_w \tag{3.11}$$

With the information given in Table 3.1, the axial displacement of the wing quarter-chord from the center of gravity is 4.86 ft.

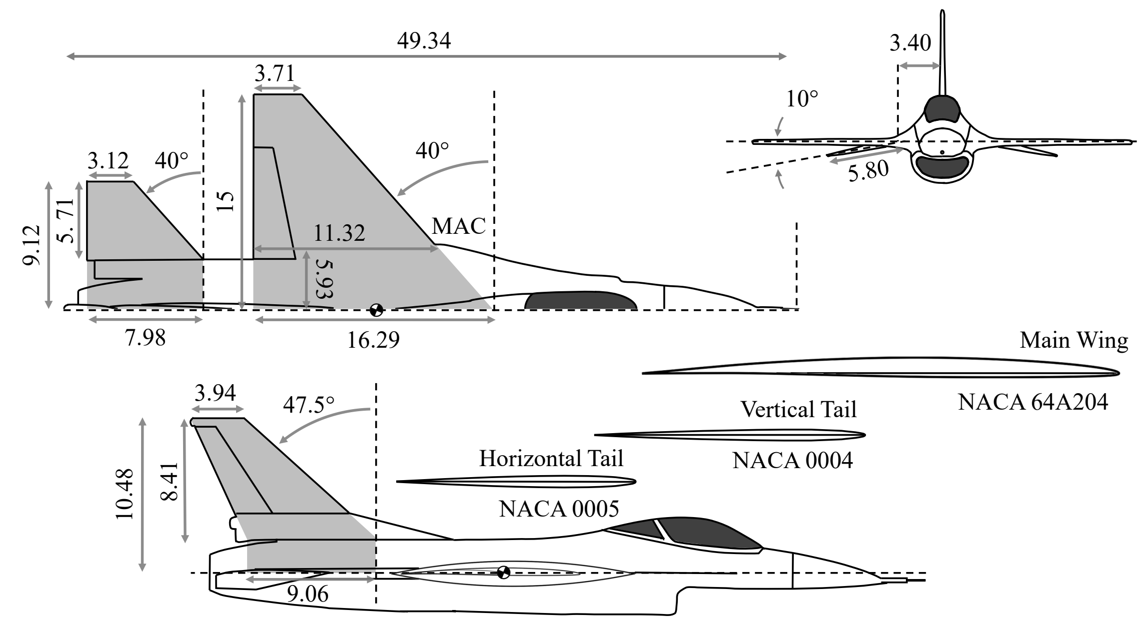
Since relationships between the location of the main wing and the horizontal and vertical tails, the axial distance from the center of gravity to the quarter-chord of the horizontal and vertical tails must be determined using the figures in Fox and Forrest [62]. By determining the distance from the leading-edge of the main wing to the leading-edge of the other lifting surface, noted as $x_{\mathrm{LE-LE}_{h,v}}$ the axial distance from the quarter-chord to the center of gravity is given by

$$x_{\mathrm{cg}_{h,v}} = x_{\mathrm{LE-LE}_{h,v}} + 0.25c_{r_{h,v}} - l_{\mathrm{cg}_w} \tag{3.12}$$

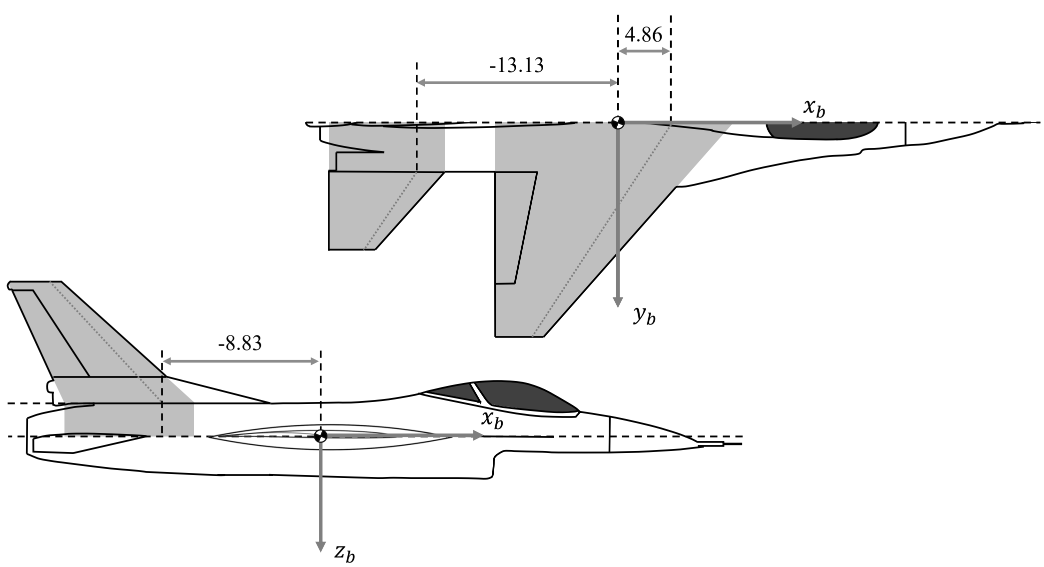
For the horizontal tail, $x_{\mathrm{LE-LE}_h} = 20.1$ ft and for the vertical tail, $x_{\mathrm{LE-LE}_v} = 15.5$ ft. With the lifting surfaces and their relation to the center of gravity defined, a simplified model of the baseline aircraft can be constructed.

From the preceding sections and the information in Table 3.1, the aircraft geometry is defined as shown in Fig. 3.2a. Shaded sections of the figure show the lifting surfaces as represented in the aerodynamic tool. Figure 3.2b gives the location of the quarter-chord of each lifting surface in relation to the center of gravity of the aircraft. This geometric data is summarized in Table 3.2.

The final geometric characterization that needs to be made for the baseline aircraft regards its control surfaces. When flown at supersonic speeds, many aircraft rotate entire



(a) Lifting surface geometry.



(b) Center of gravity references.

Fig. 3.2: The modeled geometry of the baseline aircraft. All dimensions in feet.

lifting surfaces to mitigate the loss of control efficiency encountered due to shock waves [68]. This is the case with the baseline aircraft, which rotates the entire horizontal tail about an axis parallel to the body-fixed y-axis, a configuration often referred to as a stabilator

Table 3.2: Geometric characteristics of the lifting surfaces on the baseline fighter aircraft.

Parameter	**Main Wing**	**Horizontal Tail**	**Vertical Tail**
Planform Area, S [ft^2]	300	63.675	54.675
Exposed Span, b [ft]	30	11.605	8.42*
Aspect Ratio, R_A	3	2.116	1.44
Taper Ratio, R_T	0.2275	0.391	0.52
Root Chord, c_r	16.293	7.980	7.70
Mean Aerodynamic Chord, $\bar{c}$ [ft]	11.320	5.906	6.03
Mean Geometric Chord, $\bar{c}_g$ [ft]	10.0	5.550	5.852
Leading-Edge Sweep, Λ_{LE} [deg]	40	40	47.5
Quarter-Chord Sweep, $\Lambda_{c/4}$ [deg]	32	32	43
Half-Chord Sweep, $\Lambda_{c/2}$ [deg]	23	22	38
Dihedral, Γ [deg]	0	-10	90
Quarter-Chord Location, x_{cg} [ft]	4.86	-13.13	-8.83
Airfoil	NACA 64A204	NACA 0005	NACA 0004

* Represents the exposed semispan of the vertical tail

[9]. The stabilators of the baseline aircraft are able to deflect both symmetrically and antisymmetrically to control flight in the supersonic regime.

In addition to the control offered by the stabilator, the main wing of the aircraft employs trailing-edge ailerons and the vertical tail houses a trailing-edge rudder. Each of these control surfaces are more efficient at lower Mach numbers and help to produce rolling and yawing moments [68]. To simplify the control system for the baseline aircraft, the differential stabilator deflections are coupled to deflections of the main wing ailerons in a ratio of 1:4 [64]. These deflections are assumed to be antisymmetric, meaning that the magnitude of the deflection on each side is equal while the direction is opposite.

Thus, the control surfaces of the baseline aircraft include: ailerons on the main wing, δ_a, which are coupled with differential deflection of the stabilator, δ_d, symmetric stabilator deflections, δ_e, and rudder deflections on the vertical tail, δ_r. Table 3.3 details the saturation limits and actuation rate of the control surfaces modeled on the baseline aircraft as described by Nguyen et al. [64]. The span fraction denotes the fraction of the span of the lifting surface occupied by the control surface and was estimated using the drawings presented by Fox and Forrest [62]. Likewise, the chord fraction represents the portion of the chord at the corresponding span fraction location that the control surface covers. Note that these fractions include the portion of each wing that extends into the fuselage as can be seen from referencing Fig. 3.2.

Table 3.3: Description of the control surfaces on the baseline aircraft.

Control Surface	**Saturation Limits, [deg]**	**Span Fraction**	**Chord Fraction**	**Actuation Rate, [deg/s]**
Aileron*, δ_a	± 21.5	0.23/0.76	0.22/0.22	80
Differential Deflection*, δ_d	± 5.375	0.37/1.0	1.0/1.0	80
Stabilator Deflection†, δ_e	± 25	0.37/1.0	1.0/1.0	60
Rudder, δ_r	± 30	0.36/0.95	0.32/0.32	120

* Antisymmetric Deflection
† Symmetric Deflection

Both Stevens and Lewis [69] and Nguyen et al. [64] provide the inertial information for the baseline aircraft. The inertial characteristics of the aircraft are used to evaluate conditions of trim in Chapter 6 and examine aircraft control characteristics in Chapter 8. Included in the works of Stevens and Lewis [69] and Nguyen et al. [64] is the total weight of the aircraft, W, the components of the inertia tensor (e.g. I_{xx}), and the angular momentum produced by the engines during flight (e.g. h_x). These are each listed in Table 3.4.

Table 3.4: Inertial properties of the baseline aircraft.

Parameter	**Value**
Weight, W [lbs.]	20,500
Inertia, I_{xx} [slug-ft^2]	9,496
Inertia, I_{yy} [slug-ft^2]	55,814
Inertia, I_{zz} [slug-ft^2]	63,100
Inertia, I_{xy} [slug-ft^2]	0
Inertia, I_{xz} [slug-ft^2]	982
Inertia, I_{yz} [slug-ft^2]	0
Engine Momentum, h_x [slug-ft^2/s]	160
Engine Momentum, h_y [slug-ft^2/s]	0
Engine Momentum, h_z [slug-ft^2/s]	0

The moments of inertia, I_{xx}, I_{yy}, and I_{zz}, are a measure of the resistance of the aircraft to angular accelerations about each body-fixed axis [70]. These are, by definition, positive quantities. In contrast, the products of inertia, I_{xy}, I_{xz}, and I_{yz}, are not sign-restricted and are a measure of the location of mass with respect to the given plane [71]. If the product of inertia contains an axis perpendicular to a plane of symmetry, then the product of inertia will be zero, since the mass across that axis is equally distributed on both sides of the aircraft. With respect to the products of inertia in Table 3.4, we note that the y-axis lies perpendicular to the x-z plane of symmetry. Therefore, both I_{xy} and I_{yz} are equal to zero, which is the case for most aircraft.

3.2 Description of the BIRE Variant

The characteristic geometry change incorporated into the design of the BIRE aircraft is the removal of the vertical tail. Anhedral on the horizontal tail section of the baseline aircraft was also replaced in the BIRE with a planar tail configuration. While maintaining some degree of dihedral on the tail would add lateral stability to the BIRE [72], it was determined that the control fidelity of the aircraft would be first be analyzed without changes in dihedral. Once potential problems with the planar design are identified, future work can investigate the role of dihedral on the aircraft aerodynamics and controls.

Finally, various changes were made to the aft portion of the outer mold line of the BIRE to simplify the design of the rotation mechanism of the tail and present a more tractable mechanical system. These changes are outlined in the work by Bolander et al. [73] and Ives et al. [74]. As these outer mold line changes did not effect how the lifting surfaces would be modeled in the aerodynamic software, they will not be outlined again here.

As a counterpart to Fig. 3.2, a basic outline of the entire geometry of the BIRE is included for reference in Fig. 3.3. Note that the location of the quarter-chord of the main wing and horizontal tail remain consistent between the baseline aircraft and the BIRE. Therefore, Fig. 3.2b provides a sufficient reference for these dimensions when analyzing the BIRE. Additionally, Fig. 3.4 shows a cutaway view of the empennage of each aircraft to highlight the differences in design.

As mentioned above, the BIRE design is nearly identical to the baseline aircraft in terms of its modeling, with the exception being the presence of the vertical tail and the

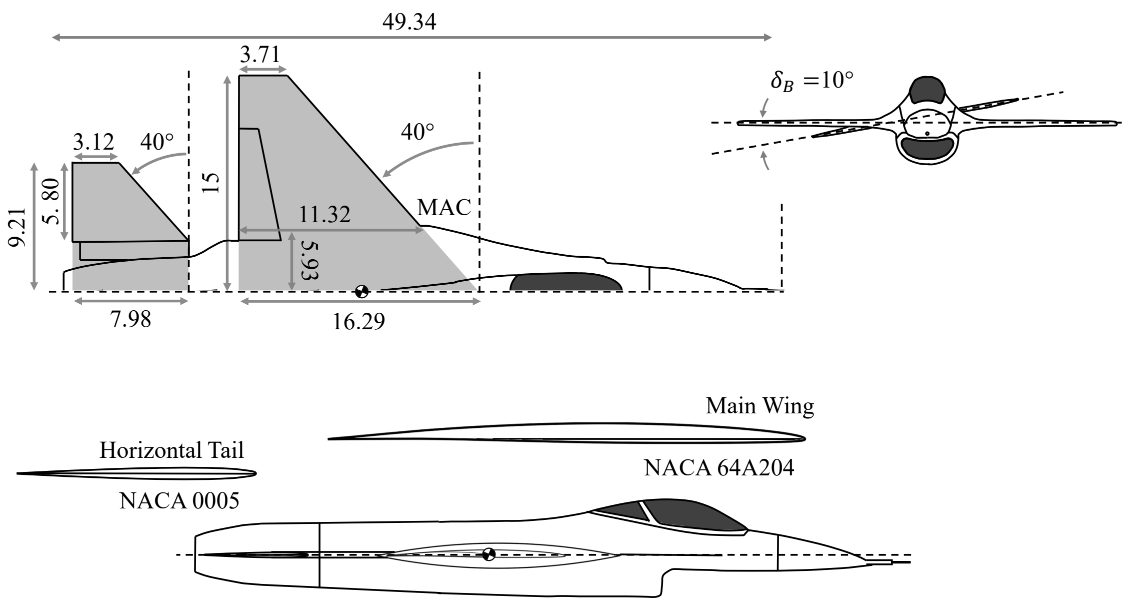
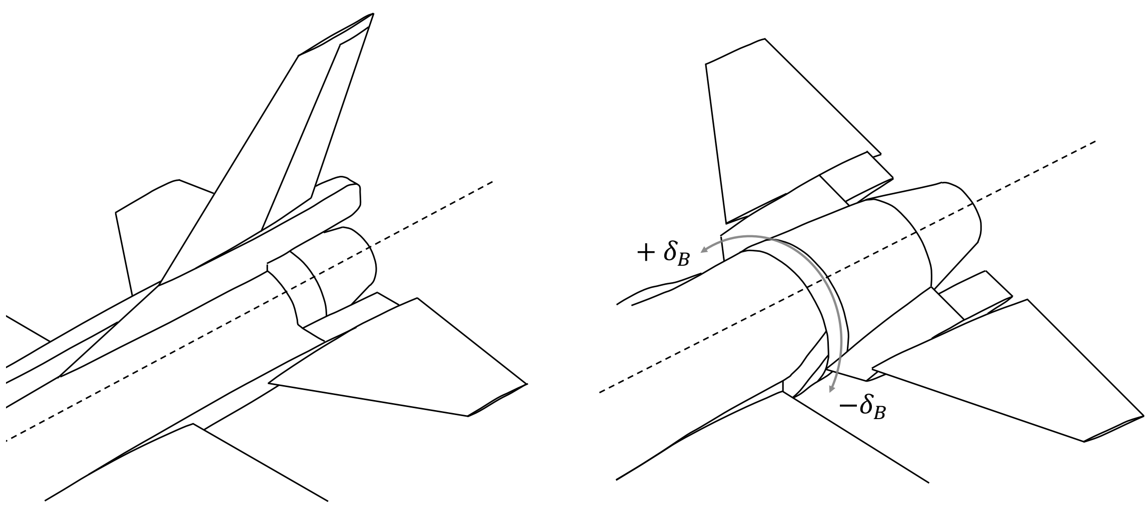


Fig. 3.3: The modeled geometry of the BIRE aircraft. All dimensions in feet.



(a) Baseline aircraft

(b) BIRE variant

Fig. 3.4: Empennage geometry of the baseline aircraft and its BIRE variant.

lack of anhedral on the horizontal tail. For completeness, Table 3.5 contains a summary of the geometric data of the BIRE variant analogous to the information in Table 3.2. Again, it is assumed that the center of gravity and each lifting surface lies on the centerline of the aircraft with no y- or z- displacements. This assumption is important for the BIRE because the mechanism required to actuate tail rotations will be more tractable with a symmetric distribution of weight on the tail.

Table 3.5: Geometric characteristics of the lifting surfaces on the BIRE variant.

Parameter	**Main Wing**	**Rotating Tail**
Planform Area, S [ft^2]	300	63.675
Span, b [ft]	30	11.605
Aspect Ratio, R_A	3	2.116
Taper Ratio, R_T	0.2275	0.391
Root Chord, c_r	16.293	7.980
Mean Aerodynamic Chord, $\bar{c}$ [ft]	11.320	5.906
Leading-Edge Sweep, Λ_{LE} [deg]	40	40
Quarter-Chord Sweep, $\Lambda_{c/4}$ [deg]	32	32
Half-Chord Sweep, $\Lambda_{c/2}$ [deg]	23	22
Dihedral, Γ [deg]	0	0
Quarter-Chord Location, x_{cg} [ft]	4.86	-13.13
Airfoil	NACA 64A204	NACA 0005

The BIRE design employs a three degree-of-freedom control system, including symmetric (1) and antisymmetric (2) stabilator deflections as well as the solid-body rotation of the empennage about the centerline of the aircraft (3). Table 3.6 provides information on the control surfaces of the BIRE variant. Like the baseline aircraft, the ailerons and differential tail deflections are anti-symmetric deflections and the differential tail deflections are coupled to the ailerons with a ratio of 1:4. The notation for the overlapping control surfaces between the baseline aircraft and BIRE are consistent with the exception of a superscript B, which will be used to distinguish between the baseline and BIRE control surface deflections when plotted together. The tail rotations of the BIRE are denoted δ_B and given the name BIRE rotation angle and here it is assumed that the tail is able to rotate at the same speed as the slowest actuation rate among the control surfaces on the BIRE. As this is a preliminary estimate, a conservative assumption was made that recognizes the mechanical difficulty in rotating the entirety of the horizontal tail.

The inertial properties of the BIRE will vary substantially from those of the baseline aircraft, shown in Table 3.4, when the empennage is rotated. A more thorough analysis

Table 3.6: Description of the control surfaces on the BIRE aircraft.

Control Surface	**Saturation Limits, [deg]**	**Span Fraction**	**Chord Fraction**	**Actuation Rate, [deg/s]**
Aileron*, δ_a^B	± 21.5	0.23/0.76	0.22/0.22	80
Differential Deflection*, δ_d^B	± 5.375	0.37/1.0	1.0/1.0	80
Stabilator Deflection†, δ_e^B	± 25	0.37/1.0	1.0/1.0	60
BIRE Rotation, δ_B	± 180	–	–	60

* Antisymmetric Deflection
† Symmetric Deflection

of the structure of each aircraft would be required to establish the appropriate weight estimates of each components and the effect of mechanism design on the weight and center of gravity of the aircraft. This analysis is outside the scope of this work and, instead, preliminary estimates based on CAD reconstructions of the geometry and load estimates will be leveraged to make an informed estimate. Preliminary calculations of the weight of the vertical tail using an aluminum skin gives a weight of approximately 500 lbs. Ideally, the weight reduction caused by removing the vertical tail would be balanced or maintained slightly net-negative by the addition of actuators to rotate the tail. However, as a conservative estimate, it is assumed that the actuators will increase the weight of the baseline aircraft by 500 lbs and that this weight will be added without substantially changing the location of the center of gravity.

Using a CAD model of the BIRE aircraft, the moments and products of inertia were calculated at several different tail rotation angles as reported in Table 3.7. As intuition would suggest, these inertial changes for the BIRE are periodic in nature with a given amplitude, frequency, phase shift, and offset. As an offset sinusoid, the moment of inertia I_{yy}, for example, can be written in the form

$$\tilde{I}_{yy} = A_{yy} \sin\left(\omega_{yy}\delta_B + \phi_{yy}\right) + z_{yy} \tag{3.13}$$

The values of the amplitude, A_{yy}, frequency, ω_{yy}, phase shift, ϕ_{yy}, and offset, z_{yy}, can be determined using a least-squares optimization and the resulting periodic fits for the inertial terms were given by Bolander et al. [75] and reproduced in Table 3.8. These fits are visualized, alongside the data in Table 3.7, in Fig. 3.5. Note that changes in the term $\tilde{I}_{yz}$ are best described using the absolute value of a sinusoid, rather than the phase-shifted sinusoids of $\tilde{I}_{yy}$ and $\tilde{I}_{zz}$. The disparity introduced by using the absolute value of a sinusoid will have no substantial repercussions on the analysis in this work. Note that the changes in the inertia with tail rotation angle are small. This is because the weight of the tail is very small with respect to the total weight of the aircraft.

Table 3.7: Inertial data as a function of BIRE rotation angle of the BIRE variant.

Parameter	**BIRE Rotation Angle, δ_B [deg]**						
	0	±15	±30	±45	±60	±75	±90
Weight, $\tilde{W}$ [lbs]	21,000	21,000	21,000	21,000	21,000	21,000	21,000
Inertia, $\tilde{I}_{xx}$ [slug-ft^2]	9,280	9,280	9,280	9,280	9,280	9,280	9,280
Inertia, $\tilde{I}_{yy}$ [slug-ft^2]	58,127	58,149	58,207	58,288	58,368	58,427	58,449
Inertia, $\tilde{I}_{zz}$ [slug-ft^2]	65,766	65,745	65,686	65,606	65,525	65,466	65,445
Inertia, $\tilde{I}_{xy}$ [slug-ft^2]	0	0	0	0	0	0	0
Inertia, $\tilde{I}_{xz}$ [slug-ft^2]	-5	-5	-5	-5	-5	-5	-5
Inertia, $\tilde{I}_{yz}$ [slug-ft^2]	0	-80	-139	-161	-139	-80	0

Table 3.8: Inertial properties of the BIRE variant.

Parameter	**Value**
Weight, $\tilde{W}$ [lbs]	21,000
Inertia, $\tilde{I}_{xx}$ [slug-ft^2]	9,280
Inertia, $\tilde{I}_{yy}$ [slug-ft^2]	$58{,}288 - 161\cos(2\delta_B)$
Inertia, $\tilde{I}_{zz}$ [slug-ft^2]	$65{,}606 + 161\cos(2\delta_B)$
Inertia, $\tilde{I}_{xy}$ [slug-ft^2]	0
Inertia, $\tilde{I}_{xz}$ [slug-ft^2]	-5
Inertia, $\tilde{I}_{yz}$ [slug-ft^2]	$-161\,\lvert\sin(2\delta_B)\rvert$

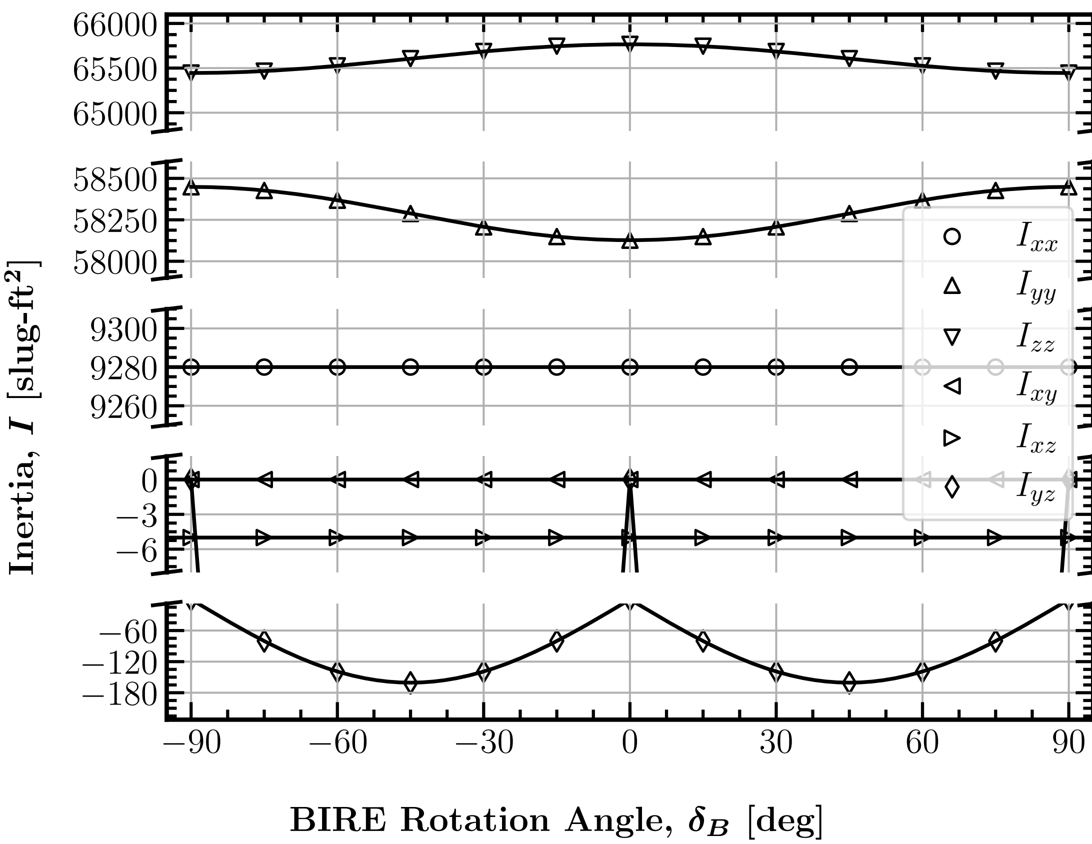


Fig. 3.5: Changes in the moments and products of inertia of the BIRE aircraft as a function of BIRE rotation angle.

CHAPTER 4

FORMULATING AN AERODYNAMIC MODEL FOR THE BASELINE AND BIRE AIRCRAFT

The development of an aerodynamic model for the baseline aircraft and BIRE variant depend on understanding the interactions between the aircraft equations of motion and the aerodynamic forces and moments acting on the aircraft. For a rigid-body aircraft, the rigid-body 6-DOF equations of motion are [76]

$$\begin{Bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{Bmatrix} = \frac{g}{W} \begin{Bmatrix} F_{x_b} \\ F_{y_b} \\ F_{z_b} \end{Bmatrix} + g \begin{Bmatrix} -s_\theta \\ s_\phi c_\theta \\ c_\phi c_\theta \end{Bmatrix} + \begin{Bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{Bmatrix} \tag{4.1}$$

$$\begin{aligned} \begin{Bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{Bmatrix} = & \begin{bmatrix} I_{xx_b} & -I_{xy_b} & -I_{xz_b} \\ -I_{xy_b} & I_{yy_b} & -I_{yz_b} \\ -I_{xz_b} & -I_{yz_b} & I_{zz_b} \end{bmatrix}^{-1} \left(\begin{bmatrix} 0 & -h_{z_b} & h_{y_b} \\ h_{z_b} & 0 & -h_{x_b} \\ -h_{y_b} & h_{x_b} & 0 \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \right. \\ & \left. + \begin{Bmatrix} M_{x_b} + (I_{yy_b} - I_{zz_b})qr + I_{yz_b}(q^2 - r^2) + I_{xz_b}pq - I_{xy_b}pr \\ M_{y_b} + (I_{zz_b} - I_{xx_b})pr + I_{xz_b}(r^2 - p^2) + I_{xy_b}qr - I_{yz_b}pq \\ M_{z_b} + (I_{xx_b} - I_{yy_b})pq + I_{xy_b}(p^2 - q^2) + I_{yz_b}pr - I_{xz_b}qr \end{Bmatrix} \right) \end{aligned} \tag{4.2}$$

$$\begin{Bmatrix} \dot{x}_f \\ \dot{y}_f \\ \dot{z}_f \end{Bmatrix} = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} + \begin{Bmatrix} V_{wx_f} \\ V_{wy_f} \\ V_{wz_f} \end{Bmatrix} \tag{4.3}$$

and

$$\begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} 1 & s_\phi s_\theta / c_\theta & c_\phi s_\theta / c_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi / c_\theta & c_\phi / c_\theta \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \tag{4.4}$$

where the short-hand s and c indicate the sine or cosine, respectively, of the angle in the subscript.

Phillips [77] mentions that Eqs. (4.1) and (4.2) with changes in mass and inertia so long as these are accounted for in the "pseudo-aerodynamic" forces and moments acting on the aircraft. However, changes in the inertia, such as those shown in Table 3.8, are not accounted for in Eq. (4.2). Taking these changes in inertia into account would produce an additional matrix-vector multiplication with the time derivative of the inverted inertia tensor. In this work, it is assumed that the time derivative of the inertia tensor can be neglected. It stands to reason that a diagonally-dominant matrix like the inertia tensor, once inverted, will result in a very small additional term. Future work can justify this approximation further.

In Eq. (4.1), the inertial properties of the aircraft are defined identically to those in Tables 3.4 and 3.8 with the only addition being the gravitational constant g. The terms u, v, and w, represent the translational velocity components of the aircraft in a body-fixed coordinate system and the terms p, q, and r represent the rotational velocity in that same system [77]. Using the traditional Euler angle description, the bank, elevation, and heading angles are given by ϕ, θ, and ψ [78]. V_{wx_f}, V_{wy_f}, and V_{wz_f} in Eq. (4.3) represent the velocity of the wind with respect to to the earth-fixed reference frame. Finally, F_{x_b}, F_{y_b}, and F_{z_b} represent the body-fixed aerodynamic forces on the aircraft, including the thrust generated by the aircraft, and M_{x_b}, M_{y_b}, and M_{z_b} represent the body-fixed aerodynamic moments acting on the aircraft, which also includes the effects of thrust [77]. These moments are often referred to as the rolling, pitching, and yawing moments of the aircraft.

The effects of thrust, as mentioned above, can be represented as an additional term in Eq. (4.1). The forces in the wind system are related to the body-fixed aerodynamic forces by

$$\begin{Bmatrix} F_{x_b} \\ F_{y_b} \\ F_{z_b} \end{Bmatrix} = \begin{Bmatrix} F_{P_x} \\ F_{P_y} \\ F_{P_z} \end{Bmatrix} + \begin{Bmatrix} F_{X_b} \\ F_{Y_b} \\ F_{Z_b} \end{Bmatrix} \tag{4.5}$$

where F_{P_x}, F_{P_y}, and F_{P_z} are the components of propulsive forces in the body-fixed coordinate system and F_{X_b}, F_{Y_b}, and F_{Z_b} are the components of the aerodynamic forces in the body-fixed system. The aerodynamic moments are similarly represented:

$$\begin{Bmatrix} M_{x_b} \\ M_{y_b} \\ M_{z_b} \end{Bmatrix} = \begin{Bmatrix} M_{P_x} \\ M_{P_y} \\ M_{P_z} \end{Bmatrix} + \begin{Bmatrix} M_{X_b} \\ M_{Y_b} \\ M_{Z_b} \end{Bmatrix} \tag{4.6}$$

where M_{P_x}, M_{P_y}, and M_{P_z} are the components of propulsive moments in the body-fixed coordinate system and M_{X_b}, M_{Y_b}, and M_{Z_b} are the components of the aerodynamic moments in the body-fixed system.

A static analysis of an aircraft using Eqs. (4.1) and (4.2) can be performed by setting the rate terms on the left-hand side equal to zero and solving for the state of the aircraft (see Chapter 6). Likewise, Eqs. (4.1), (4.2), and (4.4) can be utilized to analyze control characteristics of each aircraft system (see Chapter 8). In either case, a description of the aerodynamic forces and moments for each aircraft is required to evaluate Eqs. (4.1) and (4.2). The description of these forces and moments constitute an aerodynamic model for the aircraft.

4.1 Aerodynamic Forces and Moments

The aerodynamic forces and moments acting on an aircraft are caused by pressure stresses acting normal to the aircraft surface and shear stresses acting tangent to the aircraft surface [79,80]. Changes in the aerodynamic forces and moments acting on the aircraft can come from changes in the velocity of the aircraft, both translational and rotational, as well as changes in the acceleration experienced by the aircraft. Additionally, changes in the geometry of the aircraft, of which this work will focus on control surface deflections, will have an impact on the aerodynamic forces and moments.

While changes in each of these parameters have an effect on the forces and moments experienced by the aircraft, they do not all do so in equal proportion. As part of this preliminary analysis, the effects of both translational and rotational accelerations on the aerodynamic forces and moments will be ignored. In a trimmed state, where the accelerations in the body-fixed frame are all zero, these accelerations will have no effect on the aerodynamics. Thus, the analysis in Chapter 6 is independent of changes in the aerodynamic forces and moments due to body-fixed accelerations. The simulations performed in Chapter 8 have both translational and rotational accelerations present; however, by analyzing small disturbances to the aircraft, the effects on the aerodynamic forces and moments can be reasonably neglected in this case as well. By ignoring the effects of acceleration on the aerodynamic forces and moments, a model of each will be constructed in terms of the translational and rotational velocities as well as control surface deflections. The relationship between the aerodynamic forces and moments can therefore be written as

$$\mathcal{F}_b = f(u, v, w, p, q, r, \delta) \tag{4.7}$$

where $\mathcal{F}$ represents all of the aerodynamic forces and moments in the body-fixed coordinate system and δ represents each of the control surfaces on the aircraft.

4.1.1 Nondimensional Forces and Moments

The relationships between the aerodynamic forces and moments and the velocities can be made more convenient by considering a model using the nondimensional aerodynamic force and moment coefficients. Nondimensionalizing the forces given in Eq. (4.1) by the dynamic pressure, $\frac{1}{2}\rho V^2 S_w$, yields

$$C_X = \frac{F_{x_b}}{\frac{1}{2}\rho V^2 S_w} \tag{4.8}$$

$$C_Y = \frac{F_{y_b}}{\frac{1}{2}\rho V^2 S_w} \tag{4.9}$$

and

$$C_Z = \frac{F_{z_b}}{\frac{1}{2}\rho V^2 S_w} \tag{4.10}$$

Whereas the nondimensional moments corresponding to the moments in Eq. (4.2) are nondimensionalized using the dynamic pressure and a reference length and are given by

$$C_\ell = \frac{M_{x_b}}{\frac{1}{2}\rho V^2 S_w b_w} \tag{4.11}$$

$$C_m = \frac{M_{y_b}}{\frac{1}{2}\rho V^2 S_w \bar{c}_w} \tag{4.12}$$

and

$$C_n = \frac{M_{z_b}}{\frac{1}{2}\rho V^2 S_w b_w} \tag{4.13}$$

Note that the reference lengths in the definition of the nondimensional aerodynamic moments are arbitrarily chosen to be the span of the main wing, b_w, and the mean aerodynamic chord of the main wing, $\bar{c}_w$.

In addition to the dependencies of the aerodynamic forces and moments themselves, the aerodynamic coefficients in Eqs. (4.8) - (4.13) are generally considered functions of two nondimensional numbers related to the velocity magnitude. The first of these numbers is the Reynolds number [81], defined as

$$R_e \equiv \frac{\rho V l_{\text{ref}}}{\mu} \tag{4.14}$$

where ρ is the density, l_{ref} is a reference length, and μ is the dynamic viscosity. The Reynolds number is a ratio of the effects of inertia to viscosity in a given flow, meaning that higher

Reynolds numbers correspond to flows in which the effects of viscosity are substantially less than the effects of inertia. The aerodynamic coefficients can change dramatically at low Reynolds numbers, where the flow is transitioning from laminar to turbulent conditions [20, 82]. However, at large Reynolds numbers, the aerodynamic coefficients are generally considered weak functions of Reynolds number.

The other nondimensional number is the Mach number, defined as

$$M \equiv \frac{V}{a} \tag{4.15}$$

where a is the speed of sound. Since the speed of sound is directly associated with the compressibility of air, the Mach number indicates regions in which compressibility effects are significant to flow characteristics. Mach numbers below 0.3 are generally considered to be incompressible, and therefore the aerodynamic coefficients are considered to be weak functions of Mach number in this range [83]. Thus, at high Reynolds numbers and low Mach numbers, the effect of velocity magnitude on the force and moment coefficients of the aircraft are negligible [20].

While the aerodynamic coefficients can be considered independent of the velocity magnitude under the preceding circumstances, they are not independent of the velocity as a vector. Specifically, the aerodynamic coefficients are each a function of the aerodynamic angles, α and β. Figure 4.1 shows these angles with regards to the freestream velocity. From the geometry in Fig. 4.1, the angle of attack, α, is defined as

$$\alpha \equiv \tan^{-1}\left(\frac{w}{u}\right) \tag{4.16}$$

The other aerodynamic angle, β, is called the sideslip angle and is defined as

$$\beta \equiv \sin^{-1}\left(\frac{v}{V}\right) \tag{4.17}$$

In a sense, these angles can be seen as a nondimensionalization of the body-fixed velocity components v and w. To solve for the body-fixed velocity components using the aerodynamic angles, it follows from the geometry presented in Fig. 4.1 that

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = V \begin{Bmatrix} c_\alpha c_\beta \\ s_\beta \\ s_\alpha c_\beta \end{Bmatrix} \tag{4.18}$$

With the aerodynamic angles, the effects of the translational velocity on the aerodynamic coefficients can be characterized completely.

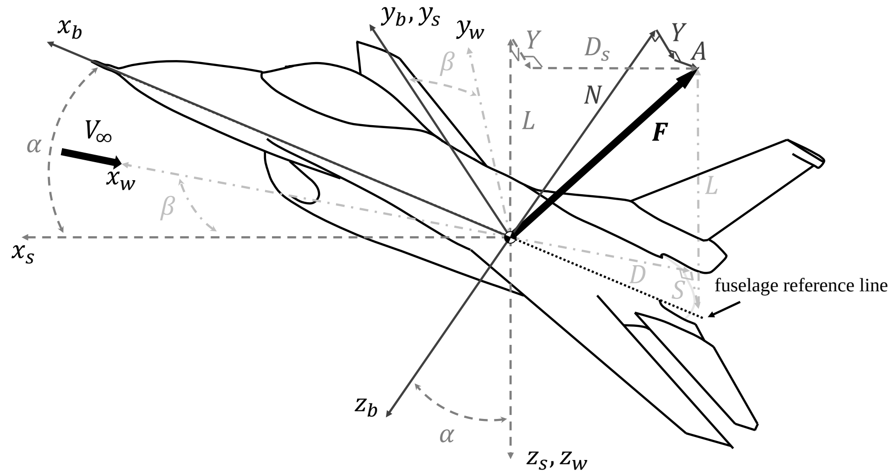


Fig. 4.1: A representation of the transformation between the wind, stability, and body-fixed aircraft coordinate systems.

Since the control surface deflections are already represented nondimensionally in angle form, the final relationship in Eq. (4.7) that must be nondimensionalized is the rotational body-fixed velocities. The traditional nondimensionalizations for the body-fixed rotational velocities are

$$\overline{p} \equiv \frac{pb_w}{2V} \tag{4.19}$$

$$\overline{q} \equiv \frac{q\bar{c}_w}{2V} \tag{4.20}$$

$$\overline{r} \equiv \frac{rb_w}{2V} \tag{4.21}$$

With these definitions, the relationship given in Eq. (4.7) can be rewritten for the nondimensional coefficients in Eqs. (4.8) - (4.13) as

$$C_{\mathcal{F}_b} = f(\alpha, \beta, \overline{p}, \overline{q}, \overline{r}, \delta) \tag{4.22}$$

With this form, an aerodynamic model can be created for each of the aerodynamic coefficients. However, the relationship between the aerodynamic force coefficients and the aerodynamic angles are much more conveniently expressed in the wind coordinate system. Therefore, the transformation between the body-fixed and wind coordinate systems must be given before presenting the forms of the aerodynamic models.

4.1.2 Coordinate Systems

In Chapter 2, the body-fixed and wind coordinate systems were briefly introduced. Here, they are explored in more depth to provide context for the transformations of the aerodynamic coefficients between the two systems. The body-fixed coordinate system is shown in Fig. 4.2. In this system, x_b is aligned with a fuselage reference line and positive out the nose of the aircraft, y_b is measured positive out the right-side of the aircraft, and z_b is positive out the underside of the aircraft.

Though the equations of motion are defined using this body-fixed system, the aerodynamic forces on the aircraft are more readily defined when they are given with respect to a coordinate system aligned with the freestream velocity. Called the wind coordinate system, coordinate system has the x-axis co-linear to, and in the opposite direction of, the freestream velocity. A third coordinate system, called the stability coordinate system, is

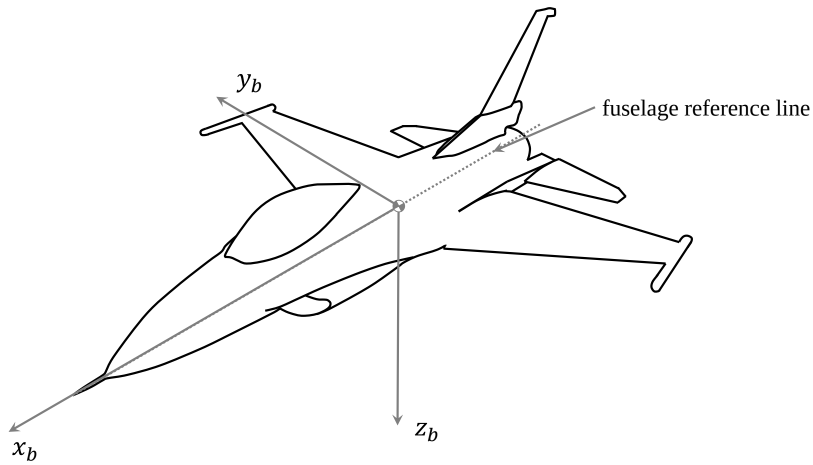


Fig. 4.2: A representation of the body-fixed coordinate system.

sometimes used in flight dynamics analysis and acts as an intermediate system between the body-fixed and wind systems. Each of these coordinate systems are depicted in Fig. 4.1. Since the dynamics of the aircraft system are traditionally written in the body-fixed system and the aerodynamic forces lend themselves to being defined using the wind system, we require a transformation to map the forces from one coordinate system to another.

The transformation from the body-fixed to the wind coordinate system is made through the aerodynamic angles α and β. The force coefficients as given in the body-fixed system are C_X, C_Y, and C_Z, which will be referred to as the body-fixed x, y, and z coefficients. Forces in the body-fixed system are often referred to as the axial force, $A = -F_{x_b}$, the side force $Y = F_{y_b}$, and the normal force $N = -F_{z_b}$, and are labeled as such in Fig. 4.1. In the wind coordinate system, the corresponding force coefficients are C_L, C_S, and C_D, which are referred to as the lift, side force, and drag coefficients respectively. Given the body-fixed force coefficients, the lift, side force, and drag coefficients are calculated as

$$\begin{Bmatrix} C_D \\ C_S \\ C_L \end{Bmatrix} = \begin{Bmatrix} C_A c_\alpha c_\beta - C_Y s_\beta + C_N s_\alpha c_\beta \\ C_A c_\alpha s_\beta + C_Y c_\beta + C_N s_\alpha s_\beta \\ C_N c_\alpha - C_A s_\alpha \end{Bmatrix} \tag{4.23}$$

with A, Y, and N as defined above. If instead the lift, drag, and side force coefficients are given in the wind system and the body-fixed coefficients are desired, the transformation becomes

$$\begin{Bmatrix} C_X \\ C_Y \\ C_Z \end{Bmatrix} = \begin{Bmatrix} -C_A \\ C_Y \\ -C_N \end{Bmatrix} = \begin{Bmatrix} C_D c_\alpha c_\beta + C_S c_\alpha s_\beta - C_L s_\alpha \\ C_S c_\beta - C_D s_\beta \\ C_D s_\alpha c_\beta + C_S s_\alpha s_\beta + C_L c_\alpha \end{Bmatrix} \tag{4.24}$$

Note that the terms longitudinal and lateral will be used to describe the forces and moments acting on the baseline aircraft and BIRE variant throughout the rest of this work. To review, there are two longitudinal forces, lift (or normal force) and drag (or axial force) and one longitudinal moment, the pitching moment. There are three lateral forces and moments: a lateral force, the side force, and two lateral moments, the rolling and yawing moments. In certain scenarios, the longitudinal and lateral forces and moments can be analyzed independently of one another. Since the BIRE fundamentally couples control of pitch with control of yaw in its design, the instances in which they can be considered independently are reduced. With this terminology and the basis for the aerodynamic coefficients and their modeling parameters established, the aerodynamic models used in this work can now be introduced.

4.1.3 Compressibility Corrections

As mentioned previously, flight conditions at high Reynolds numbers and Mach numbers below 0.3 have aerodynamic force and moment coefficients that are nearly independent of velocity magnitude. Fighter aircraft will commonly cruise at Mach numbers substantially higher than 0.3; therefore, it becomes important to adjust the aerodynamic force and moment coefficients to account for the effects of compressibility. The most common, and least accurate, method for adjusting the lift slope for the effects of compressibility in subsonic flow is the Prandtl-Glauert correction [84–86], given as

$$\underline{C}_{L,\alpha} = \frac{C_{L,\alpha}}{\sqrt{1 - M^2}} \tag{4.25}$$

where $\underline{C}$ indicates a coefficient corrected for compressibility. This correction does not take into account the effects of sweep or aspect ratio, which are each important in the aerodynamics of a fighter aircraft.

A more appropriate approximation for the effects of compressibility in subsonic flow can be made by applying a compressibility correction given by Anderson [86]. The corrected lift slope according to the correction given by Anderson is

$$\underline{C}_{L,\alpha} = \frac{C_{L,\alpha}\cos\Lambda_{c/2}}{\sqrt{1 - M^2\cos^2\Lambda_{c/2} + \left[C_{L,\alpha}\cos\Lambda_{c/2})/(\pi R_A)\right]^2} + (C_{L,\alpha}\cos\Lambda_{c/2})/(\pi R_A)} \tag{4.26}$$

This correction takes into account both the aspect ratio and sweep angle, and is therefore a better approximation than a simple Prandtl-Glauert correction. To avoid the over-use of notation, the underbar will be neglected from the coefficients in future equations and it will simply be indicated whether the results presented are made using compressibility corrections.

Generally, compressibility corrections such as given in Eq. (4.26) are applied to the lift slope; however, the correction is derived by considering the effect of compressibility on pressure forces, it is reasonable to expect that these effects can be applied to each of the aerodynamic coefficients [87, 88]. Since the lift, pitching moment, and rolling moment are most substantially influenced by flow over the main wing, its geometric parameters are used in their compressibility corrections in Eq. (4.26). The side force and yawing moment are most significantly influenced by flow over the vertical tail of the baseline aircraft or the rotating tail of the BIRE. Therefore, for those coefficients, the geometric parameters of the vertical tail and horizontal tail, respectively, will be used in Eq. (4.26).

4.2 A Description of the Aerodynamic Models

Since this dissertation represents an exploratory study of the aerodynamic and control properties of an aircraft employing the BIRE control system, the study benefits from simple and efficient analysis methods that identify general aerodynamic trends rapidly. Thus, though many types of aerodynamic models could be used to analyze trim and control, this dissertation will provide a benchmark for exploration of the BIRE concept by presenting two low-fidelity aerodynamic models. One of these aerodynamic models is linear, with the exception of the drag coefficient, while the other includes selected nonlinear effects to better predict the salient aerodynamics of a fighter aircraft. The coefficients in the linear model will be calculated using analytical relationships presented by Phillips [8] that are based on the geometry of each aircraft. Coefficients in the nonlinear model will be determined using aerodynamic data from a numerical lifting-line solver. This lifting-line solver, MachUpX[1], was developed in-house at Utah State University and has been shown in many instances to correctly identify trends in aircraft design [89, 90].

Using low-fidelity methods to construct an aerodynamic model allows for aerodynamic trends to be identified more rapidly at the expense of an inability to describe higher-fidelity physical phenomena for a given flight condition. Aircraft with straight, high aspect-ratio wings are unlikely to experience physical phenomena requiring higher-fidelity aerodynamic analysis in most flight conditions. The baseline aircraft and its BIRE variant have highly-swept lifting surfaces with low aspect ratios, which can enable the development of leading-edge vortexes and spanwise flow. These physical phenomena will not captured be captured in either of the models presented here; however, the general trends necessary for this exploratory study can be obtained. Additionally, the effects of physical phenomena unaccounted for by a low-fidelity model can be approximated by comparing aerodynamic model coefficients produced by the lifting-line data to the same coefficients produced using high-fidelity wind tunnel data for the baseline aircraft [64].

4.2.1 A Linear Aerodynamic Model

Consider the aerodynamic forces and moments acting on an aircraft flying directly into the wind ($\alpha = \beta = 0$) and with zero body-fixed rotation rates and control deflections. The forces and moments acting on the aircraft in this condition will be denoted with a subscript 0. Assuming small disturbances from this condition, the relationship between the aerodynamic forces and the aerodynamic angles, nondimensional body-fixed rotation rates,

[1] https://github.com/usuaero/MachUpX

and control surface deflections are all zero. When linearized about this flight condition, the lift and side force coefficients in the wind coordinate system acting on the baseline aircraft are given by

$$C_L = C_{L_0} + C_{L,\alpha}\alpha + C_{L,\beta}\beta + C_{L,\overline{p}}\overline{p} + C_{L,\overline{q}}\overline{q} + C_{L,\overline{r}}\overline{r} + C_{L,\delta_a}\delta_a + C_{L,\delta_e}\delta_e + C_{L,\delta_r}\delta_r \quad (4.27)$$

and

$$C_S = C_{S_0} + C_{S,\alpha}\alpha + C_{S,\beta}\beta + C_{S,\overline{p}}\overline{p} + C_{S,\overline{q}}\overline{q} + C_{S,\overline{r}}\overline{r} + C_{S,\delta_a}\delta_a + C_{S,\delta_e}\delta_e + C_{S,\delta_r}\delta_r \quad (4.28)$$

The aerodynamic moments in the body-fixed coordinate system acting on the baseline aircraft are given by

$$C_\ell = C_{\ell_0} + C_{\ell,\alpha}\alpha + C_{\ell,\beta}\beta + C_{\ell,\overline{p}}\overline{p} + C_{\ell,\overline{q}}\overline{q} + C_{\ell,\overline{r}}\overline{r} + C_{\ell,\delta_a}\delta_a + C_{\ell,\delta_e}\delta_e + C_{\ell,\delta_r}\delta_r \quad (4.29)$$

$$C_m = C_{m_0} + C_{m,\alpha}\alpha + C_{m,\beta}\beta + C_{m,\overline{p}}\overline{p} + C_{m,\overline{q}}\overline{q} + C_{m,\overline{r}}\overline{r} + C_{m,\delta_a}\delta_a + C_{m,\delta_e}\delta_e + C_{m,\delta_r}\delta_r \quad (4.30)$$

and

$$C_n = C_{n_0} + C_{n,\alpha}\alpha + C_{n,\beta}\beta + C_{n,\overline{p}}\overline{p} + C_{n,\overline{q}}\overline{q} + C_{n,\overline{r}}\overline{r} + C_{n,\delta_a}\delta_a + C_{n,\delta_e}\delta_e + C_{n,\delta_r}\delta_r \quad (4.31)$$

Equations (4.27) – (4.31) constitute a linear aerodynamic model for the aerodynamic forces and moments.

For the case of a symmetric aircraft at small sideslip angles, the change in longitudinal aerodynamic forces and moments with respect to the sideslip angle, lateral rotation rates, and lateral control surfaces are nearly zero

$$\begin{gathered} C_{L,\beta} \approx C_{L,\overline{p}} \approx C_{L,\overline{r}} \approx C_{L,\delta_a} \approx C_{L,\delta_r} \approx 0 \\ C_{m,\beta} \approx C_{m,\overline{p}} \approx C_{m,\overline{r}} \approx C_{m,\delta_a} \approx C_{m,\delta_r} \approx 0 \end{gathered} \quad (4.32)$$

These terms are nearly zero because changes in the lateral aerodynamic parameters about the aircraft plane of symmetry (the lateral or x-z plane) will produce symmetrical effects on aerodynamic forces and moments in the perpendicular (longitudinal or x-y) plane [91]. As an example, pure changes in sideslip on a symmetrical aircraft will produce identical changes to the lift regardless of the sign of the change: that is, $C_L(\beta) = C_L(-\beta)$. Taking the derivative with respect to the sideslip angle, we have $C_{L,\beta}(\beta) = -C_{L,\beta}(-\beta)$ by application of the chain rule. By evaluating our model at a location very near to the zero sideslip condition, we have that $C_{L,\beta}(\varepsilon) = -C_{L,\beta}(\varepsilon)$ with $\varepsilon \approx 0$. With $\beta = 0$ exactly, we have that $C_{L,\beta} = 0$; therefore, we can assume that small sideslip angles result in a derivative that is nearly equal to zero, that is, $C_{L,\beta} \approx 0$.

By the same logic, the change in lateral aerodynamic forces and moments with respect to the longitudinal aerodynamic parameters are approximately equal to zero

$$\begin{aligned} C_{S,\alpha} \approx C_{S,\overline{q}} \approx C_{S,\delta_e} \approx 0 \\ C_{\ell,\alpha} \approx C_{\ell,\overline{q}} \approx C_{\ell,\delta_e} \approx 0 \\ C_{n,\alpha} \approx C_{n,\overline{q}} \approx C_{n,\delta_e} \approx 0 \end{aligned} \tag{4.33}$$

Finally, an aircraft that is symmetric about the lateral plane can have no resulting lateral forces and moments when the aerodynamic parameters are zero. Therefore, by symmetry we also have that

$$C_{S_0} = C_{\ell_0} = C_{n_0} = 0 \tag{4.34}$$

Applying the symmetric assumptions in Eqs. (4.32)-(4.34) to the model in Eqs. (4.27)-(4.31) results in the linear model for the baseline aircraft,

$$C_L = C_{L_0} + C_{L,\alpha}\alpha + C_{L,\overline{q}}\overline{q} + C_{L,\delta_e}\delta_e \tag{4.35}$$

$$C_S = C_{S,\beta}\beta + C_{S,\overline{p}}\overline{p} + C_{S,\overline{r}}\overline{r} + C_{S,\delta_a}\delta_a + C_{S,\delta_r}\delta_r \tag{4.36}$$

$$C_\ell = C_{\ell,\beta}\beta + C_{\ell,\overline{p}}\overline{p} + C_{\ell,\overline{r}}\overline{r} + C_{\ell,\delta_a}\delta_a + C_{\ell,\delta_r}\delta_r \tag{4.37}$$

$$C_m = C_{m_0} + C_{m,\alpha}\alpha + C_{m,\bar{q}}\bar{q} + C_{m,\delta_e}\delta_e \tag{4.38}$$

and

$$C_n = C_{n,\beta}\beta + C_{n,\bar{p}}\bar{p} + C_{n,\bar{r}}\bar{r} + C_{n,\delta_a}\delta_a + C_{n,\delta_r}\delta_r \tag{4.39}$$

In the aerodynamic model presented thus far, the drag coefficient has been neglected. From Prandtl's lifting-line theory, the drag induced by the lift on an aircraft can be estimated as [24]

$$C_{D_i} = \frac{C_L^2}{\pi R_A e_s} \tag{4.40}$$

where

$$e_s = \frac{1}{1+\kappa_D} \tag{4.41}$$

The term κ_D is an induced drag factor can be estimated by referring to plots published by Phillips [24].

Using Eq. (4.35) in Eq. (4.40), the drag can be written in 16 terms with various degrees of non-linearity. Keeping only the linear terms, the induced drag can be written

$$C_D = C_{D_0} + C_{D,\alpha}\alpha + C_{D,\bar{q}}\bar{q} + C_{D,\delta_e}\delta_e \tag{4.42}$$

This description of the drag coefficient neglects the effects of skin friction and parasitic drag, which will affect each of the terms in Eq. (4.42) [92]. As such, the drag predicted by Eq. (4.42) will under-predict the total drag acting on an aircraft in flight, but is still useful for identifying expected trends using the linear model. In addition, the drag coefficient is predominantly used as a way to identify a correct throttle setting for the aircraft.

Adjustments for the BIRE Aircraft

The aerodynamic model used for the BIRE variant differs from that of the baseline aircraft in several ways. Most significantly, the rotation of the empennage by the angle δ_B is an unconventional control mechanism, as it is not a flap deflection. If only small deflections for δ_B were needed, it could be treated in the same way as the control surface deflections in the baseline model; however, in order to provide sufficient lateral control, we expect δ_B to be large in some cases where significant yawing moments are necessary. To account for these large expected values of δ_B, we can instead allow the aerodynamic coefficients to vary with the empennage rotation angle δ_B. Allowing each of the sensitivity coefficients to vary with BIRE rotation angle produces a linear model of the form given in Eqs. (4.27)–(4.31) for each value of δ_B.

Using the model outlined above requires the symmetry assumptions in Eqs. (4.32)-(4.34) to hold. When the horizontal tail is rotated to an angle other than $\delta_B = 0°$ or $\delta_B = \pm 180°$, the aircraft will no longer be symmetric. Since the relationships for the coefficients in Eqs. (4.32)-(4.34) have not been studied analytically, they will have to be estimated based on physical intuition. They will be estimated by considering the trade-off between longitudinal and lateral control given by rotating the horizontal tail.

Changes in the coefficients in Eqs. (4.27)-(4.31) due to changes in BIRE rotation angle can be estimated by assuming that the coefficient is periodic of the form

$$\hat{C} = A \sin(\omega \delta_B + \varphi) + \zeta \tag{4.43}$$

where the nomenclature $\hat{C}$ indicates that the aerodynamic coefficient is a function of the BIRE rotation angle δ_B. The exact nature of these coefficients will be explored in further detail when the coefficients are derived in Chapter 5.

While Eqs. (4.32)-(4.34) are not completely satisfied with the asymmetries present with BIRE rotation, certain coefficients can be reasonably removed from Eqs. (4.27)-(4.31). In terms of the effect of BIRE rotation on the lift coefficient, symmetry in the x-z plane intuitively results in $\hat{C}_{L,\overline{p}} \approx 0$, since the increased angle of attack during rotation on one side of the tail will equal that on the other side of the tail. Likewise, the change in lift due to aileron deflection is small and thus $\hat{C}_{L,\delta_a}$ can be ignored. Since the drag coefficient in the linear model is composed entirely of the effect on drag of the lift, its structure will mirror that of the lift coefficient. The terms $\hat{C}_{m,\overline{p}}$ and $\hat{C}_{m,\delta_a}$ can be ignored for the same reason.

In terms of the lateral aerodynamic coefficients, $\hat{C}_\ell$ is the least-impacted by tail rotation. Again, the symmetry in the x-z plane means that $\hat{C}_{\ell_0} \approx \hat{C}_{\ell,\alpha} \approx \hat{C}_{\ell,\overline{q}} \approx \hat{C}_{\ell,\delta_e} \approx 0$. Also, the lateral coefficients in $\hat{C}_\ell$ should remain relatively constant with BIRE rotation angle. The same is true for $\hat{C}_{S,\overline{p}}$ and $\hat{C}_{S,\delta_a}$ along with the corresponding derivatives in yawing moment coefficient. Changes in C_{S_0} and C_{n_0} should also be very small and caused mostly by changes in downwash, and will be neglected here. Finally, all terms related to the rudder will be dropped from the model, since that control surface does not exist on the BIRE.

Applying the above assumptions gives the linear aerodynamic model for the BIRE as

$$\hat{C}_L = \hat{C}_{L_0} + \hat{C}_{L,\alpha}\alpha + \hat{C}_{L,\beta}\beta + \hat{C}_{L,\overline{q}}\overline{q} + \hat{C}_{L,\overline{r}}\overline{r} + \hat{C}_{L,\delta_e}\delta_e \tag{4.44}$$

$$\hat{C}_S = \hat{C}_{S,\alpha}\alpha + \hat{C}_{S,\beta}\beta + C_{S,\overline{p}}\overline{p}\hat{C}_{S,\overline{q}}\overline{q} + \hat{C}_{S,\overline{r}}\overline{r} + C_{S,\delta_a}\delta_a + \hat{C}_{S,\delta_e}\delta_e \tag{4.45}$$

$$\hat{C}_D = \hat{C}_{D_0} + \hat{C}_{D,\alpha}\alpha + \hat{C}_{D,\beta}\beta + \hat{C}_{D,\overline{q}}\overline{q} + \hat{C}_{D,\overline{r}}\overline{r} + \hat{C}_{D,\delta_e}\delta_e \tag{4.46}$$

$$\hat{C}_\ell = C_{\ell,\beta}\beta + C_{\ell,\overline{p}}\overline{p} + C_{\ell,\overline{r}}\overline{r} + \hat{C}_{\ell,\delta_a}\delta_a \tag{4.47}$$

$$\hat{C}_m = \hat{C}_{m_0} + \hat{C}_{m,\alpha}\alpha + \hat{C}_{m,\beta}\beta + \hat{C}_{m,\overline{q}}\overline{q} + \hat{C}_{m,\overline{r}}\overline{r} + \hat{C}_{m,\delta_e}\delta_e \tag{4.48}$$

and

$$\hat{C}_n = \hat{C}_{n,\alpha}\alpha + \hat{C}_{n,\beta}\beta + \hat{C}_{n,\overline{q}}\overline{q} + \hat{C}_{n,\overline{r}}\overline{r} + \hat{C}_{n,\delta_e}\delta_e \tag{4.49}$$

4.2.2 A Non-Linear Aerodynamic Model

The aerodynamic model given in Eqs. (4.35)-(4.39) and (4.42) is accurate only for small aerodynamic angles, nondimensional rotation rates, and control-surface deflections; therefore, capturing relevant aerodynamics over a larger range of angles below stall requires that some additional, nonlinear relationships be included. The nature of these relationships can be understood by applying the results of analytical studies. For the sake of brevity in this analysis, we will define a pseudo-lift coefficient that neglects changes in lift due to sideslip, rotation rates, and control surface deflections as

$$C_{L_1} \equiv C_{L_0} + C_{L,\alpha}\alpha \tag{4.50}$$

We will also define a pseudo-side-force coefficient that neglects changes in side force due to angle of attack, rotation rates, and control surface deflections as

$$C_{S_1} \equiv C_{S_0} + C_{S,\beta}\beta \tag{4.51}$$

From lifting-line theory it can be shown that the effects of rolling rate and aileron deflection on the yawing moment can each be approximated as a linear function of lift [20,93]. Hence, the influence of rolling rate on yawing moment given in Eq. (4.31), $C_{n,\overline{p}}\overline{p}$, can be approximated as $(C_{n,L\overline{p}}C_{L_1} + C_{n,\overline{p}})\,\overline{p}$. Likewise, the influence of aileron deflection on yawing moment, $C_{n,\delta_a}\delta_a$ can be approximated as $(C_{n,L\delta_a}C_{L_1} + C_{n,\delta_a})\,\delta_a$. Additionally, an analytic approximation by Phillips [20] shows that the change in rolling moment with respect to yawing rate depends in a linear fashion on the lift coefficient of the main wing. Thus, the influence of yawing rate on the rolling moment, $C_{\ell,\overline{r}}\overline{r}$, can be written as $(C_{\ell,L\overline{r}}C_{L_1} + C_{\ell,\overline{r}})\,\overline{r}$.

Other non-linear relationships have not been investigated analytically and instead apply specifically to the baseline aircraft. These relationships are identified based upon trends observed in wind tunnel data taken from the baseline aircraft as published by Nguyen et al. [64]. Trends in the wind tunnel data indicate that the change in side force with respect to the roll rate can be written as a function of lift as $(C_{S,L\overline{p}}C_{L_1} + C_{S,\overline{p}})\,\overline{p}$. Applying all of the preceding relationships to Eqs. (4.28), (4.29), and (4.31) yields

$$\begin{aligned} C_S = C_{S_1} + C_{S,\alpha}\alpha + (C_{S,L\overline{p}}C_{L_1} + C_{S,\overline{p}})\overline{p} + C_{S,\overline{q}}\overline{q} + C_{S,\overline{r}}\overline{r} \\ + C_{S,\delta_a}\delta_a + C_{S,\delta_e}\delta_e + C_{S,\delta_r}\delta_r \end{aligned} \tag{4.52}$$

$$
\begin{aligned}
C_\ell = C_{\ell_0} + C_{\ell,\alpha}\alpha + C_{\ell,\beta}\beta + C_{\ell,\overline{p}}\overline{p} + C_{\ell,\overline{q}}\overline{q} + (C_{\ell,L\overline{r}}C_{L_1} + C_{\ell,\overline{r}})\overline{r} \\
+ C_{\ell,\delta_a}\delta_a + C_{\ell,\delta_e}\delta_e + C_{\ell,\delta_r}\delta_r
\end{aligned}
\tag{4.53}
$$

and

$$
\begin{aligned}
C_n = C_{n_0} + C_{n,\alpha}\alpha + C_{n,\beta}\beta + (C_{n,L\overline{p}}C_{L_1} + C_{n,\overline{p}})\overline{p} + C_{n,\overline{q}}\overline{q} + C_{n,\overline{r}}\overline{r} \\
+ (C_{n,L\delta_a}C_{L_1} + C_{n,\delta_a})\delta_a + C_{n,\delta_e}\delta_e + C_{n,\delta_r}\delta_r
\end{aligned}
\tag{4.54}
$$

Additional nonlinear terms can be included in our model for the drag coefficient by using our understanding of the relationship that the drag has with the lift and side force. From lifting-line theory and a host of computational and experimental results, it is well-understood that drag below stall can be approximated as a quadratic function of the lift coefficient [94–96]. As an extension of this principle, the effects of side force on drag can also be approximated using a quadratic.

Therefore, an approximation for the drag coefficient can be expressed as

$$
C_D = C_{D_0} + C_{D,L}C_L + C_{D,L^2}C_L^2 + C_{D,S}C_S + C_{D,S^2}C_S^2 \tag{4.55}
$$

where C_{D_0} is the drag at zero lift and zero side force. Not only does writing the drag coefficient in this form provide additional fidelity over the form given in Eq. (4.42), it expands and highlights additional relationships between the drag and the aerodynamic modeling parameters. Keeping all of the terms created by substituting Eqs. (4.27) and (4.52) into Eq. (4.55) would result in 201 terms. Not only would an analysis of that magnitude be extremely computationally costly, many of the terms would add very little to the fidelity of the model. By removing terms based upon an order-of-magnitude analysis, inconsequential terms can be removed from the expanded form of Eq. (4.55) to maintain simplicity in the model and highlight larger-scale trends in the drag acting on the aircraft.

Using the psuedo-force coefficients from Eqs. (4.50) and (4.51) in Eqs. (4.27) and (4.52), and applying the resulting expressions to Eq. (4.55) gives

$$
\begin{aligned}
C_D = C_{D_0} &+ C_{D,L}\big(C_{L_1} + C_{L,\beta}\beta + C_{L,\overline{p}}\overline{p} + C_{L,\overline{q}}\overline{q} + C_{L,\overline{r}}\overline{r} \\
&\qquad + C_{L,\delta_a}\delta_a + C_{L,\delta_e}\delta_e + C_{L,\delta_r}\delta_r\big) \\
&+ C_{D,L^2}\big(C_{L_1} + C_{L,\beta}\beta + C_{L,\overline{p}}\overline{p} + C_{L,\overline{q}}\overline{q} + C_{L,\overline{r}}\overline{r} \\
&\qquad + C_{L,\delta_a}\delta_a + C_{L,\delta_e}\delta_e + C_{L,\delta_r}\delta_r\big)^2 \\
&+ C_{D,S}\big(C_{S_1} + C_{S,\alpha}\alpha + (C_{S,L\overline{p}}C_{L_1} + C_{S,\overline{p}})\overline{p} + C_{S,\overline{q}}\overline{q} + C_{S,\overline{r}}\overline{r} \\
&\qquad + C_{S,\delta_a}\delta_a + C_{S,\delta_e}\delta_e + C_{S,\delta_r}\delta_r\big) \\
&+ C_{D,S^2}\big(C_{S_1} + C_{S,\alpha}\alpha + (C_{S,L\overline{p}}C_{L_1} + C_{S,\overline{p}})\overline{p} + C_{S,\overline{q}}\overline{q} + C_{S,\overline{r}}\overline{r} \\
&\qquad + C_{S,\delta_a}\delta_a + C_{S,\delta_e}\delta_e + C_{S,\delta_r}\delta_r\big)^2
\end{aligned}
\tag{4.56}
$$

The products of the aerodynamic angles, dimensionless rotation rates, and control-surface deflections in Eq. (4.56) can be neglected in nearly all cases, since the terms themselves are usually small. An exception to the neglected terms is the square of the elevator deflection, δ_e^2, which is another instance of using familiarity with the wind tunnel data to make informed modeling decisions. The reason for the importance of the δ_e^2 term is not well-understood and is not the focus of this work [64]. Another example of utilizing trends in the experimental data is that the change in drag with pitch rate can be a significant function of the square of the lift coefficient. Therefore, the term $(C_{D,L\overline{q}}C_{L_1} + C_{D,\overline{q}})$ is expanded to include $C_{D,L^2\overline{q}}C_{L_1}^2\overline{q}$, which better approximates trends in the experimental data.

In addition to neglecting terms with products of negligible influence, the drag model can be simplified by combining and renaming redundant constants. As an example, the combination of coefficients $C_{D,L}C_{L,\overline{p}}$ is equivalent to

$$
C_{D,L}C_{L,\overline{p}} = \frac{\partial C_D}{\partial C_L}\frac{\partial C_L}{\partial \overline{p}} = \frac{\partial C_D}{\partial \overline{p}} = C_{D,\overline{p}}
\tag{4.57}
$$

This coefficient shows that changes in drag are linearly related to the change in rolling rate experienced by the aircraft. Equation (4.56) also contains the term $C_{D,S}C_{S,\overline{p}}$, which can also be simplified to $C_{D,\overline{p}}$. There is no additional fidelity added to the drag model by keeping both of these terms. Whether the change in drag is due to the effect of rolling rate on lift or its effect on side force is not important in the analysis that follows. By including only the term $C_{D,\overline{p}}$, we capture the changes in drag due to changes in rolling rate caused by either lift or side force. An application of this reasoning to other constants allows many of the terms from Eq. (4.56) to be combined and renamed in a similar fashion.

Removing and combining the aforementioned coefficients results in a nonlinear drag model for a general aircraft below stall of the form

$$\begin{aligned}
C_D = {} & C_{D_0} + C_{D,L}C_{L_1} + C_{D,L^2}C_{L_1}^2 + C_{D,S}C_{S_1} + C_{D,S^2}C_{S_1}^2 \\
& + C_{D,S\alpha}C_{S_1}\alpha + C_{D,L\beta}C_{L_1}\beta \\
& + \left[C_{D,L\overline{p}}C_{L_1} + \left(C_{D,S\overline{p}} + C_{D,SL\overline{p}}C_{L_1}\right)C_{S_1} + C_{D,\overline{p}}\right]\overline{p} \\
& + \left(C_{D,L^2\overline{q}}C_{L_1}^2 + C_{D,L\overline{q}}C_{L_1} + C_{D,S\overline{q}}C_{S_1} + C_{D,\overline{q}}\right)\overline{q} \\
& + \left(C_{D,L\overline{r}}C_{L_1} + C_{D,S\overline{r}}C_{S_1} + C_{D,\overline{r}}\right)\overline{r} + \left(C_{D,L\delta_a}C_{L_1} + C_{D,S\delta_a}C_{S_1} + C_{D,\delta_a}\right)\delta_a \\
& + \left(C_{D,L\delta_e}C_{L_1} + C_{D,S\delta_e}C_{S_1} + C_{D,\delta_e}\right)\delta_e + C_{D,\delta_e^2}\delta_e^2 \\
& + \left(C_{D,L\delta_r}C_{L_1} + C_{D,S\delta_r}C_{S_1} + C_{D,\delta_r}\right)\delta_r
\end{aligned} \tag{4.58}$$

A slightly simpler model can be obtained without significant loss of fidelity by neglecting coupling terms between longitudinal and lateral components. Neglecting the terms $C_{L_1}\beta$, $C_{L_1}\overline{p}$, $C_{L_1}\overline{r}$, $C_{L_1}\delta_a$, $C_{L_1}\delta_r$, $C_{S_1}\alpha$, $C_{S_1}\overline{q}$, and $C_{S_1}\delta_e$ gives

$$\begin{aligned}
C_D = {} & C_{D_0} + C_{D,L}C_{L_1} + C_{D,L^2}C_{L_1}^2 + C_{D,S}C_{S_1} + C_{D,S^2}C_{S_1}^2 \\
& + \left(C_{D,S\overline{p}}C_{S_1} + C_{D,\overline{p}}\right)\overline{p} + \left(C_{D,L^2\overline{q}}C_{L_1}^2 + C_{D,L\overline{q}}C_{L_1} + C_{D,\overline{q}}\right)\overline{q} \\
& + \left(C_{D,S\overline{r}}C_{S_1} + C_{D,\overline{r}}\right)\overline{r} + \left(C_{D,S\delta_a}C_{S_1} + C_{D,\delta_a}\right)\delta_a \\
& + \left(C_{D,L\delta_e}C_{L_1} + C_{D,\delta_e}\right)\delta_e + C_{D,\delta_e^2}\delta_e^2 + \left(C_{D,S\delta_r}C_{S_1} + C_{D,\delta_r}\right)\delta_r
\end{aligned} \tag{4.59}$$

By retaining only 19 of the 201 coefficients from Eq. (4.55), this nonlinear drag model retains enough fidelity to be useful for analysis while also remaining computationally efficient. Equations (4.27), (4.30), (4.59), and (4.52)–(4.54) constitute a general, nonlinear aerodynamic model for the lift, side force, drag, and aerodynamic moments, respectively, of an aircraft below stall. Since the baseline aircraft is symmetric about the x-z plane, further simplifications, similar to those outlined in the description of the linear model, can be made.

Symmetric Aircraft

With the assumptions of symmetry given in Eqs. (4.32)–(4.34) applied, the general model outlined above can be simplified into a nonlinear aerodynamic model for symmetric aircraft, given by

$$C_L = C_{L_0} + C_{L,\alpha}\alpha + C_{L,\overline{q}}\overline{q} + C_{L,\delta_e}\delta_e \tag{4.60}$$

$$C_S = C_{S,\beta}\beta + (C_{S,L\overline{p}}C_{L_1} + C_{S,\overline{p}})\overline{p} + C_{S,\overline{r}}\overline{r} + C_{S,\delta_a}\delta_a + C_{S,\delta_r}\delta_r \tag{4.61}$$

$$\begin{aligned} C_D = {} & C_{D_0} + C_{D,L}C_{L_1} + C_{D,L^2}C_{L_1}^2 + C_{D,S^2}C_{S_1}^2 \\ & + C_{D,S\overline{p}}C_{S_1}\overline{p} + \left(C_{D,L^2\overline{q}}C_{L_1}^2 + C_{D,L\overline{q}}C_{L_1} + C_{D,\overline{q}}\right)\overline{q} + C_{D,S\overline{r}}C_{S_1}\overline{r} \\ & + C_{D,S\delta_a}C_{S_1}\delta_a + \left(C_{D,L\delta_e}C_{L_1} + C_{D,\delta_e}\right)\delta_e + C_{D,\delta_e^2}\delta_e^2 + C_{D,S\delta_r}C_{S_1}\delta_r \end{aligned} \tag{4.62}$$

$$C_\ell = C_{\ell,\beta}\beta + C_{\ell,\overline{p}}\overline{p} + (C_{\ell,L\overline{r}}C_{L_1} + C_{\ell,\overline{r}})\overline{r} + C_{\ell,\delta_a}\delta_a + C_{\ell,\delta_r}\delta_r \tag{4.63}$$

$$C_m = C_{m_0} + C_{m,\alpha}\alpha + C_{m,\overline{q}}\overline{q} + C_{m,\delta_e}\delta_e \tag{4.64}$$

$$C_n = C_{n,\beta}\beta + (C_{n,L\overline{p}}C_{L_1} + C_{n,\overline{p}})\overline{p} + C_{n,\overline{r}}\overline{r} + (C_{n,L\delta_a}C_{L_1} + C_{n,\delta_a})\delta_a + C_{n,\delta_r}\delta_r \tag{4.65}$$

where C_{L_1} is given in Eq. (4.50) and C_{S_1} is given in Eq. (4.51) with $C_{S_0} = 0$. Equations (4.60)–(4.65) comprise a reasonable aerodynamic model below stall for a symmetric aircraft and will be used for the evaluation of the aerodynamics of the baseline aircraft in this work.

Adjustments for the BIRE Aircraft

As mentioned in the section treating the linear aerodynamic model, any nonzero value of δ_B will result in an aircraft configuration that is not symmetric about the x-z plane, and so we cannot apply the symmetric assumptions given in Eqs. (4.32)–(4.34). In this case, the form of the BIRE aerodynamic model will follow that of Eqs. (4.27), (4.52), (4.59), (4.53), (4.30), and (4.54). Removing the dependence on rudder and letting hats over the coefficients represent each coefficient's dependence on BIRE rotation angle, we can re-write these equations as

$$\hat{C}_L = \hat{C}_{L_0} + \hat{C}_{L,\alpha}\alpha + \hat{C}_{L,\beta}\beta + \hat{C}_{L,\overline{p}}\overline{p} + \hat{C}_{L,\overline{q}}\overline{q} + \hat{C}_{L,\overline{r}}\overline{r} + \hat{C}_{L,\delta_a}\delta_a + \hat{C}_{L,\delta_e}\delta_e \tag{4.66}$$

$$\hat{C}_S = \hat{C}_{S_0} + \hat{C}_{S,\alpha}\alpha + \hat{C}_{S,\beta}\beta + \left(\hat{C}_{S,L\overline{p}}\hat{C}_{L_1} + \hat{C}_{S,\overline{p}}\right)\overline{p} + \hat{C}_{S,\overline{q}}\overline{q} + \hat{C}_{S,\overline{r}}\overline{r} + \hat{C}_{S,\delta_a}\delta_a + \hat{C}_{S,\delta_e}\delta_e \tag{4.67}$$

$$\begin{aligned}\hat{C}_D = {} & \hat{C}_{D_0} + \hat{C}_{D,L}\hat{C}_{L_1} + \hat{C}_{D,L^2}\hat{C}_{L_1}^2 + \hat{C}_{D,S}\hat{C}_{S_1} + \hat{C}_{D,S^2}\hat{C}_{S_1}^2 \\ & + \left(\hat{C}_{D,S\overline{p}}\hat{C}_{S_1} + \hat{C}_{D,\overline{p}}\right)\overline{p} + \left(\hat{C}_{D,L^2\overline{q}}\hat{C}_{L_1}^2 + \hat{C}_{D,L\overline{q}}\hat{C}_{L_1} + \hat{C}_{D,\overline{q}}\right)\overline{q} \\ & + \left(\hat{C}_{D,S\overline{r}}\hat{C}_{S_1} + \hat{C}_{D,\overline{r}}\right)\overline{r} + \left(C_{D,S\delta_a}\hat{C}_{S_1} + \hat{C}_{D,\delta_a}\right)\delta_a \\ & + \left(\hat{C}_{D,L\delta_e}\hat{C}_{L_1} + \hat{C}_{D,\delta_e}\right)\delta_e + \hat{C}_{D,\delta_e^2}\delta_e^2\end{aligned} \tag{4.68}$$

$$\hat{C}_\ell = \hat{C}_{\ell_0} + \hat{C}_{\ell,\alpha}\alpha + \hat{C}_{\ell,\beta}\beta + \hat{C}_{\ell,\overline{p}}\overline{p} + \hat{C}_{\ell,\overline{q}}\overline{q} + \left(\hat{C}_{\ell,L\overline{r}}\hat{C}_{L_1} + \hat{C}_{\ell,\overline{r}}\right)\overline{r} + \hat{C}_{\ell,\delta_a}\delta_a + \hat{C}_{\ell,\delta_e}\delta_e \tag{4.69}$$

$$\hat{C}_m = \hat{C}_{m_0} + \hat{C}_{m,\alpha}\alpha + \hat{C}_{m,\beta}\beta + \hat{C}_{m,\overline{p}}\overline{p} + \hat{C}_{m,\overline{q}}\overline{q} + \hat{C}_{m,\overline{r}}\overline{r} + \hat{C}_{m,\delta_a}\delta_a + \hat{C}_{m,\delta_e}\delta_e \tag{4.70}$$

and

$$\begin{aligned}\hat{C}_n = {} & \hat{C}_{n_0} + \hat{C}_{n,\alpha}\alpha + \hat{C}_{n,\beta}\beta + \left(\hat{C}_{n,L\overline{p}}\hat{C}_{L_1} + \hat{C}_{n,\overline{p}}\right)\overline{p} + \hat{C}_{n,\overline{q}}\overline{q} + \hat{C}_{n,\overline{r}}\overline{r} \\ & + \left(\hat{C}_{n,L\delta_a}\hat{C}_{L_1} + \hat{C}_{n,\delta_a}\right)\delta_a + \hat{C}_{n,\delta_e}\delta_e\end{aligned} \tag{4.71}$$

The pseudo-lift and -side force coefficients, C_{L_1} and C_{S_1} can be re-defined using this hat notation to be

$$\hat{C}_{L_1} \equiv \hat{C}_{L_0} + \hat{C}_{L,\alpha}\alpha \tag{4.72}$$

and

$$\hat{C}_{S_1} \equiv \hat{C}_{S_0} + \hat{C}_{S,\beta}\beta \tag{4.73}$$

These coefficients vary with the BIRE rotation angle according to the model given in Eq. (4.43).

CHAPTER 5

EVALUATION OF THE AERODYNAMIC COEFFICIENTS

With the aerodynamic models presented, the individual coefficients can be evaluated according to the methodologies referenced in Section 4.2. Recall that the coefficients in the linear aerodynamic model will be evaluated using analytical relationships backed by physical intuition to estimate the effect of tail rotation on the BIRE aerodynamics. Evaluation of the coefficients in the linear model is not complete, but results for the longitudinal coefficients are given in Appendix A. The physical intuition used here will be, in part, supported by aerodynamic data from MachUpX, an in-house numerical lifting-line tool, which will be used to estimate the coefficients in the nonlinear aerodynamic model for both aircraft geometries.

The resulting coefficients will then be used in their respective aerodynamic models to estimate the aerodynamic forces and moments acting on each aircraft as a function of the aerodynamic modeling parameters. The evaluation of each of these coefficients is dependent on the aerodynamics of the airfoils which make up the lifting surfaces of each aircraft. Therefore, before proceeding further with the coefficient evaluation, the airfoils used in this study will be presented with an aerodynamic analysis justifying their modeling.

5.1 Airfoil Aerodynamics

The airfoils used on each lifting surface of the baseline and BIRE aircraft are included in Tables 3.2 and 3.5, respectively. Fox and Forrest indicate that the horizontal and vertical tails of the baseline aircraft are constructed with biconvex airfoils as indicated in Table 3.1 [62]. The available aerodynamic data for biconvex airfoils is limited; therefore, the baseline aircraft and BIRE variant instead use thin, symmetric NACA airfoils of approximately the same thickness-to-chord ratio as the biconvex airfoils indicated. Figure 5.1 shows each of the airfoils given in Tables 3.2 and 3.5 and Tables 5.1–5.3 give the stations and ordinates of the airfoils measured in percent-chord.

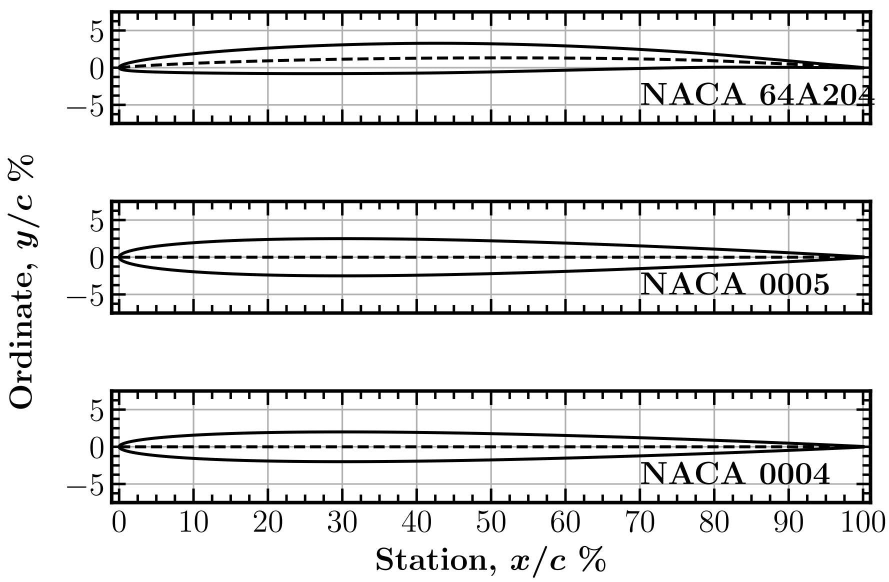


Fig. 5.1: Basic forms of the NACA 64A204, 0005, and 0004 airfoils. Measurements taken as a percentage of the airfoil chord.

Upper Surface		**Lower Surface**	
Station, x_u/c	**Ordinate, y_u/c**	**Station, x_l/c**	**Ordinate, y_l/c**
0	0	0	0
1.661	0.723	1.747	-0.434
6.647	1.507	6.751	-0.631
14.594	2.269	14.695	-0.751
24.96	2.882	25.04	-0.799
37.036	3.237	37.082	-0.759
49.998	3.212	50.002	-0.571
62.956	2.808	62.926	-0.274
75.026	2.148	74.974	-0.008
85.384	1.333	85.326	0.059
93.316	0.612	93.287	0.025
98.302	0.174	98.291	-0.03
100	-0.016	100	-0.016

Table 5.1: Stations and ordinates of the NACA 64A204 airfoil given in percent of airfoil chord.

For the linear aerodynamic analysis, evaluating the aerodynamic coefficients requires knowledge of several properties of each airfoil section. These include the lift slope, $\tilde{C}_{L,\alpha}$,

Upper Surface		Lower Surface	
Station, x_u/c	**Ordinate, y_u/c**	**Station, x_l/c**	**Ordinate, y_l/c**
0	0	0	0
1.704	0.913	1.704	-0.913
6.699	1.673	6.699	-1.673
14.645	2.212	14.645	-2.212
25	2.476	25	-2.476
37.059	2.458	37.059	-2.458
50	2.206	50	-2.206
62.941	1.798	62.941	-1.798
75	1.317	75	-1.317
85.355	0.838	85.355	-0.838
93.301	0.429	93.301	-0.429
98.296	0.151	98.296	-0.151
100	0.052	100	-0.052

Table 5.2: Stations and ordinates of the NACA 0005 airfoil given in percent of airfoil chord.

Upper Surface		Lower Surface	
Station, x_u/c	**Ordinate, y_u/c**	**Station, x_l/c**	**Ordinate, y_l/c**
0	0	0	0
1.704	0.73	1.704	-0.73
6.699	1.338	6.699	-1.338
14.645	1.769	14.645	-1.769
25	1.98	25	-1.98
37.059	1.966	37.059	-1.966
50	1.765	50	-1.765
62.941	1.438	62.941	-1.438
75	1.053	75	-1.053
85.355	0.67	85.355	-0.67
93.301	0.343	93.301	-0.343
98.296	0.121	98.296	-0.121
100	0.042	100	-0.042

Table 5.3: Stations and ordinates of the NACA 0004 airfoil given in percent of airfoil chord.

pitching moment about the aerodynamic center, $\tilde{C}_{m_{\mathrm{ac}}}$, zero-lift angle of attack, α_{L_0}, and the change in pitching moment with flap deflection, $\tilde{C}_{m,\delta}$. These coefficients overlap with many of those required for the nonlinear aerodynamic model. The nonlinear aerodynamic model coefficients will be calculated using MachUpX, which can characterize an airfoil using a database, polynomial fit, or linear characteristics. To define the database and polynomial fits used by MachUpX, the Airfoil Database[1] python model can be used to generate airfoil section data using XFOIL[2]. Unfortunately, the airfoils used by the baseline and BIRE aircraft are so thin that XFOIL cannot converge consistently. Therefore, linear airfoil characterizations for the NACA 64A204, 0005, and 0004 airfoils will be generated by a combination of thin airfoil theory and available wind tunnel data.

To characterize a linear airfoil model in MachUpX, the following will be defined: the zero-lift angle of attack, lift slope, pitching moment at zero lift, moment slope, and the coefficients for the drag polar. From thin airfoil theory, the section lift slope is defined as [97]

$$\tilde{C}_{L,\alpha} = 2\pi \tag{5.1}$$

while the zero-lift angle of attack is given as [97]

$$\alpha_{L_0} = \frac{1}{\pi}\int_{\theta=0}^{\pi} \frac{dy_c}{dx}\left(1-\cos\theta\right)\, d\theta \tag{5.2}$$

In Eq. (5.2), the term $\frac{dy_c}{dx}$ is the slope of the camber line in the chordwise direction and θ represents a change of variables where

$$x(\theta) = \frac{c}{2}\left(1-\cos\theta\right) \tag{5.3}$$

with c equal to the chord of the airfoil. Using the coordinates for the outline of the NACA 64A204 airfoil in Table 5.1, the camber line can be estimated using an iterative method included in the Airfoil Database module. A sampling of the camber line is included in Table 5.4 nd can be numerically differentiated to find $\frac{dy_c}{dx}$.

The main wing of each aircraft employs a NACA 6A-series airfoil designed to provide a high critical Mach number while reducing the complexities of fabrication present in the NACA 6-series [98]. Loftin Jr. provides aerodynamic data for several 6A-series airfoils, but not that of the 64A204 used in the baseline aircraft [98]. Based on wind tunnel measure-

[1] https://github.com/usuaero/AirfoilDatabase
[2] http://web.mit.edu/drela/Public/web/xfoil/

Station, x_c/c	**Ordinate, y_c/c**	**Station, x_c/c**	**Ordinate, y_c/c**
0	0	51.02	1.321
2.041	0.169	53.061	1.32
4.082	0.299	55.102	1.316
6.122	0.409	57.143	1.308
8.163	0.507	59.184	1.297
10.204	0.594	61.224	1.282
12.245	0.674	63.265	1.264
14.286	0.747	65.306	1.241
16.327	0.813	67.347	1.214
18.367	0.875	69.388	1.183
20.408	0.931	71.429	1.147
22.449	0.983	73.469	1.105
24.49	1.03	75.51	1.058
26.531	1.074	77.551	1.003
28.571	1.113	79.592	0.939
30.612	1.149	81.633	0.861
32.653	1.181	83.673	0.772
34.694	1.21	85.714	0.68
36.735	1.235	87.755	0.587
38.776	1.257	89.796	0.491
40.816	1.276	91.837	0.391
42.857	1.292	93.878	0.29
44.898	1.304	95.918	0.191
46.939	1.313	97.959	0.089
48.98	1.319	100	-0.016

Table 5.4: Camber line for the NACA 64A204 airfoil sampled at 50 stations. All measurements in percent-chord.

ments, Loftin Jr. found that the section lift slope of the 6A-series airfoils he studied were nearly independent of airfoil thickness [98]. Figure 5.2 shows the lift slope taken from Loftin Jr.'s wind tunnel results for three airfoils: the 64A210, 64A212, and 64A215, compared to the results of thin airfoil theory [98]. The section lift slopes of the other 6A-series airfoils are nearly the same as that predicted by thin airfoil theory , we can assume that the NACA 64A204 employed in the baseline and BIRE aircraft can reasonably be approximated using thin airfoil theory.

The zero-lift angle of attack measured by Loftin Jr. for the three airfoils mentioned previously are included in Fig. 5.3 in comparison with thin airfoil theory. Differences in

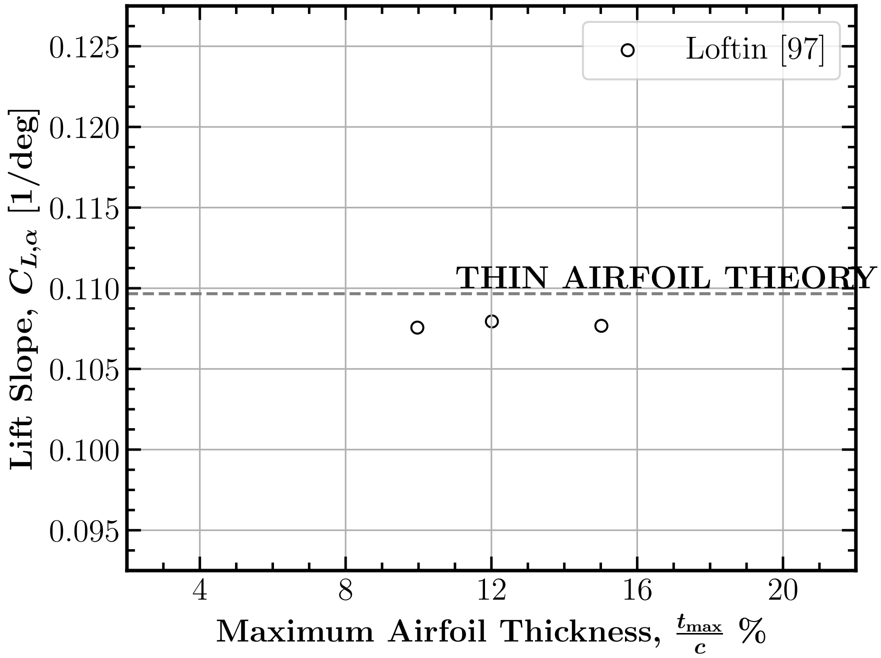


Fig. 5.2: A comparison of the section lift slope of three NACA 6A-series airfoils to thin airfoil theory results.

zero-lift angle of attack between the wind tunnel data and thin airfoil theory are more pronounced in this case than in Fig. 5.2. However, it is reasonable to assume that the trend towards the thin airfoil theory value given by the airfoils of 12% and 10% thickness makes the thin airfoil theory result an appropriate approximation for the 64A204.

Thin airfoil theory shows that the quarter-chord of an airfoil is the location of its aerodynamic center [97]. Thus, the quarter-chord pitching moment is the pitching moment about the aerodynamic center, and is independent of angle of attack [97]. From this knowledge, thin airfoil theory suggests that the zero-lift pitching moment, $\tilde{C}_{m_{L_0}}$ is equivalent to the section quarter-chord pitching moment, calculated as [97]

$$\tilde{C}_{m_{c/4}} = \frac{1}{2} \int_{\theta=0}^{\pi} \frac{dy_c}{dx} \left[\cos(2\theta) - \cos\theta\right] d\theta \tag{5.4}$$

Loftin Jr. notes that the location of the aerodynamic center in the 64A-series airfoils he studied were nearly constant and located just aft of the quarter-chord [98]. Therefore, for

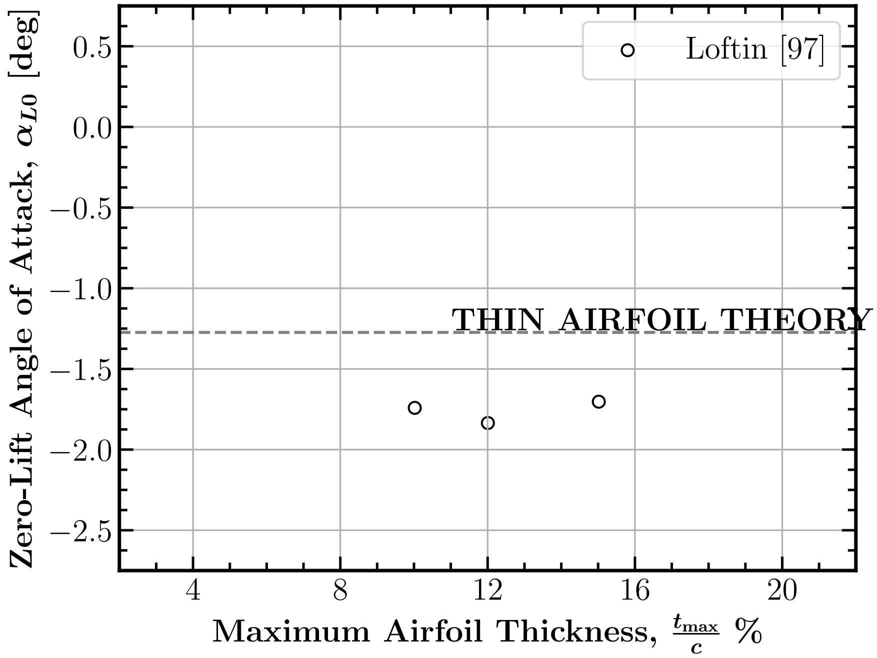


Fig. 5.3: A comparison of the section zero-lift angle of attack of three NACA 6A-series airfoils to thin airfoil theory results.

the purposes of this study, the thin airfoil value of the quarter-chord pitching moment will be considered equivalent to the zero-lift pitching moment. As a result, the section pitching moment slope, $\tilde{C}_{m,\alpha}$, will be set to zero.

Finally, the characterization of the main wing airfoil can be completed by considering the section drag polar for the 64A204 airfoil. Thin airfoil theory presents no way in which to calculate the components of the drag polar. However, in comparing the 6A-series airfoils to the 6-series airfoils, Loftin noted that the minimum drag coefficients are nearly identical between series [98].

Under this assumption, additional 6-series airfoil data can be used to estimate the minimum drag coefficient of the 64A204 airfoil. This data can be obtained by referring to Abbot et al., who reported wind tunnel data for the 64-206 airfoil in their work [99]. The minimum drag coefficient from each of these airfoils follow a linear pattern and can therefore be used to approximate the minimum drag coefficient, $\tilde{C}_{D_0}$, for the 64A204 as shown in Fig. 5.5.

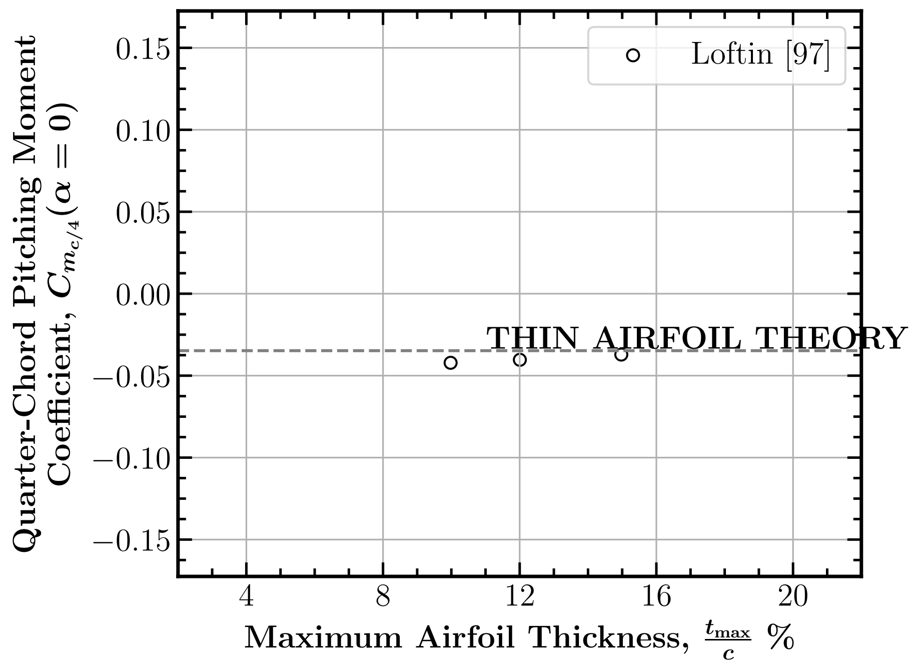


Fig. 5.4: A comparison of the section quarter-chord pitching moment of three NACA 6A-series airfoils to thin airfoil theory results.

The linear and quadratic terms of the section drag polar must be estimated using drag data from Loftin Jr. [98] and assumed to be linear in nature as well. Figure 5.6 shows this data and the accompanying estimates for the 64A204. The data presented to this point is sufficient to characterize a linear model of the NACA 64A204 airfoil for use in MachUpX. All that remains is to perform a similar analysis on the NACA 0005 and 0004 airfoils for the horizontal and vertical tails, respectively.

Since both the NACA 0005 and 0004 represent thin, symmetric airfoils, the analysis is simplified from that of the 64A204. First, symmetric airfoils generate zero lift at zero degrees angle of attack. Therefore, the zero-lift angle of attack is zero degrees. Since these airfoils are so thin, it can also be assumed that thin airfoil theory correctly predicts the section lift slope as $\tilde{C}_{L,\alpha} = 2\pi$. Symmetric airfoils also produce equivalent pressure distributions along their upper and lower surfaces; therefore, they produce zero pitching moment at all angles of attack and about all chord stations. Thus, $\tilde{C}_{m_{L_0}}$ and $\tilde{C}_{m,\alpha}$ can both be approximated to be zero.

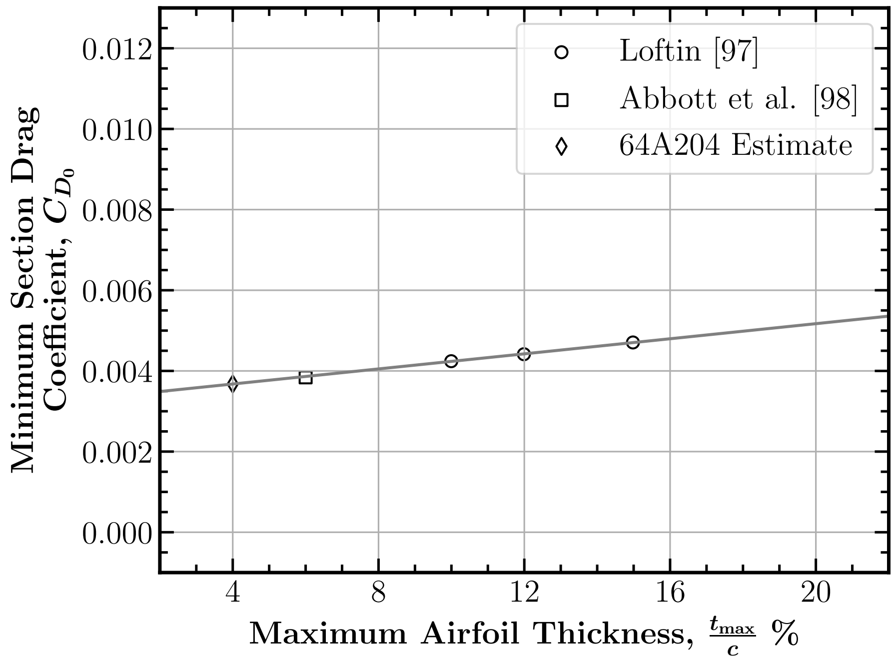


Fig. 5.5: Data representing the section minimum drag of three NACA 6A-series airfoils and one 6-series airfoil with an approximation for the NACA 64A204 airfoil.

All that remains in the characterization of the 0005 and 0004 airfoils is to approximate the section drag polar of each. Here, the same methodology that was used to characterize the drag polar for the 64A204 airfoil can be used. In their work, Abbott et al. [99] included wind tunnel data for the NACA 0006 and 0009 airfoils. Fitting a parabolic function to this data results in the drag coefficients shown in Fig. 5.7, which can then be used to estimate the section drag polar coefficients of the NACA 0004 and 0005 airfoils. The full characterizations of each airfoil covered in this analysis are summarized in Table 5.5.

5.2 Linear Aerodynamic Model

The coefficients in the linear aerodynamic model for the baseline aircraft, given in Eqs. (4.35)–(4.39), can be evaluated by considering the longitudinal and lateral forces and moments acting on the aircraft separately in terms of the nondimensional aerodynamic coefficients. Then, by following the methodology presented by Phillips, the coefficients can be calculated [8]. Using the baseline aerodynamic coefficients, and intuition regarding the

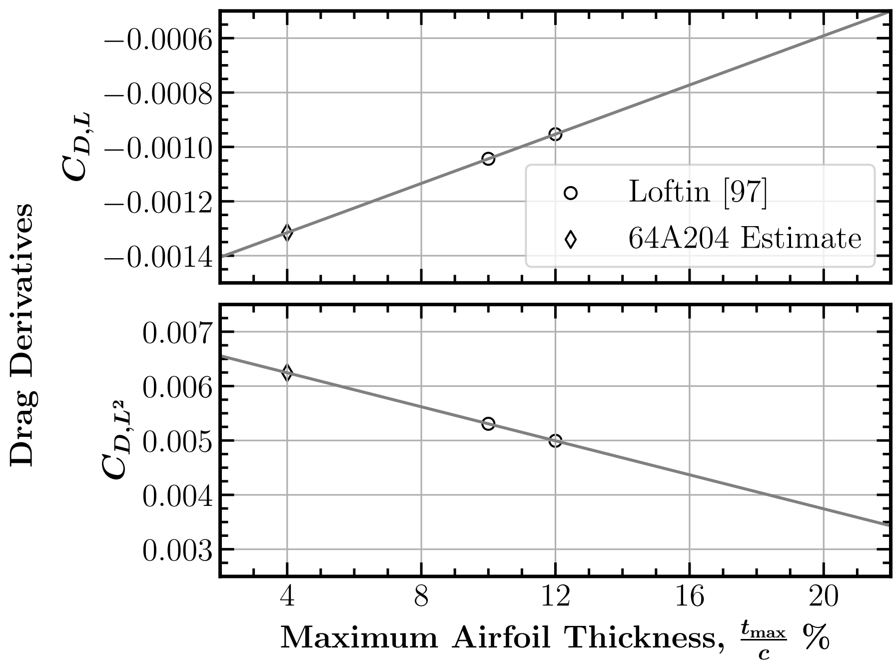


Fig. 5.6: Data representing the drag derivatives of three NACA 6A-series airfoils with an approximation for the NACA 64A204 airfoil.

Section Parameter	**64A204**	**0005**	**0004**
Zero-Lift Angle of Attack, α_{L_0} [rad]	-0.0222	0	0
Lift Slope, $\tilde{C}_{m,\alpha}$ [1/rad]	2π	2π	2π
Zero-Lift Pitching Moment, $\tilde{C}_{m_{L_0}}$	-0.0348	0	0
Pitching Moment Slope, $\tilde{C}_{m,\alpha}$	0	0	0
Minimum Drag Coefficient, $\tilde{C}_{D_0}$	0.0037	0.0045	0.0045
First Drag Coefficient Derivative, $\tilde{C}_{D,L}$	-0.0013	-0.0024	-0.0028
Second Drag Coefficient Derivative, $\tilde{C}_{D,L^2}$	0.0062	0.0076	0.0082

Table 5.5: Linear airfoil models for the NACA airfoils used in the baseline and BIRE aircraft.

effects of BIRE rotation angle, the coefficients in the linear aerodynamic model for the BIRE, given in Eqs. (4.44)-(4.49), can be estimated. This process is covered in Appendix A and includes results for the longitudinal forces and moments.

5.3 Non-Linear Aerodynamic Model

The coefficients in the non-linear aerodynamic model, given in Eqs. (4.60)–(4.65) for the baseline aircraft and Eqs. (4.66)–(4.71) for the BIRE, are evaluated using a numerical

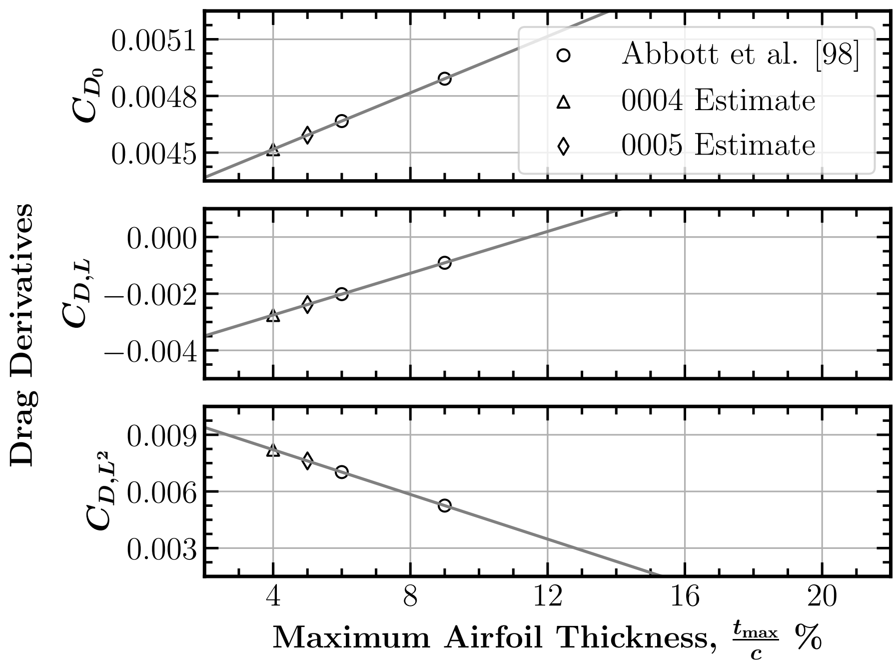


Fig. 5.7: Data representing the drag derivatives of two symmetric NACA 4-digit airfoils with an approximation of the NACA 0005 and 0004 derivatives.

lifting-line code developed at USU called MachUpX. MachUpX was introduced previously and further details about its development and use cases can be found in the work by Goates and Hunsaker [90].

To use any numerical simulation tool, it is essential to ensure that properly grid-resolved solutions are obtained; therefore, in this section a grid resolution study will be presented first. Then, the method for approximating the aerodynamic sensitivity coefficients in the non-linear model will be given along with the evaluated coefficients themselves. The availability of experimental data from Nguyen et al. [64] for the baseline aircraft gives a unique opportunity to compare the results of the aerodynamics generated by MachUpX to those generated using the NASA wind tunnel data. In terms of trimming the aircraft in the majority of the cases examined in this work, some of the coefficients are far more important than others. Therefore, a coefficient sensitivity study is presented here using the trimming techniques discussed in Chapter 6. Any terms that cause changes in trim results below a specified threshold are not adapted to better match the wind tunnel data.

As indicated in Chapter 4, the aerodynamic sensitivity coefficients for the BIRE are modeled as a function of BIRE rotation angle. The process of creating the periodic models is discussed here and the resulting fits are discussed in terms of physical aerodynamic intuition. The BIRE coefficients are shown in comparison to the sensitivity coefficients of the baseline aircraft, further providing context for the physics shown in the coefficient fits.

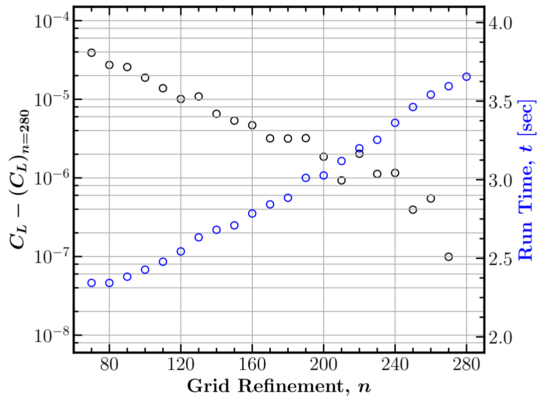
5.3.1 Grid Resolution Study

MachUpX is an implementation of numerical lifting-line theory, which places horseshoe vortices along a lifting surface to predict the aerodynamic forces and moments acting on the surface [89,90]. In this case, the "grid" that needs to be properly resolved is the number of horseshoe vortices used to represent a given wing segment.

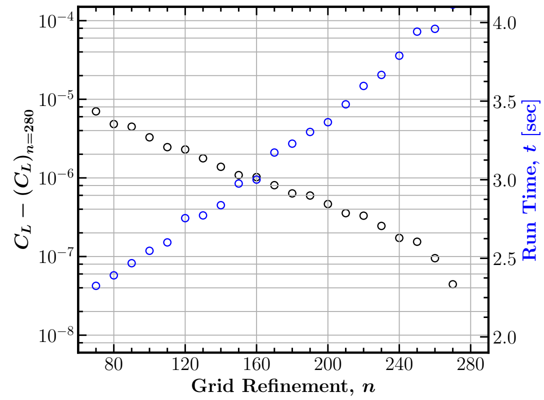
To do so, the baseline and BIRE aircraft were modeled in MachUpX according to the information given in Chapters 3 and 4. The input files[3] for MachUpX for the baseline aircraft and BIRE are given in Appendix B. Then, the aerodynamic coefficients were determined for the case of zero sideslip angle, rotation rates, and control surface deflections with the angle of attack at $\alpha = 5°$. In this configuration, Figures 5.8a and 5.8b were generated for the baseline aircraft and Figures 5.9a and 5.9b for the BIRE, where n is the number of horseshoe vortices on the surface. The convergence was measured by subtracting the lift coefficient at each grid resolution from that obtained with $n = 280$ horseshoe vortices.

These figures show that an appropriate level of convergence is obtained even with only 80 horseshoe vortices on each surface. A difference of $C_L - (C_L)_{n=280} = 10^{-4}$ is beyond the level of resolution that the aerodynamics can reasonably be assumed to be accurate in MachUpX, and therefore the main wing and horizontal tail of the baseline aircraft were modeled with $n = 80$ horseshoe vortices. To ensure that this level of convergence was maintained, even with the horizontal tail rotated in the BIRE, Fig. 5.9c was generated evaluating the convergence of the lift coefficient with the BIRE rotation angle at $\delta_B = 45°$. With the horizontal tail rotated, the convergence results are nearly identical; therefore the BIRE horizontal tail can be set to 80 horseshoe vortices.

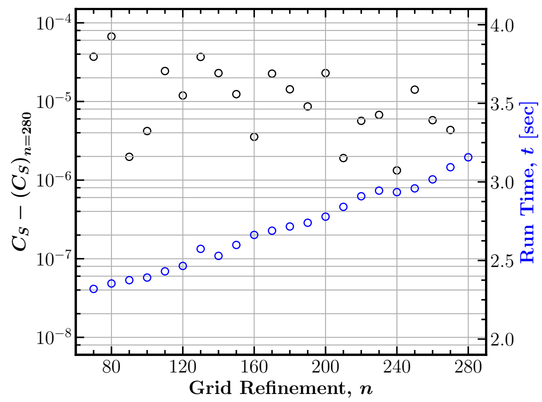
[3] `https://machupx.readthedocs.io/en/latest/creating_input_files.html`



(a) Main Wing



(b) Horizontal Tail



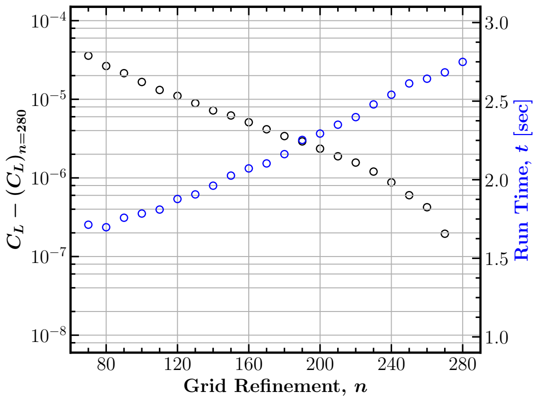
(c) Vertical Tail

Fig. 5.8: Grid convergence and run time of the component surfaces of the baseline aircraft.

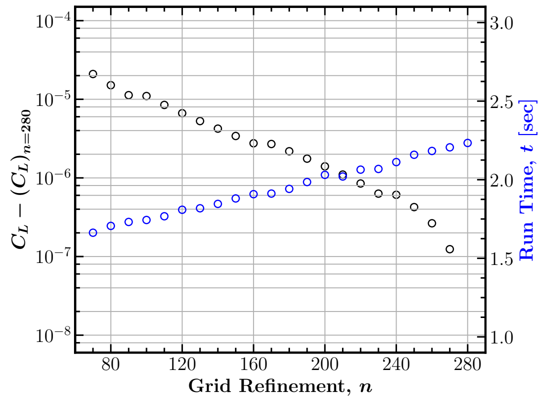
Finally, under the condition of zero angle of attack, rotation rates, and control surface deflections with a sideslip angle of $\beta = 5°$, a convergence study on the vertical tail of the baseline was conducted. The results of this study are shown in Fig. 5.8c. Again, the level of fidelity is sufficient at $n = 80$ with a slightly-less monotonic decrease in side force convergence than was seen in the lift.

5.3.2 Baseline Aircraft

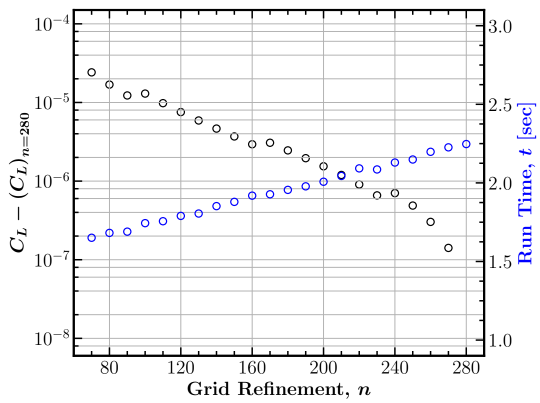
A model for the baseline aircraft of the form given in Eqs. (4.60)–(4.65) requires that we calculate the sensitivity coefficients for each force and moment coefficient in the model. In this work, finite differences and linear regression techniques were used to estimate these sensitivity coefficients. These techniques require aerodynamic data at many different flight



(a) Main Wing



(b) Horizontal Tail



(c) Horizontal Tail at $\delta_B = 45°$

Fig. 5.9: Grid convergence and run time of the component surfaces of the BIRE aircraft.

conditions. This aerodynamic data was evaluated using MachUpX and stored in a database that was then used to estimate sensitivities.

A database of aerodynamic coefficients was created using MachUpX for the baseline aircraft across a range of aerodynamic angles, rotation rates, and control surface deflections. The parameters of the database, the parameter limits, and the number of points included for each parameter in the database are shown in Table 5.6. Parameter limits and values were chosen to coincide with the wind tunnel data given by Nguyen et al. [64] whenever possible. This way, actual wind tunnel values could be used and linear assumptions would not need to be made between data points. The difference in the number of data points in the control surface deflection dimensions and body-fixed rate dimensions is primarily due to limitations in the data reported by Nguyen et al. [64].

Table 5.6: Limits for each degree of freedom in the baseline fighter aircraft database.

Parameter Name	**Description**	**Limits**	**Number of Points**, N
α	Angle of Attack	$\pm 10°$	5
β	Sideslip Angle	$\pm 6°$	7
δ_e	Stabilator Deflection Angle	$\pm 10°$	3
δ_a	Aileron Deflection Angle	$\pm 20°$	3
δ_r	Rudder Deflection Angle	$\pm 30°$	3
p	Body-Fixed Roll Rate	± 90 deg / s	3
q	Body-Fixed Pitch Rate	± 30 deg / s	3
r	Body-Fixed Yaw Rate	± 30 deg / s	3

In the case of the body-fixed rotation rates, changes in p, q, and r were not measured by Nguyen et al. in their wind tunnel studies; rather, they measured the change in the aerodynamic coefficients with the rotation rates at various angles of attack [64]. Thus, to estimate the limits of the body-fixed rotation rates, simulation data presented by Nguyen et al. was used wherein several common maneuvers were performed and the maximum rates presented [64]. For the purposes of efficiency, the total number of data points indicated by a strict combination of all possible cases was not used. Rather, the control surface deflections and body-fixed rates were selectively run through angles of attack and sideslip sweeps. Thus, the total number of cases for the baseline aircraft is calculated as

$$N_t = N_\alpha \left(1 + N_{\delta_e} + N_{\delta_a} + N_{\overline{p}} + N_{\overline{q}} + N_{\overline{r}}\right) + N_\beta \left(1 + N_{\delta_a} + N_{\delta_r} + N_{\overline{p}} + N_{\overline{r}}\right) \tag{5.5}$$

The database produced by MachUpX for the baseline aircraft is included in Table D.1 of Appendix D.

Using the aerodynamic coefficients estimated by MachUpX, the coefficients in Eqs. (4.60)–(4.65) can be approximated. When possible, a least-squares polynomial fit was used to estimate the coefficients. For example, across a range of angles of attack, C_{L_0} and $C_{L,\alpha}$ are the intercept and slope of the line given with α along the abscissa and the lift coefficient along the ordinate. The coefficients $C_{S,\beta}$, $C_{\ell,\beta}$, C_{m_0} and $C_{m,\alpha}$, and $C_{n,\beta}$ can likewise be used to approximate the intercepts and slopes of their respective relationships. The terms of the drag polar in lift, C_{D_0}, $C_{D,L}$, and C_{D,L^2}, and the terms of the drag polar in side force

$C_{D,S}$ and C_{D,S^2} were estimated using a least-squares quadratic fit of the drag as a function of lift and side force, respectively.

When the parameter ranges are more limited, i.e. for the deflection angles and rotation rates, coefficients dependent on those parameters were calculated using a centered difference derivative approximation method [100]. Many of the component coefficients in the aerodynamic model are also a slight function of the angle of attack or sideslip angle. These changes are likely due to changes in downwash and sidewash. To better account for these slight changes, the results of the centered difference approximations were averaged across the angles of attack and sideslip angles present in the database.

For example, the coefficient C_{L,δ_e} changes slightly as a function of angle of attack due to the effects of downwash. From the database in Table D.1, the values of C_L for all angles of attack and stabilator deflections ($5 \times 3 = 15$ data points in total) were taken. Then, a centered difference for each angle of attack was calculated as

$$C_{L,\delta_e}[\alpha] = \frac{\left(C_L|_{\delta_e=10^\circ}[\alpha] - C_L|_{\delta_e=-10^\circ}[\alpha]\right)}{2\left(10^\circ \times \pi/180\right)} \tag{5.6}$$

Taking the average of this set of 15 centered difference approximations gives a better approximation of the value of C_{L,δ_e} that takes into account changes in downwash.

Coefficients in the nonlinear model that varied with lift; for example, the terms $\big(C_{S,L\overline{p}}C_{L_1}$ $+C_{S,\overline{p}}\big)$, required a combination of the centered difference scheme and a least-squares linear fit. After performing the centered difference approximation across the range of angles of attack or sideslip angle, a linear least-squares polynomial fit was performed on the resulting derivatives. In our example above, the centered difference approximation produces $C_{S,\overline{p}}$ as a function of the angle of attack. The linear polynomial fit in terms of C_{L_1} then produces the coefficient $C_{S,\overline{p}}$, the y-intercept, and $C_{S,L\overline{p}}$, the slope.

Performing these steps using both the MachUpX data and the wind tunnel data from Nguyen et al. [64] gives the coefficients reported in Tables 5.7 and 5.8. These tables document the coefficient as found in Eqs. (4.60)–(4.65), the value evaluated from MachUpX data, the value evaluated using the NASA wind tunnel data, and the percent error normalized by the NASA result. Note that there are substantial differences between certain coefficients predicted by MachUpX when compared to the wind tunnel data. These differences can broadly be separated into three categories: errors in geometry modeling, errors caused by the presence of important non-linear effects, and differences in drag modeling.

Table 5.7: A comparison of the aerodynamic force coefficients predicted by MachUpX and the wind tunnel data in the non-linear aerodynamic model.

Coefficient	**MachUpX**	**Wind Tunnel**	**% Error**	**Notes**
C_{L_0}	0.0456	0.0935	-51.2%	AM/LEV
$C_{L,\alpha}$	3.5791	3.8434	-6.9%	–
$C_{L,\overline{q}}$	3.3916	28.9082	-88.3%	LEV
C_{L,δ_e}	0.7474	0.5652	32.2%	CSM/LEV
$C_{S,\beta}$	-0.7224	-1.0793	-33.1%	FE/VF
$C_{S,\overline{p}}$	-0.0153	-0.0307	-50.1%	LEV
$C_{S,L\overline{p}}$	0.3318	0.2061	61.0%	VF
$C_{S,\overline{r}}$	0.4357	0.8275	-47.3%	LEV
C_{S,δ_a}	0.1104	0.0656	68.3%	CSM
C_{S,δ_r}	0.1992	0.1698	17.3%	CSM
C_{D_0}	0.0064	0.0218	-70.8%	DM
$C_{D,L}$	-0.0036	-0.034	-89.4%	LEV
C_{D,L^2}	0.112	0.1834	-38.9%	LEV
C_{D,S^2}	0.4963	0.7199	-31.1%	FE/VF
$C_{D,S\overline{p}}$	0.0768	-0.1663	-146.2%	LEV
$C_{D,\overline{q}}$	0.0368	-1.0947	-103.4%	–
$C_{D,L\overline{q}}$	0.775	4.6249	-83.2%	LEV
$C_{D,L^2\overline{q}}$	-0.1844	6.0809	-103.0%	LEV
$C_{D,S\overline{r}}$	-0.7239	0.7591	-195.4%	LEV
C_{D,δ_e}	-0.0032	-0.0093	-65.5%	–
$C_{D,L\delta_e}$	0.1775	0.1557	14.0%	–
C_{D,δ_e^2}	0.2854	0.4418	-35.4%	LEV
$C_{D,S\delta_a}$	0.1118	0.0675	65.8%	CSM
$C_{D,S\delta_r}$	0.18	0.1603	12.3%	–

AM - Airfoil Modeling
DM - Drag Modeling
CSM - Control Surface Modeling
FE - Fuselage Effects
LEV - Leading-Edge Vortices
VF - Ventral Fin Effects

Table 5.8: A comparison of the aerodynamic moment coefficients predicted by MachUpX and the wind tunnel data in the non-linear aerodynamic model.

Coefficient	**MachUpX**	**Wind Tunnel**	**% Error**	**Note**
$C_{\ell,\beta}$	-0.0685	-0.0888	-22.8%	LEV/VF
$C_{\ell,\overline{p}}$	-0.3182	-0.3349	-5.0%	–
$C_{\ell,\overline{r}}$	0.0469	0.0312	50.1%	VF
$C_{\ell,L\overline{r}}$	0.1067	0.217	-50.8%	VF
C_{ℓ,δ_a}	-0.0741	-0.1457	-49.1%	AM/LEV
C_{ℓ,δ_r}	0.0257	0.028	-8.2%	–
C_{m_0}	0.0099	-0.0097	-202.4%	AM/CGL
$C_{m,\alpha}$	-0.1099	0.1766	-162.2%	CGL
$C_{m,\overline{q}}$	-4.8503	-4.2425	14.3%	CGL
C_{m,δ_e}	-0.8795	-0.5881	49.5%	CGL/CSM
$C_{n,\beta}$	0.2752	0.2099	31.1%	LEV/VF
$C_{n,\overline{p}}$	0.0131	0.0345	-62.1%	LEV
$C_{n,L\overline{p}}$	-0.1607	-0.0402	299.3%	LEV/VF
$C_{n,\overline{r}}$	-0.1787	-0.3565	-49.9%	LEV/VF
C_{n,δ_a}	-0.0398	-0.0276	44.2%	CSM
$C_{n,L\delta_a}$	-0.0177	0.0077	-329.7%	CSM
C_{n,δ_r}	-0.0899	-0.0877	2.5%	–

AM - Airfoil Modeling
CGL - Center of Gravity Location
CSM - Control Surface Modeling
FE - Fuselage Effects
LEV - Leading-Edge Vortices
VF - Ventral Fin Effects

There are several potential sources of error from differences in geometry between the NASA model and MachUpX. One important geometric difference is that the NASA test vehicle included ventral fins, which were not modeled in MachUpX. Modeling the ventral fins in MachUpX poses many potential problems, since lifting surfaces modeled in its numerical lifting-line algorithm return the section lift coefficient to zero at the wing tips unless specified as one piece-wise continuous lifting surface. The ventral fins are located on the undercarriage of the baseline aircraft and forcing the lift distribution to zero at their root would introduce physical inconsistencies. Likewise, extending them to the centerline of the aircraft would likely cause more errors in estimating the aerodynamic coefficients.

In addition to modeling constraints, this work attempts to identify whether the BIRE can provide the appropriate stability and control by itself. Ventral fins are generally used to provide improved yaw stability across a variety of flight conditions and especially at large sideslip angles [101]. Therefore, removing the ventral fins from the model allows for the BIRE to be tested independent of the additional yaw stability offered by the fins.

The coefficients in Tables 5.7 and 5.8 that are primarily impacted by the lack of ventral fins in the MachUpX model are those associated with lateral model parameters: specifically, β, $\overline{p}$, and $\overline{r}$. In addition, since the ventral fins are unimpeded by the blanketing effects of increased angle of attack [101], they are more important to lateral terms that change with lift coefficient. The effects on the lateral coefficients from the ventral fins are very similar to the lateral effects of the fuselage. Although some effort has been made to estimate fuselage effects by extending the main wing, horizontal tail, and vertical tail to the fuselage centerline, there is a margin of error to be expected by these estimates. Thus, only a portion of the errors affecting the coefficients due to fuselage and ventral fin effects need to be accounted for in any changes to the model.

