



**STUDY OF CHAOTIC BEHAVIOR IN THE
NONLINEAR DYNAMIC RESPONSE OF AN
AIRFOIL WITH A TRAILING EDGE FLAP**

THESIS

Joshua J. Lee, First Lieutenant, USAF
AFIT-ENY-MS-17-M-272

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENY-MS-17-M-272

STUDY OF CHAOTIC BEHAVIOR IN THE NONLINEAR DYNAMIC
RESPONSE OF AN AIRFOIL WITH A TRAILING EDGE FLAP

THESIS

Presented to the Faculty
Department of Aeronautical Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Aeronautical Engineering

Joshua J. Lee, B.S.M.E.

First Lieutenant, USAF

March 3, 2017

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENY-MS-17-M-272

STUDY OF CHAOTIC BEHAVIOR IN THE NONLINEAR DYNAMIC
RESPONSE OF AN AIRFOIL WITH A TRAILING EDGE FLAP

Joshua J. Lee, B.S.M.E.
First Lieutenant, USAF

Committee Membership:

Dr. Anthony Palazotto, PhD
Chair

Dr. Donald Kunz, PhD
Member

Dr. William Baker, PhD
Member

Dr. Crystal L. Pasiliao
Member

Abstract

Currently, airfoils are used to calculate the speed at which the aerodynamic forces and the structure create a self exciting and unstable system. This flutter speed is described as violent, unstable, and self exciting, but not chaotic. Even though flutter is a linear phenomenon, no connections between Limit Cycle Oscillation (LCO), a non-linear version of flutter, and chaos have been made. To make this connection, first, an internal structure was optimized to represent the average internal structure spanning the wing. Next, any changes to the aerodynamics of the deformed airfoil were quantified to be negligible before proceeding to building the Finite Element Model (FEM). The FEM utilized as many specifications as possible from the experimental model utilized by Conner [1]. The chaos of the system was analyzed by changing the initial position of the flap deflection angle, simulating with and without freeplay, exciting at two different frequencies, and applying the excitation force at two different locations. The dynamic analysis of the system displayed that the short duration of the simulations did not allow for proper chaos analysis. Therefore, longer simulations were performed and showed that the linear system, without freeplay, had a Lyapunov exponent converging to zero. This proved that the parameters used to calculate the Lyapunov exponent were acceptable, because a linear system cannot have a positive Lyapunov exponent. For the nonlinear system, with freeplay, the chaos or lack thereof could not be determined. This is due to the constantly fluctuating Lyapunov exponent, suggesting that longer simulations must be performed. Throughout all of the simulations, when freeplay was introduced into the system, the Power Spectral Density (PSD) became more erratic showing that the system became nonlinear, a prerequisite for being chaotic. Because the nonlinear system could not be classified as chaotic or not, no connection between LCO and chaos could be made.

Acknowledgments

First I would like to thank Dr. Palazotto for his patience, knowledge, communication skills, and leadership throughout my time at AFIT. Without our many talks and idea bouncing meetings, this thesis would never have made it this far. Many thanks go to my friends and colleagues Kevin, Carol, Jordan, Zach, and Chazz for their endless help and support through the thesis process. Thank you to my parents for always being there for me, even if that means answering late night calls from 9 hours away. Most of all, thank you to Amelia for your patience, calm demeanor, and most importantly your ability to know when I needed support and when I needed to be alone to think. Finally I would like to thank Dr. Kunz, Dr. Baker and Dr. Pasiliao for their interest and support in the topic.

Joshua J. Lee

Table of Contents

	Page
Abstract	iv
Acknowledgments	v
List of Figures	ix
List of Tables	xiv
List of Acronyms	xv
List of Symbols	xvi
I. Introduction	1
1.1 Chapter Overview	1
1.2 Objective	1
1.3 Motivation	2
1.4 Background	3
1.5 Methodology	4
1.6 Overview	5
II. Background	6
2.1 Chapter Overview	6
2.2 Topology Optimization	6
2.3 Computational Aerodynamics (CA)	8
2.3.1 Potential Flow Theory	8
2.3.2 Panel Method	10
2.3.3 Computational Fluid Dynamics (CFD)	11
2.3.4 Finite Difference	11
2.4 Aeroelasticity	12
2.5 Dynamics	14
2.5.1 Single Degree of Freedom	15
2.5.2 Two Degree of Freedom	16
2.5.3 Multi Degree of Freedom	17
2.6 Finite Element Analysis	18
2.7 Previous Work	19
2.8 Chaos	22
2.8.1 Single Pendulum	22
2.8.2 Double Pendulum	23
2.8.3 Lyapunov Exponent	24
2.8.4 PSD	25
2.9 Summary	26

	Page
III. Research Methodology	27
3.1 Chapter Overview	27
3.2 Initial Loading Conditions	28
3.2.1 XFOIL Inviscid	29
3.2.2 XFOIL Viscous	30
3.2.3 XFOIL Outputs	30
3.3 Topology Optimization Model	30
3.3.1 Loads and Boundary Conditions	32
3.3.2 Design Variables, Constraints, and Objective	35
3.4 Fluid Structure Interaction	36
3.5 Dynamic Finite Element Model	38
3.5.1 Solidworks and Abaqus Transition	38
3.5.2 Kinematics and Interaction	40
3.5.3 Boundary Conditions	41
3.5.4 Modeling Pitch, Plunge, and Flap Springs	42
3.5.5 Frequency Analysis	44
3.5.6 Dynamic Simulation	44
3.5.7 Initial Conditions	46
3.6 Summary	47
IV. Results and Analysis	48
4.1 Chapter Overview	48
4.2 Initial Loading Conditions	48
4.3 Topology Optimization	49
4.3.1 Constraining Center of Gravity	50
4.3.2 Final Optimized Internal Structure	57
4.3.3 Beam Buckling	58
4.4 Fluid Structure Interaction	59
4.4.1 Mesh and Model Convergence	59
4.4.2 Skin Deformation Effects	62
4.5 Dynamic Finite Element Model	68
4.5.1 Frequency Analysis	68
4.5.2 Freeplay vs. No Freeplay	70
4.5.3 Changing Initial Conditions	83
4.5.4 Lyapunov Exponent	92
4.5.5 Effects of Frequency on Dynamics	93
4.5.6 Longer Simulations	97
V. Conclusions and Recommendations	107
5.1 Chapter Overview	107
5.2 Conclusions of Research	107
5.3 Significance of Research	108

	Page
5.4 Lessons Learned	108
5.5 Recommendations for Future Research	109
Appendix A. XFoil Command Line Commands	110
Appendix B. National Advisory Committee for Aeronautics (NACA) 0012 Defining Points	111
Appendix C. Matlab Code	114
Bibliography	143

List of Figures

Figure		Page
1.1	Nonlinear, snapback displacement of Lighter Than Air Vehicle (LTAV) structure under quasistatic load	4
2.1	Figures displaying the spring constant of a beam in bending and torsion	13
2.2	Graphic of wing section with spring boundary conditions	13
2.3	Example of Finite Element Analysis (FEA) method of creating a discrete model	19
2.4	Airfoil with control surface configuration	20
2.5	Restoring moment as a function of flap deflection angle	20
2.6	Single pendulum motion with phase space diagrams, without (left) and with (right) friction	23
2.7	Two different initial conditions for a double pendulum	24
3.1	Pictures of an internal structure consisting of spars and ribs	28
3.2	Figure used to calculate equivalent nodal forces	33
3.3	Figure of boundary conditions used for optimization	33
3.4	Figures of forces acting on surface of airfoil	34
3.5	Overall structure sketched on top of optimization picture	39
3.6	Figures of transition to solidworks	40
3.7	Figures of transition to solidworks	40
3.8	Figure of coupling constraint between wing body and flap	41
3.9	Figure of model with spring and pitch springs attached	42
3.10	Figure of spring interaction between wing body and flap	43
3.11	Figure of freeplay; gap in force response when displacements are present	44

Figure		Page
3.12	Figure of model with input force acting at the aerodynamic center	45
3.13	Figure of model with input force acting at the tip of the flap	46
4.1	Plot of C_p over the chord length output from XFOIL	49
4.2	Element density outputs from Optistruct with different center of gravity constraints	52
4.3	Element density outputs from Optistruct with different center of gravity constraints	53
4.4	Deformed structure plots (scale = 1000) with undeformed edge for different center of gravity constraints	55
4.5	Deformed skin plots of all optimizations	56
4.6	Element density plot of the final optimized structure	57
4.7	vonMises stress within the final optimized structure	58
4.8	Displacement of the final optimized internal structure scaled by 1000	58
4.9	Comparison of the coarse, medium, and fine grids using the k- ω turbulence model	60
4.10	Comparison of the medium, and fine grids using the k- ϵ turbulence model	61
4.11	Comparison of the k- ϵ and k- ω turbulence models for the fine mesh	62
4.12	Total flow field for very fine grid	63
4.13	Stagnation region of very fine grid	64
4.14	Plot of C_p over the chord length for the deformed and undeformed airfoils	64
4.15	Contour of velocity magnitude at the nose of the airfoil	66

Figure	Page
4.16	Contour of velocity magnitude zoomed in to show boundary layer effects 66
4.17	Plot of y^+ value over the surface of the airfoil 67
4.18	Mode shapes for modes 1 thru 4 69
4.19	Mode shapes for modes 4 thru 8 70
4.20	Displacement over time when $\beta = -1^\circ$ initially and the input force is located at the aerodynamic center 71
4.21	Displacement over time when $\beta = 0^\circ$ initially and the input force is located at the aerodynamic center 72
4.22	Displacement over time when $\beta = 1^\circ$ initially and the input force is located at the aerodynamic center 72
4.23	Displacement over time when $\beta = -1^\circ$ initially and the input force is located at the tip of the flap 73
4.24	Displacement over time when $\beta = 0^\circ$ initially and the input force is located at the tip of the flap 74
4.25	Displacement over time when $\beta = 1^\circ$ initially and the input force is located at the tip of the flap 74
4.26	PSD plots when $\beta = -1^\circ$ initially and the input force is located at the aerodynamic center 76
4.27	PSD plots when $\beta = 0^\circ$ initially and the input force is located at the aerodynamic center 77
4.28	PSD plots when $\beta = 1^\circ$ initially and the input force is located at the aerodynamic center 78
4.29	PSD plots when $\beta = -1^\circ$ initially and the input force is located at the tip of the flap 80
4.30	PSD plots when $\beta = 0^\circ$ initially and the input force is located at the tip of the flap 81
4.31	PSD plots when $\beta = 1^\circ$ initially and the input force is located at the tip of the flap 82

Figure	Page
4.32	Displacement vs time for all three initial conditions with no freeplay and force at the aerodynamic center 83
4.33	Displacement vs time for all three initial conditions with freeplay and force at the aerodynamic center 84
4.34	Displacement vs time for all three initial conditions with no freeplay and force at the tip of the flap 85
4.35	Displacement vs time for all three initial conditions with freeplay and force at the tip of the flap 86
4.36	Phase space plots for three different initial conditions with no freeplay and force at the aerodynamic center 88
4.37	Phase space plots for three different initial conditions with freeplay and force at the aerodynamic center 89
4.38	Phase space plots for three different initial conditions with no freeplay and force at the tip of the flap 90
4.39	Phase space plots for three different initial conditions with freeplay and force at the tip of the flap 91
4.40	Displacement vs time for $\beta = 0^\circ$, with and without freeplay, excited at 0.8 Hz 93
4.41	PSD plots when $\beta = 0^\circ$ initially and the input frequency is 0.8 Hz 95
4.42	PSD plots when $\beta = 1^\circ$ initially and the input force is located at the tip of the flap 96
4.43	Displacement vs time for $\beta = 0^\circ$, without freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 100 seconds 98
4.44	PSD for $\beta = 0^\circ$, without freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 100 seconds 98
4.45	Phase space plot for $\beta = 0^\circ$, without freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 100 seconds 99

Figure	Page
4.46	Displacement vs time for $\beta = 0^\circ$, without freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 300 seconds 100
4.47	PSD for $\beta = 0^\circ$, without freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 300 seconds 101
4.48	Phase space plot for $\beta = 0^\circ$, without freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 300 seconds 101
4.49	Displacement vs time for $\beta = 0^\circ$, with freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 100 seconds 102
4.50	PSD for $\beta = 0^\circ$, with freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 100 seconds 103
4.51	Phase space plot for $\beta = 0^\circ$, with freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 100 seconds 103
4.52	Displacement vs time for $\beta = 0^\circ$, with freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 300 seconds 105
4.53	PSD for $\beta = 0^\circ$, with freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 300 seconds 105
4.54	Phase space plot for $\beta = 0^\circ$, with freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 300 seconds 106

List of Tables

Table		Page
3.1	List of aircraft that utilize a NACA 0012 and their cruise speeds	28
3.2	List of model specifications	32
3.3	Table of lower and upper bounds defined for the center of gravity	35
3.4	List of all differing simulations	47
4.1	Table of optimization results	50
4.2	Modal frequencies	68
4.3	Table of Lyapunov exponents	92

List of Acronyms

CA	Computational Aerodynamics
CFD	Computational Fluid Dynamics
FEA	Finite Element Analysis
FEM	Finite Element Model
LCO	Limit Cycle Oscillation
LTAV	Lighter Than Air Vehicle
NACA	National Advisory Committee for Aeronautics
PDE	Partial Differential Equation
PSD	Power Spectral Density
RANS	Reynolds Averaged Navier-Stokes
SIMP	Solid Isotropic Material with Penalization
VF	Volume Fraction

List of Symbols

β	elevator deflection angle
δ	half of freeplay angle for flap
$\hat{n}$	unit normal vector
λ_1	dominant lyapunov exponent
ω_f	flutter frequency
ρ_{alum}	density of aluminum
A_1	half of the total area of airfoil
A_2	half of the initial design area
A_3	half of the skin area
c	chord
C_d	2D coefficient of drag
C_l	2D coefficient of lift
C_p	2D coefficient of pressure
E	modulus of elasticity
F	force
I_{min}	minimum moment of inertia
K_h	plunge spring constant
K_α	pitch spring constant
K_β	flap spring constant
L	length between nodes
L_{max}	longest unsupported beam length
p	pressure
p_∞	free stream pressure
P_{cr}	minimum buckling force
v_∞	free stream velocity
b	semi-chord
n	number of panel nodes

STUDY OF CHAOTIC BEHAVIOR IN THE NONLINEAR DYNAMIC RESPONSE OF AN AIRFOIL WITH A TRAILING EDGE FLAP

I. Introduction

1.1 Chapter Overview

Airfoils have been continuously researched and studied since the turn of the 19th century. Airfoils are used today to research the effects of the aerodynamic forces and the dynamics of the wing. The airspeed at which the aerodynamic forces cause the wing to become unstable is known as the flutter speed. A variation of flutter that is neutrally stable and determined only when known nonlinearities in the system are present is Limit Cycle Oscillation (LCO). Merriam-Webster describes chaos as “complete confusion and disorder : a state in which behavior and events are not controlled by anything.” [2] Mathematicians know chaos to be a nonlinear phenomenon. Until recent, no connections between LCO and chaos have been drawn, even though both derive from nonlinear systems. Therefore this research intends to investigate this potential connection between LCO and chaos.

This chapter will discuss the objectives for research, the motivation behind the research, background on the topic, assumptions, and briefly discuss the methodology that will be used.

1.2 Objective

The objective of this thesis is to investigate the probable connection between LCO and chaos through the use of Finite Element Analysis (FEA). Achieving this

requires proper representation of the airfoil structure, as well as calculation of the deformation effects on aerodynamic properties. Should this research find that there is, in fact, a connection between LCO and chaotic response, it would present the potential of testing for chaotic motion instead of potentially harmful finding of LCO in flight. The specific research objectives are:

- Develop a deformable internal structure for an airfoil with a trailing edge flap
- Determine the aerodynamic effects of the deformed post optimization airfoil
- Characterize the dynamic behavior of the airfoil when subjected to a time dependent sinusoidal force
- Quantify the sensitivity of the system to initial conditions
- Quantify the chaos of the system with and without freeplay

1.3 Motivation

Chaotic behavior has typically been investigated for presence within celestial mechanics, but only recently have structures started to be analyzed for chaotic behavior. Beam structures of a Lighter Than Air Vehicle (LTAV) have been investigated for chaos, but the lifting surfaces of heavier than air vehicles have not [3]. A gap of knowledge exists on the nonlinear dynamic response of the wings on aircraft.

Within the last 20 years, the nonlinearities present in the wings of aircraft have been studied for concerns of damage due to LCO. LCO is the response of a nonlinear dynamic system that is statically stable (returning to equilibrium), and bounded in displacement. The method for finding LCO is connected to the method for flutter, except the nonlinearities of the system must be known. This analysis of a nonlinear system suggests a potential for chaotic motion of the system. Not to mention the

fact that an aeroelastic airfoil with a control surface is nothing more than a double pendulum, on its side, with springs at the hinges. Therefore, there exists the possibility to identify the response of the airfoil as either chaotic or not, and potentially introduce a new prediction approach for LCO.

1.4 Background

Chaos in systems was first theorized by Henri Poincaré in the late 1800s. While walking the streets of Paris, it dawned on Poincaré that years earlier, while studying the positions, masses, and velocities of three bodies in relation to one another, also known as the three-body problem, that some trajectory solutions appeared to be stable, some appeared to be unstable, and some exhibited special characteristics from both. These special characteristics were non-periodic motion that neither converged nor diverged, indicating that the motion could not be classified as stable, unstable, or even marginally stable [4]. This newly discovered type of motion became known as chaos.

Chaos, in terms of mathematics and physics today, is defined as “the irregular and unpredictable time evolution of many nonlinear systems” by Baker and Gollub [4]. The unpredictable motion Baker and Gollub describe has been observed and analyzed within a LTAV structure. The deformation of the structure under a quasistatic pressure load can be viewed in Figure 1.1. In the breakout portion of Figure 1.1, it can be seen that as the pressure was increased, the displacement suddenly decreased, exhibiting the highly nonlinear response seen that is referred to as “snapback”. The load applied to cause the snapback was then applied dynamically to the LTAV structure and analyzed for predictability. The motion was found to be chaotic, due to the inability to predict its motion.

For a dynamic system to exhibit chaotic motion, the differential equations that

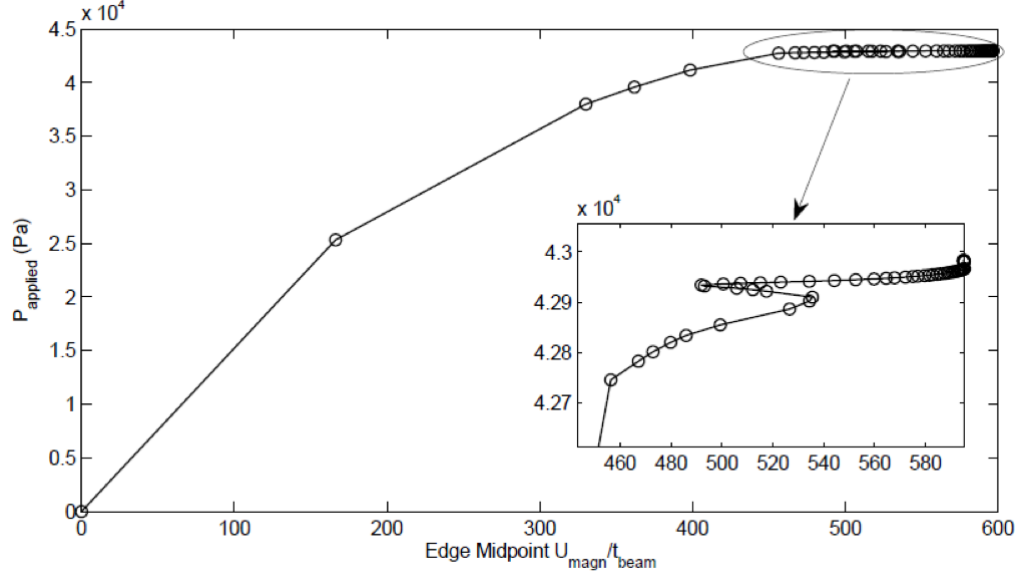


Figure 1.1. Nonlinear, snapback displacement of LTAV structure under quasistatic load [3]

define its motion must be nonlinear. In 1997, Conner developed a numerical and experimental model that analyzed a rigid/nondeformable National Advisory Committee for Aeronautics (NACA) 0012 airfoil with control surface freeplay of a trailing edge flap [1]. The freeplay in the hinge between the control surface and the airfoil provided the nonlinearity required for detection of LCO of the trailing edge flap at three different fractions of the flutter speed. This analysis provides both the linear and nonlinear systems needed for this analysis.

1.5 Methodology

To achieve the first two objectives of the research, an internal structure for the NACA 0012 airfoil must be developed. To do this, three-dimensional wing deformation must be transformed to a two-dimensional plane strain problem with pressure forces from cruise flight acting on the airfoil surface. Next, a volume fraction comparing the overall volume of the airfoil to the final volume of the airfoil must be

determined and entered into an structural optimization code. The program that will create the internal structure is called Optistruct.

To evaluate the next three objectives, a Finite Element Model (FEM) must be developed using a program called Abaqus. Within the program many parameters will vary such as the flap deflection angle's initial condition, the presence of freeplay, the frequency of the input force, and finally the placement of the input force.

1.6 Overview

- Chapter I: States the objective of this thesis, establishes the motivation, and briefly discusses the background and methodology
- Chapter II: Discusses the theory presented in relative literature related to chaotic behavior, structural dynamics, and explicit finite element modeling
- Chapter III: Lists modeling and testing methodology used for this research
- Chapter IV: Presents the results and discussion found
- Chapter V: Summarizes the results, draws conclusions, and discusses recommendations for future research

II. Background

2.1 Chapter Overview

Physical systems can be classified by many terms, including but not limited to linear, nonlinear, dynamic, static, time invariant, and time variant. The issues with attempting to classify all systems by these terms is that these terms can classify all linear systems, and that all real world systems are never strictly linear. Now that we can better compute solutions to nonlinear problems, different terms, like chaotic, have developed to describe systems.

The intention of Chapter 2 is to provide the reader with the information required to understand the investigation of possible chaotic response to a nonlinear problem used to predict a serious military aircraft problem, Limit Cycle Oscillation (LCO). To first understand the problem, one must know how dynamic systems are broken down and studied as systems of equations. Next, the basics of aeroelasticity are needed to understand flutter, the linear and unstable variant of LCO. Then, the development of the equations of motion will be covered, as well as the application of these equations into a numerical flutter solution [1]. Finally, the theory of chaos and previous work researching chaotic structures will be discussed.

2.2 Topology Optimization

Topology optimization is a method to design a material layout, in a design space, with given loads, boundary conditions, and constraints, all to maximize the performance of the structure. In 2001, O. Sigmund published a MATLAB code that did just that for a structure, following the Solid Isotropic Material with Penalization (SIMP) approach [5]. This approach assumes constant material properties for each element throughout the structure, but allows each element density to vary on a scale of 0 to

1, while ‘penalizing’ the optimization variable if elements are lower in density. With this approach, the optimization code requires an optimization variable, compliance, the inverse of stiffness. Sigmund’s code is written to minimize compliance; therefore maximizing stiffness of the structure [5]. Compliance can be written as

$$c(x) = \mathbf{U}^T \mathbf{K} \mathbf{U} = \sum_{e=1}^{N_{elem}} (x_e)^p u_e^T k_e u_e \quad (2.1)$$

where $\mathbf{U}$ is the global vector of displacements, $\mathbf{K}$ is the global stiffness matrix, N_{elem} is the number of elements, x_e is each elements density, p is the penalization constant, u_e is the local displacement vector, and k_e is the local stiffness matrix.

When optimizing, the constraint that prevents the code from placing all elements of density 1 within the design space is the volume fraction, seen in Equation 2.2 where f represents the final volume of the structure, $V(x)$, divided by the original volume of the structure, V_0 . And to quantify the stiffness and displacement variables for compliance, the model requires forces to solve Equation 2.3 [5].

$$\frac{V(x)}{V_0} = f \quad (2.2)$$

$$\mathbf{F} = \mathbf{K} \mathbf{U} \quad (2.3)$$

The desire of the code is to have an output where the densities are either a 1 or 0; however, sometimes penalizing does not result only in voids or full density material. Occasionally the code can produce a structure of semi dense elements or a structure with voids between higher density elements. This phenomenon is call ‘checkering’ and makes interpreting results extremely difficult. Countering this result will be discussed later in this thesis.