As mentioned in Chapter 4, the geometric characteristics of the control surfaces were among those that were estimated using drawings provided by Fox and Forrest [62]. The accuracy of these drawings is unclear and therefore additional geometric modeling errors can be introduced from estimations of the spanwise and chordwise fractions of the control surfaces. From these uncertainties, many of the differences in control surface sensitivities can reasonably be attributed to modeling differences between the NASA wind tunnel and MachUpX models. Potential inaccuracies from interpreting the drawings also extend to the relative location of the center of gravity to the lifting surfaces on the tail. Sensitivities in the pitching moment are likely the most impacted by discrepancies in the location of lifting surfaces with respect to the center of gravity. The above notes on ventral fins, control surface sizing, center of gravity location, and fuselage effects can reasonably be expected to produce errors in estimating aerodynamic coefficients.

In terms of non-linear physical effects, several have already been discussed. Lifting-line theory does not model spanwise changes in the circulation of a lifting surface [102]; therefore, using numerical lifting-line to estimate aerodynamic coefficients on the baseline aircraft and BIRE inevitably will introduce some errors in coefficients highly-sensitive to spanwise flow. Additionally, MachUpX is not equipped to handle the any effects from the generation of leading-edge vortices, which are common in highly-swept wings. Many of the coefficients in Tables 5.7 and 5.8, both longitudinal and lateral, could easily be effected by the leading-edge vortices shed from the main wing and vertical tail in particular. Finally, the effects of flow separation are not modeled in numerical lifting-line, which can the forces and moments produced by the sharp, transonic airfoils and highly-swept wings at even moderate angles of attack. This could be the cause of the errors noted with the elevator sensitivity coefficients, since the sharp leading-edges of the biconvex airfoils on the wind tunnel model would produce rather large separation bubbles at low speeds. These effects likely characterize most of the errors due to non-linear physics measured between the MachUpX- and NASA-produced aerodynamic coefficients.

The final category of errors which can reasonably be attributed to differences in aerodynamic coefficients is the modeling of the drag coefficient. MachUpX estimates the effects of induced and parasitic drag acting on an aircraft in flight using the airfoil drag polar. The effects of drag from flow separation, interference at wing-body junctures, and other effects are ignored. These drag effects may be responsible for some of the errors in the aerodynamic drag coefficient components in Table 5.7.

It is reasonable to assume that a low-fidelity aerodynamic tool such as MachUpX will vary by up to 20-30% from wind tunnel results with aircraft that are well-modeled using numerical lifting-line. That is, where the effects of spanwise flow are small ($R_A > 4$), where the effects of separation are minimal (gradual changes in aircraft geometry), and where sweep angles are small. In this case, we can expect, then, that the violation of some of these constraints further increases the susceptibility of these coefficients to error. The purpose of this research is to provide a preliminary look into the trim and control characteristics of the BIRE aircraft. Therefore, it is sufficient that the trends of these coefficients be accurately represented, rather than demanding accuracy in the values of the coefficients themselves. Further, if the reported differences between the MachUpX model and the NASA wind tunnel data do not substantively affect the results of trimming the aircraft, those differences can be ignored when analyzing trim.

Coefficient Sensitivity Study

To provide a closer examination of the relative importance in the differences given in Tables 5.7 and 5.8, a sensitivity study can be performed on each of the coefficients. This study indicates how sensitive the results of trimming the aircraft in various conditions are to changes in the aerodynamic model coefficients. Thus, the information in this section requires the trim algorithm developed in Chapter 6 to fully explore. In the interest of maintaining continuity, a discussion on the nature of the trim algorithm will be left until Chapter 6 and its results will be used and referred to here without explanation.

The sensitivity study was conducted by trimming the aircraft in steady-heading sideslip (Chapter 6 Section 6.2.4) and a steady, coordinated turn (Chapter 6 Section 6.2.3) using the coefficients reported using MachUpX aerodynamic data. After trimming the aircraft, the aerodynamic angles, body-fixed rotation rates, and control surface deflections required to trim the aircraft in both trim states were considered. The coefficients were then changed to represent those predicted using the NASA wind tunnel data and the required aerodynamic angles, body-fixed rotation rates, and control surface deflections to trim were recorded. Maximum sensitivities for each coefficient were then reported by taking the maximum change in trim parameter between the case of steady-heading sideslip and steady, coordinated turn.

Both the steady-heading sideslip and steady coordinated turn trim conditions were performed with a climb angle of $\gamma = 0°$ and a bank angle of $\phi = 5°$. In each case, the aircraft was trimmed in a low-altitude, low-velocity condition (H = 1,000 ft and V = 222.51 ft/s) to maximize the deflections that would be required to trim. The differences in the aerodynamic parameters required for trim in steady, level flight are shown in Table 5.9 for the aerodynamic force coefficients and in Table 5.10 for the aerodynamic moment coefficients. From this trim sensitivity analysis, many coefficients predicted by MachUpX that differed substantially from those predicted using the NASA data have relatively little effect on the trim state of the aircraft. For example, $C_{L,\bar{q}}$ differs by about 88% between the MachUpX prediction and the NASA wind tunnel data. However, when the baseline aircraft is trimmed in a steady, coordinated turn, changing the model to reflect the NASA-derived coefficient impacts the trim state by less than a degree across all of the states. Therefore, we can represent the coefficients in the nonlinear model with minimum loss of fidelity in our trim calculations by keeping the MachUpX value in our model.

Coefficient	**Error**	$\Delta\alpha$, (deg)	$\Delta\beta$, (deg)	$\Delta\delta_a$, (deg)	$\Delta\delta_e$, (deg)	$\Delta\delta_r$, (deg)	Δp, (deg/s)	Δq, (deg/s)	Δr, (deg/s)
C_{L_0}	-51.2%	0.7576	0.1032	0.0115	0.0947	0.3078	0.0078	0.0005	0.0055
$C_{L,\alpha}$	-6.9%	1.2216	0.1632	0.0182	0.1527	0.487	0.0126	0.0008	0.0088
$C_{L,\overline{q}}$	-88.3%	0.0094	0	0	0.0012	0	0.0001	0	0.0001
C_{L,δ_e}	32.2%	0.0743	0.2085	0.0242	0.0093	0.6247	0.0008	0	0.0005
$C_{S,\beta}$	-33.1%	0.0619	15.8883	1.787	0.0077	47.445	0	0	0
$C_{S,\overline{p}}$	-50.1%	0	0.0008	0.0001	0	0.0025	0	0	0
$C_{S,L\overline{p}}$	61.0%	0	0.008	0.0009	0	0.0237	0	0	0
$C_{S,\overline{r}}$	-47.3%	0	0.0649	0.0073	0	0.1938	0.0001	0	0
C_{S,δ_a}	68.3%	0.0034	0.7384	0.083	0.0004	2.2048	0	0	0
C_{S,δ_r}	17.3%	0.0297	7.9564	0.8948	0.0037	23.7588	0	0	0
C_{D_0}	-70.8%	0.0874	2.0131	0.2257	0.0109	6.0093	0.0007	0	0.0005
$C_{D,L}$	-89.4%	0.2898	11.9681	1.2916	0.0361	35.5994	0.0017	0.0001	0.0013
C_{D,L^2}	-38.9%	0.475	7.9392	0.8898	0.0594	23.6988	0.0044	0.0003	0.0032
C_{D,S^2}	-31.1%	0.1038	2.3507	0.2635	0.013	7.0171	0	0	0
$C_{D,S\overline{p}}$	-146.2%	0	0	0	0	0	0	0	0
$C_{D,\overline{q}}$	-103.4%	0.0001	0	0	0	0	0	0	0
$C_{D,L\overline{q}}$	-83.2%	0.0005	0	0	0.0001	0	0	0	0
$C_{D,L^2\overline{q}}$	-103.0%	0.001	0	0	0.0001	0	0	0	0
$C_{D,S\overline{r}}$	-195.4%	0	0	0	0	0	0	0	0
C_{D,δ_e}	-65.5%	0.001	0.0254	0.0028	0.0001	0.0758	0	0	0
$C_{D,L\delta_e}$	14.0%	0.0042	0.1054	0.0118	0.0005	0.3147	0	0	0
C_{D,δ_e^2}	-35.4%	0.0007	0.0182	0.002	0.0001	0.0542	0	0	0
$C_{D,S\delta_a}$	65.8%	0.004	0.102	0.0114	0.0005	0.3045	0	0	0
$C_{D,S\delta_r}$	12.3%	0.0432	1.0427	0.1169	0.0054	3.1125	0	0	0

Table 5.9: Trim sensitivity analysis of the baseline aircraft force coefficients.

Coefficient	**Error**	$\Delta\alpha$, (deg)	$\Delta\beta$, (deg)	$\Delta\delta_a$, (deg)	$\Delta\delta_e$, (deg)	$\Delta\delta_r$, (deg)	Δp, (deg/s)	Δq, (deg/s)	Δr, (deg/s)
$C_{\ell,\beta}$	-22.8%	0.0059	0.9	5.6691	0.0007	6.5706	0	0	0
$C_{\ell,\overline{p}}$	-5.0%	0	0.0002	0.0026	0	0.0025	0	0	0
$C_{\ell,\overline{r}}$	50.1%	0	0.0007	0.0074	0	0.0071	0	0	0
$C_{\ell,L\overline{r}}$	-50.8%	0	0.0056	0.0616	0	0.0585	0	0	0
C_{ℓ,δ_a}	-49.1%	0.0012	0.191	1.2279	0.0002	1.4112	0	0	0
C_{ℓ,δ_r}	-8.2%	0.0018	0.278	1.81	0.0002	2.0691	0	0	0
C_{m_0}	-202.4%	0.263	0.0695	0.0104	1.3121	0.2148	0.0027	0.0002	0.0019
$C_{m,\alpha}$	-162.2%	1.1414	0.0932	0.006	5.6453	0.2463	0.0118	0.0007	0.0082
$C_{m,\overline{q}}$	14.3%	0.0002	0	0	0.001	0	0	0	0
C_{m,δ_e}	49.5%	0.1661	0.0396	0.0061	0.8314	0.1228	0.0017	0.0001	0.0012
$C_{n,\beta}$	31.1%	0.0655	10.6772	4.1514	0.0082	40.3833	0	0	0
$C_{n,\overline{p}}$	-62.1%	0	0.0025	0.0012	0	0.0102	0	0	0
$C_{n,L\overline{p}}$	299.3%	0	0.0163	0.0082	0	0.0669	0	0	0
$C_{n,\overline{r}}$	-49.9%	0	0.0631	0.0316	0	0.259	0.0001	0	0
C_{n,δ_a}	44.2%	0.0048	0.5489	0.1764	0.0006	1.9695	0	0	0
$C_{n,L\delta_a}$	-329.7%	0.0134	1.5159	0.4773	0.0017	5.4105	0	0	0
C_{n,δ_r}	2.5%	0.0281	3.0836	0.9371	0.0035	10.9086	0	0	0

Table 5.10: Trim sensitivity analysis of the baseline aircraft moment coefficients.

On the other hand, note that only a 2.5% difference in the value for C_{n,δ_r} produces a 10 degree difference in rudder angle required to trim. This difference is likely driven by trimming in steady-heading sideslip, which requires substantial rudder deflection [103]. To better represent the aerodynamics of the baseline aircraft, and therefore provide a good comparison with the BIRE aircraft, these coefficients can be adjusted to match the values given by the wind tunnel results when appropriate. The appropriateness of any adjustment to the aerodynamic coefficients should be motivated by identifying a reasonable source of error produced by either modeling or limitations in the physics accounted for in MachUpX.

After identifying the coefficients in Tables 5.9 and 5.10 that impacted the trim state by more than 1°, each coefficient was adjusted to account for the errors that were just enumerated. Since these adjustments to the coefficients are applied to the BIRE coefficients as well, the effects of the ventral fins were ignored when making adjustments. For example, the coefficient $C_{S,\beta}$ differs between MachUpX and NASA data by 33%, likely due to fuselage effects and the lack of ventral fins in the MachUpX model. However, since the ventral fins are not modeled in either the baseline aircraft or BIRE in MachUpX, the effect that they have on $C_{S,\beta}$ should be ignored when applying an adjustment to the coefficient. Therefore, it was determined that only 50% of the adjustment should be made to include the effects of the fuselage without also including the effects of the ventral fins.

Table 5.11 shows the adjustments made to the components of the aerodynamic force coefficients in the non-linear model for the baseline aircraft. The adjustments are labeled Δ_C so that each coefficient, C_i, in the non-linear aerodynamic model for the baseline aircraft is defined as

$$(C_i)_{\text{adj}} = (C_i)_{\text{MUX}} + \Delta_{C_i} \tag{5.7}$$

Thus, the error between the NASA-produced coefficients and the adjusted MachUpX coefficients is

$$\varepsilon_{\text{adj}} = 100 \times \frac{(C_i)_{\text{adj}} - (C_i)_{\text{NASA}}}{(C_i)_{\text{NASA}}} \% \tag{5.8}$$

The adjusted error predicted by Eq. (5.8) is included in Table 5.11 as well as the adjustments

Δ_C. Table 5.12 shows the adjusted moment coefficients for the baseline aircraft along with the adjusted error and coefficient adjustments, just as in Table 5.11.

Table 5.11: Adjustments made to the aerodynamic force component coefficients in the baseline non-linear model.

Coefficient	**MachUpX**	**MachUpX Adjusted**	**Adjusted Error**	Δ_C
C_{L0}	0.0456	0.0456	-51.2%	–
$C_{L,\alpha}$	3.5791	3.5791	-6.9%	–
$C_{L,\bar{q}}$	3.3916	3.3916	-88.3%	–
C_{L,δ_e}	0.7474	0.5652	0%	-0.1822
$C_{S,\beta}$	-0.7224	-0.9008	17%	-0.1785
$C_{S,\bar{p}}$	-0.0153	-0.0153	-50.1%	–
$C_{S,L\bar{p}}$	0.3318	0.3318	61.0%	–
$C_{S,\bar{r}}$	0.4357	0.4357	-47.3%	–
C_{S,δ_a}	0.1104	0.0656	0%	-0.0448
C_{S,δ_r}	0.1992	0.1698	0%	-0.0294
C_{D_0}	0.0064	0.0218	0%	0.0154
$C_{D,L}$	-0.0036	-0.0036	0%	-0.0304
C_{D,L^2}	0.112	0.1834	0%	0.0714
C_{D,S^2}	0.4963	0.6081	16%	0.1118
$C_{D,S\bar{p}}$	0.0768	0.0768	-146.2%	–
$C_{D,\bar{q}}$	0.0368	0.0368	-103.4%	–
$C_{D,L\bar{q}}$	0.775	0.775	-83.2%	–
$C_{D,L^2\bar{q}}$	-0.1844	-0.1844	-103.0%	–
$C_{D,S\bar{r}}$	-0.7239	-0.7239	-195.4%	–
C_{D,δ_e}	-0.0032	-0.0032	-65.5%	–
$C_{D,L\delta_e}$	0.1775	0.1775	14.0%	–
C_{D,δ_e^2}	0.2854	0.2854	-35.4%	–
$C_{D,S\delta_a}$	0.1118	0.1118	65.8%	–
$C_{D,S\delta_r}$	0.18	0.1604	0%	-0.0196

5.3.3 BIRE Aircraft Coefficients

The non-linear aerodynamic model for the BIRE aircraft follows the form given in Eqs. (4.66)–(4.71). Again, a database was created using MachUpX for the BIRE aircraft according to the dimensions and numbers of points given in Table 5.13. A truncated version of this database is included in Table D.2 of Appendix D. Note that the number of BIRE rotation angles is much larger than the other dimensions of the database to ensure that the

Table 5.12: Adjustments made to the aerodynamic moment component coefficients in the baseline non-linear model.

Coefficient	MachUpX	MachUpX Adjusted	Adjusted Error	Δ_C
$C_{\ell,\beta}$	-0.0685	-0.0787	11%	-0.0101
$C_{\ell,\bar{p}}$	-0.3182	-0.3182	-5.0%	–
$C_{\ell,\bar{r}}$	0.0469	0.0469	50.1%	–
$C_{\ell,L\bar{r}}$	0.1067	0.1067	-50.8%	–
C_{ℓ,δ_a}	-0.0741	-0.0741	-49.1%	–
C_{ℓ,δ_r}	0.0257	0.0257	-8.2%	–
C_{m_0}	0.0099	-0.0077	0%	-0.0196
$C_{m,\alpha}$	-0.1099	0.1375	0%	0.2865
$C_{m,\bar{q}}$	-4.8503	-4.8503	14.3%	–
C_{m,δ_e}	-0.8795	-0.5881	0%	0.2914
$C_{n,\beta}$	0.2752	0.2426	16%	-0.0326
$C_{n,\bar{p}}$	0.0131	0.0131	-62.1%	–
$C_{n,L\bar{p}}$	-0.1607	-0.1005	150%	0.0602
$C_{n,\bar{r}}$	-0.1787	-0.1787	-49.9%	–
C_{n,δ_a}	-0.0398	-0.0276	0%	0.0122
$C_{n,L\delta_a}$	-0.0177	0.0077	0%	0.0254
C_{n,δ_r}	-0.0899	-0.0877	2.5%	–

change in linear coefficients with BIRE rotation angle are correctly identified. Also, the limits of BIRE rotation angle are from $\delta_B = -180°$ to $\delta_B = 180°$. It is expected that for the majority of trim conditions, $|\delta_B| \leq 90°$; however, in some cases it may be necessary for the force exerted on the tail to switch rapidly. Allowing the BIRE to rotate beyond 90° will be useful in these instances.

Table 5.13: Limits for each degree of freedom in the baseline fighter aircraft database.

Parameter Name	Description	Limits	Number of Points
α	Angle of Attack	$\pm 10°$	5
β	Sideslip Angle	$\pm 6°$	7
δ_e	Stabilator Deflection Angle	$\pm 10°$	3
δ_a	Aileron Deflection Angle	$\pm 20°$	3
δ_B	BIRE Rotation Angle	$\pm 180°$	73
p	Body-Fixed Roll Rate	± 90 deg / s	3
q	Body-Fixed Pitch Rate	± 30 deg / s	3
r	Body-Fixed Yaw Rate	± 30 deg / s	3

Calculating the coefficients of the nonlinear aerodynamic model for the BIRE proceeds in much the same way as the baseline model. Two key differences include the presence of additional coefficients that were not in the baseline model and that each of these fits were performed at all BIRE rotation angles. The additional coefficients in the BIRE model are fit using least-square linear and quadratic fits as well as the centered-differences described previously. That is, a term such as $C_{L,\beta}$ is estimated by fitting a line through the lift coefficients produced at various sideslip angles and a given BIRE rotation angle. Terms like $C_{L,\overline{p}}$ that were not calculated in the baseline model are estimated using the average of centered-differences. Finally, the only new term requiring a quadratic fit is the drag sensitivity coefficients with respect to side force, $C_{D,S}$ and C_{D,S^2}.

From a fundamental aerodynamic understanding, the BIRE control system presents a trade-off between longitudinal and lateral control as the horizontal tail rotates. Therefore, longitudinal coefficients will suffer a reduction is effectiveness with BIRE rotation when rotated from $\delta_B = 0°$ to $\delta_B = 90°$. Conversely, lateral coefficients will become more effective as the tail is rotated across the same range. It can be intuitively assumed that most of these coefficients will follow a periodic pattern and can therefore be modeled as

$$\hat{C}_i = A_i \sin\left(\omega_i \delta_B + \varphi_i\right) + \zeta_i + \Delta_{C_i} \tag{5.9}$$

following the hat notation from Chapter 4 and with i representing the model sensitivity identifiers; that is, for example, $i = L{,}\alpha$ for the sensitivity of lift to angle of attack.

After collecting the aerodynamic force and moment coefficients in the database, each was examined through a coefficient sensitivity study to determine whether modeling the periodic behavior of the sensitivity coefficient made a significant impact on the total aerodynamic force or moment to which it contributed. To make this determination, one percent of the average magnitude of each aerodynamic coefficient in the database was calculated. That is, the sensitivity parameter for each aerodynamic force and moment coefficient, ς_{C_i}, was calculated as

$$\sigma_{C_i} = 0.01|\overline{C}_i| = 0.01\frac{1}{N}\sum_{i=1}^{N}|C_i| \tag{5.10}$$

where N is the total number of cases in the database and is calculated

$$N_t = N_{\delta_B} N_{t,\text{base}} \tag{5.11}$$

where $N_{t,\text{base}}$ is determined using Eq. (5.5).

The sensitivity parameters can then be compared to the maximum contribution of each sensitivity coefficient to the total aerodynamic force and moment. If the maximum contribution of a particular sensitivity coefficient is less than or equal to the sensitivity parameter ς_{C_i}, then that coefficient was left constant as a function of δ_B. The maximum contribution of each sensitivity coefficient was calculated as the difference between the maximum and minimum values of Eq. (5.9) ($2A_i$) multiplied by the maximum of the sensitivity parameter in the database. For example, the maximum contribution of the sensitivity coefficient $C_{S,L\overline{p}}$ is

$$\Omega_{S,L\overline{p}} = |2A_{S,L\overline{p}}\,(C_{L_1})_{\max}\,(\overline{p})_{\max}\,| \tag{5.12}$$

with $(C_{L_1})_{\max}$ calculated from the lift coefficients in the database where all parameters except the angle of attack are zero and $(\overline{p})_{\max}$ taken from Table 5.13. Table 5.14 shows each coefficient in the BIRE aerodynamic model along with the sensitivity parameter, maximum contribution of the coefficient, and whether it was modeled as a periodic function.

In light of the sensitivity analysis performed above, the coefficients at each BIRE rotation were fit to a function of the form given in Eq. (4.43). The amplitude of the sine way, A, its frequency, ω, the phase shift φ, the coefficient shift ζ, and the coefficient correction Δ_C are each given for the aerodynamic forces in Table 5.15 and for the aerodynamic moments in Table 5.16. The adjustments indicated in Tables 5.11 and 5.12 were added to the BIRE coefficients to make the comparison between the two aircraft reasonable given the additional physics that the adjustments represent.

Table 5.14: Sensitivity to tail rotation angle study for the aerodynamic coefficients of the BIRE aircraft.

Coefficient	σ_C	Ω_C	**Periodic**	**Coefficient**	σ_C	Ω_C	**Periodic**
$\hat{C}_{L_0}$		0.0289	Yes	$\hat{C}_{\ell_0}$		0.0004	Yes
$\hat{C}_{L,\alpha}$		0.0762	Yes	$\hat{C}_{\ell,\alpha}$		0.0016	Yes
$\hat{C}_{L,\beta}$		0.2519	Yes	$\hat{C}_{\ell,\beta}$		0.0006	Yes
$\hat{C}_{L,\overline{p}}$	0.0027	0.0021	No	$\hat{C}_{\ell,\overline{p}}$	0.0002	0.0008	Yes
$\hat{C}_{L,\overline{q}}$		0.0540	Yes	$\hat{C}_{\ell,\overline{q}}$		0.0001	No
$\hat{C}_{L,\overline{r}}$		0.0480	Yes	$\hat{C}_{\ell,\overline{r}}$		0.0001	No
$\hat{C}_{L,\delta_a}$		0.0004	No	$\hat{C}_{\ell,L\overline{r}}$		0.0002	No
$\hat{C}_{L,\delta_e}$		0.6672	Yes	$\hat{C}_{\ell,\delta_a}$		0.0105	Yes
$\hat{C}_{S_0}$		0.2120	Yes	$\hat{C}_{\ell,\delta_e}$		0.0011	Yes
$\hat{C}_{S,\alpha}$		0.1280	Yes	$\hat{C}_{m_0}$		0.0385	Yes
$\hat{C}_{S,\beta}$		0.2376	Yes	$\hat{C}_{m,\alpha}$		0.0867	Yes
$\hat{C}_{S,\overline{p}}$		0.0004	No	$\hat{C}_{m,\beta}$		0.2897	Yes
$\hat{C}_{S,L\overline{p}}$	0.0004	0.0030	Yes	$\hat{C}_{m,\overline{p}}$	0.0005	0.0022	Yes
$\hat{C}_{S,\overline{q}}$		0.0530	Yes	$\hat{C}_{m,\overline{q}}$		0.0627	Yes
$\hat{C}_{S,\overline{r}}$		0.0433	Yes	$\hat{C}_{m,\overline{r}}$		0.0541	Yes
$\hat{C}_{S,\delta_a}$		0.0011	Yes	$\hat{C}_{m,\delta_a}$		0.0006	Yes
$\hat{C}_{S,\delta_e}$		0.6416	Yes	$\hat{C}_{m,\delta_e}$		0.7954	Yes
$\hat{C}_{D_0}$		0.0002	No	$\hat{C}_{n_0}$		0.0096	Yes
$\hat{C}_{D,L}$		0.0004	No	$\hat{C}_{n,\alpha}$		0.0649	Yes
$\hat{C}_{D,L^2}$		0.0052	Yes	$\hat{C}_{n,\beta}$		0.1109	Yes
$\hat{C}_{D,S}$		0.0081	Yes	$\hat{C}_{n,\overline{p}}$		0.0001	No
$\hat{C}_{D,S^2}$		0.0155	Yes	$\hat{C}_{n,L\overline{p}}$	0.0002	0.0012	Yes
$\hat{C}_{D,\overline{p}}$		0.0001	No	$\hat{C}_{n,\overline{q}}$		0.0245	Yes
$\hat{C}_{D,S\overline{p}}$		0.0001	No	$\hat{C}_{n,\overline{r}}$		0.0204	Yes
$\hat{C}_{D,\overline{q}}$	0.0004	0.0004	No	$\hat{C}_{n,\delta_a}$		0.0002	No
$\hat{C}_{D,L\overline{q}}$		0.0077	Yes	$\hat{C}_{n,L\delta_a}$		0.0094	Yes
$\hat{C}_{D,L^2\overline{q}}$		0.0002	No	$\hat{C}_{n,\delta_e}$		0.3078	Yes
$\hat{C}_{D,\overline{r}}$		0.0001	No				
$\hat{C}_{D,S\overline{r}}$		0.0003	No				
$\hat{C}_{D,\delta_a}$		0.0059	Yes				
$\hat{C}_{D,S\delta_a}$		0.0059	Yes				
$\hat{C}_{D,\delta_e}$		0.0053	Yes				
$\hat{C}_{D,L\delta_e}$		0.1187	Yes				
$\hat{C}_{D,\delta_e^2}$		0.0362	Yes				

Table 5.15: Fit parameters for the BIRE aerodynamic force coefficients.

Coefficient	A	ω	φ	ζ	Δ_C
$\hat{C}_{L_0}$	-0.0144	2	1.5708	0.0621	–
$\hat{C}_{L,\alpha}$	0.1091	2	1.5708	3.5469	–
$\hat{C}_{L,\beta}$	-0.7216	2	0	0	–
$\hat{C}_{L,\overline{p}}$	0	0	0	0	–
$\hat{C}_{L,\overline{q}}$	2.0262	2	1.5708	1.5469	–
$\hat{C}_{L,\overline{r}}$	0.6798	2	0	0	–
$\hat{C}_{L,\delta_a}$	0	0	0	-0.0007	–
$\hat{C}_{L,\delta_e}$	0.7646	1	1.5708	0	-0.1822
$\hat{C}_{S_0}$	-0.0106	2	0	0	–
$\hat{C}_{S,\alpha}$	0.1834	2	0	0	–
$\hat{C}_{S,\beta}$	0.6805	2	1.5708	-0.6708	-0.1785
$\hat{C}_{S,\overline{p}}$	0	0	0	-0.0022	–
$\hat{C}_{S,L\overline{p}}$	0.0192	2	1.5708	0.2233	–
$\hat{C}_{S,\overline{q}}$	1.9916	2	0	0	–
$\hat{C}_{S,\overline{r}}$	-0.6134	2	1.5708	0.5976	–
$\hat{C}_{S,\delta_a}$	0.0015	2	1.5708	-0.0076	-0.0448
$\hat{C}_{S,\delta_e}$	0.7352	1	0	0	–
$\hat{C}_{D_0}$	0	0	0	0.0055	0.0154
$\hat{C}_{D,L}$	0	0	0	-0.0028	-0.0304
$\hat{C}_{D,L^2}$	0.0047	4	1.5708	0.1053	0.0714
$\hat{C}_{D,S}$	0.0255	2	0	-0	–
$\hat{C}_{D,S^2}$	0.3082	2	1.5708	0.5246	0.1118
$\hat{C}_{D,\overline{p}}$	0	0	0	0	–
$\hat{C}_{D,S\overline{p}}$	0	0	0	0.0013	–
$\hat{C}_{D,\overline{q}}$	0	0	0	0.0261	–
$\hat{C}_{D,L\overline{q}}$	0.3883	2	1.5708	0.37	–
$\hat{C}_{D,L^2\overline{q}}$	0	0	0	-0.0303	–
$\hat{C}_{D,\overline{r}}$	0	0	0	0	–
$\hat{C}_{D,S\overline{r}}$	0	0	0	-0.1146	–
$\hat{C}_{D,\delta_a}$	-0.0079	2	0	0	–
$\hat{C}_{D,S\delta_a}$	0.0492	2	1.5708	-0.0381	–
$\hat{C}_{D,\delta_e}$	-0.0061	1	1.5708	0.0015	–
$\hat{C}_{D,L\delta_e}$	0.183	1	1.5708	0	–
$\hat{C}_{D,\delta_e^2}$	-0.095	1	1.5708	0.4244	–

Table 5.16: Fit parameters for the BIRE aerodynamic moment coefficients.

Coefficient	A	ω	φ	ζ	Δ_C
$\hat{C}_{\ell_0}$	0.0002	2	0	0	–
$\hat{C}_{\ell,\alpha}$	-0.0023	4	0	0	–
$\hat{C}_{\ell,\beta}$	0.0017	2	1.5708	-0.0182	-0.0101
$\hat{C}_{\ell,\overline{p}}$	0.004	2	1.5708	-0.3069	–
$\hat{C}_{\ell,\overline{q}}$	0	0	0	0	–
$\hat{C}_{\ell,\overline{r}}$	0	0	0	0.0062	–
$\hat{C}_{\ell,L\overline{r}}$	0	0	0	0.1104	–
$\hat{C}_{\ell,\delta_a}$	0.014	2	1.5708	-0.1065	–
$\hat{C}_{\ell,\delta_e}$	0.0017	1	0	0	–
$\hat{C}_{m_0}$	0.0164	2	1.5708	-0.0022	-0.0196
$\hat{C}_{m,\alpha}$	-0.1381	2	1.5708	-0.0145	0.2865
$\hat{C}_{m,\beta}$	0.8299	2	0	0	–
$\hat{C}_{m,\overline{p}}$	-0.0102	2	0	0	–
$\hat{C}_{m,\overline{q}}$	-2.3551	2	1.5708	-2.5457	–
$\hat{C}_{m,\overline{r}}$	-0.7667	2	0	0	–
$\hat{C}_{m,\delta_a}$	0.0008	2	0	-0.0007	–
$\hat{C}_{m,\delta_e}$	-0.9115	1	1.5708	0	0.2914
$\hat{C}_{n_0}$	0.0048	2	0	0	–
$\hat{C}_{n,\alpha}$	-0.0929	2	0	0	–
$\hat{C}_{n,\beta}$	-0.3176	2	1.5708	0.313	-0.0326
$\hat{C}_{n,\overline{p}}$	0	0	0	0.001	–
$\hat{C}_{n,L\overline{p}}$	-0.0074	2	1.5708	-0.1223	0.0602
$\hat{C}_{n,\overline{q}}$	-0.9205	2	0	0	–
$\hat{C}_{n,\overline{r}}$	0.2894	2	1.5708	-0.2789	–
$\hat{C}_{n,\delta_a}$	0	0	0	0.0009	0.0122
$\hat{C}_{n,L\delta_a}$	-0.0169	2	1.5708	0.0157	0.0254
$\hat{C}_{n,\delta_e}$	-0.3527	1	0	0	–

A Discussion on the BIRE Coefficient Fits

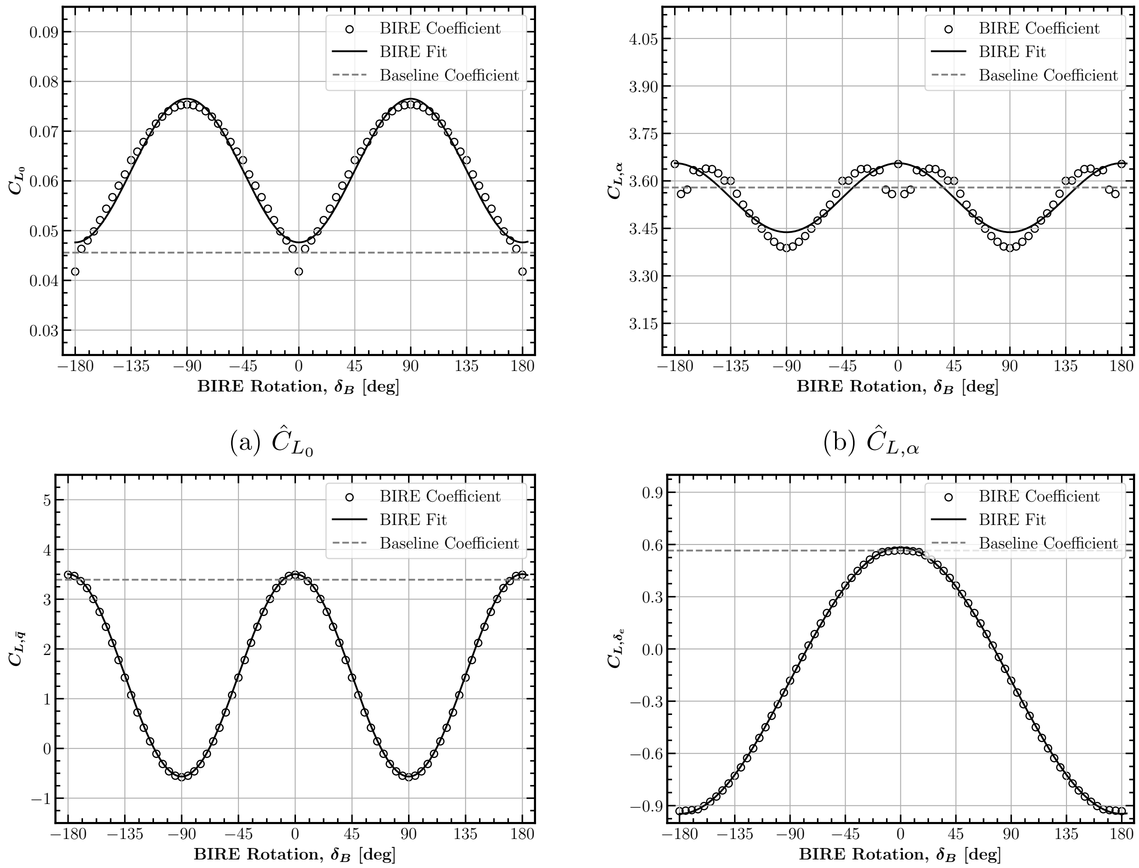
Figures 5.10–5.21 show each of the fits listed in Tables 5.15 and 5.16, along with the BIRE coefficients at each tail rotation angle and the value of the baseline coefficient given in Tables 5.11 and 5.12. These figures are separated into longitudinal and lateral components and certain trends can be noted immediately this way.

Note that, while the trends seen in these figures are largely periodic, there are certain instances of outliers in this data set. In general, these outliers have two reasonable implications. The first of these is that MachUpX, as a numerical lifting-line code, calculates the aerodynamic coefficients acting on an aircraft configuration using trailing sheets of vorticity. If the vorticity sheets from any two surfaces ever intersect, non-physical jumps in the aerodynamic coefficients can occur [89,104]. Thus, the intersection of these sheets in certain instances can cause outliers in the calculated sensitivity coefficients.

The other reason these outlier exist is more interesting from a research perspective. Since the aerodynamics of a rotating empennage have seen such limited analysis, there is a potential that these outliers represent physical non-linearities that require further study to understand. Thus, while these results represent a preliminary study into the aerodynamics of a rotating empennage, these trends require more study and research to understand the nature of the aerodynamic coefficients to a greater extent.

Beginning with the longitudinal lift coefficients in Fig. 5.10, we note that each of these coefficients are an even function of the BIRE rotation angle. The coefficients $\hat{C}_{L,\alpha}$ and $\hat{C}_{L,\bar{q}}$ follow a trend that will be seen often in the rest of the coefficients; they reach a maximum when the rotating tail is horizontal and a minimum when $\delta_B = 90°$. This trend explicitly shows the trade-off between longitudinal and lateral stability and control produced by rotating the horizontal tail.

The change in lift coefficient with elevator deflection represents another trend that is seen commonly with elevator sensitivities when rotating the horizontal tail. Note that $\hat{C}_{L,\delta_e}$ reaches a maximum with a horizontal tail and a minimum when the tail is reversed at $\delta_B = \pm 180°$, crossing zero at approximately $\delta_B = 90°$. The trends in the three longitudinal



(a) $\hat{C}_{L_0}$

(b) $\hat{C}_{L,\alpha}$

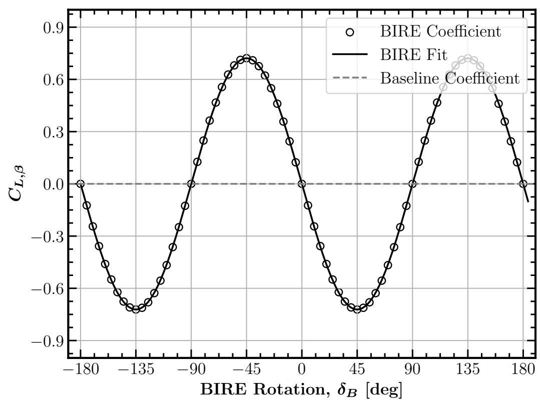
(c) $\hat{C}_{L,\overline{q}}$

(d) $\hat{C}_{L,\delta_e}$

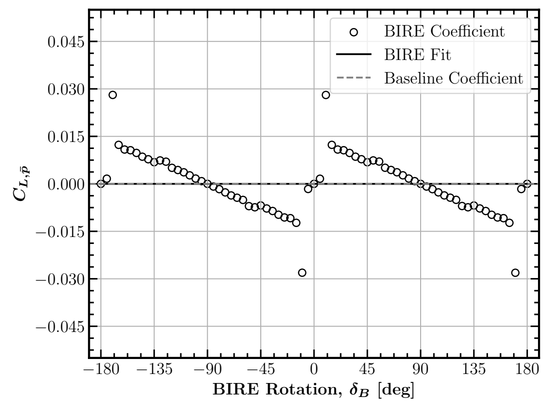
Fig. 5.10: Longitudinal BIRE Lift coefficient fits.

coefficients examined thus far are intuitive aerodynamically. However, $\hat{C}_{L_0}$ shows a trend that may, at first, appear counter-intuitive, since one would assume that the lift should attain a maximum when the tail is oriented horizontally. The effects of downwash are key to understanding this trend, as the downwash from the horizontal tail will produce a negative lift coefficient on the horizontal tail that is gradually rotated out of the downwash region until only the lift from the main wing is considered. This understanding makes the trend in Fig. 5.10a understandable and appropriate.

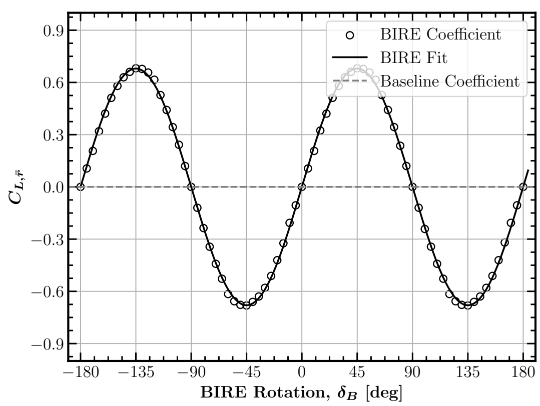
Figure 5.11 shows the lateral BIRE lift coefficients, which are all odd functions of the BIRE rotation angle. The coefficients $\hat{C}_{L,\overline{p}}$ and $\hat{C}_{L,\delta_a}$ all vary so little with tail rotation that they are considered constant at their average values. Coefficients $\hat{C}_{L,\beta}$ and $\hat{C}_{L,\overline{r}}$ represent another common trend with the BIRE fits, achieving a maximum magnitude at $\delta_B = \pm 45°$ and crossing zero when the tail is horizontal or vertical. This is logical, since these coefficients represent coupling between a longitudinal coefficient and a lateral aerodynamic parameter.



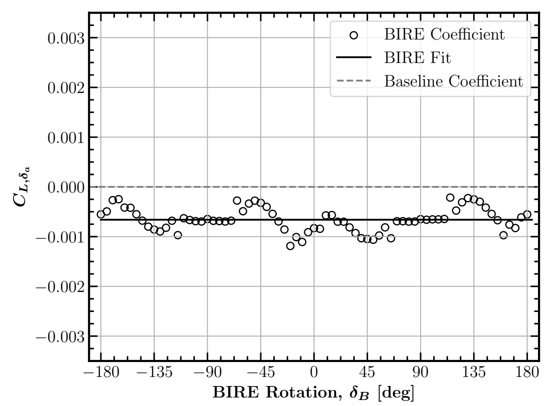
(a) $\hat{C}_{L,\beta}$



(b) $\hat{C}_{L,\overline{p}}$



(c) $\hat{C}_{L,\overline{r}}$



(d) $\hat{C}_{L,\delta_a}$

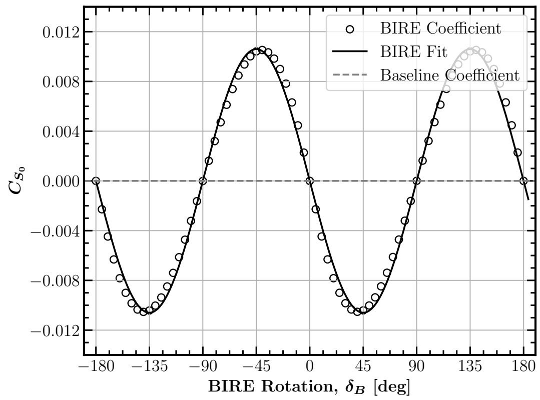
Fig. 5.11: Lateral BIRE Lift coefficient fits.

For example, the coefficient $\hat{C}_{L,\beta}$ does not influence the lift at $\delta_B = 0°$, since changes in sideslip produce equivalent changes in lift, regardless of the direction. This aligns with the assumption given in Eq. (4.32). When the horizontal tail is rotated to the vertical position, the same is true; however, at $\delta_B = 45°$, sideslip angles produce a maximum change in the lift coefficient due to the change in lift developed by the tail at positive and negative sideslip angles. The difference in sign between Figs. 5.11a and 5.11c are simply a matter of the definition of positive and negative sideslip and yaw rotation rate.

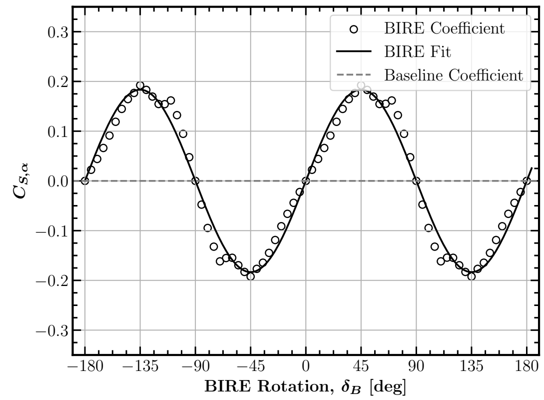
Examining now the side force coefficients, we see that the longitudinal side force coefficients in Fig. 5.12 follow the same odd-function pattern explored in the lateral components of the lift coefficient. That is, they reach a maximum at $\delta_B = 45°$ for all but the change in side force with respect to elevator deflection, which will be covered momentarily. Since a positive BIRE rotation angle moves the right-half of the horizontal tail downwards, the side force produced is positive as the longitudinal parameters α and $\overline{q}$ increase and produces a positive coefficient. In terms of pure side force, however, a positive tail rotation produces negative side force from the downwash on the main wing, which is demonstrated in the sign of $\hat{C}_{S_0}$. Finally, $\hat{C}_{S,\delta_e}$ reaches its maximum magnitude at $\delta_B = \pm 90°$, when the horizontal tail becomes essentially a large rudder and positive rotations with positive stabilator deflections produce a positive side force.

As the lateral components of the side force coefficient are sensitivities relating a lateral coefficient to lateral aerodynamic parameters, they follow the same patterns introduced in the longitudinal coefficients of lift. The coefficients $\hat{C}_{S,\beta}$, $\hat{C}_{S,L\overline{p}}$, and $\hat{C}_{S,\delta_a}$ each reach their maximum values when the tail is horizontal. As the tail is rotated vertically, the side force generated by the sideslip angle, lift and roll rate combination, and aileron deflection becomes more negative. For $\hat{C}_{S,\beta}$, this is because the increased vertical surface area generates more side force as the horizontal tail becomes a large vertical tail. As the vertical surface area of the tail increases, the lift differential induced from aileron deflections will create a more net-negative side force acting on an increasingly vertical tail, thus creating the same trend in $\hat{C}_{S,\delta_a}$. The reduction in the magnitude of $\hat{C}_{S,L\overline{p}}$ as the tail is rotated is for a similar reason: the vertical surface area introduced from tail rotation yields a negative side force component from the downwash and sidewash acting on the vertical tail.

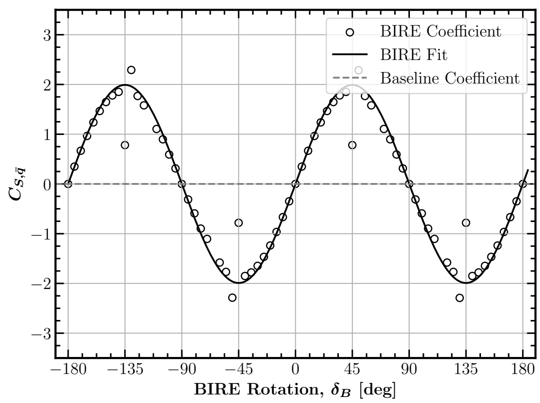
Though we see similar trends with tail rotation for the coefficients $\hat{C}_{S,\overline{p}}$ and $\hat{C}_{S,\overline{r}}$, the effect of changes in $\hat{C}_{S,\overline{p}}$ is negligible. When the tail is completely horizontal, $\hat{C}_{S,\overline{r}}$ provides no change to the total side force coefficient. However, when rotated to $\delta_B = 90°$, the tail



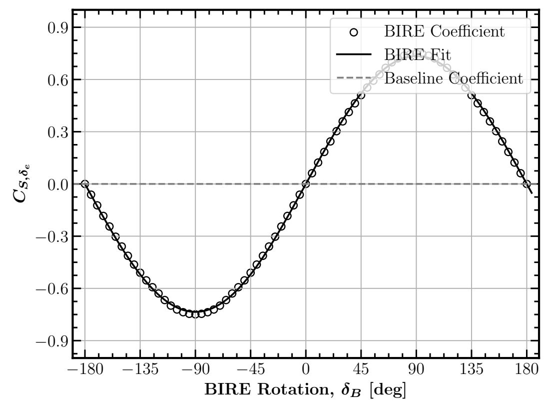
(a) $\hat{C}_{S_0}$



(b) $\hat{C}_{S,\alpha}$



(c) $\hat{C}_{S,\overline{q}}$

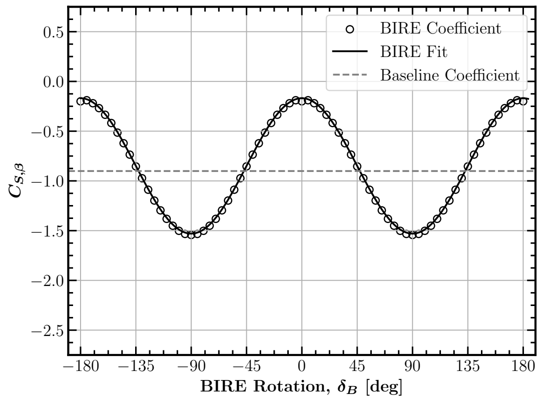


(d) $\hat{C}_{S,\delta_e}$

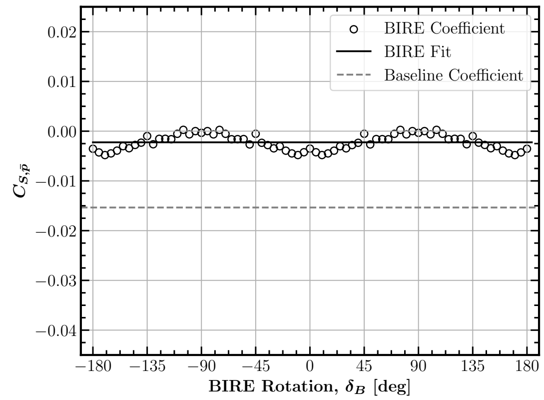
Fig. 5.12: Longitudinal BIRE Side Force coefficient fits.

provides a side force of the same sign as the direction of yawing rate. Of note is that both $\hat{C}_{S,\beta}$ and $\hat{C}_{S,\overline{r}}$ both reach magnitudes larger than the baseline aircraft at around $\delta_B = 45°$. This observation will be repeated again for later lateral coefficients, indicating that, in terms of total lateral control, the BIRE has more lateral control authority than the baseline aircraft. This will be explored in greater detail in Chapter 7.

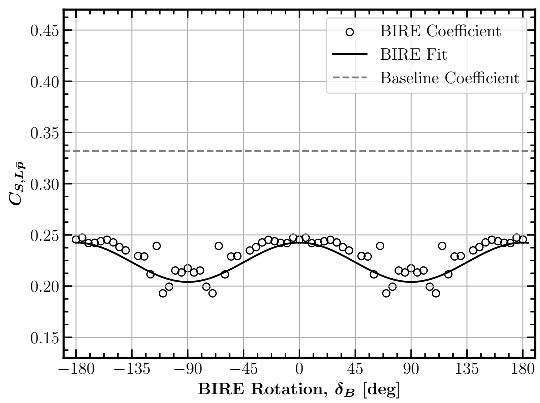
The abundance of drag coefficients necessitates a slightly longer discussion on the longitudinal and lateral components of its makeup. Emphasis here is placed on the changes in drag between the baseline and BIRE configurations, though the reader should be reminded that the drag model used focuses entirely on the drag induced by pressure differences and neglects the effects of skin friction and viscous drag, among others. Figure 5.14 shows the



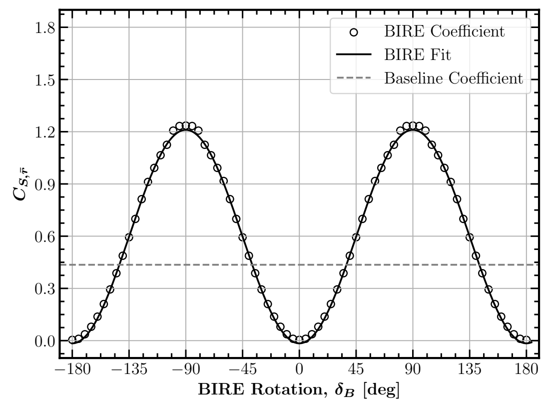
(a) $\hat{C}_{S,\beta}$



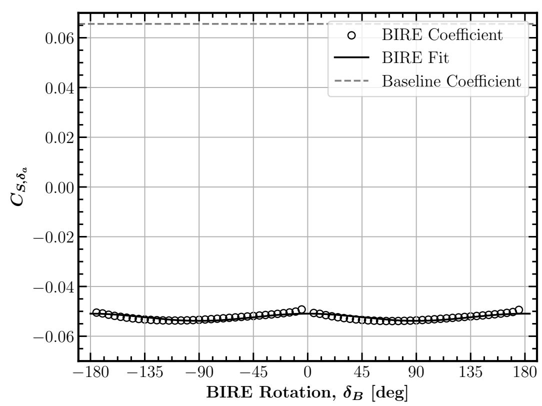
(b) $\hat{C}_{S,\overline{p}}$



(c) $\hat{C}_{S,L\overline{p}}$



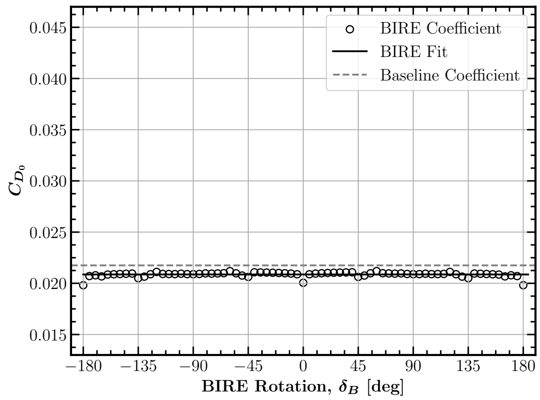
(d) $\hat{C}_{S,\overline{r}}$



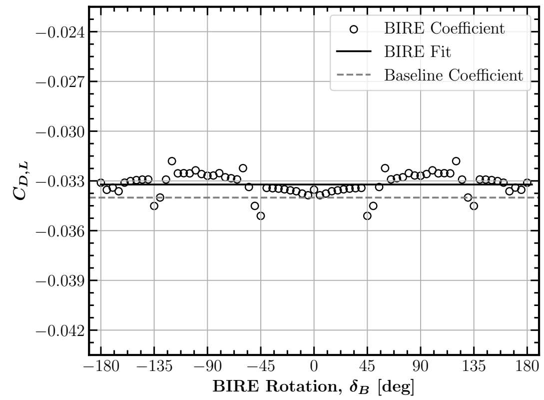
(e) $\hat{C}_{S,\delta_a}$

Fig. 5.13: Lateral BIRE Side Force coefficient fits.

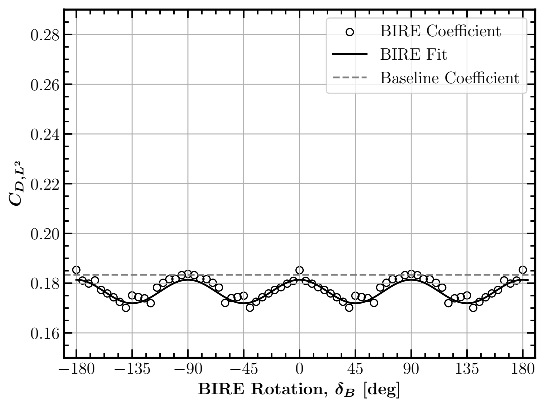
components of the drag coefficient related to longitudinal parameters. Here, it is immediately noted that the inherent drag on the BIRE aircraft, given with $\hat{C}_{D_0}$ is not modeled with



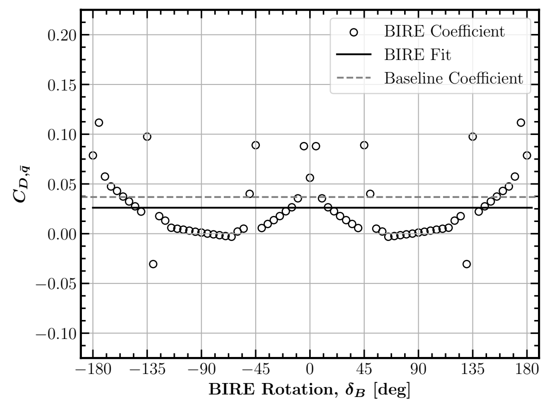
(a) $\hat{C}_{D_0}$



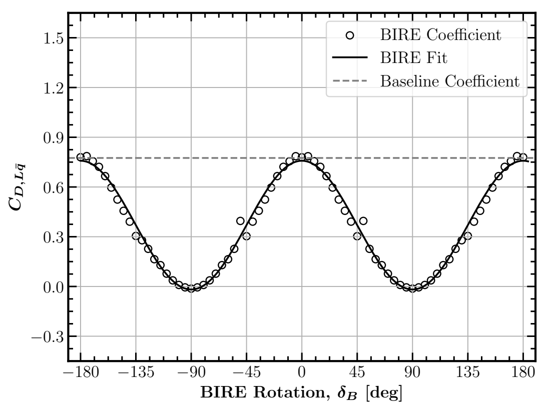
(b) $\hat{C}_{D,L}$



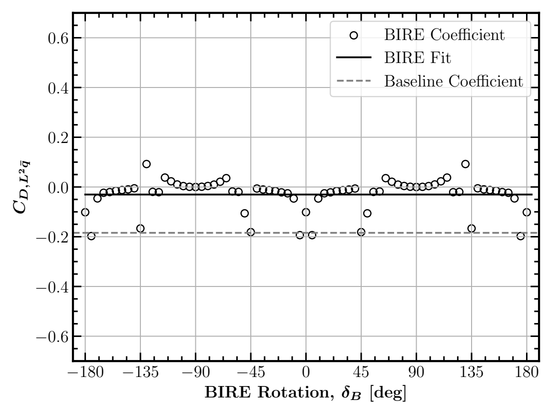
(c) $\hat{C}_{D,L^2}$



(d) $\hat{C}_{D,\overline{q}}$



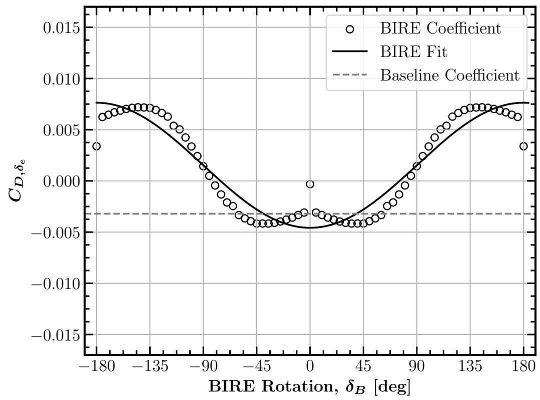
(e) $\hat{C}_{D,L\overline{q}}$



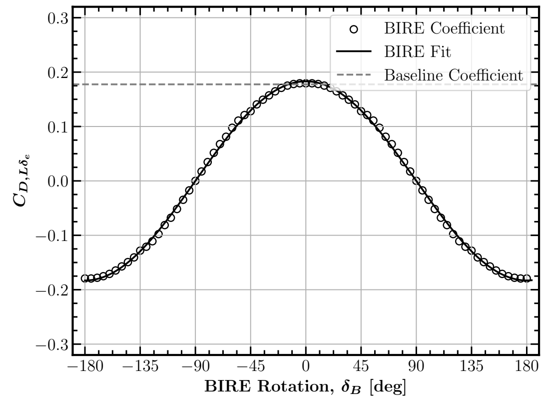
(f) $\hat{C}_{D,L^2\overline{q}}$

Fig. 5.14: Longitudinal BIRE Drag coefficient fits.

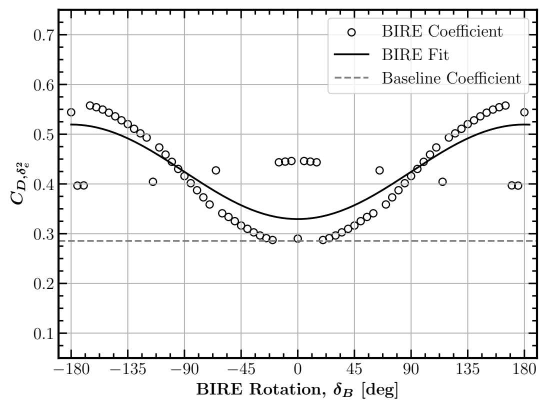
BIRE rotation (which would be largely expected) and is significantly less than that of the baseline aircraft. The shift in minimum drag location, $\hat{C}_{D,L}$ is also a very weak function of



(g) $\hat{C}_{D,\delta_e}$



(h) $\hat{C}_{D,L\delta_e}$



(i) $\hat{C}_{D,\delta_e^2}$

Fig. 5.14: Longitudinal BIRE Drag coefficient fits (continued).

tail rotation and is similar in value to that of the baseline aircraft. Finally, the overall drag polar of the BIRE is more shallow than the baseline aircraft with a lower value of $\hat{C}_{D,L^2}$. An interesting phenomena is shown in Fig. 5.14c, with the most shallow drag polar in lift existing at $\delta_B = \pm 45°$ and $\pm 135°$. This may be an effect caused by downwash, though further investigation would be required to confirm this.

Of the quadratic coefficients in pitch rate for drag, both $\hat{C}_{D,\overline{q}}$ and $\hat{C}_{D,L^2\overline{q}}$ are not a strong function of BIRE rotation angle. In fact, these coefficients are each lower in magnitude than their baseline aircraft counterparts, indicating that changes in pitch rate, in general, create a smaller increment in drag than for the baseline aircraft. The only term in this quadratic that varies significantly with BIRE rotation angle is $\hat{C}_{D,L\overline{q}}$, given in Fig. 5.14e. This value is maximized when the tail is horizontal and reaches a value of $\hat{C}_{D,L\overline{q}} \approx 0$ when rotated vertically. Thus, rotating the horizontal tail in the BIRE reduces the drag caused by changes in pitch rate, though it will be shown when examining the lateral components of drag that this too represents a longitudinal-lateral trade-off in the BIRE design. Note that the pattern developed in $\hat{C}_{D,\overline{q}}$ could be better represented with a fit of the form $\sin(|\delta_B|)$ instead of a pure sine wave. Future analysis could determine the effect of additional fit types on the aerodynamics of the aircraft.

Finally, the last three longitudinal components of drag are each related to the stabilator deflection. The trends shown in Figs. 5.14g–5.14i are each symmetric functions about a horizontal tail configuration. The data trends in Fig. 5.14g represent an example of the benefit of further analysis, since the pattern is periodic in nature, but is not exactly represented by a sinusoid. Downwash is likely a contributing factor to the behavior of the coefficient $\hat{C}_{D,\delta_e}$ near the horizontal position and beyond $\delta_B = \pm 90°$. When the tail reaches a vertical position, note again that a sensitivity coefficient with a value of zero does not mean that there is no change in drag with stabilator deflection at this point. Rather, control surface deflection in this position have an identical effect on the drag, whether the deflections are positive or negative.

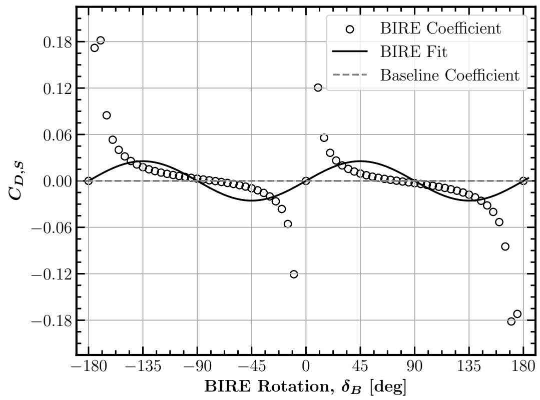
The coupling of lift and stabilator deflection represented by the coefficient $\hat{C}_{L,\delta_e}$ has a similar transition from positive to negative when passing through a vertical tail configuration. This change in sign is a result of the direction of positive stabilator deflection when the tail is inverted. Maintaining a positive drag contribution requires that when a positive deflection creates negative lift that the coefficient itself attains a negative value. Lastly, $\hat{C}_{D,\delta_e^2}$ shows an interesting trend, with the drag paid for stabilator deflection increasing consistently with the square of elevator deflection for all angles other than the horizontal. This term indicates another area where additional understanding of the aerodynamics at play with a rotating tail could help develop intuition into the cause of this trend.

With the longitudinal elements of the drag coefficient of the BIRE addressed, the lateral components of the drag given in Fig. 5.15 can be discussed. The coefficient representing the shift in minimum drag from side force, $\hat{C}_{D,S}$, is better represented by a shifted tangent function. However, for this analysis, it was determined that the consistency in definition between coefficients will be helpful when a linearized controller is analyzed in Chapter 8. The same trend in data is shown in $\hat{C}_{D,\overline{p}}$, though the function is not modeled based on the results of the sensitivity study in Table 5.14. The results here are clear; according to this model, asymmetries in the aircraft due to tail rotation will result in an increase in the drag experienced by the aircraft.

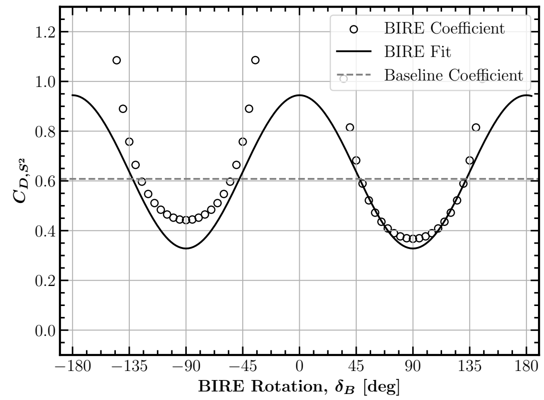
Trends for the quadratic term in the drag-side force polar, $\hat{C}_{D,S^2}$, show that tail rotation decreases this term until it reaches its vertical position. This is another case of trade-off between longitudinal and lateral effects, since referring to Fig. 5.14c shows that the corresponding lift term is maximized at $\delta_B = \pm 90°$. The changes in $\hat{C}_{D,S\overline{p}}$, $\hat{C}_{D,\overline{r}}$, and $\hat{C}_{D,S\overline{r}}$ were determined to be negligible from the sensitivity study in Table 5.14 and therefore are considered constant as a function of tail rotation.

The term $\hat{C}_{D,\delta_a}$ is another that measures asymmetry in the aircraft, and its variation with tail rotation can reasonably be concluded to be caused by downwash effects. Lastly for the lateral drag coefficients is $\hat{C}_{D,S\delta_a}$, which reaches a minimum when the tail is vertical and is nearly zero when horizontal according to the fit. Again, this is a term that is not well-understood from physical intuition alone. It can reasonably be assumed that the negative value of this coefficient (causing a decrement in drag about the linearized point) is a result of the flow being oriented so as to produce less lift, which is the primary cause of increase in pressure drag as modeled by MachUpX.

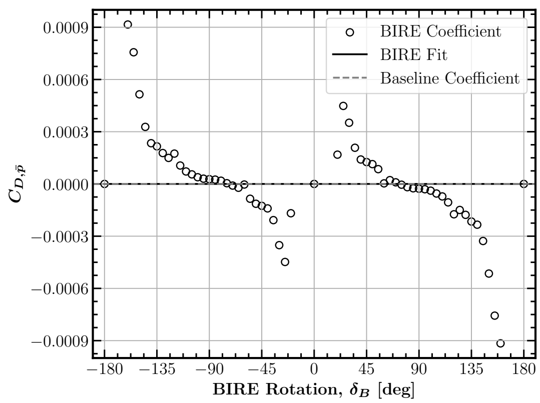
Transitioning now to the model of the aerodynamic moments, several trends will be repeated from the analysis of the aerodynamic forces. For example, $\hat{C}_{\ell_0}$ in Fig. 5.16 varies will tail rotation as a representation of asymmetries, just like $\hat{C}_{S_0}$ and $\hat{C}_{D,S}$ in Figs. 5.12a and 5.15a, respectively. It is likely primarily caused by downwash effects, as is $\hat{C}_{\ell,\alpha}$. This term cycles twice as fast as $\hat{C}_{\ell_0}$ and could potentially be caused by the shed wing-tip vortices combined with downwash effects, since the added zeros at $\delta_B = \pm 45°$ correspond to locations where the y-and z- components of distance from the centerline of the aircraft



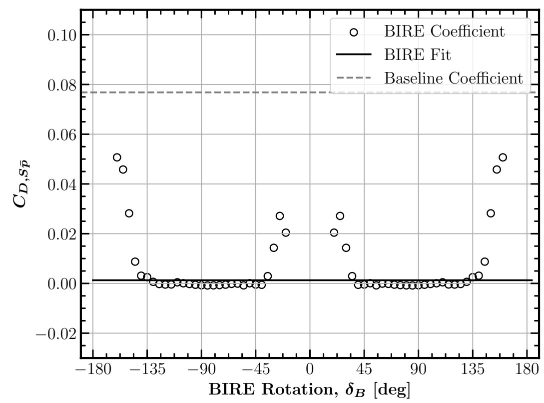
(a) $\hat{C}_{D,S}$



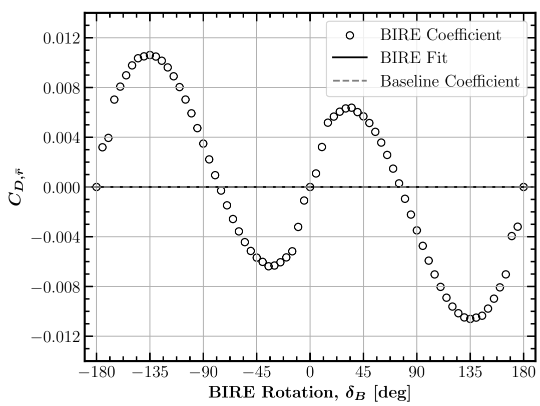
(b) $\hat{C}_{D,S^2}$



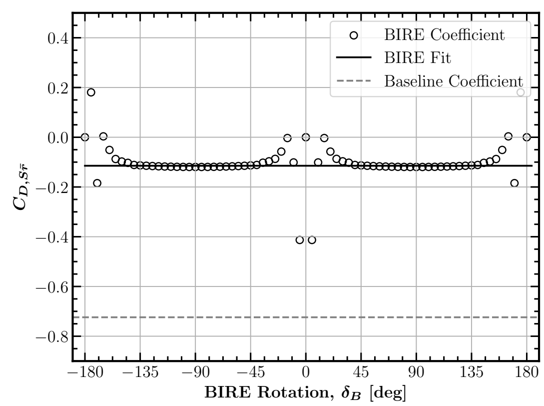
(c) $\hat{C}_{D,\overline{p}}$



(d) $\hat{C}_{D,S\overline{p}}$



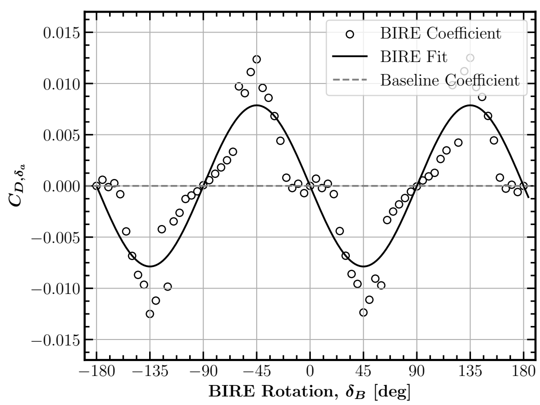
(e) $\hat{C}_{D,\overline{r}}$



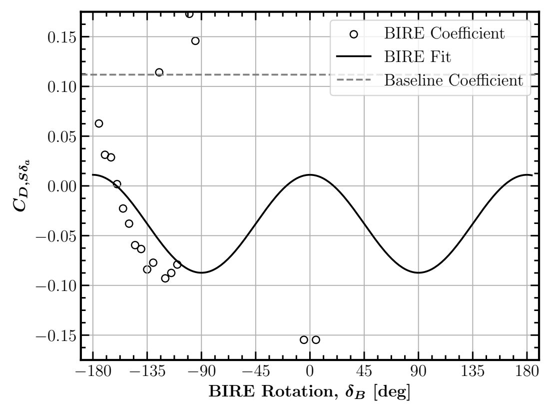
(f) $\hat{C}_{D,S\overline{r}}$

Fig. 5.15: Lateral BIRE Drag coefficient fits.

are equal. Further study, in addition to a higher fidelity physical model, would need to be conducted to understand these trends completely.

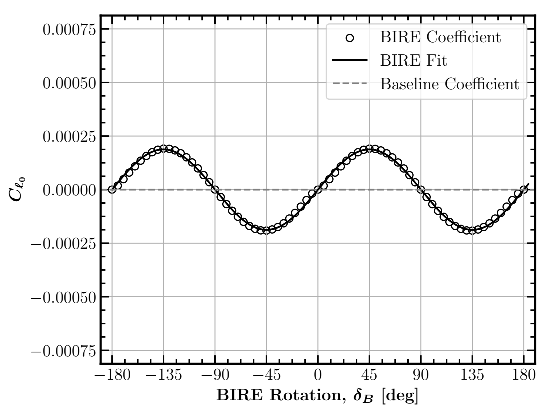


(g) $\hat{C}_{D,\delta_a}$

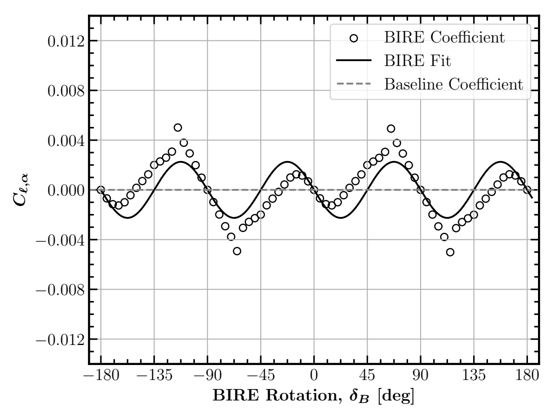


(h) $\hat{C}_{D,S\delta_a}$

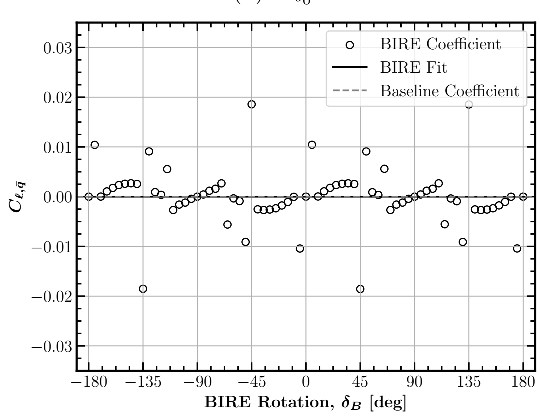
Fig. 5.15: Lateral BIRE Drag coefficient fits (continued).



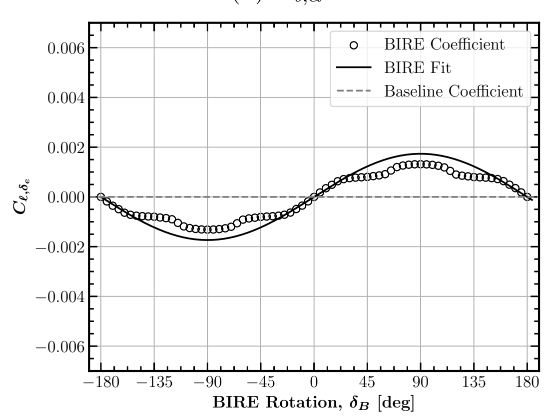
(a) $\hat{C}_{\ell_0}$



(b) $\hat{C}_{\ell,\alpha}$



(c) $\hat{C}_{\ell,\overline{q}}$



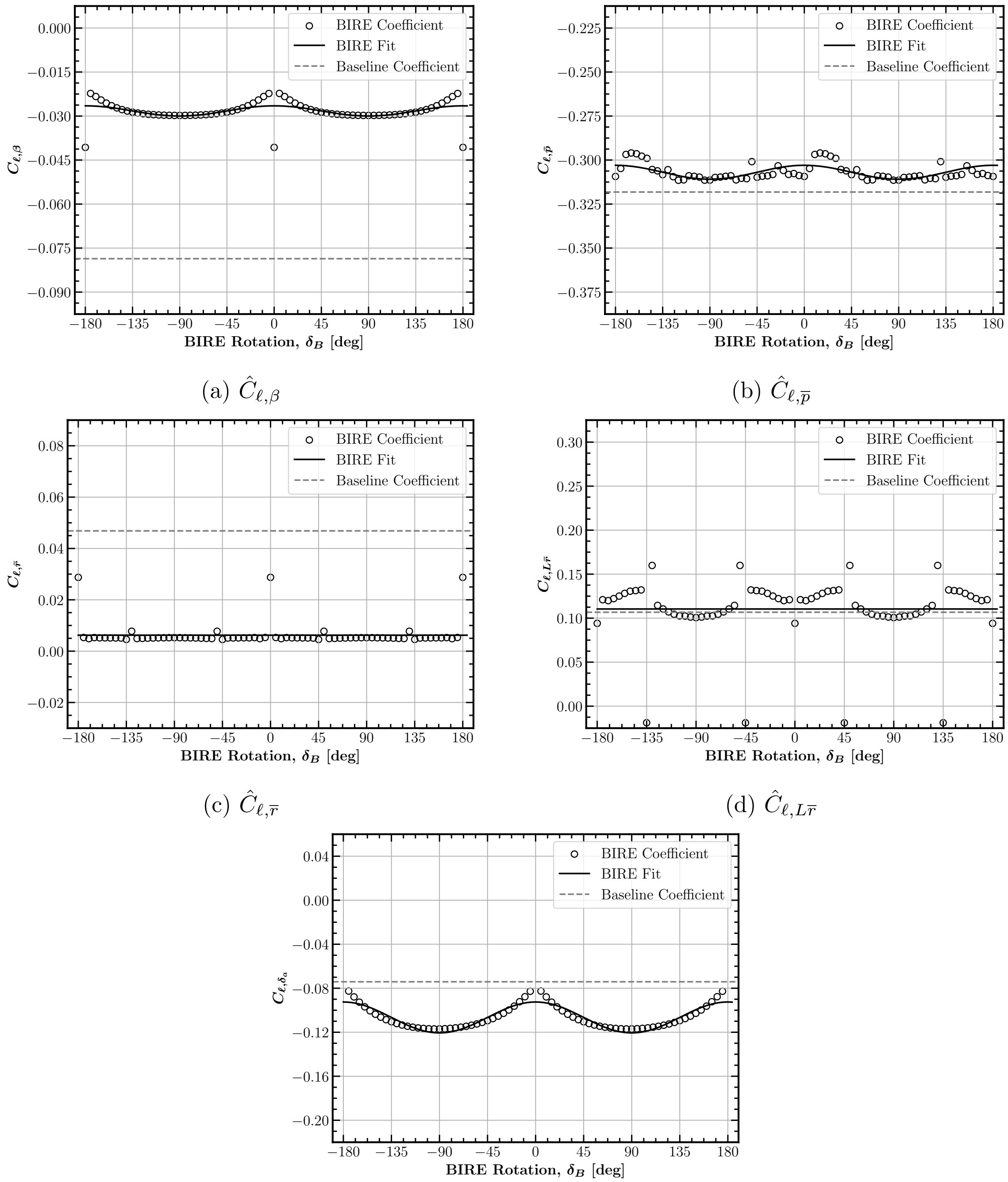
(d) $\hat{C}_{\ell,\delta_e}$

Fig. 5.16: Longitudinal BIRE Rolling Moment coefficient fits.

The term $\hat{C}_{\ell,\overline{q}}$ cycles at the same rate as $\hat{C}_{\ell,\alpha}$ and could be caused by similar effects. However, this term was determined to be inconsequential by the sensitivity study conducted for the BIRE coefficients and therefore is kept constant across the BIRE rotation angles. Last for the longitudinal components of the rolling moment coefficient is $\hat{C}_{\ell,\delta_e}$, which shows the slower frequencies characteristic of the elevator deflection. As a longitudinal-lateral coupling term, it crosses zero when the tail is horizontal, in contrast to $\hat{C}_{L,\delta_e}$ and $\hat{C}_{D,\delta_e}$. The trend seen for this coefficient is likely due to the direction of the strong wing-tip vortices shed from the main wing, which will cause the upward-facing portion of the tail to experience an increased angle of attack when compared to the lower portion. This result would yield the rolling moment coefficient patterns shown in Fig. 5.16d.

The lateral terms associated with the rolling moment coefficient are shown in Fig. 5.17. Sensitivities to yawing rate, given by $\hat{C}_{\ell,\overline{r}}$ and $\hat{C}_{\ell,L\overline{r}}$ are modeled as constants, while the terms $\hat{C}_{\ell,\beta}$, $\hat{C}_{\ell,\overline{p}}$, and $\hat{C}_{\ell,\delta_a}$ follow similar trends. These three coefficients reach a maximum when the tail is horizontal and a minimum when the tail is vertical. The roll stability derivative, $\hat{C}_{\ell,\beta}$, is dominated by the effect of the main wing; however, the orientation of the tail increases the stability of the BIRE about the roll axis when vertical. This is likely due again to the effects of wing tip vortices, which will produce a subtle increase in rolling moment under sideslip.

The effect of rolling rate on the rolling moment coefficient is also dominated by the main wing, since the lift it produces increases the discrepancy in lift on each semispan caused by plunging. However, this same effect is produced by the tail as well, and is augmented when the tail is rotated out of the downwash of the main wing. This could be the main cause of variations in the coefficient $\hat{C}_{\ell,\overline{p}}$. Finally, the rolling moment control derivative, $\hat{C}_{\ell,\delta_a}$, likely varies for nearly the same reason. When rotated out of the downwash of the main wing, the rolling moment produced by aileron deflections (and subsequently the anti-symmetric deflections of the horizontal tail) increases in magnitude. From this analysis, it should be noted that the rolling moment coefficient components are perhaps those that would benefit the most from understanding gained by higher-fidelity studies. Various interpretations have



(a) $\hat{C}_{\ell,\beta}$

(b) $\hat{C}_{\ell,\overline{p}}$

(c) $\hat{C}_{\ell,\overline{r}}$

(d) $\hat{C}_{\ell,L\overline{r}}$

(e) $\hat{C}_{\ell,\delta_a}$

Fig. 5.17: Lateral BIRE Rolling Moment coefficient fits.

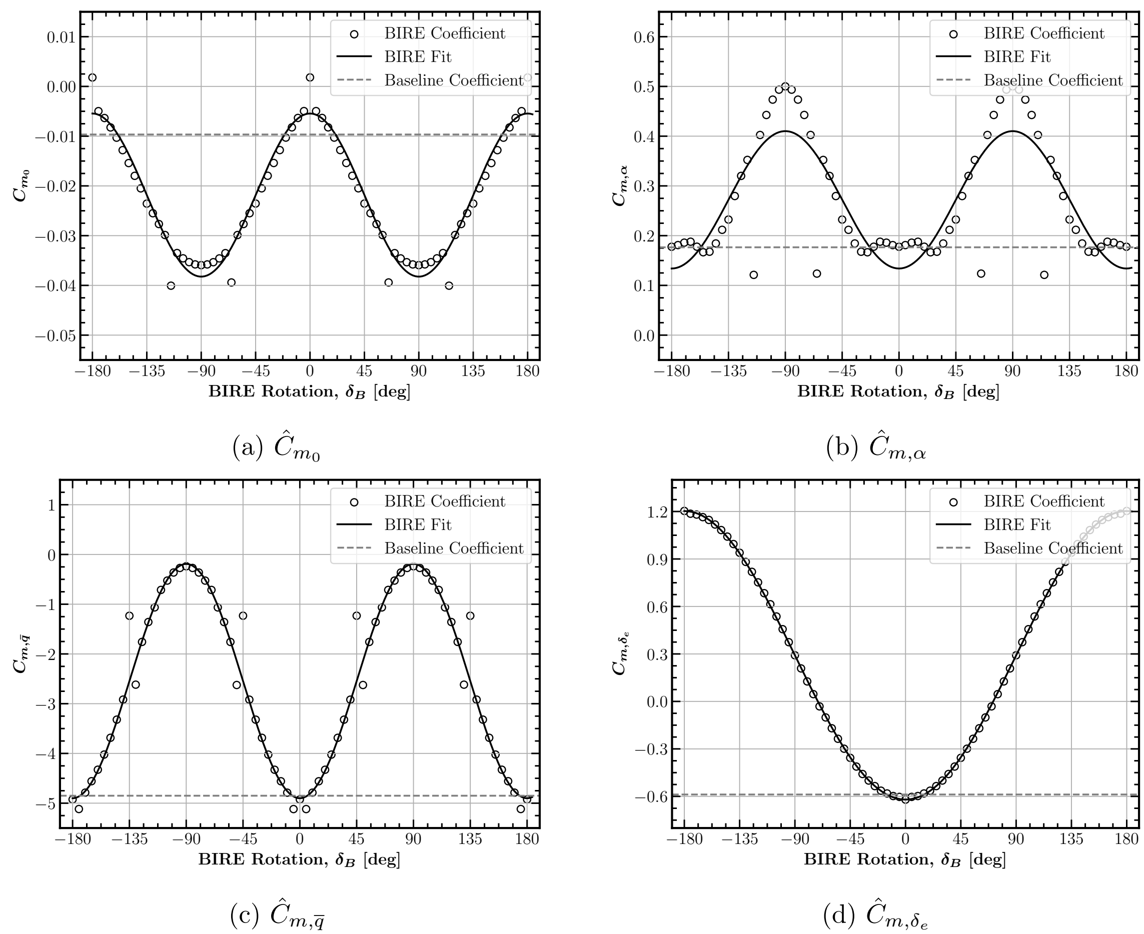
been given here, but further research is required to understand the mechanisms behind the coefficients' variation with tail rotation.

The final two aerodynamic moments, the pitching and yawing moments, are perhaps the most important to the aircraft in terms of the effects of a rotating tail. Both are dominated by tail effects and play crucial roles in stability and control. Figure 5.18 shows the longitudinal components of the pitching moment coefficient. They are each even functions of the BIRE rotation angle, with only $\hat{C}_{m_0}$ varying in its trends. The nominal pitching moment coefficient, represented by $\hat{C}_{m_0}$, attains a maximum when the tail is horizontal and a minimum when vertical. A positive nominal pitching moment is produced by the aircraft when the tail is horizontal due to the positive lift produced by the main wing forward of the center of gravity and the downwash on the tail producing a further positive moment. Rotating the horizontal tail out of the way removes its effect and lowers the nominal pitching moment accordingly. This allows the nominal pitching moment to sink even below that of the baseline aircraft, since the effects of downwash no longer push the pitching moment in the positive direction from that of the main wing.

In contrast, the pitching moment slope, pitch damping derivative, and pitch control derivative each have a minimum when the tail is horizontal and increase to a maximum when vertical. In the case of $\hat{C}_{m,\alpha}$, this results in an aircraft configuration at $\delta_B = 90°$ that is unstable in pitch ($C_{m,\alpha} > 0$) due to a lack of the contributions from a horizontal tail [9]. This cross-over point occurs at approximately $\delta_B = 45°$, indicating that tail rotation could be an interesting way to adjust aircraft stability mid-flight. The interesting behavior of this coefficient near the horizontal position suggests that further research may be fruitful in uncovering additional understanding of the physics of a rotating tail.

Since the horizontal tail contributes primarily to pitch damping in an aircraft, rotating the tail causes large changes in the pitch damping derivative, as shown in Fig. 5.18c. This illustrates another intuitive trade-off for the BIRE aircraft, which is that rotating the tail produces a decrease in the pitch damping of the aircraft, essential for favorable characteristics in the short-period and phugoid dynamic modes of the aircraft [12, 13].

Rounding out the longitudinal components of the pitching moment coefficient is the pitch control derivative, which follows a very intuitive pattern in its change with BIRE



(a) $\hat{C}_{m_0}$

(b) $\hat{C}_{m,\alpha}$

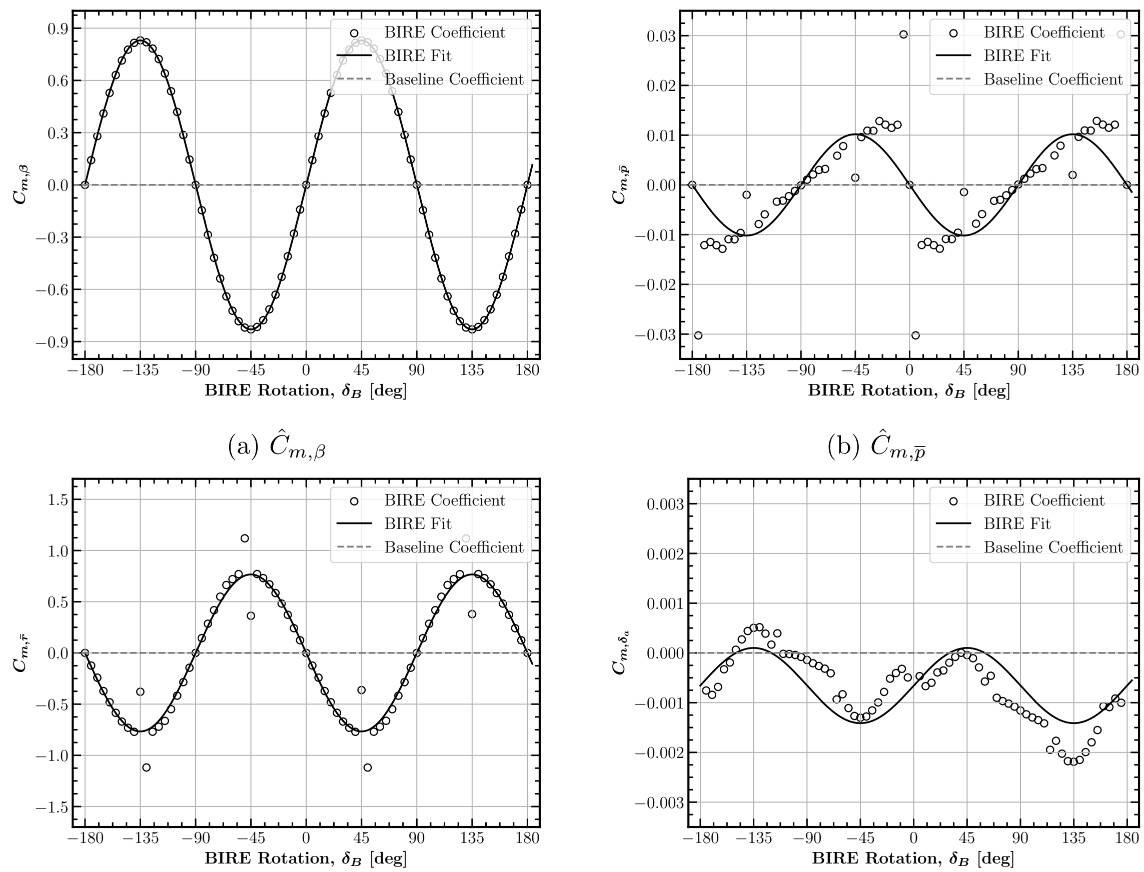
(c) $\hat{C}_{m,\overline{q}}$

(d) $\hat{C}_{m,\delta_e}$

Fig. 5.18: Longitudinal BIRE Pitching Moment coefficient fits.

rotation angle. The difference in its minimum value when compared to the baseline aircraft is likely due entirely to the lack of dihedral in the BIRE design. This allows the stabilators to be even more effective at generating pitching moment through deflection. Passing through a vertical orientation causes the pitch control coefficient to become completely ineffective at generating a pitching moment in this linear model, which becomes a concerning trade-off in terms of control authority for the BIRE design.

The lateral components of the pitching moment model shown in Fig. 5.19 are each odd functions of the BIRE rotation angle. When rotated in the positive direction, the horizontal tail in sideslip will generate lift and side force in the negative direction, causing a nose-up pitching moment as shown in Fig. 5.19a. Similarly oriented, the coefficient $\hat{C}_{m,\delta_a}$ barely contributes above the sensitivity parameter given in Eq. (5.10). Its variation with BIRE rotation angle is likely due to the effects of downwash from the main wing, since it is unclear how aileron deflections would affect the pitching moment otherwise.



(a) $\hat{C}_{m,\beta}$

(b) $\hat{C}_{m,\overline{p}}$

(c) $\hat{C}_{m,\overline{r}}$

(d) $\hat{C}_{m,\delta_a}$

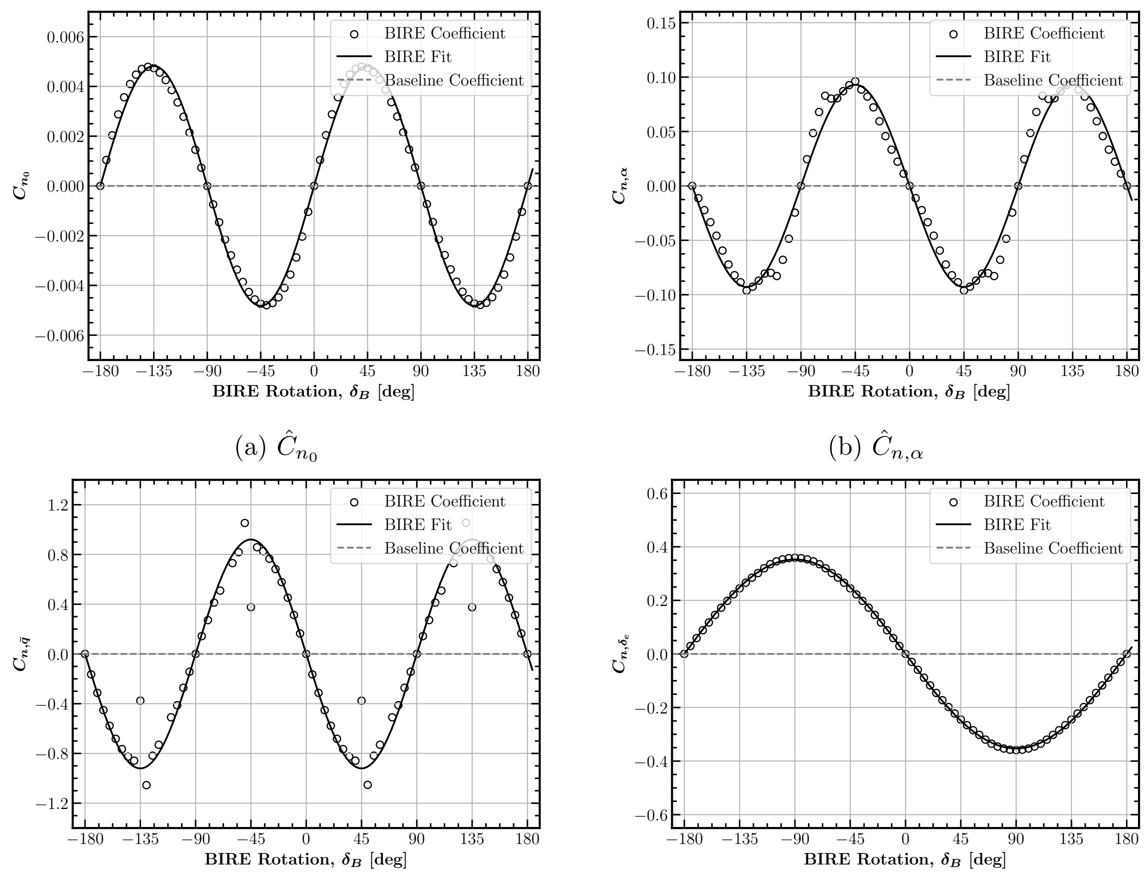
Fig. 5.19: Lateral BIRE Pitching Moment coefficient fits.

Sensitivity of the pitching moment to roll rate is shown in Fig. 5.19b. The resulting fit shows an interesting coupling between changes in roll rate and the pitching moment. It is likely that the lowered tail semispan in rotation generates more lift (creating a nose-down pitching moment) due to the combined effects of downwash and a larger perceived angle of attack to the flow. This would explain the negative pitching moment generated by a positive roll rate with the tail at $\delta_B = 45°$. When rotated vertically, the tail contributes nothing to the pitching moment, as noted in $\hat{C}_{m,\bar{r}}$ as well. The sensitivity of pitching moment to yawing rate follows the same pattern as that given by $\hat{C}_{m,\beta}$ with only a change in sign. This results from the definition of positive yaw rate and the movement of the flow, which will be in the opposite direction to the flow incidence seen by the tail under conditions of sideslip.

The final aerodynamic moment coefficient is the yawing moment, whose longitudinal components are shown in Fig. 5.20. As expected at this stage of the analysis, these cross-coupled terms are odd functions of the BIRE rotation angle. Due to the similar effect exerted by angle of attack and pitching rate, the terms $\hat{C}_{n,\alpha}$ and $\hat{C}_{n,\bar{q}}$ follow identical patterns and reach a minimum at $\delta_B = 45°$. Yet again, this demonstrates a trade-off between longitudinal and lateral control offered by the BIRE rotation. When rotated through an angle of attack or experiencing a pitching rate, a positively rotated horizontal tail generates a positive side force behind the center of gravity, which, in turn, generates a negative yawing moment.

The nominal yawing moment, $\hat{C}_{n_0}$, can be assumed to vary with BIRE rotation angle purely based on downwash and sidewash from the main wing. Thus, when rotated in the positive direction, the lowered right-half semispan of the tail would produce a negative side force, thus creating a positive yawing moment due to its position behind the center of gravity. Perhaps one of the most important control derivatives of the BIRE is $\hat{C}_{n,\delta_e}$, since it has effectively replaced rudder control in the BIRE design. Unsurprisingly, it has its maximum effect on the yawing moment when the tail is oriented vertically. Using the information in Table 5.16 and the baseline value for the yaw control derivative C_{n,δ_r} in Table 5.12, a quick calculation reveals that the BIRE can achieve an equivalent control derivative with only 14 degrees of BIRE deflection. This provides a positive benchmark for the BIRE as a control concept, since one concern is the rate at which the BIRE would need to rotate to be able to control the aircraft in yaw. Additional analysis will be performed in later chapters, but this provides an initial point of reference for those discussions.

The lateral components of the yawing moment coefficient can be seen in Fig. 5.21. The coefficients $\hat{C}_{n,\bar{p}}$ and $\hat{C}_{n,\delta_a}$ are not modeled due to their low contribution to the total yawing moment, but are both nearly zero. $\hat{C}_{n,\delta_a}$ is an important coefficient component, since it is



(a) $\hat{C}_{n_0}$

(b) $\hat{C}_{n,\alpha}$

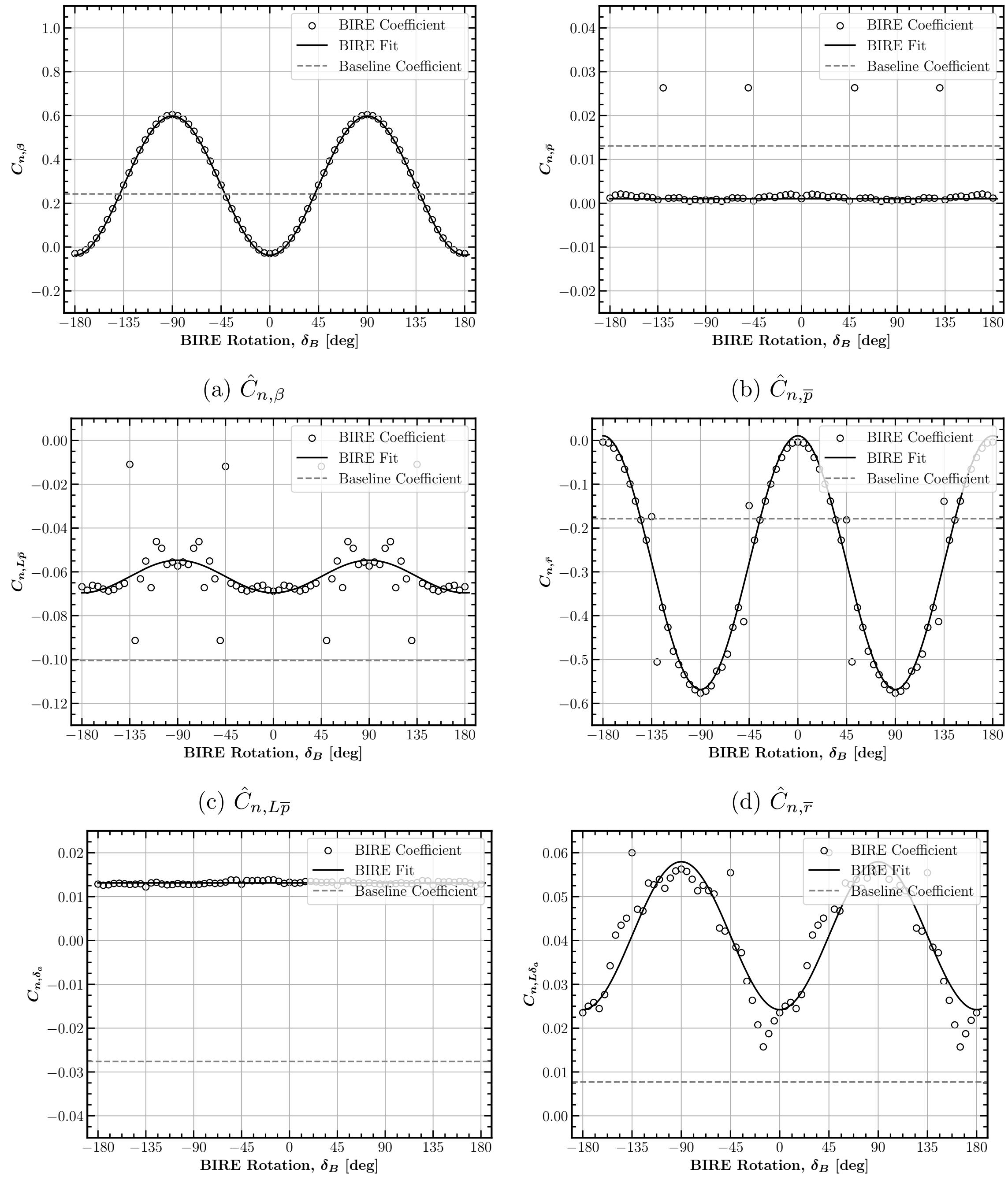
(c) $\hat{C}_{n,\overline{q}}$

(d) $\hat{C}_{n,\delta_e}$

Fig. 5.20: Longitudinal BIRE Yawing Moment coefficient fits.

generally used to describe the adverse yaw characteristics of an aircraft. Adverse yaw was described in Chapter 2, and is generally an unfavorable characteristic for an aircraft. In this case, due to the adjustments to the MachUpX coefficients, the BIRE produces even a small amount of proverse yaw and is not a significant function of BIRE rotation angle. This is an assumption that will need to be verified with further testing, though the lack of a vertical tail may mitigate some of the adverse yaw.

The yaw stability coefficient, unsurprisingly, increases from zero when horizontal to above the value of the baseline aircraft when vertical. Referring again to Tables 5.8 and 5.16, the BIRE rotation angle required to produce the same yaw stability present in the baseline aircraft is approximately $\delta_B = \pm 46°$. Referring back to the pitch stability in Fig.



(a) $\hat{C}_{n,\beta}$ (b) $\hat{C}_{n,\overline{p}}$

(c) $\hat{C}_{n,L\overline{p}}$ (d) $\hat{C}_{n,\overline{r}}$

(e) $\hat{C}_{n,\delta_a}$ (f) $\hat{C}_{n,L\delta_a}$

Fig. 5.21: Lateral BIRE Yawing Moment coefficient fits.

5.18b, we see that this still represents an aircraft with "relaxed" longitudinal stability. Also, the pitch control derivative at this BIRE rotation angle is approximately on par with that given by the baseline aircraft according to Fig. 5.18d. Again, this indicates that the BIRE should be able to maintain its yaw stability, pitch stability, and control in similar manner to that of the baseline aircraft.

The non-linear effect of lift and roll rate on the yawing moment is increased in magnitude slightly from the baseline aircraft with a horizontal tail. This is likely due to the fact that the horizontal tail generates more lift without the dihedral, which in turn is combined with roll rate to induce a negative yawing moment. When rotated vertically, the lift produced by the tail decreases, thus decreasing the magnitude of the coefficient $\hat{C}_{n,L\overline{p}}$. Also dependent on lift is the coefficient $\hat{C}_{n,L\delta_a}$, which follows the same pattern as $\hat{C}_{n,L\overline{p}}$ for similar reasons.

The yaw damping derivative, $\hat{C}_{n,\overline{r}}$, is another important derivative to the dynamics of the aircraft as well as its control capabilities [22, 23]. It is again insightful to calculate the BIRE rotation angle required to match the damping of the baseline aircraft. Using the values from Tables 5.8 and 5.16, a BIRE rotation angle of approximately $\delta_B = \pm 35°$ will give the BIRE an equivalent yaw damping derivative as the baseline aircraft. These angles have not been prohibitive thus far, and could allow the BIRE to match exactly the damping and control derivatives achieved by the baseline aircraft when necessary with a reasonable actuation device.

In conclusion, several consistent patterns can be found from examining Figs. 5.10–5.21. The first is that nearly all of the sensitivity coefficients dealing with the aerodynamic angles, rotation rates, and aileron deflection have the same frequency when measuring their variation with tail rotation. This is an interesting pattern that could provide insight into the underlying physics connecting each of the coefficients. Each of the nominal coefficients also share this frequency of variation. In contrast, the variation frequency of all sensitivity coefficients in stabilator deflection is half of that given to the other coefficients.

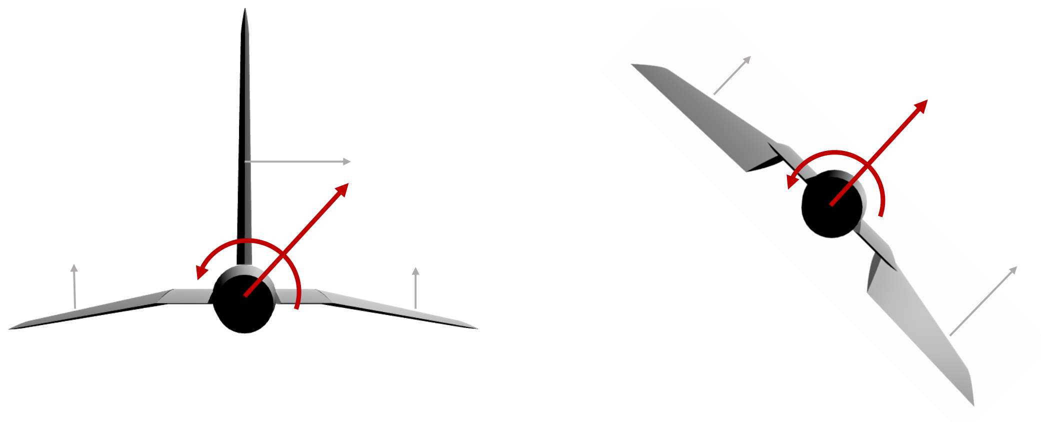
Second, coefficients representing cross-coupling (e.g. a longitudinal aerodynamic coefficient sensitivity with respect to a lateral parameter) are each modeled best, in terms of the form given in Eq. (4.43), by un-shifted sine waves. This makes them odd-functions of the BIRE rotation angle, usually passing through zero when the tail is horizontal. In addition to these observations, note that the coefficients that reach a maximum magnitude at $\delta_B = 45°$ are those that represent small trade-offs between longitudinal and lateral stability and control. In contrast, those that reach a maximum magnitude when vertical or horizontal dominate the stability and control characteristics of the aircraft. Lastly, it will again be emphasized that there are several coefficients that are not represented exactly by a sine wave. There remains much work to be done in terms of understanding these patterns and the physical mechanisms which control their variation with tail rotation angle.

CHAPTER 6

SIX-DEGREE-OF-FREEDOM STATIC TRIM

The aerodynamic models for the baseline aircraft and its BIRE variant defined in Chapters 4 and 5 allow for several studies to be performed comparing the two aircraft. This work will focus on three of these studies: a static trim investigation over various flight conditions, a comparison of the control authority of each aircraft using an attainable moment set analysis, and the effectiveness of a linear feedback controller on disturbance rejection. Additional analysis can be performed using the aerodynamic models presented in this work, but these three have been chosen to answer some fundamental questions about the effectiveness of a BIRE control system.

The fundamental questions that the static trim study in this chapter aims to answer is two-fold. First, does the longitudinal-lateral trade-off presented by the BIRE inhibit the trim envelope of the aircraft when compared to the trim envelope of the baseline aircraft? Intuitively, the BIRE should be able to produce nearly the same forces and moments as the baseline aircraft. As shown in Fig. 6.1, a traditional empennage generates individual forces and moments on the horizontal and vertical surfaces, shown in grey, that can be summed together to yield a net force and moment, shown in red. Using a combination of symmetric deflection, antisymmetric deflection, and tail rotation, the BIRE should be able to produce an equivalent net force and moment, except when large combinations of pitch and yaw moments are required simultaneously. Therefore, by leveraging an additional degree-of-freedom compared to that of a common horizontal tail, the BIRE is able to produce the lateral moments generally created by a vertical tail and rudder with only two aerodynamic lifting surfaces. The difference is that a traditional empennage can generate its maximum pitch and yaw moments independently of one another, while the BIRE may suffer reduced maximums due to their coupling.



Traditional Empennage

BIRE Design

Fig. 6.1: The BIRE design can create the same net force and moment (red) in many situations as a traditional empennage using the additional degree-of-freedom provided by rotation of the empennage.

To determine whether the intuition presented above holds up, two different trim conditions will be examined: a steady, coordinated turn and steady-heading sideslip. A steady, coordinated turn is one of the most basic trim conditions employed by aircraft and has the added benefit of being able to explore various aerodynamic loadings by changing the bank angle of trim [105]. Steady-heading sideslip is a trim state generally used in crosswind landings and is a valuable test of several lateral aerodynamic derivatives [106]. Each of these trim conditions requires various amounts of longitudinal-lateral coupling and are necessary flight conditions for any aircraft. Thus, an analysis using these two trim conditions will provide valuable information towards understanding the benefits and limitations to trim of using a rotating tail design.

The second fundamental question that can be answered using a static trim analysis concerns the increased chance of a tail strike posed by rotating the tail. Landing in a crosswind is one of the fundamental sizing constraints of yaw control mechanisms such as the rudder and is especially relevant for tailless aircraft without a rudder [107, 108]. The yawing moment necessary to balance the forces of a crosswind landing is substantial; therefore, if the BIRE rotation angle required to provide this moment is large enough, the design could be at a greater risk for tail strike than the baseline aircraft. Trim in a steady-heading sideslip condition will provide one way in which to analyze the risk of tail strike. Additional studies, such as landing simulations and landing using crab, are required to completely rule out the possibility of tail strike during a crosswind.

In general, the problem of static trim is to calculate the aerodynamic angles, rotation rates, and control surface deflections required to place the aircraft in equilibrium. For the two trim conditions examined in this work, the trim state of an aircraft is a function of the flight condition of the aircraft. In this case, the flight condition is specified by the altitude, H, velocity or Mach number, V or M, and the orientation of the aircraft, given by the bank angle ϕ and elevation angle, θ. Thus, to accurately represent the trim analysis provided here, it is important to understand the salient flight conditions at which the baseline aircraft operates.

6.1 Flight Conditions

Since the literature available detailing a control system such as the BIRE is limited, its relevant flight conditions are not well-defined. However, both Roetman et al. [6] and Dorsett and Mehl [7] presented tailless fighter aircraft designs and denoted the flight conditions they deemed most important to study. Therefore, the flight conditions analyzed in this work will be largely influenced by their choices.

Figure 6.2 shows an estimate for the flight envelope of a supersonic fighter aircraft based on that given by Conners and Sims [109]. Included in Fig. 6.2 are operational points of interest identified from the work of Roetman et al. and Dorsett and Mehl [6, 7]. The relevant flight condition information taken from Fig. 6.2, including altitude, velocity, Mach number, and Reynolds number, are tabulated in Table 6.1. Each of the flight conditions in Table 6.1 are also given a label to identify the purpose of the flight condition in assessing aircraft performance.

One may quickly note that neither transonic nor supersonic flight conditions are included in Fig. 6.1 and Table 6.1. Although the baseline aircraft is capable of flight in both of these regimes, the limitations on the baseline aerodynamic model necessitate a restriction to subsonic flight conditions with Mach number $M \leq 0.8$ for the compressibility corrections to be accurate. A higher-fidelity model is required to examine transonic and supersonic

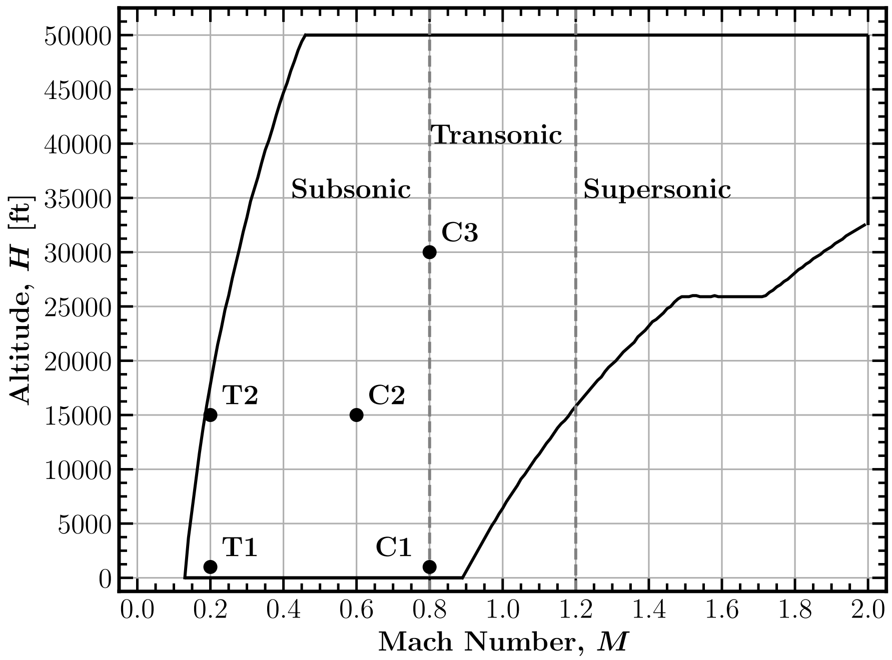


Fig. 6.2: Estimated flight envelope for the baseline aircraft with operational points of interest identified.

Table 6.1: Flight conditions considered in the static trim analysis.

Condition Label	**Altitude, [ft]**	**Velocity, [ft/s]**	**Mach Number**	**Reynold's Number**
T1	1,000	222	0.2	15,641,000
T2	15,000	201	0.19	9,919,000
C1	1,000	890	0.8	62,563,000
C2	15,000	634	0.6	31,324,000
C3	30,000	796	0.8	25,828,000

T1 – Takeoff and Approach
T2 – Power-On Departure Stall
C1 – Turbulent Penetration Speed
C2 – Air Combat Maneuver Condition
C3 – Maximum Sustained Load Factor

flight conditions, which are important for understanding the effects of extensive compressibility on aircraft trim. Nonetheless, the subsonic study in this work likely represents the majority of control-sizing cases for both the baseline aircraft and BIRE [107]. Thus, while studies in the transonic and supersonic regime are necessary in completely understanding the capabilities of the BIRE aircraft, the subsonic study provided in this work provides key information for determining the viability of the BIRE aircraft in terms of trim and control along a variety of flight conditions.

Each of the above flight conditions has a particular purpose in testing the static trim capabilities of the baseline and BIRE aircraft. The takeoff and approach flight condition (T1) is the flight condition at which most takeoff and landing is performed. Therefore, trim results at flight condition T1 are essential to understanding the impacts of BIRE rotation on trimmed flight in a crosswind. Increasing altitude to 15,000 ft while maintaining a nearly-constant Mach number from the takeoff and approach condition leads to the power-on departure stall condition (T2). Static analysis at this condition allows for control properties to be analyzed at higher angles of attack and also allows for an analysis of landing at airstrips at higher altitudes. These two conditions represent the takeoff and landing conditions of interest presented in this work.

There are three cruise flight conditions given in Fig. 6.2 and Table 6.1. These conditions include the turbulent penetration speed (C1), the air combat maneuver condition (C2), and the maximum sustained load factor (C3). Flight at the turbulent penetration speed represents the maximum speed at which the aircraft should be flown in the presence of turbulence [110]. Thus, this condition represents an upper limit on the subsonic flight regime for low-altitude flight.

The air combat maneuver condition represents the condition at which the limit load can be imposed by gusts or full deflection of the control surfaces without damage to the aircraft [103]. It too represents an upper limit on airspeed in the subsonic regime, but occurs in mid-altitude flight. Finally, the maximum sustained load factor condition represents the point at which the aircraft can be expected to maintain high-loads for longer periods of time. The baseline aircraft is rated for a positive load limit of 9-g's according to Fox and Forrest [62]. At this condition we can test whether that represents the maximum sustained load factor for the baseline aircraft and also determine whether the BIRE can achieve similar loading levels.

Each of the flight conditions in Table 6.1 and Fig. 6.2 will be explored in this work by trimming the aircraft across a range of velocities and the three altitudes presented. The flight conditions identified here are not particular to the baseline aircraft and are only estimates. However, it is reasonable to expect that these conditions are still appropriate for comparison between the baseline and BIRE aircraft.

6.2 Procedure for Finding the Trim State at a Given Flight Condition

The equations of motion for a rigid-body aircraft were given in Eqs. (4.1)–(4.2). In a trim state, the equations of motion must be satisfied such that the body-fixed translational velocities and rotation rates do not change with time. This requires that the left-hand side of Eqs. (4.1) and (4.2) are zero. Additionally, there must be no changes in the bank and elevation angles with time. Therefore, the first two equations within the system of equations given in Eq. (4.4) are zero. Applying these constraints, Eqs. (4.1), (4.2), and (4.4) can be rearranged to yield the trim equations of motion

$$\begin{Bmatrix} F_{x_b} \\ F_{y_b} \\ F_{z_b} \end{Bmatrix} = -W \begin{Bmatrix} -s_\theta \\ s_\phi c_\theta \\ c_\phi c_\theta \end{Bmatrix} - \frac{W}{g} \begin{Bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{Bmatrix} \tag{6.1}$$

$$\begin{Bmatrix} M_{x_b} \\ M_{y_b} \\ M_{z_b} \end{Bmatrix} = - \begin{bmatrix} 0 & -h_z & h_y \\ h_z & 0 & -h_x \\ -h_y & h_x & 0 \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} - \begin{Bmatrix} (I_{yy} - I_{zz})qr + I_{yz}(q^2 - r^2) + I_{xz}pq - I_{xy}pr \\ (I_{zz} - I_{xx})pr + I_{xz}(r^2 - p^2) + I_{xy}qr - I_{yz}pq \\ (I_{xx} - I_{yy})pq + I_{xy}(p^2 - q^2) + I_{yz}pr - I_{xz}qr \end{Bmatrix} \tag{6.2}$$

$$p = -(qs_\phi + rc_\phi)t_\theta \tag{6.3}$$

and

$$q = rt_\phi \tag{6.4}$$

Equations (6.1)–(6.4) represent the core system of eight equations for trim and must be satisfied for an aircraft in equilibrium.

Given mass, propulsion, gyroscopic, and aerodynamic information for an aircraft, the unknowns in these equations include the bank angle, ϕ, elevation angle, θ, roll rate, p, pitch rate, q, and yaw rate, r. The aerodynamic models can be used in Eqs. (4.5) and (4.6) to define the aerodynamic coefficients in terms of five additional unknowns: the angle of attack, α, sideslip angle, β, stabilator deflection, δ_e, aileron deflection, δ_a, and rudder deflection, δ_r, or BIRE rotation angle, δ_B. Thus, when considering the aerodynamic forces and moments, the eight equations given in Eqs. (6.1)–(6.4) are under-determined when compared to the ten unknowns present.

The aerodynamic forces and moments only constitute part of the pseudo-aerodynamic forces and moments on the left-hand side of Eqs. (6.1) and (6.2). Equations (4.5) and (4.6) show that the forces and moments generated by the propulsive forces of the aircraft must also be considered. To completely define these pseudo-aerodynamic forces, including the effects of thrust, in each body-fixed direction, a model for the propulsive force in the body-fixed coordinate system is required. Again, the wind tunnel data provided by Nguyen et al. [64] is invaluable here, as the authors included data for the engine of the baseline aircraft as a function of both Mach number and altitude. Using this data, a thrust model can be developed to determine the propulsive forces and moments produced by the aircraft as given in Eqs. (4.5) and (4.6).

6.2.1 Thrust Model

Assuming that the engine is mounted along the centerline of the aircraft, the only force that it will produce will be in the body-fixed x-direction and there will be no corresponding propulsive moments developed. Therefore, $F_{P_y} = F_{P_z} = M_{P_x} = M_{P_y} = M_{P_z} = 0$ and only F_{P_x} needs to be considered. The thrust produced by the engine will be modeled as a quadratic with respect to airspeed and a power function with respect to density [111]. This model allows the thrust to vary with both airspeed and altitude and follows the trends given in the data provided by Nguyen et al. [64]. Thus, the model of the thrust is given by

$$T = \left(\frac{\rho}{\rho_0}\right)^a \left(T_0 + T_1 V + T_2 V^2\right) \tag{6.5}$$

where ρ is the density at the altitude H, ρ_0 is the density at sea level, and a, T_0, T_1, and T_2 are constants determined by the thrust profile of the engine.

The thrust data provided by Nguyen et al. [64] is given as a function of Mach number and altitude at three thrust levels: idle, military, and maximum. When flying the aircraft, the pilot controls a throttle setting, τ, which determines how the engine produces its thrust. The relationship between the throttle setting and the power delivered by the engine, denoted P_1 in units of percent-power, is given in a table by Nguyen et al. [64]. Stevens and Lewis [69] provide the engine power P_1 as a function of throttle setting τ in equation form as

$$P_1 = \begin{cases} 64.94\tau & ,\tau \leq 0.77 \\ 217.38\tau - 117.38 & ,\tau > 0.77 \end{cases} \tag{6.6}$$

The engine power can then be converted to total thrust through the relationship [64, 69]

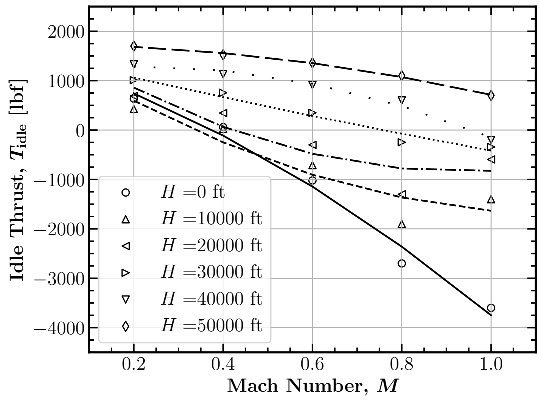
$$F_{P_x} = \begin{cases} T_{\text{idle}} + (T_{\text{mil}} - T_{\text{idle}})\,\frac{P_1}{50} & ,P_1 < 50 \\ T_{\text{mil}} + (T_{\text{max}} - T_{\text{mil}})\,\frac{P_1 - 50}{50} & ,P_1 \geq 50 \end{cases} \tag{6.7}$$

What remains is to provide a description for the thrust settings, T_{idle}, T_{mil}, and T_{max}, as a function of altitude. Fits of the form given in Eq. (6.5) were made at every altitude for each thrust level and are shown in Fig. 6.3. Each thrust model coefficient given in Eq. (6.5) is plotted as a function of altitude in Fig. 6.4 for each thrust setting. The data for all thrust settings in Fig. 6.4 can be well-described as a quadratic. A quadratic fit denoted by

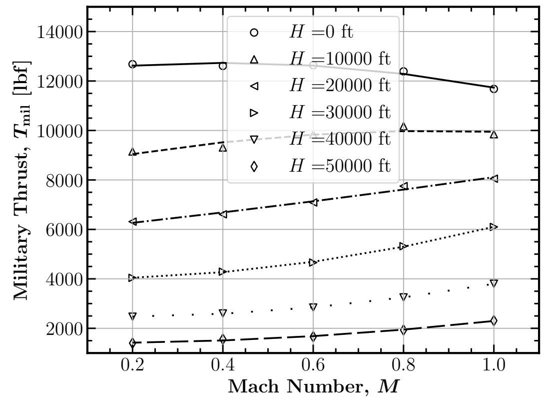
$$T_i = c_0 + c_H H + c_{H^2} H^2 \tag{6.8}$$

where T_i represents each of the thrust model coefficients and c_0, c_H, and c_{H^2} represent the quadratic fit coefficients for each thrust model parameter, was used to fit each thrust setting as a function of altitude.

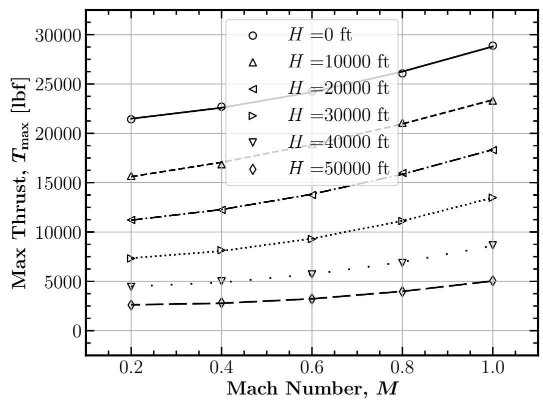
The values of c_0, c_H, and c_{H^2} for each of the thrust model coefficients are given in Table 6.2 at idle, military, and maximum thrust settings. Section C.4 of Appendix C shows



(a) Idle Thrust



(b) Military Thrust



(c) Maximum Thrust

Fig. 6.3: Thrust fits according to Eq. (6.5) for three thrust levels at six altitudes.

the code used to evaluate these coefficients of the thrust model. With this information, the propulsive thrust in the body-fixed x-direction is calculated as follows. First, the throttle setting is used in Eq. (6.6) to determine the power delivered to the engine. The percent-power delivered to the engine, P_1, is then used to determine the thrust delivered according to Eq. (6.7), with T_{idle}, T_{mil}, and T_{max} determined based on altitude using the fit coefficients in Table 6.2 in Eq. (6.8).

The thrust model therefore contributes one more unknown in the trim equation, the throttle setting, τ, assuming that the altitude is provided. With eleven unknowns and only eight independent equations, additional information is required to guarantee a closed system with a single solution. Two of the required equations are provided by specifying information on the orientation of the aircraft directly. That is, the orientation can be given

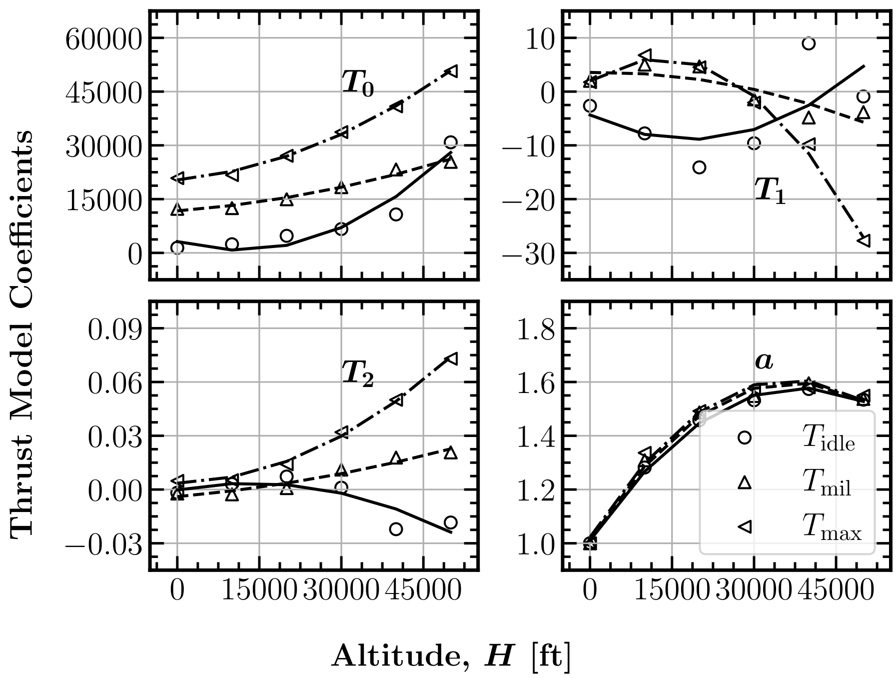


Fig. 6.4: Thrust coefficients and their fits as a function of altitude for three thrust levels.

Table 6.2: Thrust model coefficient fits as a function of altitude.

Parameter		T_{idle} Fit	T_{mil} Fit	T_{max} Fit
T_0	c_0	3145	11716	20341
	c_H	-0.4185	0.1156	0.1454
	$c_{H^2} \times 10^5$	1.8313	0.3474	0.9283
T_1	c_0	-4.3491	3.5689	1.9886
	$c_H \times 10^4$	-4.9703	0.1409	6.3926
	$c_{H^2} \times 10^8$	1.3557	-0.3982	-2.4428
T_2	$c_0 \times 10^3$	-0.2321	-3.9793	3.5201
	$c_H \times 10^7$	5.5629	2.6931	0.7574
	$c_{H^2} \times 10^{11}$	-2.0550	0.5281	2.6665
a	c_0	1.0104	1.0148	1.0225
	$c_H \times 10^5$	2.9484	3.1355	3.1984
	$c_{H^2} \times 10^{10}$	-3.8270	-4.2106	-4.3617

in terms of the elevation angle or climb angle and the bank angle or normal load factor. These two options will be explored in detail in the following subsection.