2.3 Computational Aerodynamics (CA)

Computational aerodynamics is an all encompassing general term used to describe numerically solving aerodynamic flow properties around complex objects. There are many ways to solve for these flow properties, some of which include semi-empirical methods, potential flow methods, and Computational Fluid Dynamics (CFD) methods. Semi-empirical methods combine both theory and experimental data. The experimental results fills in where the theoretical methods lack. This makes semi-empirical the simplest and lowest fidelity of the methods, because the others require solving of equations as opposed to looking up values.

2.3.1 Potential Flow Theory

The next highest fidelity model to solve for aerodynamic flow properties is the potential flow model. Known as the oldest and longest used of computer potential flow methods is the panel method [6]. Panel method excels at determining the aerodynamics of a streamlined shape at cruise conditions. The downfall of the panel methods lies with the assumptions that must be made to use potential flow theory. Potential flow theory starts by making the assumptions that the flow is steady, inviscid, and incompressible with no body forces. Applying these to the conservation of mass and momentum result in Equations 2.4 and 2.5, which form the basis for potential flow [6].

$$\vec{\nabla} \cdot \vec{V} = 0 \tag{2.4}$$

$$\rho(\vec{V} \cdot \nabla \vec{V}) = \rho \vec{\nabla} \left(\frac{V^2}{2} \right) - \rho \vec{V} \times \vec{\nabla} \times \vec{V} = -\vec{\nabla} p \tag{2.5}$$

Adding the assumption that the differential fluid element does not physically

rotate, irrotational flow, means that the curl equals zero, as seen in Equation 2.6 [6].

$$\vec{\omega} = \vec{\nabla} \times \vec{V} = 0 \quad (2.6)$$

Combining Equations 2.5 and 2.6 results in Euler's equation for the conservation of momentum, seen in Equation 2.7 [6].

$$\rho \vec{\nabla} \left(\frac{V^2}{2} \right) = -\vec{\nabla} p \quad (2.7)$$

Equations 2.4, 2.7, and 2.6 are integrated along a streamline and reduced to two dimensional flow resulting in the continuity equation, Bernoulli's equation, and the irrotational condition (Equations 2.8, 2.9, and 2.10 respectively) [6].

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \quad (2.8)$$

$$p + \frac{\rho V^2}{2} = p_0 \quad (2.9)$$

$$\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} = 0 \quad (2.10)$$

Using the velocity potential function, Φ , and the streamline potential function, Ψ , to combine and rewrite Equations 2.8 and 2.10, resulting in the Laplace Equations, seen in Equations 2.11a and 2.11b [6].

$$\frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} = \Phi_{xx} + \Phi_{yy} = \nabla^2 \Phi = 0 \quad (2.11a)$$

$$\frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} = \Psi_{xx} + \Psi_{yy} = \nabla^2 \Psi = 0 \quad (2.11b)$$

The Laplace equations are ordinary differential equations, allowing them to be

solved for individual shapes and added using linear superposition to provide the equations used for the panel method of computational aerodynamics.

2.3.2 Panel Method

Extending the potential flow theory to be used with panels means solving the three dimensional Laplace equation seen in Equation 2.12 [6].

$$\frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} + \frac{\partial^2 \Phi}{\partial z^2} = \Phi_{xx} + \Phi_{yy} = \nabla^2 \Phi = 0 \quad (2.12)$$

The equation is integrated starting with the Gauss divergence theorem and ending with Equation 2.13 [6].

$$\Phi = \Phi_{\infty} - \frac{1}{4\pi} \iint_S \left[\sigma \frac{1}{r} - \mu \frac{\partial}{\partial n} \left(\frac{1}{r} \right) \right] dS \quad (2.13)$$

Assuming two dimensional and replacing the doublet singularity with vortex singularity results in Equation 2.14 [6].

$$\Phi = \Phi_{\infty} + \int_S \left[\frac{q(s)}{2\pi} \ln r - \frac{\gamma(s)}{2\pi} \theta \right] dS \quad (2.14)$$

where

$$\Phi_{\infty} = V_{\infty}(x \cos(\alpha) + y \sin(\alpha))$$

The key to the panel method is the set up to achieve a system of algebraic equations. First, the surface of the aerodynamic shape must be represented by a number of straight line segments. Next, a constant source strength, $q_i(s)$, is assumed over each panel, but varies from panel to panel. Finally, a constant vortex strength, $\gamma(s)$, is assumed over each panel and is the same for each panel. After each of these

assumptions, the solution technique varies from program to program.

2.3.3 CFD

The highest fidelity way to computationally compute aerodynamic flow properties is CFD. Computing numerical solutions to the Partial Differential Equations (PDEs) of fluid mechanics encompass the work of CFD. The equations that define viscous fluid flow, the Navier-Stokes equations, are the equations of fluid mechanics that are typically solved, represented by Equation 2.15 [6].

$$\frac{\partial Q}{\partial t} + \frac{\partial(F - F_v)}{\partial x} + \frac{\partial(G - G_v)}{\partial y} + \frac{\partial(H - H_v)}{\partial z} = 0 \quad (2.15)$$

Solving PDEs can be done a number of ways, but the most often used is the method of finite difference. To accomplish this, the flow field must first be discretized by a mesh of grid points or cells. Then, the derivatives in the equation above can be written as algebraic equations of the points or cells. With these algebraic representations, a value for Q , the conserved variables, can be calculated for the entire grid, consisting of one iteration. Multiple iterations are performed until the entire flow field is converged.

2.3.4 Finite Difference

Representing a derivative from values known at different points on the grid starts with the Taylor series expansion of the derivative, seen in Equation 2.16 [6].

$$f(x + \Delta x) = f(x) + \frac{f'(x)}{1!}\Delta x + \frac{f^{(2)}(x)}{2!}\Delta x^2 + \dots + \frac{f^{(n)}(x)}{n!}\Delta x^n \quad (2.16)$$

Assuming the second order terms and higher are small enough to be neglected, the derivative of f can be solved for and seen in Equation 2.17 [6].

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x} \quad (2.17)$$

This method of solving for the derivative is known as the forward difference method, which only has first order accuracy. A multitude of other finite difference methods exist to represent derivatives in partial differential equations. Some have higher order accuracy while others utilize more grid points. The method of approximating the derivatives can be selected to best suite the computing power and specific simulation.

2.4 Aeroelasticity

When investigating the aeroelastic effects of a wing, one must first start with understanding that the three dimensional wing can be reduced to a two dimensional airfoil. This is done by understanding the motion/deformation of the wing when in flight. The wing will move in two directions; it will twist about its elastic axis, also known as pitching, and it will displace up and down, also known as plunging. The boundary conditions are defined by the real-world ends; the fixed end is attached to the aircraft and the free end is just that, free to move. These boundary conditions allow the wing to be treated as a simple cantilever beam with the movements described above. Figure 2.1 shows how the two degrees of freedom of the beam cause the beam to act as a spring with two spring constants, $k = \frac{3EI}{L^3}$ and $k_T = \frac{GJ}{L}$.

Now that the motion of the wing is known, the spring constant can be transformed into the spring constants seen in Figure 2.2. The plunge spring constant is k_h , and the torsional spring constant is k_θ . Point P represents the location of the elastic axis, and is where both spring forces act. Point Q represent the aerodynamic center, which means the aerodynamic forces of lift and drag act at this point. The location of Point Q is determined from thin airfoil theory for subsonic flow. Point C is the center of

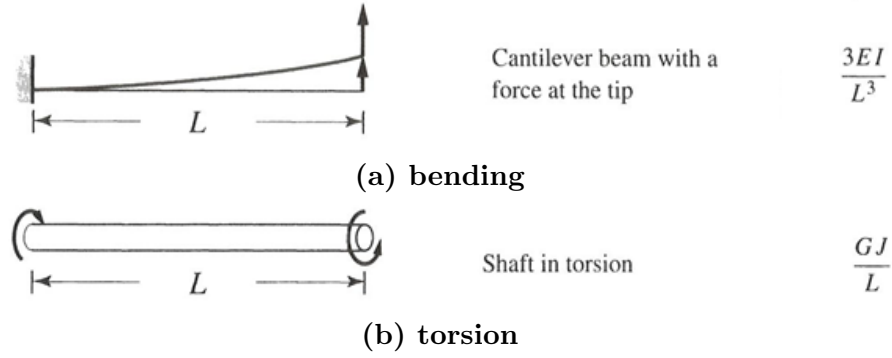


Figure 2.1. Figures displaying the spring constant of a beam in bending and torsion [7]

mass, and is where the weight force acts on the system.

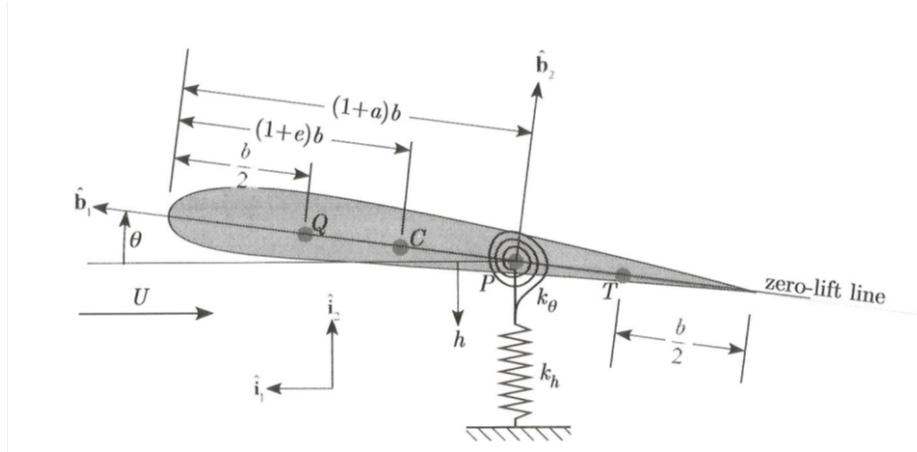


Figure 2.2. Graphic of wing section with spring boundary conditions [8] [9]

Drawing the free body diagram of the forces acting at points Q, C, and P results in the differential equations of motion similar to Equations 2.20a and 2.20b. Then the equations are reduced and expressed in terms of the two degrees of freedom: $\frac{h}{b}$ and θ . The plunging motion is described by the nondimensional displacement $\frac{h}{b}$, where b is the semi-chord of the airfoil and h is the displacement of the elastic axis. The pitching motion is defined by the nondimensional rotational displacement of the elastic axis, θ . When the differential equations of motion are combined, they result in a single equation of matrices and arrays seen below.

$$\begin{bmatrix} mb^2 & mb^2 x_\theta \\ mb^2 x_\theta & I_P \end{bmatrix} \begin{Bmatrix} \ddot{\frac{h}{b}} \\ \ddot{\theta} \end{Bmatrix} + \begin{bmatrix} mb^2 \omega_h^2 & 2\pi \rho_\infty b^2 U^2 \\ 0 & I_P \omega_\theta^2 - 2(\frac{1}{2} + a)\pi \rho_\infty b^2 U^2 \end{bmatrix} \begin{Bmatrix} \frac{h}{b} \\ \theta \end{Bmatrix} = \begin{Bmatrix} \sum F_h \\ \sum M_P \end{Bmatrix} \quad (2.18)$$

Because the wing is treated as a cantilever beam, Equation 2.18 lacks any damping terms. This is due to the fact that structural damping was assumed to be zero, leaving the equation with only a mass matrix and a stiffness matrix.

2.5 Dynamics

Typically, linear dynamic-systems theory is applied to many real-world applications to very closely approximate the actual response of the system. All dynamics problems start with applying Newton's second law. Equations 2.19a and 2.19b show the application of Newton's Second Law for a constant mass, linear/translational system and rotational system respectively [7].

$$\sum \mathbf{F} = \dot{\mathbf{p}} = m\mathbf{a} = m\dot{\mathbf{v}} = m\ddot{\mathbf{r}} \quad (2.19a)$$

$$\sum \mathbf{M} = \dot{\mathbf{H}} = I\alpha = I\dot{\omega} = I\ddot{\theta} \quad (2.19b)$$

where

$\mathbf{F}$ = Force

m = Mass

$\dot{\mathbf{p}} = \frac{d\mathbf{p}}{dt}$ = Time Derivative of Linear Momentum

$\mathbf{a}$ = Acceleration

$\dot{\mathbf{v}} = \frac{d\mathbf{v}}{dt}$ = Time Derivative of Velocity

$\ddot{\mathbf{r}} = \frac{d^2\mathbf{r}}{dt^2}$ = Second Time Derivative of Position

$\mathbf{M}$ = Moment

I = Mass Moment of Inertia

$\dot{\mathbf{H}} = \frac{d\mathbf{H}}{dt}$ = Time Derivative of Angular Momentum

α = Angular Acceleration

$\dot{\omega} = \frac{d\omega}{dt}$ = Time Derivative of Angular Velocity

$\ddot{\theta} = \frac{d^2\theta}{dt^2}$ = Second Time Derivative of Angular Position

2.5.1 Single Degree of Freedom

Analyzing real-world systems, dissects into discrete parts: springs, dampers, and masses. This method can be applied to both translational and rotational systems. Springs transform displacement, either translational or rotational, into force. Dampers convert a change in displacement over time (i.e. velocity), into force. Masses convert a change in velocity over time (i.e. acceleration), into force. Summing these discrete parts and setting them equal to the sum of the external forces acting on the system results in Equation 2.20a for translational systems and 2.20b for rotational systems. [7] [10]

$$m\ddot{\mathbf{x}} + c\dot{\mathbf{x}} + k\mathbf{x} = \sum \mathbf{F} \quad (2.20a)$$

$$I\ddot{\theta} + c_T\dot{\theta} + k_T\theta = \sum \mathbf{M} \quad (2.20b)$$

where

c = Damping Constant

k = Spring Constant

c_T = Torsional Damping Constant

k_T = Torsional Spring Constant

2.5.2 Two Degree of Freedom

Equations 2.20a and 2.20b are derived for single degree of freedom systems. “The number of degrees of freedom is the number of independent coordinates necessary to describe the motion of a system fully” [7]. For a system that requires two coordinates to fully describe its motion, two differential equations must be written. These equations can then be combined into a matrix or system of equations that are functions of the two independent coordinates. This can be seen in Equation 2.21 [7].

$$[M] \begin{Bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{Bmatrix} + [C] \begin{Bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{Bmatrix} + [K] \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} \sum F_1 \\ \sum F_2 \end{Bmatrix} \quad (2.21)$$

where

$$[M] = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} = \text{Mass Matrix}$$

$$\begin{Bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{Bmatrix} = \text{Second Time Derivative of Both Degrees of Freedom}$$

$$[C] = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \text{Damping Matrix}$$

$$\begin{Bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{Bmatrix} = \text{First Time Derivative of Both Degrees of Freedom}$$

$$[K] = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} = \text{Stiffness Matrix}$$

$$\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \text{Array of Both Degrees of Freedom}$$

$$\begin{Bmatrix} \sum F_1 \\ \sum F_2 \end{Bmatrix} = \text{Sum of Forces Acting on Both Degrees of Freedom}$$

2.5.3 Multi Degree of Freedom

With the addition of another degree of freedom, the above mentioned two degree of freedom system can be extended to any degree of freedom system. Each associated degree of freedom has an associated mass that will have forces acting upon it and will generate a differential equation of motion like those seen in Equations 2.20a and 2.20b. These equations will be written in matrix form and can be seen in Equation 2.22 [7].

$$\begin{bmatrix} m_{11} & \dots & m_{1n} \\ \vdots & \ddots & \vdots \\ m_{n1} & \dots & m_{nn} \end{bmatrix} \begin{Bmatrix} \ddot{x}_1 \\ \vdots \\ \ddot{x}_n \end{Bmatrix} + \begin{bmatrix} c_{11} & \dots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \dots & c_{nn} \end{bmatrix} \begin{Bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_n \end{Bmatrix} + \begin{bmatrix} k_{11} & \dots & k_{1n} \\ \vdots & \ddots & \vdots \\ k_{n1} & \dots & k_{nn} \end{bmatrix} \begin{Bmatrix} x_1 \\ \vdots \\ x_n \end{Bmatrix} = \begin{Bmatrix} \sum F_1 \\ \vdots \\ \sum F_n \end{Bmatrix} \quad (2.22)$$

The idea of having a system of differential equations is crucial to methods that will be discussed later in this thesis.

2.6 Finite Element Analysis

Developing equations of motion, like those in Equations 2.21 and 2.22, is barely half of the problem. Solving that differential equation possesses most of the effort and time. That differential equation must be solved to determine a displacement function, $f(x_i, t)$, that will approximate/predict the position of its respective mass, m_i , and that approximation is only as good as the differential equation used to represent the system. In some cases, the differential equation developed is too complex to be solved. Some restrictions can be, but are not limited to: geometry of the problem, boundary conditions and loads, and material properties. One way to work around the problem is a continuation of the method mentioned in Section 2.2, breaking the problem up into smaller problems. [11]

Creating a finite element model starts with dividing the system into a finite number of masses. An example of this can be seen in Figure 2.3 when a cantilever beam with an irregular cross section was divided into three masses. Each mass has its own material properties, dimensions, is classified by a particular type of element, and has a stiffness. The stiffness of the element is defined by the material properties, dimensions, and type of element. An example of this can be seen in Figure 2.1, when a two dimensional beam was effectively described as a two dimensional spring with a spring constant. The mass and stiffness for each elements depends on certain degrees of freedom. In Figure 2.3, the middle element depends on the second and third degrees of freedom. Knowing this, the finite element model can then be represented by a four degree of freedom system, similar to Equation 2.22, without any damping. Each matrix would be a 4×4 matrix, while each array would be a 4×1 array.

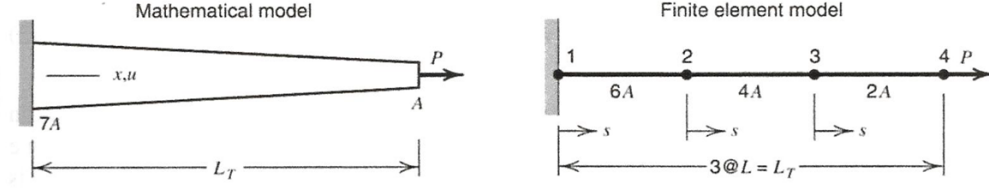


Figure 2.3. Example of Finite Element Analysis (FEA) method of creating a discrete model [11]

The method explained is used to create the finite element model, which is essentially an equation of matrices. The ability of the model to properly predict the displacement of the system at any point depends on the number of masses, also called the “fidelity” of the model. The higher the fidelity, the more masses/divisions, and the better the model can predict displacement.

To solve these large matrix equations, many solvers exist; however, for the purpose of this thesis, Abaqus will be the only solver discussed. Abaqus not only has the ability to solve static problems, like the one pictured in Figure 2.3, but it also has the ability to solve the dynamic problems like those described in Equation 2.22. Abaqus requires the use of direct-integration within Abaqus/Standard when a nonlinear dynamic response is being studied [12].

2.7 Previous Work

Wings alone can make an aircraft fly, but to maneuver an aircraft, control surfaces are required. Conner investigated a typical section, like the one in Figure 2.2, with the addition of a plain flap [1] [13]. The motion of the plain flap can be completely described by the rotation about its hinge; therefore, the two degree of freedom system just became a three degree of freedom system. To hold the plain flap in its neutral position, a torsional spring is attached to the hinge, and is assigned a spring constant.

The typical section, with the addition of the control surface, with freeplay, used by Conner, Tang, and Dowell can be seen below in Figure 2.4. Freeplay is when the

flap is allowed to move without any reaction from the torsional spring attached to it. This lack of reaction force can be seen as the horizontal portion of the restoring moment plot, labeled 2δ , in Figure 2.5.

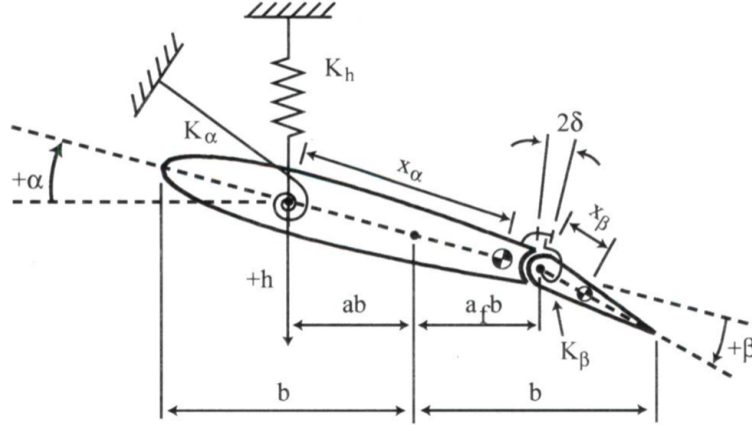


Figure 2.4. Airfoil with control surface configuration [14]

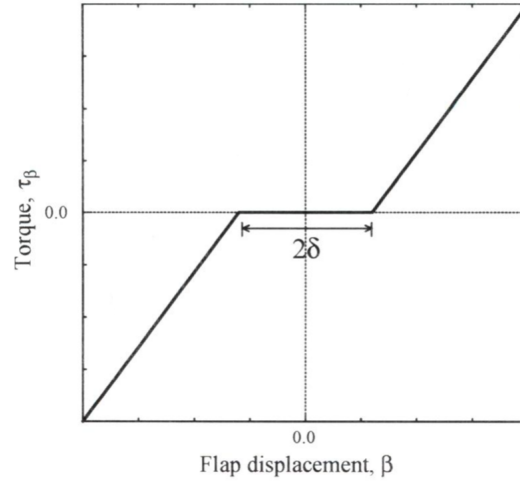


Figure 2.5. Restoring moment as a function of flap deflection angle (modified from [14])

Typically, spring constants are linear when performing aeroelastic calculations, but from Figure 2.5 it can be seen that the restoring spring constant for the plain flap is nonlinear, turning what has been a linear system into a nonlinear system. The equation used to determine the restoring moment plot above can be seen in Equation

2.23 [14].

$$\tau_\beta = \begin{cases} 0 & |\beta| \leq \delta \\ K_\beta(\beta - \delta) & |\beta| > \delta \end{cases} \quad (2.23)$$

Incorporating the new degree of freedom used to describe the plain flap was performed by Conner and Tang; however, Dowell, omitted the inclusion of the damping terms and published Equation 2.24 [1], [15], [14].

$$\left(-\bar{\omega}^2 [M] + \frac{4}{\mu V^2} [K] \right) \begin{Bmatrix} \bar{h} \\ \bar{\alpha} \\ \bar{\beta} \end{Bmatrix} = \frac{4}{\pi \mu} \begin{Bmatrix} -\bar{c}_l \\ 2\bar{c}_m \\ 2\bar{c}_n \end{Bmatrix} \quad (2.24)$$

where

$$[M] = \begin{bmatrix} M/m & x_\alpha & x_\beta \\ x_\alpha & r_\alpha^2 & (a_f - a)x_\beta + r_\beta^2 \\ x_\beta & (a_f - a)x_\beta + r_\beta^2 & r_\beta^2 \end{bmatrix}$$

$$[K] = \begin{bmatrix} \left(\frac{\omega_h}{\omega_\alpha}\right)^2 & 0 & 0 \\ 0 & r_\alpha^2 & 0 \\ 0 & 0 & r_\beta^2 \left(\frac{\omega_\beta}{\omega_\alpha}\right)^2 \end{bmatrix}$$

From this system of nonlinear equations, numerical integration determined that the theoretical flutter speed was 23.9 meters per second. Conner also put together an aeroelastic experimental model to test not only this flutter speed, but percentages of this speed for detection of LCO. Experimentally the flutter speed was found to be 20.6 meters per second and three speeds were determined to force the airfoil into LCO. They are 27%, 49%, and 73% of the flutter speed [1]. Because this experimental model is well documented, it was chosen to be analyzed for a connection between

chaos and flutter.

2.8 Chaos

Baker and Gollub define chaos as “the irregular and unpredictable time evolution of many nonlinear systems” [4]. They go on to further state that “interest in chaos has grown rapidly since 1963” [4]. This interest is most likely due to an increase in computational capacity. Even though interest has grown since 1963, defining chaos in terms of other systems can be challenging. A chaotic system is neither stable, unstable, nor marginally stable; all terms are used to describe linear system responses. The best way to describe a chaotic system is to compare two very similar systems; one of which is non chaotic, while the other is chaotic. The two systems are a single and double pendulum.

2.8.1 Single Pendulum

A single pendulum is a simple system, whose equation of motion can easily be solved for a displacement function. When disregarding friction, all of the potential energy from the initial conditions (top of the swing) will convert into kinetic energy (bottom of the swing) and then back into potential energy (top of the other side of the swing). This can be visualized by the left side of Figure 2.6, when the swing of the pendulum does not appear to be slowing. When friction is accounted for, the motion of the pendulum will be the same, with one exception. As the potential energy is transformed to kinetic energy, some energy is taken from the system in the form of friction. Therefore, looking at the right side of Figure 2.6, it can be seen that the swing of the pendulum does slow. The phase space diagram, with axes of velocity and position, shows that when friction is accounted for, the single pendulum moves toward an attractor. Crutchfield defines an attractor as “what the behavior of

the system settles down to, or is attracted to” [16]; therefore, the single pendulum’s attractor is halted motion.

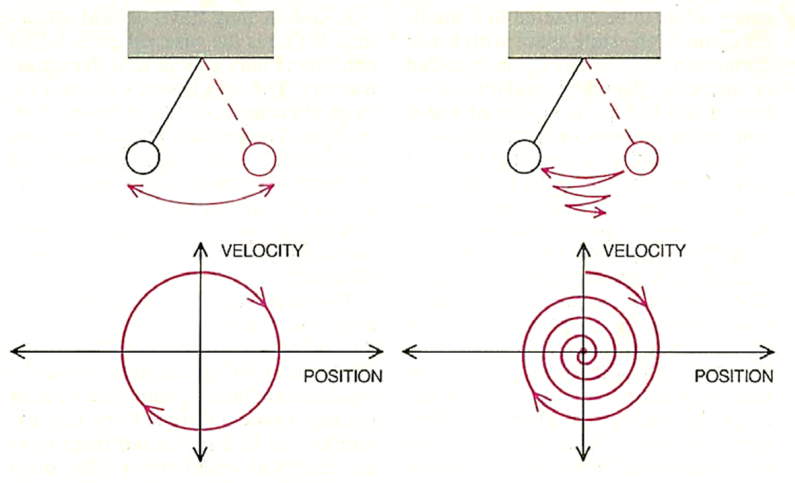


Figure 2.6. Single pendulum motion with phase space diagrams, without (left) and with (right) friction [16]

2.8.2 Double Pendulum

The double pendulum is a system of single pendulums linked at the end of the first pendulum. The equations of motion, while complicated, are known and solvable for given initial conditions. Figure 2.7 shows the motion for two different initial conditions of the same double pendulum. The response of the system is very sensitive to the initial conditions; this is another way that Baker defines chaotic systems [4]. The initial conditions of the two double pendulums in Figure 2.7 vary by only a few degrees, yet the responses vary greatly.

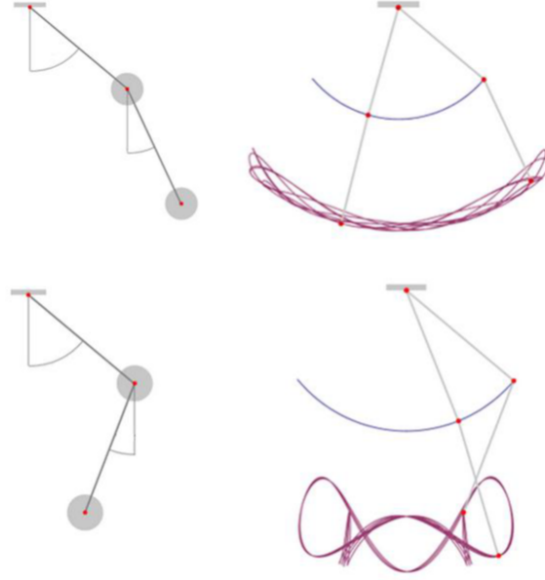


Figure 2.7. Two different initial conditions for a double pendulum [3]

2.8.3 Lyapunov Exponent

Once a time series displacement solution has been determined, a Lyapunov exponent may be calculated. Lyapunov exponents describe the divergence or convergence to/from the systems attractor, without going unstable. The magnitude of the exponent describes how fast this change happens. A positive exponent describes a chaotic system, while a negative exponent describes a non chaotic system. Wolf published a discrete way to calculate the exponential divergence from system attractors, and that can be seen in Equation 2.25 [17].

$$\lambda_1 = \frac{1}{t_N - t_0} \sum_{k=1}^n \log_2 \frac{L'_p(t_k)}{L_p(t_{k-1})} \quad (2.25)$$

where

λ_1 = Lyapunov Exponent

t_0 = Initial Time

t_N = Time at Step N

N = Total Number of Steps

$L_p(t_{k-1})$ = Distance Between Two Points on the Trajectory

$L'_p(t_k)$ = Evolved Distance Between Two Points at a Later Time

2.8.4 Power Spectral Density (PSD)

When dynamically analyzing a time dependent system, the displacement time function is key. However, when investigating for chaos, the frequency response of the system is also highly sought after, because it can be matched to the modal analysis performed within the Finite Element Model (FEM). To achieve this simply, the PSD function is used. The PSD shows the power of the system response for a range of frequencies. The frequencies are dependent upon the time step of the output variable. To calculate the PSD function, the autocorrelation function, which relates a variables value at one time to the same variables value at a second time, is utilized. The PSD function is simply the Fourier Transform of the autocorrelation function. The Fourier transform is a way of representing time dependent variables in the frequency domain. The autocorrelation function and PSD function can be seen in Equations 2.26a and 2.26b [7].

$$R_x(\tau) = \lim_{T \rightarrow \infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} x(t)x(t + \tau)dt \quad (2.26a)$$

$$S_x(\omega) = \int_{-\infty}^{\infty} R_x(\tau)e^{-i\omega\tau}d\tau \quad (2.26b)$$

The PSD provides a great deal of knowledge about the system from which the variable comes from, especially with regards to structures. Resonance peaks will form

at the natural frequencies of the structure, making no two PSDs alike [7]. If these peaks are distinct with smooth curves surrounding them, then the PSD is describing a linear system. If a nonlinearity is present, then the PSD will appear to be erratic or fuzzy in nature, making it more difficult to distinguish the natural frequencies of the system [4].

2.9 Summary

Research by Just and Forral have shown that chaos can be present in structures described by nonlinear equations. Continuing on research performed by Dowell, Tang, and Conner, who developed nonlinear equations to describe the aeroelastic motion of a two dimensional airfoil with control surface freeplay, it is possible for the airfoil structure to exhibit chaotic behavior: sensitivity to initial conditions, exponential error from its attractor, or general unpredictability.

The remainder of this thesis will discuss the methodology used within Optistruct, XFOIL, MATLAB, Abaqus, and ANSYS Fluent to develop the internal structure, test for changes in aerodynamic forces, and inspect the FEM for chaotic behavior.

III. Research Methodology

3.1 Chapter Overview

Associating that an airfoil with a trailing edge flap can be loosely perceived as a double pendulum on its side justifies the initial thought of investigating for chaos. Including the nonlinear spring holding the flap adds additional nonlinearity to an already chaotic double pendulum like system further solidifies the need for this investigation of the potential chaos looming in the system. Previously performed analyses on the National Advisory Committee for Aeronautics (NACA) 0012 airfoil, with a nonlinear spring retaining the trailing edge flap, have focused on the aeroelastic response of the rigid airfoil; however, the chaotic tendencies of the airfoil have not been investigated with the use of deformable geometry within Finite Element Analysis (FEA). Deformation of the internal structure could cause deformation of the airfoil skin and in turn, changes to the aerodynamic forces acting on the aeroelastic system. To add the deformation into the analysis, FEA is used to analyze the dynamics of the aeroelastic system with a deformable internal structure. It can be seen in Figure 3.1 that the internal structure of a wing consists of connected spars and ribs. Therefore, the cross sectional internal structure varies along the span of the wing and no exact two dimensional internal structure can be determined. To overcome this, a wing average internal structure is optimized using Altair Optistruct, which optimizes geometry based on several variables, including but not limited to constraints, desired responses, minimizing functions, and loads acting on the structure. This chapter breaks down the processes and materials used to create a Finite Element Model (FEM), perform a dynamic simulation of the system, and investigate the NACA 0012 airfoil in question for chaos.

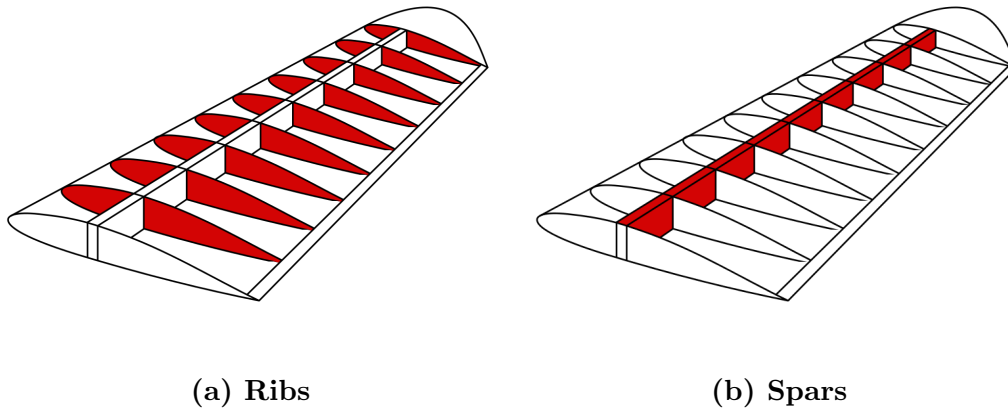


Figure 3.1. Pictures of an internal structure consisting of spars and ribs [18]

3.2 Initial Loading Conditions

The first step in the process outlined above begins with establishing initial loading conditions for the topology optimization process. When deciding on the initial loading conditions, many factors should be considered; however the condition that most makes sense is cruising speed and altitude for aircraft that fly with the NACA 0012 airfoil profile at some position along the span of the wing. These conditions present an easy computing problem as well, because time is not a factor. Table 3.1 lists a series of Cessna aircraft that fly with a tip profile of the NACA 0012 and their cruise speed. Along with the cruise speed, a cruise altitude of 7,000 feet is assumed. These conditions can be used to compute the aerodynamic properties surrounding the airfoil, providing the optimization force.

Table 3.1. List of aircraft that utilize a NACA 0012 and their cruise speeds

Aircraft	Cruise Speed (mph)
Cessna 140	105
Cessna 150	94
Cessna 152	123
Cessna 172	140
Cessna 182	167
Cessna 205	163
Cessna 210	220

3.2.1 XFOIL Inviscid

When looking at the conditions outlined above, the computationally easiest method that comes up is to use the panel method to calculate the pressure acting on the surface. The NACA 0012 is a very well defined airfoil, so a program that focuses on two dimensional aerodynamics would be preferred. This led to the use of XFOIL as the program to compute the optimization pressure acting on the airfoil. XFOIL is a free and easy to download program. It was developed by Drela at MIT [19]. The only limitations on utilizing XFOIL are in the input flow parameters. In order to properly calculate the aerodynamic properties, XFOIL requires the input of the Reynolds number and Mach number, 1.1×10^6 and 0.1885 respectively. Both of these numbers are within the convergence abilities of the code and do not require the calculation of compressibility effects. XFOILs inviscid solution starts with Equation 3.1, the linear-vorticity streamline formula.

$$\Psi = u_{\infty}y - v_{\infty}x + \frac{1}{2\pi} \int_S \gamma(s) \ln r(s) dS + \frac{1}{2\pi} \int_S \sigma(s) \theta(s) dS \quad (3.1)$$

Assuming that each separate panel has a constant source strength, σ_j , a constant vortex strength, γ_j , and that the streamline function has a constant value of Ψ_0 at each node of the airfoil results in Equation 3.2 [19]. Equation 3.2 along with the Kutta condition, Equation 3.3 provides the $N + 1$ equation system required to solve for the inviscid flow properties [19].

$$\sum_{j=1}^N \alpha_{ij} \gamma_j - \Psi_0 = -u_{\infty}y + v_{\infty}x - \sum_{j=1}^N b_{ij} \sigma_j \quad (3.2)$$

$$\gamma_1 + \gamma_N = 0 \quad (3.3)$$

3.2.2 XFOIL Viscous

The viscous solution for XFOIL begins with two governing equations; one for momentum, Equation 3.4, and the other for kinetic energy, Equation 3.5 [19].

$$\frac{d\theta}{d\xi} + (2 + H - M_e^2) \frac{\theta}{u_e} \frac{du_e}{d\xi} = \frac{C_f}{2} \quad (3.4)$$

$$\theta \frac{dH}{d\xi} + (2H + H(1 - H)) \frac{\theta}{u_e} \frac{du_e}{d\xi} = 2C_D - H \frac{C_f}{2} \quad (3.5)$$

These equations are discretized by utilizing the two point central difference method. The viscous effects are solved simultaneously with the inviscid equations by using a global Newtonian method.

3.2.3 XFOIL Outputs

XFOIL provides output in the form of C_p at each x location along the chord for the upper and lower surface. These values are pressure coefficients defined by Equation 3.6a. Equation 3.6b is a rearranged form of Equation 3.6a used to calculate the pressure acting normal to the surface of the airfoil. This pressure is what is used as the initial loading conditions for the topology optimization.

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty v_\infty^2} \quad (3.6a)$$

$$p = C_p \left(\frac{1}{2} \rho_\infty v_\infty^2 \right) + p_\infty \quad (3.6b)$$

3.3 Topology Optimization Model

The foundation for any FEM is understanding and realistically modeling the structure of the system in question. An airfoil is a two dimensional representation of a

wing, and it would not usually be used for FEA; however, due to the complexity of the system, a two dimensional analysis must be performed first. To generate the structure of the airfoil, Optistruct, a topology optimization software program developed by Hyperworks, is utilized. Optistruct takes the solid model, the loads, constraints, and an optimization variable as inputs and generates the optimal structure to withstand the loads present utilizing the Solid Isotropic Material with Penalization (SIMP) approach outlined earlier.

Building the model begins with the NACA 0012 airfoil shape generated as the outer skin. The equation to calculate the skin if a symmetric airfoil can be seen in Equation 3.7, where y_t is the thickness or distance from the flat chord for both the upper and lower surface. To generate this surface the chord of the airfoil must be known. As stated earlier, because of the well documented experimental model presented by Conner, specifications are drawn from that model for this investigation. Therefore, a chord length of 10 inches is used [1].

$$y_t = 5t \left[0.2969 \sqrt{\frac{x}{c}} - 0.1260 \left(\frac{x}{c} \right) - 0.3516 \left(\frac{x}{c} \right)^2 + 0.2843 \left(\frac{x}{c} \right)^3 - 0.1015 \left(\frac{x}{c} \right)^4 \right] \quad (3.7)$$

Next in the process of developing the model is deciding between the design and no design space. Any wing or airfoil internal structure requires to be connected to a skin. The skin is just an inward offset from the surfaces calculated above. The skin thickness of Conner's experimental model and this model are both 0.01 inches [1]. Thinking ahead to forming the dynamic FEM, a few points must lie within the internal structure to attach the springs. These points are the elastic axis and the flap hinge, 2.5 inches and 7.5 inches respectively, with the measurements coming from Conner's model again. The complete list of specifications used from Conner's model can be seen in Table 3.2

Table 3.2. List of model specifications [1]

Description	Value
Chord	10 (in)
Skin thickness	0.01 (in)
Mass per unit span	0.08724 (lb/in)
Elastic axis	2.5 (in)
Center of gravity	4.67 (in)
Flap hinge	7.5 (in)
Material	Aluminum

3.3.1 Loads and Boundary Conditions

The initial loading conditions must now be calculated as forces acting at the nodes in order to feed into the optimization program. Figure 3.2 and Equation 3.8 show how to calculate the equivalent nodal forces from the assumed constant pressure, p , acting over the surface between nodes. The Matlab code used to calculate and print the force vectors can be seen in Appendix C. The file with the forces is then uploaded back into Hyperworks for the the rest of the model to be determined. Figure 3.4 show the variance of the forces over the skin and a close up of the forces acting normal to the skin. The variance in force vectors can be explained by the change in length between the nodes and the the difference in C_p along the surface of the airfoil. To prevent the model from moving, boundary conditions must be applied. For simplicity and symmetry purposes, the boundary conditions are clamped on either end; the translational and rotational displacements are set to zero. These boundary conditions allow the airfoil to act as a beam. They can be seen in Figure 3.3.

$$F1 = F2 = \frac{\hat{n}pL}{2} \quad (3.8)$$

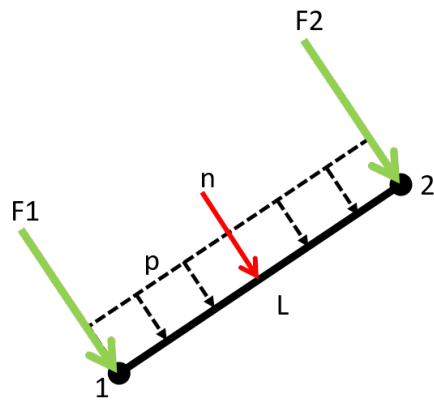


Figure 3.2. Figure used to calculate equivalent nodal forces

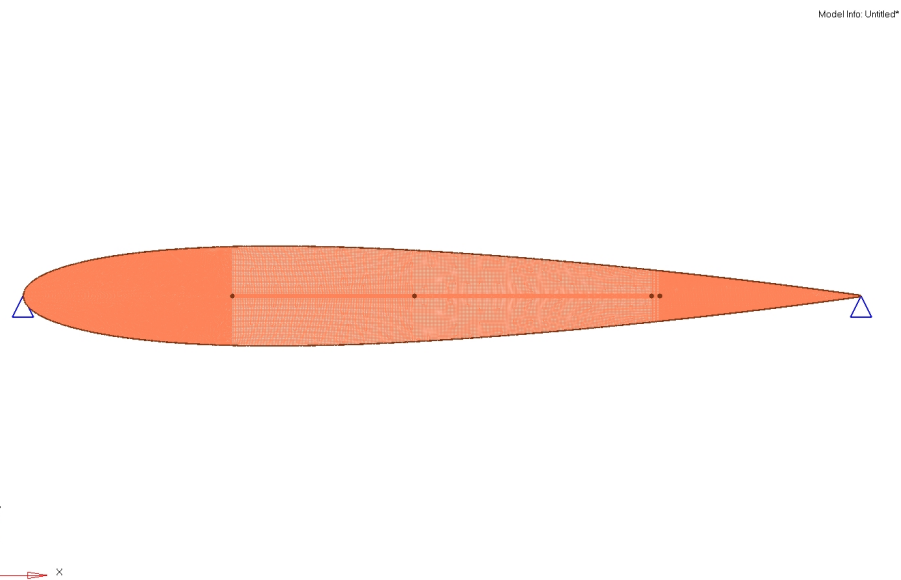


Figure 3.3. Figure of boundary conditions used for optimization

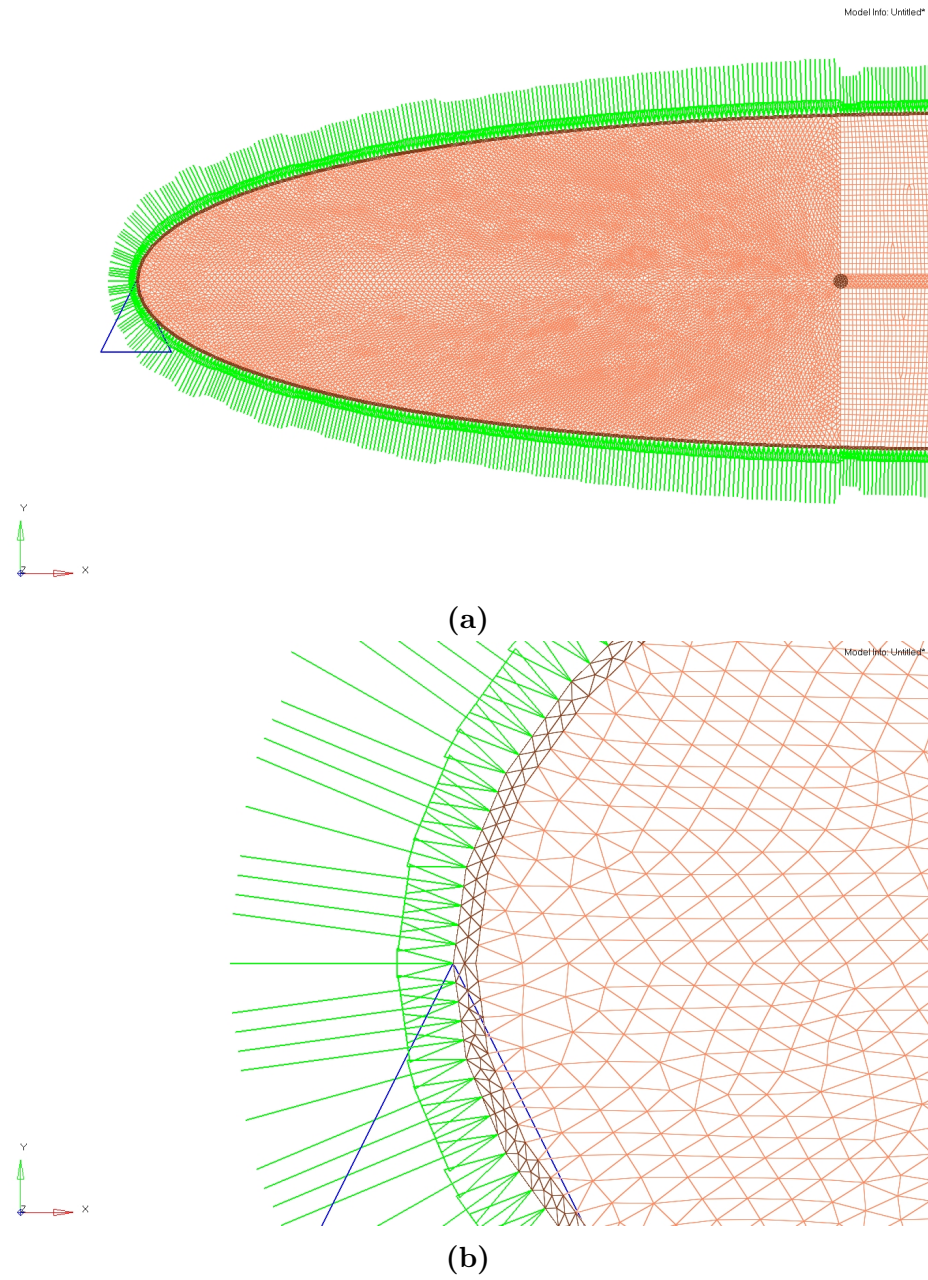


Figure 3.4. Figures of forces acting on surface of airfoil

3.3.2 Design Variables, Constraints, and Objective

Unlike the 99 lines code presented in Chapter 2, Optistruct allows the user to select more constraints than just volume fraction. However, just as in 99 lines, the objective of the optimization is to create an internal structure that is most resistant to deformation. Therefore, the design objective is to minimize compliance of the structure. The distance between the elastic axis and the center of gravity is one of the main reasons why aeroelastic problems exist and is also the reason why the center of gravity must be constrained. Conner’s experimental model had a center of gravity at 4.67 inches from the leading edge; however a study of the compliance for where the center of gravity is located was performed. The lower and upper bounds for this study can be seen in Table 3.3.

Table 3.3. Table of lower and upper bounds defined for the center of gravity

Lower Bound	Upper Bound
No Constraint	
4.6 (in)	4.7 (in)
4.65 (in)	4.69 (in)
4.67 (in)	4.69 (in)

The weight force acts at the center of gravity that is away from the elastic axis, causing rotational motion; therefore, the mass of the airfoil is crucial to the dynamics and aeroelastic analysis. To account for this, the volume fraction constraint was utilized. Volume Fraction (VF) is defined as the final volume required in the design area divided by the initial volume of the design area. This can be seen in Equation 3.9. Because the model is planar with unit thickness, the volume fraction can be calculated as an area fraction. Therefore in Equation 3.9, each volume used to calculate VF can be calculated as an area. A_3 is half of the area the skin creates and A_2 is half of the total internal area. Based on the mass per unit span given by Conner and seen in Table 3.2, the final volume fraction used to define the optimization is 0.0857, or 8.57%.

The volume fraction constraint is given with an upper bound, because Optistruct will continue to add mass to minimize compliance.

$$VF = \frac{\text{final volume of design space}}{\text{initial volume of design space}} = \frac{\left(\frac{mass}{\rho_{alum}}\right) - 2A_3}{2A_2} \quad (3.9)$$

With all of the design variables, constraints, and objective defined, the internal structure can now be optimized. It should be known that the optimization is not limited by iterations or computational time, only the constraints and objective variables.