6.2.2 Specifying Aircraft Orientation

It is often more convenient for a pilot to specify the orientation of an aircraft in terms of the climb rate V_c or climb angle γ rather than an elevation angle θ. Sometimes it is also convenient to specify the load factor instead of the bank angle, since certain load factors determine the design of an aircraft. These two sets of parameters are related: the climb angle to the elevation angle and the bank angle to the load factor. Therefore, it is only a matter of preference for which of these parameters are specified to the trim algorithm. The trim algorithm given in this work will require the user to input the climb angle and bank angle. Given a climb angle and bank angle, the associated elevation angle and load factor can be computed as follows.

Elevation Angle for a Given Climb Angle

The climb rate is defined as the change in vertical location with respect to time, i.e. $V_c \equiv -\dot{z}_f$, and the climb angle is related to the climb rate according to

$$V_c = V s_\gamma = -\dot{z}_f \tag{6.9}$$

The climb rate can be related to the aircraft orientation and velocity components using the third equation within Eq. (4.3)

$$\dot{z}_f = -s_\theta u + s_\phi c_\theta v + c_\phi c_\theta w \tag{6.10}$$

Using Eq. (6.9) in Eq. (6.10) gives a relationship between the climb angle, bank angle, elevation angle, and body-fixed velocity components

$$V s_\gamma = u s_\theta - (v s_\phi + w c_\phi) c_\theta \tag{6.11}$$

Equation 6.11 can be rearranged and solved for the elevation angle using the quadratic formula to yield

$$s_\theta = \frac{uVs_\gamma \pm (vs_\phi + wc_\phi)\sqrt{u^2 + (vs_\phi + wc_\phi)^2 - V^2 s_\gamma^2}}{u^2 + (vs_\phi + wc_\phi)^2} \tag{6.12}$$

Note that solving a quadratic always yields two solutions, which may be mathematically valid but physically inconsistent. The physically-consistent elevation angle is the root that satisfies Eq. (6.11). Therefore, provided that the aerodynamic velocity components of the aircraft are known, Eq. (6.12) can be used to solve for a pair of elevation angles and Eq. (6.11) can be used to determine which is physically consistent with the climb angle specified.

Normal Load Factor for a Given Bank Angle

The term *load factor* is nearly universally defined as the ratio of lift to weight, i.e. L/W [112, 113]. Note that this is the ratio of the aerodynamic force (Eq. (4.24)) perpendicular to the direction of flight to the aircraft weight. However, in application and discussion, it is treated nearly universally as the ratio of pseudo-aerodynamic force (Eq. (4.5)) in the lift direction to the weight. This is an important difference, since the pseudo-aerodynamic force includes thrust, whereas the lift is the aerodynamic force without thrust. Therefore, the normal load factor is more specifically defined as

$$n \equiv \frac{-F_{z_s}}{W} = \frac{-F_{z_w}}{W} = \frac{-F_{z_b}c_\alpha + F_{x_b}s_\alpha}{W} \tag{6.13}$$

In a trim condition, the load factor is related to the bank angle through the third equation in Eq. (4.4). Using the first and third equations from Eq. (4.4) in Eq. (6.13) gives the relationship

$$n = \left[c_\theta c_\phi + (qu - pv)/g\right] c_\alpha + \left[s_\theta - (rv - qw)/g\right] s_\alpha \tag{6.14}$$

For a given bank angle and trim solution, the load factor at the trim condition can be computed from either Eq. (6.13) or (6.14).

By specifying the climb angle and bank angle, thus removing the elevation angle and bank angle from the list of unknowns, only one equation remains to completely close the trim system of equations. This final equation comes from the choice in trim condition itself. That is, the steady, coordinated turn and steady-heading sideslip conditions each prescribe an additional trim equation by definition. Therefore, the steady, coordinated turn and steady-heading sideslip trim conditions are defined as follows.

6.2.3 Steady, Coordinated Turn

In a steady, coordinated turn, the side force due to gravity and the bank angle perfectly balance the side force produced by rotational velocities [112]. Therefore, the aerodynamic side force on the vehicle is zero, i.e. $F_{y_b} = 0$, since there is no contribution from the propulsion system in this direction. Referring to the second equation from Eq. (6.1), this restriction on the side force requires

$$g s_\phi c_\theta = ru - pw \tag{6.15}$$

Thus, from the definition of the steady, coordinated turn, Eq. (6.15) constitutes the closing constraint on the trim system of equations.

Combining Eqs. (6.3), (6.4), and (6.15) gives three equations that can be solved for the rotation rates in the steady-coordinated turn as a function of the body-fixed velocities and aircraft orientation. These equations are written as

$$\begin{Bmatrix} p \\ q \\ r \end{Bmatrix} = \frac{g s_\phi c_\theta}{u c_\theta c_\phi + w s_\theta} \begin{Bmatrix} -s_\theta \\ s_\phi c_\theta \\ c_\phi c_\theta \end{Bmatrix} \tag{6.16}$$

Equations (6.1), (6.2), and (6.16) comprise a full set of nine equations and nine unknowns (α, β, p, q, r, δ_a, δ_e, δ_r or δ_B, and τ) for a steady-coordinated turn. In this work, an additional two equations with satisfied unknowns are given by specifying the climb angle and bank angle.

6.2.4 Steady-Heading Sideslip

Steady-heading sideslip is a trim condition in which the aircraft maintains its heading, ψ, sideslip angle, β, airspeed, and altitude [106]. This trim condition can be solved by specifying either the sideslip angle or bank angle. However, once one of these angles is specified, the other becomes a dependent variable and is fixed. For this work, we will examine only the case where the bank angle is specified. Examining the last equation in Eq. (4.4), we can see that, for any condition besides steady, level flight, a steady heading angle requires that the aircraft has no rotational velocity. Therefore, the constraint which closes our system of equations in steady, heading sideslip is given by

$$\begin{Bmatrix} p \\ q \\ r \end{Bmatrix} = 0 \tag{6.17}$$

Equations (6.1), (6.2), and (6.17) comprise our full set of nine equations for steady-heading sideslip.

6.2.5 Trim Algorithm

To solve the non-linear trim system of equations for its nine unknowns, an iterative trim algorithm can be implemented. Here we consider the case when the flight condition is specified by a freestream velocity, altitude, bank angle, and climb angle. With the freestream velocity, altitude, bank angle, and climb angle specified, the following iterative algorithm can be employed to solve for the aerodynamic angles, body-fixed rotation rates, control surface deflections, and throttle setting.

(1) Begin with the initial guess of all aerodynamic angles and controls set to zero ($\alpha = \beta = \delta_a = \delta_e = \delta_r = \delta_B = \tau = 0$).

(2) Initialize the rotation rates to zero ($p = q = r = 0$).

(3) Calculate the body-fixed velocities from Eq. (4.18).

(4) Calculate the elevation angle using Eqs. (6.12) and (6.11).

(5) For the case of a steady-coordinated turn, use Eq. (6.16) to compute the rotation rates. These remain equal to zero in steady-heading sideslip.

(6) Use the compressibility-corrected aerodynamic model to find the aerodynamic angles, throttle setting, and control-surface deflections that satisfy Eqs. (6.1) and (6.2).

(7) Using the updated values for the aircraft orientation, aerodynamic angles, throttle setting, and control-surface deflections, repeat steps (3)–(6) until the solution converges below a desired tolerance.

Note that there are several differences here between the algorithm for the baseline aircraft and the BIRE variant. The first is that the states initialized in step (1) differ by specifying the rudder deflection δ_r for the baseline aircraft and the BIRE rotation angle δ_B for the BIRE variant. Secondly, step (6) uses the aerodynamic model specified in Eqs. (4.60)–(4.65) for the baseline aircraft and Eqs. (4.66)–(4.71) for the BIRE variant. Finally, the inertial information for the BIRE aircraft differs from the baseline aircraft according to the information in Tables 3.4 and 3.8. Since the inertial components are functions of the BIRE rotation angle, the value of the inertia must be modified in step (6) according to the value of δ_B.

6.2.6 Solving For the States of the Aerodynamic Model

There are many ways in which to solve for the aerodynamic angles, throttle setting, and control surface deflections in step (6). Two simple options include a fixed-point iteration and a multi-variate Newton-Raphson method [100]. The fixed-point iteration method has linear convergence in comparison to quadratic convergence for the Newton-Raphson method. While the difference in convergence would suggest that the Newton-Raphson method is more efficient, in a multi-variate implementation the effect of an inverted matrix multiplication is required. Thus, in some cases the cost of such a calculation makes the Newton-Raphson method more computationally costly than a fixed-point iteration method.

In this section, both of these solution methods for step (6) will be given. In most situations, the Newton-Raphson method is preferable, so it will be treated first. One area where the Newton-Raphson method performs poorly is in the situation where multiple roots are present [100]. By referring back to Fig. 6.1, we can see that the BIRE likely has several configurations that will result in the same forces and moments as the traditional empennage. Thus, it is reasonable to assume that there are multiple configurations of the tail that will result in a trim state, the most trivial of which is that all BIRE rotation angles can be rotated by an additional 2π radians to result in the exact same configuration.

To counteract this, limits can be placed on the BIRE rotation angle returned by the trim algorithm; for example, restricting $-90^\circ \leq \delta_B \leq 90^\circ$. Certain levels of analysis also allow for the trim solution to be "soft-started" by replacing the initial guess of 0 in steps (1) and (2) with the solution of a similar trim condition. In particular, this approach will be used for most of the analysis performed in this chapter. Regardless, the fixed-point iteration method will be described first, followed by a description of the multi-variate Newton-Raphson method for solving step (6).

To use fixed-point iteration, each equation in Eqs. (6.1) and (6.2) is solved in succession for the unknown that is dominant in that particular equation. Each equation is dependant on each of the unknown aerodynamic angles, throttle setting and control surface deflections. However, especially in the case of the baseline aircraft, each of these parameters is predominantly used to control one particular aerodynamic force or moment acting on the aircraft. For example, the angle of attack, α, is the main source of changes in the lift acting on the aircraft. In this light, Table 6.3 shows the presumed dominant terms for each of the pseudo-aerodynamic forces and moments.

Given the values of the aerodynamic angles, throttle setting, and control surface deflections at any iteration i, improved estimates for the aerodynamic parameters can be obtained

Table 6.3: Dominant terms in the pseudo-aerodynamic forces and moments.

Pseudo-Aerodynamic Force/Moment	**Dominant Term**
F_{x_b}	τ
F_{y_b}	β
F_{z_b}	α
M_{x_b}	δ_a
M_{y_b}	δ_e
M_{z_b}	δ_r or δ_B

from

$$\tau_{i+1} = \tau_i - \lambda \left[\frac{F_{x_b} - W s_\theta + (rv - qw)W/g}{F_{P_x}} \right] \tag{6.18}$$

where F_{P_x} is defined using Eq. (6.7),

$$\alpha_{i+1} = \alpha_i + \lambda \left[\frac{F_{z_b} + W c_\phi c_\theta + (qu - pv)W/g}{\frac{1}{2}\rho V^2 S_w C_{L,\alpha} c_\alpha} \right] \tag{6.19}$$

$$\beta_{i+1} = \beta_i - \lambda \left[\frac{F_{y_b} + W s_\phi c_\theta + (pw - ru)W/g}{\frac{1}{2}\rho V^2 S_w C_{S,\beta} c_\beta} \right] \tag{6.20}$$

$$\delta_{a_{i+1}} = \delta_{a_i} - \lambda \left[\frac{M_{x_b} - h_z q + h_y r + (I_{yy} - I_{zz})qr + I_{yz}(q^2 - r^2) + I_{xz}pq - I_{xy}pr}{\frac{1}{2}\rho V^2 S_w b_w C_{\ell,\delta_a}} \right] \tag{6.21}$$

$$\delta_{e_{i+1}} = \delta_{e_i} - \lambda \left[\frac{M_{y_b} + h_z p - h_x r + (I_{zz} - I_{xx})pr + I_{xz}(r^2 - p^2) + I_{xy}qr - I_{yz}pq}{\frac{1}{2}\rho V^2 S_w \bar{c}_w C_{m,\delta_e}} \right] \tag{6.22}$$

and

$$\delta_{r_{i+1}} = \delta_{r_i} - \lambda \left[\frac{M_{z_b} - h_y p + h_x q + (I_{xx} - I_{yy})pq + I_{xy}(p^2 - q^2) + I_{yz}pr - I_{xz}qr}{\frac{1}{2}\rho V^2 S_w b_w C_{n,\delta_r}} \right] \tag{6.23}$$

or

$$\delta_{B_{i+1}} = \delta_{B_i} - \lambda \left[\frac{M_{z_b} - h_y p + h_x q + (I_{xx} - I_{yy})pq + I_{xy}(p^2 - q^2) + I_{yz}pr - I_{xz}qr}{\frac{1}{2}\rho V^2 S_w b_w C_{n,\delta_B}} \right] \tag{6.24}$$

depending on if the aircraft being trimmed is the baseline or BIRE aircraft, respectively. In Eqs. (6.18)–(6.24), the term λ is a relaxation factor between 0 and 1 that allows for an additional level of convergence control. This is another way in which the problem of multiple roots in the BIRE aircraft can be addressed in the trim algorithm. The right-hand side of each of Eqs. (6.18)–(6.24) is computed using the current estimate for the set of unknown aerodynamic parameters.

For the BIRE rotation angle in Eq. (6.24), the value of C_{n,δ_B} can be evaluated in many ways. With an analytical solution given in Eq. (4.71) and the form of the sensitivity coefficients provided by Eq. (4.43), C_{n,δ_B} can be evaluated analytically. It can also easily be computed using a finite difference [100]. Regardless, Eqs. (6.18)–(6.24) can be used in the iterative solution process discussed above to solve for the aerodynamic angles, throttle setting, and control-surface deflections needed in Step (6).

A multi-variate Newton-Raphson method can be developed to solve step (6) in the trim algorithm as follows. The residual, $\mathbf{R}$, can be expressed as a vector of the unknown aerodynamic parameters using Eqs. (6.1) and (6.2) as

$$\mathbf{R} \equiv f(\mathbf{G}) = \begin{bmatrix} F_{x_b} - W s_\theta + (rv - qw)W/g \\ F_{y_b} + W s_\phi c_\theta + (pw - ru)W/g \\ F_{z_b} + W c_\phi c_\theta + (qu - pv)W/g \\ M_{x_b} - h_z q + h_y r + (I_{yy} - I_{zz})qr + I_{yz}(q^2 - r^2) + I_{xz}pq - I_{xy}pr \\ M_{y_b} + h_z p - h_x r + (I_{zz} - I_{xx})pr + I_{xz}(r^2 - p^2) + I_{xy}qr - I_{yz}pq \\ M_{z_b} - h_y p + h_x q + (I_{xx} - I_{yy})pq + I_{xy}(p^2 - q^2) + I_{yz}pr - I_{xz}qr \end{bmatrix} \tag{6.25}$$

For the baseline aircraft, $\mathbf{G} = \{\alpha, \beta, \delta_a, \delta_e, \delta_r, \tau\}$ and for the BIRE, $\mathbf{G} = \{\alpha, \beta, \delta_a, \delta_e, \delta_B, \tau\}$.

The residual represents the difference between the left- and right-hand sides of the trim equations of motion in Eqs. (6.1) and (6.2). Thus, to satisfy those equations, the residual must be driven to zero. A linear Taylor series expansion of the residual about $\mathbf{G}$ is given as

$$f(\mathbf{G} + \Delta\mathbf{G}) \approx f(\mathbf{G}) + \mathbf{J}(\mathbf{G})\Delta\mathbf{G} \tag{6.26}$$

where $\mathbf{J}$ is the Jacobian of the residual and is defined as

$$\mathbf{J} = \frac{\partial f_i}{\partial G_j} \tag{6.27}$$

Again, the Jacobian of the residual can be calculated using analytical methods on the aerodynamic models presented in Chapter 4 or by using a finite difference routine.

Recall that $\mathbf{R} \equiv f(\mathbf{G})$ and therefore Eq. (6.26) should be driven to zero to satisfy the trim equations of motion. To this end, the Newton step, $\Delta\mathbf{G}$, can be calculated

$$\Delta\mathbf{G} = -\mathbf{J}^{-1}\mathbf{R} \tag{6.28}$$

which can also be relaxed at each iteration to provide additional control over convergence. The final iterative scheme for solving Step (6) is given with the relaxation factor λ as

$$\mathbf{G}_{i+1} = \mathbf{G}_i + \lambda\Delta\mathbf{G} \tag{6.29}$$

The solution can be iterated upon until the residual (i.e. the trim equations of motion) converges to zero.

Note that the solution method in Eq. (6.29) requires that the Jacobian of the residual be invertible. Therefore, in cases where gradients are very large or very small, the Newton-Raphson method may not be able to produce a trim solution. Adjusting the relaxation factor can help with this, though this often-times has to be addressed on a case-by-case basis. Using either fixed-point iteration or the Newton-Raphson method, the aerodynamic parameters in step (6) can be determined and a convergent trim solution for the nine unknowns can be given.

6.2.7 Example Trim Cases

Consider the baseline aircraft at the takeoff and approach flight condition (T1 in Table 6.1) with a climb angle of 10 degrees and a bank angle of 6.5 degrees. The baseline and BIRE aircraft are trimmed in a steady-heading sideslip trim condition with a relaxation factor of $\Gamma = 0.5$ using Newton's method. This case was chosen because places the baseline aircraft in a condition with nearly maximum rudder deflection ($\delta_r = 30°$). Results for the trim parameters at this flight condition in steady-heading sideslip are shown in Table 6.4 for the baseline aircraft and 6.5 for the BIRE. Note that, in this example, the BIRE rotation angle required to trim the BIRE variant is much larger than the required rudder of the baseline aircraft shown in Table 6.4. These trends will be examined in more detail with the static trim analysis that follows, but is likely due to the low speed of the and the relatively large climb angle.

Parameter Description	Trim Parameter	Trim Value
Elevation Angle, [deg]	θ	26.8922
Bank Angle, [deg]	ϕ	6.5
Angle of Attack, [deg]	α	15.6506
Sideslip Angle, [deg]	β	10.9877
Roll Rate, [deg/s]	p	0
Pitch Rate, [deg/s]	q	0
Yaw Rate, [deg/s]	r	0
Aileron Deflection, [deg]	δ_a	-1.2742
Stabilator Deflection, [deg]	δ_e	2.9090
Rudder Deflection, [deg]	δ_r	29.9305
Throttle Setting	τ	0.3992
Thrust, [lbf]	T	7264.8914
Load Factor	n	0.9753
Iterations	–	45

Table 6.4: Example steady-heading sideslip trim solution for the baseline aircraft.

At the same condition, but in a steady, coordinated turn, Tables 6.6 and 6.7 give the trim state of the baseline aircraft and BIRE, respectively. The differences in this condition between the baseline aircraft and the BIRE are very minor and mostly due to differences in geometry between the two aircraft.

6.3 Shifting the Center of Gravity

The trim results shown to this point have each considered the center of gravity to be located at its nominal position. Aircraft often denote aft and forward limits on the

Parameter Description	Trim Parameter	Trim Value
Elevation Angle, [deg]	θ	27.7915
Bank Angle, [deg]	ϕ	6.5
Angle of Attack, [deg]	α	16.5807
Sideslip Angle, [deg]	β	10.8665
Roll Rate, [deg/s]	p	0
Pitch Rate, [deg/s]	q	0
Yaw Rate, [deg/s]	r	0
Aileron Deflection, [deg]	δ_a	-3.5872
Stabilator Deflection, [deg]	δ_e^B	14.9963
BIRE Rotation, [deg]	δ_B	67.5351
Throttle Setting	τ	0.4756
Thrust, [lbf]	T	8333.2010
Load Factor	n_a	0.9755
Iterations	–	51

Table 6.5: Example steady-heading sideslip trim solution for the BIRE aircraft.

Parameter Description	Trim Parameter	Trim Value
Elevation Angle, [deg]	θ	25.8268
Bank Angle, [deg]	ϕ	6.5
Angle of Attack, [deg]	α	15.9181
Sideslip Angle, [deg]	β	0.0100
Roll Rate, [deg/s]	p	-0.3754
Pitch Rate, [deg/s]	q	0.0878
Yaw Rate, [deg/s]	r	0.7706
Aileron Deflection, [deg]	δ_a	0.1778
Stabilator Deflection, [deg]	δ_e	2.9497
Rudder Deflection, [deg]	δ_r	-0.0885
Throttle Setting	τ	0.3928
Thrust, [lbf]	T	7176.3606
Load Factor	n	0.9901
Iterations	–	45

Table 6.6: Example steady, coordinated turn trim solution for the baseline aircraft.

center of gravity location for performance reasons. For example, Nguyen et al. [64] and Clayton et al. [114] both indicate that the baseline aircraft may have an aft center of gravity limit located at around 40% of the mean aerodynamic chord. In addition, the baseline aircraft is part of a line of fighter aircraft with "relaxed" static-stability, meaning that the center of gravity is located such that the aircraft is intentionally destabilized to improve maneuverability [64]. The BIRE design is likely to be highly sensitive to changes

Parameter Description	Trim Parameter	Trim Value
Elevation Angle, [deg]	θ	26.8827
Bank Angle, [deg]	ϕ	6.5
Angle of Attack, [deg]	α	16.9793
Sideslip Angle, [deg]	β	0.0072
Roll Rate, [deg/s]	p	-0.3861
Pitch Rate, [deg/s]	q	0.0862
Yaw Rate, [deg/s]	r	0.7566
Aileron Deflection, [deg]	δ_a	0.1554
Stabilator Deflection, [deg]	δ_e^B	-3.1044
BIRE Rotation, [deg]	δ_B	0.2750
Throttle Setting	τ	0.3958
Thrust, [lbf]	T	7218.0537
Load Factor	n	0.9900
Iterations	–	46

Table 6.7: Example steady, coordinated turn trim solution for the BIRE aircraft.

in center of gravity location and thus the trim algorithm given previously must be modified slightly to allow for these changes.

Changes in the center of gravity have a substantial impact to the aerodynamic moments produced by the aerodynamic angles, body-fixed rotation rates, and control surface deflections. The body-fixed aerodynamic moments resulting from a change in center of gravity location given by $\Delta\text{cg} = \{\Delta x_{\text{cg}}, \Delta y_{\text{cg}}, \Delta z_{\text{cg}}\}$ are given as

$$M_{x_b} = \frac{1}{2}\rho V^2 S_w b_w C_\ell - F_{z_b}\Delta y_{\text{cg}} + F_{y_b}\Delta z_{\text{cg}} \tag{6.30}$$

$$M_{y_b} = \frac{1}{2}\rho V^2 S_w C_m \bar{c}_w - F_{x_b}\Delta z_{\text{cg}} + F_{z_b}\Delta x_{\text{cg}} \tag{6.31}$$

and

$$M_{z_b} = \frac{1}{2}\rho V^2 S_w b_w C_n - F_{y_b}\Delta x_{\text{cg}} + F_{x_b}\Delta y_{\text{cg}} \tag{6.32}$$

Thus, whenever the center of gravity is shifted, Eqs. (6.30)–(6.32) should be used in Eqs. (6.2), (6.18)–(6.24), and (6.25). In this study, changes in center of gravity location will be limited to shifts forward and aft in the body-fixed x-direction; that is, $\Delta y_b = \Delta z_b = 0$. Thus, whenever the center of gravity is shifted

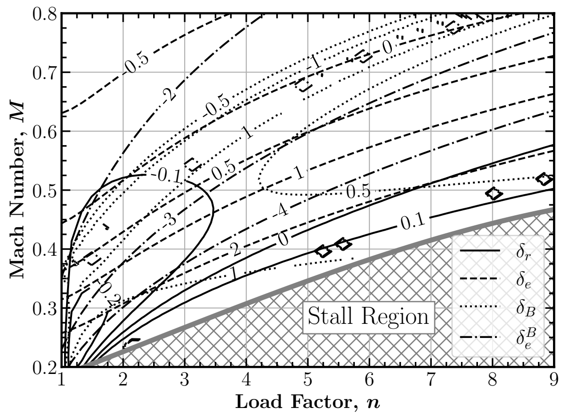
6.4 Static Trim Analysis

The results and analysis contained in this section will be split between the two trim conditions that have been described in Sections 6.2.3 and 6.2.4: a steady, coordinated turn and steady-heading sideslip. In the steady, coordinated turn analysis, each of the points in Fig. 6.2 and Table 6.1 will be examined across a range of load factors. The steady-heading sideslip analysis will be performed at the same altitudes and Mach numbers as the steady, coordinated turn analysis, but will instead look at how change in the trim bank angle affect the results. Trim in steady-heading sideslip is generally a condition used while landing an aircraft. Therefore, an altitude of 1,000 ft is a good candidate for this type of analysis and an altitude of 15,000 ft roughly matches the altitude of the highest-altitude airport in the world [115]. Although the steady-heading sideslip trim condition is generally used only in landing scenarios, there are situations, such as mid-air refueling, that require an aircraft to maintain its heading in sideslip. Thus, the altitude of the maximum sustained load flight condition (C3) will also be examined in the steady-heading sideslip trim study.

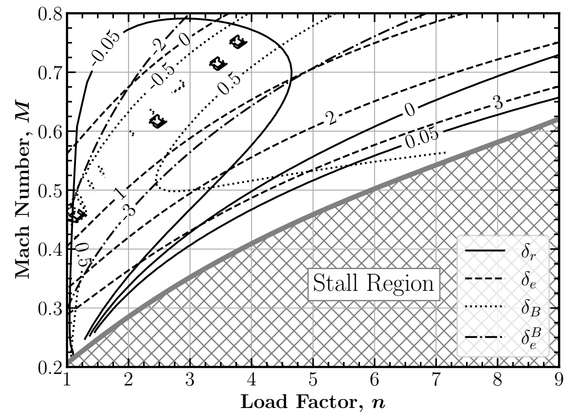
6.4.1 Steady, Coordinated Turn Analysis

To produce various load factors across the range of Mach numbers and altitudes given in Fig. 6.2, the bank angle of each aircraft can be adjusted using the relationship in Eq. (6.14) to produce a given load factor. This required an optimization routine that would vary the bank angle, find the trim condition of each aircraft, and then converge to the given load factor. In this analysis, the climb angle, γ is set equal to zero. Code containing this process is included in Section C.4 of Appendix C.

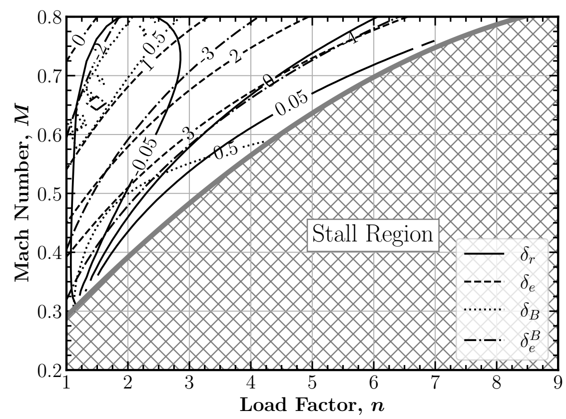
The results of the comparison between the trim condition of the baseline and BIRE aircraft are given in Fig. 6.5. Contours for the rudder and stabilator deflections of the baseline aircraft as well as the BIRE rotation angle and BIRE stabilator deflection angle (given a superscript B) are shown at each altitude given in Table 6.1. The stall region was calculated using a maximum lift coefficient of $C_{L_{\max}} = 1.9$ for the baseline aircraft as measured in Nguyen et al.'s wind tunnel results [64]. As the lift was determined across given load factors, any region where the lift coefficient exceeded the maximum lift coefficient was deemed to be in the stall region. As expected, we note that the stall region increases with altitude, due to the difference in the ambient dynamic pressure. Thus, higher Mach numbers are required by the aircraft to maintain trim as the altitude increases.



(a) Altitude : 1,000 ft



(b) Altitude : 15,000 ft



(c) Altitude : 30,000 ft

Fig. 6.5: Steady, coordinated turn analysis with the center of gravity at its nominal position.

The takeoff and approach condition (T1) is located at the bottom left corner of Fig. 6.5a. In general, an aircraft must takeoff and land be 10% higher than the stall speed for the aircraft [111]. In Fig. 6.5a, we can see that this trend is approximately held for both the baseline aircraft and the BIRE. Flight condition T2, which is the power-on departure stall condition is located at the bottom left corner of Fig. 6.5b. As expected, at this flight condition both aircraft will be very nearly stalled in steady, level flight ($n = 1$).

Flight condition C1, located at the turbulent penetration speed, is represented by a horizontal line at $M_\infty = 0.8$ in Fig. 6.5a. The magnitude of the deflection contours in this region of the figure are quite small and can vary rapidly at intermediate to larger load factors. In the presence of turbulence, deviations to the angle of attack and sideslip could easily require changes in deflections that rival those in this plot. Therefore, the analysis here seems to indicate that this does, indeed, represent a volatile limit in the each aircraft's flight envelope. In fact, the BIRE deflection angle is much more sensitive to deviations in this region than the rudder.

Lastly, there is a discrepancy between the general shape of the contours between the baseline aircraft and BIRE in Fig. 6.5a. The baseline aircraft has very gradual contours in both the Mach number and load factor directions. That is, a pilot does not need to change the rudder angle substantially to coordinate a turn at many different load factors. On the other hand, the magnitude of the BIRE rotation angles are slightly larger and have sharper gradients in both directions at low to intermediate Mach numbers. This indicates a sensitivity to small demands in lateral-directional control that are to be expected when coupling pitch and yaw. Fortunately, the magnitude of these deflections are still very small, so a pilot or on-board computer system could achieve these trim conditions with reasonably-sized actuators.

The air combat maneuver condition (C2), is located at $M_\infty = 0.6$ in Fig. 6.5b. From Fig. 6.5b, we note that both the BIRE and the baseline aircraft can produce large load factors with small control-surface deflections without much risk of stall. At this altitude, the difference in magnitude of deflection between the rudder and tail rotation are much more pronounced. It is worth pointing out, however, that this level of sensitivity requires much more precision for the baseline aircraft. Therefore, the larger magnitudes of deflection for the BIRE may be a benefit to the aircraft, at least when in trim state dominated by longitudinal control.

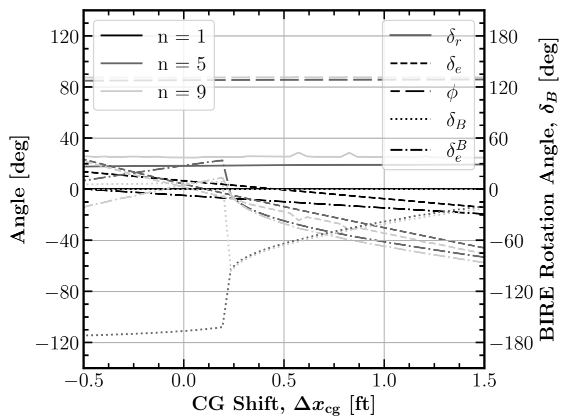
Finally, at $M_\infty = 0.8$ in Fig. 6.5c, we see that the maximum load factor that can be sustained by the baseline aircraft and BIRE variant is approximately $n = 8.5$, an 8.5-g

loading. This is reasonably consistent with the 9-g loading for which the baseline aircraft is structured [64]. Again, a similar level of magnitude difference between the rudder and BIRE rotation is measured here and the increase in dynamic pressure has squeezed the contours closer together when compared to 6.5b.

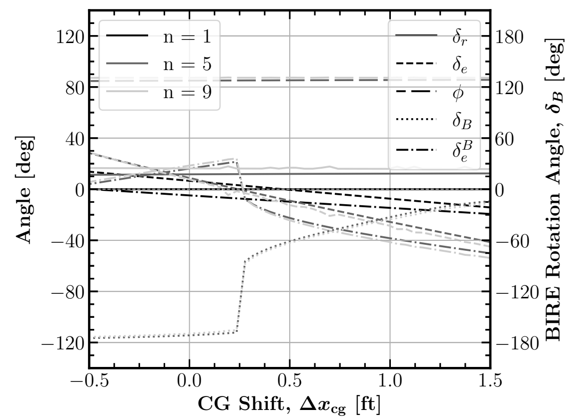
We note in Figs. 6.5a and 6.5b that there are regions where the contours are not smooth for deflection angles of the BIRE aircraft. These areas are regions where the trim algorithm is very sensitive and are generally located where the BIRE transitions from one direction of deflection to another. This is likely due to sensitivities in the aerodynamic model and can be caused by large gradients in δ_B. In addition, since each trim condition is soft-started using the previous trim state, changing the direction of the BIRE can cause the algorithm to over-step a trim state. Further improvements to the trim algorithm, such as moving to a higher-fidelity root-finding method, could improve these sensitivities.

One way in which the sensitivity to BIRE rotation angle may be reduced is by shifting the location of the center of gravity. By doing so, the load acting on the tail can be manipulated to ensure that it is consistently acting in the same direction. Figure 6.6 shows the baseline aircraft rudder and elevator deflection, BIRE rotation angle, and BIRE elevator deflection as a function of the change in CG location, Δx_{cg}, for various load factors. The data in Fig. 6.6 is all taken at the stall velocity for each given altitude and the ordinate on the left corresponds to all angles except the BIRE rotation angle, whose axis is on the right. Note that $\Delta x_{\text{cg}} = 0$ indicates that the aircraft is at its nominal CG location of 35% of the mean aerodynamic chord.

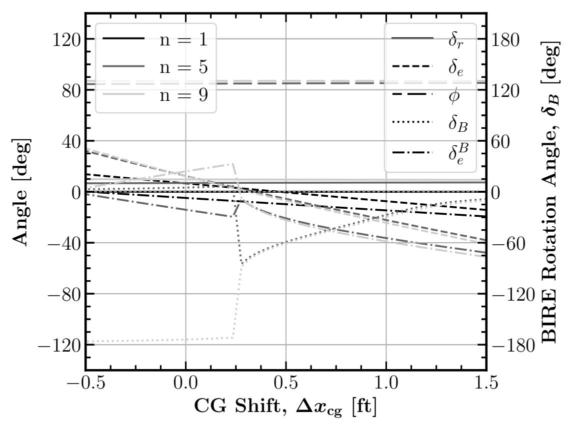
In examining Fig. 6.6, note that as the center of gravity moves forward, the deflections predicted by the BIRE become closer to those of the baseline aircraft. In addition, there is a position just forward of the nominal CG location where the BIRE deflection angle changes rapidly. To understand whether this was caused by multiple possible trim points or is a sensitivity to CG location unique to the BIRE control surface, each trim solution was soft-started with the solution obtained with a CG location immediately forward of the current CG location. Therefore, it can be reasonably assumed that the jump seen in each figure



(a) Altitude : 1,000 ft



(b) Altitude : 15,000 ft

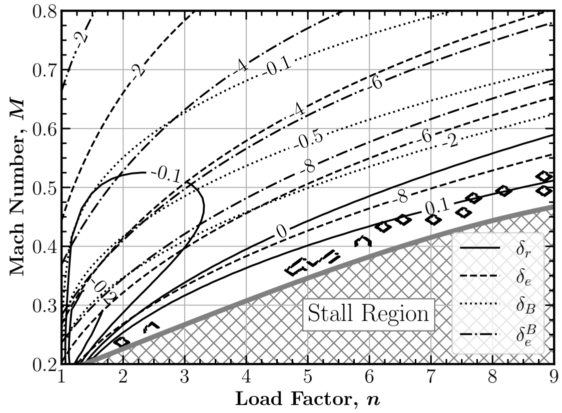


(c) Altitude : 30,000 ft

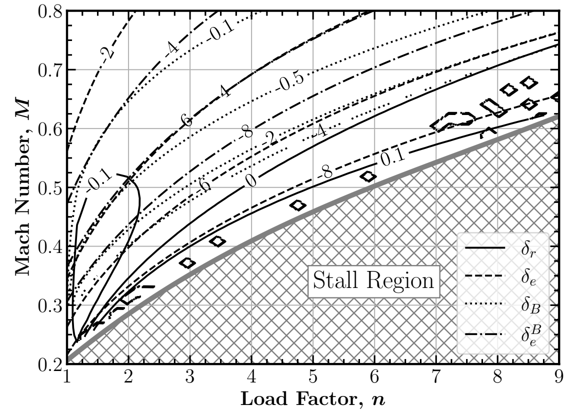
Fig. 6.6: Control surface deflections for the baseline aircraft and its BIRE variant as a function of center of gravity location at stall speed.

is a result of sensitivities of the BIRE to CG location rather than a issue with the trim algorithm. With a center of gravity shift of approximately one foot forward, the baseline aircraft and BIRE see similar stabilator deflection angles and are sufficiently far forward of the sharp change in BIRE rotation angle observed near the nominal position to ensure that these sensitivities are avoided. This location for the center of gravity can be further explored by repeating the study shown in Fig. 6.5 with the center of gravity relocated.

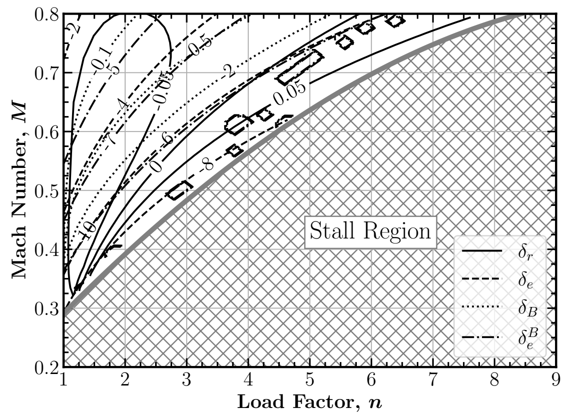
Moving the CG forward by one foot on each aircraft results in the Mach-versus-load factor plots shown in Fig. 6.7. Note that, comparing Fig. 6.5 to Fig. 6.7, the BIRE variant is much better behaved and even has trends similar to the stabilator deflections. Therefore,



(a) Altitude : 1,000 ft



(b) Altitude : 15,000 ft



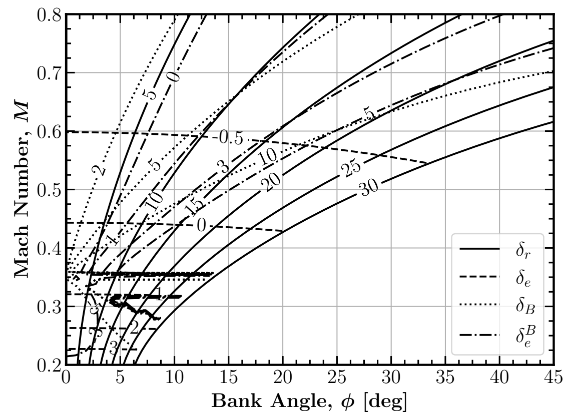
(c) Altitude : 30,000 ft

Fig. 6.7: Steady, coordinated turn analysis with the center of gravity moved forward.

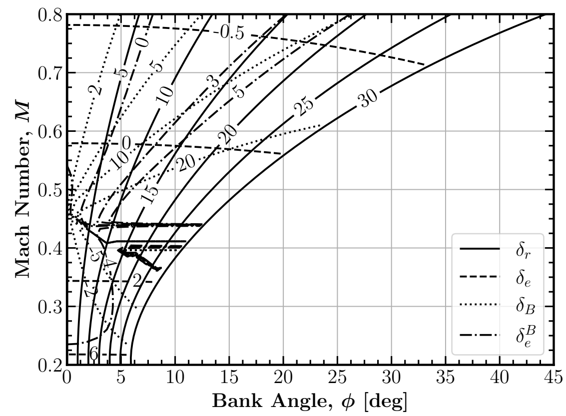
large changes in load factors are less likely to be experienced with small deviations of the BIRE rotation angle. In addition, the BIRE rotation angle does not change directions with the center of gravity moved forward, presumably because the tail always carries a negative load to trim. The benefits of moving the CG forward in this way will be explored further in the next section, which will focus on the steady-heading sideslip trim analysis.

6.4.2 Steady-Heading Sideslip Analysis

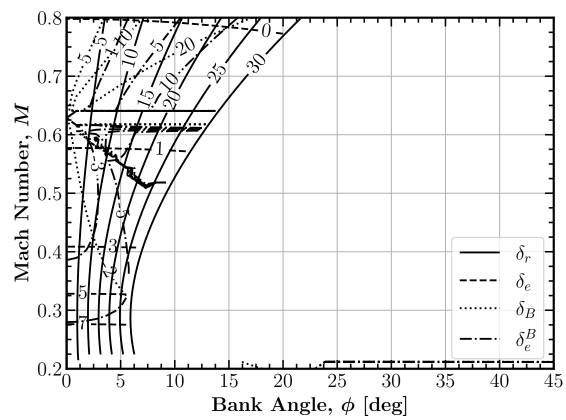
The steady-heading sideslip conditions of the baseline aircraft and the BIRE can be compared by first examining the control surface deflections required to maintain steady-heading sideslip across a range of velocities and bank angles. Again, the climb angle is set to zero for this entire analysis. Figure 6.8 shows the stabilator deflections for each aircraft, as well as the rudder deflection and BIRE rotation angle at 1,000, 15,000, and 30,000 ft altitude. Only data up to the maximum rudder deflection of the baseline aircraft is reported so as to better compare the trim capabilities of each aircraft.



(a) Altitude : 1,000 ft



(b) Altitude : 15,000 ft



(c) Altitude : 30,000 ft

Fig. 6.8: Steady-Heading sideslip analysis with the center of gravity at its nominal position.

One immediate benefit shown by the rotating tail in these figures is the direction of the gradients. The deflections of the rudder at each altitude have a strong gradient in the dimension of the bank angle and a smaller gradient in the Mach number dimension. In contrast, the BIRE rotation angles has nearly an equal gradient in each direction and these gradients are nearly independent of the altitude. Thus, it appears from Fig. 6.8

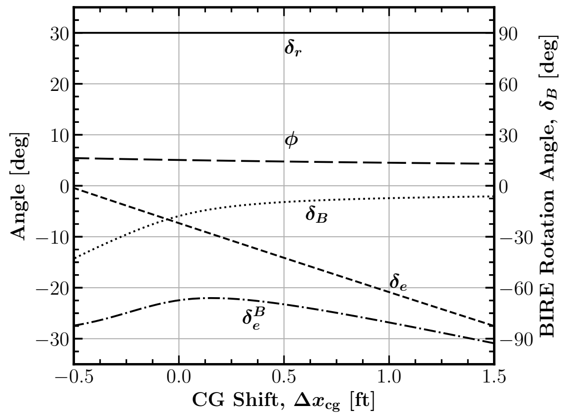
that the BIRE has a much larger trim envelope than the baseline aircraft in steady-heading sideslip. While certainly a promising result, the impacts of this increase in trim envelope are dependent on the aircraft requirements themselves, which is outside the scope of the current analysis.

Intuition indicates that in steady level flight ($\phi = 0$), the trends in stabilator deflection for both the BIRE and baseline aircraft would be similar. Steady level flight is an entirely longitudinal state, and therefore the rotating tail of the BIRE would be expected to be in the horizontal position ($\delta_B = 0$). At first, it may be hard to see this pattern, due to a large discontinuity near Mach 0.3, 0.4, and 0.6 in Figures 6.10a, 6.10b, and 6.10c, respectively. However, examining either above or below the discontinuity, it can be noted that the BIRE rotation angle approaches zero and the stabilator deflections of the BIRE are approaching a horizontal state. Thus, in steady-level flight, the BIRE acts exactly the same as a traditional stabilator configuration, as expected.

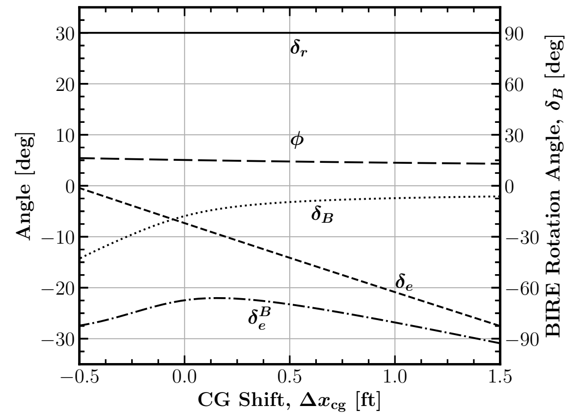
The large discontinuity that exists at moderate Mach numbers for the BIRE aircraft in Fig. 6.8 is again a result of the location of the center of gravity. That is, the load required to trim the aircraft switches direction and thus the horizontal tail must rapidly rotate to a new orientation. This can be investigated in a similar manner as was performed when examining the influence of center of gravity location in a steady, coordinated turn.

The maximum crosswind landing condition that can be maintained by the baseline aircraft is limited by its maximum rudder deflection. Since this is a crucial landing condition, the effect of changes in the position of the center of gravity will be studied at the stall speed and with a bank angle that corresponds to the maximum rudder deflection. This ensures that the baseline aircraft is in its maximum crosswind landing position. Figure 6.9 shows the rudder deflection, bank angle, BIRE rotation angle, and stabilator deflections of both aircraft as a function of the change in CG location, Δx_{cg}. Again note that all angles except for the BIRE rotation angle are plotted using the scale on the left-hand side.

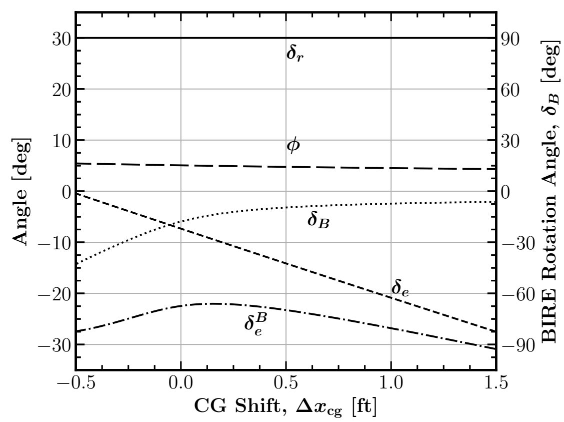
Note that the lack of a discontinuity near the stall speed in Figs. 6.10a–6.10c results in no discontinuities across any of the figures in Fig. 6.9. Also worth noting is that the



(a) Altitude : 1,000 ft



(b) Altitude : 15,000 ft

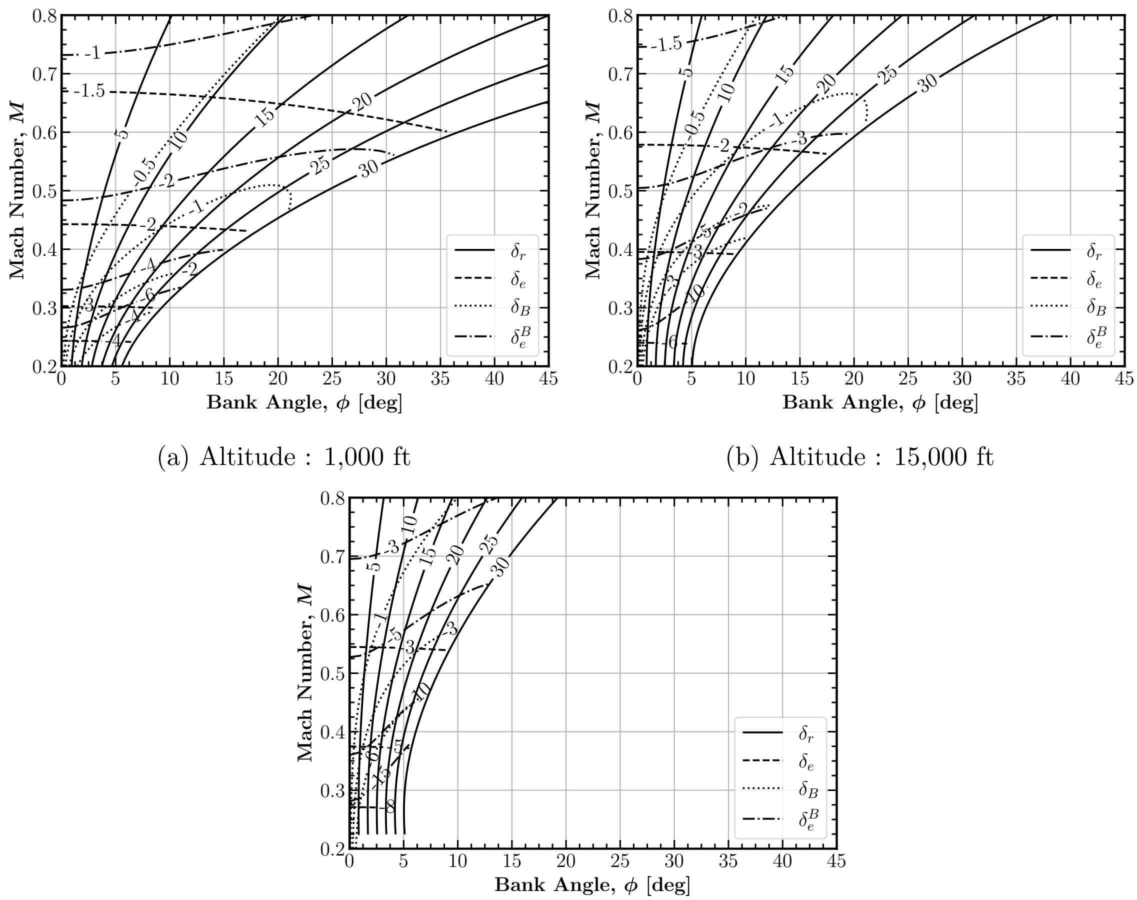


(c) Altitude : 30,000 ft

Fig. 6.9: Control surface deflections for the baseline aircraft and its BIRE variant as a function of center of gravity location at maximum crosswind landing.

maximum crosswind condition is nearly independent of altitude, as each of the figures are virtually identical. If, instead, this study were performed at the Mach number corresponding to the discontinuities in Figs. 6.10a–6.10c, it would certainly show a discontinuity like that shown in Fig. 6.6c.

Moving the CG 1 foot forward from its nominal position results in the contour plot shown in Fig. 6.10. Note that the discontinuities that were present in Fig. 6.8 are no longer visible across the range of bank angles and Mach numbers considered here. Secondly, the BIRE rotation angle contours are nearly identical in trend with the rudder deflection angles, only with slightly larger magnitudes. Finally, the overall magnitudes of the BIRE rotation angle contours are reduced, indicating that the BIRE is able to control within the same envelope as the baseline aircraft with much smaller deflections. From this study, we note the benefits of moving the center of gravity forward in the BIRE aircraft. Doing so allows the benefits of the rotating tail to be highlighted in comparison to a traditional empennage design.



(a) Altitude : 1,000 ft

(b) Altitude : 15,000 ft

(c) Altitude : 30,000 ft

Fig. 6.10: Steady-heading sideslip analysis with the center of gravity moved forward.

From the trim analysis thus far, one of the fundamental questions that were posed at the beginning of this chapter has been answered. It has been shown in both a steady, coordinated turn and in steady-heading sideslip that the trim capabilities of the BIRE aircraft are roughly equal to those of the baseline aircraft. When the center of gravity is moved forward, the benefits of the BIRE design are particularly apparent, as its trim envelope extends much farther than the baseline aircraft with a similar shift in center of gravity location. The question that remains to be answered concerns the potential of an increased risk of tail strike in the BIRE design.

6.4.3 Tail Strike Analysis

The steady-heading sideslip analysis performed previously allows for an initial estimate for the risk of tail strike when landing in a crosswind. Figure 6.11 shows the baseline and BIRE aircraft when landing in steady-heading sideslip. While landing, the aircraft can assume an elevation angle, bank angle, and have the stabilators deflected, each of which can present the aircraft with an opportunity to strike the ground. The BIRE can also have its tail rotated while landing, which will be a primary focus of this study. In this study, three points of potential contact will be considered: the engine and the trailing corner of each semispan of the stabilator.

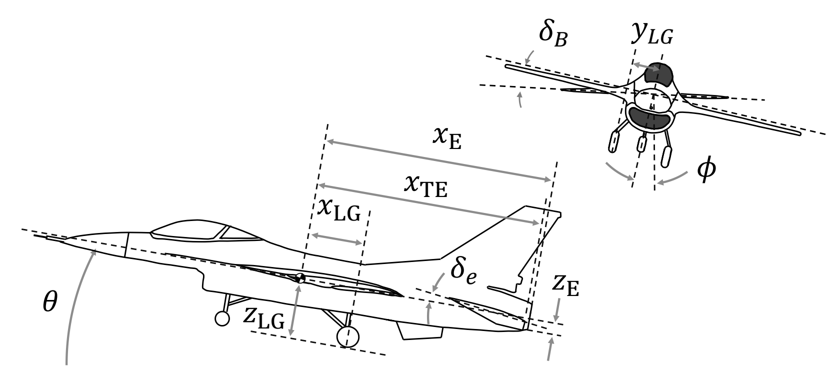


Fig. 6.11: Aircraft configuration while landing in steady-heading sideslip.

To proceed with a tail strike analysis, information about the landing gear on the baseline and BIRE aircraft must be known or assumed. When referencing the drawings provided by Fox and Forrest [62], the ventral fin is cut off at an angle that suggests it may be influenced by tail strike considerations. Therefore, a line projected along the ventral fin to the ground gives an estimate for the axial position of the main landing gear.

An estimate of the distance in the body-fixed z-direction from the center of gravity to the portion of the landing gear in contact with the ground must also be determined. This distance will be estimated by assuming that the landing gear height from the undercarriage of the aircraft is approximately equal to the distance from the centerline of the aircraft to its undercarriage. Estimations of each of these distances can be made by referencing the drawings provided by Fox and Forrest [62].

Figure 6.12 shows the coordinate systems of interest in the scenario of a tail strike. A tail strike will occur if any of the points of interest shown in Fig. 6.12 fall below the landing gear. Thus, the coordinates for the landing gear, engine, and the trailing-edge corner of each semispan of the horizontal tail must be determined and the appropriate rotations applied to determine if this is the case.

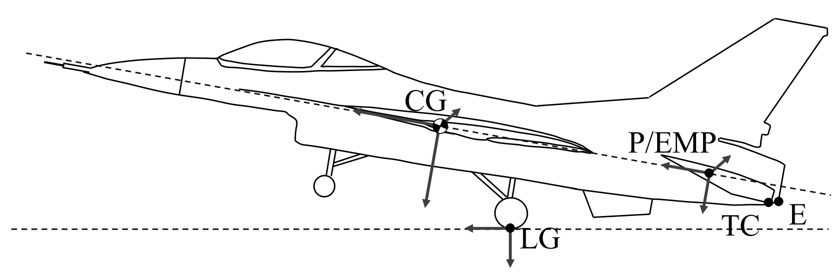


Fig. 6.12: Coordinate systems considered in the tail strike analysis.

Based on the location of the drawings from Fox and Forrest, and assuming that the landing gear has the same cant angle as that given for the ventral fins (15°), the location of the landing gear in relation to the center of gravity is given in Table 6.8 [62]. The location of the portion of the engine on the undercarriage of the aircraft (point E in Fig. 6.12) must also be approximated by referencing the aircraft drawings given by Fox and Forrest [62]. Calculating the vector from the landing gear gives the results in Table 6.8.

Table 6.8: Vectors from the landing gear to points of interest in a tail strike for each aircraft in level flight.

Point of Interest	**Vector From Landing Gear**					
	Baseline			**BIRE**		
	x [ft]	y [ft]	z [ft]	x [ft]	y [ft]	z [ft]
Center of Gravity	0.3063	1.5570	-5.8120	0.3063	1.5570	-5.8120
Engine	-18.4037	1.5570	-4.1330	-18.4037	1.5570	-4.1330
Center of Empennage	–	–	–	-14.5002	1.557	-5.8120
Left Stabilator Pivot	-14.5002	-1.8430	-5.8120	-14.5002	-1.8430	-5.8120
Right Stabilator Pivot	-14.5002	4.9570	-5.8120	-14.5002	4.9570	-5.8120
Left Stabilator TE	-18.8112	-7.5549	-4.2144	-18.8112	-7.6430	-5.8120
Right Stabilator TE	-18.8112	10.6689	-4.2144	-18.8112	10.7570	-5.8120

Due to the BIRE rotations, the location the center of the empennage rotations is required. This vector is determined by adding the vector from the landing gear to the center of gravity to the vector made by following the fuselage centerline to the pivot of the horizontal tail. Butcher provides the location of the pivot to be at 46 % of the root chord of the horizontal stabilizer [63]. Thus, the vector from the center of gravity to the center of the empennage is given by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\text{EMP}} = \begin{bmatrix} -l_h - 0.21c_{r_h} \\ 0 \\ 0 \end{bmatrix} \tag{6.33}$$

where l_h and c_{r_h} are given in Table 3.2. Adding this vector to that given in Table 6.8 for the vector to the center of gravity gives the resulting center of empennage vector in Table 6.8.

In level flight, the vector from the center of gravity to the pivot point (P in Fig. 6.12) of the horizontal stabilizer is identical for both aircraft. Equation (6.33) gives the vector from the center of gravity to the location of the pivot along the vehicle centerline. Thus, to find the vector from the center of gravity to either the left or right pivot, the body-fixed

y-distance from the centerline to the root of the stabilator (given in Figs. 3.2a and 3.3) need only be added to Eq. (6.33). This yields

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\mathrm{P_{L,R}}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\mathrm{EMP}} + \begin{bmatrix} 0 \\ \pm b_{h_f} \\ 0 \end{bmatrix} \tag{6.34}$$

where b_{h_f} is the distance in the body-fixed y-direction from the centerline to the root of the stabilator and can be determined using Fig. 3.2a.

Finally, the distance from the pivot to the trailing-edge of the horizontal stabilator must be determined. Unlike the previous vectors, this particular vector differs between the baseline and BIRE aircraft by virtue of the anhedral in the baseline. The vector spans from the pivot to the trailing corner of the stabilator, thus covering the remaining root of the stabilator and its remaining span. Thus, the vector from the pivot to the trailing-corner of the wing for the baseline aircraft is

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\mathrm{TC_{L,R}}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\mathrm{P_{L,R}}} + \begin{bmatrix} -0.54 c_{r_h} \\ \pm(b_h - b_{h_f}) \cos(\Gamma_h) \\ -(b_h - b_{h_f}) \sin(\Gamma_h) \end{bmatrix} \tag{6.35}$$

where the negative sign in front of the z-component is to account for the negative dihedral angle.

For the BIRE, this relationship is much simpler, and is simply

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\mathrm{TC_{L,R}}} = \begin{bmatrix} -0.54 c_{r_h} \\ \pm(b_h - b_{h_f}) \\ 0 \end{bmatrix} \tag{6.36}$$

Each of the vectors determined using Eqs. (6.33)–(6.34) are reported in Table 6.8 by adding the vector from the landing gear to the center of gravity. In contrast, Eq. (6.35) and (6.36) must be added to the resultant vector from Eq. (6.34).

When in a steady-heading sideslip trim condition, each vector must be rotated through the bank angle, ϕ, the elevation angle, θ, as well as the elevator deflection, δ_e or δ_e^B. Furthermore, the BIRE aircraft must undergo an additional rotation through the BIRE rotation angle, δ_B. The order of the rotations is not commutative, and therefore a consistent order must be established. For this work, we will use the traditional order adopted in flight mechanics for Euler angle rotations, which is to rotate first through the heading angle, ψ, then through the elevation angle, θ, and finally through the bank angle, ϕ [78]. Afterward, the rotations through the control deflections are made, first through the BIRE rotation angle and then through the elevator deflection. Technically, each aircraft will also assume an anti-symmetric tail deflection, δ_a; however, these are generally quite small and will be neglected in this study.

As an example, assume that, when trimmed in steady-heading sideslip, the baseline aircraft assumes a bank angle, ϕ, an elevation angle, θ, and a stabilator deflection, δ_e. With each vector being denoted as $\vec{x}$, the vector from the landing gear to the engine is given by

$$\vec{x}_{\mathrm{LG-E}} = R_\theta R_\phi \vec{x}_{\mathrm{E}} \tag{6.37}$$

where

$$R_\theta = \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \tag{6.38}$$

and

$$R_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi & c_\phi \end{bmatrix} \tag{6.39}$$

If the z-component of this vector is greater than zero (recall that the z-axis in the landing gear frame is pointed downward), then the engine has struck the ground.

The vector from the landing gear to the trailing corner of the left stabilator is given by

$$\vec{x}_{\mathrm{LG-TC_L}} = R_\theta R_\phi \left[\vec{x}_{\mathrm{P_L}} + R_{\delta_e}\vec{x}_{\mathrm{TC_L}}\right] \tag{6.40}$$

where

$$R_{\delta_e} = \begin{bmatrix} c_{\delta_e} & 0 & s_{\delta_e} \\ 0 & 1 & 0 \\ -s_{\delta_e} & 0 & c_{\delta_e} \end{bmatrix} \tag{6.41}$$

Likewise, the vector from the landing gear to the trailing corner of the right stabilator is given by

$$\vec{x}_{\mathrm{LG-TC_R}} = R_\theta R_\phi \left[\vec{x}_{\mathrm{P_R}} + R_{\delta_e}\vec{x}_{\mathrm{TC_R}}\right] \tag{6.42}$$

Again, if either of these vectors have a z-component greater than zero, they will have struck the ground.

The only modification that must be made for the BIRE is with regards to Eqs. (6.40) and (6.42). In its case, it must also rotate through the BIRE rotation angle. Thus, the vector from the landing gear to the trailing corner of the left BIRE stabilator is

$$\vec{x}_{\mathrm{LG-TC_L}} = R_\theta R_\phi \left[\vec{x}_{\mathrm{EMP}} + R_{\delta_B}\left(\vec{x}_{\mathrm{P_L}} + \{R_{\delta_e}\vec{x}_{\mathrm{TC_L}}\}\right)\right] \tag{6.43}$$

where

$$R_{\delta_B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_{\delta_B} & -s_{\delta_B} \\ 0 & s_{\delta_B} & c_{\delta_B} \end{bmatrix} \tag{6.44}$$

The corresponding vector to the right BIRE stabilator is also calculated as

$$\vec{x}_{\mathrm{LG-TC_R}} = R_\theta R_\phi \left[\vec{x}_{\mathrm{EMP}} + R_{\delta_B}\left(\vec{x}_{\mathrm{P_R}} + \{R_{\delta_e}\vec{x}_{\mathrm{TC_R}}\}\right)\right] \tag{6.45}$$

With the ability to determine whether the engine or either trailing-edge tip of the stabilator has struck the ground, the information from the steady-heading sideslip analysis can then be employed to determine whether a tail strike will occur. Since landing will occur at either condition T1 or T2, only the data with $M = 0.2$ will be used. Figures 6.13 and 6.14 show the distance to the ground for each point of interest in the baseline and BIRE aircraft as a function of bank angle. Recall that these points of interest are the engine and each trailing corner of the horizontal tail.

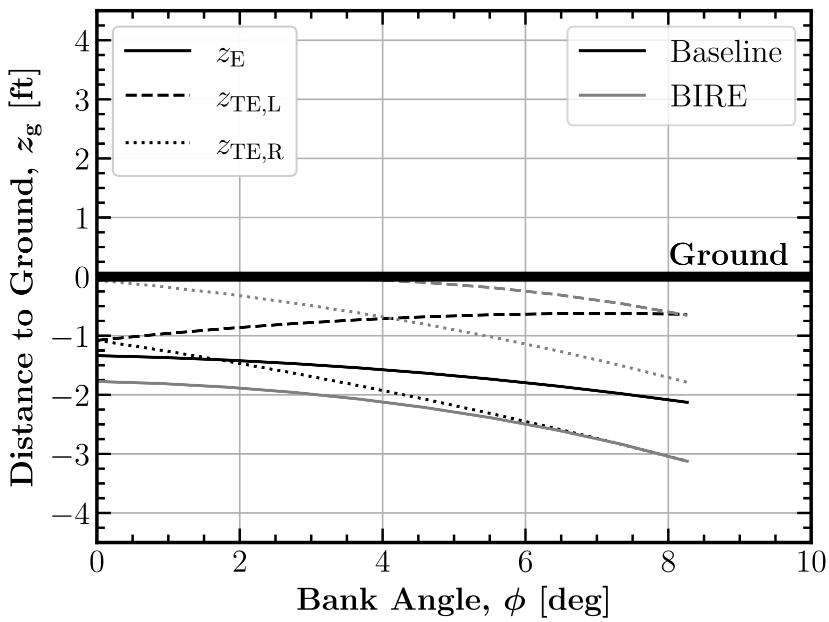


Fig. 6.13: Distances to the ground of the points of interest of each aircraft at flight condition T1.

The trends shown in each of these figures indicate that both the baseline aircraft and the BIRE will strike their tails when landing. Note that this is certainly not the case, especially at zero bank angle; however, this analysis has neglected the effect of leading-edge flaps and other high-lift devices that would be engaged when landing. High-lift devices would require each aircraft to assume a lower elevation angle, which moves each of the lines to a larger y-intercept in Figs. 6.13 and 6.14. Thus, we can see why this would not be a favorable condition for either the baseline aircraft or the BIRE to attempt to land.

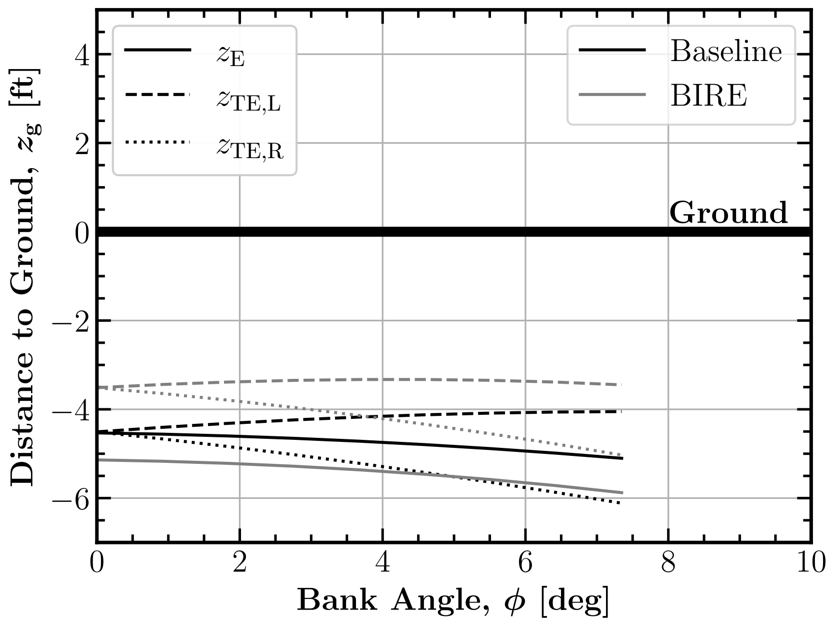


Fig. 6.14: Distances to the ground of the points of interest of each aircraft at flight condition T2.

The fundamental question at hand is whether the BIRE carries a greater risk of tail strike in this trim condition. Ignoring the engine strike scenario, both Figs. 6.13 and 6.14 show that, for bank angles less than $\phi = 4°$, the BIRE has a more favorable distance to the ground from each tail than the baseline aircraft. There are two reasons to which this result may be attributed. The first is that the BIRE lacks the anhedral present in the baseline aircraft, which explains the difference in each plot at $\phi = 0°$. This difference amounts to about 1.5 ft, when referring to Table 6.8.

The other reason that the BIRE may pose less of a chance of tail strike than the BIRE when at low bank angles is that the tail can rotate against the imposed bank angle. Each of the trim conditions along the $M = 0.2$ curve for the BIRE aircraft, as shown in Figs. 6.10a and 6.10b require negative BIRE angles, which lift the right semispan of the tail above the horizon. This rotation increases the distance between the tail and the ground when assuming a positive bank angle.

It stands to reason that shifting the center of gravity may affect the chance of a tail strike, given that the trim conditions changed rather drastically when analyzing a steady-heading sideslip condition. Thus, Figs. 6.15 and 6.16 show this data as a function of bank angle. Note in this analysis that the range of bank angles at which the tail of the BIRE aircraft is above the lines given by the baseline aircraft is reduced from $\phi \approx 4°$ to $\phi \approx 2°$ for Fig. 6.15 and $\phi \approx 1°$ for Fig. 6.16. Thus, for realistic bank angles during flight, the BIRE poses a greater risk of tail strike than the baseline aircraft with the center of gravity moved forward.

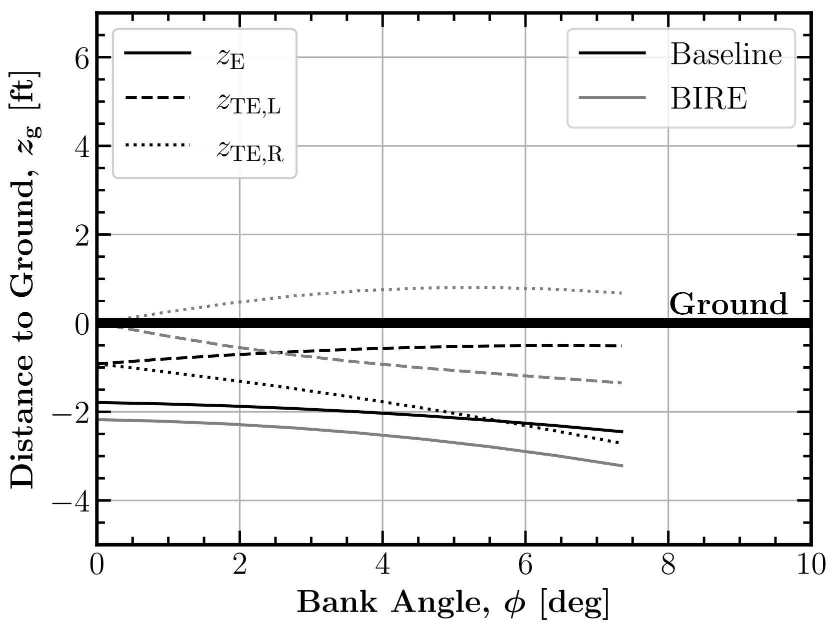


Fig. 6.15: Distances to the ground of the points of interest of each aircraft at flight condition T1 with the CG forward.

In reality, fighter aircraft generally do not land with a bank angle, and instead crab into the wind during a crosswind landing. Then, once the aircraft has touched down, the pilot deflects the rudder to point the nose straight on the runway. By referring to Figs. 6.13–6.16, one can easily see why this is the case. This kind of dynamic landing maneuver is not one that can be analyzed using the tools presented in this work. Therefore, future

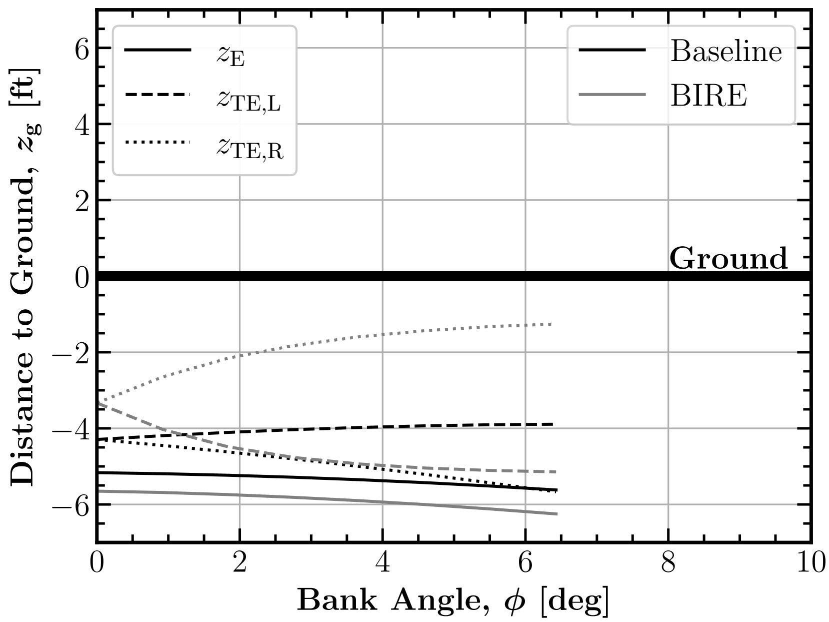


Fig. 6.16: Distances to the ground of the points of interest of each aircraft at flight condition T2 with the CG forward.

studies will be necessary to examine landing by crabbing into a crosswind. Nevertheless, this analysis has shown that tail strike is a concern for the BIRE aircraft that requires further study to properly characterize.

CHAPTER 7
ATTAINABLE MOMENT SET ANALYSIS

In Chapter 6, the static trim analysis provided context for some situations in which longitudinal and lateral control were both involved in a trim situation. Particularly, the steady-heading sideslip trim condition required both lateral and longitudinal control. It was shown throughout the chapter that the BIRE not only provided the control required to trim, but expanded the trim envelope beyond that of the baseline aircraft. However, steady-heading sideslip is still predominantly a lateral trim maneuver, while a steady, coordinated turn is predominantly a longitudinal maneuver. Thus, the exact nature of the trade-off between lateral and longitudinal control remains unclear, while it has been determined that it does not substantially reduce trim capability.

One method that can be used to more directly identify the trade-off between lateral and longitudinal control is by looking at static control authority. Trimming either aircraft, even in steady level flight, requires a portion of the control authority available to the aircraft to be sacrificed to maintain equilibrium. In the case of steady level flight, the control authority remaining in the pitch axis is limited based on the deflection of the stabilator. Thus, by calculating the moments that each aircraft is able to produce while remaining trimmed in pitch will give a good indication of the lateral and longitudinal trade-offs available to each aircraft.

The analysis just proposed is very similar to an attainable moment set (AMS) analysis. Durham first performed an attainable moment set analysis by assuming that each moment was a linear function of the controls used to produce that moment [116,117]. The purpose of this analysis was to solve a control allocation problem, determining the best way to produce a given moment with constraints on the available control surface deflections. Bolender and Doman extended Durham's work by allowing the relationship between the control effectors and the moments to be nonlinear [118].

The attainable moment set, as defined in this work, is a bounded volume in moment-space (where the dimensions are given by the aerodynamic moments on the aircraft) composed of all possible moment combinations that can be achieved by the aircraft while remaining trimmed in pitch. Thus, this analysis differs from that presented by Durham and Bolender and Doman, in that there is no particular focus on finding an appropriate control allocation, though that certainly could be the focus of future work. Rather, the intent of this AMS analysis is to highlight the trade-off presented by the BIRE control system between longitudinal and lateral control.

7.1 Moment Set Generation

To compare the attainable moment sets between the baseline and BIRE aircraft, a "cloud" of moment set combinations must be generated in moment-space. These points are calculated in reference to a trim condition, determined to be steady, level flight at each flight condition in Table 6.1. After being trimmed at each flight condition, each aircraft was allowed to deflect each of its control surfaces while maintaining the pitching moment required for trim. This process was repeated across a range of pitching moment coefficients determined by the maximum and minimum pitching moments able to be produced by the BIRE. The pitching moments produced by the BIRE were determined to be the largest by a small margin, due to the lack of anhedral in the horizontal tail.

Thus, the moment set generation problem can be summarized for the baseline aircraft as

$$\begin{aligned} &\max_{\delta_a,\delta_e,\delta_r} (C_\ell, C_n) \\ \text{s.t.} \quad & -25^\circ \leq \delta_e \leq 25^\circ \\ & -21.5^\circ \leq \delta_a \leq 21.5^\circ \\ & -30^\circ \leq \delta_r \leq 30^\circ \\ & C_m = \hat{C}_m \end{aligned} \tag{7.1}$$

Equation (7.1), in essence, provides a Pareto front of rolling moment and yawing moment coefficients at each pitching moment coefficient, given by $\hat{C}_m$. The definition of the moment set generation for the BIRE is nearly identical, given as

$$\begin{aligned} &\max_{\delta_a,\delta_e,\delta_B} (C_\ell, C_n) \\ &\text{s.t.} \quad -25^\circ \le \delta_e \le 25^\circ \\ &\qquad\quad -21.5^\circ \le \delta_a \le 21.5^\circ \\ &\qquad\quad -90^\circ \le \delta_B \le 90^\circ \\ &\qquad\quad C_m = \hat{C}_m \end{aligned} \tag{7.2}$$

where the only difference between Eq. (7.1) and Eq. (7.2) is that the BIRE uses the BIRE rotation angle δ_B as its third control input rather than the rudder of the baseline aircraft. The aerodynamic models given in Eqs. (4.60)–(4.65) and Eqs. (4.66)–(4.71) were used to determine the lateral Pareto front for the baseline and BIRE aircraft, respectively.

Equations (7.1) and (7.2) were implemented using the bounded, Sequential Least Squares Programming (SLSQP) method [119]. Thus, by constraining the stabilator angle and prescribing the aileron, rudder or BIRE angle, and desired pitching moment coefficient, the optimization routine would find the elevator deflection that would produce the appropriate pitching moment. For the baseline aircraft, sweeping through the aileron and rudder angles required no change in elevator angle from that prescribed to produce a given pitching moment. However, the coupled nature of the longitudinal and lateral moments when rotating the BIRE would cause changes in the control surface deflections to require adjustments to the stabilator deflection.

Figure 7.1 shows the moment set combinations for the baseline and BIRE aircraft at flight condition T1 in the form of a point cloud in moment space. While viewing the moment sets in this way can be helpful, more detail on any given set of moment combinations can be found by examining two-dimensional slices of moment space. In this case, it is convenient to view these slices at only the outer-edge of the control moment volume. Outlining the extreme points of each slice is readily performed using the mathematical concept of a convex hull.

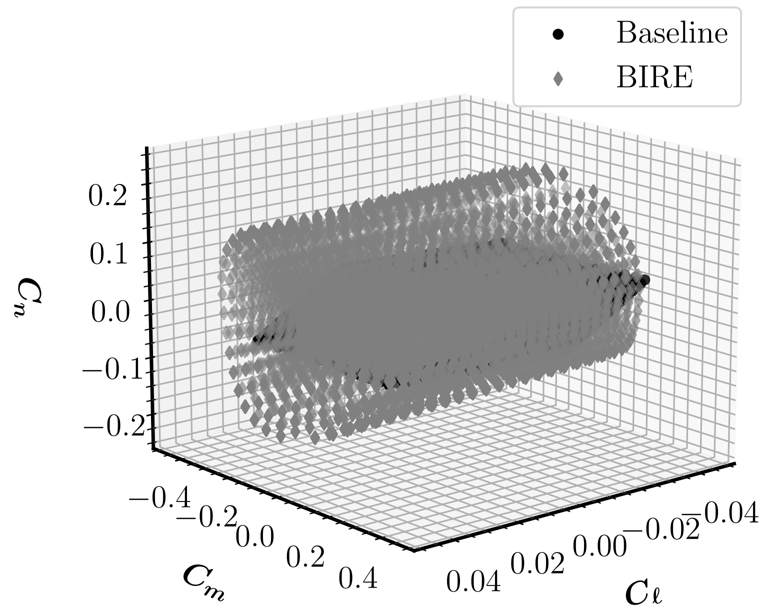
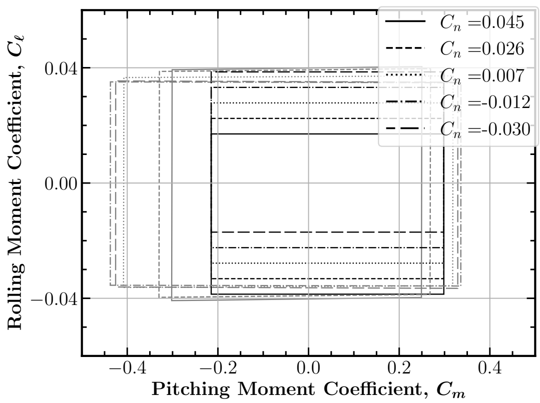


Fig. 7.1: Moment set combinations for the baseline and BIRE aircraft at flight condition T1.

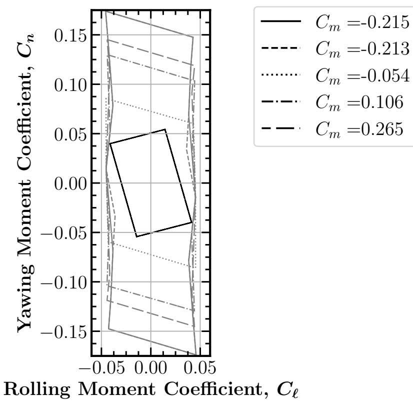
A convex hull, as the name suggests, is a mathematical construct that creates a convex bound around a set of data [120]. However, Fig. 7.1 and the rest of the moment set combinations at other flight conditions is not convex in all dimensions, and therefore the convex hull will not accurately show the bounding moment for any concave sections of data. A computational extension of the convex hull in two-dimensions that can account for concavity, called an alpha shape, was presented by Akkiraju et al. [121]. By adjusting the parameter α in the alpha shape routine, the algorithm can be adjusted to accommodate different levels of concavity in a data set and identify its extreme points.

7.2 Attainable Moment Set Comparison

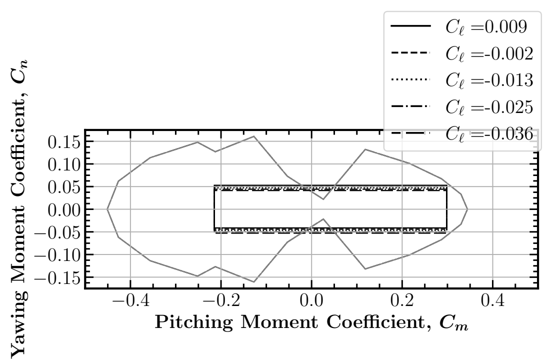
Figure 7.2 shows the alpha shapes for two-dimensional slices along each moment axis in Fig. 7.1. The coefficient in the perpendicular direction at which the slice is taken is denoted in the legend for each plot. To distinguish between the attainable moment sets of the baseline aircraft and the BIRE, the baseline slices are colored black while the BIRE slices are gray. Note that Fig. 7.2a is scaled in the rolling moment coefficient direction for visibility reasons, while Figs. 7.2b and 7.2c are given by the aspect ratio corresponding to the data.



(a) Pitch/Roll (Scaled)



(b) Roll/Yaw



(c) Pitch/Yaw

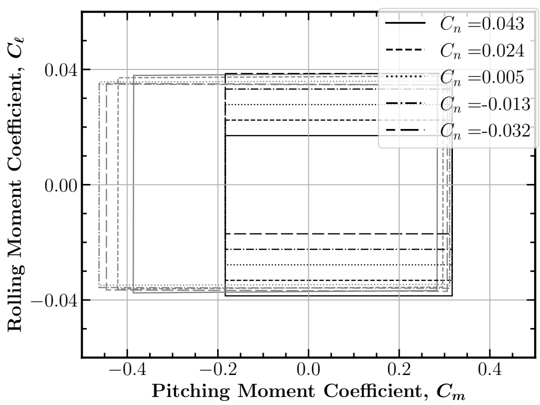
Fig. 7.2: Attainable moment sets at flight condition T1. Black lines correspond to the baseline aircraft and grey lines to the BIRE.

From Fig. 7.2a, it is noted that the BIRE is centered on the rolling moment axis, while the baseline aircraft steadily shifts from being centered on a negative rolling moment to centering on a positive rolling moment as the yawing moment decreases. This fact will be made more clear in conjunction with an analysis of Fig. 7.2b, which will be covered shortly. In terms of trade-offs between roll and pitch, it can be noted that the BIRE loses only minimum roll authority at large yawing moments and has a small loss in pitch authority at large, positive yawing moments. This is to be expected from the coupling between yaw and pitch, but the magnitude of the loss is fairly small. As with the analysis in Chapter 6, the impact of these trade-offs are dependent on the operating envelope of the aircraft, which has not been examined in detail here. Further research is required to determine whether these trade-offs are detrimental to the mission of the aircraft.

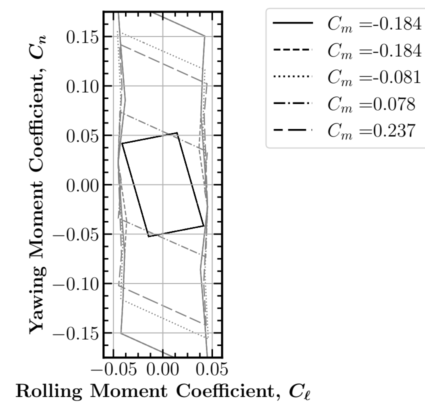
The information in Fig. 7.2b provides interesting insight into the relationship between the lateral moments of the baseline and those of the BIRE. Contours for the baseline aircraft are constant with pitching moment, since there is no coupling between the lateral and longitudinal degrees of freedom induced by the controls. Note, however, that the baseline contours are angled, with a negative correlation between the rolling and yawing moments. This is the exact same coupling that causes adverse yaw to occur when an aircraft turns during flight. The BIRE contours, however, do not have the same level of bias towards a negative correlation. In Chapter 2, it was mentioned by Thomas that tail rotations could be used to counteract adverse yaw [26]. Trends in Fig. 7.3b could provide additional evidence of the veracity of that statement.

Finally, Fig. 7.2c highlights the trade-offs between longitudinal and lateral control in the BIRE. While the baseline aircraft changes very little in its attainable yawing moments with changes in pitching moment, the BIRE has substantially reduced yaw control when trimmed in steady level flight ($C_m = 0$). The overall pitching moment accessible to the BIRE is similar to that of the baseline, with the pitching moment offset of the BIRE alpha shape in Fig. 7.2c caused by the increased negative stabilator deflection required for trim.

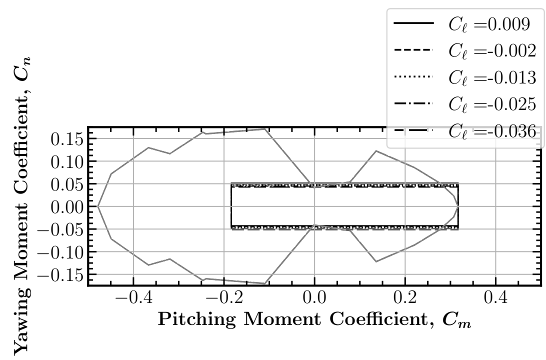
This analysis can be repeated for each of the flight conditions in Table 6.1, which are shown in Figs. 7.3–7.6. Many of the trends discussed in Fig. 7.2 are repeated at these other flight conditions, but a few special cases will be mentioned. The trade-offs between pitch and roll are nearly constant with flight condition, with the only noticeable difference being that the slices in yawing moment for the BIRE are more centered and vary less with yawing moment at higher altitudes. This is likely corresponding to the change in trim condition, which reduces pitching moment offsets in the pitch and yaw trade-offs as well.



(a) Pitch/Roll (Scaled)



(b) Roll/Yaw

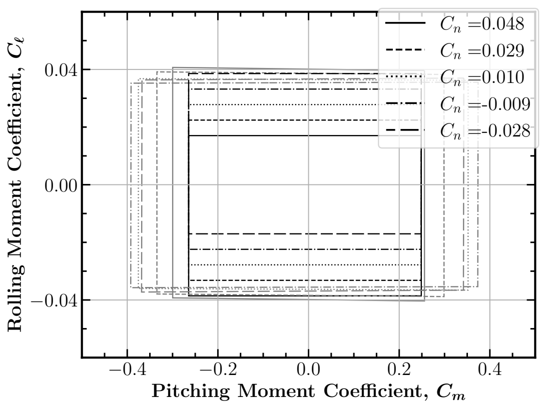


(c) Pitch/Yaw

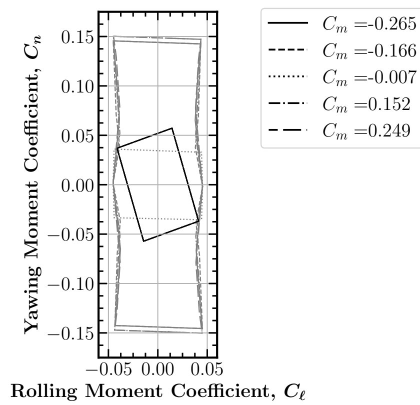
Fig. 7.3: Attainable moment sets at flight condition T2. Black lines correspond to the baseline aircraft and grey lines to the BIRE.

The roll and yaw moment figures become increasingly more symmetrical for the BIRE aircraft as the altitude and Mach number are increased. Again, this indicates that the BIRE is not limited in producing coupled roll-yaw moments during flight and could potentially reduce adverse yaw effects during turning flight. An important note here is that pitching moments near trim always limit the lateral control that the BIRE can produce, which could limit the capability of the BIRE when experiencing severe lateral disturbances.

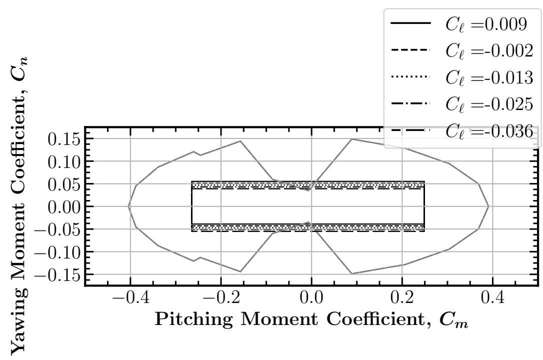
The trend showing reduced yaw control near trim is consistent at all but two flight conditions. First, in the power-on stall condition shown in Fig. 7.3c, there appears to be no loss of pitch control near trim. This is also represented in the turbulent penetration flight condition in Fig. 7.4c. The discrepancy here is a result of resolution, and increasing the



(a) Pitch/Roll (Scaled)



(b) Roll/Yaw

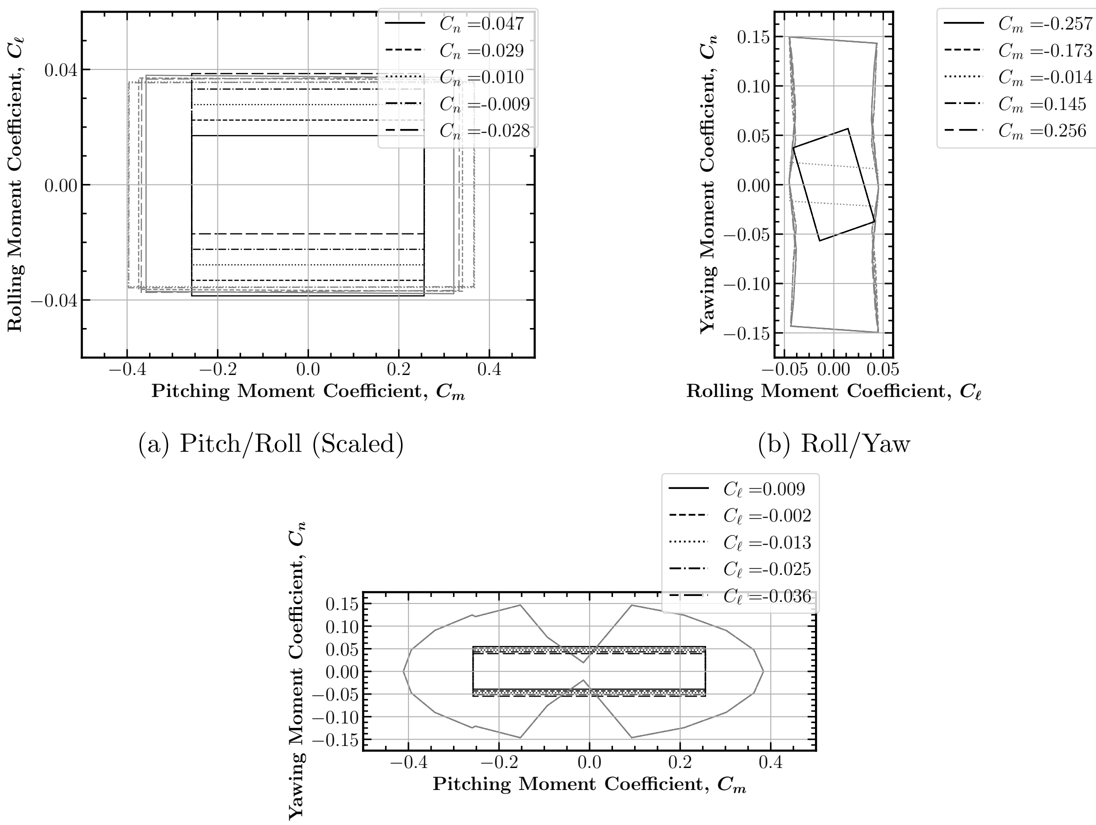


(c) Pitch/Yaw

Fig. 7.4: Attainable moment sets at flight condition C1. Black lines correspond to the baseline aircraft and grey lines to the BIRE.

number of pitching moment coefficients for which the data is produced reveals that there is always a decrease in yawing moment coefficient at pitching moments near trim.

The sharp gradient in the yawing moment in these studies is interesting, and indicates that the BIRE is proficient at providing a substantial amount of yaw control very quickly with small changes in pitch. Thus, in conditions where lateral control is necessary and pitch control must be sacrificed, the BIRE may be able to quickly damp out disturbances in yaw and return to a trim condition. This too requires additional, dynamic studies beyond the scope of this work. Another driving factor in this trade-off is the amount of yaw authority required by either aircraft in a given situation. While the baseline aircraft has been sized according to some maximum-yaw flight condition, it could be that the BIRE requires less



(a) Pitch/Roll (Scaled)

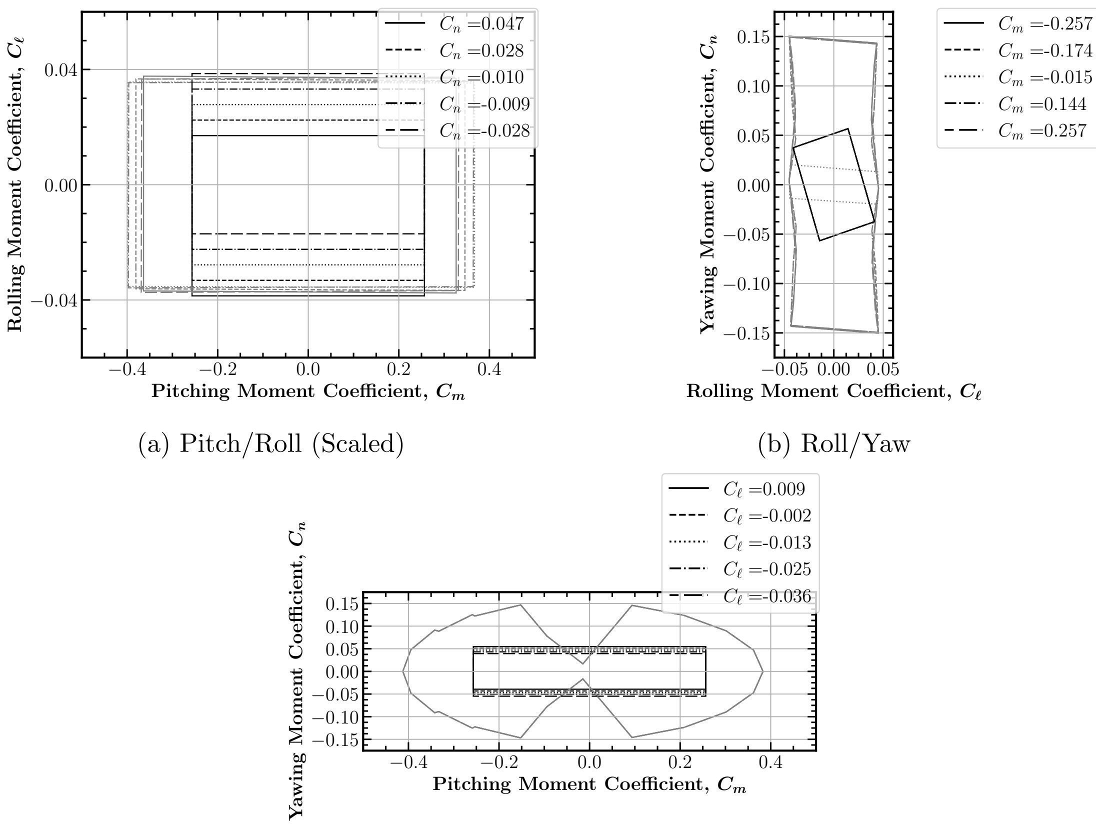
(b) Roll/Yaw

(c) Pitch/Yaw

Fig. 7.5: Attainable moment sets at flight condition C2. Black lines correspond to the baseline aircraft and grey lines to the BIRE.

yaw authority for that same condition by virtue of its lack of vertical tail. Again, these questions require additional research to understand completely, but must be understood to further compare the BIRE with traditional aircraft controls.

As a final note, the maximum available control authority for each aircraft is noted for all flight conditions of interest in Table 7.1. This gives an idealized look at the total control authority available to each aircraft without the constraints of trim, where the benefits of the BIRE are maximized in comparison with the baseline aircraft and the trade-offs with pitch are not highlighted. Still, the nearly three-fold increase in yaw authority of the BIRE aircraft over the baseline aircraft is worthy of note. Additionally, the BIRE is able to access a large portion of that authority quickly, as was noted previously in Figs. 7.2–7.6.



(a) Pitch/Roll (Scaled)

(b) Roll/Yaw

(c) Pitch/Yaw

Fig. 7.6: Attainable moment sets at flight condition C3. Black lines correspond to the baseline aircraft and grey lines to the BIRE.

7.3 A Comparison Between Yaw Control and Drag

An additional comparison made available through the aerodynamic models developed in this work involves looking at the drag associated with lateral control. The impetus behind the development of a rotating tail for control is to reduce drag and weight from a vertical tail. As mentioned in the introduction to this work, one branch of research that has been studied extensively involves removing the tail entirely and using the wing for control of all three degrees of freedom. Of the methods being investigated to do so, control of lateral moments using wing twist has been shown to be one potential solution [122–124].

Table 7.1: Maximum and minimum moments produced by the baseline and BIRE aircraft at each flight condition in Table 6.1.

Flight Condition	Moment	Baseline		BIRE	
		Max	Min	Max	Min
T1	C_ℓ	0.041	-0.041	0.045	-0.045
	C_m	0.289	-0.224	0.347	-0.449
	C_n	0.054	-0.054	0.168	-0.168
T2	C_ℓ	0.041	-0.041	0.045	-0.045
	C_m	0.25	-0.263	0.375	-0.42
	C_n	0.057	-0.057	0.155	-0.155
C1	C_ℓ	0.041	-0.041	0.045	-0.045
	C_m	0.313	-0.2	0.33	-0.465
	C_n	0.052	-0.052	0.181	-0.181
C2	C_ℓ	0.041	-0.041	0.045	-0.045
	C_m	0.256	-0.257	0.371	-0.424
	C_n	0.057	-0.057	0.156	-0.156
C3	C_ℓ	0.041	-0.041	0.045	-0.045
	C_m	0.256	-0.257	0.371	-0.425
	C_n	0.057	-0.057	0.156	-0.156

Montgomery has shown that the theoretical minimum induced drag increment from a pure yaw maneuver using a point load on the outermost tip of a wing is [125]

$$\Delta C_{D_i} = 2|C_n| \tag{7.3}$$

This drag increment can be compared to the drag produced using the rudder of the baseline aircraft to generate a yawing moment. By sweeping from minimum to maximum rudder deflection in the baseline aircraft with each of the other aerodynamic parameters in Eq. (4.65) set to zero, the drag increment of the baseline aircraft for a given yawing moment can be calculated using Eq. (4.62). Both Eq. (7.3) from Montgomery [125] and the drag increment induced by the baseline aircraft from Eq. (4.62) exerting the same yawing moment is shown in Fig. 7.7.

For its contribution to the study, the BIRE was made to produce the same yawing moment as the baseline aircraft using a combination of BIRE rotation and stabilator deflection. This process required another optimization, this time varying stabilator deflection and

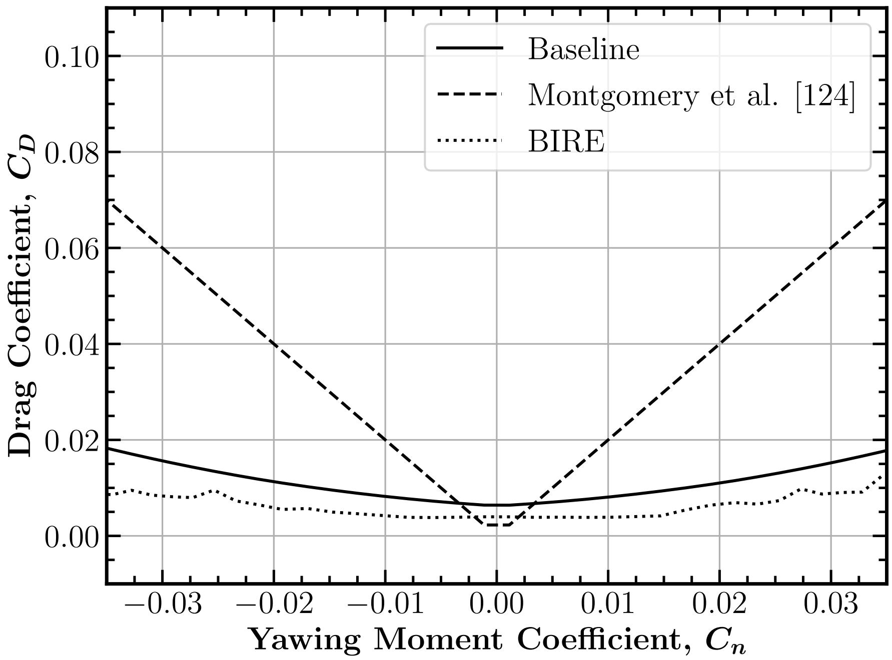


Fig. 7.7: Comparison of the drag produced using only the wing, a traditional aircraft control system, and the BIRE to produce yawing moments.

BIRE rotation to converge the BIRE yawing moment, given by Eq. (4.71), to the yawing moment produced by the baseline. In this case, the Nelder-Mead method was used [126,127]. Then, with the elevator and BIRE rotation known, Eq. (4.68) was used to calculate the drag. The resulting yawing moment versus drag curve is also shown in Fig. 7.7.

Figure 7.7 shows an intriguing result; the BIRE produces less drag for a given yawing moment than the baseline aircraft or the minimum increment predicted by Montgomery et al. [125]. The exception to that statement is for very small yawing moment coefficients, where using the wing to produce a pure yawing moment will show slightly less drag than the BIRE. Nevertheless, in terms of producing substantial yawing moments, the BIRE design seems to promise improved performance over traditional designs and using a point force at the wing tip in terms of drag production.

The reduction in drag provided by the BIRE is likely due to being able to leverage a larger surface area, thus requiring less deflection to produce a given yawing moment. In addition, the wings rely on drag production and manipulation of the lift distribution to produce yawing moments. Therefore, separation effects would provide a large increase in drag. The BIRE, however, does not rely on drag to produce yawing moments, and its parasitic drag contributions are likely lower than that of an aircraft using the wing for yaw control.

Controlling yaw using the wing alone provides the benefit of completely removing the weight and drag produced by the empennage of an aircraft. Nevertheless, the result in Fig. 7.7 provides an intriguing reason to continue higher-fidelity research that includes viscous and other drag effects that the model in Eqs. (4.62) and (4.68) does not include.

CHAPTER 8

A LINEARIZED CONTROL SYSTEM ANALYSIS

Aircraft with "relaxed" static stability are generally made flyable by pilots through the implementation of a stability augmentation control system. These control systems make the aircraft feel stable to the pilot using the control inputs available to the aircraft, such as the traditional elevator, aileron, and rudder control surfaces. In fact, the baseline aircraft is controlled in the longitudinal plane with a pilot-commanded normal acceleration system with pitch rate and normal acceleration feedback and a forward-loop integration on the acceleration response [64] Laterally, the pilot commands the roll rate and rudder with yaw-rate and lateral acceleration feedback [64]. The baseline aircraft control system is described in much more detail by Nguyen et al. [64].

By removing the vertical tail, these control systems grow much more complex and non-linear, since each control input is not linearly associated with only one of the aerodynamic moments. The BIRE design is no different, and the aim of the study in this chapter is to provide a linearized system that can be used to identify potential challenges to controlling the BIRE aircraft.

The equations of motion for an aircraft, given in Eqs. (4.1)–(4.4), are decidedly non-linear, as are the aerodynamic models used for the aircraft. It is common practice to begin a control analysis using instead a system linearized about a given point or trajectory, since this simplifies the analysis and development of a controller [128]. In this chapter, a linearized system will be developed from the aircraft equations of motion given in Eqs. (4.1)–(4.4) and using the aerodynamic model of the baseline aircraft in Eqs. (4.60)–(4.65) and the aerodynamic model of the BIRE given by Eqs. (4.66)–(4.71). This linearized system is presented in such a way that changes in center of gravity location and the equilibrium point about which it is generated are easily changed. An analysis of the control properties of the linearized system will then be presented and a linear feedback controller developed for each aircraft using the linear quadratic regulator (LQR) method. The effectiveness of the designed control system for each aircraft will then be determined through a twelve degree-of-freedom simulation.

An immediate concern that arises when considering the BIRE as a control concept is the lack of yaw stability and damping in the aircraft when the tail is held horizontally. In addition, the trade-offs between longitudinal and lateral control were discussed in the previous chapter, and indicate that there may be situations in which the aircraft cannot produce both sufficient pitch control and yaw control at the same time. These two issues form the basis for this exploration of the control properties of the BIRE system. This chapter addresses these issues by looking at typical MIMO system control properties that govern performance and robustness and then simulating a disturbance in the form of a wind gust acting on each aircraft and analyzing the response of each aircraft when using a linear feedback controller [128].

8.1 Linearizing the Equations of Motion

The first step in the process of linearizing the equations of motion is to choose the states that define the linearized model. In an effort to simplify the problem of generating a feedback controller, two restrictions will be made. Since a linearized system requires a point or trajectory about which to be linearized, this work will focus only on equilibrium conditions in steady level flight. Second, since a primary purpose of this study is to analyze the disturbance rejection properties of the aircraft, the states chosen will be those that are perceived to be of primary concern to remaining in this trim condition.

By controlling the body-fixed velocities, rotation rates, and the elevation and bank angle, the controller will be able to maintain a given steady level flight condition. Thus, for both aircraft, the state vector is defined as

$$x = \begin{bmatrix} u & v & w & p & q & r & \theta & \phi \end{bmatrix}^T \tag{8.1}$$

Likewise, we must define the control inputs for each aircraft. For the case of the baseline aircraft, these are

$$\tilde{u} = \begin{bmatrix} \delta_a & \delta_e & \delta_r \end{bmatrix}^T \tag{8.2}$$

and for the BIRE variant

$$\tilde{u} = \begin{bmatrix} \delta_a & \delta_e^B & \delta_B \end{bmatrix}^T \tag{8.3}$$

Note that the nomenclature of the inputs is adjusted so as to avoid any confusion with the body-fixed velocity in the x-direction. With the states and control inputs defined, a linear, state-space description of the aircraft system dynamics of the form

$$\dot{x} = Ax + B\tilde{u} \tag{8.4}$$

must be determined, where A is the linearized state matrix and B is the linearized control input matrix.

Including the effects of wind gusts in Eq. (4.1) and rewriting Eqs. (4.2) and (4.4), the aircraft state dynamics can be written as

$$\begin{Bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{Bmatrix} = \frac{g}{W} \begin{Bmatrix} F_{x_b} \\ F_{y_b} \\ F_{z_b} \end{Bmatrix} + g \begin{Bmatrix} -s_\theta \\ s_\phi c_\theta \\ c_\phi c_\theta \end{Bmatrix} + \begin{Bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{Bmatrix} + \begin{Bmatrix} \dot{V}_{g_x} \\ \dot{V}_{g_y} \\ \dot{V}_{g_z} \end{Bmatrix} \tag{8.5}$$

$$\begin{Bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{Bmatrix} = \mathcal{I}^{-1} \begin{Bmatrix} M_{x_b} + (I_{yy_b} - I_{zz_b})\, qr + I_{yz_b}(q^2 - r^2) + I_{xz_b} pq - I_{xy_b} pr \\ M_{y_b} + (I_{zz_b} - I_{xx_b})\, pr + I_{xz_b}\left(r^2 - p^2\right) + I_{xy_b} qr - I_{yz_b} pq \\ M_{z_b} + (I_{xx_b} - I_{yy_b})\, pq + I_{xy_b}(p^2 - q^2) + I_{yz_b} pr - I_{xz_b} qr \end{Bmatrix} \tag{8.6}$$

and

$$\begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \end{Bmatrix} = \begin{bmatrix} 1 & s_\phi s_\theta / c_\theta & c_\phi s_\theta / c_\theta \\ 0 & c_\phi & -s_\phi \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \tag{8.7}$$

With the state defined as the vector x, where

$$x \equiv \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \end{bmatrix}^T = \begin{bmatrix} u & v & w & p & q & r & \phi & \theta \end{bmatrix}^T \tag{8.8}$$

the dynamics of the system can be written as

$$\dot{x} \equiv \begin{bmatrix} \dot{x}_1 & \dot{x}_2 & \dot{x}_3 & \dot{x}_4 & \dot{x}_5 & \dot{x}_6 & \dot{x}_7 & \dot{x}_8 \end{bmatrix}^T = \begin{bmatrix} \dot{u} & \dot{v} & \dot{w} & \dot{p} & \dot{q} & \dot{r} & \dot{\phi} & \dot{\theta} \end{bmatrix}^T \tag{8.9}$$

according to the form given in Eq. (8.4).

In Eq. (8.6), the inertia tensor is defined as

$$\mathcal{I} = \begin{bmatrix} I_{xx_b} & -I_{xy_b} & -I_{xz_b} \\ -I_{xy_b} & I_{yy_b} & -I_{yz_b} \\ -I_{xz_b} & -I_{yz_b} & I_{zz_b} \end{bmatrix} \tag{8.10}$$

and its inverse is defined

$$\mathcal{I}^{-1} = \frac{1}{\det(\mathcal{I})} \operatorname{adj}(\mathcal{I}) \tag{8.11}$$

where $\operatorname{adj}(\mathcal{I})$ is the adjoint of the inertia matrix. The determinant of the inertia tensor required by Eq. (8.11) can be found to be

$$\det \mathcal{I} = I_{xx_b}\left(I_{yy_b} I_{zz_b} - I_{yz_b}^2\right) - I_{xy_b}\left(I_{yz_b} I_{xz_b} + I_{xy_b} I_{zz_b}\right) - I_{xz_b}\left(I_{xy_b} I_{yz_b} + I_{yy_b} I_{xz_b}\right) \tag{8.12}$$

and the adjoint of the inertia tensor is

$$\operatorname{adj}(\mathcal{I}) = \begin{bmatrix} I_{yy_b} I_{zz_b} - I_{yz_b}^2 & I_{xy_b} I_{zz_b} + I_{xz_b} I_{yz_b} & I_{xy_b} I_{yz_b} + I_{xz_b} I_{yy_b} \\ I_{xy_b} I_{zz_b} + I_{yz_b} I_{xz_b} & I_{xx_b} I_{zz_b} - I_{xz_b}^2 & I_{xx_b} I_{yz_b} + I_{xy_b} I_{xz_b} \\ I_{xy_b} I_{yz_b} + I_{xz_b} I_{yy_b} & I_{xx_b} I_{yz_b} + I_{xz_b} I_{xy_b} & I_{xx_b} I_{yy_b} - I_{xy_b}^2 \end{bmatrix} \tag{8.13}$$

These definitions will prove useful when determining the linearization of the BIRE dynamical system.

Consider an equilibrium state which is, $\hat{x}$ given by a steady level flight trim condition

$$\hat{x} = \begin{bmatrix} \hat{u} & \hat{v} & \hat{w} & \hat{p} & \hat{q} & \hat{r} & \hat{\phi} & \hat{\theta} \end{bmatrix}^T \tag{8.14}$$

A change of state can be imposed, given by

$$z = x - \hat{x} \tag{8.15}$$

which goes to zero when $x = \hat{x}$ or when the state of the aircraft is in its trimmed condition. The dynamics of the change of state are straight-forward to calculate, being

$$\dot{z} = \dot{x} - \dot{\hat{x}} = \dot{x} \tag{8.16}$$

since $\dot{\hat{x}}$ is, by the definition of a trim state, equal to zero. Thus, the dynamics of the shifted system are identical to the dynamics of the original, un-shifted system.

Now consider a perturbation to the state and control inputs, given by

$$\Delta z = z - \hat{z} \tag{8.17}$$

and

$$\Delta \tilde{u} = \tilde{u} - \hat{\tilde{u}} \tag{8.18}$$

respectively. A first-order Taylor-series expansion of the shifted system dynamics, $\dot{z}$, about the equilibrium point, $\left(\hat{z}, \hat{\tilde{u}}\right)$ yields

$$\dot{z}\left(\hat{z} + \Delta z, \hat{\tilde{u}} + \Delta \tilde{u}\right) \approx \dot{z}\left(\hat{z}, \hat{\tilde{u}}\right) + \frac{\partial \dot{z}}{\partial z}\left(\hat{z}, \hat{\tilde{u}}\right)\Delta z + \frac{\partial \dot{z}}{\partial u}\left(\hat{z}, \hat{\tilde{u}}\right)\Delta \tilde{u} \tag{8.19}$$

where $\dot{z}\left(\hat{z},\hat{\tilde{u}}\right)=0$. The Jacobians given in Eq. (8.19) can be evaluated as

$$\frac{\partial \dot{z}_i}{\partial z_j}\left(\hat{z},\hat{\tilde{u}}\right)=\left.\begin{bmatrix} \frac{\partial \dot{z}_1}{\partial z_1} & \frac{\partial \dot{z}_1}{\partial z_2} & \cdots & \frac{\partial \dot{z}_1}{\partial z_8} \\ \frac{\partial \dot{z}_2}{\partial z_1} & \frac{\partial \dot{z}_2}{\partial z_2} & \cdots & \frac{\partial \dot{z}_2}{\partial z_8} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \dot{z}_8}{\partial z_1} & \frac{\partial \dot{z}_8}{\partial z_2} & \cdots & \frac{\partial \dot{z}_8}{\partial z_8} \end{bmatrix}\right|_{(z=\hat{z},\tilde{u}=\hat{\tilde{u}})} \tag{8.20}$$

and

$$\frac{\partial \dot{z}_i}{\partial \tilde{u}_j}\left(\hat{z},\hat{\tilde{u}}\right)=\left.\begin{bmatrix} \frac{\partial \dot{z}_1}{\partial \tilde{u}_1} & \frac{\partial \dot{z}_1}{\partial \tilde{u}_2} & \frac{\partial \dot{z}_1}{\partial \tilde{u}_3} \\ \frac{\partial \dot{z}_2}{\partial \tilde{u}_1} & \frac{\partial \dot{z}_2}{\partial \tilde{u}_2} & \frac{\partial \dot{z}_2}{\partial \tilde{u}_3} \\ \vdots & \vdots & \vdots \\ \frac{\partial \dot{z}_8}{\partial \tilde{u}_1} & \frac{\partial \dot{z}_8}{\partial \tilde{u}_2} & \frac{\partial \dot{z}_8}{\partial \tilde{u}_3} \end{bmatrix}\right|_{(z=\hat{z},\tilde{u}=\hat{\tilde{u}})} \tag{8.21}$$

The matrices in Eqs. (8.20) and (8.21) are constants, and will be denoted A and B, respectively. Thus, Eq. (8.19) can be rewritten as

$$\dot{z}\left(\hat{z}+\Delta z,\hat{\tilde{u}}+\Delta\tilde{u}\right)\approx A\Delta z+B\Delta\tilde{u} \tag{8.22}$$

where A will be called the linearized state matrix and B will be called the linearized control matrix. Evaluated at the trim position, $\left(\hat{z},\hat{\tilde{u}}\right)$, this relationship is exact; however, it can also be reasonably accurate so long as the perturbations Δz and $\Delta\tilde{u}$ are within some "small" region around the trim condition.

If the system in Eq. (8.22) is uncontrolled (i.e. $B = 0$) and locally asymptotically stable, the states will tend towards zero in the absence of perturbations or when the perturbations vanish sufficiently fast. The stability of an uncontrolled linear system can be determined using the eigenvalues of the matrix A. So long as the real portion of the eigenvalues of A are negative, the linearized system is asymptotically stable [129].

For a controlled linear system (A, B), as given in Eq. (8.4), the stability of the system is dependent on the control matrix as well. If the system given in Eq. (8.22) is not asymptotically stable, the inputs to the linearized system, $\Delta\tilde{u}$, may be used to stabilize the system. To do so, it must be shown that the system is stabilizable to design a controller using an infinite-horizon LQR method that will produce an asymptotically stable system [130]. A stronger requirement is that the controllability matrix of the system, given by [131]

$$\Gamma = \begin{bmatrix} B & AB & A^2B & \cdots & A^nB \end{bmatrix} \tag{8.23}$$

has rank equal to the number of states in the system ($\text{rank}(\Gamma) = n$). If this is the case, then the system is completely controllable and $\Delta\tilde{u}$ can be used to stabilize the system.

One method of stabilizing a linear system is to use a state-feedback controller. A state-feedback controller can be described as [130]

$$\Delta\tilde{u} = -Kz \tag{8.24}$$

Note that the control input to the system is defined as a linear combination of the states of the system. The coefficients in these linear combinations are contained within the matrix K, which is often referred to as the feedback gain matrix.

To calculate the linearized state and control matrices for the linearized system in Eq. (8.22), the change of state given in Eq. (8.15) must be applied to the aircraft dynamics in Eqs. (8.5)–(8.7). Doing so for the dynamics of the aircraft velocity yields

$$\begin{Bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \end{Bmatrix} = \frac{g}{W}\begin{Bmatrix} F_{x_b} \\ F_{y_b} \\ F_{z_b} \end{Bmatrix} + g\begin{Bmatrix} -s_{(z_8+\hat{x}_8)} \\ s_{(z_7+\hat{x}_7)}c_{(z_8+\hat{x}_8)} \\ c_{(z_7+\hat{x}_7)}c_{(z_8+\hat{x}_8)} \end{Bmatrix} + \begin{Bmatrix} (z_6+\hat{x}_6)(z_2+\hat{x}_2) - (z_5+\hat{x}_5)(z_3+\hat{x}_3) \\ (z_4+\hat{x}_4)(z_3+\hat{x}_3) - (z_6+\hat{x}_6)(z_1+\hat{x}_1) \\ (z_5+\hat{x}_5)(z_1+\hat{x}_1) - (z_4+\hat{x}_4)(z_2+\hat{x}_2) \end{Bmatrix} \tag{8.25}$$

and likewise, the rotational velocity dynamics can be written as

$$\begin{Bmatrix} \dot{z}_4 \\ \dot{z}_5 \\ \dot{z}_6 \end{Bmatrix} = \mathcal{I}^{-1}\begin{Bmatrix} M_1 \\ M_2 \\ M_3 \end{Bmatrix} \tag{8.26}$$

where

$$M_1 = M_{x_b} + (I_{yy_b} - I_{zz_b})(z_5 + \hat{x}_5)(z_6 + \hat{x}_6) + I_{yz_b}\left[(z_5 + \hat{x}_5)^2 - (z_6 + \hat{x}_6)^2\right] \\ + I_{xz_b}(z_4 + \hat{x}_4)(z_5 + \hat{x}_5) \tag{8.27}$$

$$M_2 = M_{y_b} + (I_{zz_b} - I_{xx_b})(z_4 + \hat{x}_4)(z_6 + \hat{x}_6) + I_{xz_b}\left[(z_6 + \hat{x}_6)^2 - (z_4 + \hat{x}_4)^2\right] \\ - I_{yz_b}(z_4 + \hat{x}_4)(z_5 + \hat{x}_5) \tag{8.28}$$

and

$$M_3 = M_{z_b} + (I_{xx_b} - I_{yy_b})(z_4 + \hat{x}_4)(z_5 + \hat{x}_5) + I_{yz_b}(z_4 + \hat{x}_4)(z_6 + \hat{x}_6) \\ - I_{xz_b}(z_5 + \hat{x}_5)(z_6 + \hat{x}_6) \tag{8.29}$$

Finally, the orientation dynamics of Eq. (8.7) are rewritten as

$$\begin{Bmatrix} \dot{z}_7 \\ \dot{z}_8 \end{Bmatrix} = \begin{bmatrix} 1 & s_{(z_7+\hat{x}_7)} s_{(z_8+\hat{x}_8)} / c_{(z_8+\hat{x}_8)} & c_{(z_7+\hat{x}_7)} s_{(z_8+\hat{x}_8)} / c_{(z_8+\hat{x}_8)} \\ 0 & c_{(z_7+\hat{x}_7)} & -s_{(z_7+\hat{x}_7)} \end{bmatrix} \begin{Bmatrix} (z_4 + \hat{x}_4) \\ (z_5 + \hat{x}_5) \\ (z_6 + \hat{x}_6) \end{Bmatrix} \tag{8.30}$$

in the shifted system.

The dynamics of the shifted system in Eqs. (8.25)–(8.30) require a description of the aerodynamic forces and moments acting on the aircraft. For both the baseline aircraft and the BIRE, with the thrust aligned with the centerline of the aircraft, these are given as

$$F_{x_b} = \frac{1}{2}\rho V^2 S_w C_X + F_{P_x} = -\left(C_D c_\alpha c_\beta + C_S c_\alpha s_\beta - C_L s_\alpha\right)\frac{1}{2}\rho V^2 S_w + F_{P_x} \tag{8.31}$$

$$F_{y_b} = \frac{1}{2}\rho V^2 S_w C_Y = \left(C_S c_\beta - C_D s_\beta\right)\frac{1}{2}\rho V^2 S_w \tag{8.32}$$

$$F_{z_b} = \frac{1}{2}\rho V^2 S_w C_Z = -\left(C_D s_\alpha c_\beta + C_S s_\alpha s_\beta + C_L c_\alpha\right)\frac{1}{2}\rho V^2 S_w \tag{8.33}$$

$$M_{x_b} = \frac{1}{2}\rho V^2 S_w b_w C_\ell - F_{z_b}\Delta y_b + F_{y_b}\Delta z_b \tag{8.34}$$

$$M_{y_b} = \frac{1}{2}\rho V^2 S_w C_m \bar{c}_w - F_{x_b}\Delta z_b + F_{z_b}\Delta x_b \tag{8.35}$$

$$M_{z_b} = \frac{1}{2}\rho V^2 S_w b_w C_n - F_{y_b}\Delta x_b + F_{x_b}\Delta y_b \tag{8.36}$$

with the propulsive force, F_{P_x}, given according to the model in Eq. (6.7). Equations (8.31)–(8.36) provide the final information required to completely define the linearized state matrix for each aircraft.

8.2 Constructing the Linearized A Matrix

As shown in Eq. (8.20), the linearized state matrix is the Jacobian of the system dynamics with respect to the states evaluated at a given trim condition. The first three rows of this matrix, corresponding to the dynamics of the shifted velocity components evaluated at the trim condition, are given by

$$\begin{Bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \end{Bmatrix} = \frac{g}{W}\begin{Bmatrix} \frac{\partial F_{x_b}}{\partial z} \\ \frac{\partial F_{y_b}}{\partial z} \\ \frac{\partial F_{z_b}}{\partial z} \end{Bmatrix} + \begin{bmatrix} 0 & \hat{x}_6 & -\hat{x}_5 & 0 & -\hat{x}_3 & \hat{x}_2 & 0 & -gc_{\hat{x}_8} \\ -\hat{x}_6 & 0 & \hat{x}_4 & \hat{x}_3 & 0 & -\hat{x}_1 & gc_{\hat{x}_7}c_{\hat{x}_8} & -gs_{\hat{x}_7}s_{\hat{x}_8} \\ \hat{x}_5 & -\hat{x}_4 & 0 & -\hat{x}_2 & \hat{x}_1 & 0 & -gs_{\hat{x}_7}c_{\hat{x}_8} & -gc_{\hat{x}_7}s_{\hat{x}_8} \end{bmatrix} \tag{8.37}$$

Next, the dynamics of the body-fixed rotation rates, associated with the next three rows of A, are given by

$$\begin{Bmatrix} \dot{z}_4 \\ \dot{z}_5 \\ \dot{z}_6 \end{Bmatrix} = \mathcal{I}^{-1}\left(\begin{Bmatrix} \frac{\partial M_{x_b}}{\partial z} \\ \frac{\partial M_{y_b}}{\partial z} \\ \frac{\partial M_{z_b}}{\partial z} \end{Bmatrix} + \begin{bmatrix} W_1 & W_2 & W_3 \end{bmatrix}\right) \tag{8.38}$$

where

$$W_1 = 0_{[3\times 3]} \tag{8.39}$$

$$W_2 = \begin{bmatrix} W_{21} & W_{22} & W_{23} \end{bmatrix} \tag{8.40}$$

and

$$W_3 = 0_{[3\times 2]} \tag{8.41}$$

The sub-matrices of W_2 are given by

$$W_{21} = \begin{Bmatrix} I_{xz_b}\hat{x}_5 - I_{xy_b}\hat{x}_6 \\ (I_{zz_b} - I_{xx_b})\hat{x}_6 - 2I_{xz_b}\hat{x}_4 - I_{yz_b}\hat{x}_5 \\ (I_{xx_b} - I_{yy_b})\hat{x}_5 + 2I_{xy_b}\hat{x}_4 + I_{yz_b}\hat{x}_6 \end{Bmatrix}, W_{22} = \begin{Bmatrix} (I_{yy_b} - I_{zz_b})\hat{x}_6 - 2I_{yz_b}\hat{x}_5 + I_{xz_b}\hat{x}_4 \\ I_{xy_b}\hat{x}_6 - I_{yz_b}\hat{x}_4 \\ (I_{xx_b} - I_{yy_b})\hat{x}_4 - 2I_{xy_b}\hat{x}_5 - I_{xz_b}\hat{x}_6 \end{Bmatrix}$$

$$W_{23} = \begin{Bmatrix} (I_{yy_b} - I_{zz_b})\hat{x}_5 + 2I_{yz_b}\hat{x}_6 - I_{xy_b}\hat{x}_4 \\ (I_{zz_b} - I_{xx_b})\hat{x}_4 + 2I_{xz_b}\hat{x}_6 + I_{xy_b}\hat{x}_5 \\ I_{yz_b}\hat{x}_4 - I_{xz_b}\hat{x}_5 \end{Bmatrix} \tag{8.42}$$

Lastly, the dynamics of the orientation states make up the last two rows of the A matrix, and are specified

$$\begin{Bmatrix} \dot{z}_7 \\ \dot{z}_8 \end{Bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & s_{\hat{x}_7} t_{\hat{x}_8} & c_{\hat{x}_7} t_{\hat{x}_8} & t_{\hat{x}_8}\left(c_{\hat{x}_7}\hat{x}_5 - s_{\hat{x}_7}\hat{x}_6\right) & s_{\hat{x}_7}/c^2_{\hat{x}_8}\hat{x}_5 + c_{\hat{x}_7}/c^2_{\hat{x}_8}\hat{x}_6 \\ 0 & 0 & 0 & 0 & c_{\hat{x}_7} & -s_{\hat{x}_7} & -s_{\hat{x}_7}\hat{x}_5 - c_{\hat{x}_7}\hat{x}_6 & 0 \end{bmatrix} \tag{8.43}$$

Since the aerodynamic forces and moments for the baseline aircraft and the BIRE are closed-form relationships, the above equations represent a closed-form solution for the matrix A linearized about the trim condition $\hat{x}$. Equation (8.30) can be solved by knowing only the trim solution of the aircraft; however, Eqs. (8.25) and (8.26) require knowledge of the aerodynamic force and moment derivatives of each aircraft. The aerodynamic forces and moments for the baseline and BIRE aircraft differ substantially according to the models presented in Chapter 4. Therefore the derivatives of the forces and moments with respect to the shifted state z will be derived alongside the derivatives of the aerodynamic forces and moments from each aircraft.

8.2.1 Aerodynamic Force and Moment State Derivatives

The derivative of the aerodynamic forces given in Eqs. (8.31)–(8.33) with respect to the states z and evaluated at the trim condition are given as

$$\frac{\partial F_{x_b}}{\partial z} = \frac{1}{2}\rho V^2 S_w \frac{\partial C_X}{\partial z} + \rho V S_w C_X \frac{\partial V}{\partial z} + \frac{\partial F_{P_x}}{\partial z} \tag{8.44}$$

$$\frac{\partial F_{y_b}}{\partial z} = \frac{1}{2}\rho V^2 S_w \frac{\partial C_Y}{\partial z} + \rho V S_w C_Y \frac{\partial V}{\partial z} \tag{8.45}$$

$$\frac{\partial F_{z_b}}{\partial z} = \frac{1}{2}\rho V^2 S_w \frac{\partial C_Z}{\partial z} + \rho V S_w C_Z \frac{\partial V}{\partial z} \tag{8.46}$$

Likewise, the aerodynamic moment derivatives evaluated at trim are calculated by

$$\frac{\partial M_{x_b}}{\partial z} = \frac{1}{2}\rho V^2 S_w b_w \frac{\partial C_\ell}{\partial z} + \rho V S_w b_w C_\ell \frac{\partial V}{\partial z} - \frac{\partial F_{z_b}}{\partial z}\Delta y + \frac{\partial F_{y_b}}{\partial z}\Delta z \tag{8.47}$$

$$\frac{\partial M_{y_b}}{\partial z} = \frac{1}{2}\rho V^2 S_w \bar{c}_w \frac{\partial C_m}{\partial z} + \rho V S_w \bar{c}_w C_m \frac{\partial V}{\partial z} - \frac{\partial F_{z_b}}{\partial z}\Delta x + \frac{\partial F_{x_b}}{\partial z}\Delta z \tag{8.48}$$

$$\frac{\partial M_{z_b}}{\partial z} = \frac{1}{2}\rho V^2 S_w b_w \frac{\partial C_n}{\partial z} + \rho V S_w b_w C_n \frac{\partial V}{\partial z} - \frac{\partial F_{y_b}}{\partial z}\Delta x + \frac{\partial F_{x_b}}{\partial z}\Delta y \tag{8.49}$$

Each of the equations above requires the derivative of the velocity magnitude with respect to the states z, also evaluated at the trim condition. This vector is written ass

$$\frac{\partial V}{\partial z} = \begin{bmatrix} \frac{\hat{x}_1}{\hat{V}} & \frac{\hat{x}_2}{\hat{V}} & \frac{\hat{x}_3}{\hat{V}} & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.50}$$

where

$$\hat{V} = \sqrt{\hat{x}_1^2 + \hat{x}_2^2 + \hat{x}_3^2} \tag{8.51}$$

The relationship between the aerodynamic x-, y-, and z-force coefficients and the lift, side force, and drag coefficients used in the aerodynamic model derived in Chapter 4 are given by

$$C_X = -\left(C_D c_\alpha c_\beta + C_S c_\alpha s_\beta - C_L s_\alpha\right) \tag{8.52}$$

$$C_Y = C_S c_\beta - C_D s_\beta \tag{8.53}$$

and

$$C_Z = -\left(C_D s_\alpha c_\beta + C_S s_\alpha s_\beta + C_L c_\alpha\right) \tag{8.54}$$

where α and β are given in terms of the shifted states z and evaluated at the trim condition as

$$\alpha = \tan^{-1}\left(\frac{z_3 + \hat{x}_3}{z_1 + \hat{x}_1}\right) \tag{8.55}$$

and

$$\beta = \sin^{-1}\left(\frac{z_2 + \hat{x}_2}{\hat{V}}\right) \tag{8.56}$$

Using this information, the derivatives of the aerodynamic x-, y-, and z-force coefficients with respect to the states z are given as

$$\begin{aligned}\frac{\partial C_X}{\partial z} = & -\frac{\partial C_D}{\partial z} c_\alpha c_\beta + C_D s_\alpha c_\beta \frac{\partial \alpha}{\partial z} + C_D c_\alpha s_\beta \frac{\partial \beta}{\partial z} \\ & - \frac{\partial C_S}{\partial z} c_\alpha s_\beta + C_S s_\alpha s_\beta \frac{\partial \alpha}{\partial z} - C_S c_\alpha c_\beta \frac{\partial \beta}{\partial z} \\ & + \frac{\partial C_L}{\partial z} s_\alpha + C_L c_\alpha \frac{\partial \alpha}{\partial z}\end{aligned} \tag{8.57}$$

$$\frac{\partial C_Y}{\partial z} = \frac{\partial C_S}{\partial z} c_\beta - C_S s_\beta \frac{\partial \beta}{\partial z} - \frac{\partial C_D}{\partial z} s_\beta - C_D c_\beta \frac{\partial \beta}{\partial z} \tag{8.58}$$

and

$$\begin{aligned}\frac{\partial C_Z}{\partial z} = & -\frac{\partial C_D}{\partial z} s_\alpha c_\beta - C_D c_\alpha c_\beta \frac{\partial \alpha}{\partial z} + C_D s_\alpha s_\beta \frac{\partial \beta}{\partial z} \\ & - \frac{\partial C_S}{\partial z} s_\alpha s_\beta - C_S c_\alpha s_\beta \frac{\partial \alpha}{\partial z} - C_S s_\alpha c_\beta \frac{\partial \beta}{\partial z} \\ & - \frac{\partial C_L}{\partial z} c_\alpha + C_L s_\alpha \frac{\partial \alpha}{\partial z}\end{aligned} \tag{8.59}$$

where

$$\frac{\partial \alpha}{\partial z} = \begin{bmatrix} -\frac{\hat{x}_3}{\hat{x}_1^2 + \hat{x}_3^2} & 0 & \frac{\hat{x}_1}{\hat{x}_1^2 + \hat{x}_3^2} & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.60}$$

and

$$\frac{\partial \beta}{\partial z} = \begin{bmatrix} -\frac{\hat{x}_2 \hat{x}_1}{\hat{V}^2 \sqrt{\hat{x}_1^2 + \hat{x}_3^2}} & \frac{\sqrt{\hat{x}_1^2 + \hat{x}_3^2}}{\hat{V}^2} & -\frac{\hat{x}_2 \hat{x}_3}{\hat{V}^2 \sqrt{\hat{x}_1^2 + \hat{x}_3^2}} & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.61}$$

The derivative of the propulsive force with respect to the shifted states z, $\frac{\partial F_{P_x}}{\partial z}$ in Eq. (8.44), can be determined by referring to Eqs. (6.5)–(6.7). In these equations, the only state variables are related to the velocity, V. Thus, the derivative of the propulsive force with respect to the shifted states evaluated at trim is

$$\frac{\partial F_{P_x}}{\partial z} = \begin{cases} \frac{\partial T_{\text{idle}}}{\partial z} + \left(\frac{\partial T_{\text{mil}}}{\partial z} - \frac{\partial T_{\text{idle}}}{\partial z}\right)\frac{P_1}{50} & , P_1 < 50 \\ \frac{\partial T_{\text{mil}}}{\partial z} + \left(\frac{\partial T_{\text{max}}}{\partial z} - \frac{\partial T_{\text{mil}}}{\partial z}\right)\frac{P_1 - 50}{50} & , P_1 \geq 50 \end{cases} \tag{8.62}$$

where

$$\frac{\partial T}{\partial z} = \left(\frac{\rho}{\rho_0}\right)^a \left(T_1 \frac{\partial V}{\partial z} + 2T_2 \hat{V} \frac{\partial V}{\partial z}\right) \tag{8.63}$$

with the coefficients given as a function of altitude according to Table 6.2.

The only unknowns remaining at this point in the derivation of the linearized state matrix, A, are the derivatives of the aerodynamic coefficients in the wind system in Eqs. (8.57)–(8.59). These derivatives, of course, vary between the baseline aircraft and the BIRE aircraft according to the models presented in Chapter 4. First, these derivatives will be defined for the baseline aircraft using Eqs. (4.60)–(4.65) and then Eqs. (4.66)–(4.71) will be used to derive the corresponding derivatives of the BIRE aircraft.

Baseline Aircraft

Since the coefficients in the baseline aircraft aerodynamic model are each constant, the derivatives of the aerodynamic forces and moments are rather straight-forward to calculate. The derivative of the lift coefficient given in Eq. (4.60) with respect to the states z and evaluated at trim is given by

$$\frac{\partial C_L}{\partial z} = \frac{\partial C_{L_1}}{\partial z} + C_{L,\overline{q}} \frac{\partial \overline{q}}{\partial z} \tag{8.64}$$

where

$$\frac{\partial C_{L_1}}{\partial z} = C_{L,\alpha} \frac{\partial \alpha}{\partial z} \tag{8.65}$$

$$\frac{\partial \overline{q}}{\partial z} = \frac{\partial q}{\partial z} \frac{\overline{c}_w}{2\hat{V}} + \frac{\partial V^{-1}}{\partial z} \frac{\overline{c}_w \hat{x}_5}{2} \tag{8.66}$$

and

$$\frac{\partial q}{\partial z} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{8.67}$$

with C_{L_1} defined in Eq. (4.50). Each of these, of course, is also evaluated at the trim condition. The derivative of the reciprocal of the velocity, used in Eq. (8.66), is evaluated according to

$$\frac{\partial V^{-1}}{\partial z} = \begin{bmatrix} -\frac{\hat{x}_1}{\hat{V}^3} & -\frac{\hat{x}_2}{\hat{V}^3} & -\frac{\hat{x}_3}{\hat{V}^3} & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.68}$$

This quantity will be used repeatedly when defining the derivatives of the other two nondimensional rotation rates.

Likewise, the derivative of the side force coefficient given in Eq. (4.61) with respect to the states z is

$$\frac{\partial C_S}{\partial z} = \frac{\partial C_{S_1}}{\partial z} + C_{S,L\overline{p}} \frac{\partial C_{L_1}}{\partial z} \hat{\overline{p}} + (C_{S,L\overline{p}} C_{L_1} + C_{S,\overline{p}}) \frac{\partial \overline{p}}{\partial z} + C_{S,\overline{r}} \frac{\partial \overline{r}}{\partial z} \tag{8.69}$$

where the derivative of C_{S_1}, given in Eq. (4.51), with respect to the states for the baseline aircraft is

$$\frac{\partial C_{S_1}}{\partial z} = C_{S,\beta} \frac{\partial \beta}{\partial z} \tag{8.70}$$

and the nondimensional roll rate evaluated at trim given in terms of the states as

$$\hat{\overline{p}} = \frac{b_w \hat{x}_4}{2\hat{V}} \tag{8.71}$$

The derivative of the nondimensional roll rate with the shifted states evaluated at trim is given by

$$\frac{\partial \overline{p}}{\partial z} = \frac{\partial p}{\partial z} \frac{b_w}{2\hat{V}} + \frac{\partial V^{-1}}{\partial z} \frac{b_w \hat{x}_4}{2} \tag{8.72}$$

where

$$\frac{\partial p}{\partial z} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.73}$$

Finally, the derivative of the nondimensional yaw rate with respect to z is evaluated

$$\frac{\partial \overline{r}}{\partial z} = \frac{\partial r}{\partial z}\frac{b_w}{2\hat{V}} + \frac{\partial V^{-1}}{\partial z}\frac{b_w \hat{x}_6}{2} \tag{8.74}$$

with

$$\frac{\partial r}{\partial z} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{8.75}$$

The last aerodynamic coefficient derivative for the baseline aircraft is the drag coefficient, given in Eq. (4.62). Its derivative with respect to the states z is

$$\begin{aligned}\frac{\partial C_D}{\partial z} &= C_{D,L}\frac{\partial C_{L_1}}{\partial z} + C_{D,L^2}\frac{\partial C_{L_1}^2}{\partial z} + C_{D,S^2}\frac{\partial C_{S_1}^2}{\partial z} + C_{D,S\overline{p}}\frac{\partial C_{S_1}}{\partial z}\hat{\overline{p}} + C_{D,S\overline{p}}C_{S_1}\frac{\partial \overline{p}}{\partial z} \\ &+ \left(C_{D,L^2\overline{q}}\frac{\partial C_{L_1}^2}{\partial z} + C_{D,L\overline{q}}\frac{\partial C_{L_1}}{\partial z}\right)\hat{\overline{q}} + \left(C_{D,L^2\overline{q}}C_{L_1}^2 + C_{D,L\overline{q}}C_{L_1} + C_{D,\overline{q}}\right)\frac{\partial \overline{q}}{\partial z} \\ &+ C_{D,S\overline{r}}\frac{\partial C_{S_1}}{\partial z}\hat{\overline{r}} + C_{D,S\overline{r}}C_{S_1}\frac{\partial \overline{r}}{\partial z} + C_{D,S\delta_a}\frac{\partial C_{S_1}}{\partial z}\hat{\overline{u}}_2 + C_{D,L\delta_e}\frac{\partial C_{L_1}}{\partial z}\hat{\overline{u}}_3 + C_{D,S\delta_r}\frac{\partial C_{S_1}}{\partial z}\hat{\overline{u}}_4\end{aligned} \tag{8.76}$$

where

$$\frac{\partial C_{L_1}^2}{\partial z} = 2C_{L_1}\big|_{(\hat{z},\hat{u})}\frac{\partial C_{L_1}}{\partial z} \tag{8.77}$$

$$\frac{\partial C_{S_1}^2}{\partial z} = 2C_{S_1}\big|_{(\hat{z},\hat{u})}\frac{\partial C_{S_1}}{\partial z} \tag{8.78}$$

$$\hat{\overline{q}} = \frac{\overline{c}_w \hat{x}_5}{2\hat{V}} \tag{8.79}$$

and

$$\hat{\overline{r}} = \frac{b_w \hat{x}_6}{2\hat{V}} \tag{8.80}$$

Note that, in this case, an unfortunate use of nomenclature requires that the hat notation not be used with coefficients, such as C_{L_1} and C_{S_1}, so as to not create confusion with the BIRE coefficients of the same name.

Each of the preceding definitions can be used in succession to build-up the aerodynamic force derivatives of the baseline aircraft. That is, Eqs. (8.64), (8.69), and (8.76) can be used in Eqs. (8.57)–(8.59) to evaluated the derivative of the coefficients in the body-fixed frame.

Then, the coefficients in the body-fixed frame can be substituted into Eqs. (8.44)–(8.46), which are then used to evaluate Eq. (8.25). Recall that this constitutes the first three rows of the linearized state matrix, A.

To find the next three rows of the state matrix, the derivatives of the aerodynamic moments with respect to the states z are required. They can be composed in a similar procedure as that given for the aerodynamic force derivatives. Beginning with the rolling moment coefficient given in Eq. (4.63), its derivative evaluated at the trim condition is

$$\frac{\partial C_\ell}{\partial z} = C_{\ell,\beta}\frac{\partial \beta}{\partial z} + C_{\ell,\overline{p}}\frac{\partial \overline{p}}{\partial z} + C_{\ell,L\overline{r}}\frac{\partial C_{L_1}}{\partial z}\hat{\overline{r}} + (C_{\ell,L\overline{r}}C_{L_1} + C_{\ell,\overline{r}})\frac{\partial \overline{r}}{\partial z} \tag{8.81}$$

The pitching moment derivative, evaluated at the trim condition, is given by differentiating Eq. (4.64) to find

$$\frac{\partial C_m}{\partial z} = C_{m,\alpha}\frac{\partial \alpha}{\partial z} + C_{m,\overline{q}}\frac{\partial \overline{q}}{\partial z} \tag{8.82}$$

Lastly, the derivative of the yawing moment given in Eq. (4.65) with respect to the states z is

$$\frac{\partial C_n}{\partial z} = C_{n,\beta}\frac{\partial \beta}{\partial z} + C_{n,L\overline{p}}\frac{\partial C_{L_1}}{\partial z}\hat{\overline{p}} + (C_{n,L\overline{p}}C_{L_1} + C_{n,\overline{p}})\frac{\partial \overline{p}}{\partial z} + C_{n,\overline{r}}\frac{\partial \overline{r}}{\partial z} + C_{n,L\delta_a}\frac{\partial C_{L_1}}{\partial z}\hat{\hat{u}}_2 \tag{8.83}$$

Equations (8.81)–(8.83) can be used in Eqs. (8.47)–(8.49) to solve for rows 4 through 6 in the linearized state matrix, given in Eq. (8.26), for the baseline aircraft.

BIRE Variant

With the BIRE possessing a different aerodynamic model, the derivative of the lift, drag, side force, and aerodynamic moments with respect to the states z differ from those presented for the baseline aircraft. Using the lift force coefficient given in Eq. (4.66), its derivative with respect to the shifted states at trim is

$$\frac{\partial C_L}{\partial z} = \frac{\partial \hat{C}_{L_1}}{\partial z} + \hat{C}_{L,\beta}\frac{\partial \beta}{\partial z} + \hat{C}_{L,\overline{p}}\frac{\partial \overline{p}}{\partial z} + \hat{C}_{L,\overline{q}}\frac{\partial \overline{q}}{\partial z} + \hat{C}_{L,\overline{r}}\frac{\partial \overline{r}}{\partial z} \tag{8.84}$$

with

$$\frac{\partial \hat{C}_{L_1}}{\partial z} = \hat{C}_{L,\alpha} \frac{\partial \alpha}{\partial z} \tag{8.85}$$

and $\hat{C}_{L_1}$ as shown in Eq. (4.72). Note that each BIRE aerodynamic coefficient in Eqs. (8.84) and (8.85), as well as subsequent BIRE aerodynamic coefficients, are evaluated at the trim BIRE rotation angle, $\hat{\delta_B} = \hat{\bar{u}}_4$. The derivatives of the dimensionless body-fixed rotation rates and aerodynamic angles are equivalent to those defined in Eqs. (8.60), (8.61), (8.66), (8.72), and (8.74).

The derivative of the side force coefficient in Eq. (4.67) with respect to the states z and evaluated at trim is given by

$$\frac{\partial C_S}{\partial z} = \hat{C}_{S,\alpha} \frac{\partial \alpha}{\partial z} + \frac{\partial \hat{C}_{S_1}}{\partial z} + \hat{C}_{S,L\overline{p}} \frac{\partial \hat{C}_{L_1}}{\partial z} \hat{\overline{p}} + \left(\hat{C}_{S,L\overline{p}} \hat{C}_{L_1} + \hat{C}_{S,\overline{p}} \right) \frac{\partial \overline{p}}{\partial z} + \hat{C}_{S,\overline{q}} \frac{\partial \overline{q}}{\partial z} + \hat{C}_{S,\overline{r}} \frac{\partial \overline{r}}{\partial z} \tag{8.86}$$

where $\hat{C}_{S_1}$ is defined in Eq. (4.73) and

$$\frac{\partial \hat{C}_{S_1}}{\partial z} = \hat{C}_{S,\beta} \frac{\partial \beta}{\partial z} \tag{8.87}$$

Finally, the derivative of the drag coefficient in Eq. (4.68) is given by

$$\begin{aligned} \frac{\partial C_D}{\partial z} = {} & \hat{C}_{D,L} \frac{\partial \hat{C}_{L_1}}{\partial z} + \hat{C}_{D,L^2} \frac{\partial \hat{C}^2_{L_1}}{\partial z} + \hat{C}_{D,S} \frac{\partial \hat{C}_{S_1}}{\partial z} + \hat{C}_{D,S^2} \frac{\partial \hat{C}^2_{S_1}}{\partial z} + \hat{C}_{D,S\overline{p}} \frac{\partial \hat{C}_{S_1}}{\partial z} \hat{\overline{p}} \\ & + \left(\hat{C}_{D,S\overline{p}} \hat{C}_{S_1} + \hat{C}_{D,\overline{p}} \right) \frac{\partial \overline{p}}{\partial z} + \left(\hat{C}_{D,L^2\overline{q}} \frac{\partial \hat{C}^2_{L_1}}{\partial z} + \hat{C}_{D,L\overline{q}} \frac{\partial \hat{C}_{L_1}}{\partial z} \right) \hat{\overline{q}} \\ & + \left(\hat{C}_{D,L^2\overline{q}} \hat{C}^2_{L_1} + \hat{C}_{D,L\overline{q}} \hat{C}_{L_1} + \hat{C}_{D,\overline{q}} \right) \frac{\partial \overline{q}}{\partial z} + \hat{C}_{D,S\overline{r}} \frac{\partial \hat{C}_{S_1}}{\partial z} \hat{\overline{r}} \\ & + \left(\hat{C}_{D,S\overline{r}} \hat{C}_{S_1} + \hat{C}_{D,\overline{r}} \right) \frac{\partial \overline{r}}{\partial z} + \hat{C}_{D,S\delta_a} \frac{\partial \hat{C}_{S_1}}{\partial z} \hat{\bar{u}}_2 + \hat{C}_{D,L\delta_e} \frac{\partial \hat{C}_{L_1}}{\partial z} \hat{\bar{u}}_3 \end{aligned} \tag{8.88}$$

with

$$\frac{\partial \hat{C}^2_{L_1}}{\partial z} = 2 \hat{C}_{L_1} \big|_{(\hat{z},\hat{u})} \frac{\partial \hat{C}_{L_1}}{\partial z} \tag{8.89}$$

and

$$\frac{\partial \hat{C}^2_{S_1}}{\partial z} = 2 \hat{C}_{S_1} \big|_{(\hat{z},\hat{u})} \frac{\partial \hat{C}_{S_1}}{\partial z} \tag{8.90}$$

Again, these equations are made explicit in terms of the evaluation at the trim state to avoid an abuse of notation and any subsequent confusion. The first three rows of the linearized state matrix, given in Eq. (8.25), can be produced using Eqs. (8.84)–(8.90).

The derivatives of the aerodynamic moments are similarly derived from Eqs. (4.69)–(4.71) as

$$\frac{\partial C_\ell}{\partial z} = \hat{C}_{\ell,\alpha}\frac{\partial \alpha}{\partial z} + \hat{C}_{\ell,\beta}\frac{\partial \beta}{\partial z} + \hat{C}_{\ell,\overline{p}}\frac{\partial \overline{p}}{\partial z} + \hat{C}_{\ell,\overline{q}}\frac{\partial \overline{q}}{\partial z} + \hat{C}_{\ell,L\overline{r}}\frac{\partial \hat{C}_{L_1}}{\partial z}\hat{\overline{r}} + \left(\hat{C}_{\ell,L\overline{r}}\hat{C}_{L_1} + \hat{C}_{\ell,\overline{r}}\right)\frac{\partial \overline{r}}{\partial z} \quad (8.91)$$

$$\frac{\partial C_m}{\partial z} = \hat{C}_{m,\alpha}\frac{\partial \alpha}{\partial z} + \hat{C}_{m,\beta}\frac{\partial \beta}{\partial z} + \hat{C}_{m,\overline{p}}\frac{\partial \overline{p}}{\partial z} + \hat{C}_{m,\overline{q}}\frac{\partial \overline{q}}{\partial z} + \hat{C}_{m,\overline{r}}\frac{\partial \overline{r}}{\partial z} \quad (8.92)$$

and

$$\begin{aligned}\frac{\partial C_n}{\partial z} = \hat{C}_{n,\alpha}\frac{\partial \alpha}{\partial z} + \hat{C}_{n,\beta}\frac{\partial \beta}{\partial z} + \hat{C}_{L,\overline{p}}\frac{\partial \hat{C}_{L_1}}{\partial z}\hat{\overline{p}} + \left(\hat{C}_{n,L\overline{p}}\hat{C}_{L_1} + \hat{C}_{n,\overline{p}}\right)\frac{\partial \overline{p}}{\partial z} + \hat{C}_{n,\overline{q}}\frac{\partial \overline{q}}{\partial z} \\ + \hat{C}_{n,\overline{r}}\frac{\partial \overline{r}}{\partial z} + \hat{C}_{n,L\delta_a}\frac{\partial \hat{C}_{L_1}}{\partial z}\hat{\tilde{u}}_2\end{aligned} \quad (8.93)$$

These equations can be used in combination with Eqs. (8.84)–(8.88) to produce rows 4–6 of the linearized state matrix, given by Eq. (8.26).

8.2.2 Example Case

As an example, the baseline aircraft can be trimmed in steady level flight with the center of gravity in its nominal position at the air combat maneuver condition (C2) given in Table 6.1. The resulting states of the aircraft when trimmed are given by

$$\hat{x} = \begin{bmatrix} 633.6030 & 0 & 32.0539 & 0 & 0 & 0 & 0 & 0.0505 \end{bmatrix}^T \quad (8.94)$$

and the control inputs required to maintain trim are

$$\hat{\tilde{u}} = \begin{bmatrix} 0 & -0.0013 & 0 \end{bmatrix}^T \quad (8.95)$$

Based on this trim condition, the baseline aircraft has a linearized state matrix given by

$$A = \begin{bmatrix} -0.0111 & 0 & 0.0515 & 0 & -32.0934 & 0 & 0 & -32.0868 \\ 0 & -0.2063 & 0 & 32.2543 & 0 & -632.1447 & 32.0868 & 0 \\ -0.0605 & 0 & -0.8063 & 0 & 629.3122 & 0 & 0 & -1.6233 \\ 0 & -0.0337 & 0 & -2.1514 & 0 & 0.4615 & 0 & 0 \\ -0.0003 & 0 & 0.0051 & 0 & -0.7927 & 0 & 0 & 0 \\ 0 & 0.0159 & 0 & -0.0433 & 0 & -0.1743 & 0 & 0 \\ 0 & 0 & 0 & 1.0000 & 0 & 0.0506 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0000 & 0 & 0 & 0 \end{bmatrix} \tag{8.96}$$

The stability of the baseline aircraft in this configuration can be determined using the eigenvalues of the state matrix, which are given as

$$\lambda_i = \begin{bmatrix} 0.9983 & -2.5914 & -0.0086 \pm 0.0794j & -0.2058 \pm 3.3325j & -2.1247 & 0.0044 \end{bmatrix} \tag{8.97}$$

Since one eigenvalue has positive real part, the linearized model and the nonlinear model are unstable in this trim condition.

Trimming the BIRE aircraft in the same condition results in the trim state

$$\hat{x} = \begin{bmatrix} 633.5031 & 0 & 33.9723 & 0 & 0 & 0 & 0 & 0.0536 \end{bmatrix}^T \tag{8.98}$$

and control inputs

$$\hat{\tilde{u}} = \begin{bmatrix} 0 & -0.0149 & 0 \end{bmatrix}^T \tag{8.99}$$

Linearizing the shifted BIRE dynamics about this trim condition yields the linearized state matrix

$$A = \begin{bmatrix} -0.0048 & 0 & 0.0526 & 0 & -33.9942 & 0 & 0 & -32.0817 \\ 0 & -0.0393 & 0 & 34.1625 & 0 & -633.5561 & 32.0817 & 0 \\ -0.0598 & 0 & -0.8202 & 0 & 628.9828 & 0 & 0 & -1.7204 \\ 0 & -0.0182 & 0 & -3.1133 & 0 & 0.3401 & 0 & 0 \\ 0.0002 & 0 & -0.0042 & 0 & 0.7691 & 0 & 0 & 0 \\ 0 & -0.0003 & 0 & -0.0295 & 0 & 0.0102 & 0 & 0 \\ 0 & 0 & 0 & 1.0000 & 0 & 0.0536 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0000 & 0 & 0 & 0 \end{bmatrix} \tag{8.100}$$

Again, the stability of this linearized system can be determined using its eigenvalues, which are calculated as

$$\lambda_i = \begin{bmatrix} -0.7942 \pm 1.6317j & -0.0028 \pm 0.0653j & -3.0020 & -0.0732 \pm 0.3241j & 0.0060 \end{bmatrix} \tag{8.101}$$

From the eigenvalues in Eq. (8.101), this system is noted to be unstable.

With an example of the linearized state matrix given for the baseline and BIRE aircraft, the next step is to construct the linearized control matrix, B. The procedure for doing so is very similar to that of the linearized state matrix, the only difference being that derivatives with respect to the control inputs, $\tilde{u}$ are required instead of the derivatives with respect to the shifted states, z. With the linearized control matrix developed, Eq. (8.23) can be used to determine whether each system is completely controllable.

8.3 Constructing the Linearized Control Matrix

Equation (8.21) shows that the linearized control matrix B is the Jacobian of the system dynamics with respect to the control inputs evaluated at the given trim condition. Again, any parameters that are not explicitly shown to be evaluated at the trim point are considered to be evaluated there for ease of notation.

The portion of the linearized control matrix relating to the shifted body-fixed velocities is considerably easier to define at the system dynamics level, since only the aerodynamic forces and moments acting on the aircraft are a function of the control inputs. Therefore, the first three rows of the matrix are given by

$$\frac{\partial}{\partial \tilde{u}} \begin{Bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \end{Bmatrix} = \frac{g}{W} \begin{Bmatrix} \frac{\partial F_{x_b}}{\partial \tilde{u}} \\ \frac{\partial F_{y_b}}{\partial \tilde{u}} \\ \frac{\partial F_{z_b}}{\partial \tilde{u}} \end{Bmatrix} \tag{8.102}$$

This is not the case for the following three rows of the linearized control matrix when analyzing the BIRE aircraft, due to the dependence of the inertia matrix on the BIRE rotation angle. Thus, the definition of the linearized control matrix differs here between the two aircraft. In the baseline aircraft, the inverse of the inertia tensor is constant, and therefore the derivative of these rows with respect to the control inputs are given by

$$\frac{\partial}{\partial \tilde{u}} \begin{Bmatrix} \dot{z}_4 \\ \dot{z}_5 \\ \dot{z}_6 \end{Bmatrix} = \mathcal{I}^{-1} \begin{Bmatrix} \frac{\partial M_{x_b}}{\partial \tilde{u}} \\ \frac{\partial M_{y_b}}{\partial \tilde{u}} \\ \frac{\partial M_{z_b}}{\partial \tilde{u}} \end{Bmatrix} \tag{8.103}$$

For the BIRE variant, the inertia tensor is a function of the control input $\delta_B = \tilde{u}_3$ and therefore the derivative of the inertia tensor must also be calculated. To perform this differentiation, we note that the derivative of an $M \times N$ matrix $P(x)$ with respect to the components x_q of a vector x is given by [132]

$$\frac{\partial P}{\partial x_q} = \begin{bmatrix} \frac{\partial p_{11}}{\partial x_q} & \cdots & \frac{\partial p_{1N}}{\partial x_q} \\ \vdots & \ddots & \vdots \\ \frac{\partial p_{M1}}{\partial x_q} & \cdots & \frac{\partial p_{MN}}{\partial x_q} \end{bmatrix} \tag{8.104}$$

and, by the product differentiation rule for matrices [132], the derivative of the product of an $M \times N$ matrix $P(x)$ and an $N \times L$ matrix $R(x)$, defined as $Q = PR$ with dimension

$M \times L$ can be written as

$$\frac{\partial Q}{\partial x_q} = \frac{\partial P}{\partial x_q} R + P \frac{\partial R}{\partial x_q} \tag{8.105}$$

In the BIRE system, according to the nomenclature in Eqs. (8.104) and (8.105), $\mathcal{I}^{-1} = P \in 3 \times 3$, $\begin{Bmatrix} M_1 & M_2 & M_3 \end{Bmatrix}^T = R \in 3 \times 1$, and therefore $Q = \mathcal{I}^{-1} \begin{Bmatrix} M_1 & M_2 & M_3 \end{Bmatrix}^T \in 3 \times 1$. Taking the derivative with respect to $\tilde{u}$ of this resulting vector gives

$$\frac{\partial}{\partial \tilde{u}} \begin{Bmatrix} \dot{z}_4 \\ \dot{z}_5 \\ \dot{z}_6 \end{Bmatrix} = \frac{\partial}{\partial \tilde{u}} \left(\mathcal{I}^{-1} \begin{Bmatrix} M_1 \\ M_2 \\ M_3 \end{Bmatrix} \right) = \frac{\partial \mathcal{I}^{-1}}{\partial \tilde{u}} \begin{Bmatrix} M_1 \\ M_2 \\ M_3 \end{Bmatrix} + \mathcal{I}^{-1} \begin{Bmatrix} \frac{\partial M_1}{\partial \tilde{u}} \\ \frac{\partial M_2}{\partial \tilde{u}} \\ \frac{\partial M_3}{\partial \tilde{u}} \end{Bmatrix} \tag{8.106}$$

with M_1, M_2, and M_3 given in Eqs. (8.27)–(8.29). Referring to Eq. (8.105), note that the first term on the right-hand side of Eq. (8.106) is equivalent to multiplying the matrices in the first dimension of the tensor $\frac{\partial \mathcal{I}^{-1}}{\partial \tilde{u}} \in \mathbb{R}^{3\times3\times4}$ into the vector $\begin{Bmatrix} M_1 & M_2 & M_3 \end{Bmatrix}^T$. This multiplication produces a matrix of the appropriate dimension in $\mathbb{R}^{3\times4}$ as required. Equation (8.106) is evaluated using the definitions of control derivatives given in the section detailing the BIRE variant.

The derivative of the orientation dynamics of both aircraft with respect to the control inputs are trivially given as

$$\frac{\partial}{\partial \tilde{u}} \begin{Bmatrix} \dot{z}_7 \\ \dot{z}_8 \end{Bmatrix} = 0_{[2\times4]} \tag{8.107}$$

The construction of the linearized control matrix for the baseline aircraft is predominantly focused on defining the control derivatives of the aerodynamic forces and moments. These derivatives are relatively straight-forward for the baseline aircraft, since the aerodynamic model for the baseline aircraft has constant coefficients. The BIRE variant, on the other hand, varies its coefficients periodically with the BIRE rotation angle, making its derivatives slightly more cumbersome to evaluate. In addition, the BIRE variant varies its inertial properties with BIRE rotation angle, as mentioned previously, and therefore derivatives relating to those parameters must also be considered.

8.3.1 Aerodynamic Force and Moment Control Derivatives

The definitions of the aerodynamic forces and moments acting on the aircraft have been given in Eqs. (8.31)–(8.36). The derivative of these forces and moments with respect to the control inputs $\tilde{u}$ are

$$\frac{\partial F_{x_b}}{\partial \tilde{u}} = \frac{1}{2}\rho V^2 S_w \frac{\partial C_X}{\partial \tilde{u}} + \frac{\partial F_{P_x}}{\partial \tilde{u}} \tag{8.108}$$

$$\frac{\partial F_{y_b}}{\partial \tilde{u}} = \frac{1}{2}\rho V^2 S_w \frac{\partial C_Y}{\partial \tilde{u}} \tag{8.109}$$

$$\frac{\partial F_{z_b}}{\partial \tilde{u}} = \frac{1}{2}\rho V^2 S_w \frac{\partial C_Z}{\partial \tilde{u}} \tag{8.110}$$

$$\frac{\partial M_{x_b}}{\partial \tilde{u}} = \frac{1}{2}\rho V^2 S_w b_w \frac{\partial C_\ell}{\partial \tilde{u}} - \frac{\partial F_{z_b}}{\partial \tilde{u}}\Delta y + \frac{\partial F_{y_b}}{\partial \tilde{u}}\Delta z \tag{8.111}$$

$$\frac{\partial M_{y_b}}{\partial \tilde{u}} = \frac{1}{2}\rho V^2 S_w \bar{c}_w \frac{\partial C_m}{\partial \tilde{u}} - \frac{\partial F_{z_b}}{\partial \tilde{u}}\Delta x + \frac{\partial F_{x_b}}{\partial \tilde{u}}\Delta z \tag{8.112}$$

and

$$\frac{\partial M_{z_b}}{\partial \tilde{u}} = \frac{1}{2}\rho V^2 S_w b_w \frac{\partial C_n}{\partial \tilde{u}} - \frac{\partial F_{y_b}}{\partial \tilde{u}}\Delta x + \frac{\partial F_{x_b}}{\partial \tilde{u}}\Delta y \tag{8.113}$$

Again, the evaluation of these coefficients requires derivatives of aerodynamic force and moment coefficients in the body-fixed frame. Since the throttle is not included in the control inputs available to either aircraft, its contribution can be given as

$$\frac{\partial F_{P_x}}{\partial \tilde{u}} = 0 \tag{8.114}$$

The derivative of the body-fixed aerodynamic forces with respect to the control inputs can be evaluated by taking the derivative of Eqs. (8.52)–(8.54), which yields

$$\frac{\partial C_X}{\partial \tilde{u}} = -\left(\frac{\partial C_D}{\partial \tilde{u}} c_\alpha c_\beta + \frac{\partial C_S}{\partial \tilde{u}} c_\alpha s_\beta - \frac{\partial C_L}{\partial \tilde{u}} s_\alpha\right) \tag{8.115}$$

$$\frac{\partial C_Y}{\partial \tilde{u}} = \frac{\partial C_S}{\partial \tilde{u}} c_\beta - \frac{\partial C_D}{\partial \tilde{u}} s_\beta \tag{8.116}$$

and

$$\frac{\partial C_Z}{\partial \tilde{u}} = -\left(\frac{\partial C_D}{\partial \tilde{u}} s_\alpha c_\beta + \frac{\partial C_S}{\partial \tilde{u}} s_\alpha s_\beta + \frac{\partial C_L}{\partial \tilde{u}} c_\alpha\right) \tag{8.117}$$

Completely defining these derivatives, again, requires finding the derivative of each wind-coordinate-system force in the baseline and BIRE aerodynamic models. The derivatives of the wind-system moments with respect to the control inputs are required to evaluate Eqs. (8.112)–(8.113) as well. Thus, these will be defined for each aircraft using the appropriate aerodynamic model.

Baseline Aircraft

The model for the lift coefficient of the baseline aircraft is given in Eq. (4.60) and is a function only of the stabilator deflection, δ_e. Its derivative with respect to the control inputs is therefore given by

$$\frac{\partial C_L}{\partial \tilde{u}} = C_{L,\delta_e} \frac{\partial \delta_e}{\partial \tilde{u}} \tag{8.118}$$

where

$$\frac{\partial \delta_e}{\partial \tilde{u}} = \begin{Bmatrix} 0 & 1 & 0 \end{Bmatrix} \tag{8.119}$$

which is again only used to assign the derivative to its appropriate column in the linearized control matrix.

The side force of the baseline aircraft is a function of both the aileron deflection and the rudder deflection, δ_a and δ_r, respectively. Referring to Eq. (4.61), the derivative of the side force with respect to the control inputs is

$$\frac{\partial C_S}{\partial \tilde{u}} = C_{S,\delta_a} \frac{\partial \delta_a}{\partial \tilde{u}} + C_{S,\delta_r} \frac{\partial \delta_r}{\partial \tilde{u}} \tag{8.120}$$

where

$$\frac{\partial \delta_a}{\partial \tilde{u}} = \begin{Bmatrix} 1 & 0 & 0 \end{Bmatrix} \tag{8.121}$$

and

$$\frac{\partial \delta_r}{\partial \tilde{u}} = \begin{Bmatrix} 0 & 0 & 1 \end{Bmatrix} \tag{8.122}$$

Last for the aerodynamic forces is the drag coefficient of the baseline aircraft, given in Eq. (4.62), which is a function of each of the control inputs with the exception of the throttle. The derivative of the drag coefficient with respect to the control inputs is

$$\frac{\partial C_D}{\partial \tilde{u}} = C_{D,S\delta_a} C_{S_1}\big|_{(\hat{z},\hat{\tilde{u}})} \frac{\partial \delta_a}{\partial \tilde{u}} + \left(C_{D,L\delta_e} C_{L_1}\big|_{(\hat{z},\hat{\tilde{u}})} + C_{D,\delta_e} \right) \frac{\partial \delta_e}{\partial \tilde{u}} + C_{D,\delta_e^2} \frac{\partial \delta_e^2}{\partial \tilde{u}} + C_{D,S\delta_r} C_{S_1}\big|_{(\hat{z},\hat{\tilde{u}})} \frac{\partial \delta_r}{\partial \tilde{u}} \tag{8.123}$$

with

$$\frac{\partial \delta_e^2}{\partial \tilde{u}} = 2\hat{u}_2 \frac{\partial \delta_e}{\partial \tilde{u}} \tag{8.124}$$

Again, note that avoiding any abuse of notation requires that the hat notation be replaced with an explicit notation depicting the evaluation of C_{L_1} and C_{S_1} at the trim condition $(\hat{z}, \hat{\tilde{u}})$. Equations (8.118), (8.120), and (8.123) can be used in Eqs. (8.115)–(8.117) and finally substituted into the force derivatives in Eqs. (8.108)–(8.110) to compute the first three rows of the linearized control matrix given in Eq. (8.102).

For the baseline aircraft, the derivatives of the moments M_1, M_2, and M_3 given in Eq. (8.103) are given by Eqs. (8.112)–(8.113), respectively. These require the derivatives of the aerodynamic forces with respect to the control inputs, given above, in addition to the control derivative of each of the aerodynamic moments as given in the baseline aerodynamic model. The derivative of the rolling moment coefficient given in Eq. (4.63) with respect to the control inputs is

$$\frac{\partial C_\ell}{\partial \tilde{u}} = C_{\ell,\delta_a} \frac{\partial \delta_a}{\partial \tilde{u}} + C_{\ell,\delta_r} \frac{\partial \delta_r}{\partial \tilde{u}} \tag{8.125}$$

being a function of both the aileron deflection and rudder deflection.

As a function of only the stabilator deflection, the derivative of the pitching moment coefficient in Eq. (4.64) is

$$\frac{\partial C_m}{\partial \tilde{u}} = C_{m,\delta_e} \frac{\partial \delta_e}{\partial \tilde{u}} \tag{8.126}$$

The yawing moment coefficient of the baseline aircraft is, like the rolling moment coefficient, a function of both the aileron and rudder deflections. Thus, its derivative can be calculated from Eq. (4.65) to be

$$\frac{\partial C_n}{\partial \tilde{u}} = \left(C_{n,L\delta_a} C_{L_1} + C_{n,\delta_a}\right) \frac{\partial \delta_a}{\partial \tilde{u}} + C_{n,\delta_r} \frac{\partial \delta_r}{\partial \tilde{u}} \tag{8.127}$$

Using these equations along with the force derivatives in Eqs. (8.112)–(8.113) allows rows 4–6 in the linearized control matrix to be evaluated for the baseline aircraft.

BIRE Variant

For the BIRE variant, the control derivatives in Eqs. (8.108)–(8.113) are more complicated than for the baseline aircraft. Rather than having constant coefficients, each coefficient in the aerodynamic model is a function of the final control input, δ_B. Fortunately, by maintaining a general form for each coefficient, given in Eq. (5.9), a general form for the derivative of these coefficients can be used. This general form is given according to the

$$\frac{\partial \hat{C}_i}{\partial \tilde{u}} = \left[A_i \omega_i \cos\left(\omega_i \delta_B + \varphi_i\right)\right] \frac{\partial \delta_B}{\partial \tilde{u}} \tag{8.128}$$

with

$$\frac{\partial \delta_B}{\partial \tilde{u}} = \begin{Bmatrix} 0 & 0 & 1 \end{Bmatrix} \tag{8.129}$$

The form of the derivative given in Eq. (8.128) can be applied to each of the coefficients in the BIRE aerodynamic model.

The derivative of each force coefficient in the BIRE aerodynamic model can be computed as follows. Beginning with the lift coefficient model given in Eq. (4.66), its derivative with respect to the control inputs is defined as

$$\begin{aligned} \frac{\partial C_L}{\partial \tilde{u}} = \frac{\partial \hat{C}_{L_1}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{L,\beta}}{\partial \tilde{u}} \hat{\beta} + \frac{\partial \hat{C}_{L,\overline{p}}}{\partial \tilde{u}} \hat{\overline{p}} + \frac{\partial \hat{C}_{L,\overline{q}}}{\partial \tilde{u}} \hat{\overline{q}} + \frac{\partial \hat{C}_{L,\overline{r}}}{\partial \tilde{u}} \hat{\overline{r}} \\ + \frac{\partial \hat{C}_{L,\delta_a}}{\partial \tilde{u}} \hat{\tilde{u}}_1 + \hat{C}_{L,\delta_a} \frac{\partial \delta_a}{\partial \tilde{u}} + \frac{\partial \hat{C}_{L,\delta_e}}{\partial \tilde{u}} \hat{\tilde{u}}_2 + \hat{C}_{L,\delta_e} \frac{\partial \delta_e}{\partial \tilde{u}} \end{aligned} \tag{8.130}$$

where the derivative of the pseudo-lift force with respect to the control inputs is

$$\frac{\partial \hat{C}_{L_1}}{\partial \tilde{u}} = \frac{\partial \hat{C}_{L_0}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{L,\alpha}}{\partial \tilde{u}} \hat{\alpha} \tag{8.131}$$

and the angle of attack and sideslip angle at trim are given by

$$\hat{\alpha} = \tan^{-1}\left(\frac{\hat{x}_3}{\hat{x}_1}\right) \tag{8.132}$$

and

$$\hat{\beta} = \sin^{-1}\left(\frac{\hat{x}_2}{\hat{V}}\right) \tag{8.133}$$

The side force coefficient for the BIRE variant given in Eq. (4.67) has its derivative with respect to control inputs given by

$$\begin{aligned} \frac{\partial C_S}{\partial \tilde{u}} = \frac{\partial \hat{C}_{S_1}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{S,\alpha}}{\partial \tilde{u}} \hat{\alpha} + \left(\frac{\partial \hat{C}_{S,L\overline{p}}}{\partial \tilde{u}} \hat{C}_{L_1} + \hat{C}_{S,L\overline{p}} \frac{\partial \hat{C}_{L_1}}{\partial u} + \frac{\partial \hat{C}_{S,\overline{p}}}{\partial \tilde{u}} \right) \hat{\overline{p}} \\ + \frac{\partial \hat{C}_{S,\overline{q}}}{\partial \tilde{u}} \hat{\overline{q}} + \frac{\partial \hat{C}_{S,\overline{r}}}{\partial \tilde{u}} \hat{\overline{r}} + \frac{\partial \hat{C}_{S,\delta_a}}{\partial \tilde{u}} \hat{\tilde{u}}_1 + \hat{C}_{S,\delta_a} \frac{\partial \delta_a}{\partial \tilde{u}} + \frac{\partial \hat{C}_{S,\delta_e}}{\partial \tilde{u}} \hat{\tilde{u}}_2 + \hat{C}_{S,\delta_e} \frac{\partial \delta_e}{\partial \tilde{u}} \end{aligned} \tag{8.134}$$

where

$$\frac{\partial \hat{C}_{S_1}}{\partial \tilde{u}} = \frac{\partial \hat{C}_{S_0}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{S,\beta}}{\partial \tilde{u}} \hat{\beta} \tag{8.135}$$

Finally, the derivative of the drag force coefficient as given in Eq. (4.68) with respect to the control inputs is

$$
\begin{aligned}
\frac{\partial C_D}{\partial \tilde{u}} &= \frac{\partial \hat{C}_{D_0}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{D,L}}{\partial \tilde{u}}\hat{C}_{L_1} + \hat{C}_{D,L}\frac{\partial \hat{C}_{L_1}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{D,L^2}}{\partial \tilde{u}}\hat{C}_{L_1}^2 + \hat{C}_{D,L^2}\frac{\partial \hat{C}_{L_1}^2}{\partial \tilde{u}} \\
&+ \frac{\partial \hat{C}_{D,S}}{\partial \tilde{u}}\hat{C}_{S_1} + \hat{C}_{D,S}\frac{\partial \hat{C}_{S_1}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{D,S^2}}{\partial \tilde{u}}\hat{C}_{S_1}^2 + \hat{C}_{D,S^2}\frac{\partial \hat{C}_{S_1}^2}{\partial \tilde{u}} \\
&+ \left(\frac{\partial \hat{C}_{D,S\overline{p}}}{\partial \tilde{u}}\hat{C}_{S_1} + \hat{C}_{D,S\overline{p}}\frac{\partial \hat{C}_{S_1}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{D,\overline{p}}}{\partial \tilde{u}}\right)\hat{\overline{p}} \\
&+ \left(\frac{\partial \hat{C}_{D,L^2\overline{q}}}{\partial \tilde{u}}\hat{C}_{L_1}^2 + \hat{C}_{D,L^2\overline{q}}\frac{\partial \hat{C}_{L_1}^2}{\partial \tilde{u}} + \frac{\partial \hat{C}_{D,L\overline{q}}}{\partial \tilde{u}}\hat{C}_{L_1} + \hat{C}_{D,L\overline{q}}\frac{\partial \hat{C}_{L_1}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{D,\overline{q}}}{\partial \tilde{u}}\right)\hat{\overline{q}} \\
&+ \left(\frac{\partial \hat{C}_{D,S\overline{r}}}{\partial \tilde{u}}\hat{C}_{S_1} + \hat{C}_{D,S\overline{r}}\frac{\partial \hat{C}_{S_1}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{D,\overline{r}}}{\partial \tilde{u}}\right)\hat{\overline{r}} \\
&+ \left(\frac{\partial \hat{C}_{D,S\delta_a}}{\partial \tilde{u}}\hat{C}_{S_1} + \hat{C}_{D,S\delta_a}\frac{\partial \hat{C}_{S_1}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{D,\delta_a}}{\partial \tilde{u}}\right)\hat{u}_1 + \left(\hat{C}_{D,S\delta_a}\hat{C}_{S_1} + \hat{C}_{D,\delta_a}\right)\frac{\partial \delta_a}{\partial \tilde{u}} \\
&+ \left(\frac{\partial \hat{C}_{D,L\delta_e}}{\partial \tilde{u}}\hat{C}_{L_1} + \hat{C}_{D,L\delta_e}\frac{\partial \hat{C}_{L_1}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{D,\delta_e}}{\partial \tilde{u}}\right)\hat{u}_2 + \left(\hat{C}_{D,L\delta_e}\hat{C}_{L_1} + \hat{C}_{D,\delta_e}\right)\frac{\partial \delta_e}{\partial \tilde{u}} \\
&+ \frac{\partial \hat{C}_{D,\delta_e^2}}{\partial \tilde{u}}\hat{u}_2^2 + \hat{C}_{D,\delta_e^2}\frac{\partial \delta_e^2}{\partial \tilde{u}}
\end{aligned}
\tag{8.136}
$$

where

$$
\frac{\partial \hat{C}_{L_1}^2}{\partial \tilde{u}} = 2\hat{C}_{L_1}|_{(\hat{z},\hat{\hat{u}})}\frac{\partial \hat{C}_{L_1}}{\partial \tilde{u}} \tag{8.137}
$$

and

$$
\frac{\partial \hat{C}_{S_1}^2}{\partial \tilde{u}} = 2\hat{C}_{S_1}|_{(\hat{z},\hat{\hat{u}})}\frac{\partial \hat{C}_{S_1}}{\partial \tilde{u}} \tag{8.138}
$$

The derivatives of the aerodynamic moments for the BIRE aircraft, given by Eqs. (4.69)–(4.71), are

$$
\begin{aligned}
\frac{\partial C_\ell}{\partial \tilde{u}} &= \frac{\partial \hat{C}_{\ell_0}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{\ell,\alpha}}{\partial \tilde{u}}\hat{\alpha} + \frac{\partial \hat{C}_{\ell,\beta}}{\partial \tilde{u}}\hat{\beta} + \frac{\partial \hat{C}_{\ell,\overline{p}}}{\partial \tilde{u}}\hat{\overline{p}} + \frac{\partial \hat{C}_{\ell,\overline{q}}}{\partial \tilde{u}}\hat{\overline{q}} \\
&+ \left(\frac{\partial \hat{C}_{\ell,L\overline{r}}}{\partial \tilde{u}}\hat{C}_{L_1} + \hat{C}_{\ell,L\overline{r}}\frac{\partial \hat{C}_{L_1}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{\ell,\overline{r}}}{\partial \tilde{u}}\right)\hat{\overline{r}} \\
&+ \frac{\partial \hat{C}_{\ell,\delta_a}}{\partial \tilde{u}}\hat{u}_1 + \hat{C}_{\ell,\delta_a}\frac{\partial \delta_a}{\partial \tilde{u}} + + \frac{\partial \hat{C}_{\ell,\delta_e}}{\partial \tilde{u}}\hat{u}_2 + \hat{C}_{\ell,\delta_e}\frac{\partial \delta_e}{\partial \tilde{u}}
\end{aligned}
\tag{8.139}
$$

$$\frac{\partial C_m}{\partial \tilde{u}} = \frac{\partial \hat{C}_{m_0}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{m,\alpha}}{\partial \tilde{u}}\hat{\alpha} + \frac{\partial \hat{C}_{m,\beta}}{\partial \tilde{u}}\hat{\beta} + \frac{\partial \hat{C}_{m,\overline{p}}}{\partial \tilde{u}}\hat{\overline{p}} + \frac{\partial \hat{C}_{m,\overline{q}}}{\partial \tilde{u}}\hat{\overline{q}} + \frac{\partial \hat{C}_{m,\overline{r}}}{\partial \tilde{u}}\hat{\overline{r}} \\ + \frac{\partial \hat{C}_{m,\delta_a}}{\partial \tilde{u}}\hat{u}_1 + \hat{C}_{m,\delta_a}\frac{\partial \delta_a}{\partial \tilde{u}} + \frac{\partial \hat{C}_{m,\delta_e}}{\partial \tilde{u}}\hat{u}_2 + \hat{C}_{m,\delta_e}\frac{\partial \delta_e}{\partial \tilde{u}} \tag{8.140}$$

and

$$\frac{\partial C_n}{\partial \tilde{u}} = \frac{\partial \hat{C}_{n_0}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{n,\alpha}}{\partial \tilde{u}}\hat{\alpha} + \frac{\partial \hat{C}_{n,\beta}}{\partial \tilde{u}}\hat{\beta} + \left(\frac{\partial \hat{C}_{n,L\overline{p}}}{\partial \tilde{u}}\hat{C}_{L_1} + \hat{C}_{n,L\overline{p}}\frac{\partial \hat{C}_{L_1}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{n,\overline{p}}}{\partial \tilde{u}}\right)\hat{\overline{p}} \\ + \frac{\partial \hat{C}_{n,\overline{q}}}{\partial \tilde{u}}\hat{\overline{q}} + \frac{\partial \hat{C}_{n,\overline{r}}}{\partial \tilde{u}}\hat{\overline{r}} + \left(\frac{\partial \hat{C}_{n,L\delta_a}}{\partial \tilde{u}}\hat{C}_{L_1} + \hat{C}_{n,L\delta_a}\frac{\partial \hat{C}_{L_1}}{\partial \tilde{u}} + \frac{\partial \hat{C}_{n,\delta_a}}{\partial \tilde{u}}\right)\hat{u}_1 \\ + \left(\hat{C}_{n,L\delta_a}\hat{C}_{L_1} + \hat{C}_{n,\delta_a}\right)\frac{\partial \delta_a}{\partial \tilde{u}} + \frac{\partial \hat{C}_{n,\delta_e}}{\partial \tilde{u}}\hat{u}_2 + \hat{C}_{n,\delta_e}\frac{\partial \delta_e}{\partial \tilde{u}} \tag{8.141}$$

To solve for rows 4, 5, and 6 of the linearized control matrix, which are given in Eq. (8.106), the aerodynamic coefficients must not only be determined, but also the derivatives of the inertia tensor.

Since both I_{xx_b} and I_{xz_b} are independent of BIRE rotation angle, their individual contributions to the control derivatives are zero; that is,

$$\frac{\partial I_{xx_b}}{\partial \tilde{u}} = \frac{\partial I_{xz_b}}{\partial \tilde{u}} = 0 \tag{8.142}$$

Differentiating the determinant of the inertia tensor, given in Eq. (8.12), with respect to the control inputs yields

$$\frac{\partial \det \mathcal{I}}{\partial \tilde{u}} = I_{xx_b}\left[\frac{\partial I_{yy_b}}{\partial \tilde{u}}\left(I_{zz_b} - I_{yy_b}\right) - 2I_{yz_b}\frac{\partial I_{yz_b}}{\partial \tilde{u}}\right] - I_{xz_b}^2\frac{\partial I_{yy_b}}{\partial \tilde{u}} \tag{8.143}$$

The derivative of each inertia component in Eq. (8.143) with respect to control inputs can be found by consulting Table 3.8 to find

$$\frac{\partial I_{yy_b}}{\partial \tilde{u}} = 322\sin\left(2\delta_B\right)\frac{\partial \delta_B}{\partial \tilde{u}} \tag{8.144}$$

$$\frac{\partial I_{zz_b}}{\partial \tilde{u}} = -322\sin\left(2\delta_B\right)\frac{\partial \delta_B}{\partial \tilde{u}} = -\frac{\partial I_{yy_b}}{\partial \tilde{u}} \tag{8.145}$$

and

$$\frac{\partial I_{yz_b}}{\partial \tilde{u}} = -322\frac{\sin(2\delta_B)\cos(2\delta_B)}{|\sin(2\delta_B)|}\frac{\partial \delta_B}{\partial \tilde{u}} \tag{8.146}$$

Note that this last derivative is undefined at $\delta_B = 0°$ and $\delta_B = \pm 180°$ and will be set to zero at any of these rotation angles.

With the determinant of the inertia tensor and its derivative defined, what remains to calculate the derivative of the inertia tensor is to find the derivative of its adjoint and apply the quotient rule of differentiation. The derivative of the adjoint of the inertia tensor, given in Eq. (8.13), with respect to the control inputs is a $3 \times 3 \times 4$ matrix. These matrices are all 3×3 zero matrices, with the exception of the final matrix, which is

$$\frac{\partial \text{adj}(\mathcal{I})}{\partial \tilde{u}_3} = \begin{bmatrix} \frac{\partial I_{yy_b}}{\partial \delta_B}(I_{zz_b} - I_{yy_b}) - 2I_{yz_b}\frac{\partial I_{yz_b}}{\partial \delta_B} & I_{xz_b}\frac{\partial I_{yz_b}}{\partial \delta_B} & I_{xz_b}\frac{\partial I_{yy_b}}{\partial \delta_B} \\ I_{xz_b}\frac{\partial I_{yz_b}}{\partial \delta_B} & I_{xx_b}\frac{\partial I_{zz_b}}{\partial \delta_B} & \frac{\partial I_{yz_b}}{\partial \delta_B}(I_{xx_b} - I_{xz_b}) \\ \frac{\partial I_{yy_b}}{\partial \delta_B}I_{xz_b} & I_{xx_b}\frac{\partial I_{yz_b}}{\partial \delta_B} & I_{xx_b}\frac{\partial I_{yy_b}}{\partial \delta_B} \end{bmatrix} \tag{8.147}$$

since the inertia is only a function of the BIRE rotation angle. By applying the quotient rule of differentiation, a form that can be applied element-wise to the inverse of the inertia matrix to find its derivative. This form is

$$\frac{\partial \mathcal{I}^{-1}}{\partial \tilde{u}} = \frac{\det \mathcal{I}\frac{\partial \text{adj}(\mathcal{I})}{\partial \tilde{u}} - \text{adj}(\mathcal{I})\frac{\partial \det \mathcal{I}}{\partial \tilde{u}}}{(\det \mathcal{I})^2} \tag{8.148}$$

Since the derivative of the adjoint matrix is zero everywhere but along the final control input, and the derivative of the determinant is a 3×3 matrix with only the final column non-zero, Eq. (8.148) will result in a 3×3 matrix, as required by Eq. (8.106). Another example will be given here for the same case given when examining the linearized state matrix.

8.3.2 Example Case

In steady level flight at flight condition C2 and with its center of gravity in the nominal position, the baseline aircraft can be trimmed using the algorithm in Chapter 6 to find the states and control inputs given in Eqs. (8.94) and (8.95). Based on this trim condition, the baseline aircraft has a linearized control matrix given by

$$B = \begin{bmatrix} 0 & -1.0832 & 0 \\ 9.2863 & 0 & 24.0369 \\ 0 & -80.1667 & 0 \\ -21.2943 & 0 & 6.9457 \\ 0 & -10.7738 & 0 \\ -1.4418 & 0 & -3.7526 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{8.149}$$

Given the same state and control input for the BIRE trimmed at flight condition C2, given in Eqs. (8.98) and (8.99), its linearized control matrix is

$$B = \begin{bmatrix} -0.0050 & 1.5344 & 0 \\ -0.8673 & 0 & -1.7741 \\ 0.0934 & -108.3112 & 0 \\ -40.2017 & 0 & -0.0583 \\ -0.0115 & -16.0336 & 0 \\ 0.0295 & 0 & 0.2043 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{8.150}$$

While the examples examined to this point have been concerned with the linearized model of each aircraft with the center of gravity in its nominal position, the process described in this chapter is valid for any center of gravity position, as demonstrated in Eqs. (8.47)–(8.49) and (8.112)–(8.113). A code that is capable of generating the linearized state and control matrices of the baseline aircraft at any flight condition and with any center of gravity location is included in Section C.6 of Appendix C. Slight modifications, detailed in the paragraphs above, are necessary to generate a linear model for the BIRE aircraft. Here too, a code capable of generating this model for a variety of conditions is given in Section C.6 of Appendix C.

8.4 Analyzing Controllability of the BIRE System

With a linearized model available for each aircraft, several studies can be conducted to better understand any control limitations that the BIRE may face in comparison to the baseline aircraft. The first of these studies that will be given in this work is an analysis of the controllability of the BIRE aircraft as a function of the BIRE rotation angle. Afterwards, a study detailing the creation of a feedback controller using an LQR design for each aircraft will be presented with a performance analysis of each and the results of a simulation in the presence of wind gust disturbances.

Recall that the controllability matrix, Γ, given in Eq. (8.23), must be of a rank equivalent to the linearized state matrix if the linear system is controllable. Of particular concern with the BIRE is when the aircraft is linearized about a condition where the tail is completely horizontal or completely vertical ($\delta_B = 0°$ or $\delta_B = \pm 90°$, respectively). In this condition, the aircraft is devoid of either yaw or pitch control, and therefore may not be completely controllable. Steady level flight is one such condition, since the horizontal tail need not be rotated in this trim condition due to the lack of lateral forces on the aircraft.

After placing the aircraft in the steady level flight condition, the tail was rotated from $\delta_B = -90°$ to $\delta_B = 90°$ and a linearized state and control matrix was determined at each point. Then, the rank of the controllability matrix was calculated at each BIRE rotation angle. The results of this study are shown in Fig. 8.1. This analysis shows that the rank of the controllability matrix, Γ, as calculated by Eq. (8.23) indicates that the linearized BIRE system is completely controllable at this trim condition, regardless of the rotation of the horizontal tail. However, simply because the linearized BIRE system is completely controllable does not mean that the nonlinear BIRE system is completely controllable. The present controllability study must therefore be extended to determine controllability of a nonlinear system.

While Fig. 8.1 seems to indicate that the BIRE should be completely controllable regardless of the BIRE rotation angle, numerical error my also cause the rank of a matrix to appear to be full. Thus, the condition number of the controllability matrix must also

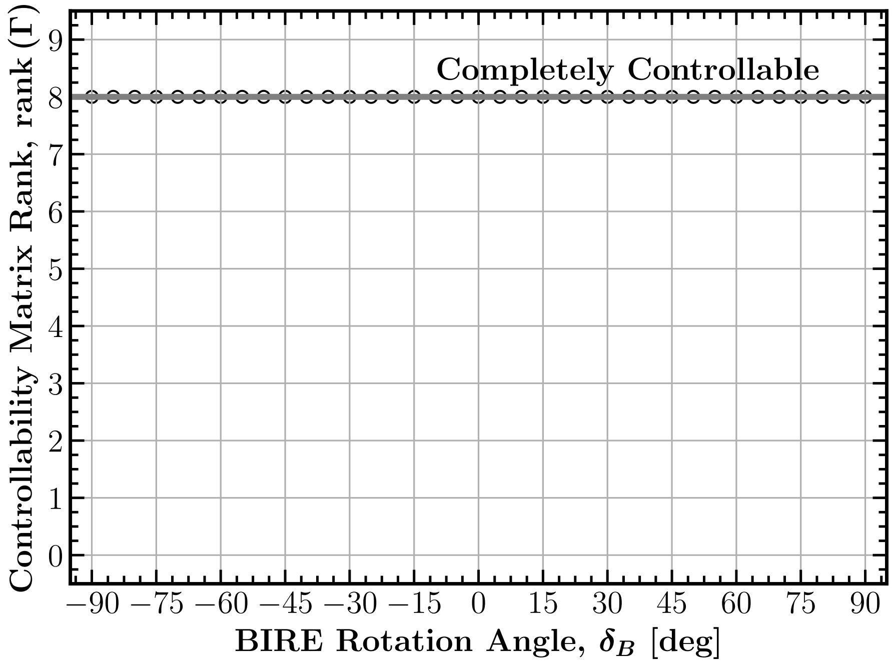


Fig. 8.1: Controllability analysis of the BIRE aircraft as a function of BIRE rotation angle.

be analyzed to determine if the results in Fig. 8.1 are accurate. Doing so reveals that the condition number of the controllability matrix at each BIRE rotation angle is indeed very large ($\approx 1 \times 10^6$). Therefore, the results shown in Fig. 8.1 are by no means conclusive, and further studies must be made to determine whether the BIRE can effectively stabilize itself using the given inputs at all BIRE rotation angles. Code for this controllability study is included in Section C.6 of Appendix C.

8.5 Disturbance Rejection Analysis

Another chief concern is the ability of the BIRE aircraft to navigate back to a trim condition when a disturbance is introduced into the system. To determine whether the linearized system of the baseline aircraft or BIRE are able to reject disturbances when equipped with a linear controller, that controller must first be developed. In this work, the gain matrix K will be produced for the baseline and BIRE aircraft using the linear quadratic regulator (LQR) method [133].

The LQR problem involves taking the linearized system in Eq. (8.4) and finding the input signal $u(t)$ that will take the system from a non-zero state $x(0)$ to the zero state in an optimal manner. This is done by minimizing the cost function

$$J = \int_0^\infty \left(x(t)^T Q x(t) + \tilde{u}(t)^T R \tilde{u}(t) \right) \, dt \tag{8.151}$$

The optimal solution to is of the form of Eq. (8.24), where

$$K = R^{-1} B^T P \tag{8.152}$$

and P is a unique, positive semi-definite solution to the algebraic Riccati equation

$$A^T P + PA - PBR^{-1} B^T P + Q = 0 \tag{8.153}$$

The constant matrices Q and R are design parameters that can be chosen by the design engineer to produce a controller that satisfies the appropriate performance metrics for the problem. They must be positive semi-definite, i.e. $Q = Q^T \geq 0$, and positive definite, $R = R^T > 0$, respectively. By virtue of their designation as design parameters, the development of any linear feedback controller is an iterative process. Thus, each of the weighting matrices included in this design have been adjusted multiple times until a solution with acceptable control inputs is able to damp out the system and return the aircraft to equilibrium.

Note that this analysis acts as a preliminary study into how the linearized systems in Sections 8.2 and 8.3 may be used to generate a feedback controller that is able to dampen out disturbances to the aircraft. Thus, the controllers here have not been entirely optimized for good performance according to traditional metrics for MIMO systems [69, 134]. Rather, a single instance has been shown wherein the linear feedback controller designed using LQR provides acceptable results in disturbance rejection. Further work will be required to improve the controllers demonstrated in this section, but this initial procedure will be helpful in giving a benchmark for future studies.

8.5.1 Gust Model

The disturbances to which each aircraft will be subjected are wind gusts in the form of damped sine waves. That is, they take the form

$$V_g = A_w e^{-\xi_w t} \sin(\omega_w t) \tag{8.154}$$

These gusts can be applied to each body-fixed direction by specifying an amplitude, A_w, a damping rate, ξ_w, and a gust frequency ω.

Note that, in Eq. (8.5), the time rate of change in the gust velocities are required to properly simulate their effect on the aircraft dynamics. Since the form given in Eq. (8.154) is analytic, its derivative with respect to time can easily be calculated to be

$$\dot{V}_g = A_w e^{-\xi_w t} \left[\omega_w \cos(\omega_w t) - \xi_w \sin(\omega_w t)\right] \tag{8.155}$$

Therefore, by prescribing a gust amplitude (in ft/s), a gust damping rate (in Hz), and a gust frequency (in rad/s), the effect of a gust on the aircraft dynamics can be modeled.

8.5.2 Baseline LQR Design

For the baseline aircraft, it was determined that the Q and R matrices would be selected according to physical intuition about the sensitivities of the aircraft. In particular, simulation showed that convergence of the system in the presence of a disturbance was very sensitive to the body-fixed rates and the elevation angle. Therefore, a weighting matrix Q

was chosen to be

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 \end{bmatrix} \tag{8.156}$$

Manipulation of the Q weighting matrix found that the LQR procedure was very sensitive to the gains on the velocity states, which has implications on the resultant feedback gain matrix. In terms of the controls, the weighting matrix R was chosen to be

$$R = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \tag{8.157}$$

to allow the elevator more authority to damp out the elevation angle and pitching rate.

Using the weighting matrices in Eqs. (8.156) and (8.157), the feedback gain matrix K was determined using Eqs. (8.151)–(8.153) in the Python controls library[1] to be

$$K = \begin{bmatrix} 0 & 0.0010 & 0 & -2.0342 & 0 & -0.7019 & -0.0827 & 0 \\ 0 & 0 & -0.0005 & 0 & -3.2265 & 0 & 0 & -4.4720 \\ 0 & -0.0004 & 0 & 0.6768 & 0 & -2.0441 & 0.1205 & 0 \end{bmatrix} \tag{8.158}$$

Note that the gains in the first row of Eq. (8.158) are not exactly zero, but are below the tolerance shown in the rest of the matrix. In fact, each of the velocity states have very small gains compared to the rest of the gains in the matrix. Thus, the velocity states may be ignored in further refinements of the linear controller.

[1] https://python-control.readthedocs.io/en/0.9.2/

8.5.3 BIRE LQR Design

For the BIRE aircraft, several changes were made from the matrices Q and R developed for the baseline aircraft. First, it was noted that the bank angle converged very slowly in the BIRE in most of the simulations conducted. Compensating for this required a higher weighting on the roll rate and bank angle. In general, the BIRE rotation angle was also shown across various gusts to produce very small angles ($\delta_B < 10°$), and was instrumental in providing control to the aircraft. Thus, the penalty in the LQR optimization was relaxed when compared to the ailerons. The elevator control was also assigned a smaller penalty to allow for its use to correct the instabilities about the pitch axis.

With these considerations in mind, the BIRE aircraft was given a weighting matrix Q equal to

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 20 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix} \tag{8.159}$$

and the matrix R was chosen as

$$R = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 0.1000 & 0 \\ 0 & 0 & 0.1000 \end{bmatrix} \tag{8.160}$$

Thus, the feedback gain matrix K was found to be

$$K = \begin{bmatrix} 0 & 0.0004 & 0 & -3.1031 & -0.0002 & -0.0330 & -2.2337 & -0.0002 \\ 0 & 0 & 0.0003 & 0.0012 & -10.0133 & 0 & 0.0009 & -10 \\ 0 & -0.0034 & 0 & -0.0781 & 0 & 11.0685 & 0.5070 & 0 \end{bmatrix} \tag{8.161}$$

Again, it can be noted that the translational velocities play a much smaller role in the control of the BIRE than any of the other states.

8.5.4 Simulation

Simulation of both the baseline and BIRE aircraft was performed using the full rigid-body equations of motion in Eqs. (4.1)–(4.4). Often, the Euler angle formulation in Eq. (4.4) is replaced with a quaternion formulation to avoid the effects of gimbal lock [135]. Since the simulated wind gusts were not expected to create changes in the elevation angle θ to the degree that this would be an issue, the slightly simpler and more intuitive Euler angles were used. If more extreme maneuvers are simulated in future work, a quaternion formulation should be implemented.

Each simulation started with the aircraft in its trim condition in steady, level flight in flight condition C2 (the flight maneuver condition). This condition was chosen since it was assumed that the need to saturate any of the control surfaces in this configuration would weigh heavily on the ability of the aircraft to perform its mission. Future work can easily change the flight condition to any of the conditions in Table 6.1 and further test the control response of each aircraft.

From the trim condition, the equations of motion were integrated forward in time using an explicit fourth-order Runge-Kutta method with stepsize control [136]. At each step, the changes in the controlled states were calculated and the linear feedback matrix in Eqs. (8.158) or (8.161) was used in Eq. (8.24) to determine the required control inputs to stabilize each aircraft. Evaluating these control inputs over time also allows for the rate of the control inputs to be calculated, which will be helpful in understanding the feasibility of using practical actuation devices to control the BIRE.

Wind gusts were defined in all three body-fixed directions with the gust in the body-fixed z-direction having half the amplitude of the others. The amplitude was defined as $A_w = 80$ ft/s, the damping rate as $\xi = 1$ Hz, and the gust frequency was given as $\omega_w = 5$ rad/s. The wind gust is shown in Fig. 8.2 and was simulated starting at $t = 1$ second. Figures 8.3 and 8.4 show the states of the aircraft as given in Eqs. (8.5)–(8.7) when simulated over 20 seconds with a time increment of $\Delta t = 0.1$ second. Note that the aircraft in both a controlled and uncontrolled state are plotted.

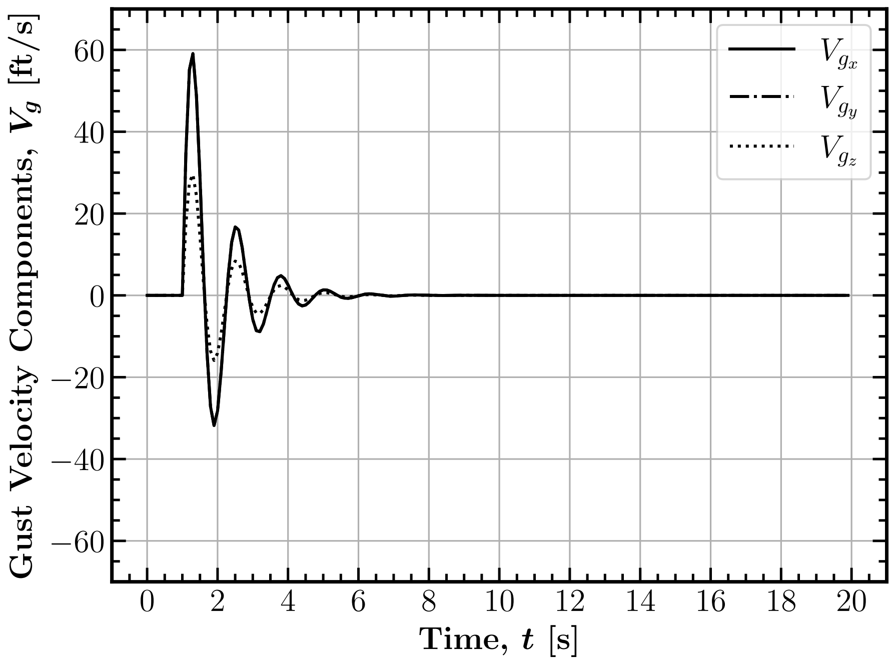


Fig. 8.2: Simulated wind gust.

It has been previously established that the uncontrolled linearized systems of both of the aircraft are unstable. However, note that there are two unstable eigenvalues for the baseline aircraft and only one for the BIRE. While any broad assumptions about the stability of either aircraft cannot be made, the amplitudes of oscillation shown in Fig. 8.4 certainly indicate a marginal improvement in stability of the uncontrolled aircraft.

The results in Figs. 8.3 and 8.4 shown that the control law given by Eq. (8.24) along with the gain matrices in Eqs. (8.158) and (8.161) have produced systems that are asymptotically stable. These details can be seen better by considering the shifted states, z. Figures 8.5 and 8.6 show the shifted states across the simulation window. These are the states upon which the feedback control system is operating; therefore, any states in Fig. 8.3 and 8.4 that are not included in these plots should not be expected to converge.

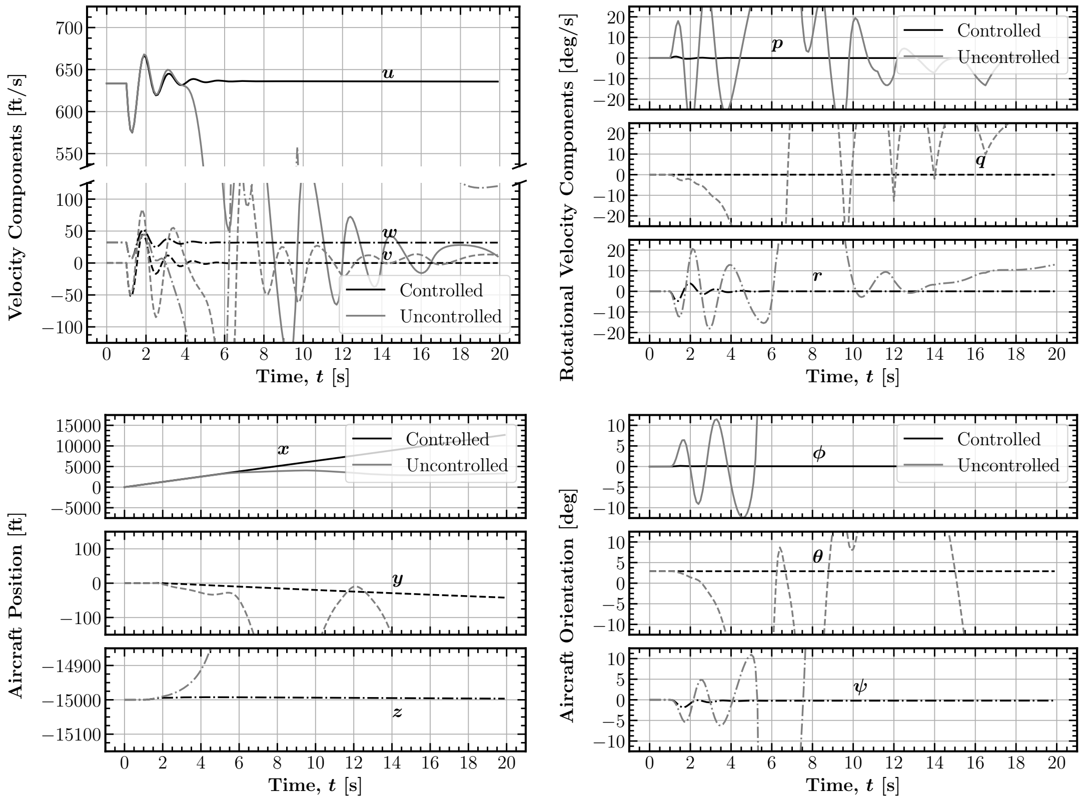


Fig. 8.3: Simulated states of the baseline aircraft in the presence of a wind gust.

Beyond only analyzing the states of each aircraft, a great deal of information can be gathered by examining the magnitude and rates of the control deflections required to reject the gust disturbance. A time history of these control inputs are shown in Fig. 8.7. These plots show that the overall magnitude of the control inputs are well below the saturation deflection of each control surface shown in Tables 3.3 and 3.6. Additionally, while the magnitude of the control inputs for the baseline could be considered “small”, the BIRE deflection angle reaches a magnitude of $\delta_B \approx 15°$. The fact that both aircraft are able to stabilize in this condition indicates that the linearization shown here is adequate for control. However, larger deflections may operate outside of the linear region where the aerodynamics and linearized system are applicable.

Figure 8.8 shows the control rates of each aircraft throughout the simulation. Required deflection rates above the actuation limits shown in Tables 3.3 and 3.6 would require more

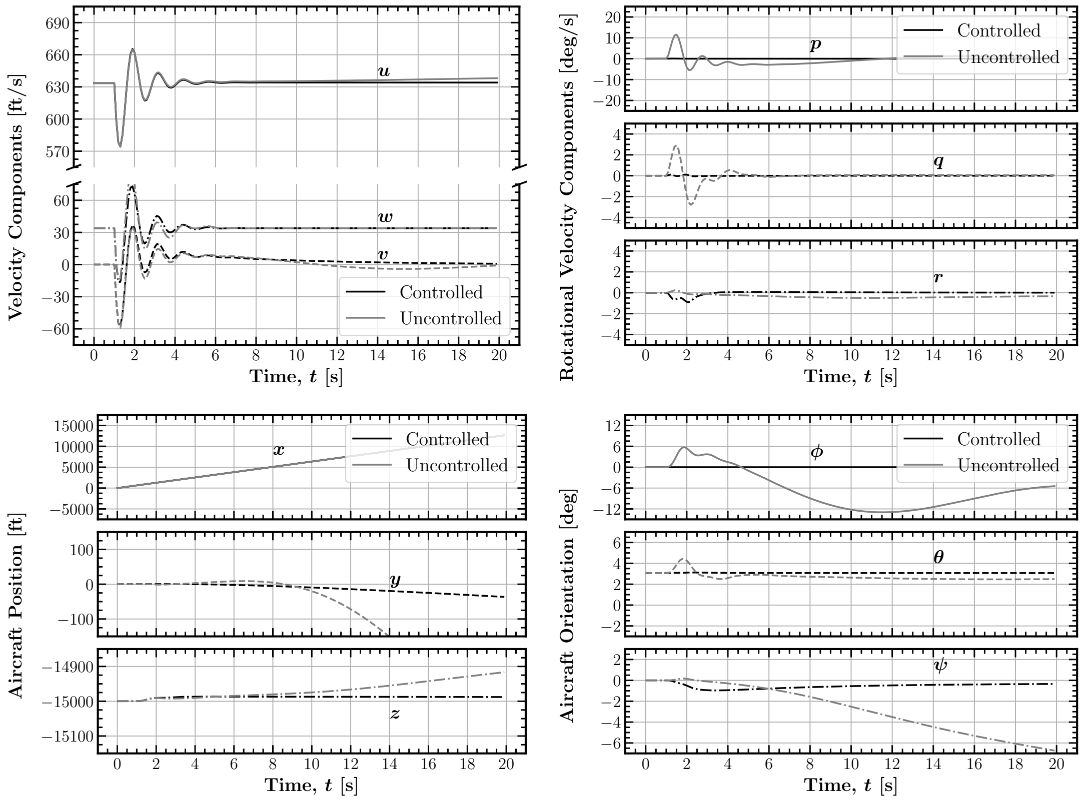


Fig. 8.4: Simulated states of the BIRE aircraft in the presence of a wind gust.

powerful actuation systems and would not be ideal. As it stands, further research into the mechanism design of the BIRE actuation system itself will further refine the actuation rates that are reasonable to suppose in the design. Nonetheless, the actuation rates predicted here for this example simulation are within reasonable limits and do not nearly reach the actuation limits of the baseline aircraft as reported by Stevens and Lewis [69].

These results have demonstrated that a linearized model of the BIRE aircraft can be useful in designing a state feedback controller. In fact, in the presence of this particular gust disturbance, the BIRE aircraft is stabilizable with the linear control design. However, multiple-input, multiple-output (MIMO) systems like the BIRE can also be sensitive to the direction of the disturbance [128]. Thus, to test the robustness of the controller in the presence of multiple gust directions, a sweep of gusts can be generated to test whether the BIRE can be stabilized.

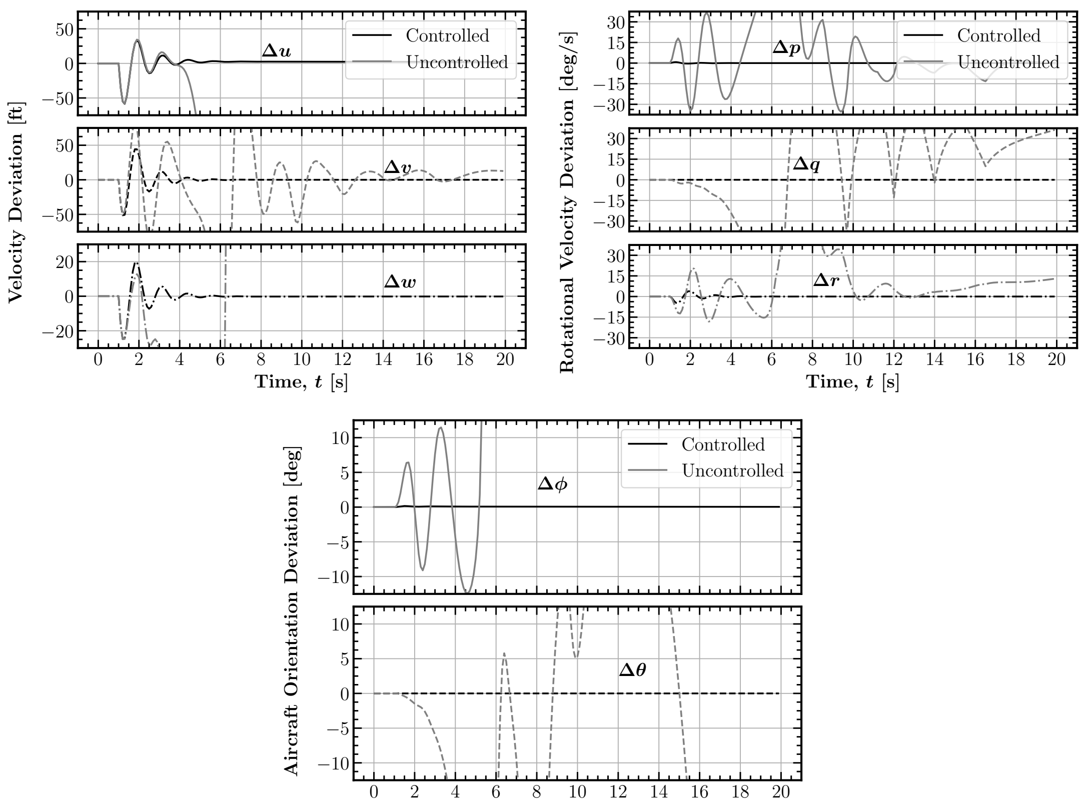


Fig. 8.5: Shifted states of the baseline aircraft in the presence of a wind gust.

Gust Directionality Test

The direction of the gust disturbance can be adjusted by rewriting the gust velocity in Eq. (8.154) as

$$V_g = s_g A_w e^{-\xi_w t} \sin(\omega_w t) \tag{8.162}$$

where

$$s_g = \begin{bmatrix} s_x \\ s_y \\ s_z \end{bmatrix} \tag{8.163}$$

with $|s_x| \leq 1$, $|s_y| \leq 1$, and $|s_z| \leq 1$. Thus, the vector s_g simply orients the amplitude of the gust velocity in the body-fixed coordinate system. By simulating the BIRE aircraft in the presence of a range of gust directions, an approximation of the robustness of the aircraft can be made.

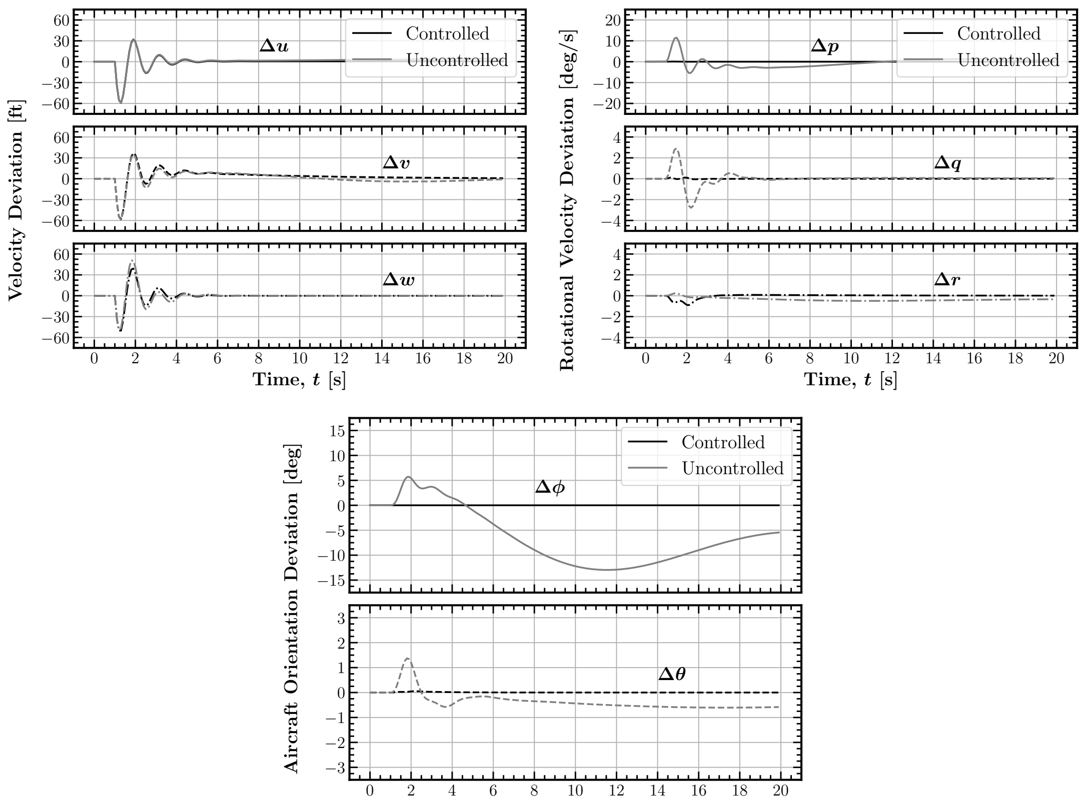
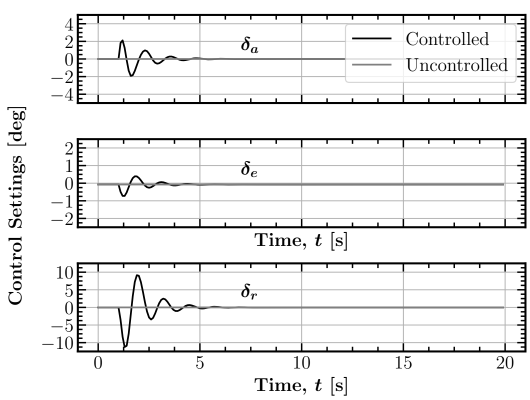
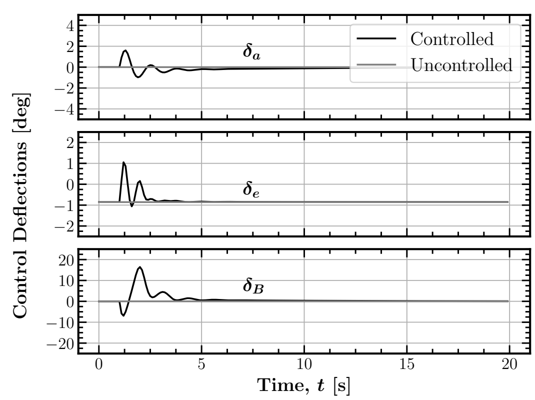


Fig. 8.6: Shifted states of the BIRE aircraft in the presence of a wind gust.



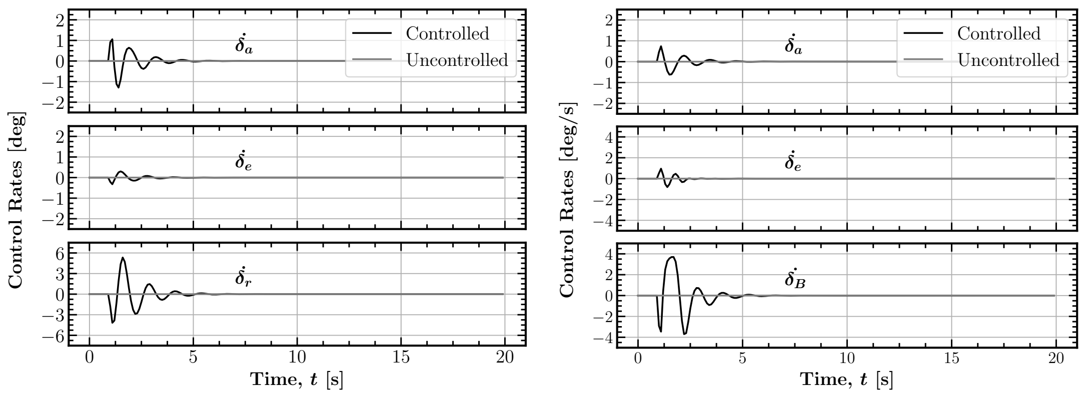
(a) Baseline Control



(b) BIRE Control

Fig. 8.7: Control inputs for each aircraft to reject a gust disturbance.

With the results of all of these simulations, a satisfactory method for determining whether the aircraft was returning to its trim condition needed to be made. It was determined that if the mean of the shifted states corresponding to the last five seconds of



(a) Baseline Rates

(b) BIRE Rates

Fig. 8.8: Control input rates for each aircraft to reject a gust disturbance.

simulation was less than 5% of the maximum value attained by that shifted state across the entire study, then the case could reasonably be assumed to be converging. This allowed the studies in which the gust was not applied in a given direction to avoid being flagged as a divergent case. In total, the directionality study was conducted with 11 points in each direction for a total of approximately 1300 cases.

By this metric, every case within the directionality study was found to be returning to its trim condition in the presence of the gust. Several of these cases were spot-checked through simulation and the method was found to be consistent. Thus, this initial study shows that the linearized state feedback controller appears to be an effective method with which to control the BIRE aircraft. Additional studies must be performed to confirm this is the case in a wider range of trim conditions and across a larger envelope of flight conditions. This study also does not consider the rate at which the aircraft is returning to its trim condition as a restriction, which certainly is important when designing a controller. However, the preliminary nature of this control study has shown that this control methodology is worth pursuing further when designing a BIRE-type aircraft.

CHAPTER 9

SUMMARY AND CONCLUSIONS

This dissertation has explored the aerodynamic implications of a novel control system called the bio-inspired rotating empennage or BIRE. As a control system, the BIRE is inspired by the maneuverability and control presented by birds during flight. Aerodynamic results and studies for a baseline aircraft and its BIRE variant were presented to provide an indication of its viability as a control system and also to better understand the aerodynamic trade-offs that it provides. While this control concept has seen little analysis in the past, this work has shown that it is worthy of further investigation and that its predicted benefits may allow future aircraft designs to leverage a portion of the weight and drag benefits of a tailless aircraft while maintaining a fairly simple control system. The analyses in this dissertation indicate the potential benefits of the BIRE control system and motivate future research into its viability as a control system.

Chapter 2 provides an aerodynamically-supported analysis of literature examining the use of the tail in bird flight. Observational, analytical, and experimental work was referenced in combination with traditional flight mechanics relationships to provide additional insight into the mechanics of the tail during a bird's flight. These relationships were meant to provide intuition into the effects of a rotating tail and to also provide context for the relationships that would be explored further into the dissertation. In Chapter 2, the available literature covering aircraft with rotating tail designs was also explored. These studies were scarce, but provided details about potential concerns and benefits that could be expected through implementing a rotating tail as a control system in an aircraft.

To develop an aerodynamic model, the geometry of a baseline aircraft and its BIRE variant were required to be characterized. In this dissertation, the baseline aircraft was modeled after a fighter aircraft with relaxed static stability. This aircraft was chosen because of the publicly-available data that could be used to characterize its geometry as well as to provide insight into potential maneuverability benefits provided by the BIRE concept. In Chapter 3, the baseline geometry was outlined using the open sources available and modified its geometric properties to develop the planform of a BIRE variant. Certain properties of the aircraft needed to be scaled off of drawings provided from the literature, which introduced a level of uncertainty into the design that needed to be addressed in the aerodynamic data produced later in the work.

With the geometry defined for each aircraft, a linear aerodynamic model was developed for each aircraft in Chapter 4. These models were created by linearizing the aerodynamics of each aircraft about a condition in which the aerodynamic angles, body-fixed rotation rates, and control surface deflections were all zero. While the linear models were simple and allowed for a basic understanding of the aerodynamics of each aircraft, it was determined that certain non-linear effects needed to be included to bring more fidelity to the model. This was especially important given the non-linear aerodynamic effects that the baseline aircraft would regularly encounter in its flight envelope due to its nature as a fighter aircraft. The higher-fidelity, nonlinear models were constructed using relationships gleaned from lifting-line theory as well as familiarity with a wind-tunnel data set published for the baseline aircraft. Finally, the non-linear aerodynamic model for the BIRE was characterized by assuming that the coefficients in its model varied with the rotation of the tail. This variation was generalized to be a shifted sine wave with an offset.

The evaluation of the coefficients in the previously-defined aerodynamic models was given in Chapter 5. Numerical lifting-line theory was used to generate aerodynamic data for the baseline aircraft and the BIRE using airfoil data estimated using thin airfoil theory and wind tunnel data. The aerodynamic coefficients of each model were then determined using finite difference methods and least-squares polynomial fits. Furthermore, a sensitivity analysis was performed to determine whether the inclusion of higher-order effects from wind tunnel data provided a significant change to the static trim analysis that would follow. From this sensitivity study, changes in the aerodynamic coefficients were modeled using deltas informed from the wind tunnel data of the baseline aircraft as necessary to include higher-order effects such as leading-edge vortices and viscous or spanwise flow effects. With the aerodynamic model determined for the baseline and BIRE aircraft, the implications of the trends were discussed in detail for the BIRE as a function of tail rotation angle. It was shown that consistent, physically-intuitive patterns could be seen across the aerodynamic coefficients in the non-linear model. In addition, several higher-order trends were noted for inspection with a higher-fidelity aerodynamic tool.

Chapters 6–8 demonstrated several studies that could be performed with the aerodynamic model for the baseline aircraft and BIRE variant. The first of these studies focused on the trim envelope available to the baseline aircraft compared to that attainable by the BIRE. To determine this, a trim algorithm was developed using two numerical methods: the fixed-point iteration and a Newton-Raphson method. A static trim analysis was then performed using a steady, coordinated turn as the trim condition across several flight conditions identified from other tailless aircraft studies. This analysis showed that the BIRE has similar trim capabilities in a steady, coordinated turn to the baseline aircraft. However, it was noted that certain discontinuities existed when the direction of the force acting on the tail had to switch rapidly, due to the relaxed static stability of the baseline aircraft. Thus, a center of gravity study was performed that showed improved convergence and smoothness of the trim data when the center of gravity was moved forward on the BIRE.

The second trim condition studied was the steady-heading sideslip condition, which is often used when landing. This analysis showed similar discontinuities in the trim conditions of the BIRE until the center of gravity was moved forward. In this case, the BIRE was shown to have a substantially larger trim envelope than the baseline aircraft. The steady-heading sideslip trim condition also provided an opportunity to test whether the BIRE was more susceptible to a tail strike than the baseline aircraft. An analysis showed that the risk of tail strike for the BIRE aircraft was less when in steady level flight at the landing conditions considered. However, in certain scenarios, the BIRE presented a higher risk of tail strike than the baseline aircraft when assuming a bank angle. Since fighter aircraft often land in a crosswind by crabbing into the wind, this condition must be checked using simulation of the aircraft beyond that described in this work to determine if a greater risk is presented for tail strike.

Chapter 7 contains an analysis of the attainable moments of each aircraft in which some of the trade-offs between longitudinal and lateral control in the BIRE were identified. By using the aerodynamic model to determine the maximum lateral moments that could be generated while maintaining a given pitching moment, an attainable moment set envelope was generated that could then identify regions where the BIRE lacked control authority. The BIRE lost a substantial amount of yaw control authority when the pitching moment requirements of trim needed to be maintained. However, the yaw authority available to the BIRE very quickly increased when even small reductions in required pitch control were allowed. An additional study in Chapter 7 included an analysis of the drag increment sustained by the baseline and BIRE aircraft when generating a given yawing moment. These results were compared to a theoretical minimum drag increment sustained by using wing twist to generate a yawing moment. Substantial benefit was noted by the BIRE, which required less drag than both the baseline aircraft and an aircraft using wing twist.

Finally, Chapter 8 focused on deriving a linearized system for the baseline and BIRE aircraft that could be used to develop a linear state-feedback control system. These linearized systems were developed using the aircraft equations of motion and the aerodynamic models developed in previous chapters. It was shown that the BIRE was completely controllable at all flight conditions examined in this work, regardless of the tail rotation angle it assumed. However, this controllability analysis was shown to be subject to numerical error, and other efforts of defining controllability must be made.

A state feedback gain matrix was then developed using a linear quadratic regulator approach and the baseline and BIRE aircraft were both shown to be able to reject a gust disturbance using this controller. In addition, the robustness of the controller was examined by subjecting the BIRE aircraft to a range of gust directions. This analysis showed that the BIRE was able to return to its trim condition using the linearized controller in all of the cases studied here.

Much of the future work available to the BIRE will likely be related to this work in controls, and the analysis presented in this dissertation gives both the linearized system as well as preliminary results with which to move forward in the analysis. Future work specifically related to what was presented in this dissertation could be looking into further analysis into nonlinear controllability implications of the BIRE as well as an analysis relating to the region of attraction of the linearized system. The latter is especially interesting, as the BIRE system is not currently restricted and could vary substantially during flight. Therefore, a linear system may be insufficient in certain flight scenarios and may need to be supplemented with certain nonlinear techniques.

The intent of this dissertation was to provide an aerodynamic analysis of a bio-inspired rotating empennage design. This has been accomplished by laying forth a methodology for modeling that can be easily replicated using higher-fidelity tools. As a benchmark, the analysis in this dissertation will provide valuable data for future researchers that continue to develop this control system and analyze its benefits.

REFERENCES

[1] Bowlus, J., Multhopp, D., and Banda, S., "Challenges and opportunities in tailless aircraft stability and control," Guidance, Navigation, and Control Conference, American Institute of Aeronautics and Astronautics, Aug. 1997, pp. 1713–1718.

[2] "Department of Defense Research & Engineering Enterprise : Research & Technology," https://rt.cto.mil, Accessed: 2020-12-22.

[3] Harvey, C., Gamble, L. L., Bolander, C. R., Hunsaker, D. F., Joo, J. J., and Inman, D. J., "A review of avian-inspired morphing for UAV flight control," Progress in Aerospace Sciences, 2022, In Review.

[4] McVeigh, K., "Industrial fishing ushers the albatross closer to extinction say researchers," https://www.theguardian.com/environment/2019/jan/31/industrial-hing-ushers-albatross-closer-to-extinction-say-researchers, Accessed: 2020-12-22.

[5] McCann, M., "How the Swallow-tailed Kite Soars With Such Grace," https://www.audubon.org/news/how-swallow-tailed-kite-soars-such-grace, Accessed: 2020-12-22.

[6] Roetman, E. L., Northcraft, S. A., and Dawdy, J. R., "Innovative Control Effectors (ICE)," Tech. Rep. 96-3074, Boeing Defense and Space Group, Seattle, WA, March 1996.

[7] Dorsett, K. M. and Mehl, D. R., "Innovative Control Effectors (ICE)," Tech. Rep. 96-3043, Lockheed Martin Tactical Aircraft Systems, Fort Worth, TX, Jan. 1996.

[8] Phillips, W. F., Mechanics of Flight, John Wiley & Sons, Inc., 2nd ed., 2010.

[9] Phillips, W. F., "Simplified Pitch Stability Analysis for a Wing-Tail Combination," Mechanics of Flight, chap. 4, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 384–400.

[10] Phillips, W. F., "Stick-Fixed Neutral Point and Static Margin," Mechanics of Flight, chap. 4, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 400–411.

[11] Phillips, W. F., "Longitudinal Motion: The Linearized Coupled Equations," Mechanics of Flight, chap. 8, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 836–846.

[12] Phillips, W. F., "Short-Period Approximation," Mechanics of Flight, chap. 8, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 847–854.

[13] Phillips, W. F., "Long-Period Approximation," Mechanics of Flight, chap. 8, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 854–871.

[14] Phillips, W. F., "Longitudinal Control and Maneuverability," Mechanics of Flight, chap. 6, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 605–623.

[15] Phillips, W. F., "Trailing-Edge Flaps and Section Flap Effectiveness," Mechanics of Flight, chap. 1, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 39–46.

[16] Anderson Jr, J. D., Introduction to Flight, McGraw-Hill Higher Education, 1989.

[17] Phillips, W. F., "Roll Stability and Dihedral Effect," Mechanics of Flight, chap. 5, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 548–566.

[18] Phillips, W. F., "Lateral Motion: The Linearized Coupled Equations," Mechanics of Flight, chap. 9, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 885–895.

[19] Phillips, W. F., "Roll Approximation," Mechanics of Flight, chap. 9, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 896–897.

[20] Phillips, W. F., "Force and Moment Derivatives," Mechanics of Flight, chap. 7, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 768–788.

[21] Phillips, W. F., "Yaw Stability and Trim," Mechanics of Flight, chap. 5, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 500–517.

[22] Phillips, W. F., "Spiral Approximation," Mechanics of Flight, chap. 9, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 897–905.

[23] Phillips, W. F., "Dutch Roll Approximation," Mechanics of Flight, chap. 9, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 906–919.

[24] Phillips, W. F., "Incompressible Flow over Finite Wings," Mechanics of Flight, chap. 1, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 46–94.

[25] Gillies, J. A., Thomas, A. L. R., and Taylor, G. K., "Soaring and manoeuvring flight of a steppe eagle Aquila nipalensis," Journal of Avian Biology, Vol. 42, No. 5, Sept. 2011, pp. 377–386.

[26] Thomas, A. L. R., "On the aerodynamics of birds' tails," Philosophical Transactions of the Royal Society B, Vol. 340, jul 1993, pp. 361–380.

[27] Storer, J. H., The flight of birds analyzed through slow-motion photography, No. 28, Cranbrook Institute of Science, 1948.

[28] Tucker, V. A., "Pitching Equilibrium, Wing Span and Tail Span in a Gliding Harris' Hawk, Parabuteo Unicinctus," Journal of Experimental Biology, Vol. 165, No. 1, April 1992, pp. 21–41.

[29] Carruthers, A. C., Thomas, A. L. R., and Taylor, G. K., "Automatic aeroelastic devices in the wings of a steppe eagle Aquila nipalensis," Journal of Experimental Biology, Vol. 210, No. 23, Dec. 2007, pp. 4136–4149.

[30] Carruthers, A. C., Thomas, A. L., Walker, S. M., and Taylor, G. K., "Mechanics and aerodynamics of perching manoeuvres in a large bird of prey," The Aeronautical Journal, Vol. 114, No. 1161, 2010, pp. 673–680.

[31] Brown, R., "The flight of birds," Biological Reviews, Vol. 38, No. 4, 1963, pp. 460–489.

[32] Oehme, H., “Die Flugsteuerung des Vogels. I. ́Uber flugmechanische Grundlagen,” Beitr Vogelkd, Leipzig, Vol. 22, 1976, pp. 58–66.

[33] Oehme, H., “Die Flugsteuerung des Vogels. III. Flugman ́over der Kornweihe (Circus cyaneous),” Beitr Vogelkd, Leipzig, Vol. 22, 1976, pp. 73–82.

[34] Hankin, E. H., Animal flight: a record of observation, Iliffe & Sons Limited, 1913.

[35] Thomas, A. L. and Taylor, G. K., “Animal Flight Dynamics I. Stability in Gliding Flight,” Journal of Theoretical Biology, Vol. 212, No. 3, Oct. 2001, pp. 399–424.

[36] Dunne, P., Sibley, D., and Sutton, C., Hawks in flight: the flight identification of North American migrant raptors, Houghton Mifflin Harcourt (HMH), 1988.

[37] Pennycuick, C. J., “A wind-tunnel study of gliding flight in the pigeon Columba livia,” Journal of Experimental Biology, Vol. 49, No. 3, 1968, pp. 509–526.

[38] Song, J., Cheney, J., Usherwood, J., and Bomphrey, R., “Virtual manipulation of bird tail postures demonstrates drag minimisation when gliding,” BioRxiv, 2020.

[39] Maybury, W. J., Rayner, J. M., and Couldrick, L. B., “Lift generation by the avian tail.” Proceedings of the Royal Society of London. Series B: Biological Sciences, Vol. 268, No. 1475, July 2001, pp. 1443–1448.

[40] Gatesy, S. M. and Dial, K. P., “Tail muscle activity patterns in walking and flying pigeons (Columba livia),” Journal of Experimental Biology, Vol. 176, No. 1, 1993, pp. 56–76.

[41] Rosén, M. and Hedenström, A., “Gliding flight in a jackdaw: a wind tunnel study,” Journal of Experimental Biology, Vol. 204, 2001, pp. 1153–1166.

[42] Henningsson, P. and Hedenström, A., “Aerodynamics of gliding flight in common swifts,” Journal of Experimental Biology, Vol. 214, No. 3, Feb. 2011, pp. 382–393.

[43] Thomas, A. L., “On the tails of birds,” Bioscience, 1997, pp. 215–225.

[44] Cheney, J. A., Stevenson, J. P. J., Durston, N. E., Maeda, M., Song, J., Megson-Smith, D. A., Windsor, S. P., Usherwood, J. R., and Bomphrey, R. J., “Raptor wing morphing with flight speed,” Journal of The Royal Society Interface, Vol. 18, No. 180, 2021, pp. 1–14.

[45] Evans, M. R., Rosén, M., Park, K. J., and Hedenström, A., “How do birds’ tails work? Delta–wing theory fails to predict tail shape during flight,” Proceedings of the Royal Society of London. Series B: Biological Sciences, Vol. 269, No. 1495, 2002, pp. 1053–1057.

[46] Pennycuick, C. and Webbe, D., “Observations on the fulmar in Spitsbergen,” British Birds, Vol. 52, 1959, pp. 321–332.

[47] Raspet, A., “Biophysics of Bird Flight,” Science, Vol. 132, No. 3421, July 1960, pp. 191–200.

[48] Pennycuick, C., "Chapter 1 - MECHANICS OF FLIGHT," Avian Biology, edited by D. S. Farner and J. R. King, Academic Press, Amsterdam, 1975, pp. 1–75.

[49] Mouillard, L.-P., L'empire de l'air: essai d'ornithologie appliquée à l'aviation, G. Masson, 1881.

[50] Usherwood, J. R., Cheney, J. A., Song, J., Windsor, S. P., Stevenson, J. P. J., Dierksheide, U., Nila, A., and Bomphrey, R. J., "High aerodynamic lift from the tail reduces drag in gliding raptors," The Journal of Experimental Biology, Vol. 223, No. 3, Feb. 2020, pp. jeb214809.

[51] Hoey, R. G., "Exploring bird aerodynamics using radio-controlled models," Bioinspiration & Biomimetics, Vol. 5, No. 4, Dec. 2010, pp. 045008.

[52] Sachs, G., "Aerodynamic yawing moment characteristics of bird wings," Journal of Theoretical Biology, Vol. 234, No. 4, June 2005.

[53] Sachs, G., "Yaw stability in gliding birds," Journal of Ornithology, Vol. 146, No. 3, July 2005.

[54] Sachs, G., "Tail effects on yaw stability in birds," Journal of Theoretical Biology, Vol. 249, No. 3, Dec. 2007, pp. 464–472.

[55] Oehme, H., "Der Flug des Fahnendrongos(Dicrurus macrocercus)," Journal für Ornithologie, Vol. 106, No. 2, April 1965, pp. 190–203.

[56] Warrick, D. R., Dial, K. P., and Biewener, A. A., "Asymmetrical Force Production in the Maneuvering Flight of Pigeons," The Auk, Vol. 115, No. 4, Oct. 1998, pp. 916–928.

[57] Hummel, D., "Aerodynamic investigations on tail effects in birds," Zeitschrift für Flugwissenschaften und Weltraumforschung, Vol. 16, No. 3, 1992.

[58] Ajanic, E., Feroskhan, M., Mintchev, S., Noca, F., and Floreano, D., "Bioinspired wing and tail morphing extends drone flight capabilities," Sci. Robot., Vol. 5, 2020, pp. eabc2897.

[59] Zheng, L., Zhou, Z., Sun, P., Zhang, Z., and Wang, R., "A novel control mode of bionic morphing tail based on deep reinforcement learning," arXiv preprint arXiv:2010.03814, 2020.

[60] Bras, M., Vale, J., Lau, F., and Suleman, A., "Flight Dynamics and Control of a Vertical Tailless Aircraft," Journal of Aeronautics & Aerospace Engineering, Vol. 2, No. 4, 2013, pp. 1–10.

[61] Parga, J. R., Reeder, M. F., Leveron, T., and Blackburn, K., "Experimental study of a micro air vehicle with a rotatable tail," Journal of aircraft, Vol. 44, No. 6, 2007, pp. 1761–1768.

[62] Fox, M. C. and Forrest, D. K., "Supersonic Aerodynamic Characteristics of an Advanced F-16 Derivative Aircraft Configuration," NASA TP-3355, 1993.

[63] Butcher, D., "Non-honeycomb F-16 horizontal stabilizer, structural design," ICAS Proc. 1982, AIAA Aircraft Syst. And Technol. Conf. Seattle, Vol. 2, 1982, pp. 586–592.

[64] Nguyen, L., Ogburn, M., Gilbert, W., Kibler, K., Brown, P., and Deal, P., "Simulator Study of Stall/Post-Stall Characteristics of a Fighter Airplane With Relaxed Longitudinal Static Stability," NASA TR-1538, 1979.

[65] Abbott, I. H. and Von Doenhoff, A. E., Theory of wing sections: including a summary of airfoil data, Courier Corporation, 2012.

[66] Anderson, J. D., Aircraft performance and design, McGraw-Hill, 1st ed., 1999.

[67] Phillips, W. F., "Effects of the Fuselage, Nacelles, and External Stores," Mechanics of Flight, chap. 4, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 472–476.

[68] Axelson, J. A., "Longitudinal Stability and Control of High-speed Airplanes with Particular Reference to Dive Recovery," Research Memorandum A7C24, NACA, Moffett Field, Calif., Sep 1947.

[69] Stevens, B. L. and Lewis, F. L., "Aircraft Control and Simulation," Aircraft Engineering and Aerospace Technology, 2004.

[70] Hibbeler, R. C., "Mass Moment of Inertia," Engineering Mechanics: Dynamics, chap. 17, Pearson Prentice Hall, 14th ed., 2016, pp. 409–417.

[71] Hibbeler, R. C., "Moments and Products of Inertia," Engineering Mechanics: Dynamics, chap. 21, Pearson Prentice Hall, 14th ed., 2016, pp. 591–595.

[72] Phillips, W. F., "Effects of Tail Dihedral on Yaw Stability," Mechanics of Flight, chap. 5, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 529–547.

[73] Bolander, C. R., Hunsaker, D. F., Myszka, D., and Joo, J. J., "Attainable Moment Set and Actuation Time of a Bio-Inspired Rotating Empennage," AIAA SCITECH 2022 Forum, 2022, p. 1670.

[74] Ives, C., Myszka, D. H., Joo, J., Bolander, C. R., and Hunsaker, D. F., "Using a Topology Optimization Results Interpreter on the Frame of an Aircraft with a Bio-Inspired Rotating Empennage," AIAA AVIATION 2022 Forum, 2022, p. 3373.

[75] Bolander, C. R., Kohler, A., Hunsaker, D. F., Myszka, D., and Joo, J. J., "Static Trim of a Bio-Inspired Rotating Empennage for a Fighter Aircraft," 2023 AIAA SciTech Forum, American Institute of Aeronautics and Astronautics, Jan 2023.

[76] Phillips, W. F., "Rigid-Body 6-DOF Equations of Motion," Mechanics of Flight, chap. 7, John Wiley & Sons, Inc., 2nd ed., 2010, p. 753.

[77] Phillips, W. F., "Newton's Second Law for Rigid-Body Dynamics," Mechanics of Flight, chap. 7, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 725–735.

[78] Phillips, W. F., "Position and Orientation: The Euler Angle Formulation," Mechanics of Flight, chap. 7, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 735–752.

[79] Anderson, J. D., "Aerodynamic Forces and Moments," Fundamentals of Aerodynamics, chap. 1, McGraw-Hill Education, 6th ed., 2017, pp. 19–32.

[80] Phillips, W. F., "Introduction and Notation," Mechanics of Flight, chap. 1, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 1–9.

[81] Stokes, G. G., "On the Effect of the Internal Friction of Fluids on the Motion of Pendulums," Transactions of the Cambridge Philosophical Society, Vol. 9, Jan. 1851, pp. 8.

[82] Anderson, J. D., "Viscous Flow: Airfoil Drag," Fundamentals of Aerodynamics, chap. 4, McGraw-Hill Education, 6th ed., 2017, pp. 379–395.

[83] Anderson, J. D., "Definition of Compressible Flow," Modern Compressible Flow, chap. 1, McGraw-Hill, 3rd ed., 2003, pp. 12–14.

[84] Glauert, H., "The effect of compressibility on the lift of an aerofoil," Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character, Vol. 118, No. 779, 1928, pp. 113–119.

[85] Phillips, W. F., "Wing Aerodynamic Center and Pitching Moment," Mechanics of Flight, chap. 1, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 120–131.

[86] Anderson, J. D., "Lift and Drag Buildup," Aircraft performance and design, chap. 2, McGraw-Hill, 1st ed., 1999, pp. 78–126.

[87] Cooper, M. and Korycinski, P. F., "The Effects of Compressibility on the Lift, Pressure, and Load Characteristics of a Tapered Wing of NACA 66-Series Airfoil Sections," Technical Note 1697, National Advisory Committee for Aeronautics, Langley Field, Va., Oct 1948.

[88] Gothert, B., "Plane and Three-Dimensional Flow at High Subsonic Speeds," Technical Memorandum 1105, National Advisory Committee for Aeronautics, Lilienthal Gesellschaft 127, Oct 1946.

[89] Phillips, W. F. and Snyder, D. O., "Modern Adaptation of Prandtl's Classic Lifting-Line Theory," Journal of Aircraft, Vol. 37, No. 4, Jul 2000, pp. 662–670.

[90] Goates, C. D. and Hunsaker, D. F., "Practical Implementation of a General Numerical Lifting-Line Method," AIAA Scitech 2021 Forum, American Institute of Aeronautics and Astronautics, Jan 2021.

[91] Phillips, W. F., "Linearized Equations of Motion," Mechanics of Flight, chap. 7, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 754–788.

[92] Anderson, J. D., "Applied Aerodynamics: Airplane Lift and Drag," Fundamentals of Aerodynamics, chap. 6, McGraw-Hill Education, 6th ed., 2017, pp. 512–523.

[93] Phillips, W. F., A. N. R. G. W. D., "Lifting-Line Analysis of Roll Control and Variable Twist," Journal of Aircraft, Vol. 41, No. 5, Sep 2004, pp. 1169–1176.

[94] Phillips, W. F., "Effects of Drag and Vertical Offset," Mechanics of Flight, chap. 4, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 436–458.

[95] Hoerner, S. F., "Induced Drag and Aspect Ratio," Fluid-Dynamic Drag, chap. 7, Sighard F. Hoerner, 2nd ed., 1965, pp. 7–2.

[96] Raymer, D. P., "Drag Due to Lift (Induced Drag)," Aircraft Design: A Conceptual Approach, chap. 12, American Institute of Aeronautics and Astronautics, Inc., 2nd ed., 1992, pp. 297–305.

[97] Phillips, W. F., "Incompressible Flow over Airfoils," Mechanics of Flight, chap. 1, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 26–39.

[98] Loftin Jr., L. K., "Theoretical and Experimental Data for a Number of NACA 6A-Series Airfoil Sections," Research Memorandum L6J01, National Advisory Committee for Aeronautics, Langley Field, Va., Dec 1946.

[99] Abbott, Ira H., v. D. A. E. and Stivers Jr., L. S., "Summary of Airfoil Data," Technical Report 824, National Advisory Committee for Aeronautics, Langley Field, Va., 1945.

[100] Chapra, S. C. and Canale, R. P., Numerical Methods for Engineers, McGraw-Hill Education, 7th ed., 2015.

[101] Raymer, D. P., "Tail Geometry and Arrangement," Aircraft Design: A Conceptual Approach, chap. 4, AIAA Education Series, 1st ed., 1992, pp. 67–76.

[102] Anderson, J. D., "The Lifting-Surface Theory and the Vortex Lattice Numerical Method," Fundamentals of Aerodynamics, chap. 5, McGraw-Hill Education, 6th ed., 2017, pp. 469–475.

[103] Service, F. S., Pilot's Handbook of Aeronautical Knowledge, United States Department of Transportation, Federal Aviation Administration, Airman Testing Standards Branch, 25th ed., 2016.

[104] Phillips, W. F., "Flow over Multiple Lifting Surfaces," Mechanics of Flight, chap. 1, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 94–107.

[105] Phillips, W. F., "Lateral Control and Maneuverability," Mechanics of Flight, chap. 6, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 666–679.

[106] Phillips, W. F., "Steady-Heading Sideslip," Mechanics of Flight, chap. 5, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 577–582.

[107] Simon, J., Blake, W., and Multhopp, D., "Control Concepts for a Vertical Tailless Fighter," Aircraft Design, Systems, and Operations Meeting, Aug 1993, p. 4000.

[108] Phillips, W. F., "Engine Failure and Minimum-Control Airspeed," Mechanics of Flight, chap. 5, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 582–596.

[109] Conners, T. and Sims, R., "Full flight envelope direct thrust measurement on a supersonic aircraft," 34th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, American Institute of Aeronautics and Astronautics, Jul 1998.

[110] “Design airspeeds,” Standard, Federal Aviation Administration, Washington, DC, Dec. 1964.

[111] Phillips, W. F., “Takeoff and Landing Performance,” Mechanics of Flight, chap. 3, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 337–353.

[112] Phillips, W. F., “The Steady Coordinated Turn,” Mechanics of Flight, chap. 3, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 319–337.

[113] Raymer, D. P., “Thrust-To-Weight Ratio,” Aircraft Design: A Conceptual Approach, chap. 5, AIAA Education Series, 1st ed., 1992, pp. 78–84.

[114] Clayton, J. D., Haller, R. L., and Hassler Jr., J. M., “Design and Fabrication of the NASA Decoupler Pylon for the F-16 Aircraft,” Tech. Rep. NASA-CR-172354, Jan. 1985.

[115] Hagen, T., Wahlen, F. T., and Corti, W. R., Nepal: the kingdom in the Himalayas, Kümmerly & Frey; distributed in USA by Rand McNally, Chicago, 1961.

[116] Durham, W. C., “Attainable Moments for the Constrained Control Allocation Problem,” Journal of Guidance, Control, and Dynamics, Vol. 17, No. 6, 1994, pp. 1371–1373.

[117] Durham, W. C., “Constrained Control Allocation:Three-Moment Problem,” Journal of Guidance, Control, and Dynamics, Vol. 17, No. 2, 1994, pp. 330–336.

[118] Bolender, M. A. and Doman, D. B., “Method for Determination of Nonlinear Attainable Moment Sets,” Journal of guidance, control, and dynamics, Vol. 27, No. 5, 2004, pp. 907–914.

[119] Nocedal, J. and Wright, S. J., Numerical optimization, Springer, 1999.

[120] Rockafellar, R. T., Convex analysis, Vol. 36, Princeton university press, 1970.

[121] Akkiraju, N., Edelsbrunner, H., Facello, M., Fu, P., Mucke, E., and Varela, C., “Alpha shapes: definition and software,” Proceedings of the 1st International Computational Geometry Software Workshop, Vol. 63, 1995, p. 66.

[122] Barbarino, S., Bilgen, O., Ajaj, R. M., Friswell, M. I., and Inman, D. J., “A Review of Morphing Aircraft,” Journal of Intelligent Material Systems and Structures, Vol. 22, No. 9, June 2011, pp. 823–877.

[123] Hunsaker, Douglas F., M. Z. S. J. J. J., “Lifting-Line Analysis of Wing Twist to Minimize Induced Drag During Pure Rolling Motion,” AIAA Scitech 2019 Forum, American Institute of Aeronautics and Astronautics, Jan 2019.

[124] Hunsaker, D. F., Montgomery, Z. S., and Joo, J. J., “Adverse-Yaw Control During Roll for a Class of Optimal Lift Distributions,” AIAA Journal, Vol. 58, No. 7, 2020, pp. 2909–2920.

[125] Montgomery, Z. S., “Control Mapping Methodology for Tailless Morphing-Wing Aircraft,” 2022.

[126] Nelder, J. A. and Mead, R., "A simplex method for function minimization," The computer journal, Vol. 7, No. 4, 1965, pp. 308–313.

[127] Spendley, W., Hext, G. R., and Himsworth, F. R., "Sequential application of simplex designs in optimisation and evolutionary operation," Technometrics, Vol. 4, No. 4, 1962, pp. 441–461.

[128] Skogestad, S. and Postlethwaite, I., Multivariable Feedback Control: Analysis and Design, John Wiley & Sons, 2nd ed., 2005.

[129] Skogestad, S. and Postlethwaite, I., "Poles," Multivariable Feedback Control: Analysis and Design, chap. 4, John Wiley & Sons, 2nd ed., 2005, pp. 135–138.

[130] Khalil, H. K., "Control Problems," Nonlinear Systems, chap. 12, Pearson Education Inc., 3rd ed., 2014, pp. 469–475.

[131] Skogestad, S. and Postlethwaite, I., "State controllability and state observability," Multivariable Feedback Control: Analysis and Design, chap. 4, John Wiley & Sons, 2nd ed., 2005, pp. 127–134.

[132] Moon, T. K. and Stirling, W. C., Mathematical Methods and Algorithms for Signal Processing, Prentice-Hall, Inc., 1st ed., 2000.

[133] Skogestad, S. and Postlethwaite, I., "LQG control," Multivariable Feedback Control: Analysis and Design, chap. 9, John Wiley & Sons, 2nd ed., 2005, pp. 344–352.

[134] Skogestad, S. and Postlethwaite, I., "Trade-offs in MIMO feedback design," Multivariable Feedback Control: Analysis and Design, chap. 9, John Wiley & Sons, 2nd ed., 2005, pp. 342–344.

[135] Phillips, W. F., "Summary of Flat-Earth Quaternion Formulation," Mechanics of Flight, chap. 11, John Wiley & Sons, Inc., 1st ed., 2004, pp. 915–922.

[136] Hairer, E., Nørsett, S. P., and Wanner, G., Solving Ordinary Differential Equations. 1, Nonstiff problems, Springer-Verlag, 1993.

[137] Hoak, D. and Finck, R., "USAF stability and control DAT-COM," Tech. rep., AFWAL TR-83-3048, October 1960, Revised, 1978.

[138] Phillips, W. F., "Estimating the Downwash Angle on an Aft Tail," Mechanics of Flight, chap. 4, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 411–421.

[139] Phillips, W., Hunsaker, D. F., and Niewoehner, R., "Estimating the Subsonic Aerodynamic Center and Moment Components for Swept Wings," Journal of Aircraft, Vol. 45, No. 3, 2008, pp. 1033–1043.

[140] Feagin, R. C. and Morrison, W. D., "Delta method, an empirical drag buildup technique," Tech. Rep. NASA-CR-151971, Dec. 1978.

[141] Nicolosi, F., Della Vecchia, P., Ciliberti, D., and Cusati, V., "Fuselage aerodynamic prediction methods," Aerospace Science and Technology, Vol. 55, 2016, pp. 332–343.

APPENDICES

APPENDIX A

LINEAR AERODYNAMIC MODEL BUILDUP

The process for developing a linear aerodynamic model using the approach given by Phillips [8] is detailed here. The longitudinal force and moment coefficients for the baseline and BIRE aircraft are discussed in detail, while the procedure for doing the same with the lateral coefficients is left to be completed. Future analysis can be performed by taking the procedure outlined here, calculating the coefficients required for Eqs. (4.27)–(4.31), and comparing the results obtained to those given in Chapter 5 for the nonlinear model.

A.1 Longitudinal Force and Moment Coefficients

The longitudinal coefficients will be examined first by referring to Fig. A.1. Assuming the horizontal is mounted with the root chord aligned with the fuselage reference axis, the lift coefficient on the main wing, horizontal tail, and fuselage can be approximated as

$$C_{L_w} \equiv \frac{L_w}{\frac{1}{2}\rho V^2 S_w} = C_{L_w,\alpha}\left(\alpha + \alpha_{0_w} - \alpha_{L0_w} + 2\overline{q}\frac{l_w}{\overline{c}_w}\right) \tag{A.1}$$

$$C_{L_h} \equiv \frac{L_h}{\frac{1}{2}\rho V^2 S_h} = C_{L_h,\alpha}\left(\alpha + \alpha_{0_h} - \alpha_{L0_h} - \varepsilon_d + 2\overline{q}\frac{l_h}{\overline{c}_w} + \delta_e\right) \tag{A.2}$$

$$C_{L_f} \equiv \frac{L_f}{\frac{1}{2}\rho V^2 S_f} = C_{L_f,\alpha}\left(\alpha + 2\overline{q}\frac{l_f}{\overline{c}_w}\right) \tag{A.3}$$

In Eqs. (A.1)–(A.3), several new terms are introduced. The lift slopes of the main wing, horizontal tail, and fuselage are given by $C_{L_w,\alpha}$, $C_{L_h,\alpha}$, and $C_{L_f,\alpha}$, respectively. As defined in Chapter 4, α represents the angle of attack, while α_0 represents the mounting angle of the wing, and α_{L_0} is the zero-lift angle of attack of each lifting surface. The term ε_d is the downwash angle induced on the horizontal tail by the main wing and the terms $\overline{q}$ and δ_e have been defined previously as the nondimensional pitch rate and elevator deflection angle. Three longitudinal reference lengths, shown in Fig. A.2, are defined as the distances between the center of gravity and the aerodynamic center of the main wing, l_w, horizontal tail, l_h, and center of pressure of the fuselage, l_h. Finally, the reference area of the fuselage is denoted S_f.

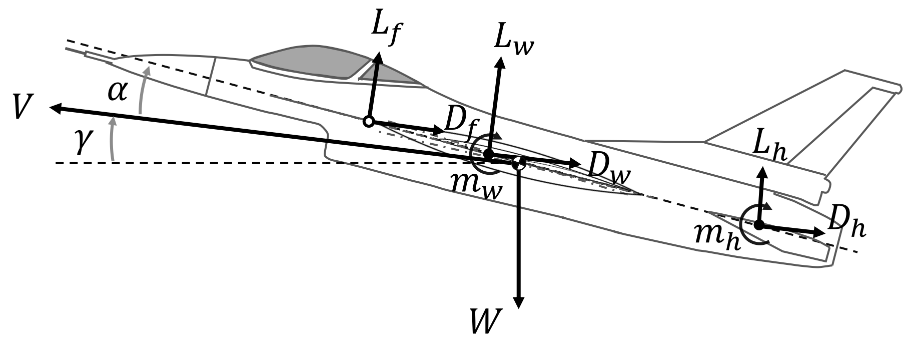


Fig. A.1: A free-body diagram of the longitudinal forces and moments acting on the baseline aircraft.

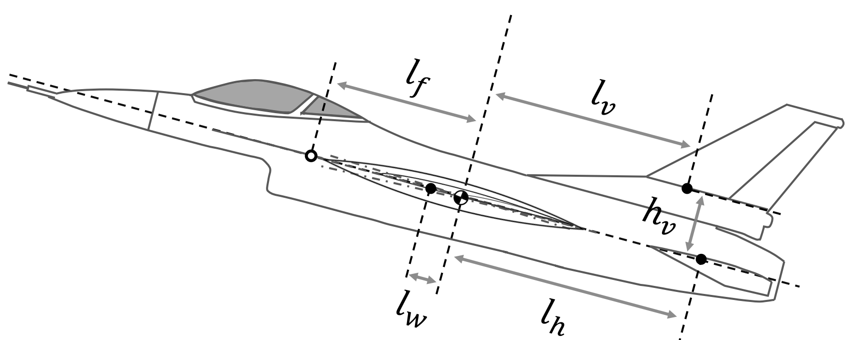


Fig. A.2: Longitudinal reference lengths for the baseline aircraft.