3.4 Fluid Structure Interaction

After the optimization of the NACA 0012 internal structure, the potential for deformation of the outer skin is present. Even though the structure is optimized to maximize the stiffness, the pressure of the initial loading conditions could still provide deformation of the outer skin. If the outer skin deformed, then any change to the aerodynamic properties must be quantified. These changes play a key role in the aeroelastic and dynamic analysis of the airfoil. To test for this, computational aerodynamics must be performed on the deformed airfoil and compared against the undeformed airfoil. While XFOIL was utilized previously in this analysis, due to the minor deformations to the airfoil, turbulence must be accounted for with higher fidelity. Therefore, Computational Fluid Dynamics (CFD) is performed within ANSYS Fluent at the experimental test speeds set by Conner [1].

ANSYS Fluent is a graphical user interface and solver that allows for a multitude of different solution possibilities to the Navier-Stokes equations presented in Chapter 2. The direct numerical solutions to the full equations can be calculated; however, those are computationally expensive and not required in this situation. It is deemed that the Reynolds Averaged Navier-Stokes (RANS) solutions will suffice for such a problem. Within the RANS classification, multiple possibilities exist. Anything from

algebraic models to Eddy Viscosity models. For the problem at hand, a two equation model of the Navier-Stokes equations is chosen. ANSYS Fluent has two types of two equation models, the $k - \omega$ and the $k - \epsilon$. Each of these models represents fluid flow with two Partial Differential Equations (PDEs) that are discretized and solved for two unknowns within the grid, k and either ω or ϵ . Equations 3.10a and 3.10b are the equations for the $k - \epsilon$ two equation model, while Equations 3.11a and 3.11b are the equations for the $k - \omega$ two equation model, both within ANSYS Fluent [20].

$$\frac{\partial}{\partial t}(\rho k) + \frac{\partial}{\partial x_i}(\rho k u_i) = \frac{\partial}{\partial x_j} \left[\left(\mu + \frac{\mu_t}{\sigma_k} \right) \frac{\partial k}{\partial x_j} \right] + G_k + G_b - \rho \epsilon - Y_m + S_k \quad (3.10a)$$

$$\frac{\partial}{\partial t}(\rho \epsilon) + \frac{\partial}{\partial x_i}(\rho \epsilon u_i) = \frac{\partial}{\partial x_j} \left[\left(\mu + \frac{\mu_t}{\sigma_\epsilon} \right) \frac{\partial \epsilon}{\partial x_j} \right] + C_{1\epsilon} \frac{\epsilon}{k} (G_k + C_{3\epsilon} G_b) - C_{2\epsilon} \rho \frac{\epsilon^2}{k} + S_\epsilon \quad (3.10b)$$

$$\frac{\partial}{\partial t}(\rho k) + \frac{\partial}{\partial x_i}(\rho k u_i) = \frac{\partial}{\partial x_j} \left(\Gamma_k \frac{\partial k}{\partial x_j} \right) + G_k - Y_k + S_k \quad (3.11a)$$

$$\frac{\partial}{\partial t}(\rho \omega) + \frac{\partial}{\partial x_i}(\rho \omega u_i) = \frac{\partial}{\partial x_j} \left(\Gamma_\omega \frac{\partial \omega}{\partial x_j} \right) + G_\omega - Y_\omega + S_\omega \quad (3.11b)$$

Knowing that the equations above are to be discretized, the grid created is vital to the solution being valid. Four major decisions were made in discretization of the flow field around the airfoil. The first is the number of points used to define the airfoil, both deformed and undeformed. This number is used to converge the mesh. The number of points are 1636, 3272, 6344, and 10000 for each model. Next, because of the potential small nature of the deformation on the skin, a structured mesh was deemed necessary. With the unstructured mesh, the $y+$ or the boundary layer viscous grid height is kept below or at a value of 1. Finally, the boundary sizing is selected to be a minimum of 10 times the chord length in all directions [6]. With all of these

being known, the grid can be created and solved to provide any insight on the changes in aerodynamic properties from the rigid to optimized airfoil.

3.5 Dynamic Finite Element Model

With the effects of the deformed skin calculated, the finite element model of the aeroelastic airfoil with an internal structure can be built. This model will be used to analyze the potential chaos of the system by using Abaqus. To get to that point, a sketch and planar model must be built in Solidworks first. This process is followed because of the ease of drawing in Solidworks, the ability to work from a sketch picture, and the ability of the user to check the center of gravity of the sketch within Solidworks.

3.5.1 Solidworks and Abaqus Transition

Transitioning to Solidworks from Optistruct is done via a picture of the final optimized structure, but first two point cloud curves must be imported into Solidworks. These point clouds are generated in Matlab from the undeformed Hyperworks file and represent the inner and outer skin of either the upper or lower surface. The picture is then set as a sketch picture and is scaled to the converted curves to ensure the airfoil picture is the proper size. Next, beams are traced along the picture, realizing that the Optistruct output has some porosity and density changes. The overall beam tracing can be seen in Figure 3.5. To see how the porosity and density changes are handled, Figures 3.6 and 3.7 can be seen below. While generating the beam, the sectional properties are monitored to ensure the center of gravity and mass match that of Conner's experimental model [1].

The transition in Solidworks is when the wing body is separated from the flap. The total drawing of the skin, wing body, and flap all connected can be seen in Figure

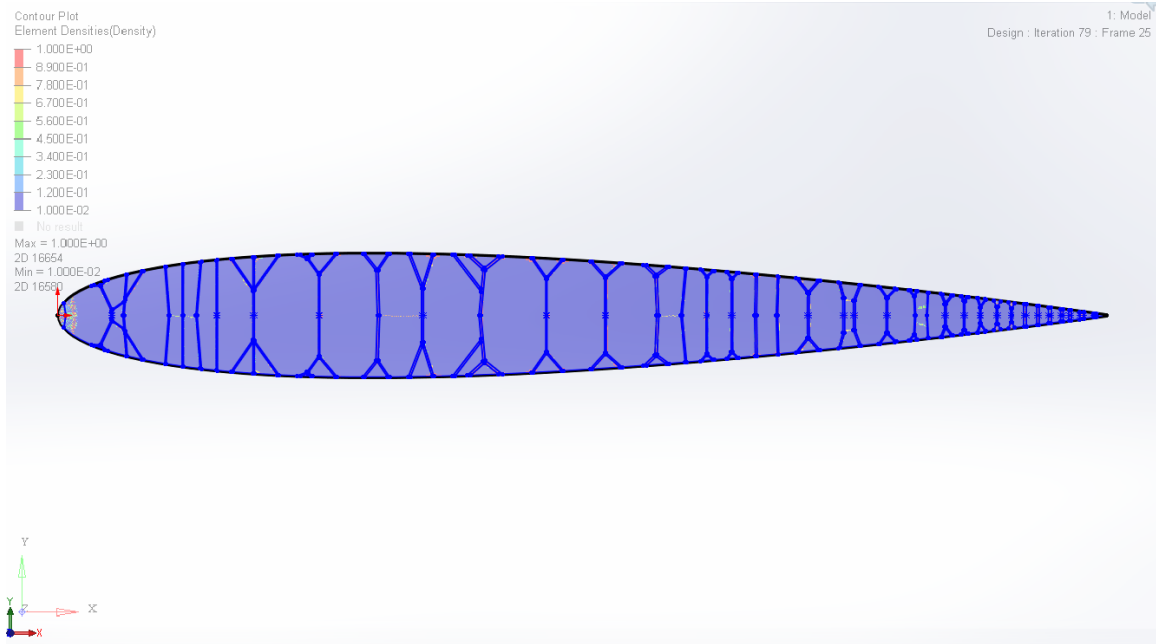


Figure 3.5. Overall structure sketched on top of optimization picture

3.7a. For the wing body and flap to be separated, the skin is removed between the first and second hinge beam. Afterwards, for each part, the other is deleted. The final end of the wing body can be seen in Figure 3.7d. The only loss in mass of the airfoil is from the 0.01 inch thick skin that is removed between the two parts, and is negligible. The distance between the wing body and flap is kept the same though, to ensure a chord length of 10 inches.

Each part is created as a planar representation, saved, and imported into Abaqus as separate parts in the same model. The parts are both inserted into the same assembly and are translated to ensure the distance between the hinge beams is the same as in Solidworks.

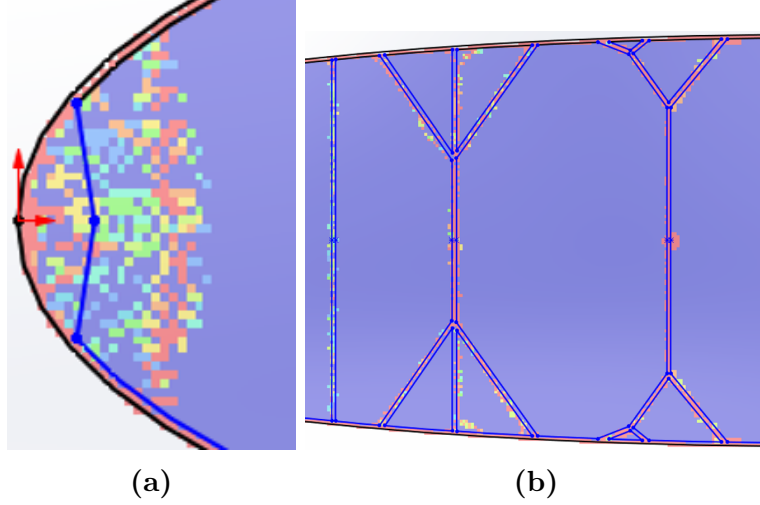


Figure 3.6. Figures of transition to solidworks

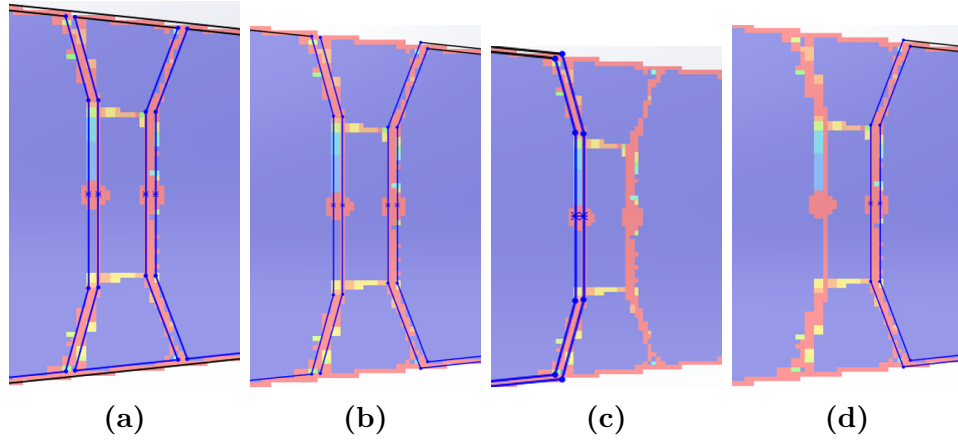


Figure 3.7. Figures of transition to solidworks

3.5.2 Kinematics and Interaction

Placing the instances into the assembly is the easy part of generating the dynamic FEM. Ensuring the instances move properly with respect to one another and the ground is a bit more difficult. The vertical and horizontal motion of the right end of the wing body must match that of the left end of the flap; however, the rotational degree of freedom is only defined by the flap spring, which is defined later. To achieve this motion, a kinematic coupling constraint is used to define the motion between the wing body and the flap. In Figure 3.8, the nodes to define the constraint can be seen.

The constrained degrees of freedom are 1 and 2 (x and y translation), while the 6 degree of freedom (z rotation) is left free from constraint.

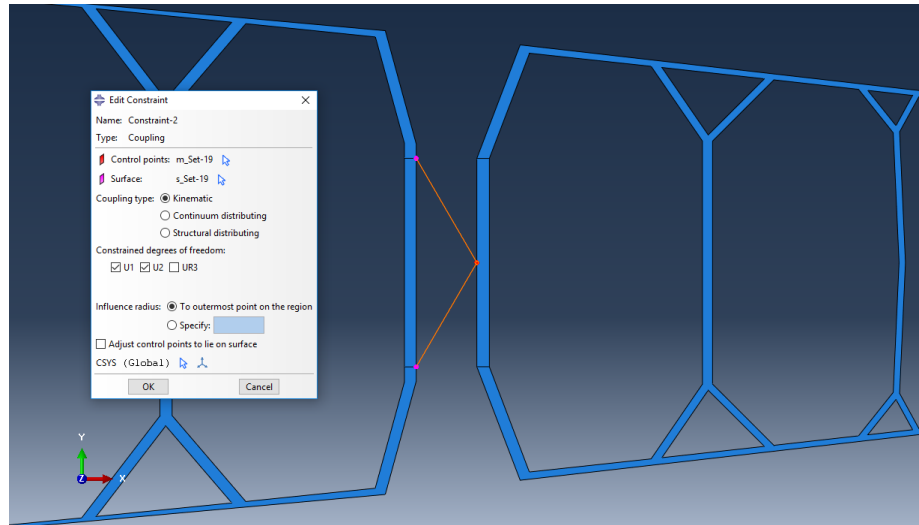


Figure 3.8. Figure of coupling constraint between wing body and flap

3.5.3 Boundary Conditions

Before springs are defined, each end of the spring must be defined first. For the springs that attach to the elastic axis on one end, they must also attach to the ground on the other end. To simulate the ground, fixed reference points are used. These reference points are all constrained from moving in the 1, 2 and 6 direction, as that is the only motion possible in a 2D planar model. The next boundary condition is created from understanding the kinematics behind aeroelasticity. The wing bending and torsion defined by the springs works if the airfoil is not free to move in the x direction. To constrain this, the elastic axis is held to zero displacement in the 1 direction.

3.5.4 Modeling Pitch, Plunge, and Flap Springs

With both ends of the springs fully defined, all that is left is to define the relative movement between them. The spring constants for all the springs are derived from Conner's experimental model [1]. For simplicity purposes, the pitch and plunge springs are defined first. Spring elements are used to connect the elastic axis of the airfoil to grounded reference points. The plunge spring requires only one axial spring element, while the pitch spring, defining the rotation of the wing and flap at the elastic axis, is defined using two axial spring elements an equal distance from the elastic axis. This assembly causes a couple at the elastic axis and responds the same as a rotational spring. These springs and how they attach to the model can be seen in Figure 3.9.

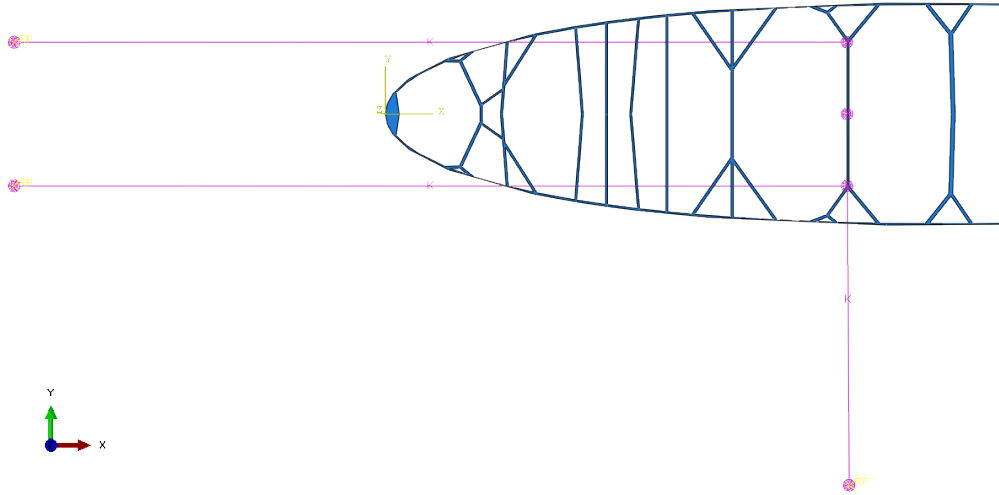


Figure 3.9. Figure of model with spring and pitch springs attached

Finally, the flap spring, restraining rotation in the z direction, or the 6 degree of freedom, between the wing body and the flap, is defined. The dynamic simulations planned require this spring to be both linear and nonlinear. The nonlinearity in

the spring is a model representation of freeplay in the control surface. The freeplay discussed can be seen in Figure 3.11, represented by the 2δ gap in displacement without any force response. From Conner's paper, δ is known to be 2.12° . To be able to represent the spring with and without freeplay, two axial connector elements are utilized. The axial connectors are set a known distance from the point of rotation, resulting in a couple. The connector elements can be seen in Figure 3.10.

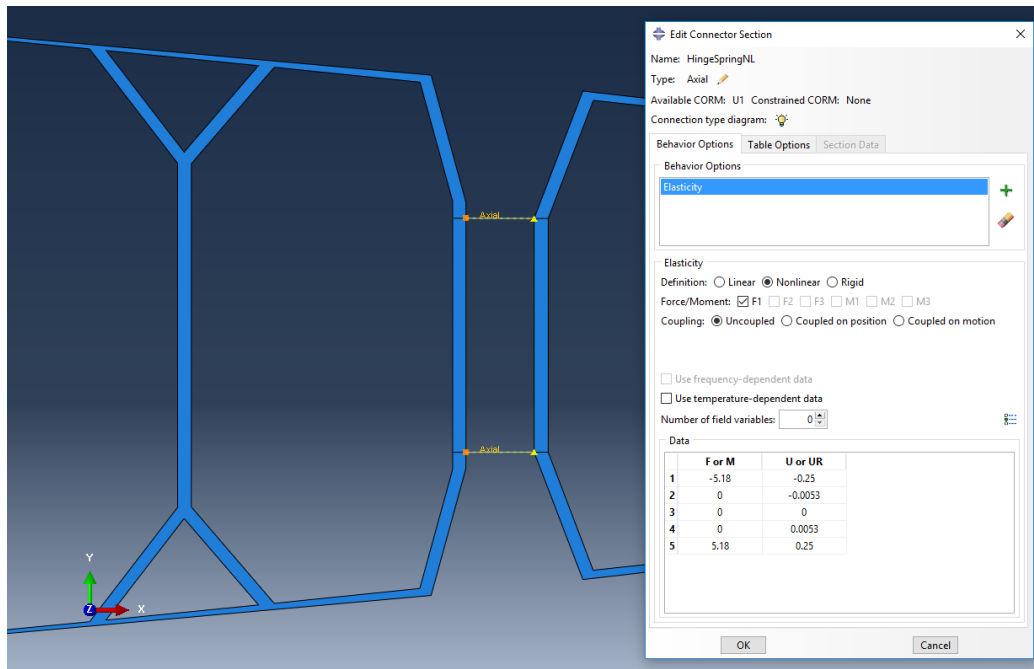


Figure 3.10. Figure of spring interaction between wing body and flap

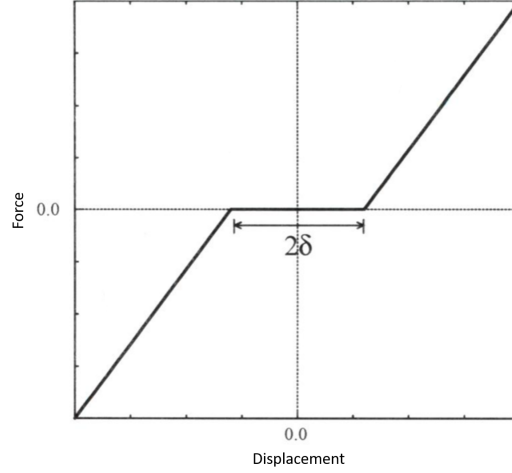


Figure 3.11. Figure of freeplay; gap in force response when displacements are present (Modified from [1])

3.5.5 Frequency Analysis

When analyzing the system for chaos, the Power Spectral Density (PSD) of the system will be calculated from the displacement vs time plot. The peaks of the PSD represent the natural frequencies of the system. If the system is nonlinear, these peaks will be less defined meaning the calculating of the PSD did not work. To know where to look for these peaks, an eigenvalue frequency analysis of the system is performed within Abaqus. Abaqus Standard utilizes the Lanczos eigensolver, represented by Equation 3.12, to solve for the eigenvalues, θ , and the eigenvectors, Φ [12].

$$[M]([K] - \sigma[M]^{-1}[M][\Phi] = \theta[M][\Phi] \quad (3.12)$$

3.5.6 Dynamic Simulation

To calculate the PSD, phase space plot, and the lyapunov exponent, vectors of time, displacement and velocity are required. Simulations are run utilizing Abaqus Standard from 0 to 10, 100 and 300 seconds and a time interval of 0.01 seconds. Abaqus Standard performs implicit dynamic analysis through direct integration [12].

The time interval will allow the PSD to be calculated up to 50 Hz. To make the connection between flutter and chaos, a sinusoidal force will excite the structure at the experimental flutter frequency, seen in Equation 3.13. The experimental flutter frequency used, ω_f , is 5.47 Hz. To test the dependence on the frequency, an arbitrary frequency of 0.8 Hz is also used to test for chaos. The amplitude of the force is -0.4 in the y direction and attached to either the aerodynamic center, Figure 3.12, or the tip of the flap, Figure 3.13. The amplitude utilized provides enough excitation to require feedback from all of the attached springs, but not so much that it causes the the flap to come in contact with the wing body.

$$f(t) = -0.4\sin(\omega_f t) \quad (3.13)$$

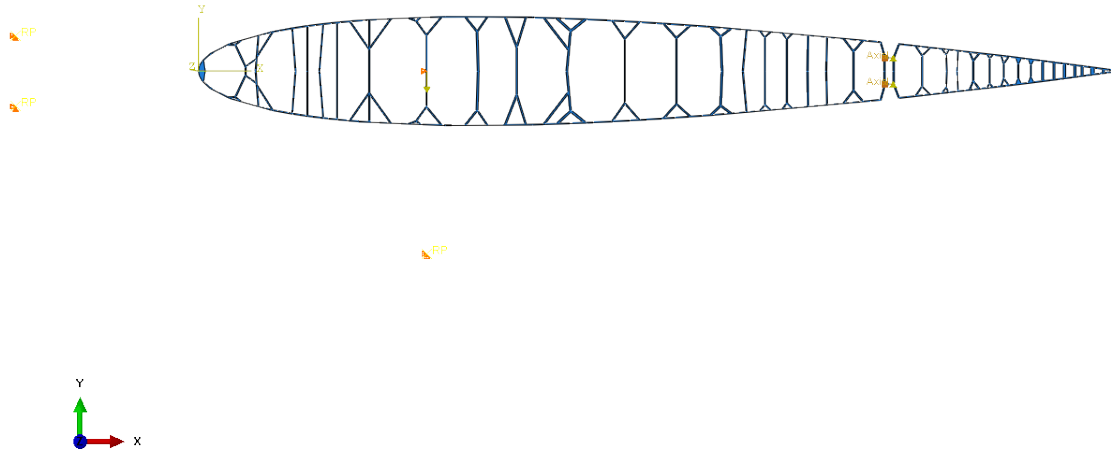


Figure 3.12. Figure of model with input force acting at the aerodynamic center

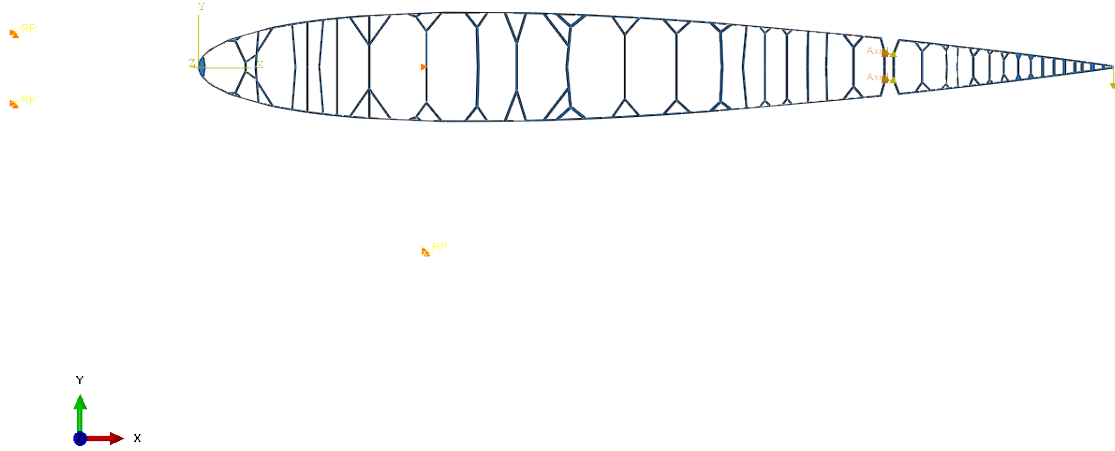


Figure 3.13. Figure of model with input force acting at the tip of the flap

3.5.7 Initial Conditions

When investigating for chaos, sensitivity to initial conditions must be explored. To do this, the flap instance in the assembly is rotated at the hinge node either up or down 1° , resulting in a different initial condition for each simulation. When the instance is rotated, the flap springs must be redefined, because the post rotation position is the new zero force, zero displacement point. Table 3.4 shows the summary of the differences between all 18 simulations.