By defining the total lift coefficient as the sum of the components in Eqs. (A.1)–(A.3)

$$C_L = \frac{L}{\frac{1}{2}\rho V^2 S_w} \equiv C_{L_w} + \frac{S_h}{S_w} C_{L_h} + \frac{S_f}{S_w} C_{L_f} \tag{A.4}$$

and scaling the lift produced by the horizontal tail and fuselage by the appropriate ratios for a consistent nondimensionalization, the total lift coefficient on the baseline aircraft can be written in the form given in Eq. (4.35). The individual coefficients are then given as

$$C_{L_0} = C_{L_w,\alpha}\left(\alpha_{0_w} - \alpha_{L_{0_w}}\right) + \frac{S_h}{S_w} C_{L_h,\alpha}\left(\alpha_{0_h} - \alpha_{L_{0_h}} - \varepsilon_{d_0}\right) \tag{A.5}$$

$$C_{L,\alpha} = C_{L_w,\alpha} + \frac{S_h}{S_w} C_{L_h,\alpha}\left(1 - \varepsilon_{d,\alpha}\right) + \frac{S_f}{S_w} C_{L_f,\alpha} \tag{A.6}$$

$$C_{L,\bar{q}} = \frac{2l_w}{\bar{c}_w} C_{L_w,\alpha} + \frac{2S_h l_h}{S_w \bar{c}_w} C_{L_h,\alpha} + \frac{2S_f l_f}{S_w \bar{c}_w} C_{L_f,\alpha} \tag{A.7}$$

$$C_{L,\delta_e} = \frac{S_h}{S_w} C_{L_h,\alpha} \tag{A.8}$$

where the downwash angle, ε_d has been defined as a linear function in angle of attack

$$\varepsilon_d = \varepsilon_{d_0} + \varepsilon_{d,\alpha}\alpha \tag{A.9}$$

At incompressible, subsonic speeds, the lift slope of a lifting surface can be estimated based upon the lift slope of the airfoil section as suggested by Phillips [85]

$$C_{L,\alpha} = \frac{\tilde{C}_{L,\alpha}\kappa_{L_\alpha}}{[1 + \tilde{C}_{L,\alpha}/(\pi R_A)](1 + \kappa_L)} \tag{A.10}$$

where κ_{L_α} and κ_L are empirical factors relating to the three-dimensional effects of sweep and lift slope respectively. These coefficients are functions of taper ratio and aspect ratio; therefore, referring to Table 3.2, κ_L and κ_{L_α} can be estimated to be the values in Table A.1 [24,85].

Parameter	**Main Wing**	**Horizontal Tail**	**Fuselage**
Lift Factor, κ_L	0.011	0.01	–
Sweep Factor, κ_{L_α}	1.06	1.08	–
Surface Lift Slope, $C_{L,\alpha}$ [1/rad]	3.953	3.454	1.806

Table A.1: Lift slope parameters for incompressible flow.

The theoretical lift slope for a thin airfoil is quite accurate for speeds below Mach 0.3; however, since most fighter aircraft spend a significant amount of time at velocities above Mach 0.3, some level of compressibility should be accounted for when analyzing this aircraft.

Thus, the compressibility correction given in Eq. (4.26) can be used to adjust the lift slopes for compressibility effects.

The lift slope of the fuselage can be estimated using a rough empirical approximation suggested by Hoak and Finck [67, 137]

$$C_{L_f,\alpha} \approx 2\left[1 - 1.76\left(\frac{d_f}{c_f}\right)^{3/2}\right] \tag{A.11}$$

This empirical estimate is a function of the maximum cross-sectional area of the fuselage S_f, the length of the fuselage, c_f, and the diameter of a circle defined as

$$d_f \equiv 2\sqrt{S_f/\pi} \tag{A.12}$$

Returning to Fig. 3.2a, the maximum cross-sectional area is approximately at the fuselage station where the engine inlet begins. The front-view in Fig. 3.2a shows that this area is approximately elliptical in nature, with the semi-major axis equal to the distance from the centerline of the aircraft to the initial spanwise location of the rudder. The semi-minor axis equal to the distance from the centerline to the root of the stabilator. Thus, the approximate maximum cross-sectional area of the fuselage is $S_f = 39.995$ ft^2, the length of the fuselage is $c_f = 49.34$ ft, and the diameter of the circle given by $d_f = 7.136$ ft. This gives an estimate for the lift slope of the fuselage as given in Table A.1. To account for the effects of compressibility at subsonic speeds, the simple Prandtl-Glauert correction, given in Eq. (4.25), will be used.

In Chapter 3, it was assumed that the main wing and horizontal tail had no mounting angle and negligible twist. Under this assumption, the terms α_{0_w} and α_{0_h} disappear from Eq. (A.6) and the terms α_{L0_w} and α_{L0_h} are equivalent to the section values of the corresponding airfoils as given in Table 5.5.

The downwash angle ε_d acting on the horizontal tail can be approximated as suggested by Phillips [138]

$$\varepsilon_d \approx \frac{\kappa_v \kappa_p \kappa_s}{\kappa_b} \frac{C_{L_w}}{R_{A_w}} = \varepsilon_{d_0} + \varepsilon_{d,\alpha}\alpha \tag{A.13}$$

where

$$\varepsilon_{d_0} = \frac{\kappa_v \kappa_p \kappa_s}{\kappa_b} \frac{C_{L_w}|_{\alpha=0}}{R_A} = \frac{\kappa_v \kappa_p \kappa_s}{\kappa_b} \frac{C_{L_w,\alpha} \left(\alpha_{0w} - \alpha_{L0w}\right)}{R_A} \tag{A.14}$$

and

$$\varepsilon_{d,\alpha} = \frac{\kappa_v \kappa_p \kappa_s}{\kappa_b} \frac{C_{L_w,\alpha}}{R_A} \tag{A.15}$$

The factor κ_v is a correction factor for the vortex strength of the main wing that adjusts its value compared to the strength of the vortices on an elliptic wing with the same lift coefficient and aspect ratio. To correct for the spanwise location of the wingtip vortices, κ_b is used. Finally, the term κ_p adjusts the downwash experienced by the tail by taking into account its position relative to the wing and κ_s adjusts the downwash by factoring in the sweep of the main wing.

The various factors listed above for calculating the downwash are given by Phillips and included in Table A.2 [138]. Also included are the resulting incompressible downwash angle, ε_{d_0}, and the incompressible downwash slope, $\varepsilon_{d,\alpha}$ calculated from Eqs. (A.14) and (A.15), respectively. Since both of the downwash coefficients is a function of the lift slope of the main wing, they can be corrected for compressibility effects using Eq. (4.26).

Parameter	**Value**
Wingtip Vortex Strength Factor, κ_v	1.05
Wingtip Vortex Span Factor, κ_b	0.74
Nondimensional Downstream Distance, $\overline{x}$	1.199
Nondimensional Vertical Distance, $\overline{y}$	0
Tail Position Factor, κ_p	0.441
Wing Sweep Factor, κ_s	1.037
Incompressible Downwash Angle, ε_{d_0} [rad]	0.019
Incompressible Downwash Slope, $\varepsilon_{d,\alpha}$ [1/rad]	0.855

Table A.2: Factors used for calculating the downwash induced by the main wing.

The final parameters that must be defined to solve for Eqs. (A.5)–(A.8) are the geometric reference lengths In Table 3.2, the x-coordinate of the quarter-chord of each lifting surface is noted. For each lifting surface, the location of the aerodynamic center will be shifted due to sweep. This axial shift in the location of the aerodynamic center relative to the aerodynamic center at the root of an unswept wing can be estimated using a relation suggested by Phillips [85]

$$\frac{x_{\mathrm{ac}} - x_{\mathrm{ac}_{\mathrm{root}}}}{\bar{c}_g} \approx \kappa_{\mathrm{ac}} R_A \left(\frac{\bar{z}_{\mathrm{ac}}}{b}\right)_{\Lambda=0} \tan \Lambda_{c/4} \tag{A.16}$$

where κ_{ac} is an empirical sweep correction factor. The term $(\bar{z}_{\mathrm{ac}}/b)_{\Lambda=0}$ represents the semispan location of the aerodynamic center of a wing with zero sweep and linear taper of the same planform as the swept wing being analyzed. Note that this shift is made in reference to the mean geometric chord of each surface, $\bar{c}_g$, and that a positive value for $\frac{x_{\mathrm{ac}} - x_{\mathrm{ac}_{\mathrm{root}}}}{\bar{c}_g}$ represents a backward shift (in the negative body-fixed x-direction).

Estimates for the parameters required in Eq. (A.16) can be obtained from plots showing each parameter as a function of taper ratio and aspect ratio [24, 85, 139]. Results of these estimates are shown in Table A.3 for each lifting surface. From this information, the moment arm lengths in Eqs. (A.5)–(A.8) for the main wing and horizontal tail can be calculated as

$$l_w = x_{\mathrm{cg}_w} - \left(\frac{(x_{\mathrm{ac}} - x_{\mathrm{ac}_{\mathrm{root}}})}{\bar{c}_g}\right)_w \bar{c}_{g_w} \tag{A.17}$$

and

$$l_h = |x_{\mathrm{cg}_h}| + \left(\frac{(x_{\mathrm{ac}} - x_{\mathrm{ac}_{\mathrm{root}}})}{\bar{c}_g}\right)_h \bar{c}_{g_h} \tag{A.18}$$

Similarly, the moment arm for the vertical tail will be used in later calculations and is given as

$$l_v = |x_{\mathrm{cg}_v}| + \left(\frac{(x_{\mathrm{ac}} - x_{\mathrm{ac}_{\mathrm{root}}})}{\bar{c}_g}\right)_v \bar{c}_{g_v} \tag{A.19}$$

In Fig. A.2, l_f represents the distance from the center of gravity of the aircraft to the center of pressure of the fuselage. The location of the center of pressure of the fuselage can be roughly estimated as half the distance between the nose of the fuselage and the location of maximum cross section of the fuselage as suggested by Hoak and Finck [67, 137]. Assuming that the leading-edge of the main wing root is located at approximately the same

Parameter	Main Wing	Horizontal Tail	Vertical Tail
Unswept Span Fraction, $\left(\frac{\bar{z}_{\text{ac}}}{b}\right)_{\Lambda_{c/4}=0}$	0.207	.212	0.213
Wing Sweep Factor, κ_{ac}	1.12	1.10	0.93
AC Shift Fraction, $\frac{\left(x_{\text{ac}}-x_{\text{ac}_{\text{root}}}\right)}{\bar{c}_g}$	0.435	0.308	0.266

Table A.3: Parameters for estimating axial shift in location of the aerodynamic center for each lifting surface.

fuselage station as the location of maximum cross sectional area, the distance to the center of pressure can be calculated as

$$l_f \approx \frac{1}{2}\left(c_f - x_{\text{LE}-\text{LE}_h} - c_{r_h}\right) \tag{A.20}$$

Table A.4 shows the values for the moments arms contained in Eqs. (A.5)–(A.8) as calculated using Eqs. (A.17)–(A.20).

Moment Arm	Value
Main Wing Moment Arm, l_w [ft]	0.514
Horizontal Tail Moment Arm, l_h [ft]	11.419
Vertical Tail Moment Arm, l_v [ft]	7.273
Fuselage Center of Pressure Moment Arm, l_f [ft]	10.628

Table A.4: Estimated moment arm values from Fig. A.2.

From the information gathered to this point, values for C_{L_0}, $C_{L,\alpha}$, $C_{L,\bar{q}}$, and C_{L,δ_e} can be calculated assuming incompressible flow. When compressibility corrections are required, the total lift coefficients, given in Eq. (4.35), can be modified using Eq. (4.26). The incompressible lift coefficient components using the analytical solutions presented thus far are given in Table A.5.

The drag coefficient can be built in a similar manner by considering the induced drag acting on each of the lifting surfaces. These are written as

$$C_{D_w} = \frac{D_w}{\frac{1}{2}\rho V^2 S_w} = \frac{C_{L_w}^2}{\pi R_{A_w} e_{s_w}} \tag{A.21}$$

Linear Coefficients	Component Contributions			
	Main Wing	**Horizontal Tail**	**Fuselage**	**Total**
C_{L_0}	0.088	-0.014	0	0.074
$C_{L,\alpha}$	3.953	0.106	0.241	4.300
$C_{L,\bar{q}}$	0.359	1.479	0.452	2.290
C_{L,δ_e}	0	0.733	0	0.733

Table A.5: Component lift coefficients used in the linear aerodynamic model for the baseline aircraft.

and

$$C_{D_h} = \frac{D_h}{\frac{1}{2}\rho V^2 S_h} = \frac{C_{L_h}^2}{\pi R_{A_h} e_{s_h}} \tag{A.22}$$

The drag contribution from the fuselage lift will be very small when compared to that produced by the main wing and horizontal tail. Additionally, fuselage lift drag is generally not considered as a prominent part of many drag build-up analyses and is less important in this analysis than the drag produced by the wing and tail [140, 141]. Therefore, its contribution will be ignored here. Summing Eqs. (A.21) and (A.22) gives the total drag

$$C_D = C_{D_w} + \frac{S_h}{S_w} C_{D_h} \tag{A.23}$$

Keeping only the linear terms from evaluating the components of Eq. (A.23) and writing it in the form given by Eq. (4.42) gives

$$C_{D_0} = \frac{1}{\pi R_{A_w} e_{s_w}} C_{L_w,\alpha}^2 \left(\alpha_{0_w} - \alpha_{L_{0_w}}\right)^2 + \frac{S_h}{S_w \pi R_{A_h} e_{s_h}} C_{L_h,\alpha}^2 \left(\alpha_{0_h} - \alpha_{L_{0_h}} - \varepsilon_{d_0}\right)^2 \tag{A.24}$$

$$C_{D,\alpha} = 2\frac{1}{\pi R_{A_w} e_{s_w}} C_{L_w,\alpha}^2 \left(\alpha_{0_w} - \alpha_{L_{0_w}}\right) + 2\frac{S_h}{S_w \pi R_{A_h} e_{s_h}} C_{L_h,\alpha}^2 \left(1 - \varepsilon_{d,\alpha}\right) \left(\alpha_{0_h} - \alpha_{L_{0_h}} - \varepsilon_{d_0}\right) \tag{A.25}$$

$$C_{D,\bar{q}} = 4\frac{l_w}{\bar{c}_w \pi R_{A_w} e_{s_w}} C_{L_w,\alpha}^2 \left(\alpha_{0_w} - \alpha_{L_{0_w}}\right) + 4\frac{S_h l_h}{S_w \bar{c}_w \pi R_{A_h} e_{s_h}} C_{L_h,\alpha}^2 \left(\alpha_{0_h} - \alpha_{L_{0_h}} - \varepsilon_{d_0}\right) \tag{A.26}$$

$$C_{D,\delta_e} = 2\frac{S_h}{S_w \pi R_{A_h} e_{s_h}} C_{L_h,\alpha}^2 \left(\alpha_{0_h} - \alpha_{L_{0_h}} - \varepsilon_{d_0}\right) \tag{A.27}$$

Each of these variables has been previously defined and given a value, except for the drag efficiency factor, defined in Eq. (4.41) using the induced drag factor κ_D. Both of these are included in Table A.6, though it can be easily noted that these approximations will have very little effect on the resulting drag coefficients. The incompressible drag components are included in Table A.7 and listed according to the contributions of the main wing and horizontal tail.

Parameter	**Main Wing**	**Horizontal Tail**	**Vertical Tail**
Induced Drag Factor, κ_D	0.005	0.002	0.001
Drag Efficiency Factor, e_s	0.995	0.998	0.999

Table A.6: Induced drag parameters for incompressible flow.

Linear Coefficients	**Component Contributions**		
	Main Wing	**Horizontal Tail**	**Total**
C_{D_0}	0.0008	0.0001	0.001
$C_{D,\alpha}$	0.0740	-0.0021	0.0719
$C_{D,\bar{q}}$	0.0067	-0.0292	-0.0225
C_{D,δ_e}	0	-0.0145	0.0145

Table A.7: Component drag coefficients used in the linear aerodynamic model for the baseline aircraft.

Returning again to Fig. A.1, the sum of the pitching moments about the center of gravity, considered positive when causing the nose to pitch up, are given as

$$C_m \equiv \frac{m}{\frac{1}{2}\rho V^2 S_w \bar{c}_w} = C_{m_w} + \frac{l_w}{\bar{c}_w} C_{L_w} + \frac{S_f l_f}{S_w \bar{c}_w} C_{L_f} + \frac{S_h \bar{c}_h}{S_w \bar{c}_w} C_{m_h} - \frac{S_h l_h}{S_w \bar{c}_w} C_{L_h} \tag{A.28}$$

The natural pitching moment about the horizontal tail is augmented by the pitching moment created through stabilator deflection and, therefore,

$$C_{m_h} = C_{m_h 0} + C_{m_h,\delta_e} \delta_e \tag{A.29}$$

Substitution of this relationship and those in Eqs. (A.1) and (A.2) yields the relationship in Eq. (4.38) with

$$C_{m_0} = C_{m_w 0} + \frac{l_w}{\bar{c}_w} C_{L_w,\alpha} \left(\alpha_{0_w} - \alpha_{L_{0_w}}\right) + \frac{S_h \bar{c}_h}{S_w \bar{c}_w} C_{m_h 0} - \frac{S_h l_h}{S_w \bar{c}_w} C_{L_h,\alpha} \left(\alpha_{0_h} - \alpha_{L_{0_h}} - \varepsilon_{d_0}\right) \tag{A.30}$$

$$C_{m,\alpha} = \frac{l_w}{\bar{c}_w} C_{L_w,\alpha} + \frac{S_f l_f}{S_w \bar{c}_w} C_{L_f,\alpha} - \frac{S_h l_h}{S_w \bar{c}_w} C_{L_h,\alpha} \left(1 - \varepsilon_{d,\alpha}\right) \tag{A.31}$$

$$C_{m,\bar{q}} = 2\frac{l_w^2}{\bar{c}_w^2} C_{L_w,\alpha} + 2\frac{S_f l_f^2}{S_w \bar{c}_w^2} C_{L_f,\alpha} - 2\frac{S_h l_h^2}{S_w \bar{c}_w^2} C_{L_h,\alpha} \tag{A.32}$$

$$C_{m,\delta_e} = \frac{S_h \bar{c}_h}{S_w \bar{c}_w} C_{m_h,\delta_e} - \frac{S_h l_h}{S_w \bar{c}_w} C_{L_h,\alpha} \tag{A.33}$$

Neglecting twist in the main wing and horizontal tail yields a pitching moment about the aerodynamic center of the wing equivalent to the pitching moment of the airfoil section about the section aerodynamic center [85]. This means that the section quarter-chord pitching moment of the NACA 64A204 airfoil, equivalent to the zero-lift pitching moment in Table 5.5, can be used to represent $C_{m_w 0}$ and that $C_{m_h 0} = 0$. All that remains is to estimate the change in pitching moment with stabilator deflection of the horizontal tail, C_{m_h,δ_e}. From thin airfoil theory, the section moment slope with respect to stabilator deflection is calculated as [15]

$$\tilde{C}_{m,\delta} = -\frac{1}{2} \int\limits_{\theta=\theta_f}^{\pi} \left[\cos\left(2\theta\right) - \cos\theta\right] \, d\theta = \frac{\sin\left(2\theta_f\right) - 2\sin\theta_f}{4} \tag{A.34}$$

where θ_f is the location of the flap in the change of variable given in Eq. (5.3). For an all-moving tail, $\theta_f = \pi$ and therefore the resulting pitching moment slope with respect to deflection on the horizontal tail is $C_{m_h,\delta_e} = 0$.

Again, values for C_{m_0}, $C_{m,\alpha}$, $C_{m,\bar{q}}$, and C_{m,δ_e} can be calculated assuming incompressible flow, with compressibility corrections given via Eq. (4.26). These coefficient components are calculated and presented in Table A.8. With this, the longitudinal aerodynamic force and moment coefficients have been analyzed for the baseline aircraft. Next, the lateral aerodynamic force and moment coefficients will be analyzed in much the same way.

Linear Coefficients	Component Contributions			
	Main Wing	**Horizontal Tail**	**Fuselage**	**Total**
C_{m_0}	-0.031	0.014	0	-0.017
$C_{m,\alpha}$	0.179	-0.107	0.226	0.298
$C_{m,\overline{q}}$	0.016	-1.492	0.425	-1.051
C_{m,δ_e}	0	-0.740	0	-0.740

Table A.8: Component pitching moment coefficients used in the linear aerodynamic model for the baseline aircraft.

A.2 Lateral Force and Moment Coefficients

The lateral force and moment coefficients could be generated in the same way as the longitudinal forces and moments were previously. These effects are generally coupled with the longitudinal coefficients, however, so their build-up is slightly more complex. Phillips uses a method of "deltas" to approximate these coefficients, which can give appropriate preliminary results. The remaining coefficients in Eqs. (4.36), (4.37), and (4.39) could be approximated using either method, and then physical intuition would be used to identify how each coefficient varies with the BIRE rotation angle. This can be the focus of future work, and a comparison between the work in this dissertation and the results of the linear aerodynamic model could indicate the effect of non-linearities in the trim solution and other analyses.

APPENDIX B

MACHUPX FILES

B.1 Baseline Aircraft Input File

```
{
    "tag" : "Baseline Input File",
    "run" : {
        "display_wireframe" : {
            "show_legend" : true,
            "filename" : "./baseline_wireframe.png"
        },
        "forces" : {
            "non_dimensional" : true
        },
        "pitch_trim" : {
            "set_state_to_trim" : true
        },
        "aero_derivatives" : {},
        "distributions" : {
            "filename" : "baseline_distributions",
            "make_plots" : ["section_CL"]
        },
        "aero_center" : {},
        "stl" : {}
    },
    "solver" : {
        "type" : "linear",
        "convergence" : 1e-8,
        "relaxation" : 0.9,
        "max_iterations" : 1000
    },
    "units" : "English",
    "scene" : {
        "atmosphere" : {
            "rho": "standard"
        },
        "aircraft" : {
            "F16" : {
                "file" : "baseline_airplane.json",
                "state" : {
                    "velocity" : 222.5211,
                    "alpha" : 0.0,
                    "beta" : 0.0
                },
                "control_state" : {
                    "elevator" : 0.0,
                    "rudder" : 0.0,
```

```
                    "aileron" : 0.0
                }
            }
        }
    }
}
```

B.2 Baseline Aircraft Airplane File

```
{
    "CG": [0.0, 0.0, 0.0],
    "weight": 20500.0,
    "reference": {
        "area": 300,
        "longitudinal_length": 11.32,
        "lateral_length": 30.0
    },
    "controls": {
        "aileron": {
            "is_symmetric": false
        },
        "elevator": {
            "is_symmetric": true
        },
        "rudder": {
            "is_symmetric": false
        }
    },
    "airfoils": {
        "NACA_64A204": {
            "type": "linear",
            "aL0": -0.02223,
            "CLa": 6.28319,
            "CmL0": -0.03476,
            "Cma": 0.0,
            "CD0": 0.00368,
            "CD1": -0.00132,
            "CD2": 0.00624,
            "geometry": {
                "outline_points": "64A204.txt"
            }
        },
        "NACA_0005": {
            "type": "linear",
            "aL0": 0.0,
            "CLa": 6.28319,
            "CmL0": 0.0,
            "Cma": 0.0,
            "CD0": 0.00452,
            "CD1": -0.00239,
            "CD2": 0.00762,
            "geometry": {
                "NACA": "0005"
            }
        },
        "NACA_0004": {
            "type": "linear",
            "aL0": 0.0,
            "CLa": 6.28319,
            "CmL0": 0.0,
            "Cma": 0.0,
            "CD0": 0.00452,
            "CD1": -0.00275,
```

```
                "CD2": 0.00821,
                "geometry": {
                    "NACA": "0004"
                }
            }
        },
        "wings": {
            "main_wing": {
                "ID": 1,
                "side": "both",
                "is_main": true,
                "connect_to": {
                    "ID": 0,
                    "dx": 4.8618
                },
                "semispan": 15.0,
                "sweep": 32.0,
                "chord": [[0.0, 16.2933], [1.0, 3.7067]],
                "twist": 0.0,
                "dihedral": 0.0,
                "airfoil": "NACA_64A204",
                "control_surface": {
                    "root_span": 0.23,
                    "tip_span": 0.76,
                    "chord_fraction": [[0.23, 0.22], [ 0.76, 0.22]],
                    "control_mixing": {
                        "aileron": 1.0
                    }
                },
                "grid": {
                    "N": 80,
                    "reid_corrections": true
                }
            },
            "h_stab": {
                "ID": 2,
                "side": "both",
                "is_main": false,
                "connect_to": {
                    "ID": 0,
                    "dx": -13.1
                },
                "semispan": 9.2,
                "sweep": [[0.0, 0.0],[0.37, 0.0],
                          [0.37, 32.0],[1.0, 32.0]],
                "chord": [[0.0,7.9833], [0.37, 7.9833],
                          [0.37, 7.9833], [1.0, 3.1167]],
                "dihedral": [[0.0, 0.0], [0.37, 0.0],
                             [0.37, -10], [1.0, -10]],
                "airfoil": "NACA_0005",
                "control_surface": {
                    "root_span": 0.37,
                    "tip_span": 1.0,
                    "chord_fraction": 1.0,
                    "saturation_angle": 25.0,
                    "control_mixing": {
```

```
                    "elevator": 1.0,
                    "aileron": 0.25
                }
            },
            "grid": {
                "N": 80,
                "reid_corrections": true
            }
        },
        "v_stab": {
            "ID": 3,
            "side": "left",
            "is_main": false,
            "connect_to": {
                "ID": 0,
                "dx": -8.8
            },
            "semispan": 10.5,
            "sweep": [[0.0, 0.0], [0.2, 0.0],
                      [0.2, 38.0], [1.0, 38.0]],
            "chord": [[0.0, 9.06], [0.2,9.06],
                      [0.2, 9.06], [1.0, 3.939]],
            "dihedral": 90.0,
            "airfoil": "NACA_0004",
            "control_surface": {
                "root_span": 0.36,
                "tip_span": 0.95,
                "chord_fraction": [[0.36, 0.32], [0.95, 0.32]],
                "saturation_angle": 30,
                "control_mixing": {
                    "rudder": -1.0
                }
            },
            "grid": {
                "N": 80,
                "reid_corrections": true
            }
        }
    }
}
```

B.3 BIRE Aircraft Input File


```
{
    "tag" : "BIRE Input File",
    "run" : {
        "display_wireframe" : {
            "show_legend" : true,
            "filename" : "./BIRE_wireframe.png"
        },
        "forces" : {
            "non_dimensional" : true
        },
        "pitch_trim" : {
            "set_state_to_trim" : true
        },
        "aero_derivatives" : {},
        "distributions" : {
            "filename" : "BIRE",
            "make_plots" : ["section_CL"]
        },
        "aero_center" : {},
        "stl" : {}
    },
    "solver" : {
        "type" : "linear",
        "convergence" : 1e-10,
        "relaxation" : 0.9,
        "max_iterations" : 1000
    },
    "units" : "English",
    "scene" : {
        "atmosphere" : {
            "rho": "standard"
        },
        "aircraft" : {
            "BIRE" : {
                "file" : "BIRE_airplane.json",
                "state" : {
                    "velocity" : 222.5211,
                    "alpha" : 0.0,
                    "beta" : 0.0
                },
                "control_state" : {
                    "elevator" : 0.0,
                    "aileron" : 0.0
                }
            }
        }
    }
}
```

B.4 BIRE Aircraft Airplane File


```
{
    "CG": [0.0, 0.0, 0.0],
    "weight": 20500.0,
    "reference": {
        "area": 300.0,
        "longitudinal_length": 11.32,
        "lateral_length": 30.0
    },
    "controls": {
        "aileron": {
            "is_symmetric": false
        },
        "elevator": {
            "is_symmetric": true
        },
        "rudder": {
            "is_symmetric": false
        }
    },
    "airfoils": {
        "NACA_64A204": {
            "type": "linear",
            "aL0": -0.02223,
            "CLa": 6.28319,
            "CmL0": -0.03476,
            "Cma": 0.0,
            "CD0": 0.00368,
            "CD1": -0.00132,
            "CD2": 0.00624,
            "geometry": {
                "outline_points": "64A204.txt"
            }
        },
        "NACA_0005": {
            "type": "linear",
            "aL0": 0.0,
            "CLa": 6.28319,
            "CmL0": 0.0,
            "Cma": 0.0,
            "CD0": 0.00452,
            "CD1": -0.00239,
            "CD2": 0.00762,
            "geometry": {
                "NACA": "0005"
            }
        },
        "NACA_0004": {
            "type": "linear",
            "aL0": 0.0,
            "CLa": 6.28319,
            "CmL0": 0.0,
            "Cma": 0.0,
            "CD0": 0.00452,
            "CD1": -0.00275,
```

```
            "CD2": 0.00821,
            "geometry": {
                "NACA": "0004"
            }
        }
    },
    "wings": {
        "main_wing": {
            "ID": 1,
            "side": "both",
            "is_main": true,
            "connect_to": {
                "ID": 0,
                "dx": 4.8618
            },
            "semispan": 15.0,
            "sweep": 32.0,
            "chord": [[0.0, 16.2933], [1.0, 3.7067]],
            "twist": 0.0,
            "dihedral": 0.0,
            "airfoil": "NACA_64A204",
            "control_surface": {
                "root_span": 0.23,
                "tip_span": 0.76,
                "chord_fraction": [[0.23, 0.22], [ 0.76, 0.22]],
                "control_mixing": {
                    "aileron": 1.0
                }
            },
            "grid": {
                "N": 80,
                "reid_corrections": true
            }
        },
        "BIRE_left": {
            "ID": 2,
            "side": "left",
            "is_main": false,
            "connect_to": {
                "ID": 0,
                "dx": -13.1
            },
            "semispan": 9.2,
            "sweep": [[0.0, 0.0], [0.37, 0.0],
                      [0.37, 32.0], [1.0, 32.0]],
            "chord": [[0.0, 7.9833], [0.37, 7.9833],
                      [0.37, 7.9833], [1.0, 3.1167]],
            "dihedral": 0.0,
            "airfoil": "NACA_0005",
            "control_surface": {
                "root_span": 0.37,
                "tip_span": 1.0,
                "chord_fraction": 1.0,
                "saturation_angle": 25.0,
                "control_mixing": {
                    "elevator": 1.0,
```

```
                    "aileron": 0.25
                }
            },
            "grid": {
                "N": 80,
                "reid_corrections": true,
                "wing_ID": 1
            }
        },
        "BIRE_right": {
            "ID": 3,
            "side": "right",
            "is_main": false,
            "connect_to": {
                "ID": 0,
                "dx": -13.1
            },
            "semispan": 9.2,
            "sweep": [[0.0, 0.0], [0.37, 0.0],
                      [0.37, 32.0], [1.0, 32.0]],
            "chord": [[0.0, 7.9833], [0.37, 7.9833],
                      [0.37, 7.9833], [1.0, 3.1167]],
            "dihedral": 0.0,
            "airfoil": "NACA_0005",
            "control_surface": {
                "root_span": 0.37,
                "tip_span": 1.0,
                "chord_fraction": 1.0,
                "saturation_angle": 25.0,
                "control_mixing": {
                    "elevator": 1.0,
                    "aileron": 0.25
                }
            },
            "grid": {
                "N": 80,
                "reid_corrections": true,
                "wing_ID": 1
            }
        }
    }
}
```

B.5 Example BIRE Rotated Tail Airplane File ($\delta_B = 10°$)


```
{
    "CG": [0.0, 0.0, 0.0],
    "weight": 20500.0,
    "reference": {
        "area": 300.0,
        "longitudinal_length": 11.32,
        "lateral_length": 30.0
    },
    "controls": {
        "aileron": {
            "is_symmetric": false
        },
        "elevator": {
            "is_symmetric": true
        },
        "rudder": {
            "is_symmetric": false
        }
    },
    "airfoils": {
        "NACA_64A204": {
            "type": "linear",
            "aL0": -0.02223,
            "CLa": 6.28319,
            "CmL0": -0.03476,
            "Cma": 0.0,
            "CD0": 0.00368,
            "CD1": -0.00132,
            "CD2": 0.00624,
            "geometry": {
                "outline_points": "64A204.txt"
            }
        },
        "NACA_0005": {
            "type": "linear",
            "aL0": 0.0,
            "CLa": 6.28319,
            "CmL0": 0.0,
            "Cma": 0.0,
            "CD0": 0.00452,
            "CD1": -0.00239,
            "CD2": 0.00762,
            "geometry": {
                "NACA": "0005"
            }
        },
        "NACA_0004": {
            "type": "linear",
            "aL0": 0.0,
            "CLa": 6.28319,
            "CmL0": 0.0,
            "Cma": 0.0,
            "CD0": 0.00452,
            "CD1": -0.00275,
```

```
            "CD2": 0.00821,
            "geometry": {
                "NACA": "0004"
            }
        }
    },
    "wings": {
        "main_wing": {
            "ID": 1,
            "side": "both",
            "is_main": true,
            "connect_to": {
                "ID": 0,
                "dx": 4.8618
            },
            "semispan": 15.0,
            "sweep": 32.0,
            "chord": [[0.0, 16.2933], [1.0, 3.7067]],
            "twist": 0.0,
            "dihedral": 0.0,
            "airfoil": "NACA_64A204",
            "control_surface": {
                "root_span": 0.23,
                "tip_span": 0.76,
                "chord_fraction": [[0.23, 0.22], [ 0.76, 0.22]],
                "control_mixing": {
                    "aileron": 1.0
                }
            },
            "grid": {
                "N": 80,
                "reid_corrections": true
            }
        },
        "BIRE_left": {
            "ID": 2,
            "side": "left",
            "is_main": false,
            "connect_to": {
                "ID": 0,
                "dx": -13.1
            },
            "semispan": 9.2,
            "sweep": [[0.0, 0.0], [0.37, 0.0],
                      [0.37, 32.0], [1.0, 32.0]],
            "chord": [[0.0, 7.9833], [0.37, 7.9833],
                      [0.37, 7.9833], [1.0, 3.1167]],
            "dihedral": -10.0,
            "airfoil": "NACA_0005",
            "control_surface": {
                "root_span": 0.37,
                "tip_span": 1.0,
                "chord_fraction": 1.0,
                "saturation_angle": 25.0,
                "control_mixing": {
                    "elevator": 1.0,
```

```
                    "aileron": 0.25
                }
            },
            "grid": {
                "N": 80,
                "reid_corrections": true,
                "wing_ID": 1
            }
        },
        "BIRE_right": {
            "ID": 3,
            "side": "right",
            "is_main": false,
            "connect_to": {
                "ID": 0,
                "dx": -13.1
            },
            "semispan": 9.2,
            "sweep": [[0.0, 0.0], [0.37, 0.0],
                      [0.37, 32.0], [1.0, 32.0]],
            "chord": [[0.0, 7.9833], [0.37, 7.9833],
                      [0.37, 7.9833], [1.0, 3.1167]],
            "dihedral": 10.0,
            "airfoil": "NACA_0005",
            "control_surface": {
                "root_span": 0.37,
                "tip_span": 1.0,
                "chord_fraction": 1.0,
                "saturation_angle": 25.0,
                "control_mixing": {
                    "elevator": 1.0,
                    "aileron": 0.25
                }
            },
            "grid": {
                "N": 80,
                "reid_corrections": true,
                "wing_ID": 1
            }
        }
    }
}
```

B.6 Other JSON Files

BIRE Inertia Input File


```
{
    "Ixx": {
        "A": 0.0,
        "w": 0.0,
        "phi": 0.0,
        "z": 9280.0
    },
    "Iyy": {
        "A": -160.80701824266598,
        "w": 2.0,
        "phi": 1.5707963267948966,
        "z": 58287.86099859672
    },
    "Izz": {
        "A": 160.83498082043252,
        "w": 2.0,
        "phi": 1.5707963267948966,
        "z": 65605.60269083234
    },
    "Ixy": {
        "A": 0.0,
        "w": 0.0,
        "phi": 0.0,
        "z": 0.0
    },
    "Ixz": {
        "A": 0.0,
        "w": 0.0,
        "phi": 0.0,
        "z": -5.0
    },
    "Iyz": {
        "A": -160.5850207505512,
        "w": 2.0,
        "phi": 0.0,
        "z": 160.5850207505512
    }
}
```

Baseline Aerodynamic Model


```
{
    "CL": {
        "CL_0": 0.0456,
        "CL_alpha": 3.5791,
        "CL_qbar": 3.3916,
        "CL_de": 0.5652
    },
    "CS": {
        "CS_beta": -0.9009,
        "CS_pbar": -0.0153,
        "CS_Lpbar": 0.3318,
        "CS_rbar": 0.4357,
        "CS_da": 0.0656,
        "CS_dr": 0.1698
    },
    "CD": {
        "CD_0": 0.0218,
        "CD_L": -0.034,
        "CD_L2": 0.1834,
        "CD_S2": 0.6081,
        "CD_Spbar": 0.0768,
        "CD_qbar": 0.0368,
        "CD_Lqbar": 0.775,
        "CD_L2qbar": -0.1844,
        "CD_Srbar": -0.7239,
        "CD_de": -0.0032,
        "CD_Lde": 0.1775,
        "CD_de2": 0.2854,
        "CD_Sda": 0.1118,
        "CD_Sdr": 0.1604
    },
    "Cell": {
        "Cl_beta": -0.0786,
        "Cl_pbar": -0.3182,
        "Cl_rbar": 0.0469,
        "Cl_Lrbar": 0.1067,
        "Cl_da": -0.0741,
        "Cl_dr": 0.0257
    },
    "Cm": {
        "Cm_0": -0.0097,
        "Cm_alpha": 0.1766,
        "Cm_qbar": -4.8503,
        "Cm_de": -0.5881
    },
    "Cn": {
        "Cn_beta": 0.2426,
        "Cn_pbar": 0.0131,
        "Cn_Lpbar": -0.1005,
        "Cn_rbar": -0.1787,
        "Cn_da": -0.0276,
        "Cn_Lda": 0.0077,
        "Cn_dr": -0.0899
```

```
    }
}
```

BIRE Aerodynamic Model


```
{
    "CL": {
        "CL_0": {
            "A": -0.014442279065396085,
            "w": 2.0,
            "phi": 1.5707963267948966,
            "z": 0.062077939554914544,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CL_alpha": {
            "A": 0.10910386001589227,
            "w": 2.0,
            "phi": 1.5707963267948966,
            "z": 3.54694564679198,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CL_beta": {
            "A": -0.7215865846558561,
            "w": 2.0,
            "phi": 0.0,
            "z": 0.0,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CL_pbar": {
            "A": 0.0,
            "w": 0.0,
            "phi": 0.0,
            "z": 0.0,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CL_qbar": {
            "A": 2.0261831569744433,
            "w": 2.0,
            "phi": 1.5707963267948966,
            "z": 1.546911315205368,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CL_rbar": {
            "A": 0.6797873262695825,
            "w": 2.0,
            "phi": 0.0,
            "z": 0.0,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CL_da": {
            "A": 0.0,
            "w": 0.0,
            "phi": 0.0,
```

```
                "z": -0.0006604559803619685,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "CL_de": {
                "A": 0.7646092720773868,
                "w": 1.0,
                "phi": 1.5707963267948966,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": -0.1822
            }
        },
        "CS": {
            "CS_0": {
                "A": -0.010598861662413759,
                "w": 2.0,
                "phi": 0.0,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "CS_alpha": {
                "A": 0.18338959119803913,
                "w": 2.0,
                "phi": 0.0,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "CS_beta": {
                "A": 0.6805478426692505,
                "w": 2.0,
                "phi": 1.5707963267948966,
                "z": -0.6707797852000175,
                "multiplier": 1.0,
                "delta": -0.1785
            },
            "CS_pbar": {
                "A": 0.0,
                "w": 0.0,
                "phi": 0.0,
                "z": -0.002241226063773226,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "CS_Lpbar": {
                "A": 0.019221962808743775,
                "w": 2.0,
                "phi": 1.5707963267948966,
                "z": 0.22327561055279319,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "CS_qbar": {
                "A": 1.9915667103205594,
```

```
            "w": 2.0,
            "phi": 0.0,
            "z": 0.0,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CS_rbar": {
            "A": -0.6134464898985744,
            "w": 2.0,
            "phi": 1.5707963267948966,
            "z": 0.5975930548706397,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CS_da": {
            "A": 0.001455253803870835,
            "w": 2.0,
            "phi": 1.5707963267948966,
            "z": -0.007582238803361364,
            "multiplier": 1.0,
            "delta": -0.0448
        },
        "CS_de": {
            "A": 0.7351623591166163,
            "w": 1.0,
            "phi": 0.0,
            "z": 0.0,
            "multiplier": 1.0,
            "delta": 0.0
        }
    },
    "CD": {
        "CD_0": {
            "A": 0.0,
            "w": 0.0,
            "phi": 0.0,
            "z": 0.005462920924405871,
            "multiplier": 1.0,
            "delta": 0.0154
        },
        "CD_L": {
            "A": 0.0,
            "w": 0.0,
            "phi": 0.0,
            "z": -0.002817424078462198,
            "multiplier": 1.0,
            "delta": -0.0304
        },
        "CD_L2": {
            "A": 0.004728784703746916,
            "w": 4.0,
            "phi": 1.5707963267948966,
            "z": 0.10525681876055337,
            "multiplier": 1.0,
            "delta": 0.0714
        },
```

```
        "CD_S": {
            "A": 0.025489272604736844,
            "w": 2.0,
            "phi": 0.0,
            "z": -2.097578362584658e-10,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CD_S2": {
            "A": 0.30818430801791286,
            "w": 2.0,
            "phi": 1.5707963267948966,
            "z": 0.5245693583006585,
            "multiplier": 1.0,
            "delta": 0.1118
        },
        "CD_pbar": {
            "A": 0.0,
            "w": 0.0,
            "phi": 0.0,
            "z": 0.0,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CD_Spbar": {
            "A": 0.0,
            "w": 0.0,
            "phi": 0.0,
            "z": 0.001284799139552095,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CD_qbar": {
            "A": 0.0,
            "w": 0.0,
            "phi": 0.0,
            "z": 0.02609587493113088,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CD_Lqbar": {
            "A": 0.38827798378374784,
            "w": 2.0,
            "phi": 1.5707963267948966,
            "z": 0.37002419248695667,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CD_L2qbar": {
            "A": 0.0,
            "w": 0.0,
            "phi": 0.0,
            "z": -0.030344552007313678,
            "multiplier": 1.0,
            "delta": 0.0
        },
```

```
        "CD_rbar": {
            "A": 0.0,
            "w": 0.0,
            "phi": 0.0,
            "z": 0.0,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CD_Srbar": {
            "A": 0.0,
            "w": 0.0,
            "phi": 0.0,
            "z": -0.11463743945164329,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CD_da": {
            "A": -0.007871406214650756,
            "w": 2.0,
            "phi": 0.0,
            "z": 2.129030892933502e-07,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CD_Sda": {
            "A": 0.04923166897259938,
            "w": 2.0,
            "phi": 1.5707963267948966,
            "z": -0.03808239733391901,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CD_de": {
            "A": -0.006108165984475004,
            "w": 1.0,
            "phi": 1.5707963267948966,
            "z": 0.0015277801829500838,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CD_Lde": {
            "A": 0.18303905710766216,
            "w": 1.0,
            "phi": 1.5707963267948966,
            "z": 0.0,
            "multiplier": 1.0,
            "delta": 0.0
        },
        "CD_de2": {
            "A": -0.09503141993378963,
            "w": 1.0,
            "phi": 1.5707963267948966,
            "z": 0.4243978489371473,
            "multiplier": 1.0,
            "delta": 0.0
        }
```

```
        },
        "Cell": {
            "Cl_0": {
                "A": 0.00018771878360712227,
                "w": 2.0,
                "phi": 0.0,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cl_alpha": {
                "A": -0.002255316074959663,
                "w": 4.0,
                "phi": 0.0,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cl_beta": {
                "A": 0.001735871361135101,
                "w": 2.0,
                "phi": 1.5707963267948966,
                "z": -0.018155114331983246,
                "multiplier": 1.0,
                "delta": -0.0101
            },
            "Cl_pbar": {
                "A": 0.003956243663223544,
                "w": 2.0,
                "phi": 1.5707963267948966,
                "z": -0.3069308764732161,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cl_qbar": {
                "A": 0.0,
                "w": 0.0,
                "phi": 0.0,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cl_rbar": {
                "A": 0.0,
                "w": 0.0,
                "phi": 0.0,
                "z": 0.006214770285768683,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cl_Lrbar": {
                "A": 0.0,
                "w": 0.0,
                "phi": 0.0,
                "z": 0.11039503382763775,
                "multiplier": 1.0,
```

```
                "delta": 0.0
            },
            "Cl_da": {
                "A": 0.014044002584287035,
                "w": 2.0,
                "phi": 1.5707963267948966,
                "z": -0.10654507947006382,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cl_de": {
                "A": 0.0017347280117371252,
                "w": 1.0,
                "phi": 0.0,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": 0.0
            }
        },
        "Cm": {
            "Cm_0": {
                "A": 0.016397915202207042,
                "w": 2.0,
                "phi": 1.5707963267948966,
                "z": -0.002242911635817831,
                "multiplier": 1.0,
                "delta": -0.0196
            },
            "Cm_alpha": {
                "A": -0.1381125910227799,
                "w": 2.0,
                "phi": 1.5707963267948966,
                "z": -0.014466146874435483,
                "multiplier": 1.0,
                "delta": 0.2865
            },
            "Cm_beta": {
                "A": 0.8299432389346949,
                "w": 2.0,
                "phi": 0.0,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cm_pbar": {
                "A": -0.01018303333732129,
                "w": 2.0,
                "phi": 0.0,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cm_qbar": {
                "A": -2.355109402945565,
                "w": 2.0,
                "phi": 1.5707963267948966,
```

```
                "z": -2.545733692648518,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cm_rbar": {
                "A": -0.7666562835289177,
                "w": 2.0,
                "phi": 0.0,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cm_da": {
                "A": 0.0007562544986244625,
                "w": 2.0,
                "phi": 0.0,
                "z": -0.0006558804458306277,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cm_de": {
                "A": -0.9114870320894053,
                "w": 1.0,
                "phi": 1.5707963267948966,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": 0.2914
            }
        },
        "Cn": {
            "Cn_0": {
                "A": 0.004822421894065489,
                "w": 2.0,
                "phi": 0.0,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cn_alpha": {
                "A": -0.09293197242941285,
                "w": 2.0,
                "phi": 0.0,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cn_beta": {
                "A": -0.31764903243224546,
                "w": 2.0,
                "phi": 1.5707963267948966,
                "z": 0.31301066324220633,
                "multiplier": 1.0,
                "delta": -0.0326
            },
            "Cn_pbar": {
                "A": 0.0,
```

```
                "w": 0.0,
                "phi": 0.0,
                "z": 0.001035994551322718,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cn_Lpbar": {
                "A": -0.00744283382448568,
                "w": 2.0,
                "phi": 1.5707963267948966,
                "z": -0.1223332777061868,
                "multiplier": 1.0,
                "delta": 0.0602
            },
            "Cn_qbar": {
                "A": -0.9204685260025309,
                "w": 2.0,
                "phi": 0.0,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cn_rbar": {
                "A": 0.2893776169921245,
                "w": 2.0,
                "phi": 1.5707963267948966,
                "z": -0.2788932189123403,
                "multiplier": 1.0,
                "delta": 0.0
            },
            "Cn_da": {
                "A": 0.0,
                "w": 0.0,
                "phi": 0.0,
                "z": 0.000931771956432961,
                "multiplier": 1.0,
                "delta": 0.0122
            },
            "Cn_Lda": {
                "A": -0.016880141841181275,
                "w": 2.0,
                "phi": 1.5707963267948966,
                "z": 0.015692621163331585,
                "multiplier": 1.0,
                "delta": 0.0254
            },
            "Cn_de": {
                "A": -0.35271359252319573,
                "w": 1.0,
                "phi": 0.0,
                "z": 0.0,
                "multiplier": 1.0,
                "delta": 0.0
            }
        }
```

```
}
```

Baseline Aircraft Properties


```
{
    "geometry" : {
        "S_w" : 300.0,
        "b_w" : 30.0,
        "c_w" : 11.32,
        "l_h" : 13.13,
        "Lam_w" : 0.4014,
        "RA_w" : 3.0,
        "Lam_v" : 0.6632,
        "RA_v" : 1.29,
        "Lam_h" : 0.3840,
        "RA_h" : 2.116
        },
    "inertia" : {
        "W" : 21000.0,
        "h_z" : 0.0,
        "h_y" : 0.0,
        "h_x" : 160.0,
        "I_xx" : 9496.0,
        "I_xy" : 0.0,
        "I_xz" : 982.0,
        "I_yy" : 55814.0,
        "I_yz" : 0.0,
        "I_zz" : 63100.0
    }
}
```

BIRE Aircraft Properties

```
{
    "geometry" : {
        "S_w" : 300.0,
        "b_w" : 30.0,
        "c_w" : 11.32,
        "l_h" : 13.13,
        "Lam_w" : 0.4014,
        "RA_w" : 3.0,
        "Lam_v" : 0.6632,
        "RA_v" : 1.29,
        "Lam_h" : 0.3840,
        "RA_h" : 2.116
        },
    "inertia" : {
        "W" : 20500.0,
        "h_z" : 0.0,
        "h_y" : 0.0,
        "h_x" : 160.0,
        "I_xx" : 9496.0,
        "I_xy" : 0.0,
        "I_xz" : 982.0,
        "I_yy" : 55814.0,
        "I_yz" : 0.0,
        "I_zz" : 63100.0
    }
}
```

APPENDIX C
SOURCE CODE

C.1 Analysis of the Aircraft Geometry

Sweep Angle Conversion Routine

```
import numpy as np

def sweep_converter(L_m, m, n, AR, TR):
    t_m = np.tan(L_m)
    C1 = 4./AR
    C2 = (n - m)*(1. - TR)/(1. + TR)
    return np.arctan(t_m - C1*C2)

if __name__ == "__main__":
    L_m = 40.*np.pi/180.  # LE Sweep
    m = 0.
    n = 0.25
    AR = 3.2
    TR = 0.2

    L_n = sweep_converter(L_m, m, n, AR, TR)
```

BIRE Inertial Fits Routine

```
import numpy as np
import scipy.optimize as optimize
import json

def model(coeff, dB, sin=True, freq=2., square=False):
    if sin:
        phi = 0.
    else:
        phi = np.pi/2.
    if not square:
        m = lambda x : x[0]*np.sin(freq*dB + phi) + x[1]
        e = lambda x : m(x) - coeff
        res = optimize.leastsq(e, [300, np.average(coeff)])
        A = res[0][0]
        z = res[0][1]
        return A, freq, phi, z
    else:
        m = lambda x : x[0]*np.abs(np.sin(freq*dB))
        e = lambda x : m(x) - coeff
        A = optimize.leastsq(e, [0])[0][0]
        return A, freq, phi, -A

Ixx = np.array([9280.]*13)
Iyy = np.array([58449., 58427., 58368., 58288.,
                58207., 58149., 58127., 58149.,
                58207., 58288., 58368., 58427., 58449.])
Izz = np.array([65445., 65466., 65525., 65606.,
                65686., 65745., 65766., 65745.,
                65686., 65606., 65525., 65466., 65445.])
Ixy = np.zeros(13)
Ixz = np.array([-5.]*13)
Iyz = np.array([0., -80., -139., -161.,
                -139., -80., 0., -80.,
                -139., -161., -139., -80., 0.])

model_coeff_keys = ["A", "w", "phi", "z"]
model_coeff_dict = {key: 0. for key in model_coeff_keys}

models_dict = {"Ixx": model_coeff_dict,
               "Iyy": model_coeff_dict,
               "Izz": model_coeff_dict,
               "Ixy": model_coeff_dict,
               "Ixz": model_coeff_dict,
               "Iyz": model_coeff_dict}

models_dict["Ixx"] = {key: x for key, x in zip(model_coeff_keys,
                                               [0., 0., 0., Ixx[0]])}
A, freq, phi, z = model(Iyy, dB_rad, sin=False, freq=2.)
models_dict["Iyy"] = {key: x for key, x in zip(model_coeff_keys,
                                               [A, freq, phi, z])}
A, freq, phi, z = model(Izz, dB_rad, sin=False, freq=2.)
models_dict["Izz"] = {key: x for key, x in zip(model_coeff_keys,
                                               [A, freq, phi, z])}
A, freq, phi, z = model(Iyz, dB_rad, freq=2.,square=True)
```

```
models_dict["Iyz"] = {key: x for key, x in zip(model_coeff_keys,
                                               [A, freq, phi, z])}
models_dict["Ixz"] = {key: x for key, x in zip(model_coeff_keys,
                                               [0., 0., 0., Ixz[0]])}

with open("bire_inertia_model.json", "w") as outfile:
    json.dump(models_dict, outfile, indent=4)
```

C.2 Aerodynamic Model Definition

Baseline Aerodynamic Model

```
import numpy as np
import json

class F16Aero:
    def __init__(self, inp_dir='./', **kwargs):
        fn = kwargs.get('fn', 'mux_model_adj.json')
        self.model_coeffs_dict = json.load(open(inp_dir + fn))
        self.CL_coeffs = self.model_coeffs_dict["CL"]
        self.CS_coeffs = self.model_coeffs_dict["CS"]
        self.CD_coeffs = self.model_coeffs_dict["CD"]
        self.Cl_coeffs = self.model_coeffs_dict["Cell"]
        self.Cm_coeffs = self.model_coeffs_dict["Cm"]
        self.Cn_coeffs = self.model_coeffs_dict["Cn"]
        self.CL0 = self.CL_coeffs["CL_0"]
        self.CLa = self.CL_coeffs["CL_alpha"]
        self.CLq = self.CL_coeffs["CL_qbar"]
        self.CLde = self.CL_coeffs["CL_de"]
        self.CSb = self.CS_coeffs["CS_beta"]
        self.CSp = self.CS_coeffs["CS_pbar"]
        self.CSLp = self.CS_coeffs["CS_Lpbar"]
        self.CSr = self.CS_coeffs["CS_rbar"]
        self.CSda = self.CS_coeffs["CS_da"]
        self.CSdr = self.CS_coeffs["CS_dr"]
        self.CD0 = self.CD_coeffs["CD_0"]
        self.CDL = self.CD_coeffs["CD_L"]
        self.CDL2 = self.CD_coeffs["CD_L2"]
        self.CDS2 = self.CD_coeffs["CD_S2"]
        self.CDSp = self.CD_coeffs["CD_Spbar"]
        self.CDq = self.CD_coeffs["CD_qbar"]
        self.CDLq = self.CD_coeffs["CD_Lqbar"]
        self.CDL2q = self.CD_coeffs["CD_L2qbar"]
        self.CDSr = self.CD_coeffs["CD_Srbar"]
        self.CDde = self.CD_coeffs["CD_de"]
        self.CDLde = self.CD_coeffs["CD_Lde"]
        self.CDde2 = self.CD_coeffs["CD_de2"]
        self.CDSda = self.CD_coeffs["CD_Sda"]
        self.CDSdr = self.CD_coeffs["CD_Sdr"]
        self.Clb = self.Cl_coeffs["Cl_beta"]
        self.Clp = self.Cl_coeffs["Cl_pbar"]
        self.Clr = self.Cl_coeffs["Cl_rbar"]
        self.ClLr = self.Cl_coeffs["Cl_Lrbar"]
        self.Clda = self.Cl_coeffs["Cl_da"]
        self.Cldr = self.Cl_coeffs["Cl_dr"]
        self.Cm0 = self.Cm_coeffs["Cm_0"]
        self.Cma = self.Cm_coeffs["Cm_alpha"]
        self.Cmq = self.Cm_coeffs["Cm_qbar"]
        self.Cmde = self.Cm_coeffs["Cm_de"]
        self.Cnb = self.Cn_coeffs["Cn_beta"]
        self.Cnp = self.Cn_coeffs["Cn_pbar"]
        self.CnLp = self.Cn_coeffs["Cn_Lpbar"]
```

```
            self.Cnr = self.Cn_coeffs["Cn_rbar"]
            self.Cnda = self.Cn_coeffs["Cn_da"]
            self.CnLda = self.Cn_coeffs["Cn_Lda"]
            self.Cndr = self.Cn_coeffs["Cn_dr"]

        def _reevaluate_coeffs(self):
            self.CL0 = self.CL_coeffs["CL_0"]
            self.CLa = self.CL_coeffs["CL_alpha"]
            self.CLq = self.CL_coeffs["CL_qbar"]
            self.CLde = self.CL_coeffs["CL_de"]
            self.CSb = self.CS_coeffs["CS_beta"]
            self.CSp = self.CS_coeffs["CS_pbar"]
            self.CSLp = self.CS_coeffs["CS_Lpbar"]
            self.CSr = self.CS_coeffs["CS_rbar"]
            self.CSda = self.CS_coeffs["CS_da"]
            self.CSdr = self.CS_coeffs["CS_dr"]
            self.CD0 = self.CD_coeffs["CD_0"]
            self.CDL = self.CD_coeffs["CD_L"]
            self.CDL2 = self.CD_coeffs["CD_L2"]
            self.CDS2 = self.CD_coeffs["CD_S2"]
            self.CDSp = self.CD_coeffs["CD_Spbar"]
            self.CDq = self.CD_coeffs["CD_qbar"]
            self.CDLq = self.CD_coeffs["CD_Lqbar"]
            self.CDL2q = self.CD_coeffs["CD_L2qbar"]
            self.CDSr = self.CD_coeffs["CD_Srbar"]
            self.CDde = self.CD_coeffs["CD_de"]
            self.CDLde = self.CD_coeffs["CD_Lde"]
            self.CDde2 = self.CD_coeffs["CD_de2"]
            self.CDSda = self.CD_coeffs["CD_Sda"]
            self.CDSdr = self.CD_coeffs["CD_Sdr"]
            self.Clb = self.Cl_coeffs["Cl_beta"]
            self.Clp = self.Cl_coeffs["Cl_pbar"]
            self.Clr = self.Cl_coeffs["Cl_rbar"]
            self.ClLr = self.Cl_coeffs["Cl_Lrbar"]
            self.Clda = self.Cl_coeffs["Cl_da"]
            self.Cldr = self.Cl_coeffs["Cl_dr"]
            self.Cm0 = self.Cm_coeffs["Cm_0"]
            self.Cma = self.Cm_coeffs["Cm_alpha"]
            self.Cmq = self.Cm_coeffs["Cm_qbar"]
            self.Cmde = self.Cm_coeffs["Cm_de"]
            self.Cnb = self.Cn_coeffs["Cn_beta"]
            self.Cnp = self.Cn_coeffs["Cn_pbar"]
            self.CnLp = self.Cn_coeffs["Cn_Lpbar"]
            self.Cnr = self.Cn_coeffs["Cn_rbar"]
            self.Cnda = self.Cn_coeffs["Cn_da"]
            self.CnLda = self.Cn_coeffs["Cn_Lda"]
            self.Cndr = self.Cn_coeffs["Cn_dr"]

        def _CL(self, alpha, beta, pbar, qbar, rbar, da, de, dr):
            CL = (self.CL0 + self.CLa*alpha + self.CLq*qbar + self.CLde*de)
            return CL

        def _CS(self, alpha, beta, pbar, qbar, rbar, da, de, dr):
            CL1 = self._CL(alpha, 0., 0., 0., 0., 0., 0., 0.)
            CS = (self.CSb*beta + (self.CSp + self.CSLp*CL1)*pbar +
                  self.CSr*rbar + self.CSda*da + self.CSdr*dr)
```

```
        return CS

    def _CD(self, alpha, beta, pbar, qbar, rbar, da, de, dr):
        CL1 = self._CL(alpha, 0., 0., 0., 0., 0., 0., 0.)
        CS1 = self._CS(0., beta, 0., 0., 0., 0., 0., 0.)
        CD = (self.CD0 + self.CDL*CL1 + self.CDL2*CL1**2 + self.CDS2*CS1**2 +
              (self.CDSp*CS1)*pbar +
              (self.CDq + self.CDLq*CL1 + self.CDL2q*CL1**2)*qbar +
              (self.CDSr*CS1)*rbar +
              (self.CDde + self.CDLde*CL1)*de + self.CDde2*de**2 +
              (self.CDSda*CS1)*da +
              (self.CDSdr*CS1)*dr)
        return CD

    def _Cl(self, alpha, beta, pbar, qbar, rbar, da, de, dr):
        CL1 = self._CL(alpha, 0., 0., 0., 0., 0., 0., 0.)
        Cl = (self.Clb*beta + self.Clp*pbar + (self.Clr + self.ClLr*CL1)*rbar +
              self.Clda*da + self.Cldr*dr)
        return Cl

    def _Cm(self, alpha, beta, pbar, qbar, rbar, da, de, dr):
        Cm = (self.Cm0 + self.Cma*alpha + self.Cmq*qbar + self.Cmde*de)
        return Cm

    def _Cn(self, alpha, beta, pbar, qbar, rbar, da, de, dr):
        CL1 = self._CL(alpha, 0., 0., 0., 0., 0., 0., 0.)
        Cn = (self.Cnb*beta + (self.Cnp + self.CnLp*CL1)*pbar + self.Cnr*rbar +
              (self.Cnda + self.CnLda*CL1)*da + self.Cndr*dr)
        return Cn

    def aero_results(self, alpha, beta, pbar, qbar, rbar, da, de, dr):
        params = alpha, beta, pbar, qbar, rbar, da, de, dr
        return [self._CL(*params), self._CS(*params), self._CD(*params),
                self._Cl(*params), self._Cm(*params), self._Cn(*params)]

if __name__ == "__main__":
    case = F16Aero()
    params = np.deg2rad([10., 10., 10., 10., 10., 10., 10., 10.])
    [CL, CS, CD, Cl, Cm, Cn] = case.aero_results(*params)
```

BIRE Aerodynamic Model

```
import numpy as np
import json
import scipy.optimize as optimize

class BIREAero:
    def __init__(self, inp_dir='./'):
        self.model_coeffs_dict = json.load(open(inp_dir + 'bire_model_adj.json'))
        self.CL_coeffs = self.model_coeffs_dict["CL"]
        self.CS_coeffs = self.model_coeffs_dict["CS"]
        self.CD_coeffs = self.model_coeffs_dict["CD"]
        self.Cl_coeffs = self.model_coeffs_dict["Cell"]
        self.Cm_coeffs = self.model_coeffs_dict["Cm"]
        self.Cn_coeffs = self.model_coeffs_dict["Cn"]
        self.deriv=False

    def evaluate_coeffs(self, d_B):
        self.CL0 = self._CL0(d_B)
        self.CLa = self._CL_alpha(d_B)
        self.CLb = self._CL_beta(d_B)
        self.CLp = self._CL_pbar(d_B)
        self.CLq = self._CL_qbar(d_B)
        self.CLr = self._CL_rbar(d_B)
        self.CLda = self._CL_da(d_B)
        self.CLde = self._CL_de(d_B)

        self.CS0 = self._CS0(d_B)
        self.CSa = self._CS_alpha(d_B)
        self.CSb = self._CS_beta(d_B)
        self.CSp = self._CS_pbar(d_B)
        self.CSLp = self._CS_Lpbar(d_B)
        self.CSq = self._CS_qbar(d_B)
        self.CSr = self._CS_rbar(d_B)
        self.CSda = self._CS_da(d_B)
        self.CSde = self._CS_de(d_B)

        self.CD0 = self._CD0(d_B)
        self.CDL = self._CD_L(d_B)
        self.CDL2 = self._CD_L2(d_B)
        self.CDS = self._CD_S(d_B)
        self.CDS2 = self._CD_S2(d_B)
        self.CDp = self._CD_pbar(d_B)
        self.CDSp = self._CD_Spbar(d_B)
        self.CDq = self._CD_qbar(d_B)
        self.CDLq = self._CD_Lqbar(d_B)
        self.CDL2q = self._CD_L2qbar(d_B)
        self.CDr = self._CD_rbar(d_B)
        self.CDSr = self._CD_Srbar(d_B)
        self.CDda = self._CD_da(d_B)
        self.CDSda = self._CD_Sda(d_B)
        self.CDde = self._CD_de(d_B)
        self.CDLde = self._CD_Lde(d_B)
        self.CDde2 = self._CD_de2(d_B)

        self.Cl0 = self._Cl0(d_B)
```

```
        self.Cla = self._Cl_alpha(d_B)
        self.Clb = self._Cl_beta(d_B)
        self.Clp = self._Cl_pbar(d_B)
        self.Clq = self._Cl_qbar(d_B)
        self.Clr = self._Cl_rbar(d_B)
        self.ClLr = self._Cl_Lrbar(d_B)
        self.Clda = self._Cl_da(d_B)
        self.Clde = self._Cl_de(d_B)

        self.Cm0 = self._Cm0(d_B)
        self.Cma = self._Cm_alpha(d_B)
        self.Cmb = self._Cm_beta(d_B)
        self.Cmp = self._Cm_pbar(d_B)
        self.Cmq = self._Cm_qbar(d_B)
        self.Cmr = self._Cm_rbar(d_B)
        self.Cmda = self._Cm_da(d_B)
        self.Cmde = self._Cm_de(d_B)

        self.Cn0 = self._Cn0(d_B)
        self.Cna = self._Cn_alpha(d_B)
        self.Cnb = self._Cn_beta(d_B)
        self.Cnp = self._Cn_pbar(d_B)
        self.CnLp = self._Cn_Lpbar(d_B)
        self.Cnq = self._Cn_qbar(d_B)
        self.Cnr = self._Cn_rbar(d_B)
        self.Cnda = self._Cn_da(d_B)
        self.CnLda = self._Cn_Lda(d_B)
        self.Cnde = self._Cn_de(d_B)

    def evaluate_derivatives(self, d_B):
        self.deriv = True
        self.dCL0 = self._CL0(d_B)
        self.dCLa = self._CL_alpha(d_B)
        self.dCLb = self._CL_beta(d_B)
        self.dCLp = self._CL_pbar(d_B)
        self.dCLq = self._CL_qbar(d_B)
        self.dCLr = self._CL_rbar(d_B)
        self.dCLda = self._CL_da(d_B)
        self.dCLde = self._CL_de(d_B)

        self.dCS0 = self._CS0(d_B)
        self.dCSa = self._CS_alpha(d_B)
        self.dCSb = self._CS_beta(d_B)
        self.dCSp = self._CS_pbar(d_B)
        self.dCSLp = self._CS_Lpbar(d_B)
        self.dCSq = self._CS_qbar(d_B)
        self.dCSr = self._CS_rbar(d_B)
        self.dCSda = self._CS_da(d_B)
        self.dCSde = self._CS_de(d_B)

        self.dCD0 = self._CD0(d_B)
        self.dCDL = self._CD_L(d_B)
        self.dCDL2 = self._CD_L2(d_B)
        self.dCDS = self._CD_S(d_B)
        self.dCDS2 = self._CD_S2(d_B)
        self.dCDp = self._CD_pbar(d_B)
```

```
            self.dCDSp = self._CD_Spbar(d_B)
            self.dCDq = self._CD_qbar(d_B)
            self.dCDLq = self._CD_Lqbar(d_B)
            self.dCDL2q = self._CD_L2qbar(d_B)
            self.dCDr = self._CD_rbar(d_B)
            self.dCDSr = self._CD_Srbar(d_B)
            self.dCDda = self._CD_da(d_B)
            self.dCDSda = self._CD_Sda(d_B)
            self.dCDde = self._CD_de(d_B)
            self.dCDLde = self._CD_Lde(d_B)
            self.dCDde2 = self._CD_de2(d_B)

            self.dCl0 = self._Cl0(d_B)
            self.dCla = self._Cl_alpha(d_B)
            self.dClb = self._Cl_beta(d_B)
            self.dClp = self._Cl_pbar(d_B)
            self.dClq = self._Cl_qbar(d_B)
            self.dClr = self._Cl_rbar(d_B)
            self.dClLr = self._Cl_Lrbar(d_B)
            self.dClda = self._Cl_da(d_B)
            self.dClde = self._Cl_de(d_B)

            self.dCm0 = self._Cm0(d_B)
            self.dCma = self._Cm_alpha(d_B)
            self.dCmb = self._Cm_beta(d_B)
            self.dCmp = self._Cm_pbar(d_B)
            self.dCmq = self._Cm_qbar(d_B)
            self.dCmr = self._Cm_rbar(d_B)
            self.dCmda = self._Cm_da(d_B)
            self.dCmde = self._Cm_de(d_B)

            self.dCn0 = self._Cn0(d_B)
            self.dCna = self._Cn_alpha(d_B)
            self.dCnb = self._Cn_beta(d_B)
            self.dCnp = self._Cn_pbar(d_B)
            self.dCnLp = self._Cn_Lpbar(d_B)
            self.dCnq = self._Cn_qbar(d_B)
            self.dCnr = self._Cn_rbar(d_B)
            self.dCnda = self._Cn_da(d_B)
            self.dCnLda = self._Cn_Lda(d_B)
            self.dCnde = self._Cn_de(d_B)
            self.deriv = False


    def _CL0(self, d_B):
        Cdict = self.CL_coeffs["CL_0"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CL_alpha(self, d_B):
        Cdict = self.CL_coeffs["CL_alpha"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
```

```
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CL_beta(self, d_B):
        Cdict = self.CL_coeffs["CL_beta"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CL_pbar(self, d_B):
        Cdict = self.CL_coeffs["CL_pbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CL_qbar(self, d_B, deriv=False):
        Cdict = self.CL_coeffs["CL_qbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        z += delta
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CL_rbar(self, d_B):
        Cdict = self.CL_coeffs["CL_rbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CL_da(self, d_B):
        Cdict = self.CL_coeffs["CL_da"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CL_de(self, d_B):
        Cdict = self.CL_coeffs["CL_de"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CS0(self, d_B):
        Cdict = self.CS_coeffs["CS_0"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
```

```
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CS_alpha(self, d_B):
        Cdict = self.CS_coeffs["CS_alpha"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CS_beta(self, d_B):
        Cdict = self.CS_coeffs["CS_beta"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        z += delta
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CS_pbar(self, d_B):
        Cdict = self.CS_coeffs["CS_pbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CS_Lpbar(self, d_B):
        Cdict = self.CS_coeffs["CS_Lpbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CS_qbar(self, d_B):
        Cdict = self.CS_coeffs["CS_qbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CS_rbar(self, d_B):
        Cdict = self.CS_coeffs["CS_rbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        z += delta
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CS_de(self, d_B):
```

```
        Cdict = self.CS_coeffs["CS_de"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CS_da(self, d_B):
        Cdict = self.CS_coeffs["CS_da"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CD0(self, d_B):
        Cdict = self.CD_coeffs["CD_0"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        A = A*sigma
        z = z*sigma
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CD_L(self, d_B):
        Cdict = self.CD_coeffs["CD_L"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        z += delta
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CD_L2(self, d_B):
        Cdict = self.CD_coeffs["CD_L2"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        z += delta
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CD_S(self, d_B):
        Cdict = self.CD_coeffs["CD_S"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CD_S2(self, d_B):
        Cdict = self.CD_coeffs["CD_S2"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
```

```
            else:
                return A*w*np.cos(w*d_B + phi)

    def _CD_pbar(self, d_B):
        Cdict = self.CD_coeffs["CD_pbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CD_Spbar(self, d_B):
        Cdict = self.CD_coeffs["CD_Spbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CD_qbar(self, d_B):
        Cdict = self.CD_coeffs["CD_qbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        z += delta
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CD_Lqbar(self, d_B):
        Cdict = self.CD_coeffs["CD_Lqbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        z += delta
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CD_L2qbar(self, d_B):
        Cdict = self.CD_coeffs["CD_L2qbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        z += delta
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CD_rbar(self, d_B):
        Cdict = self.CD_coeffs["CD_rbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _CD_Srbar(self, d_B):
        Cdict = self.CD_coeffs["CD_Srbar"]
```

```
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            z += delta
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

        def _CD_da(self, d_B):
            Cdict = self.CD_coeffs["CD_da"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

        def _CD_Sda(self, d_B):
            Cdict = self.CD_coeffs["CD_Sda"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

        def _CD_de(self, d_B):
            Cdict = self.CD_coeffs["CD_de"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            z += delta
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

        def _CD_Lde(self, d_B):
            Cdict = self.CD_coeffs["CD_Lde"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

        def _CD_de2(self, d_B):
            Cdict = self.CD_coeffs["CD_de2"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

        def _Cl0(self, d_B):
            Cdict = self.Cl_coeffs["Cl_0"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

```

```
    def _Cl_alpha(self, d_B):
        Cdict = self.Cl_coeffs["Cl_alpha"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _Cl_beta(self, d_B):
        Cdict = self.Cl_coeffs["Cl_beta"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        z += delta
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _Cl_pbar(self, d_B):
        Cdict = self.Cl_coeffs["Cl_pbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _Cl_qbar(self, d_B):
        Cdict = self.Cl_coeffs["Cl_qbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _Cl_rbar(self, d_B):
        Cdict = self.Cl_coeffs["Cl_rbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _Cl_Lrbar(self, d_B):
        Cdict = self.Cl_coeffs["Cl_Lrbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _Cl_da(self, d_B):
        Cdict = self.Cl_coeffs["Cl_da"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        z += delta
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
```

```
                return A*w*np.cos(w*d_B + phi)

    def _Cl_de(self, d_B):
        Cdict = self.Cl_coeffs["Cl_de"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _Cm0(self, d_B):
        Cdict = self.Cm_coeffs["Cm_0"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        z += delta
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _dCm0_dB(self, d_B):
        Cdict = self.Cm_coeffs["Cm_0"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        return A*w*np.cos(w*d_B + phi)

    def _Cm_alpha(self, d_B):
        Cdict = self.Cm_coeffs["Cm_alpha"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _dCma_dB(self, d_B):
        Cdict = self.Cm_coeffs["Cm_alpha"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        return A*w*np.cos(w*d_B + phi)

    def _Cm_beta(self, d_B):
        Cdict = self.Cm_coeffs["Cm_beta"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _dCmb_dB(self, d_B):
        Cdict = self.Cm_coeffs["Cm_beta"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        return A*w*np.cos(w*d_B + phi)

    def _Cm_pbar(self, d_B):
        Cdict = self.Cm_coeffs["Cm_pbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
```

```
                return A*w*np.cos(w*d_B + phi)

        def _dCmp_dB(self, d_B):
            Cdict = self.Cm_coeffs["Cm_pbar"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            return A*w*np.cos(w*d_B + phi)

        def _Cm_qbar(self, d_B):
            Cdict = self.Cm_coeffs["Cm_qbar"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            z += delta
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

        def _dCmq_dB(self, d_B):
            Cdict = self.Cm_coeffs["Cm_qbar"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            return A*w*np.cos(w*d_B + phi)

        def _Cm_rbar(self, d_B):
            Cdict = self.Cm_coeffs["Cm_rbar"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

        def _dCmr_dB(self, d_B):
            Cdict = self.Cm_coeffs["Cm_rbar"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            return A*w*np.cos(w*d_B + phi)

        def _Cm_da(self, d_B):
            Cdict = self.Cm_coeffs["Cm_da"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

        def _dCmda_dB(self, d_B):
            Cdict = self.Cm_coeffs["Cm_da"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            return A*w*np.cos(w*d_B + phi)

        def _Cm_de(self, d_B):
            Cdict = self.Cm_coeffs["Cm_de"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

        def _dCmde_dB(self, d_B):
```

```
            Cdict = self.Cm_coeffs["Cm_de"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            return A*w*np.cos(w*d_B + phi)

        def _Cn0(self, d_B):
            Cdict = self.Cn_coeffs["Cn_0"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

        def _dCn0_dB(self, d_B):
            Cdict = self.Cn_coeffs["Cn_0"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            return A*w*np.cos(w*d_B + phi)

        def _Cn_alpha(self, d_B):
            Cdict = self.Cn_coeffs["Cn_alpha"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

        def _dCna_dB(self, d_B):
            Cdict = self.Cn_coeffs["Cn_alpha"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            return A*w*np.cos(w*d_B + phi)

        def _Cn_beta(self, d_B):
            Cdict = self.Cn_coeffs["Cn_beta"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

        def _dCnb_dB(self, d_B):
            Cdict = self.Cn_coeffs["Cn_beta"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            return A*w*np.cos(w*d_B + phi)

        def _Cn_pbar(self, d_B):
            Cdict = self.Cn_coeffs["Cn_pbar"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            if not self.deriv:
                return A*np.sin(w*d_B + phi) + z
            else:
                return A*w*np.cos(w*d_B + phi)

        def _dCnp_dB(self, d_B):
            Cdict = self.Cn_coeffs["Cn_pbar"]
            [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
            return A*w*np.cos(w*d_B + phi)
```

```
    def _Cn_Lpbar(self, d_B):
        Cdict = self.Cn_coeffs["Cn_Lpbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _dCnLp_dB(self, d_B):
        CnLdict = self.Cn_coeffs["Cn_Lpbar"]
        [A_nL, w_nL, phi_nL, z_nL] = [CnLdict[c] for c in CnLdict]
        CL0dict = self.CL_coeffs["CL_0"]
        [A_0, w_0, phi_0, z_0] = [CL0dict[c] for c in CL0dict]
        CLadict = self.CL_coeffs["CL_alpha"]
        [A_a, w_a, phi_a, z_a] = [CLadict[c] for c in CLadict]
        C1 = A_nL*w_nL*np.cos(w_nL*d_B + phi_nL)*(A_0*np.sin(w_0*d_B + phi_0) + z_0)
        C2 = A_0*w_0*np.cos(w_0*d_B + phi_0)*(A_nL*np.sin(w_nL*d_B + phi_nL) + z_nL)
        C3 = A_nL*w_nL*np.cos(w_nL*d_B + phi_nL)*(A_a*np.sin(w_a*d_B + phi_a) + z_a)
        C4 = A_a*w_a*np.cos(w_a*d_B + phi_a)*(A_nL*np.sin(w_nL*d_B + phi_nL) + z_nL)
        return [C1, C2, C3, C4]

    def _Cn_qbar(self, d_B):
        Cdict = self.Cn_coeffs["Cn_qbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _dCnq_dB(self, d_B):
        Cdict = self.Cn_coeffs["Cn_qbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        return A*w*np.cos(w*d_B + phi)

    def _Cn_rbar(self, d_B):
        Cdict = self.Cn_coeffs["Cn_rbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        z += delta
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _dCnr_dB(self, d_B):
        Cdict = self.Cn_coeffs["Cn_rbar"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        return A*w*np.cos(w*d_B + phi)

    def _Cn_da(self, d_B):
        Cdict = self.Cn_coeffs["Cn_da"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)
```

```
    def _dCnda_dB(self, d_B):
        Cndadict = self.Cn_coeffs["Cn_da"]
        [A_da, w_da, phi_da, z_da] = [Cndadict[c] for c in Cndadict]
        CL0dict = self.CL_coeffs["CL_0"]
        [A_0, w_0, phi_0, z_0] = [CL0dict[c] for c in CL0dict]
        CLadict = self.CL_coeffs["CL_alpha"]
        [A_a, w_a, phi_a, z_a] = [CLadict[c] for c in CLadict]
        C1 = A_da*w_da*np.cos(w_da*d_B + phi_da)*(A_0*np.sin(w_0*d_B + phi_0) + z_0)
        C2 = A_0*w_0*np.cos(w_0*d_B + phi_0)*(A_da*np.sin(w_da*d_B + phi_da) + z_da)
        C3 = A_da*w_da*np.cos(w_da*d_B + phi_da)*(A_a*np.sin(w_a*d_B + phi_a) + z_a)
        C4 = A_a*w_a*np.cos(w_a*d_B + phi_a)*(A_da*np.sin(w_da*d_B + phi_da) + z_da)
        return [C1, C2, C3, C4]

    def _Cn_Lda(self, d_B):
        Cdict = self.Cn_coeffs["Cn_Lda"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _Cn_de(self, d_B):
        Cdict = self.Cn_coeffs["Cn_de"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        if not self.deriv:
            return A*np.sin(w*d_B + phi) + z
        else:
            return A*w*np.cos(w*d_B + phi)

    def _dCnde_dB(self, d_B):
        Cdict = self.Cn_coeffs["Cn_de"]
        [A, w, phi, z, sigma, delta] = [Cdict[c] for c in Cdict]
        return A*w*np.cos(w*d_B + phi)

    def _CL(self, alpha, beta, pbar, qbar, rbar, da, de, dB):
        CL1 = self._CL0(dB) + self._CL_alpha(dB)*alpha
        CL = (CL1 + self._CL_beta(dB)*beta + self._CL_pbar(dB)*pbar +
              self._CL_qbar(dB)*qbar + self._CL_rbar(dB)*rbar +
              self._CL_da(dB)*da + self._CL_de(dB)*de)
        return CL

    def _CS(self, alpha, beta, pbar, qbar, rbar, da, de, dB):
        CL1 = self._CL0(dB) + self._CL_alpha(dB)*alpha
        CS = (self._CS0(dB) + self._CS_alpha(dB)*alpha +
              self._CS_beta(dB)*beta +
              (self._CS_pbar(dB) + self._CS_Lpbar(dB)*CL1)*pbar +
              self._CS_qbar(dB)*qbar + self._CS_rbar(dB)*rbar +
              self._CS_da(dB)*da + self._CS_de(dB)*de)
        return CS

    def _CD(self, alpha, beta, pbar, qbar, rbar, da, de, dB):
        CL1 = self._CL0(dB) + self._CL_alpha(dB)*alpha
        CS1 = self._CS0(dB) + self._CS_beta(dB)*beta
        CD = (self._CD0(dB) + self._CD_L(dB)*CL1 + self._CD_L2(dB)*CL1**2 +
              self._CD_S(dB)*CS1 + self._CD_S2(dB)*CS1**2 +
              (self._CD_pbar(dB) + self._CD_Spbar(dB)*CS1)*pbar +
```

```
                (self._CD_qbar(dB) + self._CD_Lqbar(dB)*CL1 +
                 self._CD_L2qbar(dB)*CL1**2)*qbar +
                (self._CD_rbar(dB) + self._CD_Srbar(dB)*CS1)*rbar +
                (self._CD_da(dB) + self._CD_Sda(dB)*CS1)*da +
                (self._CD_de(dB) + self._CD_Lde(dB)*CL1)*de +
                self._CD_de2(dB)*de**2)
        return CD

    def _Cl(self, alpha, beta, pbar, qbar, rbar, da, de, dB):
        CL1 = self._CL0(dB) + self._CL_alpha(dB)*alpha
        Cl = (self._Cl0(dB) + self._Cl_alpha(dB)*alpha +
              self._Cl_beta(dB)*beta + self._Cl_pbar(dB)*pbar +
              self._Cl_qbar(dB)*qbar +
              (self._Cl_rbar(dB) + self._Cl_Lrbar(dB)*CL1)*rbar +
              self._Cl_da(dB)*da + self._Cl_de(dB)*de)
        return Cl

    def _Cm(self, alpha, beta, pbar, qbar, rbar, da, de, dB):
        Cm = (self._Cm0(dB) + self._Cm_alpha(dB)*alpha +
              self._Cm_beta(dB)*beta + self._Cm_pbar(dB)*pbar +
              self._Cm_qbar(dB)*qbar + self._Cm_rbar(dB)*rbar +
              self._Cm_da(dB)*da + self._Cm_de(dB)*de)
        return Cm

    def _dCm_dB(self, alpha, beta, pbar, qbar, rbar, da, de, dB):
        dCmdB = (self._dCm0_dB(dB) + self._dCma_dB(dB)*alpha +
                 self._dCmb_dB(dB)*beta + self._dCmp_dB(dB)*pbar +
                 self._dCmq_dB(dB)*qbar + self._dCmr_dB(dB)*rbar +
                 self._dCmda_dB(dB)*da + self._dCmde_dB(dB)*de)
        return dCmdB

    def _Cn(self, alpha, beta, pbar, qbar, rbar, da, de, dB):
        CL1 = self._CL0(dB) + self._CL_alpha(dB)*alpha
        Cn = (self._Cn0(dB) + self._Cn_alpha(dB)*alpha +
              self._Cn_beta(dB)*beta +
              (self._Cn_pbar(dB) + self._Cn_Lpbar(dB)*CL1)*pbar +
              self._Cn_qbar(dB)*qbar + self._Cn_rbar(dB)*rbar +
              (self._Cn_da(dB) + self._Cn_Lda(dB)*CL1)*da +
              self._Cn_de(dB)*de)
        return Cn

    def _dCn_dB(self, alpha, beta, pbar, qbar, rbar, da, de, dB):
        dCnLp_dB = self._dCnLp_dB(dB)
        dCnda_dB = self._dCnda_dB(dB)
        dCndB = (self._dCn0_dB(dB) + self._dCna_dB(dB)*alpha +
                 self._dCnb_dB(dB)*beta +
                 (dCnLp_dB[0] + dCnLp_dB[1] +
                  alpha*(dCnLp_dB[2] + dCnLp_dB[3]))*pbar +
                 self._dCnq_dB(dB)*qbar + self._dCnr_dB(dB)*rbar +
                 (dCnda_dB[0] + dCnda_dB[1] +
                  alpha*(dCnda_dB[2] + dCnda_dB[3]))*da +
                 self._dCnde_dB(dB)*de)
        return dCndB

    def aero_results(self, alpha, beta, pbar, qbar, rbar, da, de, dB):
        params = alpha, beta, pbar, qbar, rbar, da, de, dB
```

```
        return [self._CL(*params), self._CS(*params), self._CD(*params),
                self._Cl(*params), self._Cm(*params), self._Cn(*params)]

    def Cn_dB(self, alpha, beta, pbar, qbar, rbar, da, de, dB, method='fd', h=0.001):
        if method=='fd':
            params_p = alpha, beta, pbar, qbar, rbar, da, de, dB + h
            params_m = alpha, beta, pbar, qbar, rbar, da, de, dB - h
            Cn_plus = self._Cn(*params_p)
            Cn_minus = self._Cn(*params_m)
            Cn_dB = (Cn_plus - Cn_minus)/(2.*h)
            return Cn_dB
        elif method=='complex-step':
            h = 1e-16
            params_complex = alpha, beta, pbar, qbar, rbar, da, de, complex(dB, h)
            Cn_complex = self._Cn(*params_complex)
            Cn_dB = Cn_complex.imag/h
            return Cn_dB
        elif method=='fit':
            A = -0.0199462
            w = 2.
            phi = np.pi/2.
            z = 0.
            return A*np.sin(w*dB + phi) + z
        elif method=='analytic':
            Cdict = self.Cn_coeffs["Cn_0"]
            [A0, w0, phi0, z0] = [Cdict[c] for c in Cdict]
            Cdict = self.Cn_coeffs["Cn_alpha"]
            [Aa, wa, phia, za] = [Cdict[c] for c in Cdict]
            Cdict = self.Cn_coeffs["Cn_beta"]
            [Ab, wb, phib, zb] = [Cdict[c] for c in Cdict]
            Cdict = self.Cn_coeffs["Cn_qbar"]
            [Aq, wq, phiq, zq] = [Cdict[c] for c in Cdict]
            Cdict = self.Cn_coeffs["Cn_rbar"]
            [Ar, wr, phir, zr] = [Cdict[c] for c in Cdict]
            Cdict = self.Cn_coeffs["Cn_de"]
            [Ade, wde, phide, zde] = [Cdict[c] for c in Cdict]
            dCn0 = A0*np.cos(w0*dB + phi0)
            dCna = Aa*np.cos(wa*dB + phia)
            dCnb = Ab*np.cos(wb*dB + phib)
            dCnq = Aq*np.cos(wq*dB + phiq)
            dCnr = Ar*np.cos(wr*dB + phir)
            dCnde = Ade*np.cos(wde*dB + phide)
            Cn_dB = dCn0 + dCna*alpha + dCnb*beta + dCnq*qbar + dCnr*rbar + dCnde*de
            return Cn_dB

    def control_matrix(self,  alpha, beta, pbar, qbar, rbar, da, de, dB):
        A = np.zeros((2, 2))
        A[0, 0] = self._dCm_dB(alpha, beta, pbar, qbar, rbar, da, de, dB)
        A[0, 1] = self._Cm_de(dB)
        A[1, 0] = self._dCn_dB(alpha, beta, pbar, qbar, rbar, da, de, dB)
        A[1, 1] = self._Cn_de(dB)
        return A


if __name__ == "__main__":
    case = BIREAero()
```

```
    params = np.deg2rad([10., 10., 10., 10., 10., 10., 10., 10.])
    [CL, CS, CD, Cl, Cm, Cn] = case.aero_results(*params)
```

C.3 Aerodynamic Model Coefficient Evaluation

Standard Atmosphere Calculator

```
import numpy as np

def stdatm_si(h):
    Psa = np.zeros(9)
    zsa = [0., 11000., 20000., 32000., 47000., 52000., 61000., 79000., 9.9e20]
    Tsa = [288.15, 216.65, 216.65, 228.65, 270.65, 270.65, 252.65, 180.65,
           180.65]
    g0 = 9.80665
    R = 287.0528
    Re = 6356766.
    Psa[0] = 101325.
    z = Re*h/(Re+h)
    for i in range(1, 9):
        Lt = -(Tsa[i] - Tsa[i-1])/(zsa[i] - zsa[i-1])
        if Lt == 0:
            if z <= zsa[i]:
                t = Tsa[i-1]
                p = Psa[i-1]*np.exp(-g0*(z-zsa[i-1])/R/Tsa[i-1])
                d = p/R/t
                return z, t, p, d
            else:
                Psa[i] = Psa[i-1]*np.exp(-g0*(zsa[i] - zsa[i-1])/R/Tsa[i-1])
        else:
            ex = g0/R/Lt
            if z < zsa[i]:
                t = Tsa[i-1] - Lt*(z-zsa[i-1])
                p = Psa[i-1]*(t/Tsa[i-1])**ex
                d = p/R/t
                return z, t, p, d
            else:
                Psa[i] = Psa[i-1]*(Tsa[i]/Tsa[i-1])**ex
    t = Tsa[8]
    a = np.sqrt(1.4*287.0528*t)
    return z, t, p, d, a


def stdatm_english(h):
    hsi = h*0.3048
    zsi, tsi, psi, dsi, asi = statsi(hsi)
    z = zsi/0.3048
    t = tsi*1.8
    p = psi*0.02088543
    d = dsi*0.001940320
    a = asi/0.3048
    return z, t, p, d, a
```

NASA Wind Tunnel Model [64]

Wind tunnel data can be obtained using the data extraction tools in the GitHub repository published on the USU Aerolab GitHub page.

```
import numpy as np
from scipy.interpolate import RegularGridInterpolator as rgi
from scipy.interpolate import interp1d
import stdatmos as atmos

class F16_windtunnel:
    def __init__(self, data_dir="./NASA Data/Python Data/"):
        self.c_ref = 11.32
        self.b_w = 30.
        self.S_w = 300.
        self.xcgref_cref = 0.35
        alpha = np.concatenate((np.arange(-20., 60., 5.),
                                 np.arange(60., 100., 10.)))
        alpha_lef = np.arange(-20., 50., 5.)
        beta = np.array([-30., -25., -20., -15., -10., -8., -6., -4., -2., 0.,
                         2., 4., 6., 8., 10., 15., 20., 25., 30.])
        dh = np.array([-25., -10., 0., 10., 25.])
        dh_ds = np.array([-25., -10., 0., 10., 15., 20., 25.])
        dh_n = np.array([-25., 0., 25.])
        M = np.arange(0.0, 1.2, 0.2)
        H = np.arange(0.0, 60000., 10000.)

        # X-force coefficient data import
        C_X = np.load(data_dir + "C_X(a,b,d_h).npy")
        self.CX_abdh = rgi((dh, alpha, beta), C_X, bounds_error=False)
        C_Xlef = np.load(data_dir + "C_X,lef(a,b).npy")
        self.CXlef_ab = rgi((alpha_lef, beta), C_Xlef, bounds_error=False)
        C_Xq = np.load(data_dir + "C_X_q(a).npy")
        self.CXq_a = interp1d(alpha, C_Xq, bounds_error=False,
                              fill_value="extrapolate")
        DC_Xsb = np.load(data_dir + "DC_X,sb(a).npy")
        self.DCXsb_a = interp1d(alpha, DC_Xsb, bounds_error=False,
                                fill_value="extrapolate")
        DC_Xqlef = np.load(data_dir + "DC_X_q,lef(a).npy")
        self.DCXqlef_a = interp1d(alpha_lef, DC_Xqlef, bounds_error=False,
                                  fill_value="extrapolate")

        # Y-force coefficient data import
        C_Y = np.load(data_dir + "C_Y(a,b).npy")
        self.CY_ab = rgi((alpha, beta), C_Y, bounds_error=False)
        C_Yda20 = np.load(data_dir + "C_Y,d_a=20(a,b).npy")
        self.CYda20_ab = rgi((alpha, beta), C_Yda20, bounds_error=False)
        C_Yda20lef = np.load(data_dir + "C_Y,d_a=20,lef(a,b).npy")
        self.CYda20lef_ab = rgi((alpha_lef, beta), C_Yda20lef, bounds_error=False)
        C_Ydr30 = np.load(data_dir + "C_Y,d_r=30(a,b).npy")
        self.CYdr30_ab = rgi((alpha, beta), C_Ydr30, bounds_error=False)
        C_Ylef = np.load(data_dir + "C_Y,lef(a,b).npy")
        self.CYlef_ab = rgi((alpha_lef, beta), C_Ylef, bounds_error=False)
        C_Yp = np.load(data_dir + "C_Y_p(a).npy")
        self.CYp_a = interp1d(alpha, C_Yp, bounds_error=False,
```

```
                                  fill_value="extrapolate")
        C_Yr = np.load(data_dir + "C_Y_r(a).npy")
        self.CYr_a = interp1d(alpha, C_Yr, bounds_error=False,
                              fill_value="extrapolate")
        DC_Yplef = np.load(data_dir + "DC_Y_p,lef(a).npy")
        self.DCYplef_a = interp1d(alpha_lef, DC_Yplef, bounds_error=False,
                                  fill_value="extrapolate")
        DC_Yrlef = np.load(data_dir + "DC_Y_r,lef(a).npy")
        self.DCYrlef_a = interp1d(alpha_lef, DC_Yrlef, bounds_error=False,
                                  fill_value="extrapolate")

        # Z-force coefficient data import
        C_Z = np.load(data_dir + "C_Z(a,b,d_h).npy")
        self.CZ_abdh = rgi((dh, alpha, beta), C_Z, bounds_error=False)
        C_Zlef = np.load(data_dir + "C_Z,lef(a,b).npy")
        self.CZlef_ab = rgi((alpha_lef, beta), C_Zlef, bounds_error=False)
        C_Zq = np.load(data_dir + "C_Z_q(a).npy")
        self.CZq_a = interp1d(alpha, C_Zq, bounds_error=False,
                              fill_value="extrapolate")
        DC_Zsb = np.load(data_dir + "DC_Z,sb(a).npy")
        self.DCZsb_a = interp1d(alpha, DC_Zsb, bounds_error=False,
                                fill_value="extrapolate")
        DC_Zqlef = np.load(data_dir + "DC_Z_q,lef(a).npy")
        self.DCZqlef_a = interp1d(alpha_lef, DC_Zqlef, bounds_error=False,
                                  fill_value="extrapolate")

        # Pitching moment coefficient data import
        C_m = np.load(data_dir + "C_m(a,b,d_h).npy")
        self.Cm_abdh = rgi((dh, alpha, beta), C_m, bounds_error=False)
        C_mlef = np.load(data_dir + "C_m,lef(a,b).npy")
        self.Cmlef_ab = rgi((alpha_lef, beta), C_mlef, bounds_error=False)
        C_mq = np.load(data_dir + "C_m_q(a).npy")
        self.Cmq_a = interp1d(alpha, C_mq, bounds_error=False,
                              fill_value="extrapolate")
        DC_m = np.load(data_dir + "DC_m(a).npy")
        self.DCm_a = interp1d(alpha, DC_m, bounds_error=False,
                              fill_value="extrapolate")
        DC_mds = np.load(data_dir + "DC_m,ds(a,d_h).npy")
        self.DCmds_adh = rgi((alpha, dh_ds), DC_mds, bounds_error=False)
        DC_msb = np.load(data_dir + "DC_m,sb(a).npy")
        self.DCmsb_a = interp1d(alpha, DC_msb, bounds_error=False,
                                fill_value="extrapolate")
        DC_mqlef = np.load(data_dir + "DC_m_q,lef(a).npy")
        self.DCmqlef_a = interp1d(alpha_lef, DC_mqlef, bounds_error=False,
                                  fill_value="extrapolate")
        n_dh = np.load(data_dir + "n_d_h(d_h).npy")
        self.ndh_dh = interp1d(dh, n_dh, bounds_error=False,
                               fill_value="extrapolate")

        # Rolling moment coefficient data import
        C_l = np.load(data_dir + "C_l(a,b,d_h).npy")
        self.Cl_abdh = rgi((dh_n, alpha, beta), C_l, bounds_error=False)
        C_lda20 = np.load(data_dir + "C_l,d_a=20(a,b).npy")
        self.Clda20_ab = rgi((alpha, beta), C_lda20, bounds_error=False)
        C_lda20lef = np.load(data_dir + "C_l,d_a=20,lef(a,b).npy")
        self.Clda20lef_ab = rgi((alpha_lef, beta), C_lda20lef, bounds_error=False)
```

```
        C_ldr30 = np.load(data_dir + "C_l,d_r=30(a,b).npy")
        self.Cldr30_ab = rgi((alpha, beta), C_ldr30, bounds_error=False)
        C_llef = np.load(data_dir + "C_l,lef(a,b).npy")
        self.Cllef_ab = rgi((alpha_lef, beta), C_llef, bounds_error=False)
        C_lp = np.load(data_dir + "C_l_p(a).npy")
        self.Clp_a = interp1d(alpha, C_lp, bounds_error=False,
                              fill_value="extrapolate")
        C_lr = np.load(data_dir + "C_l_r(a).npy")
        self.Clr_a = interp1d(alpha, C_lr, bounds_error=False,
                              fill_value="extrapolate")
        DC_lb = np.load(data_dir + "DC_l_b(a).npy")
        self.DClb_a = interp1d(alpha, DC_lb, bounds_error=False,
                               fill_value="extrapolate")
        DC_lplef = np.load(data_dir + "DC_l_p,lef(a).npy")
        self.DClplef_a = interp1d(alpha_lef, DC_lplef, bounds_error=False,
                                  fill_value="extrapolate")
        DC_lrlef = np.load(data_dir + "DC_l_r,lef(a).npy")
        self.DClrlef_a = interp1d(alpha_lef, DC_lrlef, bounds_error=False,
                                  fill_value="extrapolate")

        # Yawing moment coefficient data import
        C_n = np.load(data_dir + "C_n(a,b,d_h).npy")
        self.Cn_abdh = rgi((dh_n, alpha, beta), C_n, bounds_error=False)
        C_nda20 = np.load(data_dir + "C_n,d_a=20(a,b).npy")
        self.Cnda20_ab = rgi((alpha, beta), C_nda20, bounds_error=False)
        C_nda20lef = np.load(data_dir + "C_n,d_a=20,lef(a,b).npy")
        self.Cnda20lef_ab = rgi((alpha_lef, beta), C_nda20lef, bounds_error=False)
        C_ndr30 = np.load(data_dir + "C_n,d_r=30(a,b).npy")
        self.Cndr30_ab = rgi((alpha, beta), C_ndr30, bounds_error=False)
        C_nlef = np.load(data_dir + "C_n,lef(a,b).npy")
        self.Cnlef_ab = rgi((alpha_lef, beta), C_nlef, bounds_error=False)
        C_np = np.load(data_dir + "C_n_p(a).npy")
        self.Cnp_a = interp1d(alpha, C_np, bounds_error=False,
                              fill_value="extrapolate")
        C_nr = np.load(data_dir + "C_n_r(a).npy")
        self.Cnr_a = interp1d(alpha, C_nr, bounds_error=False,
                              fill_value="extrapolate")
        DC_nb = np.load(data_dir + "DC_n_b(a).npy")
        self.DCnb_a = interp1d(alpha, DC_nb, bounds_error=False,
                               fill_value="extrapolate")
        DC_nda = np.load(data_dir + "DC_n_d_a(a).npy")
        self.DCnda_a = interp1d(alpha, DC_nda, bounds_error=False,
                                fill_value="extrapolate")
        DC_nplef = np.load(data_dir + "DC_n_p,lef(a).npy")
        self.DCnplef_a = interp1d(alpha_lef, DC_nplef, bounds_error=False,
                                  fill_value="extrapolate")
        DC_nrlef = np.load(data_dir + "DC_n_r,lef(a).npy")
        self.DCnrlef_a = interp1d(alpha_lef, DC_nrlef, bounds_error=False,
                                  fill_value="extrapolate")

        T = np.load(data_dir + "Thrust_En(M,H).npy")
        self.T_idle = rgi((M, H), T[0])
        self.T_mil = rgi((M, H), T[1])
        self.T_max = rgi((M, H), T[2])
```

```
    def set_state(self, alpha, beta, dh, dlef, dsb, da, dr,
                  xcg_cref, p, q, r, M, H, thtl):
        self.alpha = alpha
        self.beta = beta
        self.dh = dh
        self.dlef = dlef
        self.dsb = dsb
        self.da = da
        self.dr = dr
        self.xcg_cref = xcg_cref
        self.p = p
        self.q = q
        self.r = r
        self.H = H
        self.rho, self.a = atmos.stdatm_english(H)[-2:]
        self.M = M
        self.V = M*self.a
        self.pbar = p*self.b_w/(2.*self.V)
        self.qbar = q*self.c_ref/(2.*self.V)
        self.rbar = r*self.b_w/(2.*self.V)
        self.thtl = thtl

    def _CX(self):
        DC_Xlef = self.CXlef_ab([self.alpha, self.beta]) - \
                  self.CX_abdh([0., self.alpha, self.beta])
        C_Xt = self.CX_abdh([self.dh, self.alpha, self.beta]) + \
               DC_Xlef*(1. - (self.dlef/25.)) + \
               self.DCXsb_a(self.alpha)*(self.dsb/60.) + \
               self.qbar*(self.CXq_a(self.alpha) +
                          self.DCXqlef_a(self.alpha)*(1. - (self.dlef/25.)))
        return C_Xt[0]

    def _CZ(self):
        DC_Zlef = self.CZlef_ab([self.alpha, self.beta]) - \
                  self.CZ_abdh([0., self.alpha, self.beta])
        C_Zt = self.CZ_abdh([self.dh, self.alpha, self.beta]) + \
               DC_Zlef*(1. - (self.dlef/25.)) + \
               self.DCZsb_a(self.alpha)*(self.dsb/60.) + \
               self.qbar*(self.CZq_a(self.alpha) +
                          self.DCZqlef_a(self.alpha)*(1. - (self.dlef/25.)))
        return C_Zt[0]

    def _Cm(self):
        DC_mlef = self.Cmlef_ab([self.alpha, self.beta]) - \
                  self.Cm_abdh([0., self.alpha, self.beta])
        C_mt = self.Cm_abdh([self.dh, self.alpha, self.beta])*self.ndh_dh(self.dh) + \
               self.C_Zt*(self.xcgref_cref - self.xcg_cref) + \
               DC_mlef*(1. - (self.dlef/25.)) + \
               self.DCmsb_a(self.alpha)*(self.dsb/60.) + \
               self.qbar*(self.Cmq_a(self.alpha) +
                          self.DCmqlef_a(self.alpha)*(1. - (self.dlef/25.))) + \
               self.DCm_a(self.alpha) + \
               self.DCmds_adh([self.alpha, self.dh])
        return C_mt[0]

    def _CY(self):
```

```
            ab = [self.alpha, self.beta]
            DC_Ylef = self.CYlef_ab(ab) - self.CY_ab(ab)
            DC_Yda20 = self.CYda20_ab(ab) - self.CY_ab(ab)
            DC_Yda20lef = self.CYda20lef_ab(ab) - self.CYlef_ab(ab) - DC_Yda20
            DC_Ydr30 = self.CYdr30_ab(ab) - self.CY_ab(ab)
            C_Yt = self.CY_ab(ab) + \
                   DC_Ylef*(1. - (self.dlef/25.)) + \
                   (DC_Yda20 + DC_Yda20lef*(1. - (self.dlef/25.)))*(self.da/20.) + \
                   DC_Ydr30*(self.dr/30.) + \
                   self.rbar*(self.CYr_a(self.alpha) +
                              self.DCYrlef_a(self.alpha)*(1. - (self.dlef/25.))) + \
                   self.pbar*(self.CYp_a(self.alpha) +
                              self.DCYplef_a(self.alpha)*(1. - (self.dlef/25.)))
            return C_Yt[0]

        def _Cn(self):
            ab = [self.alpha, self.beta]
            Cn_abdh0 = self.Cn_abdh([0., self.alpha, self.beta])
            DC_nlef = self.Cnlef_ab(ab) - Cn_abdh0
            DC_nda20 = self.Cnda20_ab(ab) - Cn_abdh0
            DC_nda20lef = self.Cnda20lef_ab(ab) - self.Cnlef_ab(ab) - DC_nda20
            DC_ndr30 = self.Cndr30_ab(ab) - Cn_abdh0
            C_nt = self.Cn_abdh([self.dh, self.alpha, self.beta]) + \
                   DC_nlef*(1. - (self.dlef/25.)) - \
                   self.C_Yt*(self.xcgref_cref - self.xcg_cref)*(self.c_ref/self.b_w) + \
                   (DC_nda20 + DC_nda20lef*(1. - (self.dlef/25.)))*(self.da/20.) + \
                   DC_ndr30*(self.dr/30.) + \
                   self.rbar*(self.Cnr_a(self.alpha) +
                              self.DCnrlef_a(self.alpha)*(1. - (self.dlef/25.))) + \
                   self.pbar*(self.Cnp_a(self.alpha) +
                              self.DCnplef_a(self.alpha)*(1. - (self.dlef/25.))) + \
                   self.DCnb_a(self.alpha)*np.deg2rad(self.beta)
            return C_nt[0]

        def _Cl(self):
            ab = [self.alpha, self.beta]
            Cl_abdh0 = self.Cl_abdh([0., self.alpha, self.beta])
            DC_llef = self.Cllef_ab(ab) - Cl_abdh0
            DC_lda20 = self.Clda20_ab(ab) - Cl_abdh0
            DC_lda20lef = self.Clda20lef_ab(ab) - self.Cllef_ab(ab) - DC_lda20
            DC_ldr30 = self.Cldr30_ab(ab) - Cl_abdh0
            C_lt = self.Cl_abdh([self.dh, self.alpha, self.beta]) + \
                   DC_llef*(1. - (self.dlef/25.)) + \
                   (DC_lda20 + DC_lda20lef*(1. - (self.dlef/25.)))*(self.da/20.) + \
                   DC_ldr30*(self.dr/30.) + \
                   self.pbar*(self.Clp_a(self.alpha) +
                              self.DClplef_a(self.alpha)*(1. - (self.dlef/25.))) + \
                   self.rbar*(self.Clr_a(self.alpha) +
                              self.DClrlef_a(self.alpha)*(1. - (self.dlef/25.))) + \
                   self.DClb_a(self.alpha)*np.deg2rad(self.beta)
            return C_lt[0]

        def _tau_inv(self, dp):
            if dp <= 25.0:
                t_inv = 1.0
            elif dp >= 50.0:
```

```
                t_inv = 0.1
            else:
                t_inv = 1.9 - 0.036*dp
            return t_inv

    def thrust(self):
        MH = [self.M, self.H]
        if self.thtl <= 0.77:
            pow_c = 64.94*self.thtl
        else:
            pow_c = 217.38*self.thtl - 117.38
        if pow_c >= 50.0:
            T = self.T_mil(MH) + (self.T_max(MH) -
                                  self.T_mil(MH))*(pow_c - 50.0)/50.0
        else:
            T = self.T_idle(MH) + (self.T_mil(MH) - self.T_idle(MH))*pow_c/50.0
        return T

    def _body_to_stab(self, FM, alpha):
        CX, CY, CZ, Cl, Cm, Cn = FM
        CA = -CX
        CN = -CZ
        alpha_rad = np.deg2rad(alpha)
        c_a = np.cos(alpha_rad)
        s_a = np.sin(alpha_rad)
        CD_s = CA*c_a + CN*s_a
        CL = CN*c_a - CA*s_a
        Cl_s = Cl*c_a + Cn*s_a
        Cn_s = Cn*c_a  - Cl*s_a
        return [CD_s, CY, CL, Cl_s, Cm, Cn_s]

    def _body_to_wind(self, FM, alpha, beta):
        CX, CY, CZ, Cl, Cm, Cn = FM
        CA = -CX
        CN = -CZ
        alpha_rad = np.deg2rad(alpha)
        beta_rad = np.deg2rad(beta)
        c_a = np.cos(alpha_rad)
        s_a = np.sin(alpha_rad)
        c_b = np.cos(beta_rad)
        s_b = np.sin(beta_rad)
        CD = CA*c_a*c_b - CY*s_b + CN*s_a*c_b
        CS = CA*c_a*s_b + CY*c_b + CN*s_a*s_b
        CL = CN*c_a - CA*s_a
        Cl_w = Cl*c_a*c_b + Cm*s_b + Cn*s_a*c_b
        Cm_w = Cm*c_b - Cl*c_a*s_b - Cn*s_a*s_b
        Cn_w = Cn*c_a - Cl*s_a
        return [CD, CS, CL, Cl_w, Cm_w, Cn_w]


    def calc_forces(self, body_frame=True, stab_frame=False,
                    wind_frame=False, dimensional=False,
                    verbose=False):
        self.C_Xt = self._CX()
        self.C_Zt = self._CZ()
        self.C_mt = self._Cm()
```

```
        self.C_Yt = self._CY()
        self.C_nt = self._Cn()
        self.C_lt = self._Cl()
        body_fm = [self.C_Xt, self.C_Yt, self.C_Zt,
                   self.C_lt, self.C_mt, self.C_nt]
        dim_const = 0.5*self.rho*self.V**2*self.S_w
        forces = {'F16' : {}}
        if body_frame:
            body_keys = ['CX', 'CY', 'CZ', 'Cl', 'Cm', 'Cn']
            forces['F16'].update({key : fm for key, fm in zip(body_keys, body_fm)})
            if dimensional:
                body_keys_dim = ['Fx_b', 'Fy_b', 'Fz_b', 'Mx_b', 'My_b', 'Mz_b']
                body_fm_dim = [fm*dim_const for fm in body_fm]
                body_fm_dim[3] *= self.b_w
                body_fm_dim[4] *= self.c_ref
                body_fm_dim[5] *= self.b_w
                forces['F16'].update({key : fm for key, fm in zip(body_keys_dim,
                                                                  body_fm_dim)})
        if stab_frame:
            stab_keys = ['CD_s', 'CY_s', 'CL_s', 'Cl_s', 'Cm_s', 'Cn_s']
            stab_fm = self._body_to_stab(body_fm, self.alpha)
            forces['F16'].update({key : fm for key, fm in zip(stab_keys, stab_fm)})
            if dimensional:
                stab_keys_dim = ['Fx_s', 'Fy_s', 'Fz_s', 'Mx_s', 'My_s', 'Mz_s']
                stab_fm_dim = [fm*dim_const for fm in stab_fm]
                stab_fm_dim[3] *= self.b_w
                stab_fm_dim[4] *= self.c_ref
                stab_fm_dim[5] *= self.b_w
                forces['F16'].update({key : fm for key, fm in zip(stab_keys_dim,
                                                                  stab_fm_dim)})
        if wind_frame:
            wind_keys = ['CD', 'CS', 'CL', 'Cl_w', 'Cm_w', 'Cn_w']
            wind_fm = self._body_to_wind(body_fm, self.alpha, self.beta)
            forces['F16'].update({key : fm for key, fm in zip(wind_keys, wind_fm)})
            if dimensional:
                wind_keys_dim = ['Fx_w', 'Fy_w', 'Fz_w', 'Mx_w', 'My_w', 'Mz_w']
                wind_fm_dim = [fm*dim_const for fm in wind_fm]
                wind_fm_dim[3] *= self.b_w
                wind_fm_dim[4] *= self.c_ref
                wind_fm_dim[5] *= self.b_w
                forces['F16'].update({key : fm for key, fm in zip(wind_keys_dim,
                                                                  wind_fm_dim)})
        if verbose:
            print(forces['F16'])
        return forces

if __name__ == "__main__":
    case = F16_windtunnel()
    alpha = 0.
    beta = 0.
    dh = 10.
    da = 0.
    dr = 0.
    p = 0.
    q = 0.
    r = 0.
```

```
    xcg_cref = 0.
    dsb = 0.
    dlef = 0.
    H = 1000.
    M = 0.2
    thtl = 0.
    case.set_state(alpha, beta, dh, dlef, dsb, da, dr, xcg_cref, p, q, r, M, H, thtl)
    forces_options = {'body_frame': True,
                      'stab_frame': True,
                      'wind_frame': True,
                      'dimensional': True,
                      'verbose': True}
    fm = case.calc_forces(**forces_options)
```

Baseline Aerodynamic Model

```
import numpy as np
import matplotlib.pyplot as plt
import machupX as mx
import pandas as pd
from os.path import exists
import json
import windtunnelmodel as wind


def generate_data(params):
    alpha = params[0]
    beta = params[1]
    d_e = params[2]
    d_a = params[3]
    d_r = params[4]
    p = params[5]
    q = params[6]
    r = params[7]
    rates = [p, q, r]
    if nasa:
        case.set_state(alpha, beta, d_e, 0., 0., d_a, d_r,
                       0.35, p, q, r, 0.2, 1000., 0.)
        x = case.calc_forces(**forces_options)["F16"]
    else:
        my_scene.set_aircraft_state(state={"alpha": alpha,
                                           "beta": beta,
                                           "angular_rates": rates,
                                           "velocity": 222.5211})
        my_scene.set_aircraft_control_state(control_state={"elevator": d_e,
                                                           "aileron": d_a,
                                                           "rudder": d_r})
        x = my_scene.solve_forces(**forces_options)["F16"]["total"]
    fm = [x['CD'], x['CS'], x['CL'], x['Cl'], x['Cm'], x['Cn']]
    return (*params, *fm)

def plot_model(x, y, color, ls, tag, marker, ax, label):
    first = True
    for i in range(len(y)):
        if tag[i] == "data":
            if first:
                ax.scatter(x, y[i], ec=color[i], fc="None",
                           marker=marker, label=label)
            else:
                ax.scatter(x, y[i], ec=color[i], fc="None", marker=marker)
            first = False
        else:
            ax.plot(x, y[i], color=color[i], linestyle=ls[i])

def remove_outliers(data, m=2.):
    d = np.abs(data - np.median(data))
    mdev = np.median(d)
    s = d/mdev if mdev else 0.
    return s < m

```

```
def _CL0_CLalpha(CLalpha_data, plot, skip_mask=False):
    alphas = CLalpha_data[:, 0]*np.pi/180.
    CL = CLalpha_data[:, 10]
    if skip_mask:
        out_mask = [True]*len(alphas)
    else:
        out_mask = remove_outliers(CL)
    [CL_alpha, CL0] = np.polyfit(alphas[out_mask], CL[out_mask], 1)
    if plot:
        x = alphas*180./np.pi
        y = [CL, CL0 + CL_alpha*alphas]
        color = ['0.0', '0.0']
        ls = ['-', '-']
        tag = ['data', 'not']
        marker = 'o'
        return [x, y, color, ls, tag, marker]
    return CL0, CL_alpha

def _CL_de(CLde_data, plot, skip_mask=False):
    CLde_p = np.array([x[10] for x in CLde_data if x[2] == 10.])
    CLde_m = np.array([x[10] for x in CLde_data if x[2] == -10.])
    de = np.deg2rad(10.)
    CL_de = (CLde_p - CLde_m)/(2.*de)
    if skip_mask:
        out_mask = [True]*len(CLde_p)
    else:
        out_mask = remove_outliers(CL_de)
    CL_de = np.average(CL_de[out_mask])

    CL0, CL_alpha = _CL0_CLalpha(np.array([x for x in CLde_data if x[2] == 0.]),
                                 False)
    a0 = np.array([x[0] for x in CLde_data if x[2] == 0.])
    alpha = a0/180.*np.pi
    CL1 = CL0 + CL_alpha*alpha
    if plot:
        x = a0
        y = [CLde_p, CLde_m, CL1 + CL_de*de, CL1 + CL_de*-de]
        color = ['0.0']*4
        ls = ['--']*4
        tag = ['data']*2 + ['not']*2
        marker = 'v'
        return [x, y, color, ls, tag, marker]
    return CL_de

def _CL_qbar(CLqbar_data, plot, skip_mask=False):
    CLq_p = np.array([x[10] for x in CLqbar_data if x[6] == 30.*np.pi/180.])
    CLq_m = np.array([x[10] for x in CLqbar_data if x[6] == -30.*np.pi/180.])
    qbar = np.deg2rad(30.)*c_w/(2.*V)
    CL_qbar = (CLq_p - CLq_m)/(2.*qbar)
    if skip_mask:
        out_mask = [True]*len(CLq_p)
    else:
        out_mask = remove_outliers(CL_qbar)
    CL_qbar = np.average(CL_qbar[out_mask])

    CL0, CL_alpha = _CL0_CLalpha(np.array([x for x in CLqbar_data if x[6] == 0.]),
```

```
                                        False)
    a0 = np.array([x[0] for x in CLqbar_data if x[6] == 0.])
    alpha = a0/180.*np.pi
    CL1 = CL0 + CL_alpha*alpha
    if plot:
        x = a0
        y = [CLq_p, CLq_m, CL1 + CL_qbar*qbar, CL1 + CL_qbar*-qbar]
        color = ['0.0']*4
        ls = [':']*4
        tag = ['data']*2 + ['not']*2
        marker = '^'
        return [x, y, color, ls, tag, marker]
    return CL_qbar

def _CS_beta(CSbeta_data, plot, skip_mask=False):
    betas = CSbeta_data[:, 1]*np.pi/180.
    CS = CSbeta_data[:, 9]
    if skip_mask:
        out_mask = [True]*len(betas)
    else:
        out_mask = remove_outliers(CS)
    [CS_beta, CS0] = np.polyfit(betas[out_mask], CS[out_mask], 1)
    if plot:
        x = betas*180./np.pi
        y = [CS, CS0 + CS_beta*betas]
        color = ['0.0']*2
        ls = ['-']*2
        tag = ['data', 'not']
        marker = 'o'
        return [x, y, color, ls, tag, marker]
    return CS0, CS_beta

def _CS_da(CSda_data, plot, skip_mask=False):
    CS1 = np.array([x[9] for x in CSda_data if x[3] == 0.])
    CSda_p = np.array([x[9] for x in CSda_data if x[3] == 20.])
    da = np.deg2rad(20.)
    CS_da = (CSda_p - CS1)/da
    if skip_mask:
        out_mask = [True]*len(CS1)
    else:
        out_mask = remove_outliers(CS_da)
    CS_da = np.average(CS_da[out_mask])

    CS0, CS_beta = _CS_beta(np.array([x for x in CSda_data if x[3] == 0.]), False)
    b0 = np.array([x[1] for x in CSda_data if x[3] == 0.])
    beta = b0*np.pi/180.
    CS1 = CS0 + CS_beta*beta
    if plot:
        x = b0
        y = [CSda_p, CS1 + CS_da*da]
        color = ['0.0']*2
        ls = ['--']*2
        tag = ['data', 'not']
        marker = 'v'
        return [x, y, color, ls, tag, marker]
    return CS_da
```

```

def _CS_dr(CSdr_data, plot, skip_mask=False):
    CS1 = np.array([x[9] for x in CSdr_data if x[4] == 0.])
    CSdr_p = np.array([x[9] for x in CSdr_data if x[4] == 30.])
    dr = np.deg2rad(30.)
    CS_dr = (CSdr_p - CS1)/dr
    if skip_mask:
        out_mask = [True]*len(CS1)
    else:
        out_mask = remove_outliers(CS_dr)
    CS_dr = np.average(CS_dr[out_mask])

    CS0, CS_beta = _CS_beta(np.array([x for x in CSdr_data if x[4] == 0.]), False)
    b0 = np.array([x[1] for x in CSdr_data if x[4] == 0.])
    beta = b0*np.pi/180.
    CS1 = CS0 + CS_beta*beta
    if plot:
        x = b0
        y = [CSdr_p, CS1 + CS_dr*dr]
        color = ['0.0']*2
        ls = [':']*2
        tag = ['data', 'not']
        marker = '^'
        return [x, y, color, ls, tag, marker]
    return CS_dr

def _CS_rbar(CSr_data, plot, skip_mask=False):
    CSr_p = np.array([x[9] for x in CSr_data if x[7] == 30.*np.pi/180.])
    CSr_m = np.array([x[9] for x in CSr_data if x[7] == -30.*np.pi/180.])
    rbar = np.deg2rad(30.)*b_w/(2.*V)
    CS_rbar = (CSr_p - CSr_m)/(2.*rbar)
    if skip_mask:
        out_mask = [True]*len(CSr_p)
    else:
        out_mask = remove_outliers(CS_rbar)
    CS_rbar = np.average(CS_rbar[out_mask])

    CS1 = np.array([x[9] for x in CSr_data if x[7] == 0.])
    a0 = np.array([x[0] for x in CSr_data if x[7] == 0.])
    if plot:
        x = a0
        y = [CSr_p, CSr_m, CS1 + CS_rbar*rbar, CS1 + CS_rbar*-rbar]
        color = ['0.0']*4
        ls = ['-.']*4
        tag = ['data', 'data', 'not', 'not']
        marker = '<'
        return [x, y, color, ls, tag, marker]
    return CS_rbar

def _CS_pbar(CSp_data, plot, skip_mask=False):
    CS1 = np.array([x[9] for x in CSp_data if x[5] == 0.])
    CL1 = np.array([x[10] for x in CSp_data if x[5] == 0.])
    CSp_p = np.array([x[9] for x in CSp_data if x[5] == 90.*np.pi/180.])
    CSp_m = np.array([x[9] for x in CSp_data if x[5] == -90.*np.pi/180.])
    pbar = np.deg2rad(90.)*b_w/(2.*V)
    CS_pbar = (CSp_p - CSp_m)/(2.*pbar)
```

```
    if skip_mask:
        out_mask = [True]*len(CS1)
    else:
        out_mask = remove_outliers(CS_pbar)
    [CS_Lpbar, CS_pbar] = np.polyfit(CL1[out_mask], CS_pbar[out_mask], 1)

    CL0, CL_alpha = _CL0_CLalpha(np.array([x for x in CSp_data if x[5] == 0.]), False)
    a0 = np.array([x[0] for x in CSp_data if x[5] == 0.])
    alpha = a0/180.*np.pi
    CL1 = CL0 + CL_alpha*alpha
    if plot:
        x = a0
        y = [CSp_p, CSp_m, CS1 + (CS_Lpbar*CL1 + CS_pbar)*pbar,
             CS1 + (CS_Lpbar*CL1 + CS_pbar)*-pbar]
        color = ['0.0']*4
        ls = [(0, (3, 5, 1, 5, 1, 5))]*4
        tag = ['data', 'data', 'not', 'not']
        marker = '>'
        return [x, y, color, ls, tag, marker]
    return CS_pbar, CS_Lpbar

def _CD_de(CDde_data, plot, skip_mask=False):
    CD1 = np.array([x[8] for x in CDde_data if x[2] == 0.])
    CL1 = np.array([x[10] for x in CDde_data if x[2] == 0.])
    CDde_p = np.array([x[8] for x in CDde_data if x[2] == 10.])
    CDde_m = np.array([x[8] for x in CDde_data if x[2] == -10.])
    de = np.deg2rad(10.)
    CD_de = (CDde_p - CDde_m)/(2.*de)
    if skip_mask:
        out_mask = [True]*len(CD1)
    else:
        out_mask = remove_outliers(CD_de)
    [CD_Lde, CD_de] = np.polyfit(CL1[out_mask], CD_de[out_mask], 1)
    if skip_mask:
        out_mask = [True]*len(CD1)
    else:
        out_mask = remove_outliers(CD1)
    CD_de2 = np.average((CDde_p - CD1)[out_mask]/(de**2))

    CL0, CL_alpha = _CL0_CLalpha(np.array([x for x in CDde_data if x[2] == 0.]),
                                 False)
    a0 = np.array([x[0] for x in CDde_data if x[2] == 0.])
    alpha = a0/180.*np.pi
    CL = CL0 + CL_alpha*alpha
    CD_0, CD_L, CD_L2 = _CD_polar(CDde_data, False)
    CD1 = CD_0 + CD_L*CL + CD_L2*np.square(CL)
    if plot:
        x = CL1
        y = [CDde_p, CDde_m, CD1 + (CD_Lde*CL1 + CD_de)*de + CD_de2*de**2,
             CD1 + (CD_Lde*CL1 + CD_de)*-de + CD_de2*de**2]
        color = ['0.0']*4
        ls = ['--']*4
        tag = ['data', 'data', 'not', 'not']
        marker = 'v'
        return [x, y, color, ls, tag, marker]
    return CD_de, CD_Lde, CD_de2
```

```

def _CD_polar(CDalpha_data, plot, skip_mask=False):
    CD = CDalpha_data[:, 8]
    CL = CDalpha_data[:, 10]
    if skip_mask:
        out_mask = [True]*len(CD)
    else:
        out_mask = remove_outliers(CD)
        out_mask *= remove_outliers(CL)
        out_mask *= (CD >= 0.)
    [CD_L2, CD_L, CD_0] = np.polyfit(CL[out_mask], CD[out_mask], 2)
    if plot:
        x = CL
        y = [CD, CD_0 + CD_L*CL + CD_L2*np.square(CL)]
        color = ['0.0']*2
        ls = ['-']*2
        tag = ['data', 'not']
        marker = 'o'
        coeffs = [CD_0, CD_L, CD_L2]
        return [x, y, color, ls, tag, marker, coeffs]
    return CD_0, CD_L, CD_L2

def _CD_Spolar(CDbeta_data, plot, skip_mask=False):
    CD = CDbeta_data[:, 8]
    CS = CDbeta_data[:, 9]
    if skip_mask:
        out_mask = [True]*len(CD)
    else:
        out_mask = remove_outliers(CD, m=1.5)
        out_mask *= remove_outliers(CS, m=1.5)
        out_mask *= (CD >= 0.)
    [CD_S2, CD_S, CD_0] = np.polyfit(CS[out_mask], CD[out_mask], 2)
    if plot:
        x = CS
        y = [CD, CD_0 + CD_S*CS + CD_S2*np.square(CS)]
        color = ['0.3']*2
        ls = ['-']*2
        tag = ['data', 'not']
        marker = '<'
        coeffs = [CD_0, CD_S, CD_S2]
        return [x, y, color, ls, tag, marker, coeffs]
    return CD_0, CD_S, CD_S2

def _CD_pbar(CDp_data, plot, skip_mask=False):
    CD1 = np.array([x[8] for x in CDp_data if x[5] == 0.])
    CS1 = np.array([x[9] for x in CDp_data if x[5] == 0.])
    CDp_p = np.array([x[8] for x in CDp_data if x[5] == 90.*np.pi/180.])
    CDp_m = np.array([x[8] for x in CDp_data if x[5] == -90.*np.pi/180.])
    pbar = np.deg2rad(90.)*b_w/(2.*V)
    CD_pbar = (CDp_p - CDp_m)/(2.*pbar)
    if skip_mask:
        out_mask = [True]*len(CD1)
    else:
        out_mask = remove_outliers(CD_pbar)
        out_mask*= (CD1 >= 0.)
    [CD_Spbar, CD_pbar] = np.polyfit(CS1[out_mask], CD_pbar[out_mask], 1)
```

```

    # Shift to view accuracy of trends rather than discrepancy in CD_pbar
    CD1 = CDp_p[len(CD1)//2]
    if plot:
        x = CS1
        y = [CDp_p, CDp_m, CD1 + (CD_Spbar*CS1 + CD_pbar)*pbar,
             CD1 + (CD_Spbar*CS1 + CD_pbar)*-pbar]
        color = ['0.3']*4
        ls = ['--']*4
        tag = ['data', 'data', 'not', 'not']
        marker = '>'
        return [x, y, color, ls, tag, marker]
    return CD_pbar, CD_Spbar

def _CD_rbar(CDr_data, plot, skip_mask=False):
    CD1 = np.array([x[8] for x in CDr_data if x[7] == 0.])
    CS1 = np.array([x[9] for x in CDr_data if x[7] == 0.])
    CDr_p = np.array([x[8] for x in CDr_data if x[7] == 30.*np.pi/180.])
    CDr_m = np.array([x[8] for x in CDr_data if x[7] == -30.*np.pi/180.])
    rbar = np.deg2rad(30.)*b_w/(2.*V)
    CD_rbar = (CDr_p - CDr_m)/(2.*rbar)
    if skip_mask:
        out_mask = [True]*len(CD1)
    else:
        out_mask = remove_outliers(CD_rbar)
        out_mask *= (CD1 >= 0.)
    [CD_Srbar, CD_rbar] = np.polyfit(CS1[out_mask], CD_rbar[out_mask], 1)

     # Shift to view accuracy of trends rather than discrepancy in CD_rbar
    CD1 = CDr_p[len(CD1)//2]
    if plot:
        x = CS1
        y = [CDr_p, CDr_m, CD1 + (CD_Srbar*CS1 + CD_rbar)*rbar,
             CD1 + (CD_Srbar*CS1 + CD_rbar)*-rbar]
        color = ['0.3']*4
        ls = [':']*4
        tag = ['data', 'data', 'not', 'not']
        marker = 's'
        return [x, y, color, ls, tag, marker]
    return CD_rbar, CD_Srbar

def _CD_da(CDda_data, plot, skip_mask=False):
    CD1 = np.array([x[8] for x in CDda_data if x[3] == 0.])
    CS1 = np.array([x[9] for x in CDda_data if x[3] == 0.])
    CDda_p = np.array([x[8] for x in CDda_data if x[3] == 20.])
    CDda_m = np.array([x[8] for x in CDda_data if x[3] == -20.])
    da = np.deg2rad(20.)
    CD_da = (CDda_p - CDda_m)/(2.*da)
    if skip_mask:
        out_mask = [True]*len(CD1)
    else:
        out_mask = remove_outliers(CD_da)
        out_mask*= (CD1 >= 0.)
    [CD_Sda, CD_da] = np.polyfit(CS1[out_mask], CD_da[out_mask], 1)

     # Shift to view accuracy of trends rather than discrepancy in CD_da
```

```
    CD1 = CDda_p[len(CD1)//2]
    if plot:
        x = CS1
        y = [CDda_p, CDda_m, CD1 + (CD_Sda*CS1 + CD_da)*da,
             CD1 + (CD_Sda*CS1 + CD_da)*-da]
        color = ['0.3']*4
        ls = ['-.']*4
        tag = ['data', 'data', 'not', 'not']
        marker = 'h'
        return [x, y, color, ls, tag, marker]
    return CD_da, CD_Sda

def _CD_dr(CDdr_data, plot, skip_mask=False):
    CD1 = np.array([x[8] for x in CDdr_data if x[4] == 0.])
    CS1 = np.array([x[9] for x in CDdr_data if x[4] == 0.])
    CDdr_p = np.array([x[8] for x in CDdr_data if x[4] == 30.])
    CDdr_m = np.array([x[8] for x in CDdr_data if x[4] == -30.])
    dr = np.deg2rad(30.)
    CD_dr = (CDdr_p - CDdr_m)/(2.*dr)
    if skip_mask:
        out_mask = [True]*len(CD1)
    else:
        out_mask = remove_outliers(CD_dr)
        out_mask*= (CD1 >= 0.)
    [CD_Sdr, CD_dr] = np.polyfit(CS1[out_mask], CD_dr[out_mask], 1)

    CD_0, CD_S, CD_S2 = _CD_Spolar(np.array([x for x in CDdr_data if x[4] == 0.]),
                                   False)

     # Shift to view accuracy of trends rather than discrepancy in CD_dr
    CD1 = CDdr_p[len(CD1)//2]
    if plot:
        x = CS1
        y = [CDdr_p, CDdr_m, CD1 + (CD_Sdr*CS1 + CD_dr)*dr,
             CD1 + (CD_Sdr*CS1 + CD_dr)*-dr]
        color = ['0.3']*4
        ls = [(0, (3, 5, 1, 5, 1, 5))]*4
        tag = ['data', 'data', 'not', 'not']
        marker = 'd'
        return [x, y, color, ls, tag, marker]
    return CD_dr, CD_Sdr

def _CD_qbar(CDq_data, plot, skip_mask=False):
    CD1 = np.array([x[8] for x in CDq_data if x[6] == 0.])
    CL1 = np.array([x[10] for x in CDq_data if x[6] == 0.])
    CDq_p = np.array([x[8] for x in CDq_data if x[6] == 30.*np.pi/180.])
    CDq_m = np.array([x[8] for x in CDq_data if x[6] == -30.*np.pi/180.])
    qbar = np.deg2rad(30.)*c_w/(2.*V)
    CD_qbar = (CDq_p - CDq_m)/(2.*qbar)
    if skip_mask:
        out_mask = [True]*len(CD1)
    else:
        out_mask = remove_outliers(CD_qbar)
        out_mask *= (CD1 >= 0.)
    [CD_L2qbar, CD_Lqbar, CD_qbar] = np.polyfit(CL1[out_mask], CD_qbar[out_mask], 2)

```

```
    CL0, CL_alpha = _CL0_CLalpha(np.array([x for x in CDq_data if x[6] == 0.]), False)
    a0 = np.array([x[0] for x in CDq_data if x[6] == 0.])
    alpha = a0/180.*np.pi
    CL = CL0 + CL_alpha*alpha
    CD_0, CD_L, CD_L2 = _CD_polar(np.array([x for x in CDq_data if x[6] == 0.]),
                                  False)
    CD1 = CD_0 + CD_L*CL + CD_L2*np.square(CL)
    if plot:
        x = CL1
        y = [CDq_p, CDq_m,
             CD1 + (CD_L2qbar*np.square(CL1) + CD_Lqbar*CL1 + CD_qbar)*qbar,
             CD1 + (CD_L2qbar*np.square(CL1) + CD_Lqbar*CL1 + CD_qbar)*-qbar]
        color = ['0.0']*4
        ls = [':']*4
        tag = ['data', 'data', 'not', 'not']
        marker = '^'
        return [x, y, color, ls, tag, marker]
    return CD_qbar, CD_Lqbar, CD_L2qbar

def _Cl_beta(Clbeta_data, plot, skip_mask=False):
    betas = Clbeta_data[:, 1]*np.pi/180.
    Cl = Clbeta_data[:, 11]
    if skip_mask:
        out_mask = [True]*len(betas)
    else:
        out_mask = remove_outliers(Cl)
    [Cl_beta, Cl0] = np.polyfit(betas[out_mask], Cl[out_mask], 1)

    b0 = betas*180./np.pi
    if plot:
        x = b0
        y = [Cl, Cl0 + Cl_beta*betas]
        color = ['0.0']*2
        ls = ['-']*2
        tag = ['data', 'not']
        marker = 'o'
        return [x, y, color, ls, tag, marker]
    return Cl0, Cl_beta

def _Cl_pbar(Clp_data, plot, skip_mask=False):
    Cl1 = np.array([x[11] for x in Clp_data if x[5] == 0.])
    Clp_p = np.array([x[11] for x in Clp_data if x[5] == 90.*np.pi/180.])
    Clp_m = np.array([x[11] for x in Clp_data if x[5] == -90.*np.pi/180.])
    pbar = np.deg2rad(90.)*b_w/(2.*V)
    if skip_mask:
        out_mask = [True]*len(Cl1)
    else:
        out_mask = remove_outliers(Clp_p)
    Cl_pbar = np.average((Clp_p - Clp_m)[out_mask]/(2.*pbar))

    a0 = np.array([x[0] for x in Clp_data if x[5] == 0.])
    Cl1 = np.zeros(len(Cl1))
    if plot:
        x = a0
        y = [Clp_p, Clp_m, Cl1 + Cl_pbar*pbar, Cl1 + Cl_pbar*-pbar]
        color = ['0.0']*4
```

```
            ls = ['-.']*4
            tag = ['data', 'data', 'not', 'not']
            marker = '<'
            return [x, y, color, ls, tag, marker]
        return Cl_pbar

    def _Cl_rbar(Clr_data, plot, skip_mask=False):
        Cl1 = np.array([x[11] for x in Clr_data if x[7] == 0.])
        CL1 = np.array([x[10] for x in Clr_data if x[7] == 0.])
        Clr_p = np.array([x[11] for x in Clr_data if x[7] == 30.*np.pi/180.])
        Clr_m = np.array([x[11] for x in Clr_data if x[7] == -30.*np.pi/180.])
        rbar = np.deg2rad(30.)*b_w/(2.*V)
        Cl_rbar = (Clr_p - Clr_m)/(2.*rbar)
        if skip_mask:
            out_mask = [True]*len(CL1)
        else:
            out_mask = remove_outliers(Cl_rbar)
        [Cl_Lrbar, Cl_rbar] = np.polyfit(CL1[out_mask], Cl_rbar[out_mask], 1)

        a0 = np.array([x[0] for x in Clr_data if x[7] == 0.])
        Cl1 = np.zeros(len(a0))
        [CL0, CL_alpha] = _CL0_CLalpha(np.array([x for x in Clr_data if x[7] == 0.]),
                                      False)
        CL1 = CL0 + CL_alpha*a0*np.pi/180.
        if plot:
            x = a0
            y = [Clr_p, Clr_m, Cl1 + (Cl_Lrbar*CL1 + Cl_rbar)*rbar,
                 Cl1 + (Cl_Lrbar*CL1 + Cl_rbar)*-rbar]
            color = ['0.0']*4
            ls = [(0, (3, 5, 1, 5, 1, 5))]*4
            tag = ['data', 'data', 'not', 'not']
            marker = '>'
            return [x, y, color, ls, tag, marker]
        return Cl_rbar, Cl_Lrbar

    def _Cl_da(Clda_data, plot, skip_mask=False):
        Cl1 = np.array([x[11] for x in Clda_data if x[3] == 0.])
        Clda_p = np.array([x[11] for x in Clda_data if x[3] == 20.])
        da = np.deg2rad(20.)
        if skip_mask:
            out_mask = [True]*len(Cl1)
        else:
            out_mask = remove_outliers(Cl1)
        Cl_da = np.average((Clda_p - Cl1)[out_mask]/da)

        b0 = np.array([x[1] for x in Clda_data if x[3] == 0.])
        [Cl0, Cl_beta] = _Cl_beta(np.array([x for x in Clda_data if x[3] == 0.]), False)
        Cl1 = Cl0 + Cl_beta*b0*np.pi/180.
        if plot:
            x = b0
            y = [Clda_p, Cl1 + Cl_da*np.full(len(Cl1), da)]
            color = ['0.0']*2
            ls = ['--']*2
            tag = ['data', 'not']
            marker = 'v'
            return [x, y, color, ls, tag, marker]
```

```
    return Cl_da

def _Cl_dr(Cldr_data, plot, skip_mask=False):
    Cl1 = np.array([x[11] for x in Cldr_data if x[4] == 0.])
    Cldr_p = np.array([x[11] for x in Cldr_data if x[4] == 30.])
    dr = np.deg2rad(30.)
    if skip_mask:
        out_mask = [True]*len(Cl1)
    else:
        out_mask = remove_outliers(Cl1)
    Cl_dr = np.average((Cldr_p - Cl1)[out_mask]/dr)

    b0 = np.array([x[1] for x in Cldr_data if x[4] == 0.])
    [Cl0, Cl_beta] = _Cl_beta(np.array([x for x in Cldr_data if x[4] == 0.]), False)
    Cl1 = Cl0 + Cl_beta*b0*np.pi/180.
    if plot:
        x = b0
        y = [Cldr_p, Cl1 + Cl_dr*np.full(len(Cl1), dr)]
        color = ['0.0']*2
        ls = [':']*2
        tag = ['data', 'not']
        marker = '^'
        return [x, y, color, ls, tag, marker]
    return Cl_dr

def _Cm0_Cmalpha(Cmalpha_data, plot, skip_mask=False):
    alphas = Cmalpha_data[:, 0]*np.pi/180.
    Cm = Cmalpha_data[:, 12]
    if skip_mask:
        out_mask = [True]*len(alphas)
    else:
        out_mask = alphas <= 0.  # Effect of LEV
        out_mask[0] = False  # using data points centered around zero-lift alpha
    [Cm_alpha, Cm0] = np.polyfit(alphas[out_mask], Cm[out_mask], 1)

    a0 = Cmalpha_data[:, 0]
    if plot:
        x = a0
        y = [Cm, Cm0 + Cm_alpha*alphas]
        color = ['0.0']*2
        ls = ['-']*2
        tag = ['data', 'not']
        marker = 'o'
        return [x, y, color, ls, tag, marker]
    return Cm0, Cm_alpha

def _Cm_qbar(Cmq_data, plot, skip_mask=False):
    Cm1 = np.array([x[12] for x in Cmq_data if x[6] == 0.])
    Cmq_p = np.array([x[12] for x in Cmq_data if x[6] == 30.*np.pi/180.])
    Cmq_m = np.array([x[12] for x in Cmq_data if x[6] == -30.*np.pi/180.])
    qbar = np.deg2rad(30.)*c_w/(2.*V)
    if skip_mask:
        out_mask = [True]*len(Cm1)
    else:
        out_mask = remove_outliers(Cm1)
    Cm_qbar = np.average((Cmq_p - Cmq_m)[out_mask]/(2.*qbar))
```

```

    a0 = np.array([x[0] for x in Cmq_data if x[6] == 0.])
    [Cm0, Cm_alpha] = _Cm0_Cmalpha(np.array([x for x in Cmq_data if x[6] == 0.]),
                                   False)
    Cm1 = Cm0 + Cm_alpha*a0*np.pi/180.
    if plot:
        x = a0
        y = [Cmq_p, Cmq_m, Cm1 + Cm_qbar*qbar, Cm1 + Cm_qbar*-qbar]
        color = ['0.0']*4
        ls = ['--']*4
        tag = ['data', 'data', 'not', 'not']
        marker = 'v'
        return [x, y, color, ls, tag, marker]
    return Cm_qbar

def _Cm_de(Cmde_data, plot, skip_mask=False):
    Cm1 = np.array([x[12] for x in Cmde_data if x[2] == 0.])
    Cmde_p = np.array([x[12] for x in Cmde_data if x[2] == 10.])
    Cmde_m = np.array([x[12] for x in Cmde_data if x[2] == -10.])
    de = np.deg2rad(10.)
    if skip_mask:
        out_mask = [True]*len(Cm1)
    else:
        out_mask = remove_outliers(Cm1)
    Cm_de = np.average((Cmde_p - Cmde_m)[out_mask]/(2.*de))

    a0 = np.array([x[0] for x in Cmde_data if x[2] == 0.])
    [Cm0, Cm_alpha] = _Cm0_Cmalpha(np.array([x for x in Cmde_data if x[2] == 0.]),
                                   False)
    Cm1 = Cm0 + Cm_alpha*a0*np.pi/180.
    if plot:
        x = a0
        y = [Cmde_p, Cmde_m, Cm1 + Cm_de*de, Cm1 + Cm_de*-de]
        color = ['0.0']*4
        ls = [':']*4
        tag = ['data', 'data', 'not', 'not']
        marker = '^'
        return [x, y, color, ls, tag, marker]
    return Cm_de

def _Cn_beta(Cnbeta_data, plot, skip_mask=False):
    betas = Cnbeta_data[:, 1]*np.pi/180.
    Cn = Cnbeta_data[:, 13]
    if skip_mask:
        out_mask = [True]*len(Cn)
    else:
        out_mask = remove_outliers(Cn)
    [Cn_beta, Cn0] = np.polyfit(betas[out_mask], Cn[out_mask], 1)

    b0 = betas*180./np.pi
    if plot:
        x = b0
        y = [Cn, Cn0 + Cn_beta*betas]
        color = ['0.0']*2
        ls = ['-']*2
        tag = ['data', 'not']
```

```
            marker = 'o'
            return [x, y, color, ls, tag, marker]
        return Cn0, Cn_beta

    def _Cn_pbar(Cnp_data, plot, skip_mask=False):
        Cn1 = np.array([x[13] for x in Cnp_data if x[5] == 0.])
        Cnp_p = np.array([x[13] for x in Cnp_data if x[5] == 90.*np.pi/180.])
        Cnp_m = np.array([x[13] for x in Cnp_data if x[5] == -90.*np.pi/180.])
        CL1 = np.array([x[10] for x in Cnp_data if x[5] == 0.])
        pbar = np.deg2rad(90.)*b_w/(2.*V)
        Cn_pbar = (Cnp_p - Cnp_m)/(2.*pbar)
        if skip_mask:
            out_mask = [True]*len(Cn1)
        else:
            out_mask = remove_outliers(Cn_pbar)
        [Cn_Lpbar, Cn_pbar] = np.polyfit(CL1[out_mask], Cn_pbar[out_mask], 1)

        a0 = np.array([x[0] for x in Cnp_data if x[5] == 0.])
        Cn1 = np.zeros(len(Cn1))
        if plot:
            x = a0
            y = [Cnp_p, Cnp_m, Cn1 + (Cn_Lpbar*CL1 + Cn_pbar)*pbar,
                 Cn1 + (Cn_Lpbar*CL1 + Cn_pbar)*-pbar]
            color = ['0.0']*4
            ls = [':']*4
            tag = ['data', 'data', 'not', 'not']
            marker = '^'
            return [x, y, color, ls, tag, marker]
        return Cn_pbar, Cn_Lpbar

    def _Cn_rbar(Cnr_data, plot, skip_mask=False):
        Cn1 = np.array([x[13] for x in Cnr_data if x[7] == 0.])
        Cnr_p = np.array([x[13] for x in Cnr_data if x[7] == 30.*np.pi/180.])
        Cnr_m = np.array([x[13] for x in Cnr_data if x[7] == -30.*np.pi/180.])
        rbar = np.deg2rad(30.)*b_w/(2.*V)
        if skip_mask:
            out_mask = [True]*len(Cn1)
        else:
            out_mask = remove_outliers(Cnr_p)
        Cn_rbar = np.average((Cnr_p - Cnr_m)[out_mask]/(2.*rbar))

        a0 = np.array([x[0] for x in Cnr_data if x[7] == 0.])
        Cn1 = np.zeros(len(Cn1))
        if plot:
            x = a0
            y = [Cnr_p, Cnr_m, Cn1 + Cn_rbar*rbar, Cn1 + Cn_rbar*-rbar]
            color = ['0.0']*4
            ls = ['-.']*4
            tag = ['data', 'data', 'not', 'not']
            marker = '<'
            return [x, y, color, ls, tag, marker]
        return Cn_rbar

    def _Cn_da(Cnda_data, plot, skip_mask=False):
        Cn1 = np.array([x[13] for x in Cnda_data if x[3] == 0.])
        Cnda_p = np.array([x[13] for x in Cnda_data if x[3] == 20.])
```

```
    CL1 = np.array([x[10] for x in Cnda_data if x[3] == 0.])
    da = np.deg2rad(20.)
    Cn_da = (Cnda_p - Cn1)/da
    if skip_mask:
        out_mask = [True]*len(Cn1)
    else:
        out_mask = remove_outliers(Cn_da)
    [Cn_Lda, Cn_da] = np.polyfit(CL1[out_mask], Cn_da[out_mask], 1)

    a0 = np.array([x[0] for x in Cnda_data if x[3] == 0.])
    Cn1 = np.zeros(len(Cn1))
    if plot:
        x = a0
        y = [Cnda_p, Cn1 + (Cn_Lda*CL1 + Cn_da)*da]
        color = ['0.0']*2
        ls = [(0, (3, 5, 1, 5, 1, 5))]*2
        tag = ['data', 'not']
        marker = '>'
        return [x, y, color, ls, tag, marker]
    return Cn_da, Cn_Lda

def _Cn_dr(Cndr_data, plot, skip_mask=False):
    Cn1 = np.array([x[13] for x in Cndr_data if x[4] == 0.])
    Cndr_p = np.array([x[13] for x in Cndr_data if x[4] == 30.])
    dr = np.deg2rad(30.)
    if skip_mask:
        out_mask = [True]*len(Cn1)
    else:
        out_mask = remove_outliers(Cndr_p)
    Cn_dr = np.average((Cndr_p - Cn1)[out_mask]/dr)

    b0 = np.array([x[1] for x in Cndr_data if x[4] == 0.])
    [Cn0, Cn_beta] = _Cn_beta(np.array([x for x in Cndr_data if x[4] == 0.]), False)
    Cn1 = Cn0 + Cn_beta*b0*np.pi/180.
    if plot:
        x = b0
        y = [Cndr_p, Cn1 + Cn_dr*dr]
        color = ['0.0']*2
        ls = ['--']*2
        tag = ['data', 'not']
        marker = 'v'
        return [x, y, color, ls, tag, marker]
    return Cn_dr


def create_database():
    data = np.zeros((N_alpha*N_other_a + N_beta*N_other_b, 14))
    params = np.zeros(8)
    zz = 0
    #len(alpha_range) 1a
    for a in alpha_range:
        params[0] = a
        data[zz, :] = generate_data(params)
        zz += 1
    params[0] = 0.
    #len(beta_range) 1b
```

```
        for b in beta_range:
            params[1] = b
            data[zz, :] = generate_data(params)
            zz += 1
        params[1] = 0.
        #len(de_range)*len(a_range) len(de_range)*1a
        for e in de_range:
            params[2] = e
            for a in alpha_range:
                params[0] = a
                data[zz, :] = generate_data(params)
                zz += 1
        params[2] = 0.
        params[0] = 0.
        for da in da_range:
            params[3] = da
            #len(beta_range)
            for b in beta_range:
                params[1] = b
                data[zz, :] = generate_data(params)
                zz += 1
            params[1] = 0.
            #len(alpha_range)
            for a in alpha_range:
                params[0] = a
                data[zz, :] = generate_data(params)
                zz += 1
            params[0] = 0.
        params[3] = 0.
        #len(beta_range)
        for dr in dr_range:
            params[4] = dr
            for b in beta_range:
                params[1] = b
                data[zz, :] = generate_data(params)
                zz += 1
        params[1] = 0.
        params[4] = 0.
        #len(p_range)*(len(alpha_range) + len(beta_range))
        for p in p_range:
            params[5] = p
            for a in alpha_range:
                params[0] = a
                data[zz, :] = generate_data(params)
                zz += 1
            params[0] = 0.
            for b in beta_range:
                params[1] = b
                data[zz, :] = generate_data(params)
                zz += 1
            params[1] = 0.
        params[5] = 0.
        for q in q_range:
            params[6] = q
            for a in alpha_range:
                params[0] = a
```

```
                data[zz, :] = generate_data(params)
                zz += 1
        params[6] = 0.
        #len(r_range)*(len(alpha_range) + len(beta_range))
        for r in r_range:
            params[7] = r
            for a in alpha_range:
                params[0] = a
                data[zz, :] = generate_data(params)
                zz += 1
            params[0] = 0.
            for b in beta_range:
                params[1] = b
                data[zz, :] = generate_data(params)
                zz += 1
            params[1] = 0.
        params[7] = 0.
        return data

def find_model(database):
    plot = False
    df = pd.DataFrame(database,
                      columns = ['Alpha','Beta','d_e', 'd_a', 'd_r', 'p', 'q', 'r',
                                 'CD', 'CS', 'CL', 'Cl', 'Cm', 'Cn'])
    CLalpha_data = df.loc[df['Beta'] + df['d_e'] + df['d_a'] + df['d_r'] +
                          df['p'] + df['q'] + df['r'] == 0].to_numpy()
    CL_0, CL_alpha = _CL0_CLalpha(CLalpha_data, plot)
    CLde_data = df.loc[df['Beta'] + df['d_a'] + df['d_r'] +
                       df['p'] + df['q'] + df['r'] == 0].to_numpy()
    CL_de = _CL_de(CLde_data, plot)
    CLqbar_data = df.loc[df['Beta'] + df['d_e'] + df['d_a'] + df['d_r'] +
                         df['p'] + df['r'] == 0].to_numpy()
    CL_qbar = _CL_qbar(CLqbar_data, plot)
    CSbeta_data = df.loc[((df['Alpha'] + df['d_e'] + df['d_a'] + df['d_r'] +
                           df['p'] + df['q'] + df['r'] == 0) &
                          (df['Alpha'] == 0.))].sort_values(by=['Beta']).to_numpy()
    CS_0, CS_beta = _CS_beta(CSbeta_data, plot)
    CSda_data = df.loc[((df['Alpha'] + df['d_e'] + df['d_r'] +
                         df['p'] + df['q'] + df['r'] == 0) &
                        (df['Alpha'] == 0.))].sort_values(by=['Beta']).to_numpy()
    CS_da = _CS_da(CSda_data, plot)
    CSdr_data = df.loc[((df['Alpha'] + df['d_e'] + df['d_a'] +
                         df['p'] + df['q'] + df['r'] == 0) &
                        (df['Alpha'] == 0.))].sort_values(by=['Beta']).to_numpy()
    CS_dr = _CS_dr(CSdr_data, plot)
    CSr_data = df.loc[((df['Beta'] + df['d_e'] + df['d_a'] +
                        df['p'] + df['q'] + df['d_r'] == 0))].to_numpy()
    CS_rbar = _CS_rbar(CSr_data, plot)
    CSp_data = df.loc[((df['Beta'] + df['d_e'] + df['d_a'] + df['d_r'] +
                        df['q'] + df['r'] == 0))].to_numpy()
    CS_pbar, CS_Lpbar = _CS_pbar(CSp_data, plot, skip_mask=True)
    CDde_data = df.loc[((df['Beta'] + df['p'] + df['d_a'] + df['d_r'] +
                         df['q'] + df['r'] == 0))].to_numpy()
    CD_de, CD_Lde, CD_de2 = _CD_de(CDde_data, plot)
    CD_0, CD_L, CD_L2 = _CD_polar(CLalpha_data, plot)
    CD_S2 = _CD_Spolar(CSbeta_data, plot)[2]
```

```
    CD_qbar, CD_Lqbar, CD_L2qbar = _CD_qbar(CLqbar_data, plot)
    CDp_data = df.loc[((df['Alpha'] + df['d_e'] + df['d_a'] + df['d_r'] +
                        df['q'] + df['r'] == 0) &
                       (df['Alpha'] == 0.))].to_numpy()
    CDr_data = df.loc[((df['Alpha'] + df['d_e'] + df['d_a'] + df['d_r'] +
                        df['q'] + df['p'] == 0) &
                       (df['Alpha'] == 0.))].to_numpy()
    CD_pbar, CD_Spbar = _CD_pbar(CDp_data, plot, skip_mask=True)
    CD_rbar, CD_Srbar = _CD_rbar(CDr_data, plot)
    CD_da, CD_Sda = _CD_da(CSda_data, plot)
    CD_dr, CD_Sdr = _CD_dr(CSdr_data, plot)
    Cl_0, Cl_beta = _Cl_beta(CSbeta_data, plot)
    Cl_pbar = _Cl_pbar(CSp_data, plot)
    Cl_rbar, Cl_Lrbar = _Cl_rbar(CSr_data, plot)
    Cl_da = _Cl_da(CSda_data, plot)
    Cl_dr = _Cl_dr(CSdr_data, plot)
    Cm_0, Cm_alpha = _Cm0_Cmalpha(CLalpha_data, plot)
    Cm_qbar = _Cm_qbar(CLqbar_data, plot)
    Cm_de = _Cm_de(CLde_data, plot)
    Cn_0, Cn_beta = _Cn_beta(CSbeta_data, plot)
    Cn_pbar, Cn_Lpbar = _Cn_pbar(CSp_data, plot, skip_mask=True)
    Cn_rbar = _Cn_rbar(CSr_data, plot)
    Cnda_data = df.loc[((df['Beta'] + df['p'] + df['d_e'] + df['d_r'] +
                         df['q'] + df['r'] == 0))].to_numpy()
    Cn_da, Cn_Lda = _Cn_da(Cnda_data, plot)
    Cn_dr = _Cn_dr(CSdr_data, plot)
    coeff_database = {"CL": {"CL_0": CL_0,
                             "CL_alpha": CL_alpha,
                             "CL_qbar": CL_qbar,
                             "CL_de": CL_de},
                      "CS": {"CS_beta": CS_beta,
                             "CS_pbar": CS_pbar,
                             "CS_Lpbar": CS_Lpbar,
                             "CS_rbar": CS_rbar,
                             "CS_da": CS_da,
                             "CS_dr": CS_dr},
                      "CD": {"CD_0": CD_0,
                             "CD_L": CD_L,
                             "CD_L2": CD_L2,
                             "CD_S2": CD_S2,
                             "CD_Spbar": CD_Spbar,
                             "CD_qbar": CD_qbar,
                             "CD_Lqbar": CD_Lqbar,
                             "CD_L2qbar": CD_L2qbar,
                             "CD_Srbar": CD_Srbar,
                             "CD_de": CD_de,
                             "CD_Lde": CD_Lde,
                             "CD_de2": CD_de2,
                             "CD_Sda": CD_Sda,
                             "CD_Sdr": CD_Sdr},
                      "Cell": {"Cl_beta": Cl_beta,
                               "Cl_pbar": Cl_pbar,
                               "Cl_rbar": Cl_rbar,
                               "Cl_Lrbar": Cl_Lrbar,
                               "Cl_da": Cl_da,
                               "Cl_dr": Cl_dr},
```

```
                        "Cm": {"Cm_0": Cm_0,
                               "Cm_alpha": Cm_alpha,
                               "Cm_qbar": Cm_qbar,
                               "Cm_de": Cm_de},
                        "Cn": {"Cn_beta": Cn_beta,
                               "Cn_pbar": Cn_pbar,
                               "Cn_Lpbar": Cn_Lpbar,
                               "Cn_rbar": Cn_rbar,
                               "Cn_da": Cn_da,
                               "Cn_Lda": Cn_Lda,
                               "Cn_dr": Cn_dr}}
    return coeff_database

c_w = 11.46
b_w = 31.92
V = 222.5211
if __name__ == "__main__":
    plt.close('all')
    nasa = True
    save = True
    path_to_Ndb_file = './nasa_database.csv'
    path_to_Mdb_file = './f16_database.csv'
    Nfile_exists = exists(path_to_Ndb_file)
    Mfile_exists = exists(path_to_Mdb_file)
    alpha_range = np.arange(-10., 11., 5.)
    N_alpha = len(alpha_range)
    beta_range = np.arange(-6., 7., 2.)
    N_beta = len(beta_range)
    da_range = np.array([-20., 20.])
    dr_range = np.array([-30., 30.])
    de_range = np.array([-10., 10.])
    p_range = np.array([-90., 90.])*np.pi/180.
    q_range = np.array([-30., 30.])*np.pi/180.
    r_range = np.array([-30., 30.])*np.pi/180.
    N_other_a = 1 + len(de_range) + len(p_range) + len(q_range) + len(r_range) +\
                len(da_range)
    N_other_b = 1 + len(da_range) + len(p_range) + len(r_range) + len(dr_range)
    input_file = "F16_input.json"
    my_scene = mx.Scene(input_file)
    forces_options = {'body_frame': True,
                      'stab_frame': False,
                      'wind_frame': True,
                      'dimensional': False,
                      'verbose': False}
    if not Nfile_exists:
        nasa = True
        case = wind.F16_windtunnel()
        N_database = np.unique(create_database(), axis=0)
        np.savetxt(path_to_Ndb_file, N_database, delimiter=',')
    if not Mfile_exists:
        nasa = False
        M_database = np.unique(create_database(), axis=0)
        np.savetxt(path_to_Mdb_file, M_database, delimiter=',')
    if Mfile_exists*Nfile_exists:
        M_database = np.genfromtxt(path_to_Mdb_file, delimiter=',')
        N_database = np.genfromtxt(path_to_Ndb_file, delimiter=',')
```

```
    if nasa:
        N_coeff_data = find_model(N_database)
        with open("nasa_model.json", "w") as outfile:
            json.dump(N_coeff_data, outfile, indent=4)
    M_coeff_data = find_model(M_database)
    with open("f16_model.json", "w") as outfile:
        json.dump(M_coeff_data, outfile, indent=4)
```

BIRE Aerodynamic Model

```
import numpy as np
import f16_model
import matplotlib.pyplot as plt
import machupX as mx
import pandas as pd
from os.path import exists, isdir
import scipy.optimize as optimize
import json
from os import mkdir, remove

def remove_outliers(data, m=2.):
    d = np.abs(data - np.median(data))
    mdev = np.median(d)
    s = d/mdev if mdev else 0.
    return s < m

def create_inputs(inp_dir, d_B):
    rotation_angle = str(int(d_B))

    f_inp = open(inp_dir + 'BIRE_input.json',)
    inp_data = json.load(f_inp)

    f_air = open(inp_dir + 'BIRE_airplane.json',)
    air_data = json.load(f_air)

    bire_left = d_B
    bire_right = -d_B
    air_data["wings"]["BIRE_left"]["dihedral"] = bire_left
    air_data["wings"]["BIRE_right"]["dihedral"] = bire_right

    new_air_fn = inp_dir + 'BIRE_airplane_dB_' + rotation_angle + '.json'
    with open(new_air_fn, 'w') as fp:
        json.dump(air_data, fp, indent=5)

    inp_data["scene"]["aircraft"]["BIRE"]["file"] = new_air_fn
    new_inp_fn = inp_dir + 'BIRE_input_dB_' + rotation_angle + '.json'
    with open(new_inp_fn, 'w') as fp:
        json.dump(inp_data, fp, indent=5)
    return new_inp_fn

def bire_case(params, inp_dir, scene=None):
    [alpha, beta, d_e, d_a, d_B, p, q, r] = params
    rotation_angle = str(int(d_B))
    forces_options = {'body_frame': True,
                      'stab_frame': False,
                      'wind_frame': True,
                      'dimensional': False,
                      'verbose': False}
    try:
        f = open(inp_dir + 'BIRE_input_dB_' + rotation_angle + '.json',)
    except FileNotFoundError:
        create_inputs(inp_dir, d_B)
    if scene is None:
        input_file = inp_dir + 'BIRE_input_dB_' + rotation_angle + '.json'
```

```
        BIRE_scene = mx.Scene(input_file)
    else:
        BIRE_scene = scene
    rates = [p, q, r]
    BIRE_scene.set_aircraft_state(state={"alpha": alpha,
                                         "beta": beta,
                                         "angular_rates": rates,
                                         "velocity": 222.5211})
    BIRE_scene.set_aircraft_control_state(control_state={"elevator": d_e,
                                                         "aileron": d_a})
    x = BIRE_scene.solve_forces(**forces_options)["BIRE"]["total"]
    fm = [x['CD'], x['CS'], x['CL'], x['Cl'], x['Cm'], x['Cn']]
    return (*params, *fm)

def _plot_data_fit(mean, coeff_data, coeff_delta, model, params, range_1p, ylabel,
                   baseline_coeff, scale=1., **kwargs):
    fig, ax = plt.subplots()
    dB_plot = np.arange(-180., 185., 1.)*np.pi/180.
    model_plot = scale*(params[0]*np.sin(params[1]*dB_plot + params[2]) +
                        params[3] + coeff_delta)
    ax.scatter(dB_rad*180./np.pi, scale*(coeff_data + coeff_delta), facecolor='none',
               edgecolor='k', label='BIRE Coefficient')
    ax.plot(dB_plot*180/np.pi, model_plot, label='BIRE Fit', color='k')
    ax.axhline((baseline_coeff + coeff_delta)*scale, label='Baseline Coefficient',
               color='0.5', linestyle='--')
    ax.set_xlabel(r'\textbf{BIRE Rotation, }\boldmath$\delta_B$\textbf{ [deg]}',
                  fontsize=14)
    ax.set_ylabel(r'\boldmath$' + ylabel[5:], fontsize=14)
    loc = kwargs.get('loc', 'upper right')
    handles, labels = ax.get_legend_handles_labels()
    # sort both labels and handles by labels
    labels, handles = zip(*sorted(zip(labels, handles), key=lambda t: t[0]))
    order = [0, 1, 2]
    ax.legend([handles[idx] for idx in order], [labels[idx] for idx in order],
              loc=loc, fontsize=14)
    xlims = (-190, 190)
    dx = {"major": 45., "minor": 45./4.}
    ylims = kwargs.get('y_lim', (model_plot.min()*0.7, model_plot.max()*1.3))
    dy = kwargs.get('dy', {'major': (ylims[1] - ylims[0])/5,
                           'minor': (ylims[1] - ylims[0])/20})
    ax = pretty_plot(ax, xlims, ylims, dx, dy)
    ax.grid()
    plt.tight_layout()
    return fig

def _CL_beta(CLbeta_data):
    betas = CLbeta_data[:, 1]*np.pi/180.
    CL = CLbeta_data[:, 10]
    mask = remove_outliers(CL)
    [CL_beta, CL0] = np.polyfit(betas[mask], CL[mask], 1)
    return CL_beta

def _CL_pbar(CLpbar_data):
    CL1 = np.array([x[10] for x in CLpbar_data if x[5] == 0.])
    CLp_p = np.array([x[10] for x in CLpbar_data if x[5] == 90.*np.pi/180.])
    CLp_m = np.array([x[10] for x in CLpbar_data if x[5] == -90.*np.pi/180.])
```

```
    DCLpbar_p = (CLp_p - CL1)/(np.deg2rad(90.)*b_w/(2.*V))
    DCLpbar_m = (CLp_m - CL1)/(np.deg2rad(-90.)*b_w/(2.*V))
    mask = remove_outliers(DCLpbar_p)*remove_outliers(DCLpbar_m)
    CL_pbar = np.average(np.vstack((DCLpbar_p[mask], DCLpbar_m[mask])))
    return CL_pbar

def _CL_rbar(CLrbar_data):
    CL1 = np.array([x[10] for x in CLrbar_data if x[7] == 0.])
    CLr_p = np.array([x[10] for x in CLrbar_data if x[7] == 30.*np.pi/180.])
    CLr_m = np.array([x[10] for x in CLrbar_data if x[7] == -30.*np.pi/180.])
    DCLrbar_p = (CLr_p - CL1)/(np.deg2rad(30.)*b_w/(2.*V))
    DCLrbar_m = (CLr_m - CL1)/(np.deg2rad(-30.)*b_w/(2.*V))
    mask = remove_outliers(DCLrbar_m)*remove_outliers(DCLrbar_p)
    CL_rbar = np.average(np.vstack((DCLrbar_p[mask], DCLrbar_m[mask])))
    return CL_rbar

def _CL_da(CLda_data):
    CL1 = np.array([x[10] for x in CLda_data if x[3] == 0.])
    CLda_p = np.array([x[10] for x in CLda_data if x[3] == 20.])
    DCLda_p = (CLda_p - CL1)/np.deg2rad(20.)
    mask = remove_outliers(DCLda_p)
    CL_da = np.average(DCLda_p[mask])
    return CL_da

def CL_models(baseline_coeffs, plot=True):
    weight_CL0 = (abs(dB_range) > 135)*(abs(dB_range) < 45)
    modelCL0 = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) + np.average(CL0_dB)
    errorCL0 = lambda x : modelCL0(x) - CL0_dB
    params_CL0 = np.append(optimize.leastsq(errorCL0, [0.2])[0],
                           [2., np.pi/2., np.average(CL0_dB)])

    weight_CLalpha = [True]*N_dB
    weight_CLalpha[13] = False
    weight_CLalpha[59] = False
    weight_CLalpha[23] = False
    weight_CLalpha[49] = False
    modelCLalpha = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                              np.average(CLalpha_dB[weight_CLalpha])
    errorCLalpha = lambda x : (x[0]*np.sin(2.*dB_rad[weight_CLalpha] + np.pi/2.) +
                              np.average(CLalpha_dB[weight_CLalpha]) -
                              CLalpha_dB[weight_CLalpha])
    params_CLalpha = np.append(optimize.leastsq(errorCLalpha, [0.2])[0],
                               [2., np.pi/2., np.average(CLalpha_dB[weight_CLalpha])])

    modelCLbeta = lambda x : x[0]*np.sin(2.*dB_rad)
    errorCLbeta = lambda x : (x[0]*np.sin(2.*dB_rad) - CLbeta_dB)
    params_CLbeta = np.append(optimize.leastsq(errorCLbeta, [0.6])[0], [2., 0., 0.])

    modelCLpbar = lambda x: 0*dB_rad
    params_CLpbar = [0.]*4

    weight_CLqbar = [True]*N_dB
    modelCLqbar = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                             np.average(CLqbar_dB[weight_CLqbar])
    errorCLqbar = lambda x : (x[0]*np.sin(2.*dB_rad[weight_CLqbar] + np.pi/2.) +
                             np.average(CLqbar_dB[weight_CLqbar]) -
```

```
                              CLqbar_dB[weight_CLqbar])
    params_CLqbar = np.append(optimize.leastsq(errorCLqbar, [2.])[0],
                               [2., np.pi/2.,
                                np.average(CLqbar_dB[weight_CLqbar])])

    modelCLrbar = lambda x : x[0]*np.sin(2.*dB_rad)
    errorCLrbar = lambda x : (x[0]*np.sin(2.*dB_rad) - CLrbar_dB)
    params_CLrbar = np.append(optimize.leastsq(errorCLrbar, [1.0])[0], [2., 0., 0.])

    modelCLda = lambda x : 0.*dB_rad + np.average(CLda_dB)
    params_CLda = [0.]*3 + [np.average(CLda_dB)]

    modelCLde = lambda x : x[0]*np.sin(1.*dB_rad + np.pi/2.) + 0
    errorCLde = lambda x : (x[0]*np.sin(1.*dB_rad + np.pi/2.) + 0. - CLde_dB)
    params_CLde = np.append(optimize.leastsq(errorCLde, [2.])[0], [1., np.pi/2., 0.])

    models_dict["CL"]["CL_0"] = {key : coeff for key,coeff in
                                 zip(model_coeff_keys, params_CL0)}
    models_dict["CL"]["CL_alpha"] = {key : coeff for key,coeff in
                                     zip(model_coeff_keys, params_CLalpha)}
    models_dict["CL"]["CL_beta"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_CLbeta)}
    models_dict["CL"]["CL_pbar"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_CLpbar)}
    models_dict["CL"]["CL_qbar"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_CLqbar)}
    models_dict["CL"]["CL_rbar"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_CLrbar)}
    models_dict["CL"]["CL_da"] = {key : coeff for key,coeff in
                                  zip(model_coeff_keys, params_CLda)}
    models_dict["CL"]["CL_de"] = {key : coeff for key,coeff in
                                  zip(model_coeff_keys, params_CLde)}

def _CS_alpha(CSalpha_data):
    alphas = CSalpha_data[:, 0]*np.pi/180.
    CS = CSalpha_data[:, 9]
    [CS_alpha, CS0] = np.polyfit(alphas, CS, 1)
    return CS_alpha

def _CS_qbar(CSqbar_data):
    CS1 = np.array([x[9] for x in CSqbar_data if x[6] == 0.])
    CSq_p = np.array([x[9] for x in CSqbar_data if x[6] == 30.*np.pi/180.])
    CSq_m = np.array([x[9] for x in CSqbar_data if x[6] == -30.*np.pi/180.])
    DCSqbar_p = (CSq_p - CS1)/(np.deg2rad(30.)*c_w/(2.*V))
    DCSqbar_m = (CSq_m - CS1)/(np.deg2rad(-30.)*c_w/(2.*V))
    CS_qbar = np.average(np.vstack((DCSqbar_p, DCSqbar_m)))
    return CS_qbar

def _CS_de(CSde_data):
    CS1 = np.array([x[9] for x in CSde_data if x[2] == 0.])
    CSde_p = np.array([x[9] for x in CSde_data if x[2] == 10.])
    CSde_m = np.array([x[9] for x in CSde_data if x[2] == -10.])
    DCSde_p = (CSde_p - CS1)/np.deg2rad(10.)
    DCSde_m = (CSde_m - CS1)/np.deg2rad(-10.)
    CS_de = np.average(np.vstack((DCSde_p, DCSde_m)))
    return CS_de
```

```
def CS_models(baseline_coeffs, plot=True):
    modelCS0 = lambda x : x[0]*np.sin(2.*dB_rad)
    errorCS0 = lambda x : (x[0]*np.sin(2.*dB_rad) - CS0_dB)
    params_CS0 = np.append(optimize.leastsq(errorCS0, [-0.01])[0], [2., 0., 0.])


    modelCSalpha = lambda x : x[0]*np.sin(2.*dB_rad)
    errorCSalpha = lambda x : (x[0]*np.sin(2.*dB_rad) - CSalpha_dB)
    params_CSalpha = np.append(optimize.leastsq(errorCSalpha, [0.2])[0], [2., 0., 0.])

    modelCSbeta = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                             np.average(CSbeta_dB)
    errorCSbeta = lambda x : (x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                             np.average(CSbeta_dB) - CSbeta_dB)
    params_CSbeta = np.append(optimize.leastsq(errorCSbeta, [0.6])[0],
                              [2., np.pi/2., np.average(CSbeta_dB)])

    modelCSpbar = lambda x : 0.*dB_rad + np.average(CSpbar_dB)
    params_CSpbar = [0.]*3 + [np.average(CSpbar_dB)]

    modelCSLpbar = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                              np.average(CSLpbar_dB)
    errorCSLpbar = lambda x : modelCSLpbar(x) - CSLpbar_dB
    params_CSLpbar = np.append(optimize.leastsq(errorCSLpbar, [0.05])[0],
                               [2., np.pi/2., np.average(CSLpbar_dB)])

    modelCSqbar = lambda x : x[0]*np.sin(2.*dB_rad)
    errorCSqbar = lambda x : (x[0]*np.sin(2.*dB_rad) - CSqbar_dB)
    params_CSqbar = np.append(optimize.leastsq(errorCSqbar, [1.6])[0], [2., 0., 0.])

    weight_CSrbar = [True]*N_dB
    weight_CSrbar[6:9] = [False]*3
    weight_CSrbar[10:13] = [False]*3
    weight_CSrbar[24:27] = [False]*3
    weight_CSrbar[28:31] = [False]*3
    modelCSrbar = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) + np.average(CSrbar_dB)
    errorCSrbar = lambda x : (x[0]*np.sin(2.*dB_rad[weight_CSrbar] + np.pi/2.) +
                              np.average(CSrbar_dB) - CSrbar_dB[weight_CSrbar])
    params_CSrbar = np.append(optimize.leastsq(errorCSrbar, [-2.])[0],
                               [2., np.pi/2., np.average(CSrbar_dB)])

    weight_CSda = abs(CSda_dB) < 0.01
    modelCSda = lambda x : x[0]*np.sin(2.*dB_rad[weight_CSda] + np.pi/2.) +
                           np.average(CSda_dB[weight_CSda])
    errorCSda = lambda x : (x[0]*np.sin(2.*dB_rad[weight_CSda] + np.pi/2.) +
                           np.average(CSda_dB[weight_CSda]) - CSda_dB[weight_CSda])
    params_CSda = np.append(optimize.leastsq(errorCSda, [0.6])[0],
                                             [2., np.pi/2.,
                                              np.average(CSda_dB[weight_CSda])])
    # modelCSda = lambda x : 0.*dB_rad +

    modelCSde = lambda x : x[0]*np.sin(1.*dB_rad)
    errorCSde = lambda x : (x[0]*np.sin(1.*dB_rad) - CSde_dB)
    params_CSde = np.append(optimize.leastsq(errorCSde, [2.])[0], [1., 0., 0.])

```

```
    models_dict["CS"]["CS_0"] = {key : coeff for key,coeff in
                                 zip(model_coeff_keys, params_CS0)}
    models_dict["CS"]["CS_alpha"] = {key : coeff for key,coeff in
                                     zip(model_coeff_keys, params_CSalpha)}
    models_dict["CS"]["CS_beta"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_CSbeta)}
    models_dict["CS"]["CS_pbar"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_CSpbar)}
    models_dict["CS"]["CS_Lpbar"] = {key : coeff for key,coeff in
                                     zip(model_coeff_keys, params_CSLpbar)}
    models_dict["CS"]["CS_qbar"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_CSqbar)}
    models_dict["CS"]["CS_rbar"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_CSrbar)}
    models_dict["CS"]["CS_da"] = {key : coeff for key,coeff in
                                  zip(model_coeff_keys, params_CSda)}
    models_dict["CS"]["CS_de"] = {key : coeff for key,coeff in
                                  zip(model_coeff_keys, params_CSde)}

def _CD_pbar(CDpbar_data):
    CD1 = np.array([x[8] for x in CDpbar_data if x[5] == 0.])
    CDp_p = np.array([x[8] for x in CDpbar_data if x[5] == 90.*np.pi/180.])
    CDp_m = np.array([x[8] for x in CDpbar_data if x[5] == -90.*np.pi/180.])
    DCDpbar_p = (CDp_p - CD1)/(np.deg2rad(90.)*b_w/(2.*V))
    DCDpbar_m = (CDp_m - CD1)/(np.deg2rad(-90.)*b_w/(2.*V))
    CD_pbar = np.average(np.vstack((DCDpbar_p, DCDpbar_m)))
    return CD_pbar

def _CD_rbar(CDrbar_data):
    CD1 = np.array([x[8] for x in CDrbar_data if x[7] == 0.])
    CDr_p = np.array([x[8] for x in CDrbar_data if x[7] == 30.*np.pi/180.])
    CDr_m = np.array([x[8] for x in CDrbar_data if x[7] == -30.*np.pi/180.])
    DCDrbar_p = (CDr_p - CD1)/(np.deg2rad(30.)*b_w/(2.*V))
    DCDrbar_m = (CDr_m - CD1)/(np.deg2rad(-30.)*b_w/(2.*V))
    CD_rbar = np.average(np.vstack((DCDrbar_p, DCDrbar_m)))
    return CD_rbar

def _CD_da(CDda_data):
    CD1 = np.array([x[8] for x in CDda_data if x[3] == 0.])
    CDda_p = np.array([x[8] for x in CDda_data if x[3] == 20.])
    DCDda_p = (CDda_p - CD1)/np.deg2rad(20.)
    CD_da = np.average(DCDda_p)
    return CD_da

def CD_models(baseline_coeffs, plot=True):
    weight_CD0 = [True]*N_dB
    modelCD0 = lambda x : 0.*dB_rad + np.average(CD0_dB[weight_CD0])
    errorCD0 = lambda x : x[0]*np.sin(2.*dB_rad[weight_CD0] + np.pi/2.) +
                          np.average(CD0_dB[weight_CD0]) - CD0_dB[weight_CD0]
    params_CD0 = [0.]*3 + [np.average(CD0_dB[weight_CD0])]

    weight_CDL = [True]*N_dB
    weight_CDL[13] = False
    weight_CDL[59] = False
    weight_CDL[23] = False
    weight_CDL[49] = False
```

```
    modelCDL = lambda x : x[0]*np.sin(1.*dB_rad + np.pi/2.) +
                          np.average(CDL_dB[weight_CDL])
    errorCDL = lambda x : (x[0]*np.sin(1.*dB_rad[weight_CDL] + np.pi/2.) +
                          np.average(CDL_dB[weight_CDL]) - CDL_dB[weight_CDL])
    params_CDL = [0.]*3 + [np.average(CDL_dB[weight_CDL])]

    weight_CDL2 = [True]*N_dB
    weight_CDL2[11:26] = [False]*15
    weight_CDL2[47:62] = [False]*15
    modelCDL2 = lambda x : x[0]*np.sin(4.*dB_rad[weight_CDL2] + np.pi/2.) +
                           np.average(CDL2_dB[weight_CDL2])
    errorCDL2 = lambda x : modelCDL2(x) - CDL2_dB[weight_CDL2]
    params_CDL2 = np.append(optimize.leastsq(errorCDL2, [0.02])[0],
                                             [4., np.pi/2.,
                                             np.average(CDL2_dB[weight_CDL2])])

    modelCDS = lambda x : x[0]*np.sin(2.*dB_rad) + np.average(CDS_dB)
    errorCDS = lambda x : modelCDS(x) - CDS_dB
    params_CDS = np.append(optimize.leastsq(errorCDS, [0.005])[0],
                                            [2., 0., np.average(CDS_dB)])

    weight_CDS2 = [True]*N_dB
    weight_CDS2[:8] = [False]*8
    weight_CDS2[-8:] = [False]*8
    weight_CDS2[31:41] = [False]*10
    modelCDS2 = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                           np.average(CDS2_dB[weight_CDS2])
    errorCDS2 = lambda x : (x[0]*np.sin(2.*dB_rad[weight_CDS2] + np.pi/2.) +
                           np.average(CDS2_dB[weight_CDS2]) -
                           CDS2_dB[weight_CDS2])
    params_CDS2 = np.append(optimize.leastsq(errorCDS2, [1.])[0],
                            [2., np.pi/2., np.average(CDS2_dB[weight_CDS2])])

    modelCDpbar = lambda x : 0.*dB_rad
    params_CDpbar = [0.]*4

    weight_CDSpbar = [True]*N_dB
    weight_CDSpbar[:8] = [False]*8
    weight_CDSpbar[-8:] = [False]*8
    weight_CDSpbar[31:41] = [False]*10
    modelCDSpbar = lambda x : 0.*np.sin(2.*dB_rad + np.pi/2.) +
                              np.average(CDSpbar_dB[weight_CDSpbar])
    errorCDSpbar = lambda x : (x[0]*np.sin(2.*dB_rad[weight_CDSpbar] + np.pi/2.) +
                              np.average(CDSpbar_dB[weight_CDSpbar]) -
                              CDSpbar_dB[weight_CDSpbar])
    params_CDSpbar = [0.]*3 + [np.average(CDSpbar_dB[weight_CDSpbar])]

    weight_CDqbar = [True]*N_dB
    modelCDqbar = lambda x : 0.*np.sin(2.*dB_rad + np.pi/2.) +
                             np.average(CDqbar_dB[weight_CDqbar])
    errorCDqbar = lambda x : (modelCDqbar(x) - CDqbar_dB)[weight_CDqbar]
    params_CDqbar = [0.]*3 + [np.average(CDqbar_dB)]

    weight_CDLqbar = [True]*N_dB
    modelCDLqbar = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                              np.average(CDLqbar_dB[weight_CDLqbar])
```

```
    errorCDLqbar = lambda x : (x[0]*np.sin(2.*dB_rad[weight_CDLqbar] + np.pi/2.) +
                              np.average(CDLqbar_dB[weight_CDLqbar]) -
                              CDLqbar_dB[weight_CDLqbar])
    params_CDLqbar = np.append(optimize.leastsq(errorCDLqbar, [0.5])[0],
                               [2., np.pi/2.,
                                np.average(CDLqbar_dB[weight_CDLqbar])])

    weight_CDL2qbar = [True]*N_dB
    modelCDL2qbar = lambda x : 0.*dB_rad + np.average(CDL2qbar_dB[weight_CDL2qbar])
    errorCDL2qbar = lambda x : modelCDL2qbar(x)[weight_CDL2qbar]
    params_CDL2qbar = [0.]*3 + [np.average(CDL2qbar_dB[weight_CDL2qbar])]

    modelCDrbar = lambda x : 0.*dB_rad
    params_CDrbar = [0.]*4

    weight_CDSrbar = [True]*N_dB
    weight_CDSrbar[:8] = [False]*8
    weight_CDSrbar[-8:] = [False]*8
    weight_CDSrbar[31:41] = [False]*10
    modelCDSrbar = lambda x : 0.*dB_rad + np.average(CDSrbar_dB[weight_CDSrbar])
    errorCDSrbar = lambda x : (x[0]*np.sin(2.*dB_rad[weight_CDSrbar] + np.pi/2.) +
                              np.average(CDSrbar_dB[weight_CDSrbar]) -
                              CDSrbar_dB[weight_CDSrbar])
    params_CDSrbar = [0.]*3 + [np.average(CDSrbar_dB[weight_CDSrbar])]

    modelCDda = lambda x : x[0]*np.sin(2.*dB_rad) + np.average(CDda_dB)
    errorCDda = lambda x : (x[0]*np.sin(2.*dB_rad) + np.average(CDda_dB) - CDda_dB)
    params_CDda = np.append(optimize.leastsq(errorCDda, [0.015])[0],
                                          [2., 0., np.average(CDda_dB)])

    weight_CDSda = abs(CDSda_dB) < 0.1
    modelCDSda = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                            np.average(CDSda_dB[weight_CDSda])
    errorCDSda = lambda x : (modelCDSda(x) - CDSda_dB)[weight_CDSda]
    params_CDSda = np.append(optimize.leastsq(errorCDSda, 0.03)[0],
                                          [2., np.pi/2.,
                                          np.average(CDSda_dB[weight_CDSda])])

    modelCDde = lambda x : x[0]*np.sin(1.*dB_rad + np.pi/2.) + np.average(CDde_dB)
    errorCDde = lambda x : modelCDde(x) - CDde_dB
    params_CDde = np.append(optimize.leastsq(errorCDde, [0.02])[0], [1., np.pi/2., np.average(CDde_dB

    modelCDLde = lambda x : x[0]*np.sin(1.*dB_rad + np.pi/2.)
    errorCDLde = lambda x : (x[0]*np.sin(1.*dB_rad + np.pi/2.) - CDLde_dB)
    params_CDLde = np.append(optimize.leastsq(errorCDLde, [0.2])[0],
                                          [1., np.pi/2., 0.])

    modelCDde2 = lambda x : x[0]*np.sin(1.*dB_rad + np.pi/2.) + np.average(CDde2_dB)
    errorCDde2 = lambda x : (x[0]*np.sin(1.*dB_rad + np.pi/2.) + np.average(CDde2_dB) -
                             CDde2_dB)
    params_CDde2 = np.append(optimize.leastsq(errorCDde2, [0.3])[0],
                                          [1., np.pi/2., np.average(CDde2_dB)])

    models_dict["CD"]["CD_0"] = {key : coeff for key,coeff in
                                 zip(model_coeff_keys, params_CD0)}
    models_dict["CD"]["CD_L"] = {key : coeff for key,coeff in
```

```
                                  zip(model_coeff_keys, params_CDL)}
    models_dict["CD"]["CD_L2"] = {key : coeff for key,coeff in
                                   zip(model_coeff_keys, params_CDL2)}
    models_dict["CD"]["CD_S"] = {key : coeff for key,coeff in
                                  zip(model_coeff_keys, params_CDS)}
    models_dict["CD"]["CD_S2"] = {key : coeff for key,coeff in
                                   zip(model_coeff_keys, params_CDS2)}
    models_dict["CD"]["CD_pbar"] = {key : coeff for key,coeff in
                                     zip(model_coeff_keys, params_CDpbar)}
    models_dict["CD"]["CD_Spbar"] = {key : coeff for key,coeff in
                                      zip(model_coeff_keys, params_CDSpbar)}
    models_dict["CD"]["CD_qbar"] = {key : coeff for key,coeff in
                                     zip(model_coeff_keys, params_CDqbar)}
    models_dict["CD"]["CD_Lqbar"] = {key : coeff for key,coeff in
                                      zip(model_coeff_keys, params_CDLqbar)}
    models_dict["CD"]["CD_L2qbar"] = {key : coeff for key,coeff in
                                       zip(model_coeff_keys, params_CDL2qbar)}
    models_dict["CD"]["CD_rbar"] = {key : coeff for key,coeff in
                                     zip(model_coeff_keys, params_CDrbar)}
    models_dict["CD"]["CD_Srbar"] = {key : coeff for key,coeff in
                                      zip(model_coeff_keys, params_CDSrbar)}
    models_dict["CD"]["CD_da"] = {key : coeff for key,coeff in
                                   zip(model_coeff_keys, params_CDda)}
    models_dict["CD"]["CD_Sda"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_CDSda)}
    models_dict["CD"]["CD_de"] = {key : coeff for key,coeff in
                                   zip(model_coeff_keys, params_CDde)}
    models_dict["CD"]["CD_Lde"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_CDLde)}
    models_dict["CD"]["CD_de2"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_CDde2)}

def _Cl_alpha(Clalpha_data):
    alphas = Clalpha_data[:, 0]*np.pi/180.
    Cl = Clalpha_data[:, 11]
    [Cl_alpha, Cl0] = np.polyfit(alphas, Cl, 1)
    return Cl_alpha

def _Cl_qbar(Clqbar_data):
    Cl1 = np.array([x[11] for x in Clqbar_data if x[6] == 0.])
    Clq_p = np.array([x[11] for x in Clqbar_data if x[6] == 30.*np.pi/180.])
    Clq_m = np.array([x[11] for x in Clqbar_data if x[6] == -30.*np.pi/180.])
    DClqbar_p = (Clq_p - Cl1)/(np.deg2rad(30.)*c_w/(2.*V))
    DClqbar_m = (Clq_m - Cl1)/(np.deg2rad(-30.)*c_w/(2.*V))
    Cl_qbar = np.average(np.vstack((DClqbar_p, DClqbar_m)))
    return Cl_qbar

def _Cl_de(Clde_data):
    Cl1 = np.array([x[11] for x in Clde_data if x[2] == 0.])
    Clde_p = np.array([x[11] for x in Clde_data if x[2] == 10.])
    Clde_m = np.array([x[11] for x in Clde_data if x[2] == -10.])
    DClde_p = (Clde_p - Cl1)/np.deg2rad(10.)
    DClde_m = (Clde_m - Cl1)/np.deg2rad(-10.)
    Cl_de = np.average(np.vstack((DClde_p, DClde_m)))
    return Cl_de
```

```
def Cl_models(baseline_coeffs, plot=True):
    modelCl0 = lambda x : x[0]*np.sin(2.*dB_rad)
    errorCl0 = lambda x : (x[0]*np.sin(2.*dB_rad) - Cl0_dB)
    params_Cl0 = np.append(optimize.leastsq(errorCl0, [0.01])[0], [2., 0., 0.])

    weight_Clalpha = [True]*N_dB
    modelClalpha = lambda x : x[0]*np.sin(4.*dB_rad)
    errorClalpha = lambda x : x[0]*np.sin(4.*dB_rad[weight_Clalpha]) -
                              Clalpha_dB[weight_Clalpha]
    params_Clalpha = np.append(optimize.leastsq(errorClalpha, [-0.04])[0],
                                                  [4., 0., 0.])

    modelClbeta = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                             np.average(Clbeta_dB)
    errorClbeta = lambda x : (x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                              np.average(Clbeta_dB) - Clbeta_dB)
    params_Clbeta = np.append(optimize.leastsq(errorClbeta, [0.04])[0],
                               [2., np.pi/2., np.average(Clbeta_dB)])

    weight_Clpbar = [True]*N_dB
    weight_Clpbar[1:3] = [False]*2
    weight_Clpbar[8:11] = [False]*3
    weight_Clpbar[16:18] = [False]*2
    weight_Clpbar[-3:-1] = [False]*2
    weight_Clpbar[-11:-8] = [False]*3
    weight_Clpbar[-18:-16] = [False]*2
    modelClpbar = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                             np.average(Clpbar_dB[weight_Clpbar])
    errorClpbar = lambda x : (x[0]*np.sin(2.*dB_rad[weight_Clpbar] + np.pi/2.) +
                             np.average(Clpbar_dB[weight_Clpbar]) -
                             Clpbar_dB[weight_Clpbar])
    params_Clpbar = np.append(optimize.leastsq(errorClpbar, [0.02])[0],
                               [2., np.pi/2., np.average(Clpbar_dB[weight_Clpbar])])

    weight_Clqbar = abs(dB_rad) < np.pi/4
    modelClqbar = lambda x : x[0]*np.sin(4.*dB_rad[weight_Clqbar])
    errorClqbar = lambda x : modelClqbar(x) - Clqbar_dB[weight_Clqbar]
    params_Clqbar = [0.]*4

    modelClrbar = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) + np.average(Clrbar_dB)
    errorClrbar = lambda x : modelClrbar(x) - Clrbar_dB
    params_Clrbar = [0.]*3 + [np.average(Clrbar_dB)]

    modelClLrbar = lambda x : x[0]*np.sin(3.*dB_rad + np.pi/2.) +
                              np.average(ClLrbar_dB)
    errorClLrbar = lambda x : modelClLrbar(x) - ClLrbar_dB
    params_ClLrbar = [0.]*3 + [np.average(ClLrbar_dB)]

    weight_Clda = np.abs(Clda_dB) < 0.2
    modelClda = lambda x : x[0]*np.sin(2.*dB_rad[weight_Clda] + np.pi/2.) +
                           np.average(Clda_dB[weight_Clda])
    errorClda = lambda x : (x[0]*np.sin(2.*dB_rad[weight_Clda] + np.pi/2.) +
                           np.average(Clda_dB[weight_Clda]) -
                           Clda_dB[weight_Clda])
    params_Clda = np.append(optimize.leastsq(errorClda, [0.03])[0],
                                              [2., np.pi/2.,
```

```
                                             np.average(Clda_dB[weight_Clda])])

    weight_Clde = [True]*N_dB
    weight_Clde[7:31] = [False]*24
    weight_Clde[42:66] = [False]*24
    modelClde = lambda x : x[0]*np.sin(dB_rad)
    errorClde = lambda x : (x[0]*np.sin(dB_rad[weight_Clde]) - Clde_dB[weight_Clde])
    params_Clde = np.append(optimize.leastsq(errorClde, [0.0005])[0], [1., 0., 0.])

    models_dict["Cell"]["Cl_0"] = {key : coeff for key,coeff in
                                   zip(model_coeff_keys, params_Cl0)}
    models_dict["Cell"]["Cl_alpha"] = {key : coeff for key,coeff in
                                       zip(model_coeff_keys, params_Clalpha)}
    models_dict["Cell"]["Cl_beta"] = {key : coeff for key,coeff in
                                      zip(model_coeff_keys, params_Clbeta)}
    models_dict["Cell"]["Cl_pbar"] = {key : coeff for key,coeff in
                                      zip(model_coeff_keys, params_Clpbar)}
    models_dict["Cell"]["Cl_qbar"] = {key : coeff for key,coeff in
                                      zip(model_coeff_keys, params_Clqbar)}
    models_dict["Cell"]["Cl_rbar"] = {key : coeff for key,coeff in
                                      zip(model_coeff_keys, params_Clrbar)}
    models_dict["Cell"]["Cl_Lrbar"] = {key : coeff for key,coeff in
                                       zip(model_coeff_keys, params_ClLrbar)}
    models_dict["Cell"]["Cl_da"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_Clda)}
    models_dict["Cell"]["Cl_de"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_Clde)}

def _Cm_beta(Cmbeta_data, plot=False, yminmax=(None, None), fn='', dB=0.):
    betas = Cmbeta_data[:, 1]*np.pi/180.
    Cm = Cmbeta_data[:, 12]
    [Cm_beta, Cm0] = np.polyfit(betas, Cm, 1)
    if plot:
        plt.figure()
        plt.scatter(betas*180/np.pi, Cm, edgecolors='k', facecolor='None', s=60,
                    label='Data')
        plt.plot(betas*180/np.pi, Cm0 + Cm_beta*betas, color='r', label='Fit')
        plt.annotate(f'$\delta_B =$ {dB:3.2f}', (-4., 0.13), fontsize=18)
        plt.annotate(r'$C_{{m,\beta}} = {0:3.2f}$'.format(Cm_beta), (-4., 0.1),
                     fontsize=18)
        plt.xlim(-6.1, 6.1)
        plt.ylim(yminmax[0], yminmax[1])
        plt.xlabel(r'$\beta$, deg')
        plt.ylabel(r'$C_m$')
        plt.legend()
        plt.tight_layout()
        plt.savefig(fn)
        plt.close()
    return Cm_beta

def _Cm_pbar(Cmpbar_data):
    Cm1 = np.array([x[12] for x in Cmpbar_data if x[5] == 0.])
    Cmp_p = np.array([x[12] for x in Cmpbar_data if x[5] == 90.*np.pi/180.])
    Cmp_m = np.array([x[12] for x in Cmpbar_data if x[5] == -90.*np.pi/180.])
    DCmpbar_p = (Cmp_p - Cm1)/(np.deg2rad(90.)*b_w/(2.*V))
    DCmpbar_m = (Cmp_m - Cm1)/(np.deg2rad(-90.)*b_w/(2.*V))
```

```
    Cm_pbar = np.average(np.vstack((DCmpbar_p, DCmpbar_m)))
    return Cm_pbar

def _Cm_rbar(Cmrbar_data):
    Cm1 = np.array([x[12] for x in Cmrbar_data if x[7] == 0.])
    Cmr_p = np.array([x[12] for x in Cmrbar_data if x[7] == 30.*np.pi/180.])
    Cmr_m = np.array([x[12] for x in Cmrbar_data if x[7] == -30.*np.pi/180.])
    DCmrbar_p = (Cmr_p - Cm1)/(np.deg2rad(30.)*b_w/(2.*V))
    DCmrbar_m = (Cmr_m - Cm1)/(np.deg2rad(-30.)*b_w/(2.*V))
    Cm_rbar = np.average(np.vstack((DCmrbar_p, DCmrbar_m)))
    return Cm_rbar

def _Cm_da(Cmda_data):
    Cm1 = np.array([x[12] for x in Cmda_data if x[3] == 0.])
    Cmda_p = np.array([x[12] for x in Cmda_data if x[3] == 20.])
    DCmda_p = (Cmda_p - Cm1)/np.deg2rad(20.)
    Cm_da = np.average(DCmda_p)
    return Cm_da

def Cm_models(baseline_coeffs, plot=True):
    weight_Cm0 = [True]*N_dB
    modelCm0 = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                          np.average(Cm0_dB[weight_Cm0])
    errorCm0 = lambda x : (x[0]*np.sin(2.*dB_rad[weight_Cm0] + np.pi/2.) +
                          np.average(Cm0_dB[weight_Cm0]) -
                          Cm0_dB[weight_Cm0])
    params_Cm0 = np.append(optimize.leastsq(errorCm0, [0.02])[0],
                                            [2., np.pi/2., np.average(Cm0_dB)])

    weight_Cmalpha = (abs(Cmalpha_dB) < 0.3)
    modelCmalpha = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                              np.average(Cmalpha_dB[weight_Cmalpha])
    errorCmalpha = lambda x : (x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                               np.average(Cmalpha_dB[weight_Cmalpha]) -
                               Cmalpha_dB)[weight_Cmalpha]
    params_Cmalpha = np.append(optimize.leastsq(errorCmalpha, [0.4])[0],
                               [2., np.pi/2.,
                               np.average(Cmalpha_dB[weight_Cmalpha])])

    modelCmbeta = lambda x : x[0]*np.sin(2.*dB_rad)
    errorCmbeta = lambda x : (x[0]*np.sin(2.*dB_rad) - Cmbeta_dB)
    params_Cmbeta = np.append(optimize.leastsq(errorCmbeta, [0.6])[0], [2., 0., 0.])

    weight_Cmpbar = np.abs(Cmpbar_dB) <= 0.02
    modelCmpbar = lambda x : x[0]*np.sin(2.*dB_rad)
    errorCmpbar = lambda x : (x[0]*np.sin(2.*dB_rad) - Cmpbar_dB)[weight_Cmpbar]
    params_Cmpbar = np.append(optimize.leastsq(errorCmpbar, [0.02])[0], [2., 0., 0.])

    weight_Cmqbar = Cmqbar_dB < 0.
    modelCmqbar = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                             np.average(Cmqbar_dB[weight_Cmqbar])
    errorCmqbar = lambda x : (x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                              np.average(Cmqbar_dB[weight_Cmqbar]) -
                              Cmqbar_dB)[weight_Cmqbar]
    params_Cmqbar = np.append(optimize.leastsq(errorCmqbar, [2.])[0],
                               [2., np.pi/2.,
```

```
                                np.average(Cmqbar_dB[weight_Cmqbar])])

    weight_Cmrbar = [True]*N_dB
    weight_Cmrbar[8] = False
    weight_Cmrbar[10] = False
    weight_Cmrbar[-9] = False
    weight_Cmrbar[-11] = False
    modelCmrbar = lambda x : x[0]*np.sin(2.*dB_rad)
    errorCmrbar = lambda x : (x[0]*np.sin(2.*dB_rad[weight_Cmrbar]) -
                             Cmrbar_dB[weight_Cmrbar])
    params_Cmrbar = np.append(optimize.leastsq(errorCmrbar, [2.])[0], [2., 0., 0.])

    modelCmda = lambda x :x[0]*np.sin(2.*dB_rad) + np.average(Cmda_dB)
    errorCmda = lambda x : (modelCmda(x) - Cmda_dB)
    params_Cmda = np.append(optimize.leastsq(errorCmda, [0.01])[0],
                                             [2., 0., np.average(Cmda_dB)])

    modelCmde = lambda x : x[0]*np.sin(1.*dB_rad + np.pi/2.)
    errorCmde = lambda x : (x[0]*np.sin(1.*dB_rad + np.pi/2.) - Cmde_dB)
    params_Cmde = np.append(optimize.leastsq(errorCmde, [-2.])[0], [1., np.pi/2., 0.])

    models_dict["Cm"]["Cm_0"] = {key : coeff for key,coeff in
                                 zip(model_coeff_keys, params_Cm0)}
    models_dict["Cm"]["Cm_alpha"] = {key : coeff for key,coeff in
                                     zip(model_coeff_keys, params_Cmalpha)}
    models_dict["Cm"]["Cm_beta"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_Cmbeta)}
    models_dict["Cm"]["Cm_pbar"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_Cmpbar)}
    models_dict["Cm"]["Cm_qbar"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_Cmqbar)}
    models_dict["Cm"]["Cm_rbar"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_Cmrbar)}
    models_dict["Cm"]["Cm_da"] = {key : coeff for key,coeff in
                                  zip(model_coeff_keys, params_Cmda)}
    models_dict["Cm"]["Cm_de"] = {key : coeff for key,coeff in
                                  zip(model_coeff_keys, params_Cmde)}

def _Cn_alpha(Cnalpha_data):
    alphas = Cnalpha_data[:, 0]*np.pi/180.
    Cn = Cnalpha_data[:, 13]
    [Cn_alpha, Cn0] = np.polyfit(alphas, Cn, 1)
    return Cn_alpha

def _Cn_qbar(Cnqbar_data):
    Cn1 = np.array([x[13] for x in Cnqbar_data if x[6] == 0.])
    Cnq_p = np.array([x[13] for x in Cnqbar_data if x[6] == 30.*np.pi/180.])
    Cnq_m = np.array([x[13] for x in Cnqbar_data if x[6] == -30.*np.pi/180.])
    DCnqbar_p = (Cnq_p - Cn1)/(np.deg2rad(30.)*c_w/(2.*V))
    DCnqbar_m = (Cnq_m - Cn1)/(np.deg2rad(-30.)*c_w/(2.*V))
    Cn_qbar = np.average(np.vstack((DCnqbar_p, DCnqbar_m)))
    return Cn_qbar

def _Cn_de(Cnde_data):
    Cn1 = np.array([x[13] for x in Cnde_data if x[2] == 0.])
    Cnde_p = np.array([x[13] for x in Cnde_data if x[2] == 10.])
```

```
    Cnde_m = np.array([x[13] for x in Cnde_data if x[2] == -10.])
    DCnde_p = (Cnde_p - Cn1)/np.deg2rad(10.)
    DCnde_m = (Cnde_m - Cn1)/np.deg2rad(-10.)
    Cn_de = np.average(np.vstack((DCnde_p, DCnde_m)))
    return Cn_de

def Cn_models(baseline_coeffs, plot=True):
    modelCn0 = lambda x : x[0]*np.sin(2.*dB_rad)
    errorCn0 = lambda x : (x[0]*np.sin(2.*dB_rad) - Cn0_dB)
    params_Cn0 = np.append(optimize.leastsq(errorCn0, [-0.01])[0], [2., 0., 0.])

    modelCnalpha = lambda x : x[0]*np.sin(2.*dB_rad)
    errorCnalpha = lambda x : (x[0]*np.sin(2.*dB_rad) - Cnalpha_dB)
    params_Cnalpha = np.append(optimize.leastsq(errorCnalpha, [-0.2])[0],
                                            [2., 0., 0.])

    modelCnbeta = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                         np.average(Cnbeta_dB)
    errorCnbeta = lambda x : (x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                          np.average(Cnbeta_dB) - Cnbeta_dB)
    params_Cnbeta = np.append(optimize.leastsq(errorCnbeta, [1.])[0],
                                           [2., np.pi/2., np.average(Cnbeta_dB)])

    modelCnpbar = lambda x : 0.*dB_rad + Cnpbar_dB[N_dB//2]
    params_Cnpbar = [0.]*3 + [Cnpbar_dB[N_dB//2]]

    weight_CnLpbar = (CnLpbar_dB < -0.1)*(CnLpbar_dB > -0.14)
    modelCnLpbar = lambda x : x[0]*np.sin(2.*dB_rad[weight_CnLpbar] + np.pi/2.) +
                          np.average(CnLpbar_dB[weight_CnLpbar])
    errorCnLpbar = lambda x : modelCnLpbar(x) - CnLpbar_dB[weight_CnLpbar]
    params_CnLpbar = np.append(optimize.leastsq(errorCnLpbar, [0.001])[0],
                                           [2., np.pi/2.,
                                           np.average(CnLpbar_dB[weight_CnLpbar])])

    modelCnqbar = lambda x : x[0]*np.sin(2.*dB_rad)
    errorCnqbar = lambda x : (x[0]*np.sin(2.*dB_rad) - Cnqbar_dB)
    params_Cnqbar = np.append(optimize.leastsq(errorCnqbar, [1.6])[0], [2., 0., 0.])

    weight_Cnrbar = [True]*N_dB
    weight_Cnrbar[8] = False
    weight_Cnrbar[10] = False
    weight_Cnrbar[-9] = False
    weight_Cnrbar[-11] = False
    modelCnrbar = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) +
                         np.average(Cnrbar_dB[weight_Cnrbar])
    errorCnrbar = lambda x : (x[0]*np.sin(2.*dB_rad[weight_Cnrbar] + np.pi/2.) +
                         np.average(Cnrbar_dB[weight_Cnrbar]) -
                         Cnrbar_dB[weight_Cnrbar])
    params_Cnrbar = np.append(optimize.leastsq(errorCnrbar, [1.])[0],
                                           [2., np.pi/2.,
                                           np.average(Cnrbar_dB[weight_Cnrbar])])

    modelCnda = lambda x : 0.*dB_rad + np.average(Cnda_dB)
    params_Cnda = [0.]*3 + [np.average(Cnda_dB)]

    modelCnLda = lambda x : x[0]*np.sin(2.*dB_rad + np.pi/2.) + np.average(CnLda_dB)
```

```
    errorCnLda = lambda x : (x[0]*np.sin(2.*dB_rad + np.pi/2.) + np.average(CnLda_dB) - CnLda_dB)
    params_CnLda = [0.]*3 + [np.average(CnLda_dB)]
    params_CnLda = np.append(optimize.leastsq(errorCnLda, [1.])[0], [2., np.pi/2., np.average(CnLda_d

    modelCnde = lambda x : x[0]*np.sin(dB_rad)
    errorCnde = lambda x : (x[0]*np.sin(dB_rad) - Cnde_dB)
    params_Cnde = np.append(optimize.leastsq(errorCnde, [2.])[0], [1., 0., 0.])

    models_dict["Cn"]["Cn_0"] = {key : coeff for key,coeff in
                                zip(model_coeff_keys, params_Cn0)}
    models_dict["Cn"]["Cn_alpha"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_Cnalpha)}
    models_dict["Cn"]["Cn_beta"] = {key : coeff for key,coeff in
                                   zip(model_coeff_keys, params_Cnbeta)}
    models_dict["Cn"]["Cn_pbar"] = {key : coeff for key,coeff in
                                   zip(model_coeff_keys, params_Cnpbar)}
    models_dict["Cn"]["Cn_Lpbar"] = {key : coeff for key,coeff in
                                    zip(model_coeff_keys, params_CnLpbar)}
    models_dict["Cn"]["Cn_qbar"] = {key : coeff for key,coeff in
                                   zip(model_coeff_keys, params_Cnqbar)}
    models_dict["Cn"]["Cn_rbar"] = {key : coeff for key,coeff in
                                   zip(model_coeff_keys, params_Cnrbar)}
    models_dict["Cn"]["Cn_da"] = {key : coeff for key,coeff in
                                 zip(model_coeff_keys, params_Cnda)}
    models_dict["Cn"]["Cn_Lda"] = {key : coeff for key,coeff in
                                  zip(model_coeff_keys, params_CnLda)}
    models_dict["Cn"]["Cn_de"] = {key : coeff for key,coeff in
                                 zip(model_coeff_keys, params_Cnde)}

def create_database(inp_dir):
    data = np.zeros((N_dB*(N_alpha*N_other_a + N_beta*N_other_b), 14))
    params = np.zeros(8)
    zz = 0
    k = 0
    for dB in dB_range:
        params[4] = dB
        print("BIRE Angle : ", dB)
        for a in alpha_range:
            params[0] = a
            data[zz, :] = bire_case(params, inp_dir, scenes[k])
            zz += 1
        params[0] = 0.
        for b in beta_range:
            params[1] = b
            data[zz, :] = bire_case(params, inp_dir, scenes[k])
            zz += 1
        params[1] = 0.
        for e in de_range:
            params[2] = e
            for a in alpha_range:
                params[0] = a
                data[zz, :] = bire_case(params, inp_dir, scenes[k])
                zz += 1
        params[2] = 0.
        params[0] = 0.
        for da in da_range:
```

```
                params[3] = da
                for b in beta_range:
                    params[1] = b
                    data[zz, :] = bire_case(params, inp_dir, scenes[k])
                    zz += 1
                params[1] = 0.
                for a in alpha_range:
                    params[0] = a
                    data[zz, :] = bire_case(params, inp_dir, scenes[k])
                    zz += 1
            params[0] = 0.
            params[3] = 0.
            for p in p_range:
                params[5] = p
                for a in alpha_range:
                    params[0] = a
                    data[zz, :] = bire_case(params, inp_dir, scenes[k])
                    zz += 1
                params[0] = 0.
                for b in beta_range:
                    params[1] = b
                    data[zz, :] = bire_case(params, inp_dir, scenes[k])
                    zz += 1
                params[1] = 0.
            params[5] = 0.
            for q in q_range:
                params[6] = q
                for a in alpha_range:
                    params[0] = a
                    data[zz, :] = bire_case(params, inp_dir, scenes[k])
                    zz += 1
                params[0] = 0.
                for b in beta_range:
                    params[1] = b
                    data[zz, :] = bire_case(params, inp_dir, scenes[k])
                    zz += 1
                params[1] = 0.
            params[6] = 0.
            for r in r_range:
                params[7] = r
                for a in alpha_range:
                    params[0] = a
                    data[zz, :] = bire_case(params, inp_dir, scenes[k])
                    zz += 1
                params[0] = 0.
                for b in beta_range:
                    params[1] = b
                    data[zz, :] = bire_case(params, inp_dir, scenes[k])
                    zz += 1
                params[1] = 0.
            params[7] = 0.
            k += 1
        return data

if __name__ == "__main__":
    plt.close('all')
```

```
    path_to_db_file = './BIRE_database.csv'
    file_exists = exists(path_to_db_file)
    c_w = 11.32
    b_w = 30.
    V = 222.5211
    if not file_exists:
        alpha_range = np.arange(-10., 11., 5.)
        N_alpha = len(alpha_range)
        beta_range = np.arange(-6., 7., 2.)
        N_beta = len(beta_range)
        da_range = np.array([-20., 20.])
        dB_range = np.arange(-180., 185., 5.)
        N_dB = len(dB_range)
        de_range = np.array([-10., 10.])
        p_range = np.array([-90., 90.])*np.pi/180.
        q_range = np.array([-30., 30.])*np.pi/180.
        r_range = np.array([-30., 30.])*np.pi/180.
        N_other_a = 1 + len(de_range) + len(p_range) + len(q_range) + len(r_range) +
                    len(da_range)
        N_other_b = 1 + len(p_range) + len(q_range) + len(r_range) + len(da_range)
        scenes = []
        print("Making Inputs")
        for d_B in dB_range:
            print(d_B)
            input_file = "./BIRE Inputs/BIRE_input_dB_" + str(d_B) + ".json"
            input_exists = exists(input_file)
            if not input_exists:
                input_file = create_inputs("./BIRE Inputs/", d_B)
            scenes.append(mx.Scene(input_file))
        forces_options = {'body_frame': True,
                          'stab_frame': False,
                          'wind_frame': True,
                          'dimensional': False,
                          'verbose': False}
        print("Creating Database")
        database = np.unique(create_database('./BIRE Inputs/'), axis=0)
        np.savetxt(path_to_db_file, database, delimiter=',')
    else:
        dB_range = np.arange(-180., 185., 5.)
        N_dB = len(dB_range)
        database = np.genfromtxt(path_to_db_file, delimiter=',')
    df = pd.DataFrame(database, columns = ['Alpha','Beta','d_e', 'd_a', 'd_B', 'p',
                                          'q', 'r', 'CD', 'CS', 'CL', 'Cl', 'Cm',
                                          'Cn'])

    dB_rad = np.deg2rad(dB_range)

    CL0_dB = np.zeros(N_dB)
    CLalpha_dB = np.zeros(N_dB)
    CLbeta_dB = np.zeros(N_dB)
    CLpbar_dB = np.zeros(N_dB)
    CLqbar_dB = np.zeros(N_dB)
    CLrbar_dB = np.zeros(N_dB)
    CLda_dB = np.zeros(N_dB)
    CLde_dB = np.zeros(N_dB)

```

```
    CL0_delta = 0.
    CLalpha_delta = 0.
    CLbeta_delta = 0.
    CLpbar_delta = 0.
    CLqbar_delta = 0.
    CLrbar_delta = 0.
    CLda_delta = 0.
    CLde_delta = -0.1822

    CS0_dB = np.zeros(N_dB)
    CSalpha_dB = np.zeros(N_dB)
    CSbeta_dB = np.zeros(N_dB)
    CSpbar_dB = np.zeros(N_dB)
    CSLpbar_dB = np.zeros(N_dB)
    CSqbar_dB = np.zeros(N_dB)
    CSrbar_dB = np.zeros(N_dB)
    CSLrbar_dB = np.zeros(N_dB)
    CSda_dB = np.zeros(N_dB)
    CSde_dB = np.zeros(N_dB)

    CS0_delta = 0.
    CSalpha_delta = 0.
    CSbeta_delta = -0.1785
    CSpbar_delta = 0.
    CSLpbar_delta = 0.
    CSqbar_delta = 0.
    CSrbar_delta = 0.
    CSda_delta = -0.0448
    CSde_delta = 0.

    CD0_dB = np.zeros(N_dB)
    CDL_dB = np.zeros(N_dB)
    CDL2_dB = np.zeros(N_dB)
    CDS_dB = np.zeros(N_dB)
    CDS2_dB = np.zeros(N_dB)
    CDpbar_dB = np.zeros(N_dB)
    CDSpbar_dB = np.zeros(N_dB)
    CDLqbar_dB = np.zeros(N_dB)
    CDL2qbar_dB = np.zeros(N_dB)
    CDqbar_dB = np.zeros(N_dB)
    CDrbar_dB = np.zeros(N_dB)
    CDSrbar_dB = np.zeros(N_dB)
    CDda_dB = np.zeros(N_dB)
    CDSda_dB = np.zeros(N_dB)
    CDde_dB = np.zeros(N_dB)
    CDLde_dB = np.zeros(N_dB)
    CDde2_dB = np.zeros(N_dB)

    CD0_delta = 0.0154
    CDL_delta = -0.0304
    CDL2_delta = 0.0714
    CDS_delta = 0.
    CDS2_delta = 0.1118
    CDpbar_delta = 0.
    CDSpbar_delta = 0.
    CDqbar_delta = 0.
```

```
    CDLqbar_delta = 0.
    CDL2qbar_delta = 0.
    CDrbar_delta = 0.
    CDSrbar_delta = 0.
    CDda_delta = 0.
    CDSda_delta = 0.
    CDde_delta = 0.
    CDLde_delta = 0.
    CDde2_delta = 0.

    Cl0_dB = np.zeros(N_dB)
    Clalpha_dB = np.zeros(N_dB)
    Clbeta_dB = np.zeros(N_dB)
    Clpbar_dB = np.zeros(N_dB)
    Clqbar_dB = np.zeros(N_dB)
    Clrbar_dB = np.zeros(N_dB)
    ClLrbar_dB = np.zeros(N_dB)
    Clda_dB = np.zeros(N_dB)
    Clde_dB = np.zeros(N_dB)

    Cl0_delta = 0.
    Clalpha_delta = 0.
    Clbeta_delta = -0.0101
    Clpbar_delta = 0.
    Clqbar_delta = 0.
    Clrbar_delta = 0.
    ClLrbar_delta = 0.
    Clda_delta = 0.
    Clde_delta = 0.

    Cm0_dB = np.zeros(N_dB)
    Cmalpha_dB = np.zeros(N_dB)
    Cmbeta_dB = np.zeros(N_dB)
    Cmpbar_dB = np.zeros(N_dB)
    Cmqbar_dB = np.zeros(N_dB)
    Cmrbar_dB = np.zeros(N_dB)
    Cmda_dB = np.zeros(N_dB)
    Cmde_dB = np.zeros(N_dB)

    Cm0_delta = -0.0196
    Cma_delta = 0.2865
    Cmbeta_delta = 0.
    Cmpbar_delta = 0.
    Cmqbar_delta = 0.
    Cmrbar_delta = 0.
    Cmda_delta = 0.
    Cmde_delta = 0.2914

    Cn0_dB = np.zeros(N_dB)
    Cnalpha_dB = np.zeros(N_dB)
    Cnbeta_dB = np.zeros(N_dB)
    Cnpbar_dB = np.zeros(N_dB)
    CnLpbar_dB = np.zeros(N_dB)
    Cnqbar_dB = np.zeros(N_dB)
    Cnrbar_dB = np.zeros(N_dB)
    Cnda_dB = np.zeros(N_dB)
```

```
    CnLda_dB = np.zeros(N_dB)
    Cnde_dB = np.zeros(N_dB)

    Cn0_delta = 0.
    Cnalpha_delta = 0.
    Cnbeta_delta = -0.0326
    Cnpbar_delta = 0.
    CnLpbar_delta = 0.0602
    Cnqbar_delta = 0.
    Cnrbar_delta = 0.
    Cnda_delta = 0.0122
    CnLda_delta = 0.0254
    Cnde_delta = 0.0

    for i in range(N_dB):
        print(dB_range[i])
        CLalpha_data = df.loc[(df['Beta'] + df['d_e'] + df['d_a'] + df['p'] +
                               df['q'] + df['r'] == 0) &
                              (df['d_B'] == dB_range[i])].to_numpy()
        CL0_dB[i], CLalpha_dB[i] = f16_model._CL0_CLalpha(CLalpha_data, False)


        CLbeta_data = df.loc[(df['Alpha'] + df['d_e'] + df['d_a'] + df['p'] +
                              df['q'] + df['r'] == 0) & (df['d_B'] == dB_range[i]) &
                             (df['Alpha'] == 0)].to_numpy()
        CLbeta_dB[i] = _CL_beta(CLbeta_data)

        CLpbar_data = df.loc[(df['Beta'] + df['d_e'] + df['d_a'] + df['q'] +
                              df['r'] == 0) & (df['d_B'] == dB_range[i])].to_numpy()
        CLpbar_dB[i] = _CL_pbar(CLpbar_data)

        CLqbar_data = df.loc[(df['Beta'] + df['d_e'] + df['d_a'] + df['p'] +
                              df['r'] == 0) & (df['d_B'] == dB_range[i])].to_numpy()
        CLqbar_dB[i] = f16_model._CL_qbar(CLqbar_data, False)

        CLrbar_data = df.loc[(df['Beta'] + df['d_e'] + df['d_a'] + df['p'] +
                              df['q'] == 0) & (df['d_B'] == dB_range[i])].to_numpy()
        CLrbar_dB[i] = _CL_rbar(CLrbar_data)

        CLda_data = df.loc[(df['Beta'] + df['d_e'] + df['p'] + df['q'] +
                            df['r'] == 0) & (df['d_B'] == dB_range[i])].to_numpy()
        CLda_dB[i] = _CL_da(CLda_data)

        CLde_data = df.loc[(df['Beta'] + df['d_a'] + df['p'] + df['q'] +
                            df['r'] == 0) & (df['d_B'] == dB_range[i])].to_numpy()
        CDp_data = df.loc[((df['Alpha'] + df['d_e'] + df['d_a'] + df['q'] +
                            df['r'] == 0) &
                           (df['Alpha'] == 0.) &
                           (df['d_B'] == dB_range[i]))].to_numpy()
        CDr_data = df.loc[((df['Alpha'] + df['d_e'] + df['d_a'] + df['q'] +
                            df['p'] == 0) &
                           (df['Alpha'] == 0.) &
                           (df['d_B'] == dB_range[i]))].to_numpy()

        CLde_dB[i] = f16_model._CL_de(CLde_data, False)

```

```
            CS0_dB[i], CSbeta_dB[i] = f16_model._CS_beta(CLbeta_data, False)

            CSalpha_dB[i] = _CS_alpha(CLalpha_data)

            CSpbar_dB[i], CSLpbar_dB[i] = f16_model._CS_pbar(CLpbar_data, False)

            CSqbar_dB[i] = _CS_qbar(CLqbar_data)

            CSrbar_dB[i] = f16_model._CS_rbar(CLrbar_data, False)

            CSda_data = df.loc[((df['Alpha'] + df['d_e'] + df['r'] + df['q'] +
                                 df['p'] == 0) &
                                (df['Alpha'] == 0.) &
                                (df['d_B'] == dB_range[i]))].to_numpy()
            CSda_dB[i] = f16_model._CS_da(CSda_data, False, skip_mask=True)

            CSde_dB[i] = _CS_de(CLde_data)

            CD0_dB[i], CDL_dB[i], CDL2_dB[i] = f16_model._CD_polar(CLalpha_data, False)

            CDS_dB[i], CDS2_dB[i] = f16_model._CD_Spolar(CLbeta_data, False)[1:]

            CDpbar_dB[i], CDSpbar_dB[i] = f16_model._CD_pbar(CDp_data, False)

            CDqbar_dB[i], CDLqbar_dB[i], CDL2qbar_dB[i] = f16_model._CD_qbar(CLqbar_data,
                                                                             False)

            CDrbar_dB[i], CDSrbar_dB[i] = f16_model._CD_rbar(CDr_data, False)

            CDda_dB[i], CDSda_dB[i] = f16_model._CD_da(CLda_data, False)[:2]

            CDde_dB[i], CDLde_dB[i], CDde2_dB[i] = f16_model._CD_de(CLde_data, False)

            Cl0_dB[i], Clbeta_dB[i] = f16_model._Cl_beta(CLbeta_data, False)

            Clalpha_dB[i] = _Cl_alpha(CLalpha_data)

            Clpbar_dB[i] = f16_model._Cl_pbar(CLpbar_data, False)

            Clqbar_dB[i] = _Cl_qbar(CLqbar_data)

            Clrbar_dB[i], ClLrbar_dB[i] = f16_model._Cl_rbar(CLrbar_data, False)

            Clda_dB[i] = f16_model._Cl_da(CSda_data, False)

            Clde_dB[i] = _Cl_de(CLde_data)

            Cm0_dB[i], Cmalpha_dB[i] = f16_model._Cm0_Cmalpha(CLalpha_data, False,
                                                              skip_mask=False)

            Cmbeta_dB[i] = _Cm_beta(CLbeta_data)

            Cmpbar_dB[i] = _Cm_pbar(CLpbar_data)

            Cmqbar_dB[i] = f16_model._Cm_qbar(CLqbar_data, False)

```

```
            Cmrbar_dB[i] = _Cm_rbar(CLrbar_data)

            Cmda_dB[i] = _Cm_da(CLda_data)

            Cmde_dB[i] = f16_model._Cm_de(CLde_data, False)

            Cn0_dB[i], Cnbeta_dB[i] = f16_model._Cn_beta(CLbeta_data, False)

            Cnalpha_dB[i] = _Cn_alpha(CLalpha_data)

            Cnpbar_dB[i], CnLpbar_dB[i] = f16_model._Cn_pbar(CLpbar_data, False)

            Cnqbar_dB[i] = _Cn_qbar(CLqbar_data)

            Cnrbar_dB[i] = f16_model._Cn_rbar(CLrbar_data, False)

            Cnda_dB[i], CnLda_dB[i] = f16_model._Cn_da(CLda_data, False)

            Cnde_dB[i] = _Cn_de(CLde_data)
        max_alpha = 20.*np.pi/180.
        max_beta = 10.*np.pi/180.
        max_pbar = 90.*b_w/(2.*V)*np.pi/180.
        max_qbar = 30.*c_w/(2.*V)*np.pi/180.
        max_rbar = 30.*b_w/(2.*V)*np.pi/180.
        max_da = 21.5*np.pi/180.
        max_de = 25.*np.pi/180.
        CL1_data = df.loc[(df['Beta'] + df['d_e'] + df['d_a'] + df['p'] + df['q'] +
                           df['r'] == 0)].to_numpy()
        CS1_data = df.loc[(df['Alpha'] + df['d_e'] + df['d_a'] + df['p'] + df['q'] +
                           df['r'] == 0)].to_numpy()
        max_CL1 = np.max(np.abs(CL1_data[:, 10]))
        max_CS1 = np.max(np.abs(CS1_data[:, 9]))

        meanCL_1p = np.average(np.abs(database[:, 10]))*0.01
        meanCS_1p = np.average(np.abs(database[:, 9]))*0.01
        meanCD_1p = np.average(np.abs(database[:, 8]))*0.01
        meanCl_1p = np.average(np.abs(database[:, 11]))*0.01
        meanCm_1p = np.average(np.abs(database[:, 12]))*0.01
        meanCn_1p = np.average(np.abs(database[:, 13]))*0.01

        model_coeff_keys = ["A", "w", "phi", "z"]
        model_coeff_dict = {key: 0. for key in model_coeff_keys}

        models_dict = {"CL": {
                              "CL_0" : model_coeff_dict,
                              "CL_alpha" : model_coeff_dict,
                              "CL_beta" : model_coeff_dict,
                              "CL_pbar" : model_coeff_dict,
                              "CL_qbar" : model_coeff_dict,
                              "CL_rbar" : model_coeff_dict,
                              "CL_da" : model_coeff_dict,
                              "CL_de" : model_coeff_dict
                              },
                        "CS": {
                              "CS_0" : model_coeff_dict,
                              "CS_alpha" : model_coeff_dict,
```

```
                        "CS_beta" : model_coeff_dict,
                        "CS_pbar" : model_coeff_dict,
                        "CS_Lpbar" : model_coeff_dict,
                        "CS_qbar" : model_coeff_dict,
                        "CS_rbar" : model_coeff_dict,
                        "CS_da" : model_coeff_dict,
                        "CS_de" : model_coeff_dict
                        },
                  "CD": {
                        "CD_0" : model_coeff_dict,
                        "CD_L" : model_coeff_dict,
                        "CD_L2" : model_coeff_dict,
                        "CD_S" : model_coeff_dict,
                        "CD_S2" : model_coeff_dict,
                        "CD_pbar" : model_coeff_dict,
                        "CD_Spbar" : model_coeff_dict,
                        "CD_qbar" : model_coeff_dict,
                        "CD_Lqbar" : model_coeff_dict,
                        "CD_L2qbar" : model_coeff_dict,
                        "CD_rbar" : model_coeff_dict,
                        "CD_Srbar" : model_coeff_dict,
                        "CD_da" : model_coeff_dict,
                        "CD_Sda" : model_coeff_dict,
                        "CD_de" : model_coeff_dict,
                        "CD_Lde" : model_coeff_dict,
                        "CD_de2" : model_coeff_dict
                        },
                  "Cell": {
                        "Cl_0" : model_coeff_dict,
                        "Cl_alpha" : model_coeff_dict,
                        "Cl_beta" : model_coeff_dict,
                        "Cl_pbar" : model_coeff_dict,
                        "Cl_qbar" : model_coeff_dict,
                        "Cl_rbar" : model_coeff_dict,
                        "Cl_Lrbar" : model_coeff_dict,
                        "Cl_da" : model_coeff_dict,
                        "Cl_de" : model_coeff_dict
                        },
                  "Cm": {
                        "Cm_0" : model_coeff_dict,
                        "Cm_alpha" : model_coeff_dict,
                        "Cm_beta" : model_coeff_dict,
                        "Cm_pbar" : model_coeff_dict,
                        "Cm_qbar" : model_coeff_dict,
                        "Cm_rbar" : model_coeff_dict,
                        "Cm_da" : model_coeff_dict,
                        "Cm_de" : model_coeff_dict
                        },
                  "Cn": {
                        "Cn_0" : model_coeff_dict,
                        "Cn_alpha" : model_coeff_dict,
                        "Cn_beta" : model_coeff_dict,
                        "Cn_pbar" : model_coeff_dict,
                        "Cn_Lpbar" : model_coeff_dict,
                        "Cn_qbar" : model_coeff_dict,
                        "Cn_rbar" : model_coeff_dict,
```

```
                                "Cn_da" : model_coeff_dict,
                                "Cn_Lda" : model_coeff_dict,
                                "Cn_de" : model_coeff_dict
                                }
                        }

    base_coeffs_dict = json.load(open('./f16_model.json'))


    CL_models(base_coeffs_dict["CL"], plot=False)
    CS_models(base_coeffs_dict["CS"], plot=False)
    CD_models(base_coeffs_dict["CD"], plot=False)
    Cl_models(base_coeffs_dict["Cell"], plot=False)
    Cm_models(base_coeffs_dict["Cm"], plot=False)
    Cn_models(base_coeffs_dict["Cn"], plot=False)
    with open("bire_model.json", "w") as outfile:
        json.dump(models_dict, outfile, indent=4)
```

C.4 Static Trim Analysis

Thrust Modeling

```
import numpy as np
import scipy.optimize as optimize
import matplotlib.pyplot as plt
from hunsaker_atm import stdatm_english


def find_coeffs(C, T_data, rho, V):
    [a, T0, T1, T2] = C
    T = (rho/rho_0)**a*(T0 + T1*V + T2*np.square(V))
    return np.linalg.norm(T - T_data)

def a_coeff(a, T_data, rho):
    T = (rho/rho_0)**a*(T_data)
    print(a)
    print(np.linalg.norm(T - T_data))
    return np.linalg.norm(T - T_data)

T_idle = np.array([[635, 425, 690, 1010, 1330, 1700],
                   [60, 25, 345, 755, 1130, 1525],
                   [-1020, -710, -300, 350, 910, 1360],
                   [-2700, -1900, -1300, -247, 600, 1100],
                   [-3600, -1400, -595, -342, -200, 700]])
T_mil = np.array([[12680, 9150, 6313, 4040, 2470, 1400],
                  [12610, 9312, 6610, 4290, 2600, 1560],
                  [12640, 9839, 7090, 4660, 2840, 1660],
                  [12390, 10176, 7750, 5320, 3250, 1930],
                  [11680, 9848, 8050, 6100, 3800, 2310]])
T_max = np.array([[21420, 15700, 11225, 7323, 4435, 2600],
                  [22700, 16860, 12250, 8154, 5000, 2835],
                  [24240, 18910, 13760, 9285, 5700, 3215],
                  [26070, 21075, 15975, 11115, 6860, 3950],
                  [28886, 23319, 18300, 13484, 8642, 5057]])
M = np.array([0.2, 0.4, 0.6, 0.8, 1.0])
H = np.arange(0., 60000., 10000.)
rho_0 = stdatm_english(0.)[-2]
rho = np.zeros(len(H))
a = np.zeros(len(H))
V = np.zeros_like(T_idle)
for i in range(len(H)):
    rho[i], a[i] = stdatm_english(H[i])[-2:]
    for j in range(len(M)):
        V[j, i] = M[j]*a[i]
T0_i = np.zeros_like(H)
T1_i = np.zeros_like(H)
T2_i = np.zeros_like(H)
a_i = np.zeros_like(H)
for i in range(len(H)):
    [a_i[i], T0_i[i], T1_i[i], T2_i[i]] = optimize.minimize(find_coeffs,
                                            [1.]*4,
                                            args=(T_idle[:, i],
```

```
                                                        rho[i],
                                                        V[:, i])).x
T_idle_fit = np.zeros_like(T_idle)
for i in range(len(H)):
    for j in range(len(M)):
        T_idle_fit[j, i] = (rho[i]/rho_0)**a_i[i]*(T0_i[i] +
                                                    T1_i[i]*V[j, i] +
                                                    T2_i[i]*V[j, i]**2)

T0_mil = np.zeros_like(H)
T1_mil = np.zeros_like(H)
T2_mil = np.zeros_like(H)
a_mil = np.zeros_like(H)
for i in range(len(H)):
    [a_mil[i], T0_mil[i], T1_mil[i], T2_mil[i]] =
                                  optimize.minimize(find_coeffs,
                                                    [1.]*4,
                                                    args=(T_mil[:, i],
                                                          rho[i],
                                                          V[:, i])).x
T_mil_fit = np.zeros_like(T_mil)
for i in range(len(H)):
    for j in range(len(M)):
        T_mil_fit[j, i] = (rho[i]/rho_0)**a_mil[i]*(T0_mil[i] +
                                                 T1_mil[i]*V[j, i] +
                                                 T2_mil[i]*V[j, i]**2)

T0_max = np.zeros_like(H)
T1_max = np.zeros_like(H)
T2_max = np.zeros_like(H)
a_max = np.zeros_like(H)
for i in range(len(H)):
    [a_max[i], T0_max[i], T1_max[i], T2_max[i]] =
                                  optimize.minimize(find_coeffs,
                                                    [1.]*4,
                                                    args=(T_max[:, i],
                                                          rho[i],
                                                          V[:, i])).x
T_max_fit = np.zeros_like(T_max)
for i in range(len(H)):
    for j in range(len(M)):
        T_max_fit[j, i] = (rho[i]/rho_0)**a_max[i]*(T0_max[i] +
                                                 T1_max[i]*V[j, i] +
                                                 T2_max[i]*V[j, i]**2)

T0_i_fit = np.polyfit(H, T0_i, 2)
T1_i_fit = np.polyfit(H, T1_i, 2)
T2_i_fit = np.polyfit(H, T2_i, 2)
a_i_fit = np.polyfit(H, a_i, 2)
T0_mil_fit = np.polyfit(H, T0_mil, 2)
T1_mil_fit = np.polyfit(H, T1_mil, 2)
T2_mil_fit = np.polyfit(H, T2_mil, 2)
a_mil_fit = np.polyfit(H, a_mil, 2)
T0_max_fit = np.polyfit(H, T0_max, 2)
T1_max_fit = np.polyfit(H, T1_max, 2)
T2_max_fit = np.polyfit(H, T2_max, 2)
```

```
a_max_fit = np.polyfit(H, a_max, 2)
```

Trim Algorithm

```
import numpy as np
from f16_aero import F16Aero
from bire_aero import BIREAero
from stdatmos import stdatm_english
import json

class AircraftProperties:
    def __init__(self, V, H, Gamma, path='./', bire=False, **kwargs):
        if bire:
            fn = kwargs.get('filename', 'BIRE_props.json')
            prop_dict = json.load(open(path + fn))
        else:
            fn = kwargs.get('filename', 'F16_props.json')
            prop_dict = json.load(open(path + fn))
        self.S_w = prop_dict["geometry"]["S_w"]
        self.b_w = prop_dict["geometry"]["b_w"]
        self.c_w = prop_dict["geometry"]["c_w"]
        self.l_h = prop_dict["geometry"]["l_h"]
        self.RA_w = prop_dict["geometry"]["RA_w"]
        self.Lam_w = prop_dict["geometry"]["Lam_w"]
        self.RA_v = prop_dict["geometry"]["RA_v"]
        self.Lam_v = prop_dict["geometry"]["Lam_v"]
        self.RA_h = prop_dict["geometry"]["RA_h"]
        self.Lam_h = prop_dict["geometry"]["Lam_h"]
        self.W = prop_dict["inertia"]["W"]
        self.hz = prop_dict["inertia"]["h_z"]
        self.hy = prop_dict["inertia"]["h_y"]
        self.hx = prop_dict["inertia"]["h_x"]
        if bire:
            I_model = json.load(open('./bire_inertia_model.json'))
            Ixx = I_model["Ixx"]
            Iyy = I_model["Iyy"]
            Izz = I_model["Izz"]
            Ixz = I_model["Ixz"]
            Ixy = I_model["Ixy"]
            Iyz = I_model["Iyz"]
            self.I_xx = lambda dB : Ixx["A"]*np.sin(Ixx["w"]*dB + Ixx["phi"]) +
                                    Ixx["z"]
            self.I_yy = lambda dB : Iyy["A"]*np.sin(Iyy["w"]*dB + Iyy["phi"]) +
                                    Iyy["z"]
            self.I_zz = lambda dB : Izz["A"]*np.sin(Izz["w"]*dB + Izz["phi"]) +
                                    Izz["z"]
            self.I_yz = lambda dB : Iyz["A"]*np.sin(Iyz["w"]*dB + Iyz["phi"]) +
                                    Iyz["z"]
            self.I_xy = lambda dB : Ixy["A"]*np.sin(Ixy["w"]*dB + Ixy["phi"]) +
                                    Ixy["z"]
            self.I_xz = lambda dB : Ixz["A"]*np.sin(Ixz["w"]*dB + Ixz["phi"]) +
                                    Ixz["z"]
            self.dI_xx = lambda dB : np.array([0., 0., 0.,
                                               Ixx["A"]*Ixx["w"]*np.cos(Ixx["w"]*dB +
                                               Ixx["phi"])])
            self.dI_yy = lambda dB : np.array([0., 0., 0.,
                                               Iyy["A"]*Iyy["w"]*np.cos(Iyy["w"]*dB +
                                               Iyy["phi"])])
```

```
            self.dI_zz = lambda dB : np.array([0., 0., 0.,
                                              Izz["A"]*Izz["w"]*np.cos(Izz["w"]*dB +
                                              Izz["phi"])])
            self.dI_yz = lambda dB : np.array([0., 0., 0.,
                                              Iyz["A"]*Iyz["w"]*np.cos(Iyz["w"]*dB +
                                              Iyz["phi"])])
            self.dI_xy = lambda dB : np.array([0., 0., 0.,
                                              Ixy["A"]*Ixy["w"]*np.cos(Ixy["w"]*dB +
                                              Ixy["phi"])])
            self.dI_xz = lambda dB : np.array([0., 0., 0.,
                                              Ixz["A"]*Ixz["w"]*np.cos(Ixz["w"]*dB +
                                              Ixz["phi"])])
        else:
            self.Ixx = prop_dict["inertia"]["I_xx"]
            self.Ixy = prop_dict["inertia"]["I_xy"]
            self.Iyx = self.Ixy
            self.Ixz = prop_dict["inertia"]["I_xz"]
            self.Izx = self.Ixz
            self.Iyy = prop_dict["inertia"]["I_yy"]
            self.Iyz = prop_dict["inertia"]["I_yz"]
            self.Izy = self.Iyz
            self.Izz = prop_dict["inertia"]["I_zz"]
        self.g = 32.2
        dummyz, dummyT, dummyp, self.rho, self.a = stdatm_english(H)
        dummyz, dummyT, dummyp, self.rho_0, self.a_0 = stdatm_english(H)
        self.nondim_const = 0.5*self.rho*V*V*self.S_w
        self.V = V
        self.H = H
        self.Gamma = Gamma
        self.M = self.V/self.a
        self.T0_idle = lambda H: 3145 - 0.4185*H + 1.8313e-5*H**2
        self.T0_mil = lambda H: 11716 + 0.1156*H + 0.3474e-5*H**2
        self.T0_max = lambda H: 20341 + 0.1454*H + 0.9283e-5*H**2
        self.T1_idle = lambda H: -4.3491 - 4.9703e-4*H + 1.3557e-8*H**2
        self.T1_mil = lambda H: 3.5689 + 0.1409e-4*H - 0.3982e-8*H**2
        self.T1_max = lambda H: 1.9886 + 6.3926e-4*H - 2.4428e-8*H**2
        self.T2_idle = lambda H: -0.2321e-3 + 5.5629e-7*H - 2.0550e-11*H**2
        self.T2_mil = lambda H: -3.9793e-3 + 2.6931e-7*H + 0.5281e-11*H**2
        self.T2_max = lambda H: 3.5201e-3 + 0.7574e-7*H + 2.6665e-11*H**2
        self.a_idle = lambda H: 1.0104 + 2.9484e-5*H - 3.8270e-10*H**2
        self.a_mil = lambda H: 1.0148 + 3.1355e-5*H - 4.2106e-10*H**2
        self.a_max = lambda H: 1.0225 + 3.1984e-5*H - 4.3617e-10*H**2

    def calc_BIRE_inertia(self, dB):
        self.Ixx = self.I_xx(dB)
        self.Ixy = self.I_xy(dB)
        self.Ixz = self.I_xz(dB)
        self.Iyy = self.I_yy(dB)
        self.Iyz = self.I_yz(dB)
        self.Izz = self.I_zz(dB)

class TrimSolution:
    def __init__(self):
        self.FM = np.zeros(6)
        self.rates = np.zeros(3)
        self.velocity = np.zeros(3)
```

```
        self.load = 0.
        self.load_s = 0.
        self.x = np.zeros(6)
        self.orient = np.zeros(3)
        self.num_iters = 0.
        self.vehicle = "Baseline"

def climb_2_elev(u, v, w, phi, gamma, V):
    V = np.sqrt(u**2 + v**2 + w**2)
    n_1 = u*V*np.sin(gamma)
    n_2 = (v*np.sin(phi) + w*np.cos(phi))
    n_3 = np.sqrt(u*u + n_2**2 - V**2*np.sin(gamma)**2)
    d = u**2 + n_2**2
    th_plus = np.arcsin((n_1 + n_2*n_3)/d)
    th_minus = np.arcsin((n_1 - n_2*n_3)/d)
    check_plus = (V*np.sin(gamma) - u*np.sin(th_plus) - n_2*np.cos(th_plus) < 1e-8)
    check_minus = (V*np.sin(gamma) - u*np.sin(th_minus) - n_2*np.cos(th_minus) <
                   1e-8)
    if check_plus:
        return th_plus
    elif check_minus:
        return th_minus

def v_comp(alpha, beta, V):
    u = V*np.cos(alpha)*np.cos(beta)
    v = V*np.sin(beta)
    w = V*np.sin(alpha)*np.cos(beta)
    return u, v, w

def load_2_bank(n_a, Fx, W, p, q, u, v, alpha, theta, props):
    num = n_a - Fx*np.sin(alpha)/W - (q*u - p*v)*np.cos(alpha)/props.g
    denom = np.cos(theta)*np.cos(alpha)
    phi = np.arccos(num/denom)
    return phi

def load_factor(theta, phi, alpha, p, q, r, u, v, w, props):
    C1 = np.cos(theta)*np.cos(phi) + (q*u - p*v)/props.g
    C2 = np.sin(theta) - (r*v - q*w)/props.g
    n_a = C1*np.cos(alpha) + C2*np.sin(alpha)
    return n_a

def rotation_rates(phi, theta, u, w, props):
    C_num = props.g*np.sin(phi)*np.cos(theta)
    C_denom = u*np.cos(theta)*np.cos(phi) + w*np.sin(theta)
    C = C_num/C_denom
    p = -C*np.sin(theta)
    q = C*np.sin(phi)*np.cos(theta)
    r = C*np.cos(phi)*np.cos(theta)
    return p, q, r

def tgear(tau):
    if tau <= 0.77:
        P1 = 64.94*tau
    else:
        P1 = 217.38*tau - 117.38
    return P1
```

```
def thrust(tau, V, props):
    P1 = tgear(tau)
    T0_mil = props.T0_mil(props.H)
    T1_mil = props.T1_mil(props.H)
    T2_mil = props.T2_mil(props.H)
    a_mil = props.a_mil(props.H)
    C1_mil = (props.rho/props.rho_0)**a_mil
    C2_mil = T0_mil + T1_mil*V + T2_mil*V**2
    T_mil = C1_mil*C2_mil
    if P1 >= 50.:
        T0_max = props.T0_max(props.H)
        T1_max = props.T1_max(props.H)
        T2_max = props.T2_max(props.H)
        a_max = props.a_max(props.H)
        C1_max = (props.rho/props.rho_0)**a_max
        C2_max = T0_max + T1_max*V + T2_max*V**2
        T_max = C1_max*C2_max
        T = T_mil + (T_max - T_mil)*(P1 - 50.)/50.
    else:
        T0_idle = props.T0_idle(props.H)
        T1_idle = props.T1_idle(props.H)
        T2_idle = props.T2_idle(props.H)
        a_idle = props.a_idle(props.H)
        C1_idle = (props.rho/props.rho_0)**a_idle
        C2_idle = T0_idle + T1_idle*V + T2_idle*V**2
        T_idle = C1_idle*C2_idle
        T = T_idle + (T_mil - T_idle)*P1/50.
    return T

def _tau_p1(tau, Fx, theta, q, r, v, w, T, props):
    num = Fx - props.W*np.sin(theta) + (r*v - q*w)*props.W/props.g
    denom = T
    tau_p1 = tau - props.Gamma*num/denom
    return tau_p1, num

def _beta_p1(beta, Fy, phi, theta, p, r, u, w, CSb, props):
    num = Fy + props.W*np.sin(phi)*np.cos(theta) + (p*w - r*u)*props.W/props.g
    denom = props.nondim_const*CSb*np.cos(beta)
    beta_p1 = beta - props.Gamma*num/denom
    return beta_p1, num

def _alpha_p1(alpha, Fz, phi, theta, p, q, u, v, CLa, props):
    num = Fz + props.W*np.cos(phi)*np.cos(theta) + (q*u - p*v)*props.W/props.g
    denom = props.nondim_const*CLa*np.cos(alpha)
    alpha_p1 = alpha + props.Gamma*num/denom
    return alpha_p1, num

def _da_p1(da, Mx, p, q, r, Clda, props):
    num_1 = Mx - props.hz*q + props.hy*r + (props.Iyy - props.Izz)*q*r
    num_2 = props.Iyz*(q**2 - r**2) + props.Ixz*p*q - props.Ixy*p*r
    num = num_1 + num_2
    denom = props.nondim_const*props.b_w*Clda
    da_p1 = da - props.Gamma*num/denom
    return da_p1, num
```

```python
def _de_p1(de, My, p, q, r, Cmde, props):
    num_1 = My + props.hz*p - props.hx*r + (props.Izz - props.Ixx)*p*r
    num_2 = props.Ixz*(r**2 - p**2) + props.Ixy*q*r - props.Iyz*p*q
    num = num_1 + num_2
    denom = props.nondim_const*props.c_w*Cmde
    de_p1 = de - props.Gamma*num/denom
    return de_p1, num

def _dr_p1(dr, Mz, p, q, r, Cndr, props):
    num_1 = Mz - props.hy*p + props.hx*q + (props.Ixx - props.Iyy)*p*q
    num_2 = props.Ixy*(p**2 - q**2) + props.Iyz*p*r - props.Ixz*q*r
    num = num_1 + num_2
    denom = props.nondim_const*props.b_w*Cndr
    dr_p1 = dr - props.Gamma*num/denom
    return dr_p1, num

def _dB_p1(dB, Mz, p, q, r, CndB, props):
    num_1 = Mz - props.hy*p + props.hx*q + (props.Ixx - props.Iyy)*p*q
    num_2 = props.Ixy*(p**2 - q**2) + props.Iyz*p*r - props.Ixz*q*r
    num = num_1 + num_2
    denom = props.nondim_const*props.b_w*CndB
    dB_p1 = dB - props.Gamma*num/denom
    return dB_p1, num

def _f1(Fx, theta, phi, pqr, uvw, props):
    [u, v, w] = uvw
    [p, q, r] = pqr
    return Fx - props.W*np.sin(theta) + (r*v - q*w)*props.W/props.g

def _f2(Fy, theta, phi, pqr, uvw, props):
    [u, v, w] = uvw
    [p, q, r] = pqr
    return Fy + props.W*np.sin(phi)*np.cos(theta) + (p*w - r*u)*props.W/props.g

def _f3(Fz, theta, phi, pqr, uvw, props):
    [u, v, w] = uvw
    [p, q, r] = pqr
    return Fz + props.W*np.cos(phi)*np.cos(theta) + (q*u - p*v)*props.W/props.g

def _f4(Mx, theta, phi, pqr, uvw, props):
    [p, q, r] = pqr
    C1 = Mx - props.hz*q + props.hy*r + (props.Iyy - props.Izz)*q*r
    C2 = props.Iyz*(q**2 - r**2) + props.Ixz*p*q - props.Ixy*p*r
    return C1 + C2

def _f5(My, theta, phi, pqr, uvw, props):
    [p, q, r] = pqr
    C1 = My + props.hz*p - props.hx*r + (props.Izz - props.Ixx)*p*r
    C2 = props.Ixz*(r**2 - p**2) + props.Ixy*q*r - props.Iyz*p*q
    return C1 + C2

def _f6(Mz, theta, phi, pqr, uvw, props):
    [p, q, r] = pqr
    C1 = Mz - props.hy*p + props.hx*q + (props.Ixx - props.Iyy)*p*q
    C2 = props.Ixy*(p**2 - q**2) + props.Iyz*p*r - props.Ixz*q*r
    return C1 + C2
```

```

def _recalc_forces(state, phi, gamma, coeffs, props, shss):
    V = props.V
    [tau, alpha, beta, da, de, dr] = state
    u, v, w = v_comp(alpha, beta, V)
    theta = climb_2_elev(u, v, w, phi, gamma, V)
    if not shss:
        p, q, r = rotation_rates(phi, theta, u, w, props)
        pbar = p*props.b_w/(2.*V)
        qbar = q*props.c_w/(2.*V)
        rbar = r*props.b_w/(2.*V)
    else:
        p, q, r = [0., 0., 0.]
        pbar, qbar, rbar = [0., 0., 0.]
    FM = coeffs.aero_results(alpha, beta, pbar, qbar, rbar, da, de, dr)
    [CL, CS, CD, Cl, Cm, Cn] = FM
    CX = -(CD*np.cos(alpha)*np.cos(beta) + CS*np.cos(alpha)*np.sin(beta) -
           CL*np.sin(alpha))
    CY = CS*np.cos(beta) - CD*np.sin(beta)
    CZ = -(CD*np.sin(alpha)*np.cos(beta) + CS*np.sin(alpha)*np.sin(beta) +
           CL*np.cos(alpha))
    Fx = CX*props.nondim_const + thrust(tau, V, props)
    Fy = CY*props.nondim_const
    Fz = CZ*props.nondim_const
    Mx = Cl*props.nondim_const*props.b_w - Fz*props.y_shift + Fy*props.z_shift
    My = Cm*props.nondim_const*props.c_w - Fx*props.z_shift + Fz*props.x_shift
    Mz = Cn*props.nondim_const*props.b_w - Fy*props.x_shift + Fx*props.y_shift
    FM = [Fx, Fy, Fz, Mx, My, Mz]
    return FM, [u, v, w], [p, q, r], theta


def fpi(tau, alpha, beta, rot_rates, de, da, dr, vel_comp,
        phi, theta, coeffs, FM, props, bire, dm_E=0., dn_E=0.):
    V = props.V
    [Fx, Fy, Fz, Mx, My, Mz] = FM
    [p, q, r] = rot_rates
    [u, v, w] = vel_comp
    T = thrust(tau, V, props)
    if bire:
        dB = dr
        CLa = coeffs._CL_alpha(0.)
        CLde = coeffs._CL_de(0.)
        CSb = coeffs._CS_beta(0.)
        Clda = coeffs._Cl_da(0.)
        Cmde = coeffs._Cm_de(0.)
        pbar = p*props.b_w/(2.*V)
        qbar = q*props.c_w/(2.*V)
        rbar = r*props.b_w/(2.*V)
        CndB = coeffs.Cn_dB(alpha, beta, pbar, qbar, rbar, da, de, 0.)
    else:
        CLa = coeffs.CLa
        CSb = coeffs.CSb
        Clda = coeffs.Clda
        Cmde = coeffs.Cmde
        Cndr = coeffs.Cndr
    tau_p1, num_tau = _tau_p1(tau, Fx, theta, q, r, v, w, T, props)
```

```
    beta_p1, num_beta = _beta_p1(beta, Fy, phi, theta, p, r, u, w, CSb, props)
    alpha_p1, num_alpha = _alpha_p1(alpha, Fz, phi, theta, p, q, u, v, CLa, props)
    da_p1, num_da = _da_p1(da, Mx, p, q, r, Clda, props)
    if bire:
        de_p1, num_de = _de_p1(de, My, p, q, r, Cmde, props)
        dB_p1, num_dB = _dB_p1(dB, Mz, p, q, r, CndB, props)
        error = np.array([num_tau, num_beta, num_alpha, num_da, num_de, num_dB])
        return np.array([tau_p1, alpha_p1, beta_p1, da_p1, de_p1, dB_p1]), error
    else:
        de_p1, num_de = _de_p1(de, My, p, q, r, Cmde, props)
        dr_p1, num_dr = _dr_p1(dr, Mz, p, q, r, Cndr, props)
        error = np.array([num_tau, num_beta, num_alpha, num_da, num_de, num_dr])
        return np.array([tau_p1, alpha_p1, beta_p1, da_p1, de_p1, dr_p1]), error

def jacobian(trim_state, phi, theta, gamma, coeffs, props, shss, delta=0.001):
    [tau, alpha, beta, de, da, dr] = trim_state
    J = np.zeros((6, 6))
    f = [_f1, _f2, _f3, _f4, _f5, _f6]
    for i in range(6):
        delta_state = np.zeros(6)
        delta_state[i] = delta
        for j in range(6):
            FM_p, vcomp_p, rotrates_p, theta_p = _recalc_forces([t + d for t,d in
                                                                 zip(trim_state,
                                                                    delta_state)],
                                                                 phi, gamma, coeffs,
                                                                 props, shss)
            FM_m, vcomp_m, rotrates_m, theta_m = _recalc_forces([t - d for t,d in
                                                                 zip(trim_state,
                                                                     delta_state)],
                                                                 phi, gamma, coeffs,
                                                                 props, shss)
            f_p = f[j](FM_p[j], theta_p, phi, rotrates_p, vcomp_p, props)
            f_m = f[j](FM_m[j], theta_m, phi, rotrates_m, vcomp_m, props)
            J[j, i] = (f_p - f_m)/(2.*delta)
    return J

def compressible_correction(a0, Lambda, AR, M):
    num = a0*np.cos(Lambda)
    denom_1 = np.sqrt(1. - M**2*np.cos(Lambda)**2 +
                      (num/(np.pi*AR))**2)
    denom_2 = num/(np.pi*AR)
    denom = denom_1 + denom_2
    return num/denom

def trim(V, H, gamma, phi, Gamma, trim_0=np.zeros(6),
         shss=False, bire=False, cg_shift=[0., 0., 0.], verbose=True,
         fixed_point=True, aero_dir='./', **kwargs):
    props_fn = kwargs.get('props_filename', False)
    model_fn = kwargs.get('model_filename', False)
    if not props_fn:
        props = AircraftProperties(V, H, Gamma, path=aero_dir, bire=bire)
    else:
        props = AircraftProperties(V, H, Gamma, path=aero_dir, bire=bire,
                                   filename=props_fn)
    trim_state = trim_0
```

```
    x_shift, y_shift, z_shift = cg_shift
    props.x_shift = x_shift
    props.y_shift = y_shift
    props.z_shift = z_shift
    comp_correction = kwargs.get("compressible", False)
    if bire:
        [tau, alpha, beta, da, de, dB] = trim_state
        coeffs = kwargs.get("coeffs", BIREAero(aero_dir))
    else:
        [tau, alpha, beta, da, de, dr] = trim_state
        if not model_fn:
            coeffs = kwargs.get("coeffs", F16Aero(aero_dir))
        else:
            coeffs = kwargs.get('coeffs', F16Aero(aero_dir, fn=model_fn))
    p, q, r = [0., 0., 0.]
    pbar, qbar, rbar = p, q, r
    error = 100.
    number_of_iterations = 0
    while (error > 1e-9)*(number_of_iterations <= 800):
        number_of_iterations += 1
        u, v, w = v_comp(alpha, beta, V)
        theta = climb_2_elev(u, v, w, phi, gamma, V)
        if not shss:
            p, q, r = rotation_rates(phi, theta, u, w, props)
            pbar = p*props.b_w/(2.*V)
            qbar = q*props.c_w/(2.*V)
            rbar = r*props.b_w/(2.*V)
        if bire:
            FM = coeffs.aero_results(alpha, beta, pbar, qbar, rbar, da, de, dB)
            props.calc_BIRE_inertia(dB)
        else:
            FM = coeffs.aero_results(alpha, beta, pbar, qbar, rbar, da, de, dr)
        [CL, CS, CD, Cl, Cm, Cn] = FM
        if comp_correction:
            if props.M < 1.:
                if bire:
                    CL = compressible_correction(CL, props.Lam_w, props.RA_w, props.M)
                    CS = compressible_correction(CS, props.Lam_h, props.RA_h, props.M)
                    Cl = compressible_correction(Cl, props.Lam_w, props.RA_w, props.M)
                    Cm = compressible_correction(Cm, props.Lam_w, props.RA_w, props.M)
                    Cn = compressible_correction(Cn, props.Lam_h, props.RA_h, props.M)
                else:
                    CL = compressible_correction(CL, props.Lam_w, props.RA_w, props.M)
                    CS = compressible_correction(CS, props.Lam_v, props.RA_v, props.M)
                    Cl = compressible_correction(Cl, props.Lam_v, props.RA_v, props.M)
                    Cm = compressible_correction(Cm, props.Lam_w, props.RA_w, props.M)
                    Cn = compressible_correction(Cn, props.Lam_v, props.RA_v, props.M)
            else:
                CL = CL/np.sqrt(props.M**2 - 1.)
                CS = CS/np.sqrt(props.M**2 - 1.)
                Cl = Cl/np.sqrt(props.M**2 - 1.)
                Cm = Cm/np.sqrt(props.M**2 - 1.)
                Cn = Cn/np.sqrt(props.M**2 - 1.)
        CX = -(CD*np.cos(alpha)*np.cos(beta) + CS*np.cos(alpha)*np.sin(beta) -
               CL*np.sin(alpha))
        CY = CS*np.cos(beta) - CD*np.sin(beta)
```

```
            CZ = -(CD*np.sin(alpha)*np.cos(beta) + CS*np.sin(alpha)*np.sin(beta) +
                   CL*np.cos(alpha))
            Fx = CX*props.nondim_const + thrust(trim_state[0], V, props)
            Fy = CY*props.nondim_const
            Fz = CZ*props.nondim_const
            Mx = Cl*props.nondim_const*props.b_w - Fz*y_shift + Fy*z_shift
            My = Cm*props.nondim_const*props.c_w - Fx*z_shift + Fz*x_shift
            Mz = Cn*props.nondim_const*props.b_w - Fy*x_shift + Fx*y_shift
            FM = [Fx, Fy, Fz, Mx, My, Mz]
            if fixed_point:
                if bire:
                    trimstate_p1, nums = fpi(tau, alpha, beta, [p, q, r], de, da, dB,
                                             [u, v, w], phi, theta, coeffs, FM, props,
                                             bire)
                else:
                    trimstate_p1, nums = fpi(tau, alpha, beta, [p, q, r], de, da, dr,
                                             [u, v, w], phi, theta, coeffs, FM, props,
                                             bire)
            else:
                f = [_f1, _f2, _f3, _f4, _f5, _f6]
                nums = np.array([f(FM[idx], theta, phi,
                                   [p, q, r], [u, v, w], props) for idx, f in
                                 enumerate(f)])
                try:
                    J = jacobian(trim_state, phi, theta, gamma, coeffs, props, shss)
                    D_G = np.linalg.solve(-J, nums)
                except np.linalg.LinAlgError:
                    J = jacobian(trim_state, phi, theta, gamma, coeffs, props, shss,
                                 delta=0.1)
                    D_G = np.linalg.solve(-J, nums)
                trimstate_p1 = trim_state + Gamma*D_G
            error = np.max(np.abs(nums))
            trim_state = trimstate_p1
            if bire:
                [tau, alpha, beta, da, de, dB] = trim_state
            else:
                [tau, alpha, beta, da, de, dr] = trim_state
        T = thrust(trim_state[0], V, props)
        n_a = ((np.cos(theta)*np.cos(phi) + (q*u - p*v)/props.g)*np.cos(alpha) +
               (np.sin(theta) - (r*v - q*w)/props.g)*np.sin(alpha))
        n_sa = CL/(props.W/(0.5*props.rho*V**2*props.S_w))
        if bire:
            while abs(dB) > np.pi:
                if dB >= 2.*np.pi:
                    while dB >= np.pi:
                        dB -= 2.*np.pi
                if dB > np.pi:
                    while dB > np.pi:
                        dB -= np.pi
                        de *= -1.
                if dB <= -2.*np.pi:
                    while dB <= -np.pi:
                        dB += 2.*np.pi
                if dB < -np.pi:
                    while dB < -np.pi:
                        dB += np.pi
```

```
                de *= -1.
    if verbose:
        print("------ Trim Solution ------")
        print(f"Elevation Angle (deg.) : {theta*180./np.pi:1.12g}")
        print(f"Bank Angle (deg.) : {phi*180./np.pi:1.12g}")
        print(f"Alpha (deg.) : {alpha*180./np.pi:1.12g}")
        print(f"Beta (deg.) : {beta*180./np.pi:1.12g}")
        print(f"p (deg./s) : {p*180./np.pi:1.12g}")
        print(f"q (deg./s) : {q*180./np.pi:1.12g}")
        print(f"r (deg./s) : {r*180./np.pi:1.12g}")
        print(f"Aileron (deg.) : {da*180./np.pi:1.12g}")
        print(f"Elevator (deg.) : {de*180./np.pi:1.12g}")
        if bire:
            print(f"BIRE Rotation (deg.) : {dB*180./np.pi:1.12g}")
        else:
            print(f"Rudder (deg.) : {dr*180./np.pi:1.12g}")
        print(f"Throttle : {tau:1.12g}")
        print(f"Thrust (lbf.) : {T:1.12f}")
        print(f"Load Factor : {n_a:1.12f}")
        print(f"Stability Axis Load Factor : {n_sa:1.12f}")
        print(f"Number of Iterations : {number_of_iterations:d}")
    solution = TrimSolution()
    solution.FM = np.array([CD, CS, CL, Cl, Cm, Cn])
    solution.FM_dim = np.array([Fx, Fy, Fz, Mx, My, Mz])
    solution.load = n_a
    solution.load_s = n_sa
    solution.x = trim_state
    solution.num_iters = number_of_iterations
    solution.orient = np.array([phi, theta, 0.])
    solution.velocity = np.array([u, v, w])
    solution.rates = np.array([p, q, r])
    solution.states = np.array([u, v, w, p, q, r, phi, theta])
    solution.aero = coeffs
    solution.error = error
    solution.nums = nums
    if bire:
        solution.vehicle = "BIRE"
        solution.inputs = np.array([tau, da, de, dB])
    else:
        solution.vehicle = "Baseline"
        solution.inputs = np.array([tau, da, de, dr])
    return solution
```