After each simulation completes, the displacement and velocity, at the tip of the flap, in the 2 direction is written to separate .dat files and loaded into Matlab for post processing. The PSD is calculated, the phase space is plotted, and the Lyapunov exponent is calculated using techniques discussed in Chapter 2. Both codes utilized to calculate the PSD and the Lyapunov exponent can be seen in Appendix C.

Table 3.4. List of all differing simulations

Force	β IC (deg)	Frequency (Hz)	Freeplay (deg)	Total Time
Aero Center	-1	5.47	2.12°	10
Aero Center	0	5.47	2.12°	10
Aero Center	1	5.47	2.12°	10
Aero Center	-1	5.47	0°	10
Aero Center	0	5.47	0°	10
Aero Center	1	5.47	0°	10
Aero Center	0	0.8	2.12°	10
Aero Center	0	0.8	0°	10
Aero Center	0	5.47	2.12°	100
Aero Center	0	5.47	0°	100
Aero Center	0	5.47	2.12°	300
Aero Center	0	5.47	0°	300
Tip of Flap	-1	5.47	2.12°	10
Tip of Flap	0	5.47	2.12°	10
Tip of Flap	1	5.47	2.12°	10
Tip of Flap	-1	5.47	0°	10
Tip of Flap	0	5.47	0°	10
Tip of Flap	1	5.47	0°	10

3.6 Summary

This chapter presented the procedures followed to dynamically analyze the NACA airfoil system to investigate a connection between flutter and chaos. Initial conditions to optimize an airfoil structure were outlined. An optimization program was created to design the internal structure. The internal structure output was then investigated for changes to the aerodynamic properties via CFD. Finally, the dynamic FEM was created and setup to study the aeroelastic system in question for chaotic tendencies.

IV. Results and Analysis

4.1 Chapter Overview

Chapter 3 discussed the procedures to obtain an optimized internal structure to an airfoil modeled after the experimental model outlined in Conner’s paper, any changes to the aerodynamic properties after the optimization, and analyze the dynamics of the aeroelastic system via Finite Element Analysis (FEA). The results obtained from these optimizations and simulations are discussed in this chapter.

4.2 Initial Loading Conditions

Utilizing XFOIL’s viscous solution technique outlined in Chapter 3, the coefficient of pressure across the upper and lower surfaces of the National Advisory Committee for Aeronautics (NACA) 0012 are determined for the cruise conditions; $Re = 1.1 \times 10^6$ and $Mach = 0.1885$. How the coefficient of pressure varies along the chord of the airfoil can be seen in Figure 4.1.

It can be seen that the upper and lower surfaces have the same pressure coefficients over the length of the chord. This is because a symmetric airfoil does not create a pressure difference at zero angle of attack; therefore, not generating lift. These coefficients are converted to pressure as outlined in Chapter 3 to act as the forces for the optimization of the internal structure in the next section.

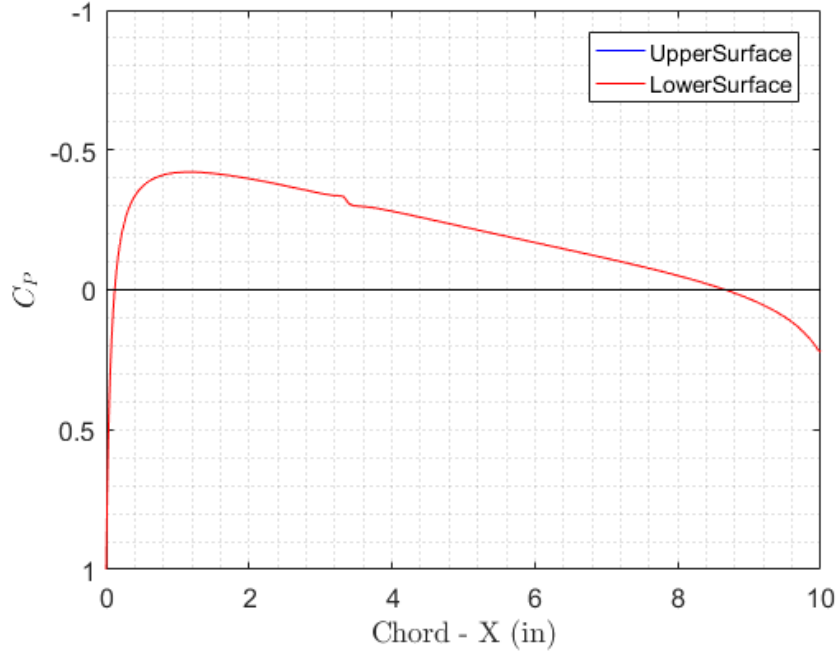


Figure 4.1. Plot of C_p over the chord length output from XFOIL

4.3 Topology Optimization

The first step in creating a dynamic Finite Element Model (FEM) was generating a realistic representative internal structure of an airfoil, assuming it would be the average of the internal structure over the span of a wing. Utilizing specifications from Conner's aeroelastic experimental model, an internal structure per unit span was optimized [1]. Two discretization of the airfoil were performed; the first resulting in 42336 elements and the second resulting in 45584 elements. The maximum displacement after optimization, due to the initial conditions shown above, for each mesh was 9.1×10^{-5} and 8.5×10^{-5} respectively. The change in displacement was only 6.1% of the first displacement; therefore, the second mesh was deemed to be converged and ready for optimization.

4.3.1 Constraining Center of Gravity

Because no previous literature could be found on this topic, multiple iterations of optimizing the internal structure were performed. A study was performed by changing the constraint of the center of gravity, because the effects of which were unknown. Also, after the first optimization, the center of gravity was 0.3 inches too forward. Instead of moving beams and mass around when transitioning to Solidworks and Abaqus, multiple optimizations were performed to test the change to the structure when accounting for the center of gravity. Table 4.1 shows how gradually constraining the center of gravity changed the outcome of the optimization.

Table 4.1. Table of optimization results

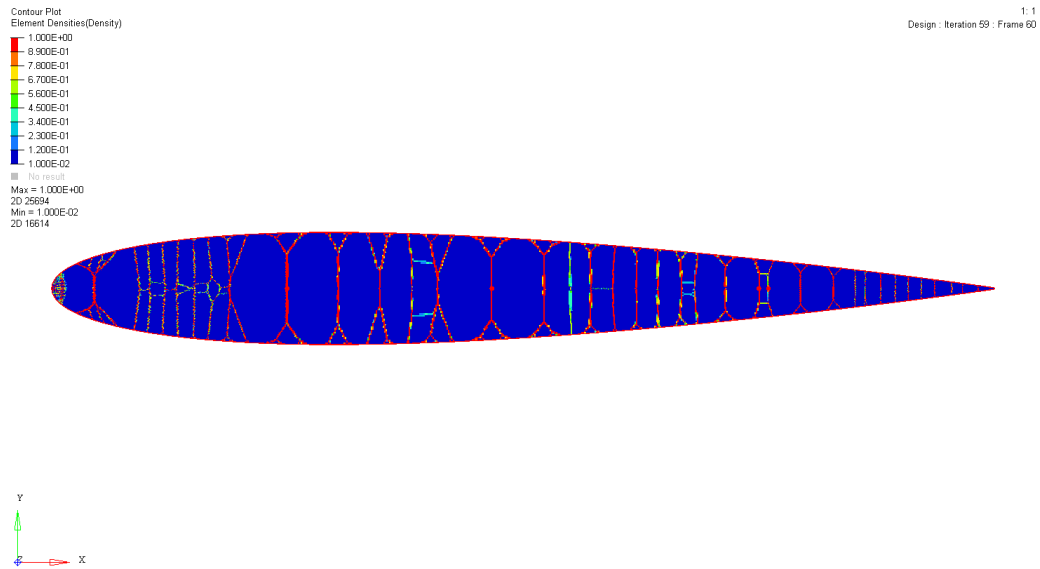
Constraint	No Constraint	4.6 - 4.7 (in)	4.65 - 4.69 (in)	4.67 - 4.69 (in)
CG (x)	4.302	4.601	4.652	4.670
Compliance	0.00211475	0.0023218	0.00365629	0.00216434
VF (%)	8.57	8.57	8.57	8.57
Iterations	59	64	56	79

As the center of gravity is forced aft by the constraint, the compliance of the structure increases, meaning the stiffness of the structure decreases. Constraining the center of gravity between 4.6 and 4.7 inches of the chord decreases the stiffness of the structure by 9.8% of the original stiffness. Forcing the center of gravity further aft, between 4.65 and 4.69 inches, decreases the stiffness by 72.9% of the original stiffness. Finally, forcing the center of gravity to be at the same location as the aeroelastic model decreases the stiffness only 2.3% of the original stiffness.

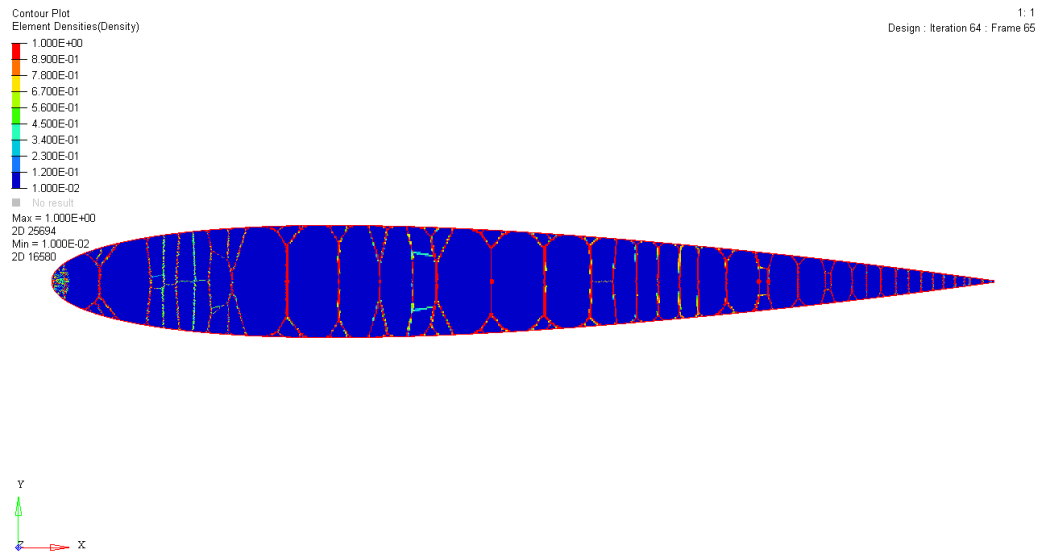
Not constraining the center of gravity results in the stiffest structure, while constraining the center of gravity between 4.65 and 4.69 inches results in the weakest structure. Oddly, constraining the center of gravity between 4.67 and 4.69 results in the second stiffest structure. Each time the center of gravity is forced aft, the optimization program attempts to reach the no constraint location of 4.302 inches, but

is stopped by the lower bound of the constraint. It appears that if the optimization program would have placed the center of gravity at 4.67 inches, it would have better fulfilled the optimization objective. The program gradually added mass to connect the skin to the no design points in the mesh; therefore, the increased stiffness from placing the center of gravity at 4.67 inches, rather than 4.6 or 4.65 inches, is due to a main structural beam forming from that no design point.

From each optimization came not only the specifications listed above, but also structural outputs comprised of element density plots. Figures 4.2 and 4.3 show each constraints different internal structure output. Figure 4.2a shows the structure when the center of gravity is not constrained. It can be seen that most of the beams are vertical, with some spreading out to a ‘y’ shape as they near the skin. This ‘y’ shape helps in creating a structure that can withstand a larger force without buckling, because there is a smaller unsupported length. As the center of gravity is forced aft to 4.6 inches, more beams are formed in the tail section of the airfoil, while less beams are formed in the forward section of the wing body.

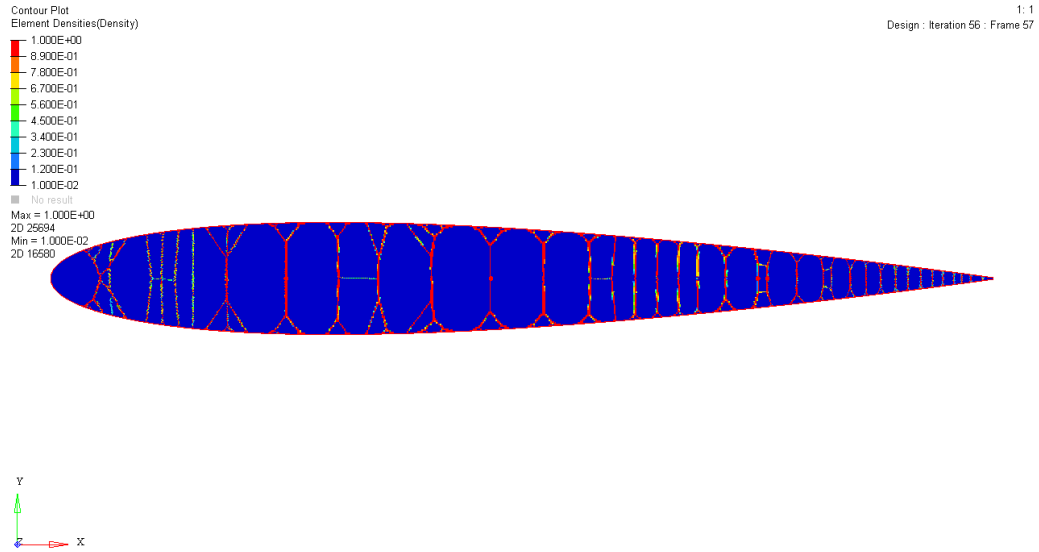


(a) No Constraint

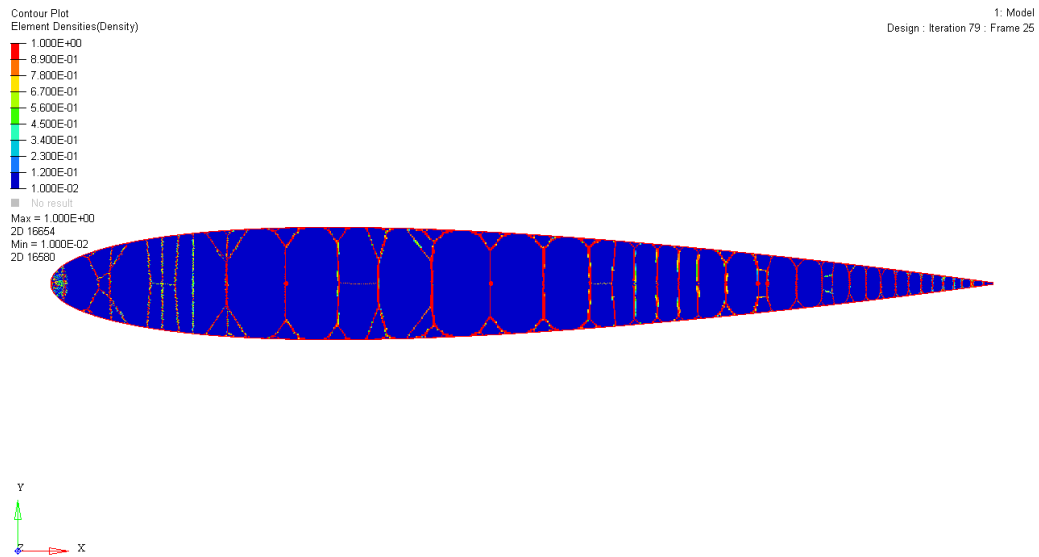


(b) 4.6 - 4.7 inches

Figure 4.2. Element density outputs from Optistruct with different center of gravity constraints



(a) 4.65 - 4.69 inches



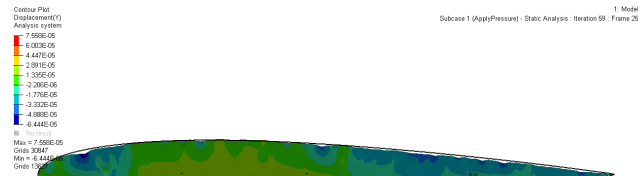
(b) 4.67 - 4.69 inches

Figure 4.3. Element density outputs from Optistruct with different center of gravity constraints

Pushing the center of gravity further aft to 4.65 inches results in the most drastic changes of any of the optimizations. Figure 4.3a shows the lack of structure in the nose of the airfoil and the combining of multiple beams into one, as compared to Figures 4.2a and 4.2b. These two major changes from the other structures explains why this is the weakest of all the structures optimized. Finally, forcing the center of gravity to 4.67 inches culminates in the structure seen in Figure 4.3b. Unlike the previous optimization, structure is present in the nose of the airfoil and nonessential beams are removed. It should be noted that the mass of each of the structures is the same.

After each optimization was complete, deformation of the final structure is present. Figure 4.4 shows how each center of gravity constraint changes the deformation of the final structure. The deformation within the Figures is scaled by 1000 to better show the deformation. The deformation is inversely proportional to the compliance of each structure. That is to say, the stiffness varies directly with the deformation of the structure. Figure 4.4b shows that moving the center of gravity to 4.6 inches from 4.3 inches results in deformation of the nose of the airfoil. Figure 4.4c reveals that moving the center of gravity another 0.05 inches the nose deforms further from the same forces. The scaled deformation results in the folding over of the nose. The final center of gravity constraint results in a deformation plot of Figure 4.4d. The scaled deformation of the nose is severely decrease, resembling that of the no constraint structure.

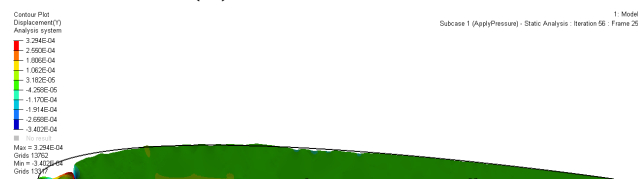
This further substantiates the need for the center of gravity constraint lower bound to be set where the center of gravity is required. If the no constraint structure was modified and utilized for further investigations, The resulting structure may have led the lack of stiffness present in Figures 4.3a and 4.4c.



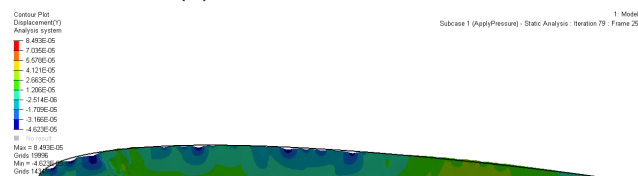
(a) No Constraint



(b) 4.6 - 4.7 inches



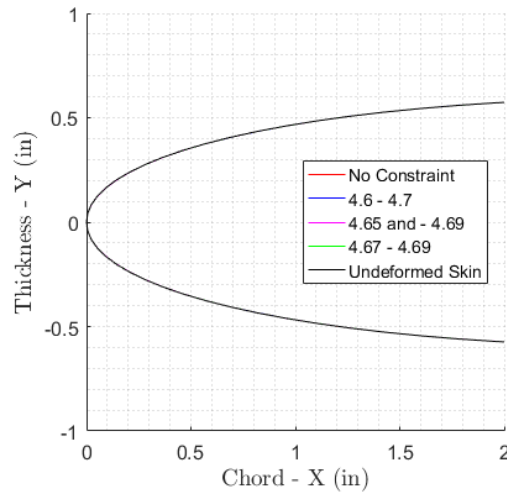
(c) 4.65 - 4.69 inches



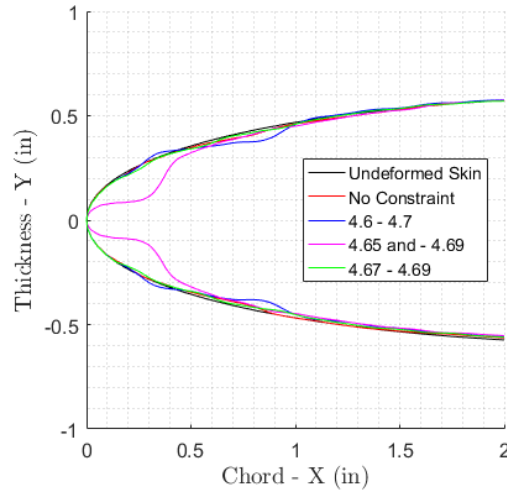
(d) 4.67-4.69 inches

Figure 4.4. Deformed structure plots (scale = 1000) with undeformed edge for different center of gravity constraints

From investigating the deformation of the final structures, it can be seen that majority of the deformation is present within the first two inches of the airfoil. Figure 4.5 zoom in on this section and show both the unscaled and scaled by 300 skin deformations. By not scaling the deformation, minimal differences from the undeformed skin can be seen. However, when the deformations are scaled by 300, a better representation of the deformation can be seen and require that any changes in the resulting pressures acting on the unscaled, deformed airfoil be investigated.



(a) Scale = 1



(b) Scale = 300

Figure 4.5. Deformed skin plots of all optimizations

4.3.2 Final Optimized Internal Structure

From Table 4.1 and Figures 4.3b and 4.5b it can be seen that the structure with the center of gravity at 4.67 inches, matching the aeroelastic model in Conner's paper, and resulting in the second stiffest of the structures will be used from this point forward in the investigation. A closer picture of the structure can be seen in Figure 4.6, while the resulting internal stress after optimization can be seen in Figure 4.7. The maximum deformation after optimization for this structure is 8.5×10^{-5} inches, which occurs 2.784 inches from the leading edge. While this is a very small deformation with respect to the skin thickness of 0.01 inches, many deformations exist on this order of magnitude all along the length of the chord and they can effect the lift force and pitch moment acting on the airfoil. The maximum stress within the airfoil structure is 1,255 psi, which is well below aluminum's yield stress of 35,000 psi. Therefore, the structure is deemed stiff enough to continue, but needs to be investigated for changes in pressure due to deformation of the skin.

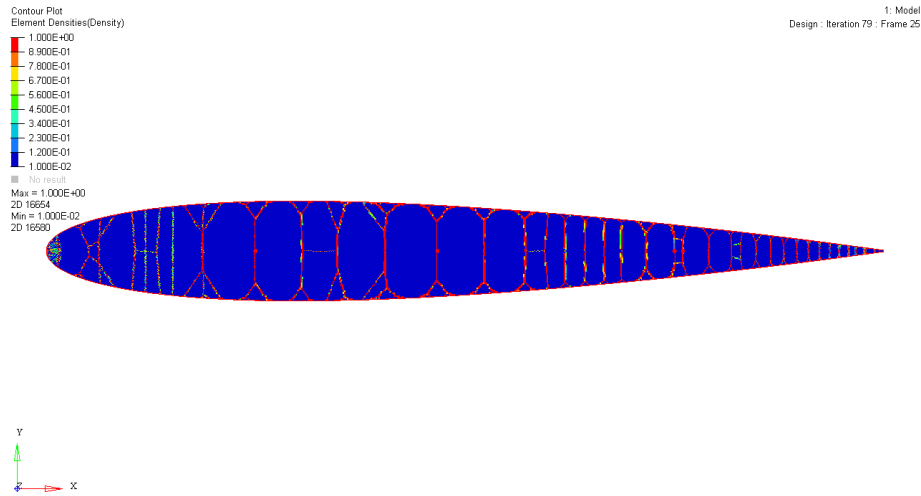


Figure 4.6. Element density plot of the final optimized structure

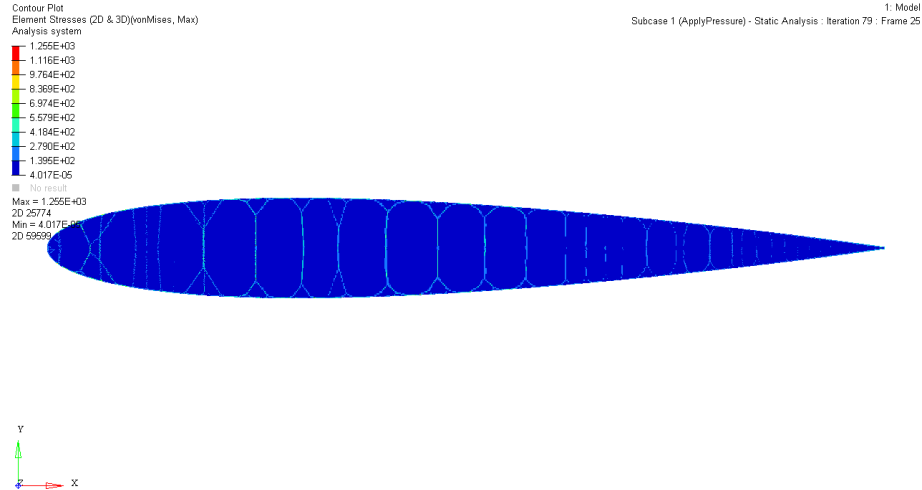


Figure 4.7. vonMises stress within the final optimized structure

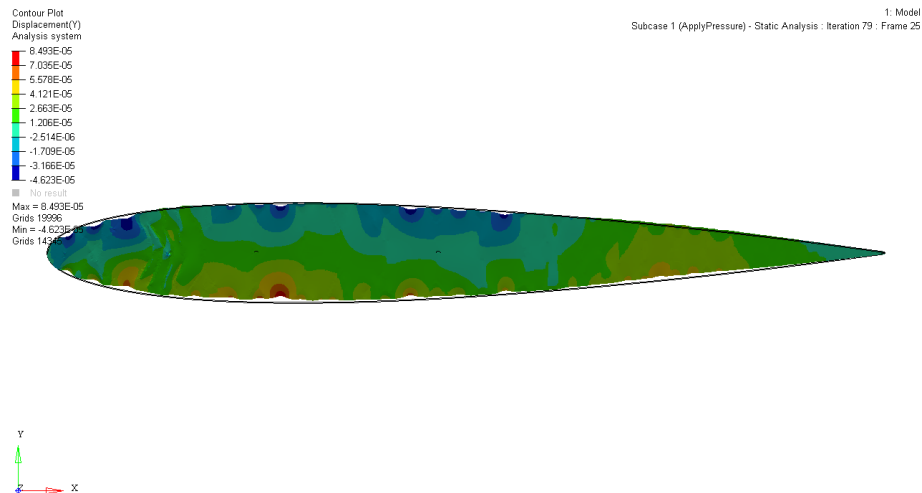


Figure 4.8. Displacement of the final optimized internal structure scaled by 1000

4.3.3 Beam Buckling

The maximum stress discussed earlier exists near the boundary conditions. The highest stress seen in the beams of the internal structure is 560 psi. The question is will the vertical web buckle under the load created from the pressure arises. From the transition to Solidworks, the smallest thickness of the beams is known to be 0.011 inches, with unit depth. The longest free standing length of the high stress beams is 0.86 inches. Using Equation 4.1, the resulting worst case scenario buckling force is

14.8 lbs [21]. Dividing this force by the worst case area results in a buckling stress of 1,345.6 psi.

$$P_{cr} = \frac{\pi^2 EI_{min}}{L_{max}^2} \quad (4.1)$$

The resulting stress from the buckling force is the minimum stress that would be present at the moment buckling begins. Because the highest stress seen in any beam of the internal structure is less than half of the buckling stress, buckling of the internal structure is not a concern in this research.