Steady, Coordinated Turn Analysis

```
import numpy as np
import aero_trim as trim
import scipy.optimize as optimize

H = 30000.
gamma = 0.
Gamma = 0.8
Gamma_B = 0.8
N = 50
M = np.load('./Crosswind Data/SHSS_Mach.npy')
V = M*a
n = np.linspace(1., 9., N)
cg_shift = [1., 0., 0.]

def find_loadfactor(phi, n, V, bire):
    if bire:
        try:
            solution_bire = trim.trim(V, H, gamma, phi[0], Gamma_B, shss=False,
                                      bire=bire, cg_shift=cg_shift, verbose=False,
                                      fixed_point=False, compressible=True)
            n_a = solution_bire.load
        except np.linalg.LinAlgError:
            solution_bire = trim.trim(V, H, gamma, phi[0], 0.1, shss=False,
                                      bire=bire, cg_shift=cg_shift, verbose=False,
                                      fixed_point=False, compressible=True)
            n_a = solution_bire.load
        return (n - n_a)**2
    else:
        try:
            solution_base = trim.trim(V, H, gamma, phi[0], Gamma, shss=False,
                                      bire=bire, cg_shift=cg_shift, verbose=False,
                                      fixed_point=False, compressible=True)
            n_a = solution_base.load
        except np.linalg.LinAlgError:
            solution_base = trim.trim(V, H, gamma, phi[0], 0.1, shss=False,
                                      bire=bire, cg_shift=cg_shift, verbose=False,
                                      fixed_point=False, compressible=True)
            n_a = solution_base.load
        return (n - n_a)**2

dr = np.zeros((N, N))
de = np.zeros((N, N))
deB = np.zeros((N, N))
dB = np.zeros((N, N))
CL_base = np.zeros((N, N))
CL_BIRE = np.zeros((N, N))
CD_base = np.zeros((N, N))
CD_BIRE = np.zeros((N, N))
Cn_base = np.zeros((N, N))
Cn_BIRE = np.zeros((N, N))
phi_base = np.zeros((N, N))
phi_0 = 0.1
for i in range(len(V)):
    print(V[i])
```

```
        CW = W/(0.5*rho*V[i]**2*S_w)
        n_stall = CLmax/CW
        for j in range(len(n)):
            print(n[j])
            if n[j] > n_stall:
                dr[i, j] = np.nan
                de[i, j] = np.nan
                deB[i, j] = np.nan
                dB[i, j] = np.nan
                CL_base[i, j] = np.nan
                CL_BIRE[i, j] = np.nan
                CD_base[i, j] = np.nan
                CD_BIRE[i, j] = np.nan
                Cn_base[i, j] = np.nan
                Cn_BIRE[i, j] = np.nan
                phi_base[i, j] = np.nan
                print('stalled')
            else:
                if j == 0:
                    phi_0 = np.arccos(1./n[j])
                else:
                    phi_0 = phi_base[i, j-1]
                phi_base[i, j] = optimize.minimize(find_loadfactor, phi_0,
                                                   args=(n[j], V[i], False),
                                                   method='Nelder-Mead',
                                                   options={'fatol': 1e-12}).x[0]
                solution_base = trim.trim(V[i], H, gamma, phi_base[i, j], Gamma,
                                          shss=False, bire=False, cg_shift=cg_shift,
                                          verbose=False, fixed_point=False,
                                          compressible=True)
                trim_base = solution_base.x
                CL_base[i, j] = solution_base.FM[2]
                CD_base[i, j] = solution_base.FM[0]
                Cn_base[i, j] = solution_base.FM[5]
                na_base = solution_base.load
                try:
                    solution_BIRE = trim.trim(V[i], H, gamma, phi_base[i, j], Gamma,
                                              shss=False, bire=True, cg_shift=cg_shift,
                                              verbose=False, fixed_point=False,
                                              compressible=True)
                    trim_BIRE = solution_BIRE.x
                    CL_BIRE[i, j] = solution_BIRE.FM[2]
                    CD_BIRE[i, j] = solution_BIRE.FM[0]
                    Cn_BIRE[i, j] = solution_BIRE.FM[5]
                    na_BIRE = solution_BIRE.load
                except np.linalg.LinAlgError:
                    trim_BIRE = [np.nan]*6
                    CL_BIRE[i, j] = np.nan
                    CD_BIRE[i, j] = np.nan
                    Cn_BIRE[i, j] = np.nan
                    na_BIRE = np.nan
                dr[i, j] = np.rad2deg(trim_base[5])
                de[i, j] = np.rad2deg(trim_base[4])
                deB[i, j] = np.rad2deg(trim_BIRE[4])
                dB[i, j] = np.rad2deg(trim_BIRE[5])
                print(na_base - n[j], na_BIRE - n[j])
```

```
        phi_0 = phi_base[i, j]
```

Steady, Coordinated Turn CG Analysis

```
import numpy as np
import matplotlib.pyplot as plt
import aero_trim
from matplotlib import colors
from stdatmos import stdatm_english
import scipy.optimize as optimize
from bire_aero import BIREAero

H = 30000.
CLmax = 1.9
rho = stdatm_english(H)[3]
W = 20500.
S_w = 300.
V_stall = np.sqrt(2.*W/S_w/CLmax/rho)

aft_cg_limit = 11.32*(0.35 - 0.4)
x_shifts = np.linspace(1.5, aft_cg_limit)
deB = np.zeros(len(x_shifts))
de = np.zeros(len(x_shifts))
dB = np.zeros(len(x_shifts))
dr = np.zeros(len(x_shifts))
phi = np.zeros(len(x_shifts))
FM = np.zeros((len(x_shifts), 6))

gamma = 0.
Gamma = 0.1
Gamma_B = 0.1
case = BIREAero()
n_target = 5.

def find_target_g(phi, x_shift):
    solution = aero_trim.trim(V_stall, H, gamma, phi[0], Gamma, shss=False,
                              cg_shift=[x_shift, 0., 0.], verbose=False,
                              fixed_point=True)
    n_a = solution.load
    return abs(n_a - n_target)**2

trim_0 = np.zeros(6)
phi_0 = 0.
for i in range(len(x_shifts)):
    res = optimize.minimize(find_target_g, 0., args=(x_shifts[i]),
                            method='Nelder-Mead',
                            options={'gtol': 1e-6, 'return_all': True})
    phi[i] = res.x[0]
    try:
        solution_base = aero_trim.trim(V_stall, H, gamma, phi[i], Gamma, shss=False,
                                       cg_shift=[x_shifts[i], 0., 0.], verbose=False)
        state_na = solution_base.x
    except TypeError:
        state_na = np.array([np.nan]*6)
    de[i] = state_na[4]*180./np.pi
    dr[i] = state_na[5]*180./np.pi

    solution_bire = aero_trim.trim(V_stall, H, gamma, phi[i], Gamma, shss=False,
```

```
                                    cg_shift=[x_shifts[i], 0., 0.], verbose=False,
                                    bire=True, fixed_point=False,
                                    trim_0=trim_0)
        state_na_bire = solution_bire.x
        deB[i] = state_na_bire[4]*180./np.pi
        dB[i] = state_na_bire[5]*180./np.pi
        trim_0 = state_na_bire
        print(dB[i])
        phi_0 = phi[i]
```

Steady-Heading Sideslip Analysis

```
import numpy as np
import aero_trim

Gamma = 0.8
Gamma_B = 0.5
H = 1000.
N = 50
phi = np.linspace(0., 45., N)
M = np.linspace(0.2, 0.8, N)
V = M*a
gamma = np.deg2rad(0.)
cg_shift = [0., 0., 0.]

rudder_deg = np.zeros((len(V), len(phi)))
V_cross = np.zeros((len(V), len(phi)))
elevator_deg = np.zeros((len(V), len(phi)))
trim_state = np.zeros(6)
CL_base = np.zeros((len(V), len(phi)))
CD_base = np.zeros_like(CL_base)
Cn_base = np.zeros_like(CL_base)
phi_deg = np.zeros((len(V), len(phi)))
theta_deg = np.zeros((len(V), len(phi)))
BIRE_rotation_deg = np.zeros((len(V), len(phi)))
BIRE_V_cross = np.zeros((len(V), len(phi)))
BIRE_elevator_deg = np.zeros((len(V), len(phi)))
BIRE_phi_deg = np.zeros((len(V), len(phi)))
BIRE_theta_deg = np.zeros((len(V), len(phi)))
CL_BIRE = np.zeros((len(V), len(phi)))
CD_BIRE = np.zeros((len(V), len(phi)))
Cn_BIRE = np.zeros_like(CL_BIRE)
trim_state_bire = np.zeros(6)
for i in range(len(V)):
    trim_0 = np.zeros(6)
    trim_state_bire = np.zeros(6)
    trim_state = np.zeros(6)
    for j in range(len(phi)):
        if trim_state[5]*180./np.pi > 35.:
            rudder_deg[i, j] = np.nan
            elevator_deg[i, j] = np.nan
            phi_deg[i, j] = np.nan
            theta_deg[i, j] = np.nan
            V_cross[i, j] = np.nan
            BIRE_rotation_deg[i, j] = np.nan
            BIRE_elevator_deg[i, j] = np.nan
            BIRE_phi_deg[i, j] = np.nan
            BIRE_theta_deg[i, j] = np.nan
            BIRE_V_cross[i, j] = np.nan
        else:
            try:
                solution_base = aero_trim.trim(V[i], H, gamma, np.deg2rad(phi[j]),
                                               Gamma, shss=True, cg_shift=cg_shift,
                                               verbose=True)
                trim_state = solution_base.x
                CL_base[i, j] = solution_base.FM[2]
```

```
                CD_base[i, j] = solution_base.FM[0]
                Cn_base[i, j] = solution_base.FM[5]
                phi_ij = solution_base.orient[0]
                theta_ij = solution_base.orient[1]
                [u, v, w] = solution_base.velocity
                rudder_deg[i, j] = trim_state[5]*180./np.pi
                elevator_deg[i, j] = trim_state[4]*180./np.pi
                phi_deg[i, j] = phi_ij*180./np.pi
                theta_deg[i, j] = theta_ij*180./np.pi
                c_a = np.cos(trim_state[1])
                s_a = np.sin(trim_state[1])
                c_b = np.cos(trim_state[2])
                s_b = np.sin(trim_state[2])
                V_cross[i, j] = -c_a*s_b*u + c_b*v - s_a*s_b*w
            except TypeError:
                trim_state = np.array([np.nan]*6)
                CL_base[i, j] = np.nan
                CD_base[i, j] = np.nan
                Cn_base[i, j] = np.nan
                phi_deg[i, j] = np.nan
                theta_deg[i, j] = np.nan
                V_cross[i, j] = np.nan
            try:
                solution_bire = aero_trim.trim(V[i], H, gamma, np.deg2rad(phi[j]),
                                               Gamma_B, shss=True, cg_shift=cg_shift,
                                               verbose=True, bire=True,
                                               fixed_point=False, trim_0=trim_0)
                trim_state_bire = solution_bire.x
                CL_BIRE[i, j] = solution_bire.FM[2]
                CD_BIRE[i, j] = solution_bire.FM[0]
                Cn_BIRE[i, j] = solution_bire.FM[5]
                phi_ij = solution_bire.orient[0]
                theta_ij = solution_bire.orient[1]
                [u, v, w] = solution_bire.velocity
                BIRE_rotation_deg[i, j] = trim_state_bire[5]*180./np.pi
                BIRE_elevator_deg[i, j] = trim_state_bire[4]*180./np.pi
                BIRE_phi_deg[i, j] = phi_ij*180./np.pi
                BIRE_theta_deg[i, j] = theta_ij*180./np.pi
                c_a = np.cos(trim_state_bire[1])
                s_a = np.sin(trim_state_bire[1])
                c_b = np.cos(trim_state_bire[2])
                s_b = np.sin(trim_state_bire[2])
                BIRE_V_cross[i, j] = -c_a*s_b*u + c_b*v - s_a*s_b*w
                trim_0 = trim_state_bire
            except TypeError:
                trim_state_bire = np.array([np.nan]*6)
                CL_BIRE[i, j] = np.nan
                phi_deg[i, j] = np.nan
                theta_deg[i, j] = np.nan
                BIRE_V_cross[i, j] = np.nan
```

Steady-Heading Sideslip CG Analysis

```
import numpy as np
import aero_trim

Gamma = 0.8
Gamma_B = 0.5
H = 1000.
N = 50
phi = np.linspace(0., 45., N)
M = np.linspace(0.2, 0.8, N)
V = M*a
gamma = np.deg2rad(0.)
cg_shift = [0., 0., 0.]

rudder_deg = np.zeros((len(V), len(phi)))
V_cross = np.zeros((len(V), len(phi)))
elevator_deg = np.zeros((len(V), len(phi)))
trim_state = np.zeros(6)
CL_base = np.zeros((len(V), len(phi)))
CD_base = np.zeros_like(CL_base)
Cn_base = np.zeros_like(CL_base)
phi_deg = np.zeros((len(V), len(phi)))
theta_deg = np.zeros((len(V), len(phi)))
BIRE_rotation_deg = np.zeros((len(V), len(phi)))
BIRE_V_cross = np.zeros((len(V), len(phi)))
BIRE_elevator_deg = np.zeros((len(V), len(phi)))
BIRE_phi_deg = np.zeros((len(V), len(phi)))
BIRE_theta_deg = np.zeros((len(V), len(phi)))
CL_BIRE = np.zeros((len(V), len(phi)))
CD_BIRE = np.zeros((len(V), len(phi)))
Cn_BIRE = np.zeros_like(CL_BIRE)
trim_state_bire = np.zeros(6)
for i in range(len(V)):
    trim_0 = np.zeros(6)
    trim_state_bire = np.zeros(6)
    trim_state = np.zeros(6)
    for j in range(len(phi)):
        if trim_state[5]*180./np.pi > 35.:
            rudder_deg[i, j] = np.nan
            elevator_deg[i, j] = np.nan
            phi_deg[i, j] = np.nan
            theta_deg[i, j] = np.nan
            V_cross[i, j] = np.nan
            BIRE_rotation_deg[i, j] = np.nan
            BIRE_elevator_deg[i, j] = np.nan
            BIRE_phi_deg[i, j] = np.nan
            BIRE_theta_deg[i, j] = np.nan
            BIRE_V_cross[i, j] = np.nan
        else:
            try:
                solution_base = aero_trim.trim(V[i], H, gamma, np.deg2rad(phi[j]),
                                               Gamma, shss=True, cg_shift=cg_shift,
                                               verbose=True)
                trim_state = solution_base.x
                CL_base[i, j] = solution_base.FM[2]
```

```
                CD_base[i, j] = solution_base.FM[0]
                Cn_base[i, j] = solution_base.FM[5]
                phi_ij = solution_base.orient[0]
                theta_ij = solution_base.orient[1]
                [u, v, w] = solution_base.velocity
                rudder_deg[i, j] = trim_state[5]*180./np.pi
                elevator_deg[i, j] = trim_state[4]*180./np.pi
                phi_deg[i, j] = phi_ij*180./np.pi
                theta_deg[i, j] = theta_ij*180./np.pi
                c_a = np.cos(trim_state[1])
                s_a = np.sin(trim_state[1])
                c_b = np.cos(trim_state[2])
                s_b = np.sin(trim_state[2])
                V_cross[i, j] = -c_a*s_b*u + c_b*v - s_a*s_b*w
            except TypeError:
                trim_state = np.array([np.nan]*6)
                CL_base[i, j] = np.nan
                CD_base[i, j] = np.nan
                Cn_base[i, j] = np.nan
                phi_deg[i, j] = np.nan
                theta_deg[i, j] = np.nan
                V_cross[i, j] = np.nan
            try:
                solution_bire = aero_trim.trim(V[i], H, gamma, np.deg2rad(phi[j]),
                                               Gamma_B, shss=True, cg_shift=cg_shift,
                                               verbose=True, bire=True,
                                               fixed_point=False, trim_0=trim_0)
                trim_state_bire = solution_bire.x
                CL_BIRE[i, j] = solution_bire.FM[2]
                CD_BIRE[i, j] = solution_bire.FM[0]
                Cn_BIRE[i, j] = solution_bire.FM[5]
                phi_ij = solution_bire.orient[0]
                theta_ij = solution_bire.orient[1]
                [u, v, w] = solution_bire.velocity
                BIRE_rotation_deg[i, j] = trim_state_bire[5]*180./np.pi
                BIRE_elevator_deg[i, j] = trim_state_bire[4]*180./np.pi
                BIRE_phi_deg[i, j] = phi_ij*180./np.pi
                BIRE_theta_deg[i, j] = theta_ij*180./np.pi
                c_a = np.cos(trim_state_bire[1])
                s_a = np.sin(trim_state_bire[1])
                c_b = np.cos(trim_state_bire[2])
                s_b = np.sin(trim_state_bire[2])
                BIRE_V_cross[i, j] = -c_a*s_b*u + c_b*v - s_a*s_b*w
                trim_0 = trim_state_bire
            except TypeError:
                trim_state_bire = np.array([np.nan]*6)
                CL_BIRE[i, j] = np.nan
                phi_deg[i, j] = np.nan
                theta_deg[i, j] = np.nan
                BIRE_V_cross[i, j] = np.nan
```

Tail Strike Analysis

```
import numpy as np
import aero_trim
from stdatmos import stdatm_english
import scipy.optimize as optimize

H = 15000.
M = np.load('./Crosswind Data/SHSS_Mach.npy')
a = stdatm_english(H)[-1]
V = M*a
phi = np.load('./Crosswind Data/SHSS_Bank_Angle.npy')
CLmax = 1.9
gamma = 0.
cg_shift = [1., 0., 0.]
BIRE_rotation_deg = np.load(f"./Crosswind Data/SHSS_BIRE_rotation{int(H):2d}" +
                            f"CG{int(cg_shift[0] - 1):2d}.npy")
BIRE_elevator_deg = np.load(f"./Crosswind Data/SHSS_BIRE_elevator{int(H):2d}" +
                            f"CG{int(cg_shift[0] - 1):2d}.npy")
phi_rad = np.load(f"./Crosswind Data/Tail_Strike_BIRE_phi{int(H):2d}" +
                  f"CG{int(cg_shift[0] - 1):2d}.npy")*np.pi/180.
theta = np.load(f"./Crosswind Data/Tail_Strike_BIRE_theta{int(H):2d}" +
                f"CG{int(cg_shift[0] - 1):2d}.npy")*np.pi/180.
base_elevator_deg = np.load(f"./Crosswind Data/SHSS_base_elevator{int(H):2d}" +
                            f"CG{int(cg_shift[0] - 1):2d}.npy")
base_phi_rad = np.load(f"./Crosswind Data/SHSS_base_phi{int(H):2d}" +
                       f"CG{int(cg_shift[0] - 1):2d}.npy")*np.pi/180.
base_theta = np.load(f"./Crosswind Data/SHSS_base_theta{int(H):2d}" +
                     f"CG{int(cg_shift[0] - 1):2d}.npy")*np.pi/180.
rudder_deg = np.load(f"./Crosswind Data/SHSS_base_rudder{int(H):2d}" +
                     f"CG{int(cg_shift[0] - 1):2d}.npy")
CL_base = np.load(f"./Crosswind Data/SHSS_base_CL{int(H):2d}" +
                  f"CG{int(cg_shift[0] - 1):2d}.npy")
V_cross = np.load(f"./Crosswind Data/SHSS_base_Vcross{int(H):2d}" +
                  f"CG{int(cg_shift[0] - 1):2d}.npy")
BIRE_V_cross = np.load(f"./Crosswind Data/Tail_Strike_BIRE_Vcross{int(H):2d}" +
                       f"CG{int(cg_shift[0] - 1):2d}.npy")
z_LG_E_base = np.zeros_like(rudder_deg)
z_LG_TEL_base = np.zeros_like(rudder_deg)
z_LG_TER_base = np.zeros_like(rudder_deg)
z_LG_E_bire = np.zeros_like(rudder_deg)
z_LG_TEL_bire = np.zeros_like(rudder_deg)
z_LG_TER_bire = np.zeros_like(rudder_deg)

h_intake = 2.906  # From Nguyen Drawing Scaled from Centerline
h_landing = h_intake*2.  # From centerline to ground is ~ two intakes
b_h = 9.2
c_rh = 7.9833 # stab root chord
s_fh = 3.4  # semispan of fuselage portion of h-stab
l_h = 13.13 + c_rh*3./4.  # from CG to TE of h-stab
G_h = -10.*np.pi/180.  # Anhedral of baseline tail
z_LG = 5.812
y_LG = 1.557
x_LG = -0.3063
x_E = -18.71
y_E = 0.
```

```
z_E = 1.679
x_TE = -l_h
y_TE = b_h  # based on BIRE
z_TE = 0.  # based on BIRE
x_SP = -l_h + 0.6*c_rh
y_SP = s_fh
z_SP = 0.
p_LG = np.array([-x_LG, y_LG, -z_LG])  # LG to CG
p_E = np.array([x_E, y_E, z_E])  # CG to engine
# Base
base_P_L = np.array([x_SP, -y_SP, z_SP])  # CG to left stab pivot
base_P_R = np.array([x_SP, y_SP, z_SP])  # CG to right stab pivot
base_TE_L = np.array([-0.6*c_rh,
                      -(y_TE - y_SP)*np.cos(G_h),
                      z_TE - b_h*np.sin(G_h)])  # left stab pivot to left TE
base_TE_R = np.array([-0.6*c_rh,
                      (y_TE - y_SP)*np.cos(G_h),
                      z_TE - b_h*np.sin(G_h)])  # right stab pivot to right TE
# BIRE
bire_EMP = np.array([x_SP, 0., z_SP]) # CG to center of empennage rotation
bire_E_P_R = np.array([0., y_SP, 0.])  # center of empennage to right stab pivot
bire_E_P_L = np.array([0., -y_SP, 0.])  # center of empennage to left stab pivot
bire_P_TE_R = np.array([-0.6*c_rh, (y_TE - y_SP), z_TE])  # right pivot to tip corner
bire_P_TE_L = np.array([-0.6*c_rh, -(y_TE - y_SP), z_TE])  # left pivot to tip corner

for i in range(len(V)):
    for j in range(len(phi)):
        dB = BIRE_rotation_deg[i, j]*np.pi/180.
        de_BIRE = BIRE_elevator_deg[i, j]*np.pi/180.
        de_base = base_elevator_deg[i, j]*np.pi/180.
        R_theta = np.array([[np.cos(theta[i, j]), 0., np.sin(theta[i, j])],
                            [0., 1., 0.],
                            [-np.sin(theta[i, j]), 0., np.cos(theta[i, j])]])
        R_phi = np.array([[1., 0., 0.],
                          [0., np.cos(phi_rad[i, j]), -np.sin(phi_rad[i, j])],
                          [0., np.sin(phi_rad[i, j]), np.cos(phi_rad[i, j])]])
        R_dB = np.array([[1., 0., 0.],
                         [0., np.cos(dB), -np.sin(dB)],
                         [0., np.sin(dB), np.cos(dB)]])
        R_de = np.array([[np.cos(de_BIRE), 0., np.sin(de_BIRE)],
                         [0., 1., 0.],
                         [-np.sin(de_BIRE), 0., np.cos(de_BIRE)]])
        P_LG_E = np.matmul(R_theta, np.matmul(R_phi, p_LG + p_E))
        P_LG_EMP = np.matmul(R_theta, np.matmul(R_phi, p_LG + bire_EMP))
        P_LG_PL = P_LG_EMP + np.matmul(R_dB, bire_E_P_L)
        P_LG_PR = P_LG_EMP + np.matmul(R_dB, bire_E_P_R)
        P_LG_TEL = np.matmul(R_de, P_LG_PL + bire_P_TE_L)
        P_LG_TER = np.matmul(R_de, P_LG_PR + bire_P_TE_R)
        z_LG_E_bire[i, j] = P_LG_E[2]
        z_LG_TEL_bire[i, j] = P_LG_TEL[2]
        z_LG_TER_bire[i, j] = P_LG_TER[2]


        R_phi = np.array([[1., 0., 0.],
                          [0., np.cos(base_phi_rad[i, j]),
                           -np.sin(base_phi_rad[i, j])],
```

```
                              [0., np.sin(base_phi_rad[i, j]),
                               np.cos(base_phi_rad[i, j])]])
            R_theta = np.array([[np.cos(base_theta[i, j]), 0., np.sin(base_theta[i, j])],
                                 [0., 1., 0.],
                                 [-np.sin(base_theta[i, j]), 0.,
                                  np.cos(base_theta[i, j])]])
            R_de = np.array([[np.cos(de_base), 0., np.sin(de_base)],
                              [0., 1., 0.],
                              [-np.sin(de_base), 0., np.cos(de_base)]])
            P_LG_E = np.matmul(R_theta, np.matmul(R_phi, p_LG + p_E))
            P_LG_PL = p_LG + base_P_L
            P_LG_TEL = np.matmul(R_de, P_LG_PL) + base_TE_L
            P_LG_PR = p_LG + base_P_R
            P_LG_TER = np.matmul(R_de, P_LG_PR) + base_TE_R
            z_LG_E_base[i, j] = P_LG_E[2]
            z_LG_TEL_base[i, j] = P_LG_TEL[2]
            z_LG_TER_base[i, j] = P_LG_TER[2]
```

C.5 Attainable Moment Set Analysis

Moment Set Generation

```
import aero_trim as trim
import numpy as np
import machupX as mx
import matplotlib.pyplot as plt
import matplotlib as mpl
from stdatmos import stdatm_english
import json
from bire_aero import BIREAero
from f16_aero import F16Aero
import alphashape
from descartes import PolygonPatch
import scipy.optimize as optimize
from scipy.interpolate import RegularGridInterpolator

mpl.rcParams['axes.linewidth'] = 1.75 #set the value globally
mpl.rcParams["font.family"] = "serif"
plt.rc('font', weight='bold')

major_dict = {"width" : 1.25, "size" : 7., "labelsize" : 16.,
              "direction" : 'in', "which" : 'major'}
minor_dict = {"width" : 1.25, "size" : 4.,
              "direction" : 'in', "which" : 'minor'}

forces_options = {'body_frame': True,
                  'stab_frame': False,
                  'wind_frame': True,
                  'dimensional': False,
                  'verbose': False}

def pretty_plot(ax, xlims, ylims, dx, dy):
    ax.set_xlim(xlims)
    ax.set_ylim(ylims)
    ax.xaxis.set_major_locator(MultipleLocator(dx["major"]))
    ax.xaxis.set_minor_locator(MultipleLocator(dx["minor"]))
    ax.yaxis.set_major_locator(MultipleLocator(dy["major"]))
    ax.yaxis.set_minor_locator(MultipleLocator(dy["minor"]))
    ax.xaxis.set_ticks_position('both')
    ax.yaxis.set_ticks_position('both')
    ax.tick_params(**major_dict)
    ax.tick_params(**minor_dict)
    return ax

def find_moment(deltas, C_moment, function):
    try:
        est_moment = function(deltas)[0]
    except ValueError:
        est_moment = 100.
    return (est_moment - C_moment)**2
```

```
def generate_AMS_data(H, M, generate_trim=True, generate_data=True, plot=True):
    a = stdatm_english(H)[-1]
    V = M*a
    gamma = 0.
    phi = 0.
    Gamma = 0.5
    if generate_trim:
        solution_base = trim.trim(V, H, gamma, phi, Gamma, bire=False,
                                  shss=False, fixed_point=False)
        solution_bire = trim.trim(V, H, gamma, phi, Gamma, bire=True,
                                  shss=False, fixed_point=False)
        np.save(f'./AMS/Trim States/base_{int(H):2d}_M{M:.1f}.npy', solution_base.x)
        np.save(f'./AMS/Trim States/BIRE_{int(H):2d}_M{M:.1f}.npy', solution_bire.x)
    else:
        solution_base = np.load(f'./AMS/Trim States/base_{int(H):2d}_M{M:.1f}.npy')
        solution_bire = np.load(f'./AMS/Trim States/BIRE_{int(H):2d}_M{M:.1f}.npy')
    [alpha_base, beta_base] = solution_base[1:3]
    da_base = solution_base[3]
    [alpha_bire, beta_bire] = solution_bire[1:3]
    da_bire = solution_bire[3]
    pbar = 0.
    qbar = 0.
    rbar = 0.

    N = 11
    da_range = np.deg2rad(np.linspace(-21.5, 21.5, N))
    de_range = np.deg2rad(np.linspace(-25., 25., N))
    dr_range = np.deg2rad(np.linspace(-30., 30., N))
    dB_range = np.deg2rad(np.linspace(-90., 90., N))
    moments_base = np.zeros((N, N, N, 3))
    moments_bire = np.zeros((N, N, N, 3))
    base_aero = F16Aero()
    bire_aero = BIREAero()
    if generate_data:
        for i in range(N):
            for j in range(N):
                for k in range(N):
                    moments_base[i, j, k, :] = base_aero.aero_results(alpha_base,
                                                                      beta_base,
                                                                      pbar, qbar,
                                                                      rbar,
                                                                      da_range[i],
                                                                      de_range[j],
                                                                      dr_range[k])[-3:]
                    moments_bire[i, j, k, :] = bire_aero.aero_results(alpha_bire,
                                                                      beta_bire,
                                                                      pbar, qbar,
                                                                      rbar,
                                                                      da_range[i],
                                                                      de_range[j],
                                                                      dB_range[k])[-3:]
        np.save(f'./AMS/Data/base_{int(H):2d}_M{M:.1f}.npy', moments_base)
        np.save(f'./AMS/Data/bire_{int(H):2d}_M{M:.1f}.npy', moments_bire)
    else:
        moments_base = np.load(f'./AMS/Data/base_{int(H):2d}_M{M:.1f}.npy')
        moments_bire = np.load(f'./AMS/Data/bire_{int(H):2d}_M{M:.1f}.npy')
```

```

    Cl_base = moments_base[:, :, :, 0]
    Cm_base = moments_base[:, :, :, 1]
    Cn_base = moments_base[:, :, :, 2]
    Cl_bire = moments_bire[:, :, :, 0]
    Cm_bire = moments_bire[:, :, :, 1]
    Cn_bire = moments_bire[:, :, :, 2]

    if plot:
        f_Clbase = RegularGridInterpolator((da_range, de_range, dr_range), Cl_base)
        f_Cmbase = RegularGridInterpolator((da_range, de_range, dr_range), Cm_base)
        f_Cnbase = RegularGridInterpolator((da_range, de_range, dr_range), Cn_base)
        f_Clbire = RegularGridInterpolator((da_range, de_range, dB_range), Cl_bire)
        f_Cmbire = RegularGridInterpolator((da_range, de_range, dB_range), Cm_bire)
        f_Cnbire = RegularGridInterpolator((da_range, de_range, dB_range), Cn_bire)

        fig_3d = plt.figure()
        ax_3d = fig_3d.add_subplot(projection='3d')
        ax_3d.scatter(Cl_base, Cm_base, Cn_base)
        ax_3d.scatter(Cl_bire, Cm_bire, Cn_bire)

        fig_ClCn, ax_ClCn = plt.subplots()
        linestyles = ['-', '--', ':', '-.', (5, (10, 3))]
        markers = ['o', '^', 'd', 's', '>']
        alphas = [3.5, 10., 2., 10., 3.5]
        Cm_legend = []
        dummy_lines = []
        for i in range(N//2):
            Cm_legend.append('$C_m = $' + f'{Cm_base[0, 2*i + 1, 0]:1.3f}')
            ClCn_tuple = [(Cl, Cn) for Cl, Cn in
                          zip(Cl_base[:, 2*i + 1, :].flatten(),
                          Cn_base[:, 2*i + 1, :].flatten())]
            alpha_Cm = alphashape.alphashape(ClCn_tuple, 1.5)
            ax_ClCn.add_patch(PolygonPatch(alpha_Cm, fc='None',
                                           ec='k', ls=linestyles[i]))
            bire_eq = optimize.minimize(find_moment, [0., 0., 0.],
                                        args=(Cm_base[0, 2*i + 1, 0],
                                        f_Cmbire)).x
            print(Cm_base[0, 2*i + 1, 0] - f_Cmbire(bire_eq))
            Cl_pts = np.array([[f_Clbire([a, bire_eq[1], b])[0] for a in
                                da_range] for b in dB_range]).flatten()
            Cn_pts = np.array([[f_Cnbire([a, bire_eq[1], b])[0] for a in
                                da_range] for b in dB_range]).flatten()
            ClCn_tuple = [(Cl, Cn) for Cl, Cn in zip(Cl_pts, Cn_pts)]
            alpha_Cm = alphashape.alphashape(ClCn_tuple, alphas[i])
            ax_ClCn.add_patch(PolygonPatch(alpha_Cm, fc='None',
                                           ec='0.5', ls=linestyles[i]))
            dummy_lines.append(ax_ClCn.plot([], [], c='k', ls=linestyles[i])[0])
        fig_ClCn.legend([dummy_lines[i] for i in range(N//2)],
                        Cm_legend, loc='upper right', fontsize=16)
        ax_ClCn.set_xlabel(r'\textbf{Rolling Moment Coefficient, }\boldmath$C_\ell$',
                           fontsize=16)
        ax_ClCn.set_ylabel(r'\textbf{Yawing Moment Coefficient, }\boldmath$C_n$',
                           fontsize=16)
        xlims = (-0.06, 0.06)
        dx = {'major': 0.05, 'minor': 0.05/4}
```

```
        ylims = (-0.175, 0.175)
        dy = {'major': 0.05, 'minor': 0.05/4}
        ax_ClCn = pretty_plot(ax_ClCn, xlims, ylims, dx, dy)
        ax_ClCn.grid()
        ax_ClCn.set_aspect('equal')
        plt.tight_layout()
        plt.savefig(f'./AMS/Cl_Cn_{int(H):2d}_M{M:.1f}.pdf', dpi=1000)

        fig_ClCm, ax_ClCm = plt.subplots()
        alphas = [2.]*5
        Cn_legend = []
        dummy_lines = []
        for i in range(N//2):
            Cn_legend.append('$C_n = $' + f'{Cn_base[0, 0, 2*i + 1]:1.3f}')
            ClCm_tuple = [(Cm, Cl) for Cl, Cm in
                          zip(Cl_base[:, :, 2*i + 1].flatten(),
                          Cm_base[:, :, 2*i + 1].flatten())]
            alpha_Cn = alphashape.alphashape(ClCm_tuple, 1.5)
            ax_ClCm.add_patch(PolygonPatch(alpha_Cn, fc='None',
                                           ec='k', ls=linestyles[i]))
            bire_eq = optimize.minimize(find_moment, [0., 0., 0.],
                                        args=(Cn_base[0, 0, 2*i + 1], f_Cnbire),
                                        method='Nelder-Mead').x
            print(Cn_base[0, 0, 2*i + 1] - f_Cnbire(bire_eq))
            Cl_pts = np.array([[f_Clbire([a, b, bire_eq[2]])[0] for a in
                                da_range] for b in de_range]).flatten()
            Cm_pts = np.array([[f_Cmbire([a, b, bire_eq[2]])[0] for a in
                                da_range] for b in de_range]).flatten()
            ClCm_tuple = [(Cm, Cl) for Cl, Cm in zip(Cl_pts, Cm_pts)]
            alpha_Cn = alphashape.alphashape(ClCm_tuple, alphas[i])
            ax_ClCm.add_patch(PolygonPatch(alpha_Cn, fc='None',
                                           ec='0.5', ls=linestyles[i]))
            dummy_lines.append(ax_ClCm.plot([], [], c='k',
                               ls=linestyles[i])[0])
        fig_ClCm.legend([dummy_lines[i] for i in range(N//2)],
                        Cn_legend, loc='upper right', fontsize=16)
        ax_ClCm.set_ylabel(r'\textbf{Rolling Moment Coefficient, }\boldmath$C_\ell$',
                           fontsize=16)
        ax_ClCm.set_xlabel(r'\textbf{Pitching Moment Coefficient, }\boldmath$C_m$',
                           fontsize=16)
        ylims = (-0.06, 0.06)
        dy = {'major': 0.04, 'minor': 0.04/4}
        xlims = (-0.5, 0.5)
        dx = {'major': 0.2, 'minor': 0.2/4}
        ax_ClCm = pretty_plot(ax_ClCm, xlims, ylims, dx, dy)
        ax_ClCm.grid()
        ax_ClCm.set_aspect('equal')
        plt.tight_layout()
        plt.savefig(f'./AMS/Cl_Cm_{int(H):2d}_M{M:.1f}.pdf', dpi=1000)

        fig_CmCn, ax_CmCn = plt.subplots()
        alphas = [2.]*5
        Cl_legend = []
        dummy_lines = []
        for i in range(N//2):
            Cl_legend.append('$C_\ell = $' + f'{Cl_base[2*i + 1, 0, 0]:1.3f}')
```

```
                CmCn_tuple = [(Cm, Cn) for Cm, Cn in
                              zip(Cm_base[2*i + 1, :, :].flatten(),
                              Cn_base[2*i + 1, :, :].flatten())]
                alpha_Cl = alphashape.alphashape(CmCn_tuple, 1.5)
                ax_CmCn.add_patch(PolygonPatch(alpha_Cl, fc='None',
                                               ec='k', ls=linestyles[i]))
                bire_eq = optimize.minimize(find_moment, [0., 0., 0.],
                                            args=(Cl_base[2*i + 1, 0, 0],
                                            f_Clbire)).x
                print(Cl_base[2*i + 1, 0, 0] - f_Clbire(bire_eq))
                Cm_pts = np.array([[f_Cmbire([bire_eq[2], a, b])[0] for a in
                                    de_range] for b in dB_range]).flatten()
                Cn_pts = np.array([[f_Cnbire([bire_eq[2], a, b])[0] for a in
                                    de_range] for b in dB_range]).flatten()
                CmCn_tuple = [(Cm, Cn) for Cm, Cn in zip(Cm_pts, Cn_pts)]
                alpha_Cl = alphashape.alphashape(CmCn_tuple, alphas[i])
                ax_CmCn.add_patch(PolygonPatch(alpha_Cl, fc='None',
                                  ec='0.5', ls=linestyles[i]))
                dummy_lines.append(ax_CmCn.plot([], [], c='k', ls=linestyles[i])[0])
            fig_CmCn.legend([dummy_lines[i] for i in range(N//2)],
                            Cl_legend, loc='upper right', fontsize=16)
            ax_CmCn.set_ylabel(r'\textbf{Pitching Moment Coefficient, }\boldmath$C_m$',
                               fontsize=16)
            ax_CmCn.set_xlabel(r'\textbf{Yawing Moment Coefficient, }\boldmath$C_n$',
                               fontsize=16)
            ylims = (-0.175, 0.175)
            dy = {'major': 0.05, 'minor': 0.05/4}
            xlims = (-0.5, 0.5)
            dx = {'major': 0.2, 'minor': 0.2/4}
            ax_CmCn = pretty_plot(ax_CmCn, xlims, ylims, dx, dy)
            ax_CmCn.grid()
            ax_CmCn.set_aspect('equal')
            plt.tight_layout()
            plt.savefig(f'./AMS/Cm_Cn_{int(H):2d}_M{M:.1f}.pdf', dpi=1000)
        else:
            max_Clbase = np.max(Cl_base)
            max_Cmbase = np.max(Cm_base)
            max_Cnbase = np.max(Cn_base)
            max_Clbire = np.max(Cl_bire)
            max_Cmbire = np.max(Cm_bire)
            max_Cnbire = np.max(Cn_bire)
            min_Clbase = np.min(Cl_base)
            min_Cmbase = np.min(Cm_base)
            min_Cnbase = np.min(Cn_base)
            min_Clbire = np.min(Cl_bire)
            min_Cmbire = np.min(Cm_bire)
            min_Cnbire = np.min(Cn_bire)
            max_moments = [max_Clbase, max_Cmbase, max_Cnbase,
                           max_Clbire, max_Cmbire, max_Cnbire]
            min_moments = [min_Clbase, min_Cmbase, min_Cnbase,
                           min_Clbire, min_Cmbire, min_Cnbire]
            return max_moments, min_moments

if __name__ == "__main__":
    plt.close('all')
    H = 30000.
```

```
    M = 0.8
    generate_trim = False
    generate_data = False
    # generate_AMS_data(H, M, generate_trim=generate_trim,
                        generate_data=generate_data)

    cases = [(1000., 0.2), (1000., 0.8), (15000., 0.2),
             (15000., 0.6), (30000., 0.8)]
    max_moments = np.zeros((len(cases), 6))
    min_moments = np.zeros((len(cases), 6))
    for i in range(len(cases)):
            max_moments[i, :], min_moments[i, :] = generate_AMS_data(cases[i][0],
                                                                     cases[i][1],
                                                                     generate_trim,
                                                                     generate_data,
                                                                     False)
    np.save('./AMS/Data/max_moments.npy', max_moments)
    np.save('./AMS/Data/min_moments.npy', min_moments)
```

Yawing Moment Versus Drag Study

```
import numpy as np
from bire_aero import BIREAero
from f16_aero import F16Aero
import scipy.optimize as optimize
import machupX as mx


def find_moment(deltas, C_moment, function):
    [delta_e, delta_B] = deltas
    results = function.aero_results(alpha, beta, pbar, qbar, rbar,
                                    da, delta_e, delta_B)
    est_moment = results[5]
    return (est_moment - C_moment)**2

def generate_data(params):
    alpha = params[0]
    beta = params[1]
    d_e = params[2]
    d_a = params[3]
    d_r = params[4]
    p = params[5]
    q = params[6]
    r = params[7]
    rates = [p, q, r]
    my_scene.set_aircraft_state(state={"alpha": alpha,
                                       "beta": beta,
                                       "angular_rates": rates,
                                       "velocity": 222.5211})
    my_scene.set_aircraft_control_state(control_state={"elevator": d_e,
                                                       "aileron": d_a,
                                                       "rudder": d_r})
    x = my_scene.solve_forces(**forces_options)["F16"]["total"]
    fm = [x['CD'], x['CS'], x['CL'], x['Cl'], x['Cm'], x['Cn']]
    return (*params, *fm)


b_aero = BIREAero()
f_aero = F16Aero()
my_scene = mx.Scene('./F16_input.json')

forces_options = {'body_frame': True,
                  'stab_frame': False,
                  'wind_frame': True,
                  'dimensional': False,
                  'verbose': False}

N = 50
max_dr = 30.*np.pi/180.
dr_range = np.linspace(-max_dr, max_dr, N)
Cn_base = np.zeros(N)
CD_base = np.zeros(N)
CD_twist = np.zeros(N)
CD_bire = np.zeros(N)

```

```
alpha = 0.
beta = 0.
pbar = 0.
qbar = 0.
rbar = 0.
da = 0.
de = 0.

for i in range(N):
    base_results = generate_data([alpha, beta, de, da, dr_range[i]*180/np.pi,
                                  pbar, qbar, rbar])
    CD_base[i] = base_results[8]
    Cn_base[i] = base_results[-1]
    res = optimize.minimize(find_moment, [0., 0.], args=(Cn_base[i], b_aero),
                            method='Nelder-Mead').x
    bire_results = b_aero.aero_results(alpha, beta, pbar, qbar, rbar, da,
                                       res[0], res[1])
    CD_bire[i] = bire_results[2]
    Cn_bire = bire_results[5]
    print('de', res[0]*180/np.pi)
    print('dB', res[1]*180/np.pi)
```

C.6 Linearized Controller Analysis

Linearization of the Baseline Aircraft

```
import numpy as np
from f16_aero import F16Aero
import aero_trim as trim
from stdatmos import stdatm_english
from control import ctrb, lqr, place
import matplotlib.pyplot as plt
from os.path import exists

class Lin_Results:
    def __init__(self, N, M):
        self.A = np.zeros((N, N))
        self.B = np.zeros((N, M))
        self.C = np.zeros((N, N))
        self.K = np.zeros((M, N))
        self.eigs = np.zeros(N)
        self.aircraft = "F16"

class LinearizationBaseline:
    def __init__(self, props, aero_dir='./', N=8, M=4):
        self.N = N
        self.M = M
        self.x_hat = np.zeros(N + M)
        self.u_hat = np.zeros(M)
        self.alpha_hat = 0.
        self.beta_hat = 0.
        self.V_hat = 0.
        self.props = props
        self.rho = props.rho
        self.rho_0 = props.rho_0
        self.S_w = props.S_w
        self.b_w = props.b_w
        self.c_w = props.c_w
        self.W = props.W
        self.g = props.g
        self.aero_dir = aero_dir
        self.tc_tau = 0.05
        self.tc_da = 0.05
        self.tc_de = 0.05
        self.tc_dr = 0.05
        self.rate_da = 80.*np.pi/180.
        self.rate_de = 60*np.pi/180.
        self.rate_dr = 120*np.pi/180.

    def set_linearization_point(self, x_hat, u_hat, alpha_hat, beta_hat, FM_hat,
                                cg_shift):
        self.x_hat = x_hat
        self.u_hat = u_hat
        self.alpha_hat = alpha_hat
        self.beta_hat = beta_hat
        self.V_hat = np.sqrt(np.sum(np.square(self.x_hat[:3])))
```

```
        [self.CD_hat, self.CS_hat, self.CL_hat,
         self.Cl_hat, self.Cm_hat, self.Cn_hat] = FM_hat
        self.I_inv = self._I_inv()
        self._W_matrix()
        self.dVinv_dz = self._dVinv_dz()
        self.dV_dz = self._dV_dz()
        self.da_dz = self._dalpha_dz()
        self.db_dz = self._dbeta_dz()
        self.Dx = cg_shift[0]
        self.Dy = cg_shift[1]
        self.Dz = cg_shift[2]
        self.dp_dz = np.array([0., 0., 0., 1., 0., 0., 0., 0.])
        self.dq_dz = np.array([0., 0., 0., 0., 1., 0., 0., 0.])
        self.dr_dz = np.array([0., 0., 0., 0., 0., 1., 0., 0.])
        self.dde_du = np.array([0., 0., 1., 0.])
        self.dda_du = np.array([0., 1., 0., 0.])
        self.dtau_du = np.array([1., 0., 0., 0.])
        self.ddr_du = np.array([0., 0., 0., 1.])
        self.dde2_du = 2.*self.u_hat[2]*self.dde_du
        aero = F16Aero(self.aero_dir)
        self.CL_0 = aero.CL0
        self.CL_a = aero.CLa
        self.CL_q = aero.CLq
        self.CL_de = aero.CLde
        self.CL1_hat = self.CL_0 + self.CL_a*self.alpha_hat
        self.CS_b = aero.CSb
        self.CS_Lp = aero.CSLp
        self.CS_p = aero.CSp
        self.CS_r = aero.CSr
        self.CS_da = aero.CSda
        self.CS_dr = aero.CSdr
        self.CD_L = aero.CDL
        self.CD_L2 = aero.CDL2
        self.CD_S2 = aero.CDS2
        self.CS1_hat = self.CS_b*self.beta_hat
        self.CD_Sp = aero.CDSp
        self.CD_L2q = aero.CDL2q
        self.CD_Lq = aero.CDLq
        self.CD_q = aero.CDq
        self.CD_Sr = aero.CDSr
        self.CD_Sda = aero.CDSda
        self.CD_Lde = aero.CDLde
        self.CD_de = aero.CDde
        self.CD_Sdr = aero.CDSdr
        self.CD_de2 = aero.CDde2
        self.Cl_b = aero.Clb
        self.Cl_p = aero.Clp
        self.Cl_Lr = aero.ClLr
        self.Cl_r = aero.Clr
        self.Cl_da = aero.Clda
        self.Cl_dr = aero.Cldr
        self.Cm_a = aero.Cma
        self.Cm_q = aero.Cmq
        self.Cm_de = aero.Cmde
        self.Cn_b = aero.Cnb
        self.Cn_Lp = aero.CnLp
```

```
            self.Cn_p = aero.Cnp
            self.Cn_r = aero.Cnr
            self.Cn_Lda = aero.CnLda
            self.Cn_da = aero.Cnda
            self.Cn_dr = aero.Cndr

        def _det_I(self):
            props = self.props
            C1 = props.Ixx*(props.Iyy*props.Izz - props.Iyz*props.Izy)
            C2 = props.Ixy*(props.Iyx*props.Izz + props.Iyz*props.Izx)
            C3 = props.Ixz*(props.Iyx*props.Izy + props.Iyy*props.Izx)
            return C1 - C2 - C3

        def _I_inv(self):
            props = self.props
            det_I = self._det_I()
            I_inv = np.zeros((3, 3))
            I_inv[0, 0] = props.Iyy*props.Izz - props.Iyz*props.Izy
            I_inv[0, 1] = props.Ixy*props.Izz + props.Ixz*props.Izy
            I_inv[0, 2] = props.Ixy*props.Iyz + props.Ixz*props.Iyy
            I_inv[1, 0] = props.Iyx*props.Izz + props.Iyz*props.Izx
            I_inv[1, 1] = props.Ixx*props.Izz - props.Ixz*props.Izx
            I_inv[1, 2] = props.Ixx*props.Iyz + props.Ixz*props.Iyz
            I_inv[2, 0] = props.Iyz*props.Izy + props.Iyy*props.Izx
            I_inv[2, 1] = props.Ixx*props.Izy + props.Ixy*props.Izx
            I_inv[2, 2] = props.Ixx*props.Iyy - props.Ixy*props.Iyx
            I_inv = I_inv/det_I
            return I_inv

        def _dz1_dz(self):
            dz1_dz = np.zeros(self.N)
            dFxdz = self._dFx_dz()
            dz1_dz = self.props.g/self.props.W*dFxdz
            dz1_dz[1] += self.x_hat[5]
            dz1_dz[2] -= self.x_hat[4]
            dz1_dz[4] -= self.x_hat[2]
            dz1_dz[5] += self.x_hat[1]
            dz1_dz[7] -= self.props.g*np.cos(self.x_hat[7])
            return dz1_dz

        def _dz2_dz(self):
            dz2_dz = np.zeros(self.N)
            dFydz = self._dFy_dz()
            dz2_dz = self.props.g/self.props.W*dFydz
            dz2_dz[0] -= self.x_hat[5]
            dz2_dz[2] += self.x_hat[3]
            dz2_dz[3] += self.x_hat[2]
            dz2_dz[5] -= self.x_hat[0]
            dz2_dz[6] += self.props.g*np.cos(self.x_hat[6])*np.cos(self.x_hat[7])
            dz2_dz[7] -= self.props.g*np.sin(self.x_hat[6])*np.sin(self.x_hat[7])
            return dz2_dz

        def _dz3_dz(self):
            dz3_dz = np.zeros(self.N)
            dFzdz = self._dFz_dz()
            dz3_dz = self.props.g/self.props.W*dFzdz
```

```
            dz3_dz[0] += self.x_hat[4]
            dz3_dz[1] -= self.x_hat[3]
            dz3_dz[3] -= self.x_hat[1]
            dz3_dz[4] += self.x_hat[0]
            dz3_dz[6] -= self.props.g*np.sin(self.x_hat[6])*np.cos(self.x_hat[7])
            dz3_dz[7] -= self.props.g*np.cos(self.x_hat[6])*np.sin(self.x_hat[7])
            return dz3_dz

        def _dz4_dz(self):
            dz4_dz = np.zeros(self.N)
            dMxdz = self._dMx_dz()
            dMydz = self._dMy_dz()
            dMzdz = self._dMz_dz()
            dM = np.array([dMxdz, dMydz, dMzdz])
            R = np.zeros((3, self.N + self.M))
            R = dM + self.W_mat
            dz4_dz = np.matmul(self.I_inv, R)[0, :]
            return dz4_dz

        def _dz5_dz(self):
            dz5_dz = np.zeros(self.N)
            dMxdz = self._dMx_dz()
            dMydz = self._dMy_dz()
            dMzdz = self._dMz_dz()
            dM = np.array([dMxdz, dMydz, dMzdz])
            R = np.zeros((3, self.N + self.M))
            R = dM + self.W_mat
            dz5_dz = np.matmul(self.I_inv, R)[1, :]
            return dz5_dz

        def _dz6_dz(self):
            dz6_dz = np.zeros(self.N)
            dMxdz = self._dMx_dz()
            dMydz = self._dMy_dz()
            dMzdz = self._dMz_dz()
            dM = np.array([dMxdz, dMydz, dMzdz])
            R = np.zeros((3, self.N + self.M))
            R = dM + self.W_mat
            dz6_dz = np.matmul(self.I_inv, R)[2, :]
            return dz6_dz

        def _dz7_dz(self):
            dz7_dz = np.zeros(self.N)
            s_7 = np.sin(self.x_hat[6])
            c_7 = np.cos(self.x_hat[6])
            s_8 = np.sin(self.x_hat[7])
            c_8 = np.cos(self.x_hat[7])
            t_8 = s_8/c_8
            dz7_dz[3] = 1.
            dz7_dz[4] = s_7*t_8
            dz7_dz[5] = c_7*t_8
            dz7_dz[6] = t_8*(c_7*self.x_hat[4] - s_7*self.x_hat[5])
            dz7_dz[7] = s_7/(c_8**2)*self.x_hat[4] + c_7/(c_8**2)*self.x_hat[5]
            return dz7_dz

        def _dz8_dz(self):
```

```
            dz8_dz = np.zeros(self.N)
            s_7 = np.sin(self.x_hat[6])
            c_7 = np.cos(self.x_hat[6])
            dz8_dz[4] = c_7
            dz8_dz[5] = -s_7
            dz8_dz[6] = -s_7*self.x_hat[4] - c_7*self.x_hat[5]
            return dz8_dz

        def _dFx_dz(self):
            dFxdz = np.zeros(self.N)
            dCXdz = self._dCX_dz()
            dVdz = self.dV_dz
            dTxdz = self._dTX_dz()
            c_a = np.cos(self.alpha_hat)
            s_a = np.sin(self.alpha_hat)
            c_b = np.cos(self.beta_hat)
            s_b = np.sin(self.beta_hat)
            CX = -(self.CD_hat*c_a*c_b + self.CS_hat*c_a*s_b -
                   self.CL_hat*s_a)
            dFxdz = (0.5*self.rho*self.V_hat**2*self.S_w*dCXdz +
                     self.rho*self.V_hat*self.S_w*CX*dVdz + dTxdz)
            return dFxdz

        def _dFy_dz(self):
            dFydz = np.zeros(self.N)
            dCYdz = self._dCY_dz()
            dVdz = self.dV_dz
            c_b = np.cos(self.beta_hat)
            s_b = np.sin(self.beta_hat)
            CY = self.CS_hat*c_b - self.CD_hat*s_b
            dFydz = (0.5*self.rho*self.V_hat**2*self.S_w*dCYdz +
                     self.rho*self.V_hat*self.S_w*CY*dVdz)
            return dFydz

        def _dFz_dz(self):
            dFzdz = np.zeros(self.N)
            dCZdz = self._dCZ_dz()
            dVdz = self.dV_dz
            c_a = np.cos(self.alpha_hat)
            s_a = np.sin(self.alpha_hat)
            c_b = np.cos(self.beta_hat)
            s_b = np.sin(self.beta_hat)
            CZ = -(self.CD_hat*s_a*c_b + self.CS_hat*s_a*s_b +
                   self.CL_hat*c_a)
            dFzdz = (0.5*self.rho*self.V_hat**2*self.S_w*dCZdz +
                     self.rho*self.V_hat*self.S_w*CZ*dVdz)
            return dFzdz

        def _dMx_dz(self):
            dMxdz = np.zeros(self.N)
            dCldz = self._dCl_dz()
            dFzdz = self._dFz_dz()
            dFydz = self._dFy_dz()
            dVdz = self.dV_dz
            dy = self.Dy
            dz = self.Dz
```

```
        dMxdz = (0.5*self.rho*self.V_hat**2*self.S_w*self.b_w*dCldz +
                 self.rho*self.V_hat*self.S_w*self.b_w*self.Cl_hat*dVdz -
                 dFzdz*dy +
                 dFydz*dz)
        return dMxdz

    def _dMy_dz(self):
        dMydz = np.zeros(self.N)
        dCmdz = self._dCm_dz()
        dFzdz = self._dFz_dz()
        dFxdz = self._dFx_dz()
        dVdz = self.dV_dz
        dx = self.Dx
        dz = self.Dz
        dMydz = (0.5*self.rho*self.V_hat**2*self.S_w*self.c_w*dCmdz +
                 self.rho*self.V_hat*self.S_w*self.c_w*self.Cm_hat*dVdz -
                 dFzdz*dx +
                 dFxdz*dz)
        return dMydz

    def _dMz_dz(self):
        dMzdz = np.zeros(self.N)
        dCndz = self._dCn_dz()
        dFxdz = self._dFx_dz()
        dFydz = self._dFy_dz()
        dVdz = self.dV_dz
        dx = self.Dx
        dy = self.Dy
        dMzdz = (0.5*self.rho*self.V_hat**2*self.S_w*self.b_w*dCndz +
                 self.rho*self.V_hat*self.S_w*self.b_w*self.Cn_hat*dVdz -
                 dFydz*dx +
                 dFxdz*dy)
        return dMzdz

    def _dV_dz(self):
        dVdz = np.zeros(self.N)
        dVdz[0] = self.x_hat[0]/self.V_hat
        dVdz[1] = self.x_hat[1]/self.V_hat
        dVdz[2] = self.x_hat[2]/self.V_hat
        return dVdz

    def _dVinv_dz(self):
        dVinvdz = np.zeros(self.N)
        dVinvdz[0] = -self.x_hat[0]/self.V_hat**3
        dVinvdz[1] = -self.x_hat[1]/self.V_hat**3
        dVinvdz[2] = -self.x_hat[2]/self.V_hat**3
        return dVinvdz

    def _dTX_dz(self):
        dVdz = self._dV_dz()
        H = self.props.H
        a_mil = self.props.a_mil(H)
        T1_mil = self.props.T1_mil(H)
        T2_mil = self.props.T2_mil(H)
        rho_ratio = (self.rho/self.rho_0)
        dTmil_dz = rho_ratio**a_mil*(T1_mil*dVdz + 2.*T2_mil*self.V_hat*dVdz)
```

```
        if self.u_hat[0] < 0.77:
            a_idle = self.props.a_idle(H)
            T1_idle = self.props.T1_idle(H)
            T2_idle = self.props.T2_idle(H)
            dTidle_dz = rho_ratio**a_idle*(T1_idle*dVdz + 2.*T2_idle*self.V_hat*dVdz)
            P1 = 64.94*self.u_hat[0]/50.
            dTX_dz = P1*(dTmil_dz - dTidle_dz) + dTidle_dz
        else:
            a_max = self.props.a_max(H)
            T1_max = self.props.T1_max(H)
            T2_max = self.props.T2_max(H)
            dTmax_dz = rho_ratio**a_max*(T1_max*dVdz + 2.*T2_max*self.V_hat*dVdz)
            P1 = (217.38*self.u_hat[0] - 117.38 - 50.)/50.
            dTX_dz = P1*(dTmax_dz - dTmil_dz) + dTmil_dz
        return dTX_dz


    def _dCX_dz(self):
        dCXdz = np.zeros(self.N)
        dCDdz = self._dCD_dz()
        dCSdz = self._dCS_dz()
        dCLdz = self._dCL_dz()
        dadz = self.da_dz
        dbdz = self.db_dz
        c_a = np.cos(self.alpha_hat)
        s_a = np.sin(self.alpha_hat)
        c_b = np.cos(self.beta_hat)
        s_b = np.sin(self.beta_hat)
        CD = self.CD_hat
        CS = self.CS_hat
        CL = self.CL_hat
        dCXdz = (-dCDdz*c_a*c_b + CD*s_a*c_b*dadz + CD*c_a*s_b*dbdz -
                 dCSdz*c_a*s_b + CS*s_a*s_b*dadz - CS*c_a*c_b*dbdz +
                 dCLdz*s_a + CL*c_a*dadz)
        return dCXdz

    def _dCY_dz(self):
        dCYdz = np.zeros(self.N)
        dCDdz = self._dCD_dz()
        dCSdz = self._dCS_dz()
        dbdz = self.db_dz
        c_b = np.cos(self.beta_hat)
        s_b = np.sin(self.beta_hat)
        CD = self.CD_hat
        CS = self.CS_hat
        dCYdz = dCSdz*c_b - CS*s_b*dbdz - dCDdz*s_b - CD*c_b*dbdz
        return dCYdz

    def _dCZ_dz(self):
        dCZdz = np.zeros(self.N)
        dCDdz = self._dCD_dz()
        dCSdz = self._dCS_dz()
        dCLdz = self._dCL_dz()
        dadz = self.da_dz
        dbdz = self.db_dz
        c_a = np.cos(self.alpha_hat)
```

```
        s_a = np.sin(self.alpha_hat)
        c_b = np.cos(self.beta_hat)
        s_b = np.sin(self.beta_hat)
        CD = self.CD_hat
        CS = self.CS_hat
        CL = self.CL_hat
        dCZdz = (-dCDdz*s_a*c_b - CD*c_a*c_b*dadz + CD*s_a*s_b*dbdz -
                 dCSdz*s_a*s_b - CS*c_a*s_b*dadz - CS*s_a*c_b*dbdz -
                 dCLdz*c_a + CL*s_a*dadz)
        return dCZdz

    def _dalpha_dz(self):
        dadz = np.zeros(self.N)
        C1 = self.x_hat[0]**2 + self.x_hat[2]**2
        dadz[0] = -self.x_hat[2]/C1
        dadz[2] = self.x_hat[0]/C1
        return dadz

    def _dbeta_dz(self):
        dbdz = np.zeros(self.N)
        C1 = np.sqrt(self.x_hat[0]**2 + self.x_hat[2]**2)
        C2 = (self.V_hat**2)*C1
        dbdz[0] = -self.x_hat[1]*self.x_hat[0]/C2
        dbdz[1] = C1/(self.V_hat**2)
        dbdz[2] = -self.x_hat[1]*self.x_hat[2]/C2
        return dbdz

    def _dCL_dz(self):
        dCLdz = np.zeros(self.N)
        dCL1dz = self._dCL1_dz()
        dqbardz = self._dqbar_dz()
        dCLdz = dCL1dz + self.CL_q*dqbardz
        return dCLdz

    def _dCL1_dz(self):
        dCL1dz = self.CL_a*self.da_dz
        return dCL1dz

    def _dqbar_dz(self):
        dqdz = self.dq_dz
        dVidz = self.dVinv_dz
        dqbardz = dqdz*self.c_w/(2.*self.V_hat) + dVidz*self.c_w*self.x_hat[4]/2.
        return dqbardz

    def _dCS_dz(self):
        dCSdz = np.zeros(self.N)
        dCS1dz = self._dCS1_dz()
        dCL1dz = self._dCL1_dz()
        xb_4 = self.b_w*self.x_hat[3]/(2.*self.V_hat)
        dpbardz = self._dpbar_dz()
        drbardz = self._drbar_dz()
        dCSdz = (dCS1dz +
                 self.CS_Lp*dCL1dz*xb_4 +
                 (self.CS_Lp*self.CL1_hat + self.CS_p)*dpbardz +
                 self.CS_r*drbardz)
        return dCSdz
```

```

    def _dCS1_dz(self):
        dCS1dz = self.CS_b*self.db_dz
        return dCS1dz

    def _dpbar_dz(self):
        dpdz = self.dp_dz
        dVidz = self.dVinv_dz
        dpbardz = dpdz*self.b_w/(2.*self.V_hat) + dVidz*self.b_w*self.x_hat[3]/2.
        return dpbardz

    def _drbar_dz(self):
        drdz = self.dr_dz
        dVidz = self.dVinv_dz
        drbardz = drdz*self.b_w/(2.*self.V_hat) + dVidz*self.b_w*self.x_hat[5]/2.
        return drbardz

    def _dCD_dz(self):
        dCDdz = np.zeros(self.N)
        dCL1dz = self._dCL1_dz()
        dCS1dz = self._dCS1_dz()
        dCL12dz = 2.*self.CL1_hat*dCL1dz
        dCS12dz = 2.*self.CS1_hat*dCS1dz
        xb_4 = self.b_w*self.x_hat[3]/(2.*self.V_hat)
        xb_5 = self.c_w*self.x_hat[4]/(2.*self.V_hat)
        xb_6 = self.b_w*self.x_hat[5]/(2.*self.V_hat)
        dpbardz = self._dpbar_dz()
        dqbardz = self._dqbar_dz()
        drbardz = self._drbar_dz()
        CL1 = self.CL1_hat
        CS1 = self.CS1_hat
        dCDdz = (self.CD_L*dCL1dz + self.CD_L2*dCL12dz + self.CD_S2*dCS12dz +
                 self.CD_Sp*dCS1dz*xb_4 + self.CD_Sp*CS1*dpbardz +
                 (self.CD_L2q*dCL12dz + self.CD_Lq*dCL1dz)*xb_5 +
                 (self.CD_L2q*CL1**2 + self.CD_Lq*CL1 + self.CD_q)*dqbardz +
                 self.CD_Sr*dCS1dz*xb_6 + self.CD_Sr*CS1*drbardz +
                 self.CD_Sda*dCS1dz*self.u_hat[1] +
                 self.CD_Lde*dCL1dz*self.u_hat[2] +
                 self.CD_Sdr*dCS1dz*self.u_hat[3])
        return dCDdz

    def _dCl_dz(self):
        dCldz = np.zeros(self.N)
        dbdz = self.db_dz
        dpbardz = self._dpbar_dz()
        drbardz = self._drbar_dz()
        dCL1dz = self._dCL1_dz()
        xb_6 = self.b_w*self.x_hat[5]/(2.*self.V_hat)
        CL1 = self.CL1_hat
        dCldz = (self.Cl_b*dbdz + self.Cl_p*dpbardz + self.Cl_Lr*dCL1dz*xb_6 +
                 (self.Cl_Lr*CL1 + self.Cl_r)*drbardz)
        return dCldz

    def _dCm_dz(self):
        dCmdz = np.zeros(self.N)
        dadz = self.da_dz
```

```
            dqbardz = self._dqbar_dz()
            dCmdz = self.Cm_a*dadz + self.Cm_q*dqbardz
            return dCmdz

        def _dCn_dz(self):
            dCndz = np.zeros(self.N)
            dbdz = self.db_dz
            dpbardz = self._dpbar_dz()
            drbardz = self._drbar_dz()
            dCL1dz = self._dCL1_dz()
            xb_4 = self.b_w*self.x_hat[3]/(2.*self.V_hat)
            CL1 = self.CL1_hat
            dCndz = (self.Cn_b*dbdz + self.Cn_Lp*dCL1dz*xb_4 +
                     (self.Cn_Lp*CL1 + self.Cn_p)*dpbardz + self.Cn_r*drbardz +
                     self.Cn_Lda*dCL1dz*self.u_hat[1])
            return dCndz

        def _dz1_du(self):
            dFxdu = self._dFx_du()
            dz1du = self.g/self.W*dFxdu
            return dz1du

        def _dz2_du(self):
            dFydu = self._dFy_du()
            dz2du = self.g/self.W*dFydu
            return dz2du

        def _dz3_du(self):
            dFzdu = self._dFz_du()
            dz3du = self.g/self.W*dFzdu
            return dz3du

        def _dz4_du(self):
            dMxdu = self._dMx_du()
            dMydu = self._dMy_du()
            dMzdu = self._dMz_du()
            dM = np.array([dMxdu, dMydu, dMzdu])
            dz4du = np.matmul(self.I_inv, dM)[0, :]
            return dz4du

        def _dz5_du(self):
            dMxdu = self._dMx_du()
            dMydu = self._dMy_du()
            dMzdu = self._dMz_du()
            dM = np.array([dMxdu, dMydu, dMzdu])
            dz5du = np.matmul(self.I_inv, dM)[1, :]
            return dz5du

        def _dz6_du(self):
            dMxdu = self._dMx_du()
            dMydu = self._dMy_du()
            dMzdu = self._dMz_du()
            dM = np.array([dMxdu, dMydu, dMzdu])
            dz6du = np.matmul(self.I_inv, dM)[2, :]
            return dz6du
```

```
    def _dCL_du(self):
        dCLdu = self.CL_de*self.dde_du
        return dCLdu

    def _dCS_du(self):
        dCSdu = self.CS_da*self.dda_du + self.CS_dr*self.ddr_du
        return dCSdu

    def _dCD_du(self):
        dCDdu = (self.CD_Sda*self.CS1_hat*self.dda_du +
                 (self.CD_Lde*self.CL1_hat + self.CD_de)*self.dde_du +
                 self.CD_de2*self.dde2_du +
                 self.CD_Sdr*self.CS1_hat*self.ddr_du)
        return dCDdu

    def _dCl_du(self):
        dCldu = self.Cl_da*self.dda_du + self.Cl_dr*self.ddr_du
        return dCldu

    def _dCm_du(self):
        dCmdu = self.Cm_de*self.dde_du
        return dCmdu

    def _dCn_du(self):
        dCndu = ((self.Cn_Lda*self.CL1_hat + self.Cn_da)*self.dda_du +
                 self.Cn_dr*self.ddr_du)
        return dCndu

    def _dTx_du(self):
        a_mil = self.props.a_mil(self.props.H)
        T0_mil = self.props.T0_mil(self.props.H)
        T1_mil = self.props.T1_mil(self.props.H)
        T2_mil = self.props.T2_mil(self.props.H)
        V = self.props.V
        T_mil = (self.rho/self.rho_0)**a_mil*(T0_mil + T1_mil*V + T2_mil*V**2)
        if self.u_hat[0] < 0.77:
            a_idle = self.props.a_idle(self.props.H)
            T0_idle = self.props.T0_idle(self.props.H)
            T1_idle = self.props.T1_idle(self.props.H)
            T2_idle = self.props.T2_idle(self.props.H)
            T_idle = (self.rho/self.rho_0)**a_idle*(T0_idle + T1_idle*V +
                                                    T2_idle*V**2)
            dTxdu = 64.94/50.*(T_mil - T_idle)
        else:
            a_max = self.props.a_max(self.props.H)
            T0_max = self.props.T0_max(self.props.H)
            T1_max = self.props.T1_max(self.props.H)
            T2_max = self.props.T2_max(self.props.H)
            T_max = (self.rho/self.rho_0)**a_max*(T0_max + T1_max*V + T2_max*V**2)
            dTxdu = 217.38/50.*(T_max - T_mil)
        return dTxdu

    def _dCX_du(self):
        dCDdu = self._dCD_du()
        dCSdu = self._dCS_du()
        dCLdu = self._dCL_du()
```

```
        c_a = np.cos(self.alpha_hat)
        s_a = np.sin(self.alpha_hat)
        c_b = np.cos(self.beta_hat)
        s_b = np.sin(self.beta_hat)
        dCXdu = -(dCDdu*c_a*c_b + dCSdu*c_a*s_b - dCLdu*s_a)
        return dCXdu

    def _dCY_du(self):
        dCDdu = self._dCD_du()
        dCSdu = self._dCS_du()
        c_b = np.cos(self.beta_hat)
        s_b = np.sin(self.beta_hat)
        dCYdu = dCSdu*c_b - dCDdu*s_b
        return dCYdu

    def _dCZ_du(self):
        dCDdu = self._dCD_du()
        dCSdu = self._dCS_du()
        dCLdu = self._dCL_du()
        c_a = np.cos(self.alpha_hat)
        s_a = np.sin(self.alpha_hat)
        c_b = np.cos(self.beta_hat)
        s_b = np.sin(self.beta_hat)
        dCZdu = -(dCDdu*s_a*c_b + dCSdu*s_a*s_b + dCLdu*c_a)
        return dCZdu

    def _dFx_du(self):
        dCXdu = self._dCX_du()
        dTxdu = self._dTx_du()
        dFxdu = 0.5*self.rho*self.V_hat**2*self.S_w*dCXdu + dTxdu
        return dFxdu

    def _dFy_du(self):
        dCYdu = self._dCY_du()
        dFydu = 0.5*self.rho*self.V_hat**2*self.S_w*dCYdu
        return dFydu

    def _dFz_du(self):
        dCZdu = self._dCZ_du()
        dFzdu = 0.5*self.rho*self.V_hat**2*self.S_w*dCZdu
        return dFzdu

    def _dMx_du(self):
        dCldu = self._dCl_du()
        dFzdu = self._dFz_du()
        dFydu = self._dFy_du()
        dMxdu = (0.5*self.rho*self.V_hat**2*self.S_w*self.b_w*dCldu -
                 dFzdu*self.Dy +
                 dFydu*self.Dz)
        return dMxdu

    def _dMy_du(self):
        dCmdu = self._dCm_du()
        dFzdu = self._dFz_du()
        dFxdu = self._dFx_du()
        dMydu = (0.5*self.rho*self.V_hat**2*self.S_w*self.c_w*dCmdu -
```

```
                      dFzdu*self.Dx +
                      dFxdu*self.Dz)
        return dMydu

    def _dMz_du(self):
        dCndu = self._dCn_du()
        dFydu = self._dFy_du()
        dFxdu = self._dFx_du()
        dMzdu = (0.5*self.rho*self.V_hat**2*self.S_w*self.b_w*dCndu -
                      dFydu*self.Dx +
                      dFxdu*self.Dy)
        return dMzdu

    def _W_matrix(self):
        self.W_mat = np.zeros((3, self.N))
        self.W_mat[:, 3] = np.array([self.props.Ixz*self.x_hat[4] -
                                     self.props.Ixy*self.x_hat[5],
                                     (self.props.Izz - self.props.Ixx)*self.x_hat[5] -
                                     2.*self.props.Ixz*self.x_hat[3] -
                                     self.props.Iyz*self.x_hat[4],
                                     (self.props.Ixx - self.props.Iyy)*self.x_hat[4] +
                                     2.*self.props.Ixy*self.x_hat[3] +
                                     self.props.Iyz*self.x_hat[5]])
        self.W_mat[:, 4] = np.array([(self.props.Iyy - self.props.Izz)*self.x_hat[5] -
                                     2.*self.props.Iyz*self.x_hat[4] +
                                     self.props.Ixz*self.x_hat[3],
                                     self.props.Ixy*self.x_hat[5] -
                                     self.props.Iyz*self.x_hat[3],
                                     (self.props.Ixx - self.props.Iyy)*self.x_hat[3] -
                                     2.*self.props.Ixy*self.x_hat[4] -
                                     self.props.Ixz*self.x_hat[5]])
        self.W_mat[:, 5] = np.array([(self.props.Iyy - self.props.Izz)*self.x_hat[4] +
                                     2.*self.props.Iyz*self.x_hat[5] -
                                     self.props.Ixy*self.x_hat[3],
                                     (self.props.Izz - self.props.Ixx)*self.x_hat[3] +
                                     2.*self.props.Ixz*self.x_hat[5] +
                                     self.props.Ixy*self.x_hat[4],
                                     self.props.Iyz*self.x_hat[3] -
                                     self.props.Ixz*self.x_hat[4]])


    def create_A_matrix(self):
        A = np.zeros((self.N, self.N))
        A[0, :] = self._dz1_dz()
        A[1, :] = self._dz2_dz()
        A[2, :] = self._dz3_dz()
        A[3, :] = self._dz4_dz()
        A[4, :] = self._dz5_dz()
        A[5, :] = self._dz6_dz()
        A[6, :] = self._dz7_dz()
        A[7, :] = self._dz8_dz()
        return A

    def create_B_matrix(self):
        B = np.zeros((self.N, self.M))
        B[0, :] = self._dz1_du()
```

```
            B[1, :] = self._dz2_du()
            B[2, :] = self._dz3_du()
            B[3, :] = self._dz4_du()
            B[4, :] = self._dz5_du()
            B[5, :] = self._dz6_du()
            return B

        def create_C_matrix(self):
            C = np.eye(self.N)
            return C

        def create_E_matrix(self):
            E = np.zeros((self.N, 3))
            E[0, 0] = 1.
            E[1, 1] = 1.
            E[2, 2] = 1.
            return E

    def create_feedback_control(trim_solution, V, H, Gamma, cg_shift,
                                p=-np.arange(1., 9.), lqr_flag=True,
                                Q=np.eye(8), R=np.eye(4),
                                N=np.zeros((8, 4))):
        aero_dir = '/home/christian/Python Projects/AFRL BIRE/Static Analysis/main/'
        x_hat = trim_solution.states
        alpha_hat = trim_solution.x[1]
        beta_hat = trim_solution.x[2]
        u_hat = trim_solution.inputs
        FM_hat = trim_solution.FM
        props = trim.AircraftProperties(V, H, Gamma, aero_dir)
        # system =
        linearization = LinearizationBaseline(props, aero_dir)
        linearization.set_linearization_point(x_hat, u_hat, alpha_hat, beta_hat, FM_hat,
                                              cg_shift)
        A = linearization.create_A_matrix()
        B = linearization.create_B_matrix()
        C = linearization.create_C_matrix()
        D = np.zeros((linearization.N, linearization.M))
        E = linearization.create_E_matrix()
        G = ctrb(A, B)
        print(np.linalg.matrix_rank(G))
        if lqr_flag:
            K, S, E = lqr(A, B, Q, R, N)
        else:
            K = place(A, B, p)
        eig_check, v_check = np.linalg.eig(A - np.matmul(B, K))
        assert all(np.real(eig_check) < 0.)
        lin_res = Lin_Results(linearization.N, linearization.M)
        lin_res.A = A
        lin_res.B = B
        lin_res.C = C
        lin_res.D = D
        lin_res.K = K
        lin_res.E = E
        lin_res.eigs = eig_check
        return lin_res
```

```
if __name__ == "__main__":
    H = 15000.
    a = stdatm_english(H)[-1]
    M = 0.6
    V = M*a
    b_w = 30.
    c_w = 11.32
    gamma = np.deg2rad(0.)
    phi = np.deg2rad(0.)
    Gamma = 0.1
    cg_shift = [0., 0., 0.]
    aero_dir = '/home/christian/Python Projects/AFRL BIRE/Static Analysis/main/'
    trim_solution = trim.trim(V, H, gamma, phi, Gamma, fixed_point=False,
                              aero_dir=aero_dir)
    x_hat = trim_solution.states
    alpha_hat = trim_solution.x[1]
    beta_hat = trim_solution.x[2]
    u_hat = trim_solution.inputs
    FM_hat = trim_solution.FM
    props = trim.AircraftProperties(V, H, Gamma, aero_dir)
    linearization = LinearizationBaseline(props, aero_dir)
    linearization.set_linearization_point(x_hat, u_hat, alpha_hat, beta_hat,
                                          FM_hat, cg_shift)
    A = linearization.create_A_matrix()
    B = linearization.create_B_matrix()
    C = linearization.create_C_matrix()
```

Linearization of the BIRE Aircraft

```
import numpy as np
from bire_aero import BIREAero
import aero_trim as trim
from stdatmos import stdatm_english
from control import ctrb, lqr
import matplotlib.pyplot as plt
import json
from os.path import exists
import pickle

class Lin_Results:
    def __init__(self, N, M):
        self.A = np.zeros((N, N))
        self.B = np.zeros((N, M))
        self.C = np.zeros((N, N))
        self.K = np.zeros((M, N))
        self.eigs = np.zeros(N)
        self.aircraft = "BIRE"

class LinearizationBIRE:
    def __init__(self, props, aero_dir='./', N=8, M=4):
        self.N = N
        self.M = M
        self.x_hat = np.zeros(N)
        self.u_hat = np.zeros(M)
        self.alpha_hat = 0.
        self.beta_hat = 0.
        self.V_hat = 0.
        self.props = props
        self.rho = props.rho
        self.rho_0 = props.rho_0
        self.S_w = props.S_w
        self.b_w = props.b_w
        self.c_w = props.c_w
        self.W = props.W
        self.g = props.g
        self.aero_dir = aero_dir
        I_model = json.load(open('./bire_inertia_model.json'))
        Ixx = I_model["Ixx"]
        Iyy = I_model["Iyy"]
        Izz = I_model["Izz"]
        Ixz = I_model["Ixz"]
        Ixy = I_model["Ixy"]
        Iyz = I_model["Iyz"]
        self.I_xx = lambda dB : Ixx["A"]*np.sin(Ixx["w"]*dB + Ixx["phi"]) + Ixx["z"]
        self.I_yy = lambda dB : Iyy["A"]*np.sin(Iyy["w"]*dB + Iyy["phi"]) + Iyy["z"]
        self.I_zz = lambda dB : Izz["A"]*np.sin(Izz["w"]*dB + Izz["phi"]) + Izz["z"]
        self.I_yz = lambda dB : Iyz["A"]*np.abs(np.sin(Iyz["w"]*dB + Iyz["phi"])) +
                                Iyz["z"]
        self.I_xy = lambda dB : Ixy["A"]*np.sin(Ixy["w"]*dB + Ixy["phi"]) + Ixy["z"]
        self.I_xz = lambda dB : Ixz["A"]*np.sin(Ixz["w"]*dB + Ixz["phi"]) + Ixz["z"]
        self.dI_xx = lambda dB : np.array([0., 0., 0.,
                                           Ixx["A"]*Ixx["w"]*np.cos(Ixx["w"]*dB +
                                                                     Ixx["phi"])])
```

```
        self.dI_yy = lambda dB : np.array([0., 0., 0.,
                                          Iyy["A"]*Iyy["w"]*np.cos(Iyy["w"]*dB +
                                                                   Iyy["phi"])])
        self.dI_zz = lambda dB : np.array([0., 0., 0.,
                                          Izz["A"]*Izz["w"]*np.cos(Izz["w"]*dB +
                                                                   Izz["phi"])])
        self.dI_xy = lambda dB : np.array([0., 0., 0.,
                                          Ixy["A"]*Ixy["w"]*np.cos(Ixy["w"]*dB +
                                                                   Ixy["phi"])])
        self.dI_xz = lambda dB : np.array([0., 0., 0.,
                                          Ixz["A"]*Ixz["w"]*np.cos(Ixz["w"]*dB +
                                                                   Ixz["phi"])])
        self.tc_tau = 0.05
        self.tc_da = 0.05
        self.tc_de = 0.05
        self.tc_dr = 0.05
        self.rate_da = 80.*np.pi/180.
        self.rate_de = 60*np.pi/180.
        self.rate_dB = 120*np.pi/180.

    def dI_yz(self, dB):
        I_model = json.load(open('./bire_inertia_model.json'))
        Iyz = I_model["Iyz"]
        if dB == 0.:
            dI_yz = np.array([0., 0., 0., 0.])
        elif abs(dB) == np.pi:
            dI_yz = np.array([0., 0., 0., 0.])
        else:
            dI_yz = np.array([0., 0., 0.,
                              Iyz["A"]*Iyz["w"]*np.sin(2.*Iyz["w"]*dB +
                                                       Iyz["phi"])/
                                                       np.abs(np.sin(Iyz["w"]*dB))])
        return dI_yz


    def set_linearization_point(self, x_hat, u_hat, alpha_hat, beta_hat, FM_hat,
                                cg_shift):
        self.x_hat = x_hat
        self.u_hat = u_hat
        self.dB_hat = self.u_hat[3]
        self.alpha_hat = alpha_hat
        self.beta_hat = beta_hat
        self.V_hat = np.sqrt(np.sum(np.square(self.x_hat[:3])))
        [self.CD_hat, self.CS_hat, self.CL_hat,
         self.Cl_hat, self.Cm_hat, self.Cn_hat] = FM_hat
        self.I_inv = self._I_inv(self.dB_hat)
        self._W_matrix()
        self.dVinv_dz = self._dVinv_dz()
        self.dV_dz = self._dV_dz()
        self.da_dz = self._dalpha_dz()
        self.db_dz = self._dbeta_dz()
        self.Dx = cg_shift[0]
        self.Dy = cg_shift[1]
        self.Dz = cg_shift[2]
        dim_const = 0.5*self.rho*self.V_hat**2*self.S_w
        CZ = -(self.CD_hat*np.sin(self.alpha_hat)*np.cos(self.beta_hat) +
```

```
                self.CS_hat*np.sin(self.alpha_hat)*np.sin(self.beta_hat) +
                self.CL_hat*np.cos(self.alpha_hat))
        CY = self.CS_hat*np.cos(self.beta_hat) - self.CD_hat*np.sin(self.beta_hat)
        CX = -(self.CD_hat*np.cos(self.alpha_hat)*np.cos(self.beta_hat) +
                self.CS_hat*np.cos(self.alpha_hat)*np.sin(self.beta_hat) -
                self.CL_hat*np.sin(self.alpha_hat))
        Tx = trim.thrust(self.u_hat[0], self.V_hat, self.props)
        FX = dim_const*CX + self.u_hat[0]*Tx
        FY = dim_const*CY
        FZ = dim_const*CZ
        self.Mx_hat = dim_const*self.b_w*self.Cl_hat - FZ*self.Dy + FY*self.Dz
        self.My_hat = dim_const*self.c_w*self.Cm_hat - FZ*self.Dx + FX*self.Dz
        self.Mz_hat = dim_const*self.b_w*self.Cn_hat - FY*self.Dx + FX*self.Dy
        self.dp_dz = np.array([0., 0., 0., 1., 0., 0., 0., 0.])
        self.dq_dz = np.array([0., 0., 0., 0., 1., 0., 0., 0.])
        self.dr_dz = np.array([0., 0., 0., 0., 0., 1., 0., 0.])
        self.dde_du = np.array([0., 0., 1., 0.])
        self.dda_du = np.array([0., 1., 0., 0.])
        self.dtau_du = np.array([1., 0., 0., 0.])
        self.ddB_du = np.array([0., 0., 0., 1.])
        self.dde2_du = 2.*self.u_hat[2]*self.dde_du
        aero = BIREAero(self.aero_dir)
        aero.evaluate_coeffs(self.dB_hat)
        aero.evaluate_derivatives(self.dB_hat)

        self.CL_0 = aero.CL0
        self.CL_a = aero.CLa
        self.CL_b = aero.CLb
        self.CL_p = aero.CLp
        self.CL_q = aero.CLq
        self.CL_r = aero.CLr
        self.CL_da = aero.CLda
        self.CL_de = aero.CLde
        self.CL1_hat = self.CL_0 + self.CL_a*self.alpha_hat

        self.CS_0 = aero.CS0
        self.CS_a = aero.CSa
        self.CS_b = aero.CSb
        self.CS1_hat = self.CS_0 + self.CS_b*self.beta_hat
        self.CS_Lp = aero.CSLp
        self.CS_p = aero.CSp
        self.CS_q = aero.CSq
        self.CS_r = aero.CSr
        self.CS_da = aero.CSda
        self.CS_de = aero.CSde

        self.CD_0 = aero.CD0
        self.CD_L = aero.CDL
        self.CD_L2 = aero.CDL2
        self.CD_S = aero.CDS
        self.CD_S2 = aero.CDS2
        self.CD_Sp = aero.CDSp
        self.CD_p = aero.CDp
        self.CD_L2q = aero.CDL2q
        self.CD_Lq = aero.CDLq
        self.CD_q = aero.CDq
```

```
        self.CD_Sr = aero.CDSr
        self.CD_r = aero.CDr
        self.CD_Sda = aero.CDSda
        self.CD_da = aero.CDda
        self.CD_Lde = aero.CDLde
        self.CD_de = aero.CDde
        self.CD_de2 = aero.CDde2

        self.Cl_0 = aero.Cl0
        self.Cl_a = aero.Cla
        self.Cl_b = aero.Clb
        self.Cl_p = aero.Clp
        self.Cl_q = aero.Clq
        self.Cl_Lr = aero.ClLr
        self.Cl_r = aero.Clr
        self.Cl_da = aero.Clda
        self.Cl_de = aero.Clde

        self.Cm_0 = aero.Cm0
        self.Cm_a = aero.Cma
        self.Cm_b = aero.Cmb
        self.Cm_p = aero.Cmp
        self.Cm_q = aero.Cmq
        self.Cm_r = aero.Cmr
        self.Cm_da = aero.Cmda
        self.Cm_de = aero.Cmde

        self.Cn_0 = aero.Cn0
        self.Cn_a = aero.Cna
        self.Cn_b = aero.Cnb
        self.Cn_Lp = aero.CnLp
        self.Cn_p = aero.Cnp
        self.Cn_q = aero.Cnq
        self.Cn_r = aero.Cnr
        self.Cn_Lda = aero.CnLda
        self.Cn_da = aero.Cnda
        self.Cn_de = aero.Cnde

        self.dCL_0 = aero.dCL0*self.ddB_du
        self.dCL_a = aero.dCLa*self.ddB_du
        self.dCL_b = aero.dCLb*self.ddB_du
        self.dCL_p = aero.dCLp*self.ddB_du
        self.dCL_q = aero.dCLq*self.ddB_du
        self.dCL_r = aero.dCLr*self.ddB_du
        self.dCL_da = aero.dCLda*self.ddB_du
        self.dCL_de = aero.dCLde*self.ddB_du
        self.dCL1_hat = self.dCL_0 + self.dCL_a*self.alpha_hat

        self.dCS_0 = aero.dCS0*self.ddB_du
        self.dCS_a = aero.dCSa*self.ddB_du
        self.dCS_b = aero.dCSb*self.ddB_du
        self.dCS1_hat = self.dCS_0 + self.dCS_b*self.beta_hat
        self.dCS_Lp = aero.dCSLp*self.ddB_du
        self.dCS_p = aero.dCSp*self.ddB_du
        self.dCS_q = aero.dCSq*self.ddB_du
        self.dCS_r = aero.dCSr*self.ddB_du
```

```
        self.dCS_da = aero.dCSda*self.ddB_du
        self.dCS_de = aero.dCSde*self.ddB_du

        self.dCD_0 = aero.dCD0*self.ddB_du
        self.dCD_L = aero.dCDL*self.ddB_du
        self.dCD_L2 = aero.dCDL2*self.ddB_du
        self.dCD_S = aero.dCDS*self.ddB_du
        self.dCD_S2 = aero.dCDS2*self.ddB_du
        self.dCD_Sp = aero.dCDSp*self.ddB_du
        self.dCD_p = aero.dCDp*self.ddB_du
        self.dCD_L2q = aero.dCDL2q*self.ddB_du
        self.dCD_Lq = aero.dCDLq*self.ddB_du
        self.dCD_q = aero.dCDq*self.ddB_du
        self.dCD_Sr = aero.dCDSr*self.ddB_du
        self.dCD_r = aero.dCDr*self.ddB_du
        self.dCD_Sda = aero.dCDSda*self.ddB_du
        self.dCD_da = aero.dCDda*self.ddB_du
        self.dCD_Lde = aero.dCDLde*self.ddB_du
        self.dCD_de = aero.dCDde*self.ddB_du
        self.dCD_de2 = aero.dCDde2*self.ddB_du

        self.dCl_0 = aero.dCl0*self.ddB_du
        self.dCl_a = aero.dCla*self.ddB_du
        self.dCl_b = aero.dClb*self.ddB_du
        self.dCl_p = aero.dClp*self.ddB_du
        self.dCl_q = aero.dClq*self.ddB_du
        self.dCl_Lr = aero.dClLr*self.ddB_du
        self.dCl_r = aero.dClr*self.ddB_du
        self.dCl_da = aero.dClda*self.ddB_du
        self.dCl_de = aero.dClde*self.ddB_du

        self.dCm_0 = aero.dCm0*self.ddB_du
        self.dCm_a = aero.dCma*self.ddB_du
        self.dCm_b = aero.dCmb*self.ddB_du
        self.dCm_p = aero.dCmp*self.ddB_du
        self.dCm_q = aero.dCmq*self.ddB_du
        self.dCm_r = aero.dCmr*self.ddB_du
        self.dCm_da = aero.dCmda*self.ddB_du
        self.dCm_de = aero.dCmde*self.ddB_du

        self.dCn_0 = aero.dCn0*self.ddB_du
        self.dCn_a = aero.dCna*self.ddB_du
        self.dCn_b = aero.dCnb*self.ddB_du
        self.dCn_Lp = aero.dCnLp*self.ddB_du
        self.dCn_p = aero.dCnp*self.ddB_du
        self.dCn_q = aero.dCnq*self.ddB_du
        self.dCn_r = aero.dCnr*self.ddB_du
        self.dCn_Lda = aero.dCnLda*self.ddB_du
        self.dCn_da = aero.dCnda*self.ddB_du
        self.dCn_de = aero.dCnde*self.ddB_du

    def _det_I(self, dB):
        Ixx = self.I_xx(dB)
        Iyy = self.I_yy(dB)
        Izz = self.I_zz(dB)
        Iyz = self.I_yz(dB)
```

```
            Ixz = self.I_xz(dB)
            Ixy = self.I_xy(dB)
            Izy = Iyz
            Izx = Ixz
            Iyx = Ixy
            C1 = Ixx*(Iyy*Izz - Iyz*Izy)
            C2 = Ixy*(Iyx*Izz + Iyz*Izx)
            C3 = Ixz*(Iyx*Izy + Iyy*Izx)
            return C1 - C2 - C3

        def _ddetI_du(self):
            Ixx = self.I_xx(self.dB_hat)
            Iyy = self.I_yy(self.dB_hat)
            Izz = self.I_zz(self.dB_hat)
            Iyz = self.I_yz(self.dB_hat)
            Ixz = self.I_xz(self.dB_hat)
            Ixy = self.I_xy(self.dB_hat)
            dIxx = self.dI_xx(self.dB_hat)
            dIyy = self.dI_yy(self.dB_hat)
            dIzz = self.dI_zz(self.dB_hat)
            dIyz = self.dI_yz(self.dB_hat)
            dIxz = self.dI_xz(self.dB_hat)
            dIxy = self.dI_xy(self.dB_hat)
            ddetIdu = (dIxx*(Iyy*Izz - Iyz**2) +
                       Ixx*(dIyy*Izz + Iyy*dIzz - 2.*Iyz*dIyz) -
                       dIxy*(Ixy*Izz + Iyz*Ixz) -
                       Ixy*(dIxy*Izz + Ixy*dIzz + dIyz*Ixz + Iyz*dIxz) -
                       dIxz*(Ixy*Iyz + Iyy*Ixz) -
                       Ixz*(dIxy*Iyz + Ixy*dIyz + dIyy*Ixz + Iyy*dIxz))
            return ddetIdu

        def _Istar(self):
            Ixx = self.I_xx(self.dB_hat)
            Iyy = self.I_yy(self.dB_hat)
            Izz = self.I_zz(self.dB_hat)
            Iyz = self.I_yz(self.dB_hat)
            Ixz = self.I_xz(self.dB_hat)
            Ixy = self.I_xy(self.dB_hat)
            Istar = np.zeros((3, 3))
            Istar[0, 0] = Iyy*Izz - Iyz**2
            Istar[0, 1] = Ixy*Izz + Ixz*Iyz
            Istar[0, 2] = Ixy*Iyz + Ixz*Iyy
            Istar[1, 0] = Istar[0, 1]
            Istar[1, 1] = Ixx*Izz - Ixz**2
            Istar[1, 2] = Ixx*Iyz + Ixy*Ixz
            Istar[2, 0] = Istar[0, 2]
            Istar[2, 1] = Istar[1, 2]
            Istar[2, 2] = Ixx*Iyy - Ixy**2
            return Istar

        def _dIstar_du(self):
            Ixx = self.I_xx(self.dB_hat)
            Iyy = self.I_yy(self.dB_hat)
            Izz = self.I_zz(self.dB_hat)
            Iyz = self.I_yz(self.dB_hat)
            Ixz = self.I_xz(self.dB_hat)
```

```
        Ixy = self.I_xy(self.dB_hat)
        dIxx = self.dI_xx(self.dB_hat)
        dIyy = self.dI_yy(self.dB_hat)
        dIzz = self.dI_zz(self.dB_hat)
        dIyz = self.dI_yz(self.dB_hat)
        dIxz = self.dI_xz(self.dB_hat)
        dIxy = self.dI_xy(self.dB_hat)
        dIstardu = np.zeros((3, 3, self.M))
        dIstardu[0, 0, :] = dIyy*Izz + Iyy*dIzz - 2.*Iyz*dIyz
        dIstardu[0, 1, :] = dIxy*Izz + Ixy*dIzz + dIxz*Iyz + Ixz*dIyz
        dIstardu[0, 2, :] = dIxy*Iyz + Ixy*dIyz + dIxz*Iyy + Ixz*dIyy
        dIstardu[1, 0, :] = dIxy*Izz + Ixy*dIzz + dIyz*Ixz + Iyz*dIxz
        dIstardu[1, 1, :] = dIxx*Izz + Ixx*dIzz - 2.*Ixz*dIxz
        dIstardu[1, 2, :] = dIxx*Iyz + Ixx*dIyz + dIxy*Ixz + Ixy*dIxz
        dIstardu[2, 0, :] = dIxy*Iyz + Ixy*dIyz + dIyy*Ixz + Iyy*dIxz
        dIstardu[2, 1, :] = dIxx*Iyz + Ixx*dIyz + dIxy*Ixz + Ixy*dIxz
        dIstardu[2, 2, :] = dIxx*Iyy + Ixx*dIyy - 2.*Ixy*dIxy
        return dIstardu

    def _I_inv(self, dB):
        Ixx = self.I_xx(dB)
        Iyy = self.I_yy(dB)
        Izz = self.I_zz(dB)
        Iyz = self.I_yz(dB)
        Ixz = self.I_xz(dB)
        Ixy = self.I_xy(dB)
        Izy = Iyz
        Izx = Ixz
        Iyx = Ixy
        det_I = self._det_I(dB)
        I_inv = np.zeros((3, 3))
        I_inv[0, 0] = Iyy*Izz - Iyz*Izy
        I_inv[0, 1] = Ixy*Izz + Ixz*Izy
        I_inv[0, 2] = Ixy*Iyz + Ixz*Iyy
        I_inv[1, 0] = Iyx*Izz + Iyz*Izx
        I_inv[1, 1] = Ixx*Izz - Ixz*Izx
        I_inv[1, 2] = Ixx*Iyz + Ixz*Iyz
        I_inv[2, 0] = Iyz*Izy + Iyy*Izx
        I_inv[2, 1] = Ixx*Izy + Ixy*Izx
        I_inv[2, 2] = Ixx*Iyy - Ixy*Iyx
        I_inv = I_inv/det_I
        return I_inv

    def _dIinv_du(self):
        dIinvdu = np.zeros((3, 3, self.M))
        dIstardu = self._dIstar_du()
        ddetI = self._ddetI_du()
        detI = self._det_I(self.dB_hat)
        Istar = self._Istar()
        dIinvdu = (detI*dIstardu - Istar[:, :, None]*ddetI[None, :])/(detI**2)
        return dIinvdu

    def _dz1_dz(self):
        dz1_dz = np.zeros(self.N)
        dFxdz = self._dFx_dz()
        dz1_dz = self.props.g/self.props.W*dFxdz
```

```
            dz1_dz[1] += self.x_hat[5]
            dz1_dz[2] -= self.x_hat[4]
            dz1_dz[4] -= self.x_hat[2]
            dz1_dz[5] += self.x_hat[1]
            dz1_dz[7] -= self.props.g*np.cos(self.x_hat[7])
            return dz1_dz

        def _dz2_dz(self):
            dz2_dz = np.zeros(self.N)
            dFydz = self._dFy_dz()
            dz2_dz = self.props.g/self.props.W*dFydz
            dz2_dz[0] -= self.x_hat[5]
            dz2_dz[2] += self.x_hat[3]
            dz2_dz[3] += self.x_hat[2]
            dz2_dz[5] -= self.x_hat[0]
            dz2_dz[6] += self.props.g*np.cos(self.x_hat[6])*np.cos(self.x_hat[7])
            dz2_dz[7] -= self.props.g*np.sin(self.x_hat[6])*np.sin(self.x_hat[7])
            return dz2_dz

        def _dz3_dz(self):
            dz3_dz = np.zeros(self.N)
            dFzdz = self._dFz_dz()
            dz3_dz = self.props.g/self.props.W*dFzdz
            dz3_dz[0] += self.x_hat[4]
            dz3_dz[1] -= self.x_hat[3]
            dz3_dz[3] -= self.x_hat[1]
            dz3_dz[4] += self.x_hat[0]
            dz3_dz[6] -= self.props.g*np.sin(self.x_hat[6])*np.cos(self.x_hat[7])
            dz3_dz[7] -= self.props.g*np.cos(self.x_hat[6])*np.sin(self.x_hat[7])
            return dz3_dz

        def _dz4_dz(self):
            dz4_dz = np.zeros(self.N)
            dMxdz = self._dMx_dz()
            dMydz = self._dMy_dz()
            dMzdz = self._dMz_dz()
            dM = np.array([dMxdz, dMydz, dMzdz])
            R = np.zeros((3, self.N + self.M))
            R = dM + self.W_mat
            dz4_dz = np.matmul(self.I_inv, R)[0, :]
            return dz4_dz

        def _dz5_dz(self):
            dz5_dz = np.zeros(self.N)
            dMxdz = self._dMx_dz()
            dMydz = self._dMy_dz()
            dMzdz = self._dMz_dz()
            dM = np.array([dMxdz, dMydz, dMzdz])
            R = np.zeros((3, self.N + self.M))
            R = dM + self.W_mat
            dz5_dz = np.matmul(self.I_inv, R)[1, :]
            return dz5_dz

        def _dz6_dz(self):
            dz6_dz = np.zeros(self.N)
            dMxdz = self._dMx_dz()
```

```
            dMydz = self._dMy_dz()
            dMzdz = self._dMz_dz()
            dM = np.array([dMxdz, dMydz, dMzdz])
            R = np.zeros((3, self.N + self.M))
            R = dM + self.W_mat
            dz6_dz = np.matmul(self.I_inv, R)[2, :]
            return dz6_dz

        def _dz7_dz(self):
            dz7_dz = np.zeros(self.N)
            s_7 = np.sin(self.x_hat[6])
            c_7 = np.cos(self.x_hat[6])
            s_8 = np.sin(self.x_hat[7])
            c_8 = np.cos(self.x_hat[7])
            t_8 = s_8/c_8
            dz7_dz[3] = 1.
            dz7_dz[4] = s_7*t_8
            dz7_dz[5] = c_7*t_8
            dz7_dz[6] = t_8*(c_7*self.x_hat[4] - s_7*self.x_hat[5])
            dz7_dz[7] = s_7/(c_8**2)*self.x_hat[4] + c_7/(c_8**2)*self.x_hat[5]
            return dz7_dz

        def _dz8_dz(self):
            dz8_dz = np.zeros(self.N)
            s_7 = np.sin(self.x_hat[6])
            c_7 = np.cos(self.x_hat[6])
            dz8_dz[4] = c_7
            dz8_dz[5] = -s_7
            dz8_dz[6] = -s_7*self.x_hat[4] - c_7*self.x_hat[5]
            return dz8_dz

        def _dFx_dz(self):
            dFxdz = np.zeros(self.N)
            dCXdz = self._dCX_dz()
            dVdz = self.dV_dz
            dTXdz = self._dTX_dz()
            c_a = np.cos(self.alpha_hat)
            s_a = np.sin(self.alpha_hat)
            c_b = np.cos(self.beta_hat)
            s_b = np.sin(self.beta_hat)
            CX = -(self.CD_hat*c_a*c_b + self.CS_hat*c_a*s_b -
                   self.CL_hat*s_a)
            dFxdz = (0.5*self.rho*self.V_hat**2*self.S_w*dCXdz +
                     self.rho*self.V_hat*self.S_w*CX*dVdz + dTXdz)
            return dFxdz

        def _dFy_dz(self):
            dFydz = np.zeros(self.N)
            dCYdz = self._dCY_dz()
            dVdz = self.dV_dz
            c_b = np.cos(self.beta_hat)
            s_b = np.sin(self.beta_hat)
            CY = self.CS_hat*c_b - self.CD_hat*s_b
            dFydz = (0.5*self.rho*self.V_hat**2*self.S_w*dCYdz +
                     self.rho*self.V_hat*self.S_w*CY*dVdz)
            return dFydz
```

```
    def _dFz_dz(self):
        dFzdz = np.zeros(self.N)
        dCZdz = self._dCZ_dz()
        dVdz = self.dV_dz
        c_a = np.cos(self.alpha_hat)
        s_a = np.sin(self.alpha_hat)
        c_b = np.cos(self.beta_hat)
        s_b = np.sin(self.beta_hat)
        CZ = -(self.CD_hat*s_a*c_b + self.CS_hat*s_a*s_b +
               self.CL_hat*c_a)
        dFzdz = (0.5*self.rho*self.V_hat**2*self.S_w*dCZdz +
                 self.rho*self.V_hat*self.S_w*CZ*dVdz)
        return dFzdz

    def _dMx_dz(self):
        dMxdz = np.zeros(self.N)
        dCldz = self._dCl_dz()
        dFzdz = self._dFz_dz()
        dFydz = self._dFy_dz()
        dVdz = self.dV_dz
        dy = self.Dy
        dz = self.Dz
        dMxdz = (0.5*self.rho*self.V_hat**2*self.S_w*self.b_w*dCldz +
                 self.rho*self.V_hat*self.S_w*self.b_w*self.Cl_hat*dVdz -
                 dFzdz*dy +
                 dFydz*dz)
        return dMxdz

    def _dMy_dz(self):
        dMydz = np.zeros(self.N)
        dCmdz = self._dCm_dz()
        dFzdz = self._dFz_dz()
        dFxdz = self._dFx_dz()
        dVdz = self.dV_dz
        dx = self.Dx
        dz = self.Dz
        dMydz = (0.5*self.rho*self.V_hat**2*self.S_w*self.c_w*dCmdz +
                 self.rho*self.V_hat*self.S_w*self.c_w*self.Cm_hat*dVdz -
                 dFzdz*dx +
                 dFxdz*dz)
        return dMydz

    def _dMz_dz(self):
        dMzdz = np.zeros(self.N)
        dCndz = self._dCn_dz()
        dFxdz = self._dFx_dz()
        dFydz = self._dFy_dz()
        dVdz = self.dV_dz
        dx = self.Dx
        dy = self.Dy
        dMzdz = (0.5*self.rho*self.V_hat**2*self.S_w*self.b_w*dCndz +
                 self.rho*self.V_hat*self.S_w*self.b_w*self.Cn_hat*dVdz -
                 dFydz*dx +
                 dFxdz*dy)
        return dMzdz
```

```

    def _dV_dz(self):
        dVdz = np.zeros(self.N)
        dVdz[0] = self.x_hat[0]/self.V_hat
        dVdz[1] = self.x_hat[1]/self.V_hat
        dVdz[2] = self.x_hat[2]/self.V_hat
        return dVdz

    def _dVinv_dz(self):
        dVinvdz = np.zeros(self.N)
        dVinvdz[0] = -self.x_hat[0]/self.V_hat**3
        dVinvdz[1] = -self.x_hat[1]/self.V_hat**3
        dVinvdz[2] = -self.x_hat[2]/self.V_hat**3
        return dVinvdz

    def _dTX_dz(self):
        dVdz = self._dV_dz()
        H = self.props.H
        a_mil = self.props.a_mil(H)
        T1_mil = self.props.T1_mil(H)
        T2_mil = self.props.T2_mil(H)
        rho_ratio = (self.rho/self.rho_0)
        dTmil_dz = rho_ratio**a_mil*(T1_mil*dVdz + 2.*T2_mil*self.V_hat*dVdz)
        if self.u_hat[0] < 0.77:
            a_idle = self.props.a_idle(H)
            T1_idle = self.props.T1_idle(H)
            T2_idle = self.props.T2_idle(H)
            dTidle_dz = rho_ratio**a_idle*(T1_idle*dVdz + 2.*T2_idle*self.V_hat*dVdz)
            P1 = 64.94*self.u_hat[0]/50.
            dTX_dz = P1*(dTmil_dz - dTidle_dz) + dTidle_dz
        else:
            a_max = self.props.a_max(H)
            T1_max = self.props.T1_max(H)
            T2_max = self.props.T2_max(H)
            dTmax_dz = rho_ratio**a_max*(T1_max*dVdz + 2.*T2_max*self.V_hat*dVdz)
            P1 = (217.38*self.u_hat[0] - 117.38 - 50.)/50.
            dTX_dz = P1*(dTmax_dz - dTmil_dz) + dTmil_dz
        return dTX_dz

    def _dCX_dz(self):
        dCXdz = np.zeros(self.N)
        dCDdz = self._dCD_dz()
        dCSdz = self._dCS_dz()
        dCLdz = self._dCL_dz()
        dadz = self.da_dz
        dbdz = self.db_dz
        c_a = np.cos(self.alpha_hat)
        s_a = np.sin(self.alpha_hat)
        c_b = np.cos(self.beta_hat)
        s_b = np.sin(self.beta_hat)
        CD = self.CD_hat
        CS = self.CS_hat
        CL = self.CL_hat
        dCXdz = (-dCDdz*c_a*c_b + CD*s_a*c_b*dadz + CD*c_a*s_b*dbdz -
                 dCSdz*c_a*s_b + CS*s_a*s_b*dadz - CS*c_a*c_b*dbdz +
                 dCLdz*s_a + CL*c_a*dadz)
```

```
            return dCXdz

    def _dCY_dz(self):
        dCYdz = np.zeros(self.N)
        dCDdz = self._dCD_dz()
        dCSdz = self._dCS_dz()
        dbdz = self.db_dz
        c_b = np.cos(self.beta_hat)
        s_b = np.sin(self.beta_hat)
        CD = self.CD_hat
        CS = self.CS_hat
        dCYdz = dCSdz*c_b - CS*s_b*dbdz - dCDdz*s_b - CD*c_b*dbdz
        return dCYdz

    def _dCZ_dz(self):
        dCZdz = np.zeros(self.N)
        dCDdz = self._dCD_dz()
        dCSdz = self._dCS_dz()
        dCLdz = self._dCL_dz()
        dadz = self.da_dz
        dbdz = self.db_dz
        c_a = np.cos(self.alpha_hat)
        s_a = np.sin(self.alpha_hat)
        c_b = np.cos(self.beta_hat)
        s_b = np.sin(self.beta_hat)
        CD = self.CD_hat
        CS = self.CS_hat
        CL = self.CL_hat
        dCZdz = (-dCDdz*s_a*c_b - CD*c_a*c_b*dadz + CD*s_a*s_b*dbdz -
                 dCSdz*s_a*s_b - CS*c_a*s_b*dadz - CS*s_a*c_b*dbdz -
                 dCLdz*c_a + CL*s_a*dadz)
        return dCZdz

    def _dalpha_dz(self):
        dadz = np.zeros(self.N)
        C1 = self.x_hat[0]**2 + self.x_hat[2]**2
        dadz[0] = -self.x_hat[2]/C1
        dadz[2] = self.x_hat[0]/C1
        return dadz

    def _dbeta_dz(self):
        dbdz = np.zeros(self.N)
        C1 = np.sqrt(self.x_hat[0]**2 + self.x_hat[2]**2)
        C2 = (self.V_hat**2)*C1
        dbdz[0] = -self.x_hat[1]*self.x_hat[0]/C2
        dbdz[1] = C1/(self.V_hat**2)
        dbdz[2] = -self.x_hat[1]*self.x_hat[2]/C2
        return dbdz

    def _dCL_dz(self):
        dCLdz = np.zeros(self.N)
        dCL1dz = self._dCL1_dz()
        dpbardz = self._dpbar_dz()
        dqbardz = self._dqbar_dz()
        drbardz = self._drbar_dz()
        dCLdz = (dCL1dz + self.CL_b*self.db_dz +
```

```
                   self.CL_p*dpbardz + self.CL_q*dqbardz + self.CL_r*drbardz)
        return dCLdz

    def _dCL1_dz(self):
        dCL1dz = self.CL_a*self.da_dz
        return dCL1dz

    def _dqbar_dz(self):
        dqdz = self.dq_dz
        dVidz = self.dVinv_dz
        dqbardz = dqdz*self.c_w/(2.*self.V_hat) + dVidz*self.c_w*self.x_hat[4]/2.
        return dqbardz

    def _dCS_dz(self):
        dCSdz = np.zeros(self.N)
        dCS1dz = self._dCS1_dz()
        dCL1dz = self._dCL1_dz()
        xb_4 = self.b_w*self.x_hat[3]/(2.*self.V_hat)
        dpbardz = self._dpbar_dz()
        dqbardz = self._dqbar_dz()
        drbardz = self._drbar_dz()
        dCSdz = (self.CS_a*self.da_dz + dCS1dz +
                 self.CS_Lp*dCL1dz*xb_4 +
                 (self.CS_Lp*self.CL1_hat + self.CS_p)*dpbardz +
                 self.CS_q*dqbardz +
                 self.CS_r*drbardz)
        return dCSdz

    def _dCS1_dz(self):
        dCS1dz = self.CS_b*self.db_dz
        return dCS1dz

    def _dpbar_dz(self):
        dpdz = self.dp_dz
        dVidz = self.dVinv_dz
        dpbardz = dpdz*self.b_w/(2.*self.V_hat) + dVidz*self.b_w*self.x_hat[3]/2.
        return dpbardz

    def _drbar_dz(self):
        drdz = self.dr_dz
        dVidz = self.dVinv_dz
        drbardz = drdz*self.b_w/(2.*self.V_hat) + dVidz*self.b_w*self.x_hat[5]/2.
        return drbardz

    def _dCD_dz(self):
        dCDdz = np.zeros(self.N)
        dCL1dz = self._dCL1_dz()
        dCS1dz = self._dCS1_dz()
        dCL12dz = 2.*self.CL1_hat*dCL1dz
        dCS12dz = 2.*self.CS1_hat*dCS1dz
        xb_4 = self.b_w*self.x_hat[3]/(2.*self.V_hat)
        xb_5 = self.c_w*self.x_hat[4]/(2.*self.V_hat)
        xb_6 = self.b_w*self.x_hat[5]/(2.*self.V_hat)
        dpbardz = self._dpbar_dz()
        dqbardz = self._dqbar_dz()
        drbardz = self._drbar_dz()
```

```
            CL1 = self.CL1_hat
            CS1 = self.CS1_hat
            dCDdz = (self.CD_L*dCL1dz + self.CD_L2*dCL12dz +
                     self.CD_S*dCS1dz + self.CD_S2*dCS12dz +
                     self.CD_Sp*dCS1dz*xb_4 +
                     (self.CD_Sp*CS1 + self.CD_p)*dpbardz +
                     (self.CD_L2q*dCL12dz + self.CD_Lq*dCL1dz)*xb_5 +
                     (self.CD_L2q*CL1**2 + self.CD_Lq*CL1 + self.CD_q)*dqbardz +
                     self.CD_Sr*dCS1dz*xb_6 +
                     (self.CD_Sr*CS1 + self.CD_r)*drbardz +
                     self.CD_Sda*dCS1dz*self.u_hat[1] +
                     self.CD_Lde*dCL1dz*self.u_hat[2])
            return dCDdz

        def _dCl_dz(self):
            dCldz = np.zeros(self.N)
            dadz = self.da_dz
            dbdz = self.db_dz
            dpbardz = self._dpbar_dz()
            dqbardz = self._dqbar_dz()
            drbardz = self._drbar_dz()
            dCL1dz = self._dCL1_dz()
            xb_6 = self.b_w*self.x_hat[5]/(2.*self.V_hat)
            CL1 = self.CL1_hat
            dCldz = (self.Cl_a*dadz + self.Cl_b*dbdz + self.Cl_p*dpbardz +
                     self.Cl_q*dqbardz + self.Cl_Lr*dCL1dz*xb_6 +
                     (self.Cl_Lr*CL1 + self.Cl_r)*drbardz)
            return dCldz

        def _dCm_dz(self):
            dCmdz = np.zeros
            dadz = self.da_dz
            dbdz = self.db_dz
            dpbardz = self._dpbar_dz()
            dqbardz = self._dqbar_dz()
            drbardz = self._drbar_dz()
            dCmdz = (self.Cm_a*dadz + self.Cm_b*dbdz + self.Cm_p*dpbardz +
                     self.Cm_q*dqbardz + self.Cm_r*drbardz)
            return dCmdz

        def _dCn_dz(self):
            dCndz = np.zeros(self.N)
            dadz = self.da_dz
            dbdz = self.db_dz
            dpbardz = self._dpbar_dz()
            dqbardz = self._dqbar_dz()
            drbardz = self._drbar_dz()
            dCL1dz = self._dCL1_dz()
            xb_4 = self.b_w*self.x_hat[3]/(2.*self.V_hat)
            CL1 = self.CL1_hat
            dCndz = (self.Cn_a*dadz + self.Cn_b*dbdz + self.Cn_Lp*dCL1dz*xb_4 +
                     (self.Cn_Lp*CL1 + self.Cn_p)*dpbardz + self.Cn_q*dqbardz +
                     self.Cn_r*drbardz + self.Cn_Lda*dCL1dz*self.u_hat[1])
            return dCndz

        def _dz1_du(self):
```

```
            dFxdu = self._dFx_du()
            dz1du = self.g/self.W*dFxdu
            return dz1du

        def _dz2_du(self):
            dFydu = self._dFy_du()
            dz2du = self.g/self.W*dFydu
            return dz2du

        def _dz3_du(self):
            dFzdu = self._dFz_du()
            dz3du = self.g/self.W*dFzdu
            return dz3du

        def _M1(self):
            Iyy = self.I_yy(self.dB_hat)
            Izz = self.I_zz(self.dB_hat)
            Iyz = self.I_yz(self.dB_hat)
            Ixz = self.I_xz(self.dB_hat)
            Ixy = self.I_xy(self.dB_hat)
            M1 = (self.Mx_hat +
                  (Iyy - Izz)*self.x_hat[4]*self.x_hat[5] +
                  Iyz*(self.x_hat[4]**2 - self.x_hat[5]**2) +
                  Ixz*self.x_hat[3]*self.x_hat[4] -
                  Ixy*self.x_hat[3]*self.x_hat[5])
            return M1

        def _dM1_du(self):
            dIyy = self.dI_yy(self.dB_hat)
            dIzz = self.dI_zz(self.dB_hat)
            dIyz = self.dI_yz(self.dB_hat)
            dIxz = self.dI_xz(self.dB_hat)
            dIxy = self.dI_xy(self.dB_hat)
            dMxdu = self._dMx_du()
            dM1du = (dMxdu +
                  (dIyy - dIzz)*self.x_hat[4]*self.x_hat[5] +
                  dIyz*(self.x_hat[4]**2 - self.x_hat[5]**2) +
                  dIxz*self.x_hat[3]*self.x_hat[4] -
                  dIxy*self.x_hat[3]*self.x_hat[5])
            return dM1du

        def _M2(self):
            Ixx = self.I_xx(self.dB_hat)
            Izz = self.I_zz(self.dB_hat)
            Iyz = self.I_yz(self.dB_hat)
            Ixz = self.I_xz(self.dB_hat)
            Ixy = self.I_xy(self.dB_hat)
            M2 = (self.My_hat +
                  (Izz - Ixx)*self.x_hat[3]*self.x_hat[5] +
                  Ixz*(self.x_hat[5]**2 - self.x_hat[3]**2) +
                  Ixy*self.x_hat[4]*self.x_hat[5] -
                  Iyz*self.x_hat[3]*self.x_hat[4])
            return M2

        def _dM2_du(self):
            dIxx = self.dI_xx(self.dB_hat)
```

```
        dIzz = self.dI_zz(self.dB_hat)
        dIyz = self.dI_yz(self.dB_hat)
        dIxz = self.dI_xz(self.dB_hat)
        dIxy = self.dI_xy(self.dB_hat)
        dMydu = self._dMy_du()
        dM2du = (dMydu +
              (dIzz - dIxx)*self.x_hat[3]*self.x_hat[5] +
              dIxz*(self.x_hat[5]**2 - self.x_hat[3]**2) +
              dIxy*self.x_hat[4]*self.x_hat[5] -
              dIyz*self.x_hat[3]*self.x_hat[4])
        return dM2du

    def _M3(self):
        Ixx = self.I_xx(self.dB_hat)
        Iyy = self.I_yy(self.dB_hat)
        Iyz = self.I_yz(self.dB_hat)
        Ixz = self.I_xz(self.dB_hat)
        Ixy = self.I_xy(self.dB_hat)
        M3 = (self.Mz_hat +
              (Ixx - Iyy)*self.x_hat[3]*self.x_hat[4] +
              Ixy*(self.x_hat[3]**2 - self.x_hat[4]**2) +
              Iyz*self.x_hat[3]*self.x_hat[5] -
              Ixz*self.x_hat[4]*self.x_hat[5])
        return M3

    def _dM3_du(self):
        dIxx = self.dI_xx(self.dB_hat)
        dIyy = self.dI_yy(self.dB_hat)
        dIyz = self.dI_yz(self.dB_hat)
        dIxz = self.dI_xz(self.dB_hat)
        dIxy = self.dI_xy(self.dB_hat)
        dMzdu = self._dMz_du()
        dM3du = (dMzdu +
                 (dIxx - dIyy)*self.x_hat[3]*self.x_hat[4] +
                 dIxy*(self.x_hat[3]**2 - self.x_hat[4]**2) +
                 dIyz*self.x_hat[3]*self.x_hat[5] -
                 dIxz*self.x_hat[4]*self.x_hat[5])
        return dM3du

    def _dz4_du(self):
        dIinv = self._dIinv_du()
        M1 = self._M1()
        M2 = self._M2()
        M3 = self._M3()
        dM1du = self._dM1_du()
        dM2du = self._dM2_du()
        dM3du = self._dM3_du()
        M = np.array([M1, M2, M3])
        dM = np.array([dM1du, dM2du, dM3du])
        dz4du = np.zeros(self.M)
        for i in range(self.M):
            dz4du[i] = np.matmul(dIinv[:, :, i], M)[0]
        dz4du = dz4du + np.matmul(self.I_inv, dM)[0, :]
        return dz4du

    def _dz5_du(self):
```

```
        dIinv = self._dIinv_du()
        M1 = self._M1()
        M2 = self._M2()
        M3 = self._M3()
        dM1du = self._dM1_du()
        dM2du = self._dM2_du()
        dM3du = self._dM3_du()
        M = np.array([M1, M2, M3])
        dM = np.array([dM1du, dM2du, dM3du])
        dz5du = np.zeros(self.M)
        for i in range(self.M):
            dz5du[i] = np.matmul(dIinv[:, :, i], M)[1]
        dz5du = dz5du + np.matmul(self.I_inv, dM)[1, :]
        return dz5du

    def _dz6_du(self):
        dIinv = self._dIinv_du()
        M1 = self._M1()
        M2 = self._M2()
        M3 = self._M3()
        dM1du = self._dM1_du()
        dM2du = self._dM2_du()
        dM3du = self._dM3_du()
        M = np.array([M1, M2, M3])
        dM = np.array([dM1du, dM2du, dM3du])
        dz6du = np.zeros(self.M)
        for i in range(self.M):
            dz6du[i] = np.matmul(dIinv[:, :, i], M)[2]
        dz6du = dz6du + np.matmul(self.I_inv, dM)[2, :]
        return dz6du

    def _dCL_du(self):
        xb_4 = self.b_w*self.x_hat[3]/(2.*self.V_hat)
        xb_5 = self.c_w*self.x_hat[4]/(2.*self.V_hat)
        xb_6 = self.b_w*self.x_hat[5]/(2.*self.V_hat)
        dCLdu = (self.dCL1_hat +
                 self.dCL_b*self.beta_hat +
                 self.dCL_p*xb_4 +
                 self.dCL_q*xb_5 +
                 self.dCL_r*xb_6 +
                 self.dCL_da*self.u_hat[1] +
                 self.CL_da*self.dda_du +
                 self.dCL_de*self.u_hat[2] +
                 self.CL_de*self.dde_du)
        return dCLdu

    def _dCS_du(self):
        xb_4 = self.b_w*self.x_hat[3]/(2.*self.V_hat)
        xb_5 = self.c_w*self.x_hat[4]/(2.*self.V_hat)
        xb_6 = self.b_w*self.x_hat[5]/(2.*self.V_hat)
        dCSdu = (self.dCS1_hat +
                 self.dCS_a*self.alpha_hat +
                 (self.dCS_Lp*self.CL1_hat +
                  self.CS_Lp*self.dCL1_hat +
                  self.dCS_p)*xb_4 +
                 self.dCS_q*xb_5 +
```

```
                  self.dCS_r*xb_6 +
                  self.dCS_da*self.u_hat[1] +
                  self.CS_da*self.dda_du +
                  self.dCS_de*self.u_hat[2] +
                  self.CS_de*self.dde_du)
        return dCSdu

    def _dCD_du(self):
        xb_4 = self.b_w*self.x_hat[3]/(2.*self.V_hat)
        xb_5 = self.c_w*self.x_hat[4]/(2.*self.V_hat)
        xb_6 = self.b_w*self.x_hat[5]/(2.*self.V_hat)
        dCL12du = 2.*self.CL1_hat*self.dCL1_hat
        dCS12du = 2.*self.CS1_hat*self.dCS1_hat
        dCDdu = (self.dCD_0 +
                  self.dCD_L*self.CL1_hat +
                  self.CD_L*self.dCL1_hat +
                  self.dCD_L2*self.CL1_hat**2 +
                  self.CD_L2*dCL12du +
                  self.dCD_S*self.CS1_hat +
                  self.CD_S*self.dCS1_hat +
                  self.dCD_S2*self.CS1_hat**2 +
                  self.CD_S2*dCS12du +
                  (self.dCD_Sp*self.CS1_hat +
                   self.CD_Sp*self.dCS1_hat +
                   self.dCD_p)*xb_4 +
                  (self.dCD_L2q*self.CL1_hat**2 +
                   self.CD_L2q*dCL12du +
                   self.dCD_Lq*self.CL1_hat +
                   self.CD_Lq*self.dCL1_hat +
                   self.dCD_q)*xb_5 +
                  (self.dCD_Sr*self.CS1_hat +
                   self.CD_Sr*self.dCS1_hat +
                   self.dCD_r)*xb_6 +
                  (self.dCD_Sda*self.CS1_hat +
                   self.CD_Sda*self.dCS1_hat +
                   self.dCD_da)*self.u_hat[1] +
                  (self.CD_Sda*self.CS1_hat + self.CD_da)*self.dda_du +
                  (self.dCD_Lde*self.CL1_hat +
                   self.CD_Lde*self.dCL1_hat +
                   self.dCD_de)*self.u_hat[2] +
                  (self.CD_Lde*self.CL1_hat + self.CD_de)*self.dde_du +
                  self.dCD_de2*self.u_hat[2]**2 +
                  self.CD_de2*self.dde2_du)
        return dCDdu

    def _dCl_du(self):
        xb_4 = self.b_w*self.x_hat[3]/(2.*self.V_hat)
        xb_5 = self.c_w*self.x_hat[4]/(2.*self.V_hat)
        xb_6 = self.b_w*self.x_hat[5]/(2.*self.V_hat)
        dCldu = (self.dCl_0 +
                  self.dCl_a*self.alpha_hat +
                  self.dCl_b*self.beta_hat +
                  self.dCl_p*xb_4 +
                  self.dCl_q*xb_5 +
                  (self.dCl_Lr*self.CL1_hat +
                   self.Cl_Lr*self.dCL1_hat +
```

```
                     self.dCl_r)*xb_6 +
                    self.dCl_da*self.u_hat[1] +
                    self.Cl_da*self.dda_du +
                    self.dCl_de*self.u_hat[2] +
                    self.Cl_de*self.dde_du)
        return dCldu

    def _dCm_du(self):
        xb_4 = self.b_w*self.x_hat[3]/(2.*self.V_hat)
        xb_5 = self.c_w*self.x_hat[4]/(2.*self.V_hat)
        xb_6 = self.b_w*self.x_hat[5]/(2.*self.V_hat)
        dCmdu = (self.dCm_0 +
                    self.dCm_a*self.alpha_hat +
                    self.dCm_b*self.beta_hat +
                    self.dCm_p*xb_4 +
                    self.dCm_q*xb_5 +
                    self.dCm_r*xb_6 +
                    self.dCm_da*self.u_hat[1] +
                    self.Cm_da*self.dda_du +
                    self.dCm_de*self.u_hat[2] +
                    self.Cm_de*self.dde_du)
        return dCmdu

    def _dCn_du(self):
        xb_4 = self.b_w*self.x_hat[3]/(2.*self.V_hat)
        xb_5 = self.c_w*self.x_hat[4]/(2.*self.V_hat)
        xb_6 = self.b_w*self.x_hat[5]/(2.*self.V_hat)
        dCndu = (self.dCn_0 +
                    self.dCn_a*self.alpha_hat +
                    self.dCn_b*self.beta_hat +
                    (self.dCn_Lp*self.CL1_hat +
                     self.Cn_Lp*self.dCL1_hat +
                     self.dCn_p)*xb_4 +
                    self.dCn_q*xb_5 +
                    self.dCn_r*xb_6 +
                    (self.dCn_Lda*self.CL1_hat +
                     self.Cn_Lda*self.dCL1_hat +
                     self.dCn_da)*self.u_hat[1] +
                    (self.Cn_Lda*self.CL1_hat +
                     self.Cn_da)*self.dda_du +
                    self.dCn_de*self.u_hat[2] +
                    self.Cn_de*self.dde_du)
        return dCndu

    def _dTx_du(self):
        a_mil = self.props.a_mil(self.props.H)
        T0_mil = self.props.T0_mil(self.props.H)
        T1_mil = self.props.T1_mil(self.props.H)
        T2_mil = self.props.T2_mil(self.props.H)
        V = self.props.V
        T_mil = (self.rho/self.rho_0)**a_mil*(T0_mil + T1_mil*V + T2_mil*V**2)
        if self.u_hat[0] < 0.77:
            a_idle = self.props.a_idle(self.props.H)
            T0_idle = self.props.T0_idle(self.props.H)
            T1_idle = self.props.T1_idle(self.props.H)
            T2_idle = self.props.T2_idle(self.props.H)
```

```
            T_idle = (self.rho/self.rho_0)**a_idle*(T0_idle + T1_idle*V +
                                                    T2_idle*V**2)
            dTxdu = 64.94/50.*(T_mil - T_idle)
        else:
            a_max = self.props.a_max(self.props.H)
            T0_max = self.props.T0_max(self.props.H)
            T1_max = self.props.T1_max(self.props.H)
            T2_max = self.props.T2_max(self.props.H)
            T_max = (self.rho/self.rho_0)**a_max*(T0_max + T1_max*V + T2_max*V**2)
            dTxdu = 217.38/50.*(T_max - T_mil)
        return dTxdu

    def _dCX_du(self):
        dCDdu = self._dCD_du()
        dCSdu = self._dCS_du()
        dCLdu = self._dCL_du()
        c_a = np.cos(self.alpha_hat)
        s_a = np.sin(self.alpha_hat)
        c_b = np.cos(self.beta_hat)
        s_b = np.sin(self.beta_hat)
        dCXdu = -(dCDdu*c_a*c_b + dCSdu*c_a*s_b - dCLdu*s_a)
        return dCXdu

    def _dCY_du(self):
        dCDdu = self._dCD_du()
        dCSdu = self._dCS_du()
        c_b = np.cos(self.beta_hat)
        s_b = np.sin(self.beta_hat)
        dCYdu = dCSdu*c_b - dCDdu*s_b
        return dCYdu

    def _dCZ_du(self):
        dCDdu = self._dCD_du()
        dCSdu = self._dCS_du()
        dCLdu = self._dCL_du()
        c_a = np.cos(self.alpha_hat)
        s_a = np.sin(self.alpha_hat)
        c_b = np.cos(self.beta_hat)
        s_b = np.sin(self.beta_hat)
        dCZdu = -(dCDdu*s_a*c_b + dCSdu*s_a*s_b + dCLdu*c_a)
        return dCZdu

    def _dFx_du(self):
        dCXdu = self._dCX_du()
        dTxdu = self._dTx_du()
        dFxdu = 0.5*self.rho*self.V_hat**2*self.S_w*dCXdu + dTxdu
        return dFxdu

    def _dFy_du(self):
        dCYdu = self._dCY_du()
        dFydu = 0.5*self.rho*self.V_hat**2*self.S_w*dCYdu
        return dFydu

    def _dFz_du(self):
        dCZdu = self._dCZ_du()
        dFzdu = 0.5*self.rho*self.V_hat**2*self.S_w*dCZdu
```

```
            return dFzdu

        def _dMx_du(self):
            dCldu = self._dCl_du()
            dFzdu = self._dFz_du()
            dFydu = self._dFy_du()
            dMxdu = (0.5*self.rho*self.V_hat**2*self.S_w*self.b_w*dCldu -
                     dFzdu*self.Dy +
                     dFydu*self.Dz)
            return dMxdu

        def _dMy_du(self):
            dCmdu = self._dCm_du()
            dFzdu = self._dFz_du()
            dFxdu = self._dFx_du()
            dMydu = (0.5*self.rho*self.V_hat**2*self.S_w*self.c_w*dCmdu -
                     dFzdu*self.Dx +
                     dFxdu*self.Dz)
            return dMydu

        def _dMz_du(self):
            dCndu = self._dCn_du()
            dFydu = self._dFy_du()
            dFxdu = self._dFx_du()
            dMzdu = (0.5*self.rho*self.V_hat**2*self.S_w*self.b_w*dCndu -
                     dFydu*self.Dx +
                     dFxdu*self.Dy)
            return dMzdu

        def _W_matrix(self):
            self.W_mat = np.zeros((3, self.N))
            Ixx = self.I_xx(self.dB_hat)
            Ixy = self.I_xy(self.dB_hat)
            Ixz = self.I_xz(self.dB_hat)
            Iyy = self.I_yy(self.dB_hat)
            Izz = self.I_zz(self.dB_hat)
            Iyz = self.I_yz(self.dB_hat)
            self.W_mat[:, 3] = np.array([Ixz*self.x_hat[4] - Ixy*self.x_hat[5],
                                         (Izz - Ixx)*self.x_hat[5] -
                                         2.*Ixz*self.x_hat[3] - Iyz*self.x_hat[4],
                                         (Ixx - Iyy)*self.x_hat[4] +
                                         2.*Ixy*self.x_hat[3] + Iyz*self.x_hat[5]])
            self.W_mat[:, 4] = np.array([(Iyy - Izz)*self.x_hat[5] -
                                         2.*Iyz*self.x_hat[4] + Ixz*self.x_hat[3],
                                         Ixy*self.x_hat[5] - Iyz*self.x_hat[3],
                                         (Ixx - Iyy)*self.x_hat[3] -
                                         2.*Ixy*self.x_hat[4] - Ixz*self.x_hat[5]])
            self.W_mat[:, 5] = np.array([(Iyy - Izz)*self.x_hat[4] +
                                         2.*Iyz*self.x_hat[5] - Ixy*self.x_hat[3],
                                         (Izz - Ixx)*self.x_hat[3] +
                                         2.*Ixz*self.x_hat[5] + Ixy*self.x_hat[4],
                                         Iyz*self.x_hat[3] - Ixz*self.x_hat[4]])


        def create_A_matrix(self):
            A = np.zeros((self.N, self.N))
```

```
        A[0, :] = self._dz1_dz()
        A[1, :] = self._dz2_dz()
        A[2, :] = self._dz3_dz()
        A[3, :] = self._dz4_dz()
        A[4, :] = self._dz5_dz()
        A[5, :] = self._dz6_dz()
        A[6, :] = self._dz7_dz()
        A[7, :] = self._dz8_dz()
        return A

    def create_B_matrix(self):
        B = np.zeros((self.N, self.M))
        B[0, :] = self._dz1_du()
        B[1, :] = self._dz2_du()
        B[2, :] = self._dz3_du()
        B[3, :] = self._dz4_du()
        B[4, :] = self._dz5_du()
        B[5, :] = self._dz6_du()
        return B

    def create_C_matrix(self):
        C = np.eye(self.N)
        return C

def create_feedback_control(trim_solution, V, H, Gamma, cg_shift, Q, R):
    aero_dir = '/home/christian/Python Projects/AFRL BIRE/Static Analysis/main/'
    x_hat = trim_solution.states
    alpha_hat = trim_solution.x[1]
    beta_hat = trim_solution.x[2]
    u_hat = trim_solution.inputs
    FM_hat = trim_solution.FM
    props = trim.AircraftProperties(V, H, Gamma, aero_dir, bire=True)
    linearization = LinearizationBIRE(props, aero_dir)
    linearization.set_linearization_point(x_hat, u_hat, alpha_hat, beta_hat, FM_hat,
                                          cg_shift)
    A = linearization.create_A_matrix()
    B = linearization.create_B_matrix()
    C = linearization.create_C_matrix()
    G = ctrb(A, B)
    print(np.rad2deg(x_hat[-1]), np.linalg.matrix_rank(G))
    K, S, E = lqr(A, B, Q, R)
    eig_check, v_check = np.linalg.eig(A - np.matmul(B, K))
    try:
        assert all(np.real(eig_check) < 0.)
    except  AssertionError:
        print("Not able to stabilize.")

    results = Lin_Results(linearization.N, linearization.M)
    results.A = A
    results.B = B
    results.C = C
    results.K = K
    results.eigs = eig_check
    return results

if __name__ == "__main__":
```

```
    plt.close('all')
    H = 15000.
    a = stdatm_english(H)[-1]
    M = 0.6
    V = M*a
    b_w = 30.
    c_w = 11.32
    gamma = np.deg2rad(0.)
    phi = np.deg2rad(0.)
    Gamma = 0.1
    cg_shift = [0., 0., 0.]
    aero_dir = '/home/christian/Python Projects/AFRL BIRE/Static Analysis/main/'
    trim_solution = trim.trim(V, H, gamma, phi, Gamma, fixed_point=False,
                              aero_dir=aero_dir, bire=True)
    x_hat = trim_solution.states
    alpha_hat = trim_solution.x[1]
    beta_hat = trim_solution.x[2]
    u_hat = trim_solution.inputs
    FM_hat = trim_solution.FM
    props = trim.AircraftProperties(V, H, Gamma, aero_dir)
    linearization = LinearizationBIRE(props, aero_dir)
    linearization.set_linearization_point(x_hat, u_hat, alpha_hat, beta_hat, FM_hat,
                                          cg_shift)
    A = linearization.create_A_matrix()
    B = linearization.create_B_matrix()
    C = linearization.create_C_matrix()
```

Controllability Study

```
import bire_linearization as bire
from control import ctrb
import aero_trim as trim
import numpy as np
from stdatmos import stdatm_english

def controllability_study(dB, linearization):
    u_hat[-1] = dB
    linearization.set_linearization_point(x_hat, u_hat, alpha_hat, beta_hat, FM_hat,
                                          cg_shift)
    A = linearization.create_A_matrix()
    B = linearization.create_B_matrix()
    G = ctrb(A, B)
    return np.linalg.matrix_rank(G)

if __name__ == "__main__":
    plt.close('all')
    H = 30000.
    a = stdatm_english(H)[-1]
    M = 0.8
    V = M*a
    gamma = np.deg2rad(0.)
    phi = np.deg2rad(0.)
    Gamma = 0.5
    cg_shift = [0., 0., 0.]
    aero_dir = '/home/christian/Python Projects/AFRL BIRE/Static Analysis/main/'
    trim_solution = trim.trim(V, H, gamma, phi, Gamma,
                              shss=False, bire=True,
                              cg_shift=cg_shift,
                              fixed_point=False,
                              compressible=False,
                              aero_dir=aero_dir)
    x_hat = trim_solution.states
    alpha_hat = trim_solution.x[1]
    beta_hat = trim_solution.x[2]
    u_hat = trim_solution.inputs
    FM_hat = trim_solution.FM
    props = trim.AircraftProperties(V, H, Gamma, aero_dir)
    linearization = bire.LinearizationBIRE(props, aero_dir)
    dB_range = np.deg2rad(np.arange(-90, 91, 5))
    rank = np.zeros_like(dB_range)
    u_hat[2] = 0.
    for i in range(len(dB_range)):
        rank[i] = controllability_study(dB_range[i], linearization)
```

Monte-Carlo Directional Robustness Study

```
import numpy as np
import pickle
from state_control_simulator import simulate
import aero_trim as trim
from stdatmos import stdatm_english

H = 15000.
a = stdatm_english(H)[-1]
M = 0.6
V = M*a
gamma = np.deg2rad(0.)
phi = np.deg2rad(0.)
Gamma = 0.5
cg_shift = [0., 0., 0.]
aero_dir = '/home/christian/Python Projects/AFRL BIRE/Static Analysis/main/'

with open('./BIRE_linearization.lin', 'rb') as f:
    BIRE_lin = pickle.load(f)
with open('./BIRE_solution.trim', 'rb') as f:
    BIRE_trim = pickle.load(f)
props = trim.AircraftProperties(V, H, Gamma, aero_dir, bire=True)

t_range = np.arange(0., 20., 0.1)
N = 11
MC_states = np.zeros((N, N, N, 8, len(t_range)))
s_range = np.linspace(-1., 1., N)
omega = 5.
model_gust = {"type": "gust", "params": {"A": 80.,
                                         "gamma": 1.,
                                         "w": omega,
                                         "s_x": 1.,
                                         "s_y": 1.,
                                         "s_z": 1.,
                                         "t_0": 1.}}

start = time.time()
cur_iter = 0
max_iter = N*N*N*8
for i in range(N):
    model_gust['params']['s_x'] = s_range[i]
    for j in range(N):
        model_gust['params']['s_y'] = s_range[j]
        for k in range(N):
            model_gust['params']['s_z'] = s_range[k]
            simulate(BIRE_trim, t_range, BIRE_lin, props, cg_shift, True, model=model_gust)
            save_dir = './Simulation Data/BIRE/'
            save_dir_controlled = save_dir + 'Controlled/'
            z_ctr = np.load(save_dir_controlled + 'shifted_states_CG_' + str(cg_shift[0]) + '.npy')
            MC_states[i, j, k, :] = z_ctr.T
            cur_iter += 1
            prstime = calcProcessTime(start,cur_iter ,max_iter)
            print("time elapsed: %s(s), time left: %s(s), estimated finish time: %s"%prstime)
np.save('./MC_states_w_' + str(int(omega)) + '.npy', MC_states)
```

APPENDIX D

AERODYNAMIC DATABASES

D.1 Baseline Aerodynamic Database

Table D.1: Aerodynamic database generated by MachUpX for the baseline aircraft.