4.4 Fluid Structure Interaction

As stated previously, the deformed skin may cause a change in flow over the airfoil, in turn causing a change in the lift force and pitching moment. Chapter 3 outlined the procedure to calculate the pressure acting on the deformed and undeformed airfoil at the flutter velocity of 20.6 meters per second using ANSYS Fluent.

4.4.1 Mesh and Model Convergence

Just as in FEA, Computational Fluid Dynamics (CFD) requires the convergence of the model; however, the convergence depends on the grid, as well as the equations used to model Navier-Stokes. As outlined in Chapter 3, three test unstructured test grids are built within Pointwise. They are labeled as coarse, medium, and fine. The coarse mesh is built by placing 1636 equally spaced points on the skin of the airfoil and extending the flow field five times the chord length from the airfoil in all directions. The number of cells for the coarse grid totals to 34,743. The coarse grid is only solved using the $k-\omega$ turbulence model from ANSYS Fluent, just to ensure a solution can be reached. The coarse grid solves in 1000 iterations with all residuals (continuity, x velocity, y velocity, k, and ω) reaching values below 1×10^{-6} .

Next, the medium grid is produced by doubling the number of points used to define the airfoil, yielding 3272 points on the airfoil. The same dimensions for the flow field are utilized totaling 232,148 cells in the flow field. Both the $k-\omega$ and $k-\epsilon$ turbulence models are used to converge to a solution with the same minimum residual values as the coarse grid. The $k-\omega$ turbulence model converged in 1261 iterations and the $k-\epsilon$ turbulence model converged in 1038 iterations. Continuing with convergence, the fine grid is generated by doubling the number of defining points for the airfoil from the medium mesh, resulting in 6344 points on the airfoil and 382,076 cells in the flow field. Both the $k-\omega$ and $k-\epsilon$ turbulence models were used to the fine grid, converging all residuals in 1647 and 1125 iterations respectively. The coefficient of pressure results from each grid can be seen in Figures 4.9 and 4.10.

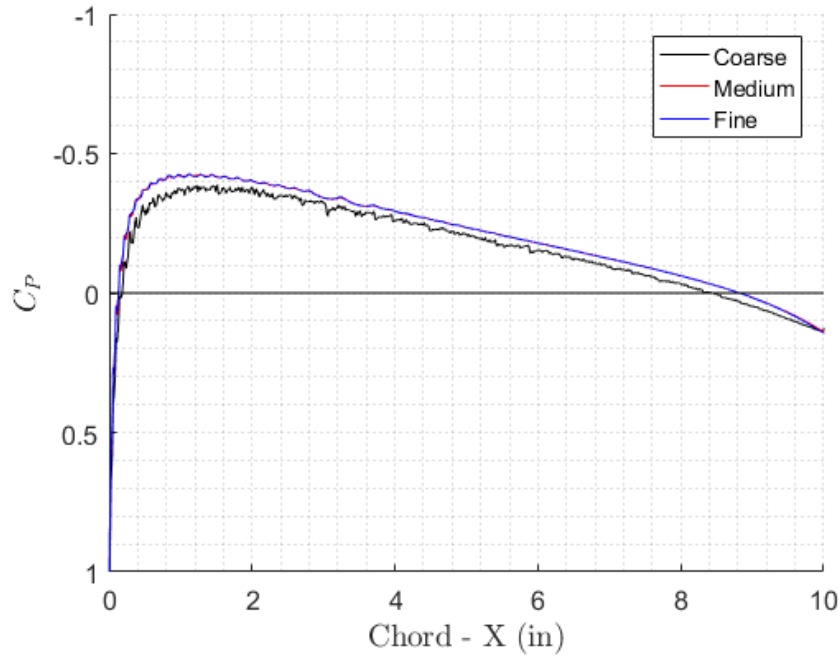


Figure 4.9. Comparison of the coarse, medium, and fine grids using the $k-\omega$ turbulence model

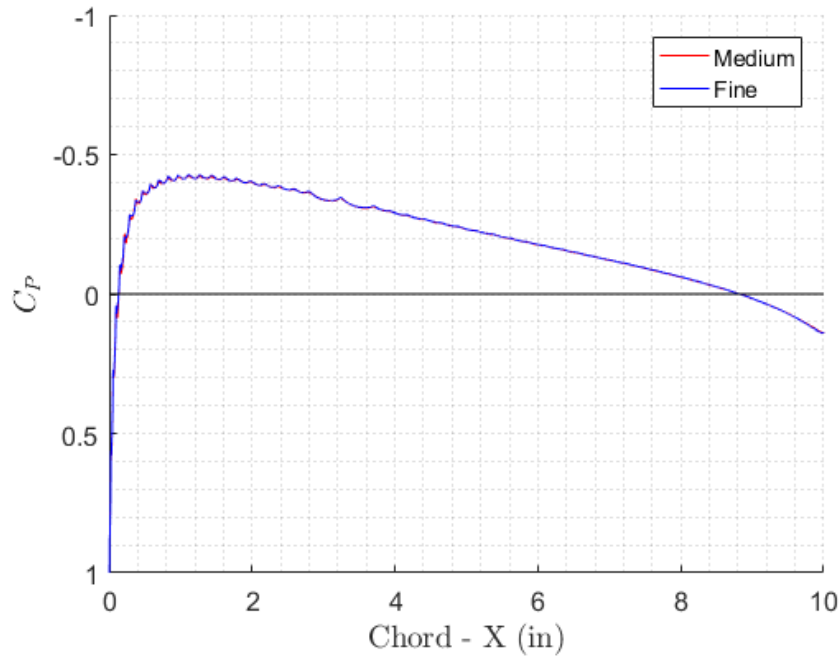


Figure 4.10. Comparison of the medium, and fine grids using the $k-\epsilon$ turbulence model

It can be seen in Figure 4.9 that the coarse mesh is not refined enough to converge with the other two grids for the $k-\epsilon$ model. The average difference from the medium to coarse mesh is 0.0375, while the average difference between the medium and fine grids are 0.0003. In Figure 4.10 the medium and fine grids are compared using the $k-\omega$ turbulence model. The average difference between the two is 0.0011. Therefore, it is deemed that the fine mesh is converged enough to compare the two turbulence models. Figure 4.11 compares the two turbulence models when both discretized via the fine grid. The difference between the two models is 0.0018. The $k-\omega$ model appears to have smoother peaks throughout the plot, and Cummings shows that the $k-\omega$ model is preferred when performing CFD on the NACA 0012 airfoil [6].

With the $k-\omega$ turbulence model chosen, a final, very refined mesh is generated to test both the deformed and undeformed airfoil coefficient of pressure values generated. For this grid, 10,000 points are used to define the shape of the airfoil, and the flow field is extended to 15 times the chord length in all directions. The resulting grid

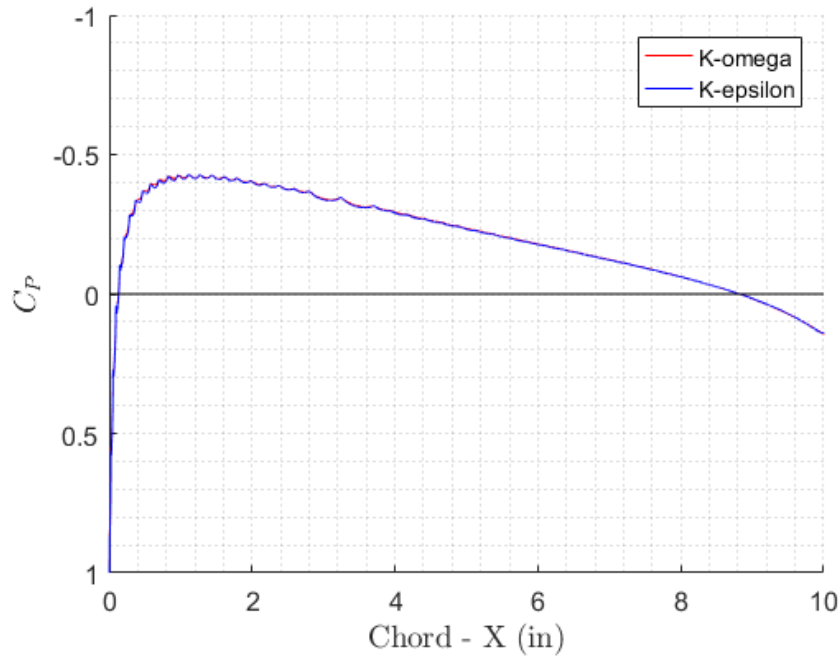


Figure 4.11. Comparison of the $k-\epsilon$ and $k-\omega$ turbulence models for the fine mesh

totals 512,000 cells and can be seen in Figures 4.12 and 4.13.

4.4.2 Skin Deformation Effects

Figure 4.14 shows the coefficient of pressure values acting on both the deformed, seen in Figure 4.8, and undeformed NACA 0012 airfoils. From looking at the plot, no visible difference can be seen in the results from the deformed to undeformed airfoil; however, the small displacements of the skin have resulted in small changes in the pressure over the airfoil.

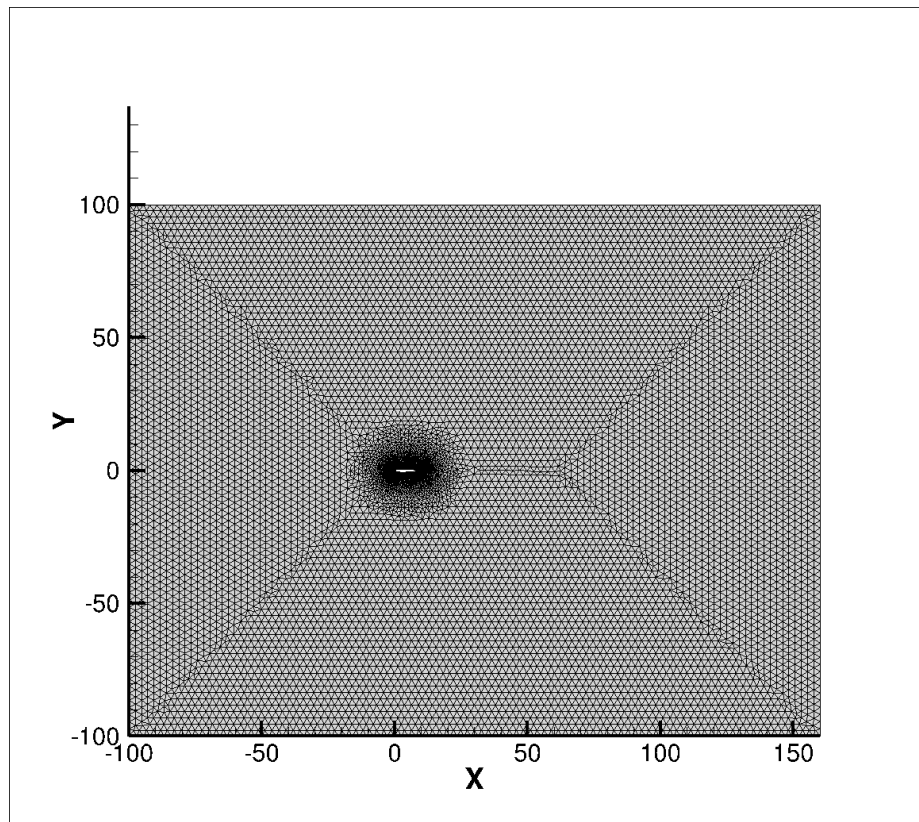


Figure 4.12. Total flow field for very fine grid

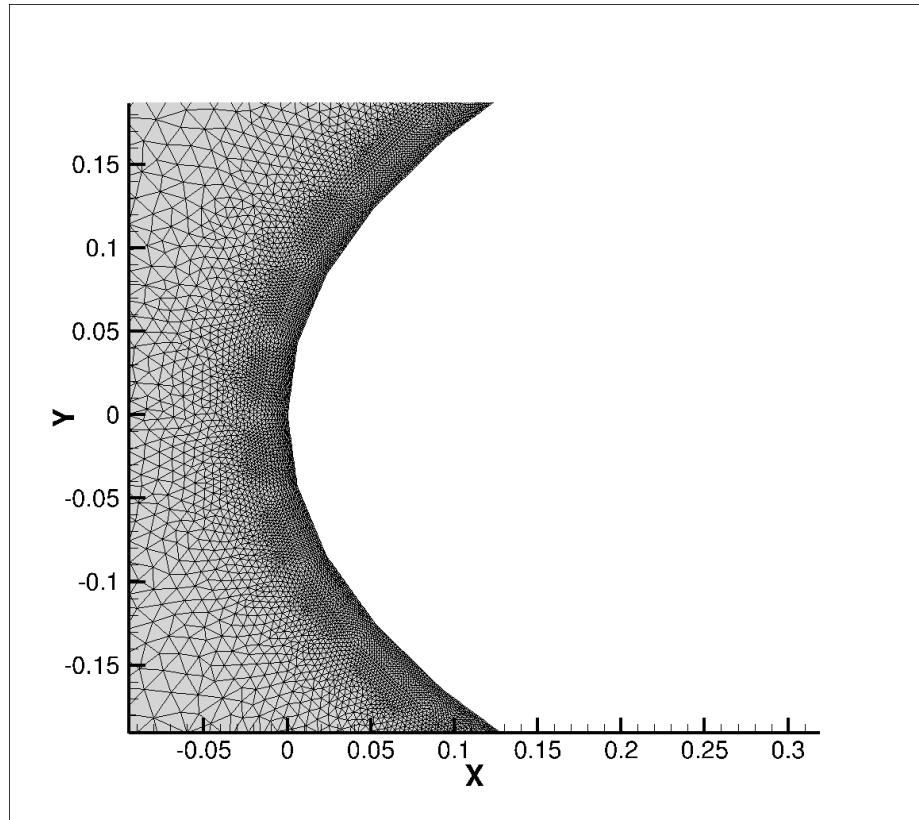


Figure 4.13. Stagnation region of very fine grid

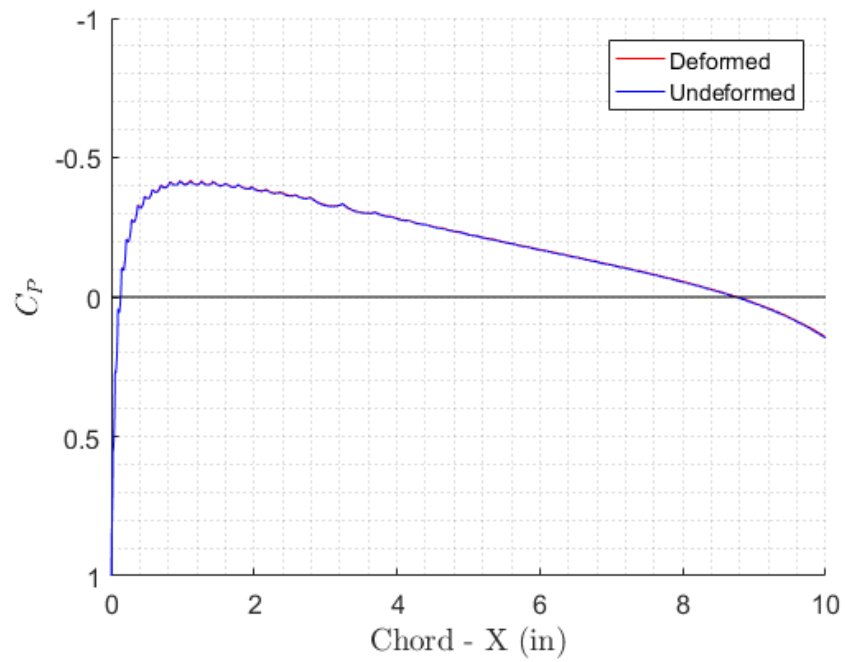


Figure 4.14. Plot of C_p over the chord length for the deformed and undeformed airfoils

The deformed airfoil has an average C_p value this is 0.0017 less than that of the undeformed airfoil. This average is the same for both the upper and lower surface. This difference in pressure comes from the interaction between the flow and the small deformations on the skin. Along with the overall change in pressure over each surface, a difference in pressure from the upper and lower surface is present. This difference results in the coefficient of lift, C_l changing from 1.162×10^{-5} for the undeformed airfoil to 1.622×10^{-5} for the undeformed airfoil. While the change seems small, it is a 39.5% change resulting from the average deformation along the span of the wing, not from the bending or torsion of the wing.

Seen in all the CFD plots is that the C_p out appears to be jagged in nature. This is due to the physical nature of the airfoil imported into pointwise. Inspecting the velocity within the boundary layer in Figures 4.15 and 4.16, it can be seen that the edge of the boundary layer remains a smooth curve, while the surface of the airfoil is very piecewise, causing the boundary layer to change in size. The effect of the piecewise nature of the skin can also be seen in a plot of the $y+$ value along the chord of the airfoil. The $y+$ value is the boundary layer viscous grid height, and while it is mostly kept below a value of 1 (a good modeling technique [6]), the value has the same jagged nature of the C_p values. The cause of the boundary layer thickness changes and the $y+$ value changes is the pointcloud used to load the deformed and undeformed airfoil into pointwise.

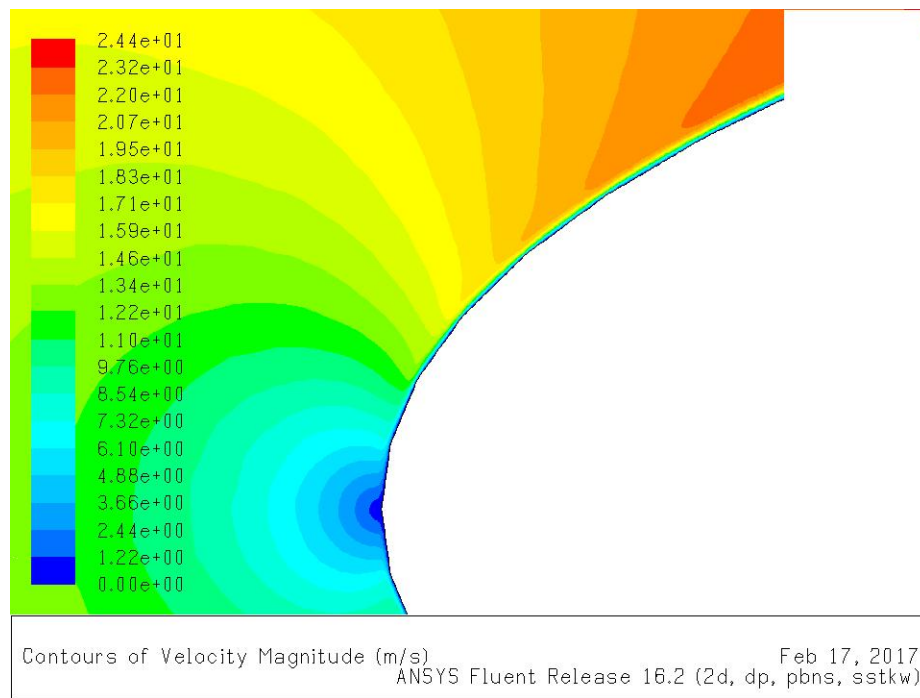


Figure 4.15. Contour of velocity magnitude at the nose of the airfoil

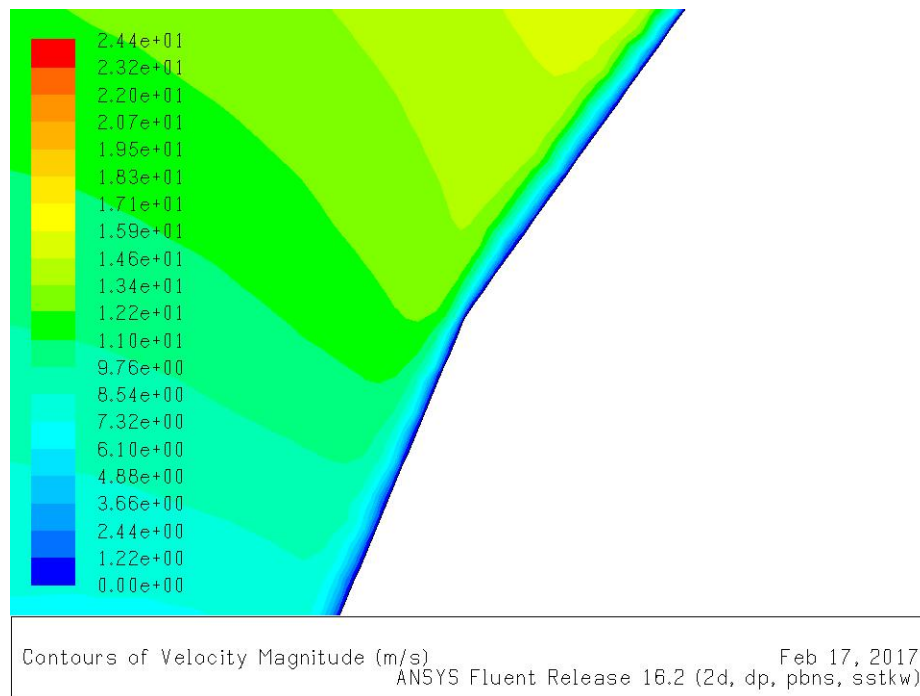


Figure 4.16. Contour of velocity magnitude zoomed in to show boundary layer effects

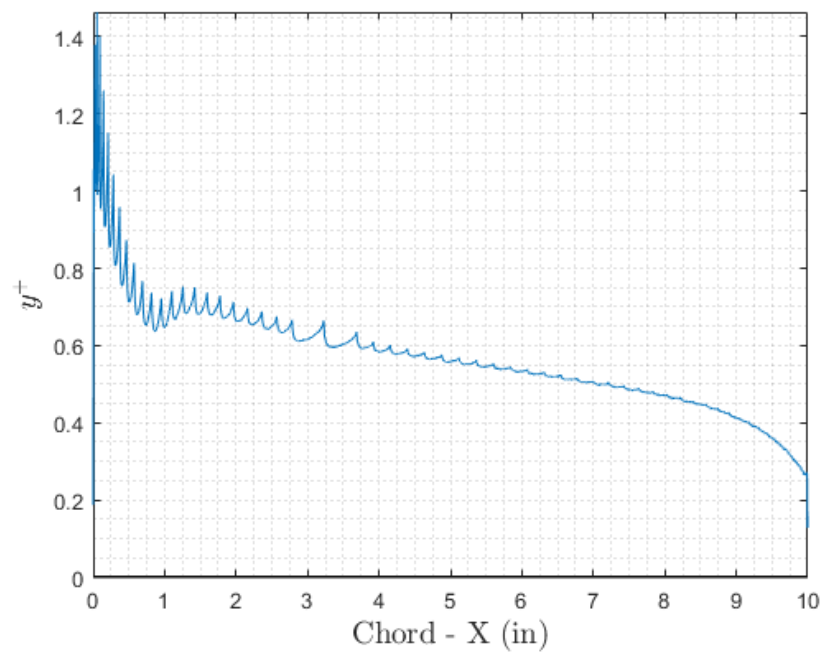


Figure 4.17. Plot of y^+ value over the surface of the airfoil

4.5 Dynamic Finite Element Model

After the changes to the flow as a result of optimizing an internal structure are quantified as negligible, the dynamic FEM is developed from the undeformed skin with the optimized internal structure. Just as with the optimization FEM, only 2 discretizations of the model were performed. The first resulted in a mesh of 24122 elements and the second utilized 25334 elements. Because no static analysis was performed, the frequency of the 4th mode (the first mostly structural mode) was compared. The coarse mesh calculated a value of 16.027 Hz, while the fine mesh calculated a value of 15.113 Hz. That is only a 5.7% difference from the coarse mesh value; therefore, the fine mesh is deemed converged. With the mesh converged, the springs, and the interactions defined, the model is ready to be dynamically analyzed.