α	β	δ_a	δ_e	δ_r	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
-10	0	-10	0	0	0	0	0	0.077	0	-0.7052	0	0.174	-0
-10	0	0	-20	0	0	0	0	0.0861	-0.0353	-0.5778	0.0346	0.0207	0.0106
-10	0	0	0	0	-90	0	0	0.0425	0.0222	-0.5823	0.0386	0.0222	-0.0115
-10	0	0	0	0	0	-30	0	0.0512	0	-0.6281	0	0.0901	-0
-10	0	0	0	0	0	0	-30	0.046	-0.0182	-0.5809	0.0006	0.0238	0.0071
-10	0	0	0	0	0	0	0	0.0461	0	-0.5801	0	0.0231	-0
-10	0	0	0	0	0	0	30	0.0459	0.0182	-0.581	-0.0006	0.0238	-0.0071
-10	0	0	0	0	0	30	0	0.0389	0	-0.5336	0	-0.0427	-0
-10	0	0	0	0	90	0	0	0.0426	-0.0222	-0.5823	-0.0386	0.0222	0.0115
-10	0	0	20	0	0	0	0	0.0858	0.0353	-0.5778	-0.0346	0.0206	-0.0106
-10	0	10	0	0	0	0	0	0.0398	0	-0.4522	0	-0.1271	-0
-5	0	-10	0	0	0	0	0	0.0366	0	-0.3867	-0	0.1628	-0
-5	0	0	-20	0	0	0	0	0.0522	-0.0371	-0.2561	0.029	0.007	0.0121
-5	0	0	0	0	-90	0	0	0.0113	0.0124	-0.2614	0.0358	0.0124	-0.0066
-5	0	0	0	0	0	-30	0	0.0167	0	-0.3068	0	0.0775	-0
-5	0	0	0	0	0	0	-30	0.0147	-0.018	-0.2586	-0.0006	0.0102	0.0071
-5	0	0	0	0	0	0	0	0.0149	0	-0.2581	0	0.0097	-0
-5	0	0	0	0	0	0	30	0.0146	0.018	-0.2586	0.0006	0.0101	-0.0071
-5	0	0	0	0	0	30	0	0.0107	-0	-0.2104	0	-0.0571	0

Table D.1: Aerodynamic database of the baseline aircraft (continued).

α	β	δ_a	δ_e	δ_r	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
-5	0	0	0	0	90	0	0	0.0113	-0.0124	-0.2614	-0.0358	0.0125	0.0066
-5	0	0	20	0	0	0	0	0.0519	0.0371	-0.2561	-0.029	0.0069	-0.0121
-5	0	10	0	0	0	0	0	0.0179	-0	-0.1278	0	-0.1439	0
0	-6	0	-20	0	0	0	0	0.0415	0.0396	0.0328	0.0338	0.0718	-0.0162
0	-6	0	0	-30	0	0	0	0.0216	-0.0285	0.0314	-0.0061	0.0428	0.0175
0	-6	0	0	0	-90	0	0	0.0053	0.0801	0.0364	0.042	0.0367	-0.0309
0	-6	0	0	0	0	0	-30	0.0101	0.0622	0.0507	0.005	0.01	-0.0233
0	-6	0	0	0	0	0	0	0.0097	0.0764	0.0394	0.0071	0.022	-0.029
0	-6	0	0	0	0	0	30	-0.0375	0.0934	-0.2076	0.0102	0.293	-0.0341
0	-6	0	0	0	90	0	0	0.0066	0.0723	0.0403	-0.0277	0.0096	-0.0268
0	-6	0	0	30	0	0	0	0.0364	0.1803	0.031	0.0211	0.0416	-0.0768
0	-6	0	20	0	0	0	0	0.0472	0.1107	0.0463	-0.0194	-0.0295	-0.041
0	-4	0	-20	0	0	0	0	0.0411	0.0113	0.0454	0.0307	0.0449	-0.0059
0	-4	0	0	-30	0	0	0	0.0222	-0.0549	0.0351	-0.0085	0.0396	0.0277
0	-4	0	0	0	-90	0	0	0.0036	0.0516	0.0341	0.0397	0.0368	-0.0203
0	-4	0	0	0	0	0	-30	0.0079	0.0328	0.0384	0.0031	0.0247	-0.0122
0	-4	0	0	0	0	0	0	0.0079	0.0498	0.0418	0.0047	0.0205	-0.019
0	-4	0	0	0	0	0	30	0.0045	0.0655	-0.0036	0.0068	0.07	-0.0251
0	-4	0	0	0	90	0	0	0.0045	0.0477	0.0345	-0.0301	0.0212	-0.0174
0	-4	0	0	30	0	0	0	0.0323	0.1537	0.0322	0.0185	0.0419	-0.0665

Table D.1: Aerodynamic database of the baseline aircraft (continued).

α	β	δ_a	δ_e	δ_r	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	-4	0	20	0	0	0	0	0.0444	0.087	0.0375	-0.021	-0.0043	-0.0319
0	-2	0	-20	0	0	0	0	0.0409	-0.0155	0.0408	0.0282	0.0363	0.0043
0	-2	0	0	-30	0	0	0	0.0234	-0.0801	0.0374	-0.0106	0.0375	0.0376
0	-2	0	0	0	-90	0	0	0.0032	0.0261	0.0489	0.0371	0.0175	-0.0107
0	-2	0	0	0	0	0	-30	0.007	0.0083	0.0491	0.0003	0.0134	-0.0029
0	-2	0	0	0	0	0	0	0.0071	0.0244	0.0485	0.0027	0.014	-0.0093
0	-2	0	0	0	0	0	30	0.0067	0.042	0.0568	0.0041	0.0045	-0.0164
0	-2	0	0	0	90	0	0	0.0038	0.0237	0.0554	-0.0322	0.0025	-0.0082
0	-2	0	0	30	0	0	0	0.0293	0.1288	0.0431	0.0163	0.0309	-0.0567
0	-2	0	20	0	0	0	0	0.0426	0.0634	0.0556	-0.0227	-0.009	-0.0226
0	0	-10	0	0	0	0	0	0.0174	0	-0.0989	-0	0.1868	-0
0	0	0	-20	0	0	0	0	0.0412	-0.0398	0.031	0.0253	0.0328	0.0137
0	0	0	0	-30	0	0	0	0.0252	-0.1039	0.0232	-0.0134	0.053	0.0468
0	0	0	0	0	-90	0	0	0.0029	0.0006	0.0401	0.0351	0.0239	-0.001
0	0	0	0	0	0	-30	0	0.0044	0	-0.003	-0	0.0848	-0
0	0	0	0	0	0	0	-30	0.0066	-0.0166	0.0543	-0.0021	0.0078	0.0066
0	0	0	0	0	0	0	0	0.0062	0	0.0314	-0	0.033	-0
0	0	0	0	0	0	0	30	0.0065	0.0166	0.0543	0.0021	0.0077	-0.0066
0	0	0	0	0	0	30	0	0.0067	0	0.0871	-0	-0.0422	-0
0	0	0	0	0	90	0	0	0.0029	-0.0006	0.0401	-0.0351	0.0239	0.001

Table D.1: Aerodynamic database of the baseline aircraft (continued).

α	β	δ_a	δ_e	δ_r	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	0	0	0	30	0	0	0	0.0258	0.1038	0.0232	0.0134	0.0534	-0.0468
0	0	0	20	0	0	0	0	0.0409	0.0398	0.031	-0.0253	0.0327	-0.0137
0	0	10	0	0	0	0	0	0.0203	0	0.1621	-0	-0.1224	-0
0	2	0	-20	0	0	0	0	0.0432	-0.0635	0.0555	0.0227	-0.0087	0.0226
0	2	0	0	-30	0	0	0	0.0289	-0.1289	0.0431	-0.0163	0.0307	0.0568
0	2	0	0	0	-90	0	0	0.0039	-0.0236	0.0554	0.0322	0.0025	0.0082
0	2	0	0	0	0	0	-30	0.007	-0.042	0.0568	-0.0041	0.0047	0.0164
0	2	0	0	0	0	0	0	0.0073	-0.0244	0.0485	-0.0027	0.0141	0.0093
0	2	0	0	0	0	0	30	0.007	-0.0083	0.0491	-0.0003	0.0134	0.0029
0	2	0	0	0	90	0	0	0.0034	-0.0261	0.0489	-0.0371	0.0176	0.0107
0	2	0	0	30	0	0	0	0.0243	0.08	0.0374	0.0106	0.0379	-0.0375
0	2	0	20	0	0	0	0	0.0407	0.0155	0.0408	-0.0282	0.0362	-0.0043
0	4	0	-20	0	0	0	0	0.0452	-0.087	0.0375	0.021	-0.004	0.0319
0	4	0	0	-30	0	0	0	0.032	-0.1538	0.0322	-0.0185	0.0417	0.0665
0	4	0	0	0	-90	0	0	0.0049	-0.0477	0.0345	0.0301	0.0213	0.0174
0	4	0	0	0	0	0	-30	0.005	-0.0655	-0.0037	-0.0068	0.0702	0.0251
0	4	0	0	0	0	0	0	0.0083	-0.0498	0.0418	-0.0047	0.0207	0.019
0	4	0	0	0	0	0	30	0.0082	-0.0327	0.0384	-0.0031	0.0248	0.0123
0	4	0	0	0	90	0	0	0.004	-0.0516	0.0341	-0.0397	0.0371	0.0203
0	4	0	0	30	0	0	0	0.0232	0.0548	0.0351	0.0085	0.0401	-0.0276

Table D.1: Aerodynamic database of the baseline aircraft (continued).

α	β	δ_a	δ_e	δ_r	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	4	0	20	0	0	0	0	0.0411	-0.0113	0.0454	-0.0307	0.0449	0.0059
0	6	0	-20	0	0	0	0	0.0482	-0.1107	0.0463	0.0194	-0.029	0.0411
0	6	0	0	-30	0	0	0	0.0363	-0.1804	0.031	-0.0211	0.0415	0.0768
0	6	0	0	0	-90	0	0	0.0072	-0.0722	0.0403	0.0277	0.0098	0.0268
0	6	0	0	0	0	0	-30	-0.0369	-0.0934	-0.2076	-0.0102	0.2934	0.0341
0	6	0	0	0	0	0	0	0.0103	-0.0763	0.0394	-0.0071	0.0223	0.029
0	6	0	0	0	0	0	30	0.0105	-0.0622	0.0507	-0.005	0.0103	0.0233
0	6	0	0	0	90	0	0	0.0059	-0.08	0.0364	-0.042	0.037	0.0309
0	6	0	0	30	0	0	0	0.0228	0.0284	0.0314	0.006	0.0435	-0.0174
0	6	0	20	0	0	0	0	0.0417	-0.0396	0.0327	-0.0338	0.072	0.0162
5	0	-10	0	0	0	0	0	0.0215	0	0.2328	-0	0.1619	-0
5	0	0	-20	0	0	0	0	0.058	-0.0431	0.3644	0.0292	0.0067	0.0165
5	0	0	0	0	-90	0	0	0.0163	-0.0104	0.3658	0.0356	0.0081	0.0045
5	0	0	0	0	0	-30	0	0.0147	0	0.319	-0	0.0721	-0
5	0	0	0	0	0	0	-30	0.0198	-0.015	0.3644	-0.0033	0.0079	0.0061
5	0	0	0	0	0	0	0	0.0199	0	0.3645	-0	0.0078	-0
5	0	0	0	0	0	0	30	0.0197	0.015	0.3645	0.0033	0.0078	-0.0061
5	0	0	0	0	0	30	0	0.0231	0	0.4104	-0	-0.0567	-0
5	0	0	0	0	90	0	0	0.0163	0.0104	0.3658	-0.0356	0.0081	-0.0045
5	0	0	20	0	0	0	0	0.0576	0.0431	0.3645	-0.0292	0.0065	-0.0165

Table D.1: Aerodynamic database of the baseline aircraft (continued).

α	β	δ_a	δ_e	δ_r	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
5	0	10	0	0	0	0	0	0.0439	0	0.4953	-0	-0.1492	-0
10	0	-10	0	0	0	0	0	0.0519	0	0.5706	-0	0.1296	-0
10	0	0	-20	0	0	0	0	0.0992	-0.0475	0.7024	0.0319	-0.0261	0.0202
10	0	0	0	0	-90	0	0	0.056	-0.0259	0.7061	0.0368	-0.0246	0.0118
10	0	0	0	0	0	-30	0	0.052	0	0.6579	-0	0.0394	-0
10	0	0	0	0	0	0	-30	0.0593	-0.0139	0.7023	-0.0045	-0.0238	0.0056
10	0	0	0	0	0	0	0	0.0594	0	0.7024	-0	-0.024	-0
10	0	0	0	0	0	0	30	0.0592	0.0139	0.7023	0.0045	-0.0239	-0.0056
10	0	0	0	0	0	30	0	0.0649	0	0.7477	-0	-0.0878	-0
10	0	0	0	0	90	0	0	0.056	0.0259	0.7061	-0.0368	-0.0246	-0.0118
10	0	0	20	0	0	0	0	0.0988	0.0475	0.7025	-0.0319	-0.0264	-0.0202
10	0	10	0	0	0	0	0	0.093	0	0.8319	-0	-0.1818	-0

D.2 BIRE Aerodynamic Database

Table D.2: Truncated aerodynamic database generated by MachUpX for the BIRE aircraft.

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
-10	0	-10	0	-90	0	0	0	0.0487	0.1333	-0.5147	0.002	-0.0513	-0.0638
-10	0	-10	0	-60	0	0	0	0.0586	0.1445	-0.6061	0.0001	0.0609	-0.0701
-10	0	-10	0	-30	0	0	0	0.0732	0.098	-0.6957	0.0023	0.165	-0.0472
-10	0	-10	0	0	0	0	0	0.0778	0	-0.7208	0	0.1926	-0
-10	0	-10	0	30	0	0	0	0.0732	-0.098	-0.6957	-0.0023	0.165	0.0472
-10	0	-10	0	60	0	0	0	0.0586	-0.1445	-0.6061	-0.0001	0.0609	0.0701
-10	0	-10	0	90	0	0	0	0.0487	-0.1333	-0.5147	-0.002	-0.0513	0.0638
-10	0	0	-20	-90	0	0	0	0.0775	-0.0197	-0.5147	0.0432	-0.0538	0.0051
-10	0	0	-20	-60	0	0	0	0.0716	0.0194	-0.5329	0.04	-0.0252	-0.0136
-10	0	0	-20	-30	0	0	0	0.0828	0.0273	-0.5692	0.0423	0.0164	-0.0154
-10	0	0	-20	0	0	0	0	0.0866	-0.0008	-0.5896	0.0388	0.0342	-0.001
-10	0	0	-20	30	0	0	0	0.087	-0.0419	-0.591	0.0399	0.033	0.0178
-10	0	0	-20	60	0	0	0	0.082	-0.0516	-0.5469	0.0449	-0.0168	0.022
-10	0	0	-20	90	0	0	0	0.0776	-0.0197	-0.5147	0.0432	-0.0535	0.0052
-10	0	0	0	-90	-90	0	0	0.0331	0.013	-0.5188	0.0356	-0.0534	-0.0071
-10	0	0	0	-90	0	-30	0	0.0364	0	-0.5081	0	-0.0503	-0
-10	0	0	0	-90	0	0	-30	0.0356	-0.0474	-0.516	0.0019	-0.0536	0.0221
-10	0	0	0	-90	0	0	0	0.0366	0	-0.5158	0	-0.0535	-0
-10	0	0	0	-90	0	0	30	0.0358	0.0474	-0.5159	-0.0019	-0.0536	-0.0221

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
-10	0	0	0	-90	0	30	0	0.0366	0	-0.5236	0	-0.0568	-0
-10	0	0	0	-90	90	0	0	0.0331	-0.013	-0.5188	-0.0356	-0.0534	0.0071
-10	0	0	0	-60	-90	0	0	0.0313	0.0484	-0.541	0.033	-0.0242	-0.025
-10	0	0	0	-60	0	-30	0	0.0371	0.0561	-0.5457	-0.0018	-0.0035	-0.0274
-10	0	0	0	-60	0	0	-30	0.0341	0.0019	-0.5187	0.0001	-0.0474	-0.0022
-10	0	0	0	-60	0	0	0	0.0361	0.0358	-0.5406	-0.0024	-0.0215	-0.018
-10	0	0	0	-60	0	0	30	0.0367	0.0702	-0.5634	-0.0052	0.0054	-0.0341
-10	0	0	0	-60	0	30	0	0.0347	0.0155	-0.5356	-0.0029	-0.0395	-0.0087
-10	0	0	0	-60	90	0	0	0.0341	0.0232	-0.5459	-0.0377	-0.0187	-0.0111
-10	0	0	0	-30	-90	0	0	0.0401	0.0526	-0.5848	0.0361	0.0265	-0.0263
-10	0	0	0	-30	0	-30	0	0.0472	0.0578	-0.6153	0.0013	0.076	-0.0276
-10	0	0	0	-30	0	0	-30	0.042	0.0245	-0.5612	0.0036	0.002	-0.012
-10	0	0	0	-30	0	0	0	0.044	0.0356	-0.5816	0.001	0.0255	-0.0173
-10	0	0	0	-30	0	0	30	0.0457	0.0467	-0.6021	-0.0017	0.0492	-0.0226
-10	0	0	0	-30	0	30	0	0.0393	0.0135	-0.5486	0.0009	-0.0244	-0.007
-10	0	0	0	-30	90	0	0	0.0415	0.02	-0.5871	-0.0342	0.0281	-0.009
-10	0	0	0	0	-90	0	0	0.0428	0.0174	-0.5943	0.0362	0.036	-0.009
-10	0	0	0	0	0	-30	0	0.0508	0	-0.6391	0	0.1022	-0
-10	0	0	0	0	0	0	-30	0.0461	-0.0002	-0.5915	0.0025	0.0359	0.0002
-10	0	0	0	0	0	0	0	0.0461	0	-0.5915	0	0.0359	-0

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\delta_a}$	$\boldsymbol{\delta_e}$	$\boldsymbol{\delta_B}$	$\boldsymbol{p}$	$\boldsymbol{q}$	$\boldsymbol{r}$	$\boldsymbol{C_D}$	$\boldsymbol{C_S}$	$\boldsymbol{C_L}$	$\boldsymbol{C_\ell}$	$\boldsymbol{C_m}$	$\boldsymbol{C_n}$
-10	0	0	0	0	0	0	30	0.0461	0.0002	-0.5915	-0.0025	0.0359	-0.0002
-10	0	0	0	0	0	30	0	0.0395	0	-0.5453	0	-0.0294	-0
-10	0	0	0	0	90	0	0	0.0428	-0.0174	-0.5943	-0.0362	0.036	0.009
-10	0	0	0	30	-90	0	0	0.0415	-0.02	-0.5871	0.0342	0.0281	0.009
-10	0	0	0	30	0	-30	0	0.0472	-0.0578	-0.6153	-0.0013	0.076	0.0276
-10	0	0	0	30	0	0	-30	0.0457	-0.0467	-0.6021	0.0017	0.0492	0.0226
-10	0	0	0	30	0	0	0	0.044	-0.0356	-0.5816	-0.001	0.0255	0.0173
-10	0	0	0	30	0	0	30	0.042	-0.0245	-0.5612	-0.0036	0.002	0.012
-10	0	0	0	30	0	30	0	0.0393	-0.0135	-0.5486	-0.0009	-0.0244	0.007
-10	0	0	0	30	90	0	0	0.0401	-0.0526	-0.5848	-0.0361	0.0265	0.0263
-10	0	0	0	60	-90	0	0	0.0341	-0.0232	-0.5459	0.0377	-0.0187	0.0111
-10	0	0	0	60	0	-30	0	0.0371	-0.0561	-0.5457	0.0018	-0.0035	0.0274
-10	0	0	0	60	0	0	-30	0.0367	-0.0702	-0.5634	0.0052	0.0054	0.0341
-10	0	0	0	60	0	0	0	0.0361	-0.0358	-0.5406	0.0024	-0.0215	0.018
-10	0	0	0	60	0	0	30	0.0341	-0.0019	-0.5187	-0.0001	-0.0474	0.0022
-10	0	0	0	60	0	30	0	0.0347	-0.0155	-0.5356	0.0029	-0.0395	0.0087
-10	0	0	0	60	90	0	0	0.0313	-0.0484	-0.541	-0.033	-0.0242	0.025
-10	0	0	0	90	-90	0	0	0.0331	0.013	-0.5188	0.0356	-0.0534	-0.0071
-10	0	0	0	90	0	-30	0	0.0364	0	-0.5081	0	-0.0503	-0
-10	0	0	0	90	0	0	-30	0.0358	-0.0474	-0.5159	0.0019	-0.0536	0.0221

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
-10	0	0	0	90	0	0	0	0.0366	0	-0.5158	0	-0.0535	-0
-10	0	0	0	90	0	0	30	0.0356	0.0474	-0.516	-0.0019	-0.0536	-0.0221
-10	0	0	0	90	0	30	0	0.0366	0	-0.5236	0	-0.0568	-0
-10	0	0	0	90	90	0	0	0.0331	-0.013	-0.5188	-0.0356	-0.0534	0.0071
-10	0	0	20	-90	0	0	0	0.0776	0.0197	-0.5147	-0.0432	-0.0535	-0.0052
-10	0	0	20	-60	0	0	0	0.082	0.0516	-0.5469	-0.0449	-0.0168	-0.022
-10	0	0	20	-30	0	0	0	0.087	0.0419	-0.591	-0.0399	0.033	-0.0178
-10	0	0	20	0	0	0	0	0.0866	0.0008	-0.5896	-0.0388	0.0342	0.001
-10	0	0	20	30	0	0	0	0.0828	-0.0273	-0.5692	-0.0423	0.0164	0.0154
-10	0	0	20	60	0	0	0	0.0716	-0.0194	-0.5329	-0.04	-0.0252	0.0136
-10	0	0	20	90	0	0	0	0.0775	0.0197	-0.5147	-0.0432	-0.0538	-0.0051
-10	0	10	0	-90	0	0	0	0.0492	-0.1332	-0.5147	-0.002	-0.0512	0.0638
-10	0	10	0	-60	0	0	0	0.0385	-0.073	-0.4738	-0.005	-0.0984	0.0344
-10	0	10	0	-30	0	0	0	0.04	-0.0271	-0.4655	-0.0002	-0.1102	0.013
-10	0	10	0	0	0	0	0	0.0398	0	-0.4599	0	-0.118	-0
-10	0	10	0	30	0	0	0	0.04	0.0271	-0.4655	0.0002	-0.1102	-0.013
-10	0	10	0	60	0	0	0	0.0385	0.073	-0.4738	0.005	-0.0984	-0.0344
-10	0	10	0	90	0	0	0	0.0492	0.1332	-0.5147	0.002	-0.0512	-0.0638
-5	0	-10	0	-90	0	0	0	0.0236	0.1282	-0.2204	0.0014	-0.0336	-0.0615
-5	0	-10	0	-60	0	0	0	0.0291	0.135	-0.3033	0.0003	0.0656	-0.0648

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
-5	0	-10	0	-30	0	0	0	0.034	0.0842	-0.3763	-0	0.1518	-0.0404
-5	0	-10	0	0	0	0	0	0.036	0	-0.3949	0	0.1727	-0
-5	0	-10	0	30	0	0	0	0.034	-0.0842	-0.3763	0	0.1518	0.0404
-5	0	-10	0	60	0	0	0	0.0291	-0.135	-0.3033	-0.0003	0.0656	0.0648
-5	0	-10	0	90	0	0	0	0.0236	-0.1282	-0.2204	-0.0014	-0.0336	0.0615
-5	0	0	-20	-90	0	0	0	0.0524	-0.0092	-0.2204	0.0414	-0.0353	0.0025
-5	0	0	-20	-60	0	0	0	0.0514	0.0157	-0.2299	0.0396	-0.0211	-0.0092
-5	0	0	-20	-30	0	0	0	0.0507	0.0132	-0.2462	0.0371	-0.0002	-0.0072
-5	0	0	-20	0	0	0	0	0.0518	-0.0002	-0.2617	0.0334	0.0132	-0.0001
-5	0	0	-20	30	0	0	0	0.0546	-0.0283	-0.2742	0.0382	0.025	0.0125
-5	0	0	-20	60	0	0	0	0.0541	-0.0331	-0.2412	0.0414	-0.0122	0.0143
-5	0	0	-20	90	0	0	0	0.0525	-0.0093	-0.2204	0.0414	-0.035	0.0025
-5	0	0	0	-90	-90	0	0	0.0078	0.0055	-0.2221	0.035	-0.035	-0.003
-5	0	0	0	-90	0	-30	0	0.0114	0	-0.2129	0	-0.0319	-0
-5	0	0	0	-90	0	0	-30	0.0105	-0.0454	-0.2208	0.0006	-0.0351	0.0212
-5	0	0	0	-90	0	0	0	0.0115	0	-0.2208	0	-0.035	-0
-5	0	0	0	-90	0	0	30	0.0107	0.0454	-0.2208	-0.0006	-0.0351	-0.0212
-5	0	0	0	-90	0	30	0	0.0115	0	-0.2286	0	-0.0382	-0
-5	0	0	0	-90	90	0	0	0.0079	-0.0055	-0.2221	-0.035	-0.035	0.003
-5	0	0	0	-60	-90	0	0	0.0083	0.0322	-0.237	0.0339	-0.0171	-0.0159

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
-5	0	0	0	-60	0	-30	0	0.0123	0.0457	-0.2412	-0.0004	0.0016	-0.0217
-5	0	0	0	-60	0	0	-30	0.0105	-0.0093	-0.215	-0.0002	-0.0409	0.0039
-5	0	0	0	-60	0	0	0	0.012	0.0245	-0.2359	-0.0009	-0.0166	-0.0119
-5	0	0	0	-60	0	0	30	0.0123	0.0591	-0.2574	-0.0018	0.0082	-0.028
-5	0	0	0	-60	0	30	0	0.0111	0.0024	-0.23	-0.0012	-0.0355	-0.0017
-5	0	0	0	-60	90	0	0	0.0087	0.018	-0.2383	-0.0356	-0.0154	-0.0084
-5	0	0	0	-30	-90	0	0	0.0096	0.0292	-0.2615	0.0333	0.0126	-0.0144
-5	0	0	0	-30	0	-30	0	0.0144	0.0432	-0.2936	-0.0003	0.0626	-0.0204
-5	0	0	0	-30	0	0	-30	0.0124	0.0103	-0.2401	0.0005	-0.0109	-0.0051
-5	0	0	0	-30	0	0	0	0.0134	0.0214	-0.2606	-0.0006	0.0128	-0.0103
-5	0	0	0	-30	0	0	30	0.014	0.0324	-0.2811	-0.0018	0.0365	-0.0155
-5	0	0	0	-30	0	30	0	0.0109	-0.0004	-0.2279	-0.0007	-0.0368	-0.0002
-5	0	0	0	-30	90	0	0	0.0103	0.014	-0.2633	-0.0343	0.0139	-0.0064
-5	0	0	0	0	-90	0	0	0.0104	0.0082	-0.2632	0.0343	0.0139	-0.0042
-5	0	0	0	0	0	-30	0	0.0157	0	-0.3114	0	0.0823	0
-5	0	0	0	0	0	0	-30	0.014	-0.0001	-0.2633	0.001	0.0153	0.0001
-5	0	0	0	0	0	0	0	0.014	0	-0.2633	0	0.0153	-0
-5	0	0	0	0	0	0	30	0.014	0.0001	-0.2633	-0.001	0.0153	-0.0001
-5	0	0	0	0	0	30	0	0.0101	0	-0.2159	0	-0.051	-0
-5	0	0	0	0	90	0	0	0.0104	-0.0082	-0.2632	-0.0343	0.0139	0.0042

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
-5	0	0	0	30	-90	0	0	0.0103	-0.014	-0.2633	0.0343	0.0139	0.0064
-5	0	0	0	30	0	-30	0	0.0144	-0.0432	-0.2936	0.0003	0.0626	0.0204
-5	0	0	0	30	0	0	-30	0.014	-0.0324	-0.2811	0.0018	0.0365	0.0155
-5	0	0	0	30	0	0	0	0.0134	-0.0214	-0.2606	0.0006	0.0128	0.0103
-5	0	0	0	30	0	0	30	0.0124	-0.0103	-0.2401	-0.0005	-0.0109	0.0051
-5	0	0	0	30	0	30	0	0.0109	0.0004	-0.2279	0.0007	-0.0368	0.0002
-5	0	0	0	30	90	0	0	0.0096	-0.0292	-0.2615	-0.0333	0.0126	0.0144
-5	0	0	0	60	-90	0	0	0.0087	-0.018	-0.2383	0.0356	-0.0154	0.0084
-5	0	0	0	60	0	-30	0	0.0123	-0.0457	-0.2412	0.0004	0.0016	0.0217
-5	0	0	0	60	0	0	-30	0.0123	-0.0591	-0.2574	0.0018	0.0082	0.028
-5	0	0	0	60	0	0	0	0.012	-0.0245	-0.2359	0.0009	-0.0166	0.0119
-5	0	0	0	60	0	0	30	0.0105	0.0093	-0.215	0.0002	-0.0409	-0.0039
-5	0	0	0	60	0	30	0	0.0111	-0.0024	-0.23	0.0012	-0.0355	0.0017
-5	0	0	0	60	90	0	0	0.0083	-0.0322	-0.237	-0.0339	-0.0171	0.0159
-5	0	0	0	90	-90	0	0	0.0079	0.0055	-0.2221	0.035	-0.035	-0.003
-5	0	0	0	90	0	-30	0	0.0114	0	-0.2129	0	-0.0319	-0
-5	0	0	0	90	0	0	-30	0.0107	-0.0454	-0.2208	0.0006	-0.0351	0.0212
-5	0	0	0	90	0	0	0	0.0115	0	-0.2208	0	-0.035	-0
-5	0	0	0	90	0	0	30	0.0105	0.0454	-0.2208	-0.0006	-0.0351	-0.0212
-5	0	0	0	90	0	30	0	0.0115	0	-0.2286	0	-0.0382	-0

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
-5	0	0	0	90	90	0	0	0.0078	-0.0055	-0.2221	-0.035	-0.035	0.003
-5	0	0	20	-90	0	0	0	0.0525	0.0093	-0.2204	-0.0414	-0.035	-0.0025
-5	0	0	20	-60	0	0	0	0.0541	0.0331	-0.2412	-0.0414	-0.0122	-0.0143
-5	0	0	20	-30	0	0	0	0.0546	0.0283	-0.2742	-0.0382	0.025	-0.0125
-5	0	0	20	0	0	0	0	0.0518	0.0002	-0.2617	-0.0334	0.0132	0.0001
-5	0	0	20	30	0	0	0	0.0507	-0.0132	-0.2462	-0.0371	-0.0002	0.0072
-5	0	0	20	60	0	0	0	0.0514	-0.0157	-0.2299	-0.0396	-0.0211	0.0092
-5	0	0	20	90	0	0	0	0.0524	0.0092	-0.2204	-0.0414	-0.0353	-0.0025
-5	0	10	0	-90	0	0	0	0.0241	-0.1282	-0.2204	-0.0014	-0.0336	0.0615
-5	0	10	0	-60	0	0	0	0.0199	-0.0859	-0.1678	-0.0021	-0.0962	0.0412
-5	0	10	0	-30	0	0	0	0.0179	-0.0414	-0.1441	-0.0012	-0.1243	0.0199
-5	0	10	0	0	0	0	0	0.0171	0	-0.1306	0	-0.1406	-0
-5	0	10	0	30	0	0	0	0.0179	0.0414	-0.1441	0.0012	-0.1243	-0.0199
-5	0	10	0	60	0	0	0	0.0199	0.0859	-0.1678	0.0021	-0.0962	-0.0412
-5	0	10	0	90	0	0	0	0.0241	0.1282	-0.2204	0.0014	-0.0336	-0.0615
0	-6	0	-20	-90	0	0	0	0.0556	0.1457	0.1147	0.0435	-0.0152	-0.0682
0	-6	0	-20	-60	0	0	0	0.053	0.1251	0.0385	0.0422	0.0716	-0.0576
0	-6	0	-20	-30	0	0	0	0.0474	0.0573	0.0099	0.0385	0.1014	-0.0255
0	-6	0	-20	0	0	0	0	0.0032	-0.001	-0.162	0.0326	0.2898	0.0045
0	-6	0	-20	30	0	0	0	0.0479	0.0166	0.1397	0.0391	-0.0482	-0.009

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	-6	0	-20	60	0	0	0	0.0532	0.0932	0.1704	0.0429	-0.0804	-0.0446
0	-6	0	-20	90	0	0	0	0.0547	0.1456	0.1147	0.0435	-0.0149	-0.0681
0	-6	0	0	-90	-90	0	0	0.0092	0.1415	0.0866	0.037	-0.0184	-0.0657
0	-6	0	0	-90	0	-30	0	0.0127	0.143	0.0831	0.0028	-0.0141	-0.0667
0	-6	0	0	-90	0	0	-30	0.0124	0.0977	0.075	0.0014	-0.0168	-0.0456
0	-6	0	0	-90	0	0	0	0.0127	0.1429	0.0752	0.0019	-0.017	-0.0667
0	-6	0	0	-90	0	0	30	0.0114	0.1887	0.0755	0.0025	-0.0172	-0.0879
0	-6	0	0	-90	0	30	0	0.0127	0.1429	0.0674	0.0011	-0.0199	-0.0666
0	-6	0	0	-90	90	0	0	0.0092	0.1451	0.0644	-0.0331	-0.0156	-0.0676
0	-6	0	0	-60	-90	0	0	0.0075	0.1172	0.0144	0.0364	0.0649	-0.0544
0	-6	0	0	-60	0	-30	0	0.0103	0.1367	-0.0008	0.0024	0.0825	-0.0637
0	-6	0	0	-60	0	0	-30	0.011	0.0809	0.025	0.0012	0.0407	-0.0377
0	-6	0	0	-60	0	0	0	0.011	0.1147	0.0046	0.0015	0.0643	-0.0535
0	-6	0	0	-60	0	0	30	0.0098	0.149	-0.0159	0.0017	0.0881	-0.0694
0	-6	0	0	-60	0	30	0	0.0112	0.0939	0.0094	0.0009	0.0468	-0.0438
0	-6	0	0	-60	90	0	0	0.0075	0.1132	-0.0049	-0.0332	0.0636	-0.0526
0	-6	0	0	-30	-90	0	0	0.0039	0.0485	-0.0033	0.0354	0.0856	-0.0223
0	-6	0	0	-30	0	-30	0	0.0059	0.0669	-0.04	0.0019	0.1269	-0.0311
0	-6	0	0	-30	0	0	-30	0.0078	0.0334	0.0128	0.0011	0.0543	-0.0156
0	-6	0	0	-30	0	0	0	0.0076	0.0447	-0.0075	0.0014	0.0778	-0.0208

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	-6	0	0	-30	0	0	30	0.007	0.056	-0.0278	0.0014	0.1013	-0.0261
0	-6	0	0	-30	0	30	0	0.0076	0.0228	0.0247	0.001	0.0289	-0.0107
0	-6	0	0	-30	90	0	0	0.0042	0.0412	-0.0107	-0.0324	0.0691	-0.0192
0	-6	0	0	0	-90	0	0	0.0016	-0.0003	0.0397	0.034	0.0363	0.0004
0	-6	0	0	0	0	-30	0	0.0031	0.0004	-0.0077	0.001	0.0893	-0.0002
0	-6	0	0	0	0	0	-30	0.0005	0.0006	-0.0319	0.0017	0.1022	-0.0002
0	-6	0	0	0	0	0	0	-0.0129	0.0025	-0.1127	0.0029	0.1911	-0.0007
0	-6	0	0	0	0	0	30	0.0013	0.0015	0.0221	-0.0028	0.0384	-0.0011
0	-6	0	0	0	0	30	0	0.0055	0.0001	0.0949	0.001	-0.0517	-0.0001
0	-6	0	0	0	90	0	0	0.0024	0.0011	0.0518	-0.0316	-0.0032	-0.0005
0	-6	0	0	30	-90	0	0	0.0036	0.0196	0.1241	0.0355	-0.0604	-0.0091
0	-6	0	0	30	0	-30	0	0.0062	0.0042	0.0906	0.0022	-0.0229	-0.0022
0	-6	0	0	30	0	0	-30	0.0069	0.0146	0.1022	0.0013	-0.0482	-0.0069
0	-6	0	0	30	0	0	0	0.0072	0.0256	0.1226	0.0021	-0.0716	-0.0121
0	-6	0	0	30	0	0	30	0.0071	0.0366	0.143	0.003	-0.095	-0.0173
0	-6	0	0	30	0	30	0	0.0067	0.0474	0.1548	0.0017	-0.1206	-0.0222
0	-6	0	0	30	90	0	0	0.0038	0.0332	0.1233	-0.0317	-0.0846	-0.0154
0	-6	0	0	60	-90	0	0	0.0068	0.0936	0.1447	0.0369	-0.0849	-0.0435
0	-6	0	0	60	0	-30	0	0.0103	0.0774	0.1308	0.0028	-0.069	-0.0362
0	-6	0	0	60	0	0	-30	0.0101	0.065	0.1152	0.0015	-0.063	-0.0304

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	-6	0	0	60	0	0	0	0.0104	0.0985	0.1359	0.0023	-0.0869	-0.046
0	-6	0	0	60	0	0	30	0.0095	0.1325	0.1568	0.0033	-0.111	-0.0618
0	-6	0	0	60	0	30	0	0.01	0.1202	0.1413	0.0016	-0.1052	-0.0561
0	-6	0	0	60	90	0	0	0.0068	0.1043	0.1278	-0.0325	-0.0891	-0.0486
0	-6	0	0	90	-90	0	0	0.0084	0.1415	0.0866	0.037	-0.0185	-0.0657
0	-6	0	0	90	0	-30	0	0.0119	0.143	0.0832	0.0028	-0.0141	-0.0667
0	-6	0	0	90	0	0	-30	0.0119	0.0977	0.075	0.0014	-0.0168	-0.0456
0	-6	0	0	90	0	0	0	0.0119	0.1429	0.0753	0.0019	-0.017	-0.0666
0	-6	0	0	90	0	0	30	0.0103	0.1887	0.0755	0.0025	-0.0173	-0.0879
0	-6	0	0	90	0	30	0	0.0119	0.1429	0.0674	0.0011	-0.0199	-0.0666
0	-6	0	0	90	90	0	0	0.0083	0.1451	0.0644	-0.0331	-0.0156	-0.0675
0	-4	0	-20	-90	0	0	0	0.0515	0.0982	0.1016	0.0428	-0.0154	-0.0458
0	-4	0	-20	-60	0	0	0	0.05	0.0877	0.0486	0.0416	0.0449	-0.0402
0	-4	0	-20	-30	0	0	0	0.0463	0.0429	0.0242	0.0377	0.0706	-0.0189
0	-4	0	-20	0	0	0	0	-0.2172	0.0469	0.7289	0.0294	-0.7077	-0.0181
0	-4	0	-20	30	0	0	0	0.0465	0.0082	0.1112	0.038	-0.0296	-0.0046
0	-4	0	-20	60	0	0	0	0.0499	0.06	0.1369	0.0419	-0.0567	-0.0287
0	-4	0	-20	90	0	0	0	0.0509	0.0982	0.1017	0.0428	-0.0151	-0.0458
0	-4	0	0	-90	-90	0	0	0.0055	0.094	0.083	0.0364	-0.0176	-0.0436
0	-4	0	0	-90	0	-30	0	0.009	0.0955	0.0833	0.0022	-0.0137	-0.0446

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\delta_a}$	$\boldsymbol{\delta_e}$	$\boldsymbol{\delta_B}$	$\boldsymbol{p}$	$\boldsymbol{q}$	$\boldsymbol{r}$	$\boldsymbol{C_D}$	$\boldsymbol{C_S}$	$\boldsymbol{C_L}$	$\boldsymbol{C_\ell}$	$\boldsymbol{C_m}$	$\boldsymbol{C_n}$
0	-4	0	0	-90	0	0	-30	0.0085	0.0504	0.0752	0.0011	-0.0166	-0.0235
0	-4	0	0	-90	0	0	0	0.009	0.0954	0.0754	0.0015	-0.0167	-0.0445
0	-4	0	0	-90	0	0	30	0.0079	0.1407	0.0756	0.002	-0.0169	-0.0656
0	-4	0	0	-90	0	30	0	0.009	0.0953	0.0675	0.0009	-0.0198	-0.0445
0	-4	0	0	-90	90	0	0	0.0054	0.0976	0.0683	-0.0334	-0.0159	-0.0455
0	-4	0	0	-60	-90	0	0	0.0047	0.0807	0.033	0.036	0.04	-0.0374
0	-4	0	0	-60	0	-30	0	0.0076	0.1017	0.0206	0.0019	0.0582	-0.0473
0	-4	0	0	-60	0	0	-30	0.0081	0.0461	0.0467	0.001	0.0161	-0.0214
0	-4	0	0	-60	0	0	0	0.0083	0.0797	0.0262	0.0013	0.0398	-0.0371
0	-4	0	0	-60	0	0	30	0.0072	0.1136	0.0057	0.0015	0.0636	-0.053
0	-4	0	0	-60	0	30	0	0.0083	0.058	0.0317	0.0008	0.0214	-0.027
0	-4	0	0	-60	90	0	0	0.0046	0.0791	0.0199	-0.0332	0.0393	-0.0368
0	-4	0	0	-30	-90	0	0	0.003	0.0355	0.0165	0.0349	0.059	-0.0162
0	-4	0	0	-30	0	-30	0	0.005	0.0556	-0.0196	0.0014	0.1038	-0.0258
0	-4	0	0	-30	0	0	-30	0.0068	0.0223	0.0335	0.0008	0.0308	-0.0103
0	-4	0	0	-30	0	0	0	0.0066	0.0333	0.0134	0.0011	0.054	-0.0155
0	-4	0	0	-30	0	0	30	0.006	0.0446	-0.007	0.0012	0.0776	-0.0208
0	-4	0	0	-30	0	30	0	0.0067	0.016	0.0377	0.0009	0.0138	-0.0074
0	-4	0	0	-30	90	0	0	0.0031	0.0314	0.0113	-0.0324	0.0483	-0.0146
0	-4	0	0	0	-90	0	0	0.0017	-0.0005	0.0388	0.0336	0.0333	0.0004

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	-4	0	0	0	0	-30	0	0.0031	0.0002	-0.0075	0.0007	0.0895	-0.0001
0	-4	0	0	0	0	0	-30	0.0037	0.0004	0.0023	0.0028	0.0656	-0.0003
0	-4	0	0	0	0	0	0	-0.1808	0.0142	0.6111	0.0026	-0.6041	0.0005
0	-4	0	0	0	0	0	30	-0.0055	0.0019	0.1604	-0.0077	-0.1098	-0.0016
0	-4	0	0	0	0	30	0	0.0055	0.0001	0.0944	0.0007	-0.0508	-0.0001
0	-4	0	0	0	90	0	0	0.0023	0.001	0.0514	-0.032	0.002	-0.0005
0	-4	0	0	30	-90	0	0	0.0028	0.0105	0.1036	0.0349	-0.0407	-0.0048
0	-4	0	0	30	0	-30	0	0.004	-0.0168	0.0514	0.0016	0.0215	0.0075
0	-4	0	0	30	0	0	-30	0.006	0.0025	0.0798	0.0008	-0.0222	-0.0013
0	-4	0	0	30	0	0	0	0.0064	0.0137	0.1005	0.0014	-0.0461	-0.0065
0	-4	0	0	30	0	0	30	0.0064	0.0247	0.1209	0.0022	-0.0695	-0.0117
0	-4	0	0	30	0	30	0	0.006	0.0355	0.1331	0.0011	-0.0954	-0.0166
0	-4	0	0	30	90	0	0	0.0029	0.0184	0.1001	-0.0324	-0.0537	-0.0087
0	-4	0	0	60	-90	0	0	0.0042	0.0586	0.1196	0.0362	-0.0595	-0.0273
0	-4	0	0	60	0	-30	0	0.0075	0.0412	0.1086	0.0021	-0.043	-0.0194
0	-4	0	0	60	0	0	-30	0.0073	0.0292	0.0933	0.001	-0.0374	-0.0137
0	-4	0	0	60	0	0	0	0.0078	0.0626	0.114	0.0017	-0.0613	-0.0293
0	-4	0	0	60	0	0	30	0.0071	0.0963	0.1348	0.0024	-0.0853	-0.045
0	-4	0	0	60	0	30	0	0.0075	0.0841	0.1193	0.0011	-0.0795	-0.0393
0	-4	0	0	60	90	0	0	0.0042	0.067	0.1088	-0.033	-0.063	-0.0313

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\delta_a}$	$\boldsymbol{\delta_e}$	$\boldsymbol{\delta_B}$	$\boldsymbol{p}$	$\boldsymbol{q}$	$\boldsymbol{r}$	$\boldsymbol{C_D}$	$\boldsymbol{C_S}$	$\boldsymbol{C_L}$	$\boldsymbol{C_\ell}$	$\boldsymbol{C_m}$	$\boldsymbol{C_n}$
0	-4	0	0	90	-90	0	0	0.0049	0.094	0.083	0.0364	-0.0177	-0.0436
0	-4	0	0	90	0	-30	0	0.0084	0.0955	0.0833	0.0022	-0.0137	-0.0445
0	-4	0	0	90	0	0	-30	0.0082	0.0505	0.0753	0.0011	-0.0166	-0.0235
0	-4	0	0	90	0	0	0	0.0085	0.0954	0.0754	0.0015	-0.0168	-0.0445
0	-4	0	0	90	0	0	30	0.0071	0.1407	0.0756	0.002	-0.0169	-0.0656
0	-4	0	0	90	0	30	0	0.0085	0.0953	0.0675	0.0009	-0.0198	-0.0445
0	-4	0	0	90	90	0	0	0.0048	0.0976	0.0683	-0.0334	-0.0159	-0.0454
0	-2	0	-20	-90	0	0	0	0.0485	0.0506	0.0885	0.0417	-0.016	-0.0234
0	-2	0	-20	-60	0	0	0	0.0478	0.0497	0.0592	0.0406	0.0174	-0.0225
0	-2	0	-20	-30	0	0	0	0.0454	0.0277	0.0403	0.0368	0.0377	-0.0121
0	-2	0	-20	0	0	0	0	-4.8944	0.0442	3.1443	0.1609	-3.4035	0.3269
0	-2	0	-20	30	0	0	0	0.0454	0.0003	0.0838	0.0368	-0.0126	-0.0003
0	-2	0	-20	60	0	0	0	0.0477	0.027	0.1034	0.0407	-0.0336	-0.0128
0	-2	0	-20	90	0	0	0	0.0482	0.0506	0.0885	0.0417	-0.0156	-0.0234
0	-2	0	0	-90	-90	0	0	0.0032	0.0461	0.0794	0.0356	-0.0169	-0.0214
0	-2	0	0	-90	0	-30	0	0.0068	0.0478	0.0833	0.0011	-0.0133	-0.0223
0	-2	0	0	-90	0	0	-30	0.0061	0.003	0.0754	0.0003	-0.0164	-0.0013
0	-2	0	0	-90	0	0	0	0.0068	0.0477	0.0755	0.0008	-0.0165	-0.0223
0	-2	0	0	-90	0	0	30	0.0059	0.0927	0.0756	0.0012	-0.0165	-0.0433
0	-2	0	0	-90	0	30	0	0.0068	0.0477	0.0676	0.0004	-0.0196	-0.0223

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	-2	0	0	-90	90	0	0	0.0032	0.0497	0.0721	-0.034	-0.0162	-0.0232
0	-2	0	0	-60	-90	0	0	0.003	0.044	0.0519	0.0353	0.0146	-0.0203
0	-2	0	0	-60	0	-30	0	0.0061	0.0661	0.0425	0.001	0.0334	-0.0307
0	-2	0	0	-60	0	0	-30	0.0063	0.0108	0.0687	0.0003	-0.0089	-0.0049
0	-2	0	0	-60	0	0	0	0.0066	0.0443	0.0481	0.0007	0.0148	-0.0206
0	-2	0	0	-60	0	0	30	0.0056	0.0779	0.0275	0.001	0.0386	-0.0363
0	-2	0	0	-60	0	30	0	0.0065	0.0226	0.0536	0.0005	-0.0036	-0.0105
0	-2	0	0	-60	90	0	0	0.0029	0.0449	0.0448	-0.0337	0.0149	-0.0209
0	-2	0	0	-30	-90	0	0	0.0025	0.0218	0.0378	0.0343	0.0306	-0.0099
0	-2	0	0	-30	0	-30	0	0.0045	0.0439	0.0018	0.0008	0.0794	-0.0203
0	-2	0	0	-30	0	0	-30	0.0061	0.0107	0.0553	0.0002	0.0059	-0.0048
0	-2	0	0	-30	0	0	0	0.006	0.0218	0.0349	0.0006	0.0295	-0.0101
0	-2	0	0	-30	0	0	30	0.0055	0.0329	0.0145	0.0009	0.0531	-0.0153
0	-2	0	0	-30	0	30	0	0.0059	-0.0005	0.0684	0.0006	-0.0209	0.0003
0	-2	0	0	-30	90	0	0	0.0025	0.0212	0.034	-0.0328	0.0266	-0.0098
0	-2	0	0	0	-90	0	0	0.002	-0.0006	0.0472	0.0331	0.0199	0.0004
0	-2	0	0	0	0	-30	0	0.003	0.0002	-0.0082	0.0006	0.0905	-0.0001
0	-2	0	0	0	0	0	-30	-0.0353	0.0004	-0.1765	-0.0055	0.2619	0.0015
0	-2	0	0	0	0	0	0	0.0055	0	0.0589	0.001	0.0021	-0
0	-2	0	0	0	0	0	30	0.0057	0.0001	0.0653	-0.0003	-0.0047	-0.0001

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\delta_a}$	$\boldsymbol{\delta_e}$	$\boldsymbol{\delta_B}$	$\boldsymbol{p}$	$\boldsymbol{q}$	$\boldsymbol{r}$	$\boldsymbol{C_D}$	$\boldsymbol{C_S}$	$\boldsymbol{C_L}$	$\boldsymbol{C_\ell}$	$\boldsymbol{C_m}$	$\boldsymbol{C_n}$
0	-2	0	0	0	0	30	0	0.0055	0	0.0939	0.0004	-0.05	-0
0	-2	0	0	0	90	0	0	0.002	0.0009	0.0425	-0.0322	0.0163	-0.0005
0	-2	0	0	30	-90	0	0	0.0023	-0.0015	0.0778	0.034	-0.0152	0.0007
0	-2	0	0	30	0	-30	0	0.0046	-0.0199	0.046	0.0008	0.0287	0.0091
0	-2	0	0	30	0	0	-30	0.0055	-0.0092	0.0582	0.0002	0.0028	0.0042
0	-2	0	0	30	0	0	0	0.0059	0.0018	0.0786	0.0007	-0.0207	-0.001
0	-2	0	0	30	0	0	30	0.006	0.0128	0.099	0.0013	-0.0441	-0.0062
0	-2	0	0	30	0	30	0	0.0056	0.0237	0.1114	0.0005	-0.0703	-0.0111
0	-2	0	0	30	90	0	0	0.0024	0.0044	0.0784	-0.033	-0.0245	-0.0022
0	-2	0	0	60	-90	0	0	0.0028	0.0241	0.0948	0.0352	-0.0346	-0.0113
0	-2	0	0	60	0	-30	0	0.006	0.0054	0.0866	0.001	-0.0174	-0.0027
0	-2	0	0	60	0	0	-30	0.0057	-0.0064	0.0715	0.0003	-0.012	0.0029
0	-2	0	0	60	0	0	0	0.0064	0.027	0.0921	0.0008	-0.0358	-0.0127
0	-2	0	0	60	0	0	30	0.0058	0.0605	0.1128	0.0014	-0.0597	-0.0284
0	-2	0	0	60	0	30	0	0.0061	0.0486	0.0976	0.0004	-0.0543	-0.0228
0	-2	0	0	60	90	0	0	0.0027	0.0301	0.0899	-0.0338	-0.0371	-0.0142
0	-2	0	0	90	-90	0	0	0.0029	0.0461	0.0794	0.0356	-0.017	-0.0214
0	-2	0	0	90	0	-30	0	0.0065	0.0478	0.0834	0.0011	-0.0134	-0.0223
0	-2	0	0	90	0	0	-30	0.006	0.003	0.0754	0.0003	-0.0164	-0.0013
0	-2	0	0	90	0	0	0	0.0065	0.0477	0.0755	0.0008	-0.0165	-0.0223

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	-2	0	0	90	0	0	30	0.0053	0.0927	0.0756	0.0012	-0.0166	-0.0433
0	-2	0	0	90	0	30	0	0.0065	0.0477	0.0676	0.0004	-0.0196	-0.0222
0	-2	0	0	90	90	0	0	0.0029	0.0497	0.0721	-0.034	-0.0161	-0.0232
0	0	-10	0	-90	0	0	0	0.0182	0.1275	0.0754	0.0002	-0.0163	-0.0612
0	0	-10	0	-60	0	0	0	0.0178	0.1186	0.0023	0.0002	0.0702	-0.0567
0	0	-10	0	-30	0	0	0	0.0174	0.0729	-0.0598	0.0002	0.1428	-0.0347
0	0	-10	0	0	0	0	0	0.0134	0	-0.1344	0	0.2264	-0
0	0	-10	0	30	0	0	0	0.0174	-0.0729	-0.0598	-0.0002	0.1428	0.0347
0	0	-10	0	60	0	0	0	0.0178	-0.1186	0.0023	-0.0002	0.0702	0.0567
0	0	-10	0	90	0	0	0	0.0182	-0.1275	0.0754	-0.0002	-0.0163	0.0612
0	0	0	-20	-90	0	0	0	0.047	0.0031	0.0753	0.0408	-0.0168	-0.001
0	0	0	-20	-60	0	0	0	0.0466	0.0116	0.0697	0.0398	-0.0105	-0.0049
0	0	0	-20	-30	0	0	0	0.0448	0.0126	0.0563	0.036	0.0046	-0.0052
0	0	0	-20	0	0	0	0	-5.9187	0.013	-0.0142	0.2944	0.084	-0.0125
0	0	0	-20	30	0	0	0	0.0448	-0.0076	0.0565	0.036	0.0041	0.0039
0	0	0	-20	60	0	0	0	0.0466	-0.0058	0.07	0.0398	-0.0108	0.003
0	0	0	-20	90	0	0	0	0.047	0.003	0.0753	0.0408	-0.0165	-0.001
0	0	0	0	-90	-90	0	0	0.0023	-0.0018	0.0758	0.0347	-0.0164	0.0009
0	0	0	0	-90	0	-30	0	0.0059	0	0.0834	-0	-0.0132	-0
0	0	0	0	-90	0	0	-30	0.005	-0.0448	0.0755	-0.0005	-0.0164	0.021

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	0	0	0	-90	0	0	0	0.006	0	0.0755	-0	-0.0164	-0
0	0	0	0	-90	0	0	30	0.0053	0.0448	0.0755	0.0005	-0.0164	-0.021
0	0	0	0	-90	0	30	0	0.0059	0	0.0677	-0	-0.0196	-0
0	0	0	0	-90	90	0	0	0.0023	0.0018	0.0758	-0.0347	-0.0165	-0.0009
0	0	0	0	-60	-90	0	0	0.0023	0.007	0.0709	0.0345	-0.0112	-0.003
0	0	0	0	-60	0	-30	0	0.0055	0.0304	0.0645	0.0001	0.0081	-0.014
0	0	0	0	-60	0	0	-30	0.0055	-0.0247	0.0907	-0.0005	-0.0342	0.0117
0	0	0	0	-60	0	0	0	0.0059	0.0087	0.0701	0	-0.0104	-0.0039
0	0	0	0	-60	0	0	30	0.0051	0.0421	0.0495	0.0004	0.0134	-0.0196
0	0	0	0	-60	0	30	0	0.0058	-0.013	0.0756	0.0001	-0.0289	0.0061
0	0	0	0	-60	90	0	0	0.0023	0.0105	0.0698	-0.0344	-0.0097	-0.0048
0	0	0	0	-30	-90	0	0	0.0022	0.0087	0.0581	0.0337	0.0031	-0.0038
0	0	0	0	-30	0	-30	0	0.0044	0.032	0.0236	0	0.0544	-0.0147
0	0	0	0	-30	0	0	-30	0.0059	-0.001	0.0771	-0.0005	-0.0189	0.0006
0	0	0	0	-30	0	0	0	0.0058	0.0101	0.0566	0	0.0046	-0.0046
0	0	0	0	-30	0	0	30	0.0053	0.0211	0.0362	0.0004	0.0281	-0.0098
0	0	0	0	-30	0	30	0	0.0056	-0.0119	0.0896	0.0001	-0.0452	0.0056
0	0	0	0	-30	90	0	0	0.0022	0.0115	0.0557	-0.0333	0.006	-0.0053
0	0	0	0	0	-90	0	0	0.002	-0.0007	0.0446	0.0327	0.0184	0.0005
0	0	0	0	0	0	-30	0	0.0031	-0	-0.0059	-0	0.0878	0

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	0	0	0	0	0	0	-30	0.0053	-0.0003	0.0753	-0.003	-0.0139	0.0003
0	0	0	0	0	0	0	0	0.004	0	0.0138	0	0.0521	-0
0	0	0	0	0	0	0	30	0.0053	0.0003	0.0753	0.003	-0.0139	-0.0003
0	0	0	0	0	0	30	0	0.0056	-0	0.0916	-0	-0.0474	0
0	0	0	0	0	90	0	0	0.002	0.0007	0.0446	-0.0327	0.0184	-0.0005
0	0	0	0	30	-90	0	0	0.0022	-0.0115	0.0557	0.0333	0.006	0.0053
0	0	0	0	30	0	-30	0	0.0044	-0.032	0.0236	-0	0.0544	0.0147
0	0	0	0	30	0	0	-30	0.0053	-0.0211	0.0362	-0.0004	0.0281	0.0098
0	0	0	0	30	0	0	0	0.0058	-0.0101	0.0566	-0	0.0046	0.0046
0	0	0	0	30	0	0	30	0.0059	0.001	0.0771	0.0005	-0.0189	-0.0006
0	0	0	0	30	0	30	0	0.0056	0.0119	0.0896	-0.0001	-0.0452	-0.0056
0	0	0	0	30	90	0	0	0.0022	-0.0087	0.0581	-0.0337	0.0031	0.0038
0	0	0	0	60	-90	0	0	0.0023	-0.0105	0.0698	0.0344	-0.0097	0.0048
0	0	0	0	60	0	-30	0	0.0055	-0.0304	0.0645	-0.0001	0.0081	0.014
0	0	0	0	60	0	0	-30	0.0051	-0.0421	0.0495	-0.0004	0.0134	0.0196
0	0	0	0	60	0	0	0	0.0059	-0.0087	0.0701	-0	-0.0104	0.0039
0	0	0	0	60	0	0	30	0.0055	0.0247	0.0907	0.0005	-0.0342	-0.0117
0	0	0	0	60	0	30	0	0.0058	0.013	0.0756	-0.0001	-0.0289	-0.0061
0	0	0	0	60	90	0	0	0.0023	-0.007	0.0709	-0.0345	-0.0112	0.003
0	0	0	0	90	-90	0	0	0.0023	-0.0018	0.0758	0.0347	-0.0165	0.0009

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\delta_a}$	$\boldsymbol{\delta_e}$	$\boldsymbol{\delta_B}$	$\boldsymbol{p}$	$\boldsymbol{q}$	$\boldsymbol{r}$	$\boldsymbol{C_D}$	$\boldsymbol{C_S}$	$\boldsymbol{C_L}$	$\boldsymbol{C_\ell}$	$\boldsymbol{C_m}$	$\boldsymbol{C_n}$
0	0	0	0	90	0	-30	0	0.0059	-0	0.0834	-0	-0.0132	0
0	0	0	0	90	0	0	-30	0.0053	-0.0448	0.0755	-0.0005	-0.0164	0.021
0	0	0	0	90	0	0	0	0.006	0	0.0755	-0	-0.0164	-0
0	0	0	0	90	0	0	30	0.005	0.0448	0.0755	0.0005	-0.0164	-0.021
0	0	0	0	90	0	30	0	0.0059	0	0.0677	-0	-0.0196	-0
0	0	0	0	90	90	0	0	0.0023	0.0018	0.0758	-0.0347	-0.0164	-0.0009
0	0	0	20	-90	0	0	0	0.047	-0.003	0.0753	-0.0408	-0.0165	0.001
0	0	0	20	-60	0	0	0	0.0466	0.0058	0.07	-0.0398	-0.0108	-0.003
0	0	0	20	-30	0	0	0	0.0448	0.0076	0.0565	-0.036	0.0041	-0.0039
0	0	0	20	0	0	0	0	-5.9187	-0.013	-0.0142	-0.2944	0.0841	0.0125
0	0	0	20	30	0	0	0	0.0448	-0.0126	0.0563	-0.036	0.0046	0.0052
0	0	0	20	60	0	0	0	0.0466	-0.0116	0.0697	-0.0398	-0.0105	0.0049
0	0	0	20	90	0	0	0	0.047	-0.0031	0.0753	-0.0408	-0.0168	0.001
0	0	10	0	-90	0	0	0	0.0187	-0.1275	0.0754	-0.0002	-0.0163	0.0612
0	0	10	0	-60	0	0	0	0.019	-0.1012	0.1377	-0.0002	-0.0908	0.0489
0	0	10	0	-30	0	0	0	0.0192	-0.0528	0.1729	-0.0002	-0.1334	0.0256
0	0	10	0	0	0	0	0	0.0185	0	0.1617	0	-0.1218	-0
0	0	10	0	30	0	0	0	0.0192	0.0528	0.1729	0.0002	-0.1334	-0.0256
0	0	10	0	60	0	0	0	0.019	0.1012	0.1377	0.0002	-0.0908	-0.0489
0	0	10	0	90	0	0	0	0.0187	0.1275	0.0754	0.0002	-0.0163	-0.0612

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\delta_a}$	$\boldsymbol{\delta_e}$	$\boldsymbol{\delta_B}$	$\boldsymbol{p}$	$\boldsymbol{q}$	$\boldsymbol{r}$	$\boldsymbol{C_D}$	$\boldsymbol{C_S}$	$\boldsymbol{C_L}$	$\boldsymbol{C_\ell}$	$\boldsymbol{C_m}$	$\boldsymbol{C_n}$
0	2	0	-20	-90	0	0	0	0.0471	-0.0445	0.062	0.0402	-0.0179	0.0214
0	2	0	-20	-60	0	0	0	0.0465	-0.0265	0.0802	0.0391	-0.0385	0.0128
0	2	0	-20	-30	0	0	0	0.0445	-0.0026	0.0723	0.0355	-0.0287	0.0016
0	2	0	-20	0	0	0	0	-4.8777	-0.0376	-3.0571	0.1603	3.4425	-0.3247
0	2	0	-20	30	0	0	0	0.0447	-0.0153	0.0295	0.0356	0.0204	0.0081
0	2	0	-20	60	0	0	0	0.0468	-0.0385	0.0366	0.0392	0.0118	0.0188
0	2	0	-20	90	0	0	0	0.0473	-0.0445	0.062	0.0402	-0.0175	0.0214
0	2	0	0	-90	-90	0	0	0.0029	-0.0497	0.0721	0.034	-0.0161	0.0232
0	2	0	0	-90	0	-30	0	0.0065	-0.0478	0.0834	-0.0011	-0.0134	0.0223
0	2	0	0	-90	0	0	-30	0.0053	-0.0927	0.0756	-0.0012	-0.0166	0.0433
0	2	0	0	-90	0	0	0	0.0065	-0.0477	0.0755	-0.0008	-0.0165	0.0223
0	2	0	0	-90	0	0	30	0.006	-0.003	0.0754	-0.0003	-0.0164	0.0013
0	2	0	0	-90	0	30	0	0.0065	-0.0477	0.0676	-0.0004	-0.0196	0.0222
0	2	0	0	-90	90	0	0	0.0029	-0.0461	0.0794	-0.0356	-0.017	0.0214
0	2	0	0	-60	-90	0	0	0.0027	-0.0301	0.0899	0.0338	-0.0371	0.0142
0	2	0	0	-60	0	-30	0	0.006	-0.0054	0.0866	-0.001	-0.0174	0.0027
0	2	0	0	-60	0	0	-30	0.0058	-0.0605	0.1128	-0.0014	-0.0597	0.0284
0	2	0	0	-60	0	0	0	0.0064	-0.027	0.0921	-0.0008	-0.0358	0.0127
0	2	0	0	-60	0	0	30	0.0057	0.0064	0.0715	-0.0003	-0.012	-0.0029
0	2	0	0	-60	0	30	0	0.0061	-0.0486	0.0976	-0.0004	-0.0543	0.0228

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	2	0	0	-60	90	0	0	0.0028	-0.0241	0.0948	-0.0352	-0.0346	0.0113
0	2	0	0	-30	-90	0	0	0.0024	-0.0044	0.0784	0.033	-0.0245	0.0022
0	2	0	0	-30	0	-30	0	0.0046	0.0199	0.046	-0.0008	0.0287	-0.0091
0	2	0	0	-30	0	0	-30	0.006	-0.0128	0.099	-0.0013	-0.0441	0.0062
0	2	0	0	-30	0	0	0	0.0059	-0.0018	0.0786	-0.0007	-0.0207	0.001
0	2	0	0	-30	0	0	30	0.0055	0.0092	0.0582	-0.0002	0.0028	-0.0042
0	2	0	0	-30	0	30	0	0.0056	-0.0237	0.1114	-0.0005	-0.0703	0.0111
0	2	0	0	-30	90	0	0	0.0023	0.0015	0.0778	-0.034	-0.0152	-0.0007
0	2	0	0	0	-90	0	0	0.002	-0.0009	0.0425	0.0322	0.0163	0.0005
0	2	0	0	0	0	-30	0	0.003	-0.0002	-0.0082	-0.0006	0.0905	0.0001
0	2	0	0	0	0	0	-30	0.0057	-0.0001	0.0653	0.0003	-0.0047	0.0001
0	2	0	0	0	0	0	0	0.0055	-0	0.0589	-0.001	0.0021	0
0	2	0	0	0	0	0	30	-0.0353	-0.0004	-0.1765	0.0055	0.2619	-0.0015
0	2	0	0	0	0	30	0	0.0055	-0	0.0939	-0.0004	-0.05	0
0	2	0	0	0	90	0	0	0.002	0.0006	0.0472	-0.0331	0.0199	-0.0004
0	2	0	0	30	-90	0	0	0.0025	-0.0212	0.034	0.0328	0.0266	0.0098
0	2	0	0	30	0	-30	0	0.0045	-0.0439	0.0018	-0.0008	0.0794	0.0203
0	2	0	0	30	0	0	-30	0.0055	-0.0329	0.0145	-0.0009	0.0531	0.0153
0	2	0	0	30	0	0	0	0.006	-0.0218	0.0349	-0.0006	0.0295	0.0101
0	2	0	0	30	0	0	30	0.0061	-0.0107	0.0553	-0.0002	0.0059	0.0048

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	2	0	0	30	0	30	0	0.0059	0.0005	0.0684	-0.0006	-0.0209	-0.0003
0	2	0	0	30	90	0	0	0.0025	-0.0218	0.0378	-0.0343	0.0306	0.0099
0	2	0	0	60	-90	0	0	0.0029	-0.0449	0.0448	0.0337	0.0149	0.0209
0	2	0	0	60	0	-30	0	0.0061	-0.0661	0.0425	-0.001	0.0334	0.0307
0	2	0	0	60	0	0	-30	0.0056	-0.0779	0.0275	-0.001	0.0386	0.0363
0	2	0	0	60	0	0	0	0.0066	-0.0443	0.0481	-0.0007	0.0148	0.0206
0	2	0	0	60	0	0	30	0.0063	-0.0108	0.0687	-0.0003	-0.0089	0.0049
0	2	0	0	60	0	30	0	0.0065	-0.0226	0.0536	-0.0005	-0.0036	0.0105
0	2	0	0	60	90	0	0	0.003	-0.044	0.0519	-0.0353	0.0146	0.0203
0	2	0	0	90	-90	0	0	0.0032	-0.0497	0.0721	0.034	-0.0162	0.0232
0	2	0	0	90	0	-30	0	0.0068	-0.0478	0.0833	-0.0011	-0.0133	0.0223
0	2	0	0	90	0	0	-30	0.0059	-0.0927	0.0756	-0.0012	-0.0165	0.0433
0	2	0	0	90	0	0	0	0.0068	-0.0477	0.0755	-0.0008	-0.0165	0.0223
0	2	0	0	90	0	0	30	0.0061	-0.003	0.0754	-0.0003	-0.0164	0.0013
0	2	0	0	90	0	30	0	0.0068	-0.0477	0.0676	-0.0004	-0.0196	0.0223
0	2	0	0	90	90	0	0	0.0032	-0.0461	0.0794	-0.0356	-0.0169	0.0214
0	4	0	-20	-90	0	0	0	0.0486	-0.0919	0.0487	0.0397	-0.0189	0.0437
0	4	0	-20	-60	0	0	0	0.0474	-0.0646	0.0906	0.0386	-0.0664	0.0304
0	4	0	-20	-30	0	0	0	0.0445	-0.018	0.0886	0.0351	-0.0621	0.0085
0	4	0	-20	0	0	0	0	-0.0988	0.0027	0.4961	0.0244	-0.5042	-0.0083

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	4	0	-20	30	0	0	0	0.0451	-0.0227	0.0029	0.0355	0.0362	0.0121
0	4	0	-20	60	0	0	0	0.0482	-0.0711	0.0034	0.039	0.0342	0.0345
0	4	0	-20	90	0	0	0	0.0491	-0.0919	0.0487	0.0397	-0.0185	0.0437
0	4	0	0	-90	-90	0	0	0.0048	-0.0976	0.0683	0.0334	-0.0159	0.0454
0	4	0	0	-90	0	-30	0	0.0084	-0.0955	0.0833	-0.0022	-0.0137	0.0445
0	4	0	0	-90	0	0	-30	0.0071	-0.1407	0.0756	-0.002	-0.0169	0.0656
0	4	0	0	-90	0	0	0	0.0085	-0.0954	0.0754	-0.0015	-0.0168	0.0445
0	4	0	0	-90	0	0	30	0.0082	-0.0505	0.0753	-0.0011	-0.0166	0.0235
0	4	0	0	-90	0	30	0	0.0085	-0.0953	0.0675	-0.0009	-0.0198	0.0445
0	4	0	0	-90	90	0	0	0.0049	-0.094	0.083	-0.0364	-0.0177	0.0436
0	4	0	0	-60	-90	0	0	0.0042	-0.067	0.1088	0.033	-0.063	0.0313
0	4	0	0	-60	0	-30	0	0.0075	-0.0412	0.1086	-0.0021	-0.043	0.0194
0	4	0	0	-60	0	0	-30	0.0071	-0.0963	0.1348	-0.0024	-0.0853	0.045
0	4	0	0	-60	0	0	0	0.0078	-0.0626	0.114	-0.0017	-0.0613	0.0293
0	4	0	0	-60	0	0	30	0.0073	-0.0292	0.0933	-0.001	-0.0374	0.0137
0	4	0	0	-60	0	30	0	0.0075	-0.0841	0.1193	-0.0011	-0.0795	0.0393
0	4	0	0	-60	90	0	0	0.0042	-0.0586	0.1196	-0.0362	-0.0595	0.0273
0	4	0	0	-30	-90	0	0	0.0029	-0.0184	0.1001	0.0324	-0.0537	0.0087
0	4	0	0	-30	0	-30	0	0.004	0.0168	0.0514	-0.0016	0.0215	-0.0075
0	4	0	0	-30	0	0	-30	0.0064	-0.0247	0.1209	-0.0022	-0.0695	0.0117

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	4	0	0	-30	0	0	0	0.0064	-0.0137	0.1005	-0.0014	-0.0461	0.0065
0	4	0	0	-30	0	0	30	0.006	-0.0025	0.0798	-0.0008	-0.0222	0.0013
0	4	0	0	-30	0	30	0	0.006	-0.0355	0.1331	-0.0011	-0.0954	0.0166
0	4	0	0	-30	90	0	0	0.0028	-0.0105	0.1036	-0.0349	-0.0407	0.0048
0	4	0	0	0	-90	0	0	0.0023	-0.001	0.0514	0.032	0.002	0.0005
0	4	0	0	0	0	-30	0	0.0031	-0.0002	-0.0075	-0.0007	0.0895	0.0001
0	4	0	0	0	0	0	-30	-0.0055	-0.0019	0.1604	0.0077	-0.1098	0.0016
0	4	0	0	0	0	0	0	-0.1808	-0.0142	0.6111	-0.0026	-0.6041	-0.0005
0	4	0	0	0	0	0	30	0.0037	-0.0004	0.0023	-0.0028	0.0656	0.0003
0	4	0	0	0	0	30	0	0.0055	-0.0001	0.0944	-0.0007	-0.0508	0.0001
0	4	0	0	0	90	0	0	0.0017	0.0005	0.0388	-0.0336	0.0333	-0.0004
0	4	0	0	30	-90	0	0	0.0031	-0.0314	0.0113	0.0324	0.0483	0.0146
0	4	0	0	30	0	-30	0	0.005	-0.0556	-0.0196	-0.0014	0.1038	0.0258
0	4	0	0	30	0	0	-30	0.006	-0.0446	-0.007	-0.0012	0.0776	0.0208
0	4	0	0	30	0	0	0	0.0066	-0.0333	0.0134	-0.0011	0.054	0.0155
0	4	0	0	30	0	0	30	0.0068	-0.0223	0.0335	-0.0008	0.0308	0.0103
0	4	0	0	30	0	30	0	0.0067	-0.016	0.0377	-0.0009	0.0138	0.0074
0	4	0	0	30	90	0	0	0.003	-0.0355	0.0165	-0.0349	0.059	0.0162
0	4	0	0	60	-90	0	0	0.0046	-0.0791	0.0199	0.0332	0.0393	0.0368
0	4	0	0	60	0	-30	0	0.0076	-0.1017	0.0206	-0.0019	0.0582	0.0473

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	4	0	0	60	0	0	-30	0.0072	-0.1136	0.0057	-0.0015	0.0636	0.053
0	4	0	0	60	0	0	0	0.0083	-0.0797	0.0262	-0.0013	0.0398	0.0371
0	4	0	0	60	0	0	30	0.0081	-0.0461	0.0467	-0.001	0.0161	0.0214
0	4	0	0	60	0	30	0	0.0083	-0.058	0.0317	-0.0008	0.0214	0.027
0	4	0	0	60	90	0	0	0.0047	-0.0807	0.033	-0.036	0.04	0.0374
0	4	0	0	90	-90	0	0	0.0054	-0.0976	0.0683	0.0334	-0.0159	0.0455
0	4	0	0	90	0	-30	0	0.009	-0.0955	0.0833	-0.0022	-0.0137	0.0446
0	4	0	0	90	0	0	-30	0.0079	-0.1407	0.0756	-0.002	-0.0169	0.0656
0	4	0	0	90	0	0	0	0.009	-0.0954	0.0754	-0.0015	-0.0167	0.0445
0	4	0	0	90	0	0	30	0.0085	-0.0504	0.0752	-0.0011	-0.0166	0.0235
0	4	0	0	90	0	30	0	0.009	-0.0953	0.0675	-0.0009	-0.0198	0.0445
0	4	0	0	90	90	0	0	0.0055	-0.094	0.083	-0.0364	-0.0176	0.0436
0	6	0	-20	-90	0	0	0	0.0518	-0.1393	0.0354	0.0397	-0.0197	0.0659
0	6	0	-20	-60	0	0	0	0.0495	-0.1028	0.1009	0.0384	-0.094	0.0481
0	6	0	-20	-30	0	0	0	0.045	-0.0331	0.1039	0.035	-0.0945	0.0153
0	6	0	-20	0	0	0	0	0.0315	-0.0012	-0.0648	0.0268	0.0928	0.0015
0	6	0	-20	30	0	0	0	0.0462	-0.0305	-0.0244	0.0358	0.0529	0.0163
0	6	0	-20	60	0	0	0	0.051	-0.1034	-0.0297	0.0392	0.0566	0.05
0	6	0	-20	90	0	0	0	0.0526	-0.1393	0.0354	0.0397	-0.0193	0.066
0	6	0	0	-90	-90	0	0	0.0083	-0.1451	0.0644	0.0331	-0.0156	0.0675

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	6	0	0	-90	0	-30	0	0.0119	-0.143	0.0832	-0.0028	-0.0141	0.0667
0	6	0	0	-90	0	0	-30	0.0103	-0.1887	0.0755	-0.0025	-0.0173	0.0879
0	6	0	0	-90	0	0	0	0.0119	-0.1429	0.0753	-0.0019	-0.017	0.0666
0	6	0	0	-90	0	0	30	0.0119	-0.0977	0.075	-0.0014	-0.0168	0.0456
0	6	0	0	-90	0	30	0	0.0119	-0.1429	0.0674	-0.0011	-0.0199	0.0666
0	6	0	0	-90	90	0	0	0.0084	-0.1415	0.0866	-0.037	-0.0185	0.0657
0	6	0	0	-60	-90	0	0	0.0068	-0.1043	0.1278	0.0325	-0.0891	0.0486
0	6	0	0	-60	0	-30	0	0.0103	-0.0774	0.1308	-0.0028	-0.069	0.0362
0	6	0	0	-60	0	0	-30	0.0095	-0.1325	0.1568	-0.0033	-0.111	0.0618
0	6	0	0	-60	0	0	0	0.0104	-0.0985	0.1359	-0.0023	-0.0869	0.046
0	6	0	0	-60	0	0	30	0.0101	-0.065	0.1152	-0.0015	-0.063	0.0304
0	6	0	0	-60	0	30	0	0.01	-0.1202	0.1413	-0.0016	-0.1052	0.0561
0	6	0	0	-60	90	0	0	0.0068	-0.0936	0.1447	-0.0369	-0.0849	0.0435
0	6	0	0	-30	-90	0	0	0.0038	-0.0332	0.1233	0.0317	-0.0846	0.0154
0	6	0	0	-30	0	-30	0	0.0062	-0.0042	0.0906	-0.0022	-0.0229	0.0022
0	6	0	0	-30	0	0	-30	0.0071	-0.0366	0.143	-0.003	-0.095	0.0173
0	6	0	0	-30	0	0	0	0.0072	-0.0256	0.1226	-0.0021	-0.0716	0.0121
0	6	0	0	-30	0	0	30	0.0069	-0.0146	0.1022	-0.0013	-0.0482	0.0069
0	6	0	0	-30	0	30	0	0.0067	-0.0474	0.1548	-0.0017	-0.1206	0.0222
0	6	0	0	-30	90	0	0	0.0036	-0.0196	0.1241	-0.0355	-0.0604	0.0091

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	6	0	0	0	-90	0	0	0.0024	-0.0011	0.0518	0.0316	-0.0032	0.0005
0	6	0	0	0	0	-30	0	0.0031	-0.0004	-0.0077	-0.001	0.0893	0.0002
0	6	0	0	0	0	0	-30	0.0013	-0.0015	0.0221	0.0028	0.0384	0.0011
0	6	0	0	0	0	0	0	-0.0129	-0.0025	-0.1127	-0.0029	0.1911	0.0007
0	6	0	0	0	0	0	30	0.0005	-0.0006	-0.0319	-0.0017	0.1022	0.0002
0	6	0	0	0	0	30	0	0.0055	-0.0001	0.0949	-0.001	-0.0517	0.0001
0	6	0	0	0	90	0	0	0.0016	0.0003	0.0397	-0.034	0.0363	-0.0004
0	6	0	0	30	-90	0	0	0.0042	-0.0412	-0.0107	0.0324	0.0691	0.0192
0	6	0	0	30	0	-30	0	0.0059	-0.0669	-0.04	-0.0019	0.1269	0.0311
0	6	0	0	30	0	0	-30	0.007	-0.056	-0.0278	-0.0014	0.1013	0.0261
0	6	0	0	30	0	0	0	0.0076	-0.0447	-0.0075	-0.0014	0.0778	0.0208
0	6	0	0	30	0	0	30	0.0078	-0.0334	0.0128	-0.0011	0.0543	0.0156
0	6	0	0	30	0	30	0	0.0076	-0.0228	0.0247	-0.001	0.0289	0.0107
0	6	0	0	30	90	0	0	0.0039	-0.0485	-0.0033	-0.0354	0.0856	0.0223
0	6	0	0	60	-90	0	0	0.0075	-0.1132	-0.0049	0.0332	0.0636	0.0526
0	6	0	0	60	0	-30	0	0.0103	-0.1367	-0.0008	-0.0024	0.0825	0.0637
0	6	0	0	60	0	0	-30	0.0098	-0.149	-0.0159	-0.0017	0.0881	0.0694
0	6	0	0	60	0	0	0	0.011	-0.1147	0.0046	-0.0015	0.0643	0.0535
0	6	0	0	60	0	0	30	0.011	-0.0809	0.025	-0.0012	0.0407	0.0377
0	6	0	0	60	0	30	0	0.0112	-0.0939	0.0094	-0.0009	0.0468	0.0438

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
0	6	0	0	60	90	0	0	0.0075	-0.1172	0.0144	-0.0364	0.0649	0.0544
0	6	0	0	90	-90	0	0	0.0092	-0.1451	0.0644	0.0331	-0.0156	0.0676
0	6	0	0	90	0	-30	0	0.0127	-0.143	0.0831	-0.0028	-0.0141	0.0667
0	6	0	0	90	0	0	-30	0.0114	-0.1887	0.0755	-0.0025	-0.0172	0.0879
0	6	0	0	90	0	0	0	0.0127	-0.1429	0.0752	-0.0019	-0.017	0.0667
0	6	0	0	90	0	0	30	0.0124	-0.0977	0.075	-0.0014	-0.0168	0.0456
0	6	0	0	90	0	30	0	0.0127	-0.1429	0.0674	-0.0011	-0.0199	0.0666
0	6	0	0	90	90	0	0	0.0092	-0.1415	0.0866	-0.037	-0.0184	0.0657
5	0	-10	0	-90	0	0	0	0.0323	0.1292	0.3711	-0.0011	0.0011	-0.062
5	0	-10	0	-60	0	0	0	0.026	0.105	0.3067	-0.0025	0.0768	-0.0498
5	0	-10	0	-30	0	0	0	0.0216	0.0616	0.2575	-0.0014	0.1335	-0.0291
5	0	-10	0	0	0	0	0	0.0195	0	0.2262	-0	0.1702	-0
5	0	-10	0	30	0	0	0	0.0216	-0.0616	0.2575	0.0014	0.1335	0.0291
5	0	-10	0	60	0	0	0	0.026	-0.105	0.3067	0.0025	0.0768	0.0498
5	0	-10	0	90	0	0	0	0.0323	-0.1292	0.3711	0.0011	0.0011	0.062
5	0	0	-20	-90	0	0	0	0.0612	0.0155	0.3708	0.0416	0.0017	-0.0045
5	0	0	-20	-60	0	0	0	0.0577	0.0089	0.3687	0.039	0.0007	-0.001
5	0	0	-20	-30	0	0	0	0.0549	0.0121	0.359	0.0368	0.0092	-0.0034
5	0	0	-20	0	0	0	0	0.056	0.0042	0.3559	0.0336	0.017	-0.0004
5	0	0	-20	30	0	0	0	0.0613	0.0133	0.3875	0.0388	-0.0172	-0.0047

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
5	0	0	-20	60	0	0	0	0.0623	0.0206	0.3804	0.0423	-0.0086	-0.0079
5	0	0	-20	90	0	0	0	0.0611	0.0154	0.3708	0.0416	0.0021	-0.0045
5	0	0	0	-90	-90	0	0	0.0166	-0.009	0.3735	0.035	0.0022	0.0048
5	0	0	0	-90	0	-30	0	0.0202	0	0.3794	-0	0.0055	-0
5	0	0	0	-90	0	0	-30	0.0192	-0.046	0.3717	-0.0016	0.0023	0.0215
5	0	0	0	-90	0	0	0	0.0201	0	0.3716	-0	0.0023	-0
5	0	0	0	-90	0	0	30	0.0194	0.0461	0.3717	0.0016	0.0024	-0.0215
5	0	0	0	-90	0	30	0	0.0201	0	0.3638	-0	-0.0009	-0
5	0	0	0	-90	90	0	0	0.0166	0.009	0.3735	-0.035	0.0022	-0.0048
5	0	0	0	-60	-90	0	0	0.0157	-0.0167	0.3777	0.0333	-0.0043	0.0092
5	0	0	0	-60	0	-30	0	0.018	0.0163	0.3692	-0.002	0.0157	-0.0068
5	0	0	0	-60	0	0	-30	0.0201	-0.0414	0.3974	-0.0035	-0.029	0.0201
5	0	0	0	-60	0	0	0	0.0194	-0.006	0.3753	-0.0016	-0.0035	0.0036
5	0	0	0	-60	0	0	30	0.0173	0.0283	0.3541	0	0.0213	-0.0125
5	0	0	0	-60	0	30	0	0.02	-0.0269	0.3805	-0.0011	-0.0217	0.0133
5	0	0	0	-60	90	0	0	0.016	0.004	0.3769	-0.0364	-0.0034	-0.0017
5	0	0	0	-30	-90	0	0	0.0149	-0.0114	0.3773	0.0328	-0.006	0.0065
5	0	0	0	-30	0	-30	0	0.0152	0.0204	0.3417	-0.0011	0.0456	-0.0089
5	0	0	0	-30	0	0	-30	0.02	-0.0123	0.3947	-0.0032	-0.0277	0.0063
5	0	0	0	-30	0	0	0	0.0189	-0.0013	0.3742	-0.0011	-0.0039	0.0012

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
5	0	0	0	-30	0	0	30	0.0174	0.0098	0.3537	0.001	0.0199	-0.0041
5	0	0	0	-30	0	30	0	0.021	-0.023	0.4069	-0.0008	-0.0536	0.0112
5	0	0	0	-30	90	0	0	0.0159	0.0085	0.3755	-0.0347	-0.0028	-0.004
5	0	0	0	0	-90	0	0	0.0147	-0.0094	0.3587	0.0344	0.0164	0.0049
5	0	0	0	0	0	-30	0	0.0126	0	0.3112	-0	0.0812	-0
5	0	0	0	0	0	0	-30	0.0183	-0.0001	0.3581	-0.0018	0.0152	0.0001
5	0	0	0	0	0	0	0	0.0183	0	0.3581	-0	0.0152	-0
5	0	0	0	0	0	0	30	0.0183	0.0001	0.3581	0.0018	0.0152	-0.0001
5	0	0	0	0	0	30	0	0.0218	0	0.4057	-0	-0.0514	-0
5	0	0	0	0	90	0	0	0.0147	0.0094	0.3587	-0.0344	0.0164	-0.0049
5	0	0	0	30	-90	0	0	0.0159	-0.0085	0.3755	0.0347	-0.0028	0.004
5	0	0	0	30	0	-30	0	0.0152	-0.0204	0.3417	0.0011	0.0456	0.0089
5	0	0	0	30	0	0	-30	0.0174	-0.0098	0.3537	-0.001	0.0199	0.0041
5	0	0	0	30	0	0	0	0.0189	0.0013	0.3742	0.0011	-0.0039	-0.0012
5	0	0	0	30	0	0	30	0.02	0.0123	0.3947	0.0032	-0.0277	-0.0063
5	0	0	0	30	0	30	0	0.021	0.023	0.4069	0.0008	-0.0536	-0.0112
5	0	0	0	30	90	0	0	0.0149	0.0114	0.3773	-0.0328	-0.006	-0.0065
5	0	0	0	60	-90	0	0	0.016	-0.004	0.3769	0.0364	-0.0034	0.0017
5	0	0	0	60	0	-30	0	0.018	-0.0163	0.3692	0.002	0.0157	0.0068
5	0	0	0	60	0	0	-30	0.0173	-0.0283	0.3541	-0	0.0213	0.0125

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
5	0	0	0	60	0	0	0	0.0194	0.006	0.3753	0.0016	-0.0035	-0.0036
5	0	0	0	60	0	0	30	0.0201	0.0414	0.3974	0.0035	-0.029	-0.0201
5	0	0	0	60	0	30	0	0.02	0.0269	0.3805	0.0011	-0.0217	-0.0133
5	0	0	0	60	90	0	0	0.0157	0.0167	0.3777	-0.0333	-0.0043	-0.0092
5	0	0	0	90	-90	0	0	0.0166	-0.009	0.3735	0.035	0.0022	0.0048
5	0	0	0	90	0	-30	0	0.0202	0	0.3794	-0	0.0055	-0
5	0	0	0	90	0	0	-30	0.0194	-0.0461	0.3717	-0.0016	0.0024	0.0215
5	0	0	0	90	0	0	0	0.0201	0	0.3716	-0	0.0023	-0
5	0	0	0	90	0	0	30	0.0192	0.046	0.3717	0.0016	0.0023	-0.0215
5	0	0	0	90	0	30	0	0.0201	0	0.3638	-0	-0.0009	-0
5	0	0	0	90	90	0	0	0.0166	0.009	0.3735	-0.035	0.0022	-0.0048
5	0	0	20	-90	0	0	0	0.0611	-0.0154	0.3708	-0.0416	0.0021	0.0045
5	0	0	20	-60	0	0	0	0.0623	-0.0206	0.3804	-0.0423	-0.0086	0.0079
5	0	0	20	-30	0	0	0	0.0613	-0.0133	0.3875	-0.0388	-0.0172	0.0047
5	0	0	20	0	0	0	0	0.056	-0.0042	0.3559	-0.0336	0.017	0.0004
5	0	0	20	30	0	0	0	0.0549	-0.0121	0.359	-0.0368	0.0092	0.0034
5	0	0	20	60	0	0	0	0.0577	-0.0089	0.3687	-0.039	0.0007	0.001
5	0	0	20	90	0	0	0	0.0612	-0.0155	0.3708	-0.0416	0.0017	0.0045
5	0	10	0	-90	0	0	0	0.0328	-0.1291	0.3711	0.001	0.0011	0.0619
5	0	10	0	-60	0	0	0	0.0376	-0.1171	0.4428	-0.0008	-0.086	0.0568

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
5	0	10	0	-30	0	0	0	0.0413	-0.0641	0.4896	-0.0008	-0.1428	0.0313
5	0	10	0	0	0	0	0	0.0423	0	0.4885	-0	-0.1408	-0
5	0	10	0	30	0	0	0	0.0413	0.0641	0.4896	0.0008	-0.1428	-0.0313
5	0	10	0	60	0	0	0	0.0376	0.1171	0.4428	0.0008	-0.086	-0.0568
5	0	10	0	90	0	0	0	0.0328	0.1291	0.3711	-0.001	0.0011	-0.0619
10	-6	0	20	-90	0	0	0	0.0862	0.0908	0.6279	-0.0303	0.0045	-0.0509
10	-6	0	20	-60	0	0	0	0.0852	0.0548	0.5831	-0.029	0.0627	-0.0324
10	-6	0	20	-30	0	0	0	0.0871	-0.008	0.6184	-0.03	0.0221	-0.002
10	-6	0	20	0	0	0	0	0.0951	-0.0099	0.6824	-0.0294	-0.053	-0.0005
10	-6	0	20	30	0	0	0	0.104	0.0491	0.7326	-0.0326	-0.1163	-0.0285
10	-6	0	20	60	0	0	0	0.1001	0.107	0.696	-0.0334	-0.0762	-0.0569
10	-6	0	20	90	0	0	0	0.0856	0.0907	0.6279	-0.0303	0.0042	-0.0508
10	-4	0	20	-90	0	0	0	0.0834	0.0591	0.6429	-0.031	0.008	-0.0352
10	-4	0	20	-60	0	0	0	0.0901	0.0134	0.6215	-0.0365	0.0376	-0.0113
10	-4	0	20	-30	0	0	0	0.0913	-0.0153	0.6492	-0.0333	0.0058	0.0023
10	-4	0	20	0	0	0	0	0.096	-0.0085	0.6862	-0.0324	-0.0385	-0.0004
10	-4	0	20	30	0	0	0	0.1008	0.0339	0.7177	-0.0357	-0.0805	-0.0208
10	-4	0	20	60	0	0	0	0.0946	0.069	0.6886	-0.0353	-0.0486	-0.0386
10	-4	0	20	90	0	0	0	0.083	0.059	0.643	-0.031	0.0074	-0.0351
10	-2	0	20	-90	0	0	0	0.09	0.0222	0.6533	-0.0364	0.0162	-0.0168

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
10	-2	0	20	-60	0	0	0	0.0937	-0.0287	0.6627	-0.0446	0.0065	0.0097
10	-2	0	20	-30	0	0	0	0.0954	-0.0215	0.6777	-0.0366	-0.0097	0.0063
10	-2	0	20	0	0	0	0	0.0967	-0.0069	0.6883	-0.0357	-0.024	-0.0002
10	-2	0	20	30	0	0	0	0.0977	0.0191	0.7026	-0.0389	-0.0463	-0.0133
10	-2	0	20	60	0	0	0	0.0894	0.0286	0.6784	-0.0377	-0.0193	-0.0195
10	-2	0	20	90	0	0	0	0.0899	0.0221	0.6534	-0.0364	0.0157	-0.0167
10	0	-10	0	-90	0	0	0	0.0662	0.1357	0.6648	-0.0015	0.0192	-0.065
10	0	-10	0	-60	0	0	0	0.051	0.0926	0.6133	-0.0058	0.0779	-0.0431
10	0	-10	0	-30	0	0	0	0.0509	0.0464	0.5804	0.0002	0.1181	-0.0218
10	0	-10	0	0	0	0	0	0.0489	0	0.561	-0	0.1417	-0
10	0	-10	0	30	0	0	0	0.0509	-0.0464	0.5804	-0.0002	0.1181	0.0218
10	0	-10	0	60	0	0	0	0.051	-0.0926	0.6133	0.0058	0.0779	0.0431
10	0	-10	0	90	0	0	0	0.0662	-0.1357	0.6648	0.0015	0.0192	0.065
10	0	0	-20	-90	0	0	0	0.095	0.0261	0.6645	0.0435	0.0206	-0.0073
10	0	0	-20	-60	0	0	0	0.0838	0.0064	0.672	0.0394	0.0037	0.0027
10	0	0	-20	-30	0	0	0	0.0946	-0.0032	0.6838	0.0426	-0.0094	0.0053
10	0	0	-20	0	0	0	0	0.0973	0.0053	0.6893	0.0391	-0.0101	0.0001
10	0	0	-20	30	0	0	0	0.0999	0.0275	0.705	0.0399	-0.0257	-0.0103
10	0	0	-20	60	0	0	0	0.0974	0.0386	0.6859	0.0462	-0.0044	-0.0155
10	0	0	-20	90	0	0	0	0.0949	0.0261	0.6645	0.0435	0.0209	-0.0073

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
10	0	0	0	-90	-90	0	0	0.0506	-0.0165	0.6696	0.0357	0.0208	0.0089
10	0	0	0	-90	0	-30	0	0.0541	0	0.6739	-0	0.0243	-0
10	0	0	0	-90	0	0	-30	0.053	-0.0484	0.6664	-0.0029	0.0211	0.0225
10	0	0	0	-90	0	0	0	0.054	0	0.6662	-0	0.021	-0
10	0	0	0	-90	0	0	30	0.0533	0.0484	0.6663	0.0029	0.0212	-0.0225
10	0	0	0	-90	0	30	0	0.0539	0	0.6585	-0	0.0177	-0
10	0	0	0	-90	90	0	0	0.0506	0.0165	0.6696	-0.0357	0.0208	-0.0089
10	0	0	0	-60	-90	0	0	0.0447	-0.0314	0.6816	0.0322	0.0022	0.0176
10	0	0	0	-60	0	-30	0	0.0483	0.0035	0.6752	-0.0039	0.0185	0.0002
10	0	0	0	-60	0	0	-30	0.0505	-0.0513	0.7035	-0.0074	-0.0273	0.0257
10	0	0	0	-60	0	0	0	0.0499	-0.0164	0.68	-0.0034	0.0006	0.0094
10	0	0	0	-60	0	0	30	0.0478	0.0179	0.6577	0.0003	0.0273	-0.0066
10	0	0	0	-60	0	30	0	0.051	-0.0364	0.6849	-0.0028	-0.0172	0.0186
10	0	0	0	-60	90	0	0	0.0484	-0.0014	0.6851	-0.0388	-0.0012	0.0011
10	0	0	0	-30	-90	0	0	0.0527	-0.0356	0.7018	0.0363	-0.0214	0.0187
10	0	0	0	-30	0	-30	0	0.0508	0.0058	0.6636	0.0012	0.032	-0.0018
10	0	0	0	-30	0	0	-30	0.0587	-0.0275	0.7169	-0.0025	-0.0417	0.0138
10	0	0	0	-30	0	0	0	0.0565	-0.0163	0.6964	0.0012	-0.018	0.0085
10	0	0	0	-30	0	0	30	0.054	-0.0051	0.6761	0.0047	0.0057	0.0032
10	0	0	0	-30	0	30	0	0.0607	-0.0385	0.7299	0.0014	-0.0684	0.0189

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
10	0	0	0	-30	90	0	0	0.0541	0.0013	0.701	-0.0341	-0.0188	-0.0009
10	0	0	0	0	-90	0	0	0.0536	-0.0186	0.695	0.0366	-0.0117	0.0097
10	0	0	0	0	0	-30	0	0.0486	0	0.6459	-0	0.0537	-0
10	0	0	0	0	0	0	-30	0.0569	-0.0002	0.6919	-0.0033	-0.0115	0.0001
10	0	0	0	0	0	0	0	0.0569	0	0.6918	-0	-0.0115	-0
10	0	0	0	0	0	0	30	0.0569	0.0002	0.6919	0.0033	-0.0115	-0.0001
10	0	0	0	0	0	30	0	0.0631	0	0.7389	-0	-0.0776	-0
10	0	0	0	0	90	0	0	0.0536	0.0186	0.695	-0.0366	-0.0117	-0.0097
10	0	0	0	30	-90	0	0	0.0541	-0.0013	0.701	0.0341	-0.0188	0.0009
10	0	0	0	30	0	-30	0	0.0508	-0.0058	0.6636	-0.0012	0.032	0.0018
10	0	0	0	30	0	0	-30	0.054	0.0051	0.6761	-0.0047	0.0057	-0.0032
10	0	0	0	30	0	0	0	0.0565	0.0163	0.6964	-0.0012	-0.018	-0.0085
10	0	0	0	30	0	0	30	0.0587	0.0275	0.7169	0.0025	-0.0417	-0.0138
10	0	0	0	30	0	30	0	0.0607	0.0385	0.7299	-0.0014	-0.0684	-0.0189
10	0	0	0	30	90	0	0	0.0527	0.0356	0.7018	-0.0363	-0.0214	-0.0187
10	0	0	0	60	-90	0	0	0.0484	0.0014	0.6851	0.0388	-0.0012	-0.0011
10	0	0	0	60	0	-30	0	0.0483	-0.0035	0.6752	0.0039	0.0185	-0.0002
10	0	0	0	60	0	0	-30	0.0478	-0.0179	0.6577	-0.0003	0.0273	0.0066
10	0	0	0	60	0	0	0	0.0499	0.0164	0.68	0.0034	0.0006	-0.0094
10	0	0	0	60	0	0	30	0.0505	0.0513	0.7035	0.0074	-0.0273	-0.0257

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
10	0	0	0	60	0	30	0	0.051	0.0364	0.6849	0.0028	-0.0172	-0.0186
10	0	0	0	60	90	0	0	0.0447	0.0314	0.6816	-0.0322	0.0022	-0.0176
10	0	0	0	90	-90	0	0	0.0506	-0.0165	0.6696	0.0357	0.0208	0.0089
10	0	0	0	90	0	-30	0	0.0541	0	0.6739	-0	0.0243	-0
10	0	0	0	90	0	0	-30	0.0533	-0.0484	0.6663	-0.0029	0.0212	0.0225
10	0	0	0	90	0	0	0	0.054	0	0.6662	-0	0.021	-0
10	0	0	0	90	0	0	30	0.053	0.0484	0.6664	0.0029	0.0211	-0.0225
10	0	0	0	90	0	30	0	0.0539	0	0.6585	-0	0.0177	-0
10	0	0	0	90	90	0	0	0.0506	0.0165	0.6696	-0.0357	0.0208	-0.0089
10	0	0	20	-90	0	0	0	0.0949	-0.0261	0.6645	-0.0435	0.0209	0.0073
10	0	0	20	-60	0	0	0	0.0974	-0.0386	0.6859	-0.0462	-0.0044	0.0155
10	0	0	20	-30	0	0	0	0.0999	-0.0275	0.705	-0.0399	-0.0257	0.0103
10	0	0	20	0	0	0	0	0.0973	-0.0053	0.6893	-0.0391	-0.0101	-0.0001
10	0	0	20	30	0	0	0	0.0946	0.0032	0.6838	-0.0426	-0.0094	-0.0053
10	0	0	20	60	0	0	0	0.0838	-0.0064	0.672	-0.0394	0.0037	-0.0027
10	0	0	20	90	0	0	0	0.095	-0.0261	0.6645	-0.0435	0.0206	0.0073
10	0	10	0	-90	0	0	0	0.0667	-0.1356	0.6647	0.0014	0.0191	0.0649
10	0	10	0	-60	0	0	0	0.0738	-0.1253	0.7453	-0.0011	-0.0818	0.0616
10	0	10	0	-30	0	0	0	0.0873	-0.0787	0.8101	0.0023	-0.1573	0.0384
10	0	10	0	0	0	0	0	0.0903	0	0.8199	-0	-0.167	-0

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

α	β	δ_a	δ_e	δ_B	p	q	r	C_D	C_S	C_L	C_ℓ	C_m	C_n
10	0	10	0	30	0	0	0	0.0873	0.0787	0.8101	-0.0023	-0.1573	-0.0384
10	0	10	0	60	0	0	0	0.0738	0.1253	0.7453	0.0011	-0.0818	-0.0616
10	0	10	0	90	0	0	0	0.0667	0.1356	0.6647	-0.0014	0.0191	-0.0649
10	2	0	20	-90	0	0	0	0.0993	-0.0749	0.6769	-0.051	0.0211	0.0317
10	2	0	20	-60	0	0	0	0.104	-0.0714	0.7194	-0.0486	-0.0284	0.0317
10	2	0	20	-30	0	0	0	0.1046	-0.0337	0.7314	-0.0434	-0.0429	0.0145
10	2	0	20	0	0	0	0	0.0978	-0.0037	0.6894	-0.0426	0.0028	-0.0001
10	2	0	20	30	0	0	0	0.0917	-0.0117	0.666	-0.0462	0.0243	0.0022
10	2	0	20	60	0	0	0	0.0843	-0.0149	0.6771	-0.04	0.0118	0.002
10	2	0	20	90	0	0	0	0.0998	-0.0749	0.6768	-0.051	0.0209	0.0317
10	4	0	20	-90	0	0	0	0.102	-0.1124	0.6904	-0.0574	0.0168	0.051
10	4	0	20	-60	0	0	0	0.1107	-0.1025	0.7507	-0.0513	-0.0523	0.0475
10	4	0	20	-30	0	0	0	0.1099	-0.0404	0.7577	-0.0472	-0.0622	0.019
10	4	0	20	0	0	0	0	0.0983	-0.002	0.6889	-0.0462	0.0143	0
10	4	0	20	30	0	0	0	0.089	-0.0257	0.6495	-0.0496	0.0544	0.0092
10	4	0	20	60	0	0	0	0.0907	-0.0716	0.6475	-0.0496	0.0611	0.0288
10	4	0	20	90	0	0	0	0.1027	-0.1124	0.6901	-0.0574	0.0168	0.051
10	6	0	20	-90	0	0	0	0.1073	-0.1436	0.702	-0.0589	0.0132	0.0669
10	6	0	20	-60	0	0	0	0.1189	-0.1376	0.7833	-0.0548	-0.0799	0.0652
10	6	0	20	-30	0	0	0	0.1155	-0.0469	0.7824	-0.0509	-0.082	0.0236

Table D.2: Truncated aerodynamic database of the BIRE aircraft (continued).

$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\delta_a}$	$\boldsymbol{\delta_e}$	$\boldsymbol{\delta_B}$	$\boldsymbol{p}$	$\boldsymbol{q}$	$\boldsymbol{r}$	$\boldsymbol{C_D}$	$\boldsymbol{C_S}$	$\boldsymbol{C_L}$	$\boldsymbol{C_\ell}$	$\boldsymbol{C_m}$	$\boldsymbol{C_n}$
10	6	0	20	0	0	0	0	0.0988	-0.0003	0.6879	-0.0498	0.0242	0.0002
10	6	0	20	30	0	0	0	0.0863	-0.0394	0.6341	-0.0529	0.0809	0.0162
10	6	0	20	60	0	0	0	0.0959	-0.1169	0.6283	-0.0582	0.0956	0.0513
10	6	0	20	90	0	0	0	0.1082	-0.1436	0.702	-0.0588	0.0129	0.067

APPENDIX E

CURRICULUM VITAE

CHRISTIAN R. BOLANDER

Utah State University, Logan, Utah | 385.321.4350 | christian.bolander@aggiemail.usu.edu
LinkedIn | ResearchGate | Google Scholar

Education

Ph.D.	**Utah State University**	Mechanical Engineering	*Est. May 2023*
B.S.	**Utah State University**	Mechanical and Aerospace Engineering	*May 2018*

Professional Positions

Engineering Math Resource Center Director	Utah State University	*Jul. 2022 - Present*
Assistant Professor of Practice	Utah State University	*Jul. 2022 - Present*
Research Assistant	Utah State University Aerolab	*May 2017 - Jul. 2022*
Engineering Tutor Center Manager	Utah State University	*Aug. 2017 - May 2018*
Systems Engineering Intern	Hill Air Force Base	*May 2017 - Dec. 2017*

Notable Projects

USU Engineering Math Resource Center

- Started a novel engineering-based math learning center on the USU campus.
- Developed online content focused on introducing and reviewing math content with students in an engineering context.

Bio-inspired Rotating Empennage

- Performed aerodynamic analysis on a bio-inspired empennage design for fighter aircraft to improve efficiency and control authority.
- Analyzed aerodynamics of a fighter aircraft using both high-fidelity (CFD) and low-fidelity (lifting-line) aerodynamic tools.

Compressible Fluid Flow Instructor

- Developed a course to teach undergraduate students in their junior or senior year the fundamentals of compressible fluid flow and supersonic aerodynamic analysis.
- Nominated for Graduate Student Instructor of the Year by students in first semester of teaching.

Sonic Boom Loudness Mitigation

- Implemented a procedure to estimate the human-perceived loudness of a sonic boom using pressure information from an aircraft.
- The resulting code, PyLdB, is open-source and has been used by researchers in academia, including researchers at Texas A & M University and the U.S. Department of Transportation, to analyze sonic boom mitigation.
- Analyzed the effect of aircraft shape-deformation actuation technology on sonic boom loudness using mid-fidelity (3D panel methods) aerodynamic tools.

Ship Deck Motion Prediction

- Designed a methodology for predicting ship deck motion in 6 degrees of freedom using simulated acceleration data and signal analysis techniques.

Folding One-Step Rod Cutter Design

- Designed and analyzed loads on a hydraulic mechanism to lift and store two 6,000 lb. bean harvesters for transportation on a tractor.
- Prototype in development at Pickett Equipment for production.

Publications

1. **Bolander**, C. R., Kohler, A. J., Hunsaker, D. F., Myszka, D., and Joo, J. J., ”Static Trim of a Bio-Inspired Rotating Empennage for a Fighter Aircraft,” AIAA Scitech Forum, January 2023, DOI: 10.2514/6.2023-0624
2. Kohler A. J., **Bolander**, C. R., Hunsaker, D. F., Joo, J. J., ”Linearized Rigid-Body Static and Dynamic Stability of an Aircraft with a Bio-Inspired Rotating Empennage,” AIAA Scitech Forum, January 2023, DOI: 10.2514/6.2023-0621

3. Harvey, C. Gamble, L. L., **Bolander**, C. R., Hunsaker, D. F., Joo, J. J., and Inman, D. J., "A review of avian-inspired morphing for UAV flight control," Progress in Aerospace Sciences 132, July 2022, DOI: 10.1016/j.paerosci.2022.100825
4. Ives, C., Myszka, D. H., Joo, J. J., **Bolander**, C. R., and Hunsaker, D. F., "Attainable Moment Set and Actuation Time of a Bio-Inspired Rotating Empennage," AIAA Scitech Forum, January 2022, DOI: 10.2514/6.2022-1670
5. **Bolander**, C. R., Hunsaker, D. F., Myszka, D., and Joo, J. J., "Attainable Moment Set and Actuation Time of a Bio-Inspired Rotating Empennage," AIAA Scitech Forum, January 2022, DOI: 10.2514/6.2022-1670
6. **Bolander**, C. R., and Hunsaker, D. F., "Near-field Pressure Signature Splicing for Low-Fidelity Design Space Exploration of Supersonic Aircraft," AIAA Scitech Forum, January 2020, DOI: 10.2514/6.2020-0789
7. Carpenter, F. L., Cizmas, P., **Bolander**, C. R., Giblette, T. N., and Hunsaker, D. F., "A Multi-Fidelity Prediction of Aerodynamic and Sonic Boom Characteristics of the JAXA Wing Body," AIAA Aviation 2019 Forum, June 2019, DOI: 10.2514/6.2019-3237
8. **Bolander**, C. R., Hunsaker, D. F., Shen, H., and Carpenter, F. L., "Procedure for the Calculation of the Perceived Loudness of Sonic Booms," AIAA Scitech Forum, January 2019, DOI: 10.2514/6.2019-2091
9. **Bolander**, C., and Hunsaker, D. F., "A Sine-Summation Algorithm for the Prediction of Ship Deck Motion," OCEANS 2018 MTS/IEEE Charleston, October 2018, DOI: 10.1109/OCEANS.2018.8604888

Awards

Graduated Magna Cum Laude
3.89 GPA
Seely-Hinckley Scholarship
1 of 8 selected from Utah State University graduate student body
NSF Graduate Research Fellowship Program Honorable Mention
2019 Cohort
Tau Beta Pi Fellowship
$10,000 Award

Service

Chief Advisor, Tau Beta Pi Engineering Honor Society, UT Gamma Chapter, *Aug 2022 - Present*
VP of Professional Development, Tau Beta Pi Engineering Honor Society, *May 2018 - August 2019*
President, Tau Beta Pi Engineering Honor Society, *August 2017 - May 2018*