4.5.1 Frequency Analysis

To determine the location of the power peaks in the Power Spectral Density (PSD), the natural frequencies of the model are determined. Table 4.2 presents the natural frequencies of the model up to 50 Hz.

Table 4.2. Modal frequencies

Mode	Frequency (Hz)	Mode	Frequency (Hz)
1	0.37657	5	18.693
2	0.98170	6	26.088
3	2.1966	7	38.013
4	15.113	8	48.273

For each natural frequency, a unique mode shape exists. The first four modes can be seen in Figure 4.18, while the remaining four modes can be seen in Figure 4.19. Mode 1, seen in Figure 4.18a, is mostly dominated by the plunge spring, with some interaction from the pitch and flap springs. Mode 2, seen in Figure 4.18b, has motion

mostly defined by the flap spring, with some interaction from the other springs. Mode 3, seen in Figure 4.18c, is defined by the pitch and flap spring, with some interaction from the plunge spring and the structure. It should be noted that mode 3 is the first mode to incorporate any structural deformation, other than the springs that represent the span wise deformation of a wing. Mode 4 can be described as structural bending of the beam passing through the aerodynamic center. This mode is the first mode that is mostly an internal structural mode.

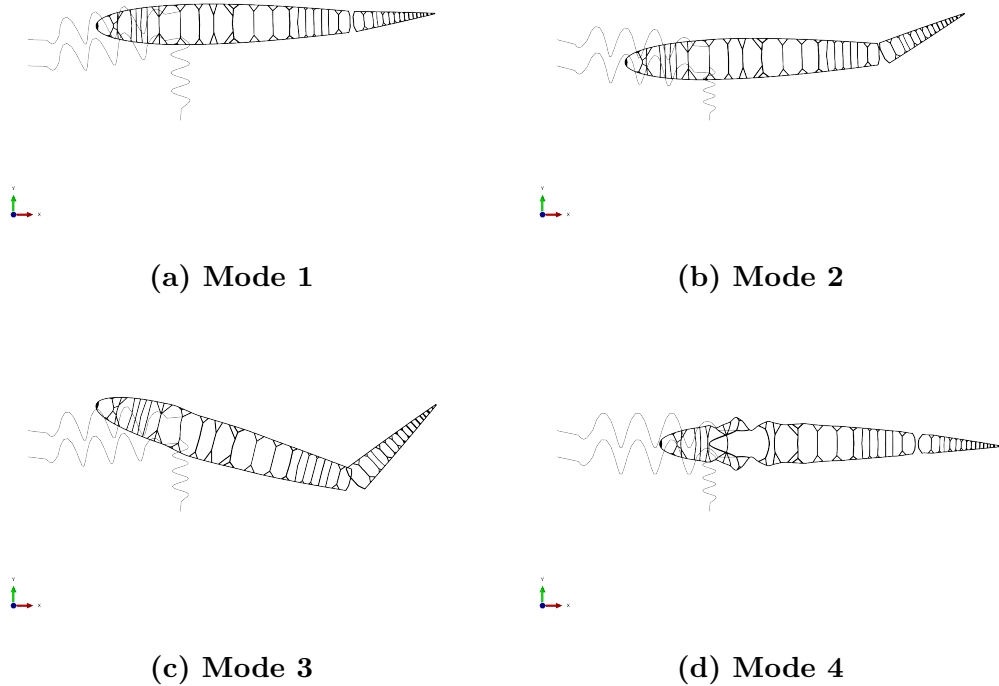


Figure 4.18. Mode shapes for modes 1 thru 4

Mode 5 can be described as the first bending mode of the internal structure. Mode 6 is the second bending mode, because the left side of the structure is down, while the right side is up and roughly, the mid-node is near stationary. The 7th mode can then be described as the third bending mode of the internal structure and the 8th

mode as the fourth bending mode.

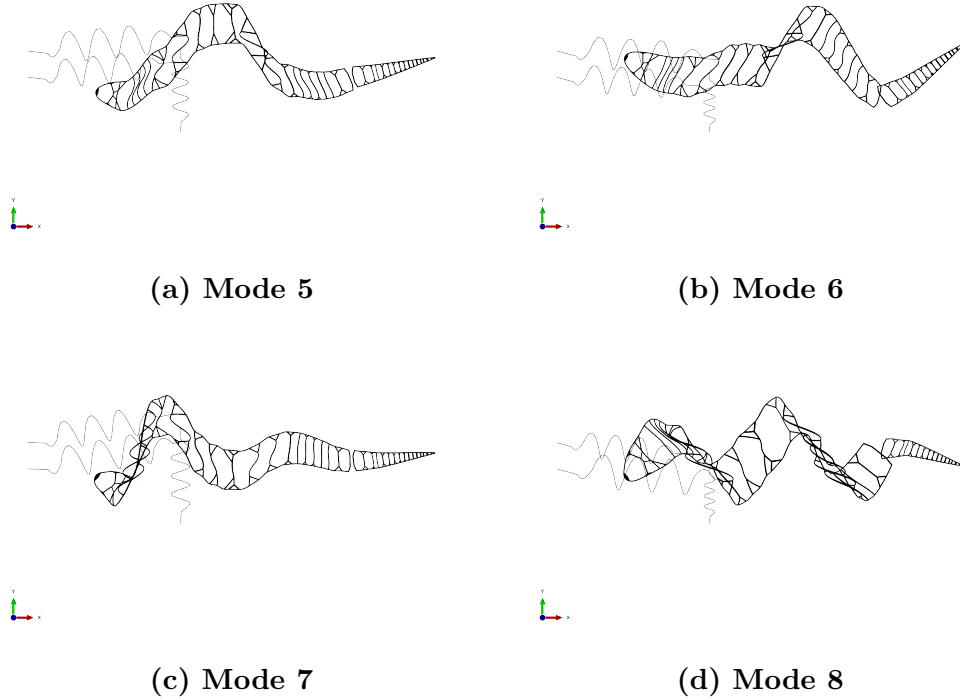


Figure 4.19. Mode shapes for modes 4 thru 8

Knowing and understanding the modes of the structure are crucial in any dynamics problem, because together they describe the motion of the structure. Forward, these modes will be referred to by their numbers and the natural frequency at which they occur.

4.5.2 Freeplay vs. No Freeplay

The first test of the system chaos deals with the introduction of the nonlinear flap spring. Without freeplay, this spring acts just as a linear spring, while with freeplay, there exists a ‘deadzone’ where no returning moment acts when a rotational change

of displacement occurs. Figures 4.20, 4.21, and 4.22 show the displacement over time when β is set to -1, 0, and 1, respectively, when the input force is placed at the aerodynamic center. They compare the displacement of the tip of the flap when δ is 0° and 2.12° .

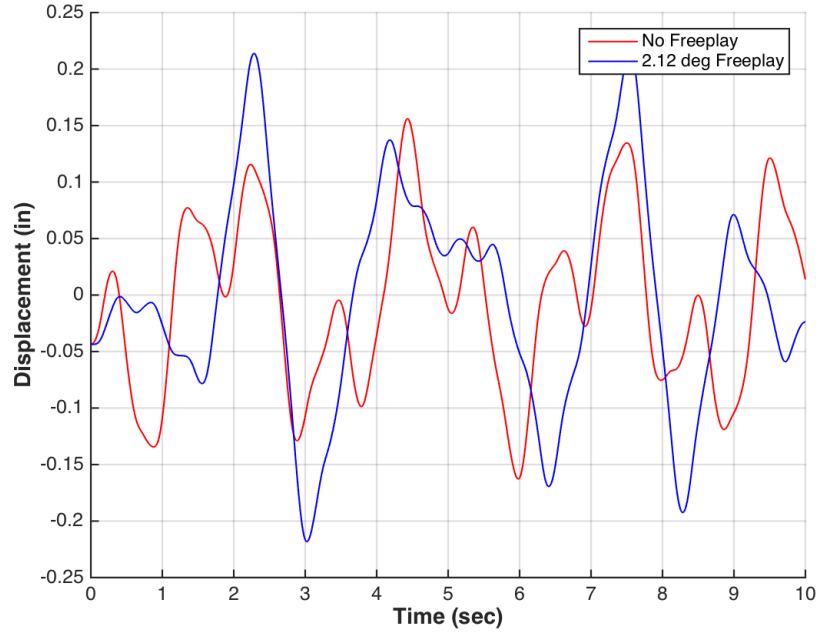


Figure 4.20. Displacement over time when $\beta = -1^\circ$ initially and the input force is located at the aerodynamic center

In Figure 4.20 it can be seen that both responses appear to be cyclical. However, with no freeplay, the frequency of oscillations of the flap is $\frac{3}{2}$ times that of the response with freeplay, but the peak amplitude is lower by 0.15 inches.

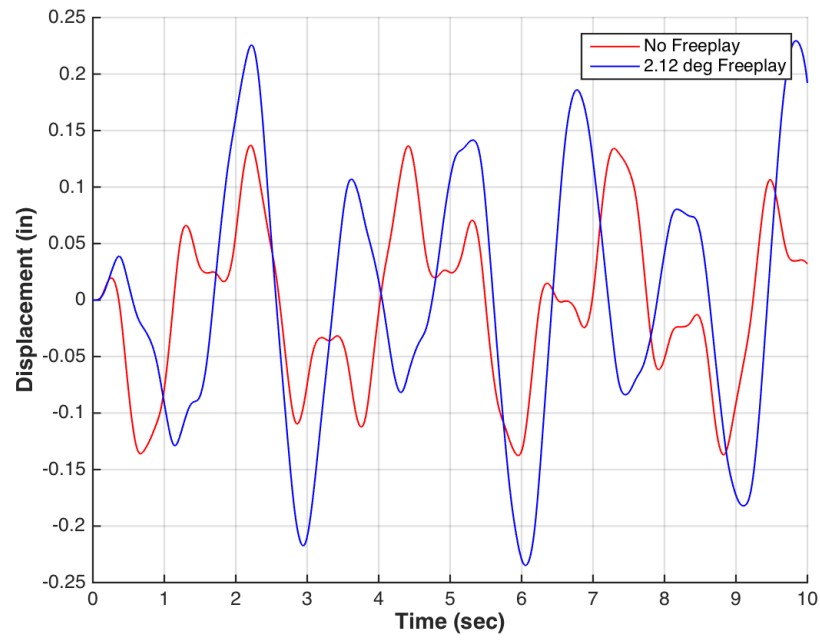


Figure 4.21. Displacement over time when $\beta = 0^\circ$ initially and the input force is located at the aerodynamic center

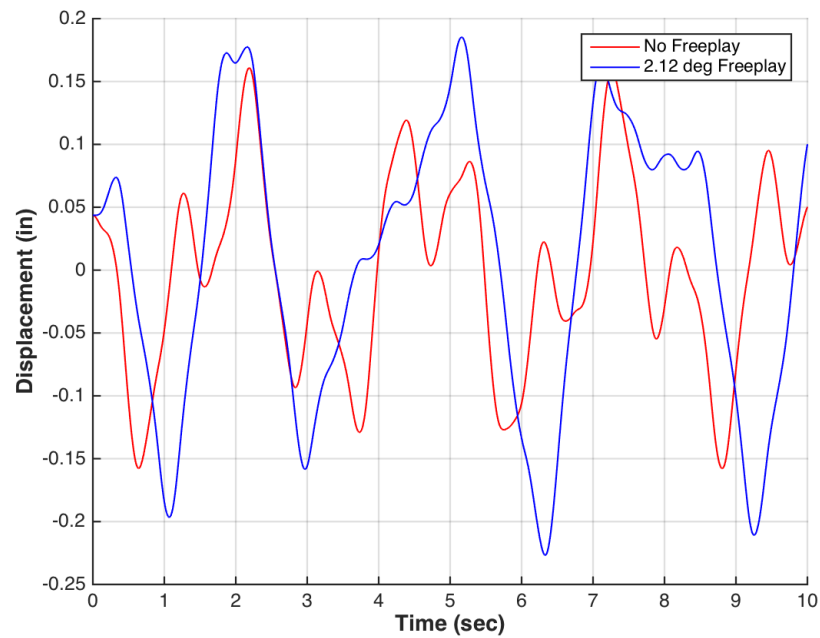


Figure 4.22. Displacement over time when $\beta = 1^\circ$ initially and the input force is located at the aerodynamic center

Figures 4.21 and 4.22 show the exact same fraction of oscillations when comparing the freeplay and no freeplay responses. When β is 0° initially, the average amplitude difference is greatest, but when β is 1° initially, the peak amplitude difference is greatest at 0.2 inches.

When the input force is changed from the aerodynamic center to the tip of the flap, the displacement changes to the plots seen in Figures 4.23, 4.24, and 4.25, for β equal to -1, 0, and 1, respectively. In all three figures, it can be seen that when freeplay is introduced, the frequency of the response is $\frac{7}{9}$ of the frequency without freeplay. Contrary to when the excitation force is placed at the aerodynamic center, there is not much difference in the peak amplitude from the two responses. In fact, they resemble each other greatly, because of the close proximity between the input force and the output displacement.

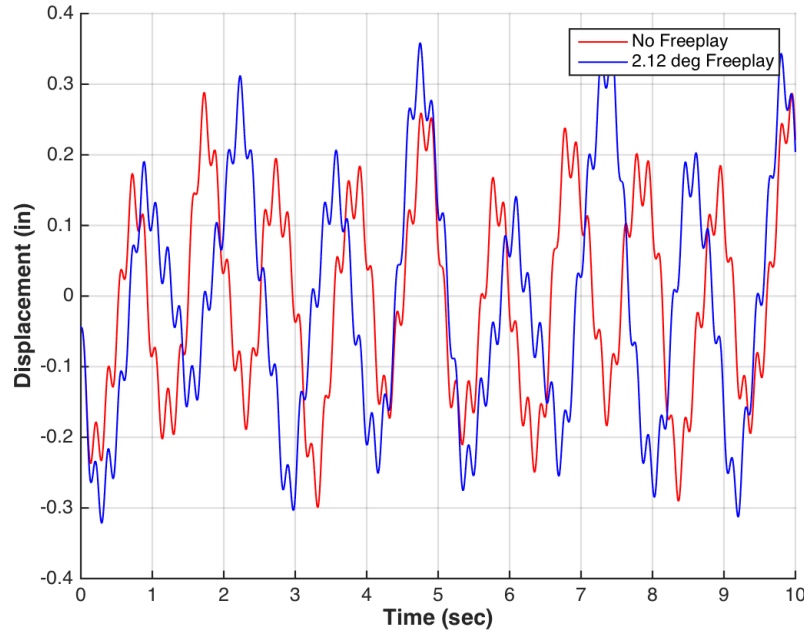


Figure 4.23. Displacement over time when $\beta = -1^\circ$ initially and the input force is located at the tip of the flap

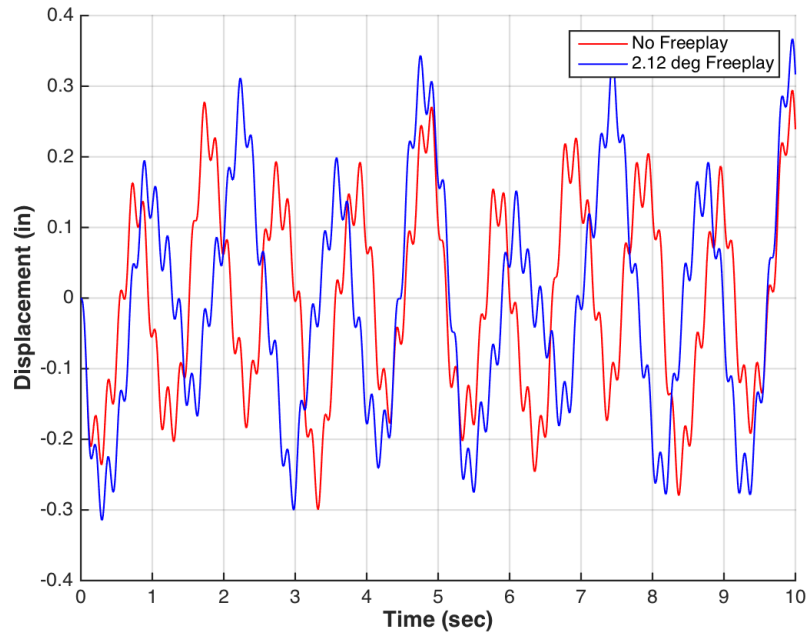


Figure 4.24. Displacement over time when $\beta = 0^\circ$ initially and the input force is located at the tip of the flap

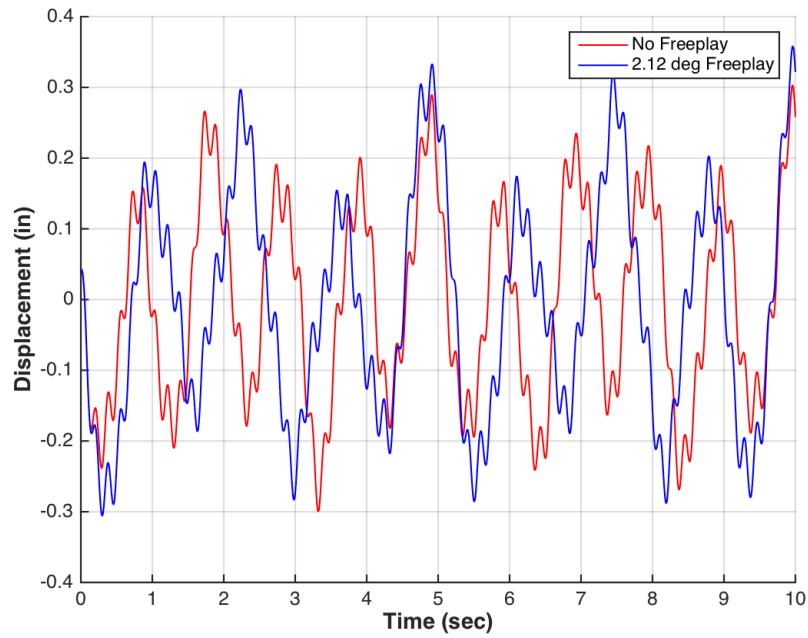


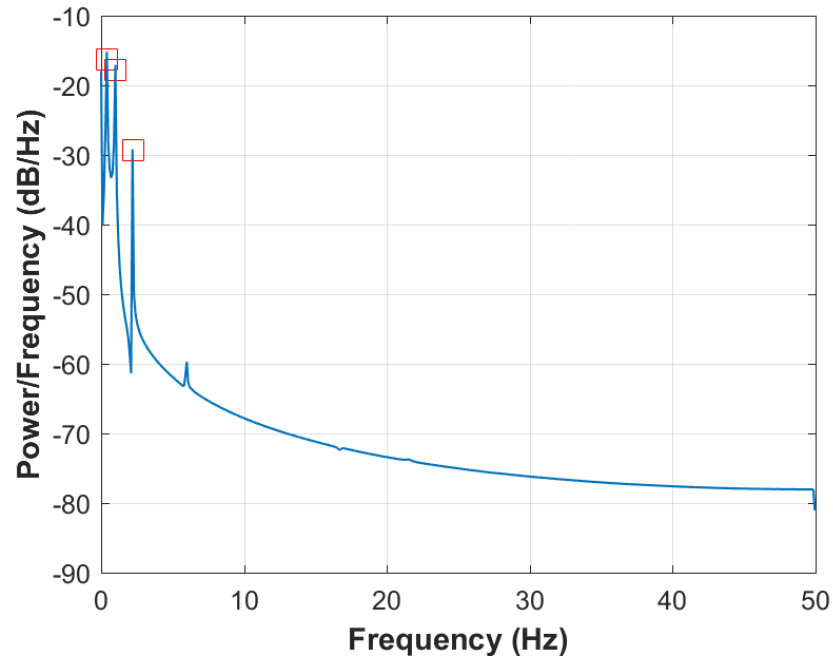
Figure 4.25. Displacement over time when $\beta = 1^\circ$ initially and the input force is located at the tip of the flap

The displacement vector and time vector are fed into a MATLAB script that determines the PSD comparing system with and without freeplay for each initial condition. The red boxes in the plots outline the power at the natural frequencies determined in the previous sections. Figures 4.26 through 4.31 show each comparison of the PSDs.

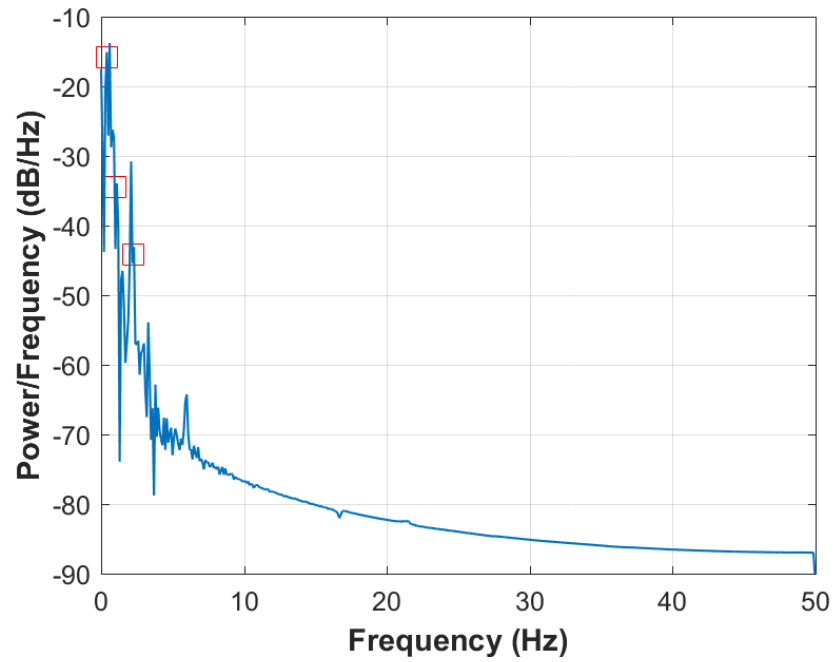
In Figure 4.26a the first three natural frequencies can be clearly seen by three well defined peaks, indicating a linear system. However, in Figure 4.26b not only are the peaks not at the natural frequencies, but below 10 Hz, the entire PSD is jagged and irregular. This is indicative of a nonlinear system that is created from the introduction of freeplay. Being a nonlinear system is a prerequisite for that system to be classified as chaotic.

When $\beta = 0^\circ$ initially and there is no freeplay in the flap spring the same three clear peaks can be distinguished easily. Yet, when freeplay is present only one peak can be slightly distinguished, but less power is present at that natural frequency. Figure 4.27b is less erratic than Figure 4.26b; however, any erratic response on a PSD shows that the system is nonlinear, which proves that the freeplay was added properly.

Meanwhile, β is changed to 1° for Figure 4.28 and the same three well defined peaks can be seen in Figure 4.28a. Again, when the spring is changed to include freeplay, the PSD becomes erratic, implying a nonlinear system.

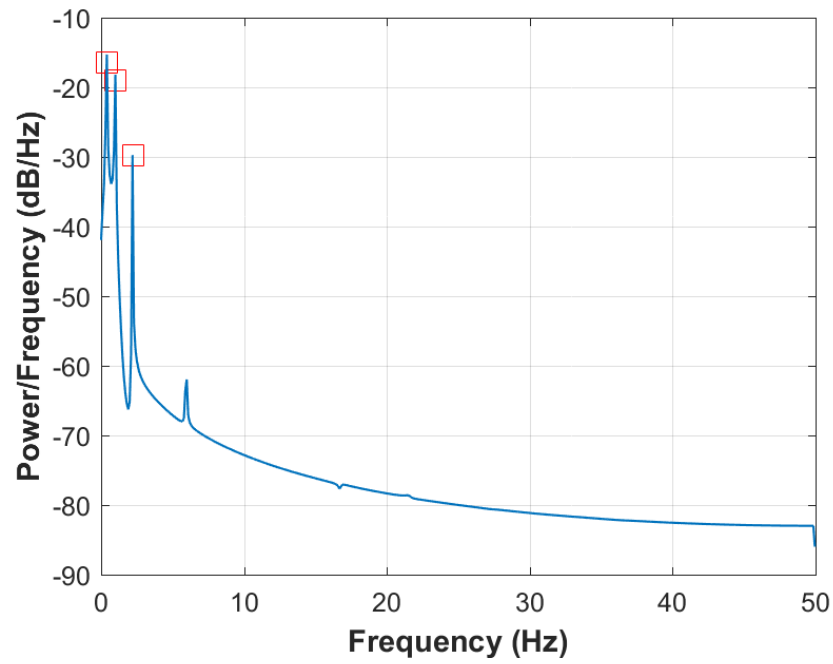


(a) No Freeplay

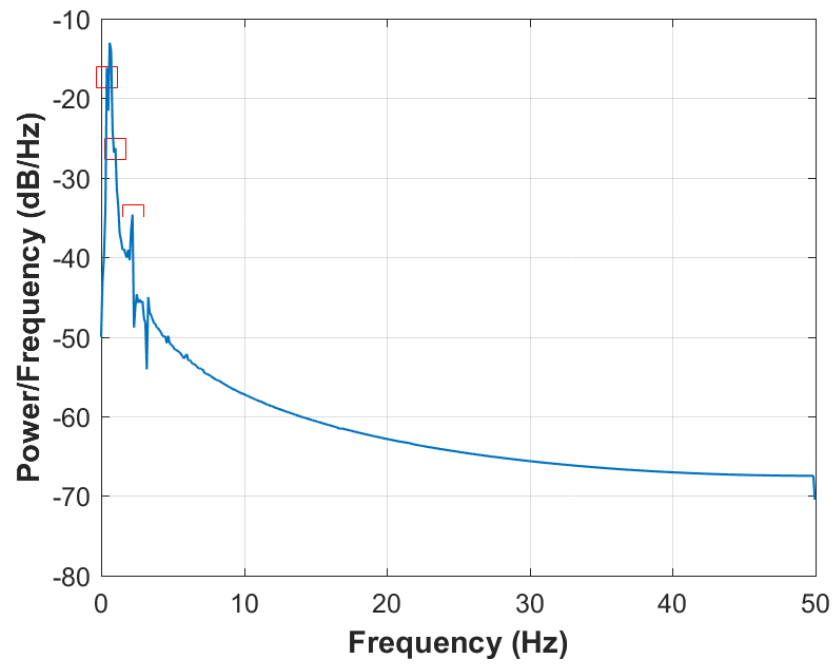


(b) 2.12° Freeplay

Figure 4.26. PSD plots when $\beta = -1^\circ$ initially and the input force is located at the aerodynamic center

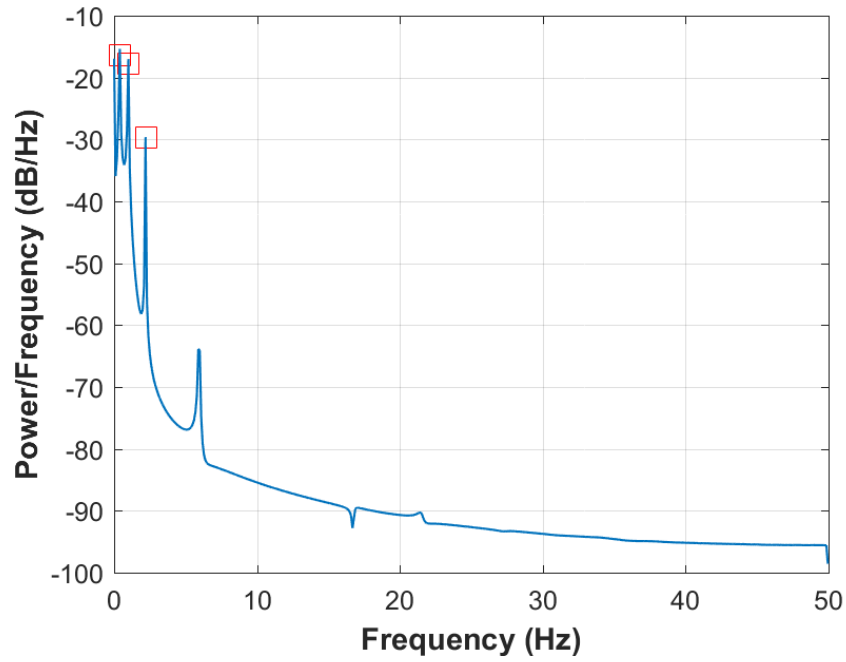


(a) No Freeplay

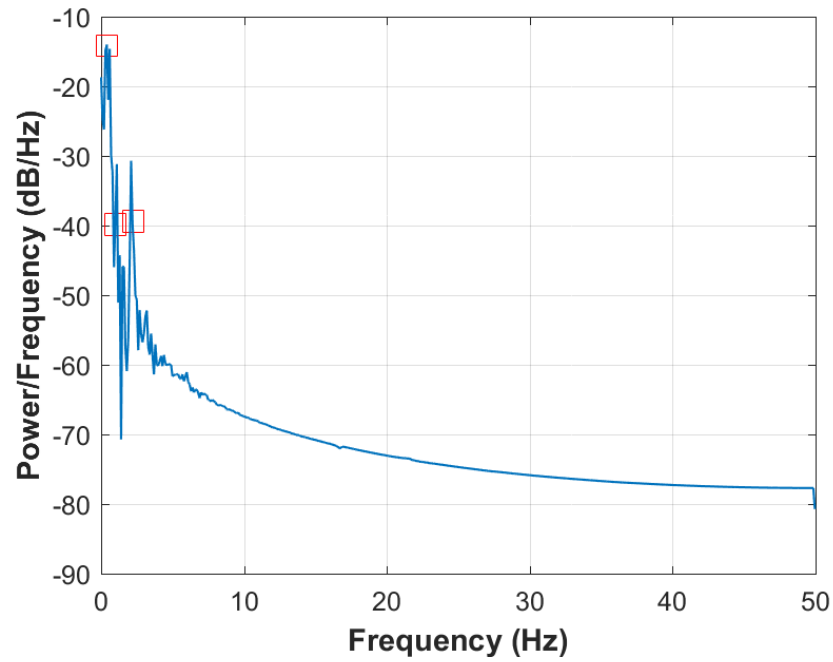


(b) 2.12° Freeplay

Figure 4.27. PSD plots when $\beta = 0^\circ$ initially and the input force is located at the aerodynamic center



(a) No Freeplay

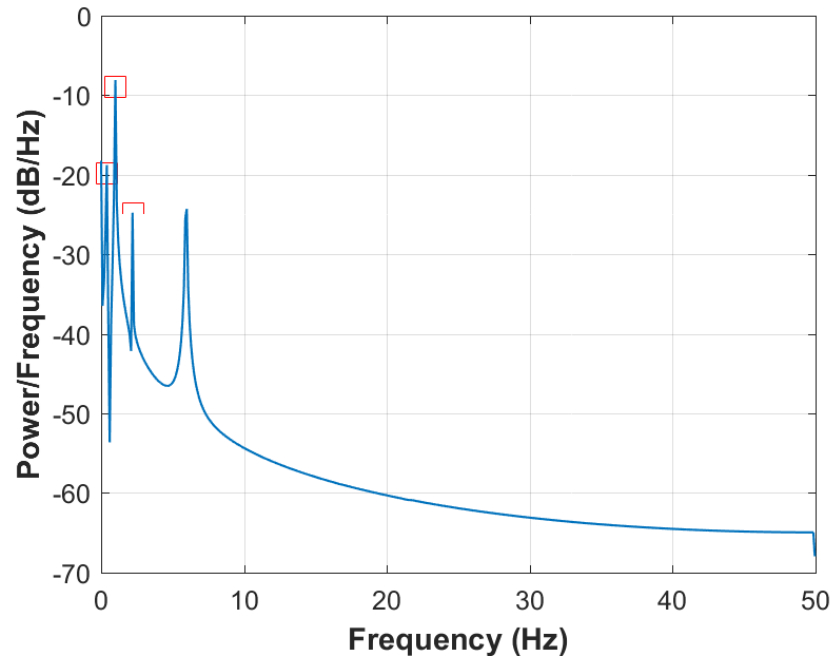


(b) 2.12° Freeplay

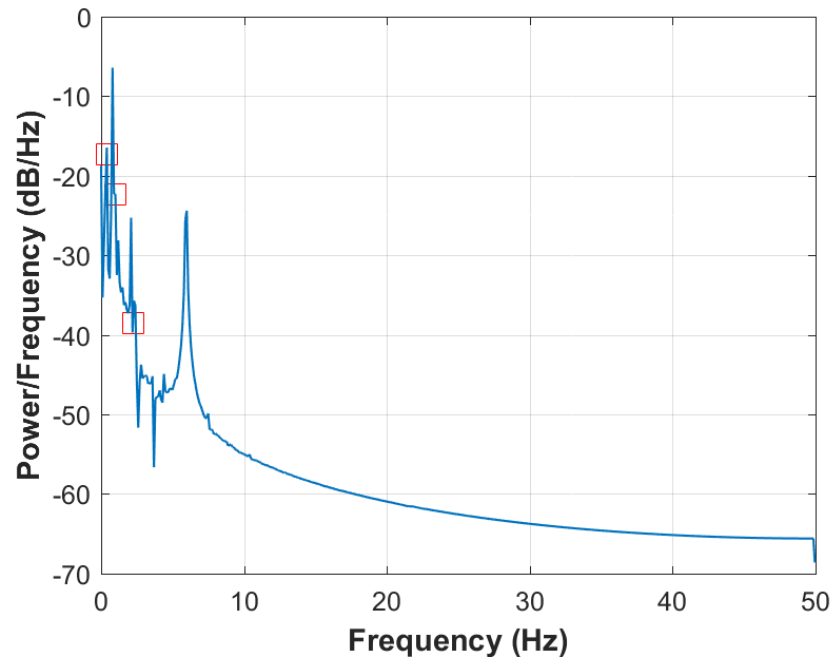
Figure 4.28. PSD plots when $\beta = 1^\circ$ initially and the input force is located at the aerodynamic center

When the placement of the force is changed from the aerodynamic center to the tip of the flap, Figures 4.29 through 4.31 are developed from the displacement of the tip of the flap. In Figure 4.29a three distinct peaks can be seen at the frequencies of the first three modes. A fourth peak arises at 6 Hz. This frequency was not found to be a natural frequency by ABAQUS, because it uses an eigenvalue solver which can sometimes miss natural frequencies. When freeplay is added to the system, the PSD, seen in Figure 4.29b, begins to fluctuate, except at 6 Hz. This peak, while sporadic around it, still is tall and distinct.

Investigating Figures 4.30a and 4.30b it can be seen that without freeplay, the same four peaks are distinct, but with freeplay, only the fourth peak is distinct. Again, other than the peak at 6 Hz, the PSD is erratic, but less erratic than Figure 4.29b. In fact, the same can be said for both PSDs in Figure 4.31, except the PSD in Figure 4.31b is the least erratic of the three conditions with freeplay. These results indicate that the system becomes nonlinear when freeplay becomes present.

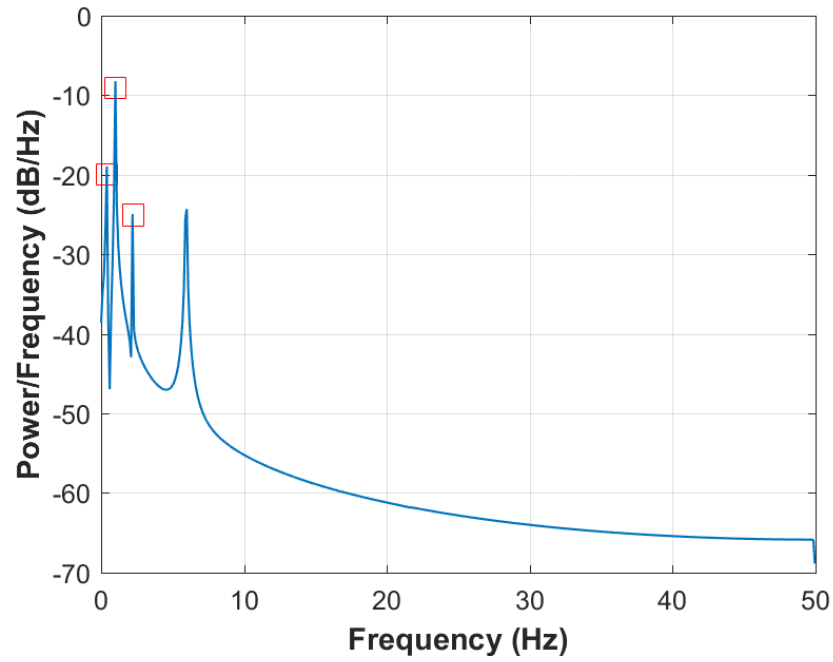


(a) No Freeplay

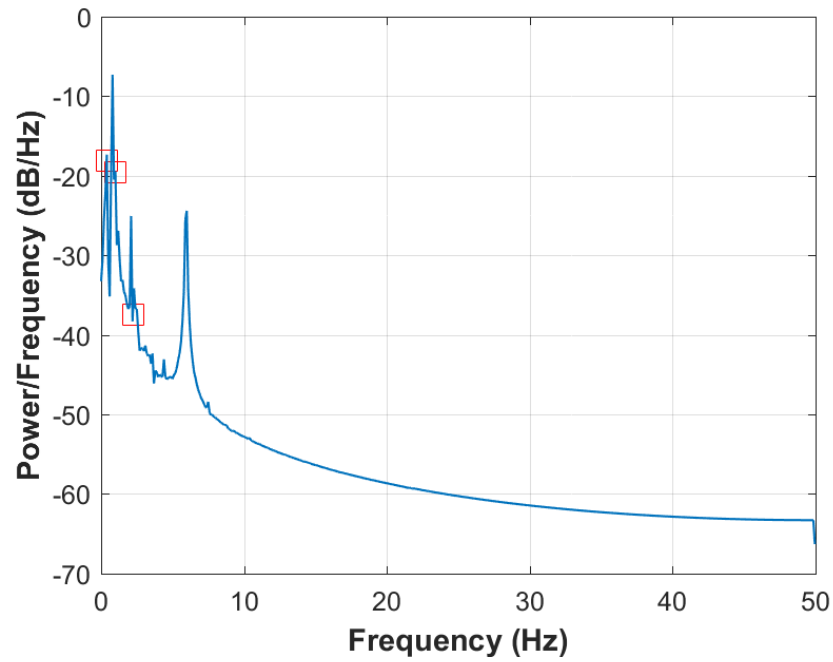


(b) 2.12° Freeplay

Figure 4.29. PSD plots when $\beta = -1^\circ$ initially and the input force is located at the tip of the flap

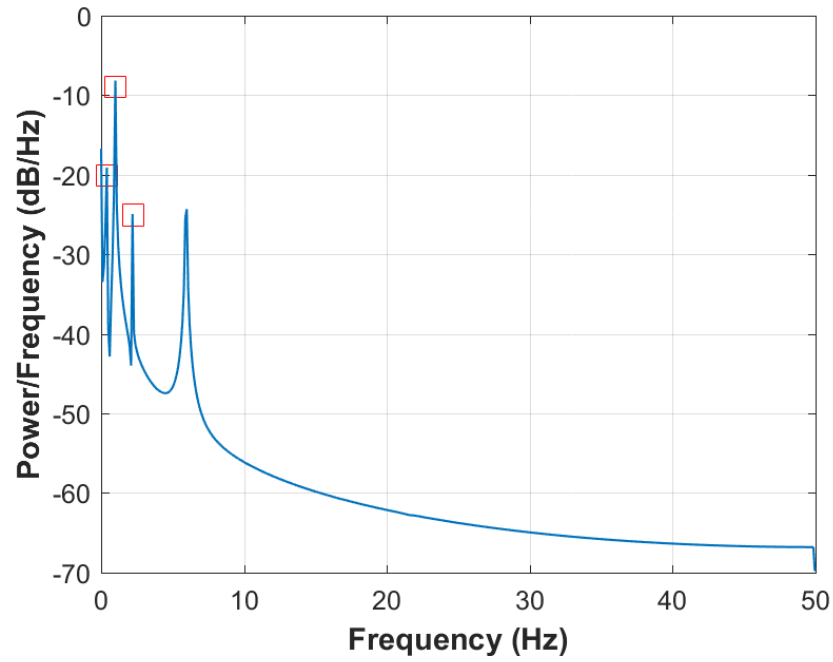


(a) No Freeplay

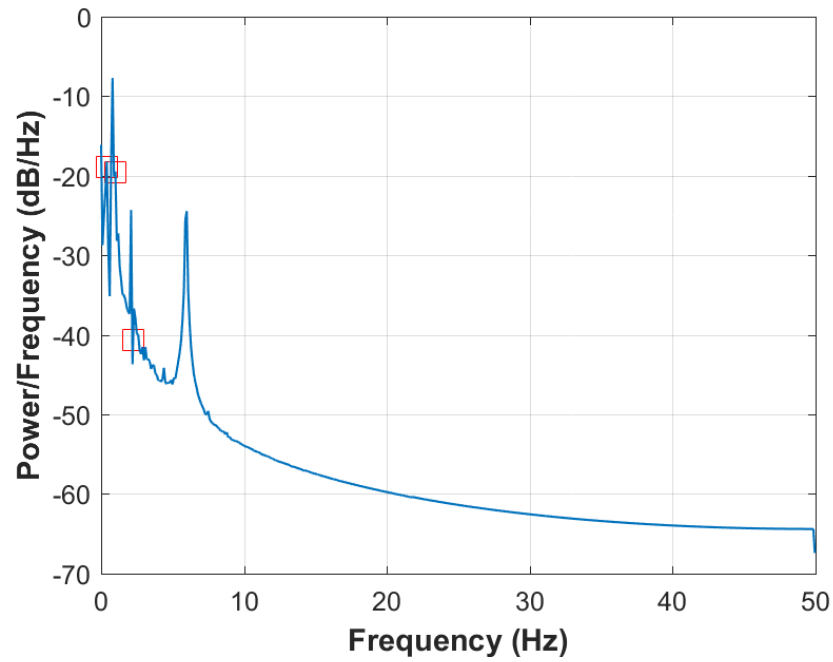


(b) 2.12° Freeplay

Figure 4.30. PSD plots when $\beta = 0^\circ$ initially and the input force is located at the tip of the flap



(a) No Freeplay



(b) 2.12° Freeplay

Figure 4.31. PSD plots when $\beta = 1^\circ$ initially and the input force is located at the tip of the flap

4.5.3 Changing Initial Conditions

Because chaotic systems have a sensitivity to initial conditions, the displacement vs time plots are repeated. Instead of comparing freeplay vs no freeplay, they compare the three initial conditions. Figure 4.32 compares the three initial conditions for β when there is not freeplay in the flap spring and the force is placed at the aerodynamic center. For the majority of the plot, all three of the displacement lie on top of one another. The frequency is the same between all three. The only differences lie in the initial displacement and less than 0.05 inches of displacement difference at peaks. Because the change in initial conditions is roughly 30% of the flap displacement, any conclusions based on initial conditions can not be made. Smaller changes in initial conditions must be made.

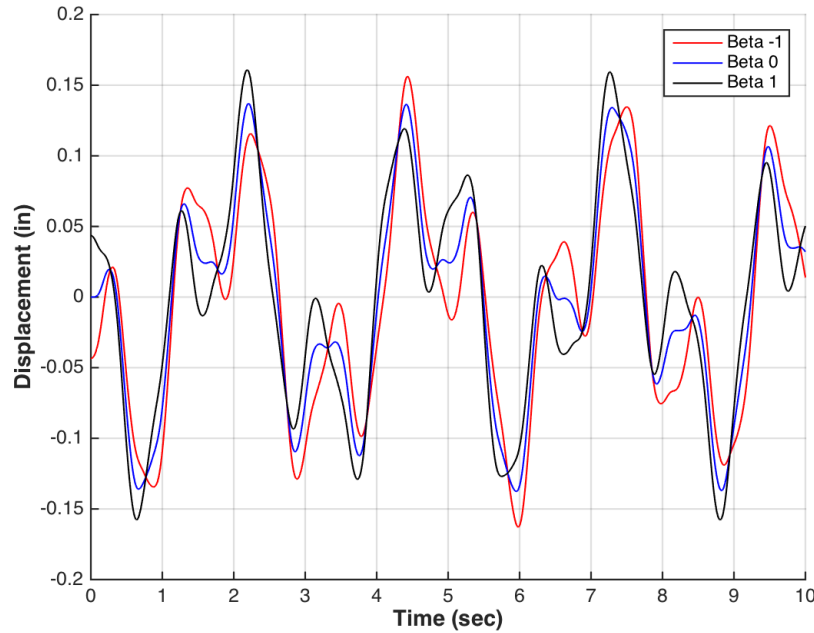


Figure 4.32. Displacement vs time for all three initial conditions with no freeplay and force at the aerodynamic center

Figure 4.33 looks at the displacement for the different initial conditions when freeplay is present in the system. From 1.5 to 3 seconds, a large amount of overlap

exists, but other than that, the three different responses are just that, different. It is not even possible to distinguish if they are responding at the same frequency. This suggests that the transient response still exists in the system and the steady state response has not been reached. Also, the sensitivity to initial conditions cannot be spoken to because of the 25% change in initial conditions from the largest displacement of the tip of the flap.

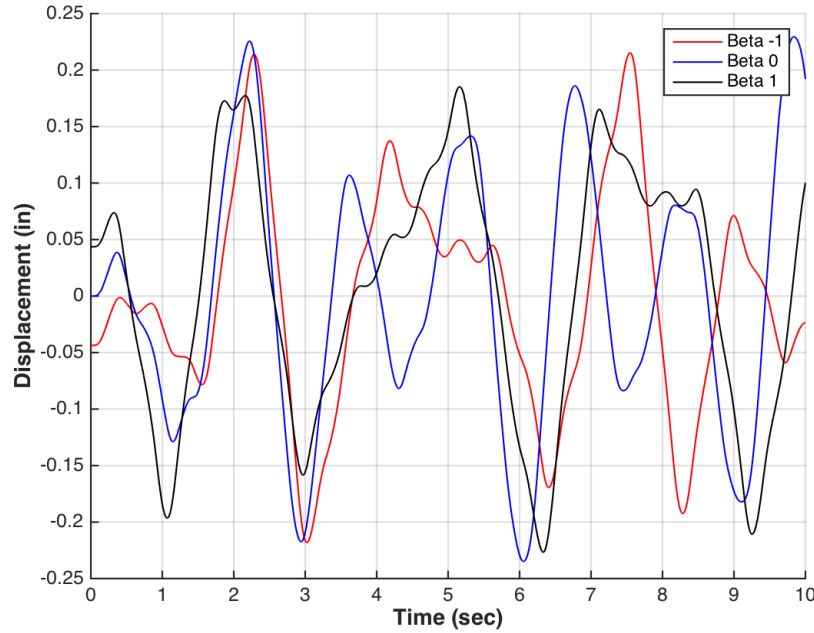


Figure 4.33. Displacement vs time for all three initial conditions with freeplay and force at the aerodynamic center

Switching the force to the tip of the flap and removing the freeplay from the flap spring brings the displacement to the plots shown in Figure 4.34. Other than small shifts in displacement peaks, it is very difficult to discern between the three different initial conditions. Even introducing freeplay into the flap spring the displacement plots, seen in Figure 4.35, are still very difficult to distinguish between. This implies the opposite of what Figure 4.33 shows. From these results, it can be said that exciting the structure at the same point as reading the displacement is not the best

method for analyzing the structure.

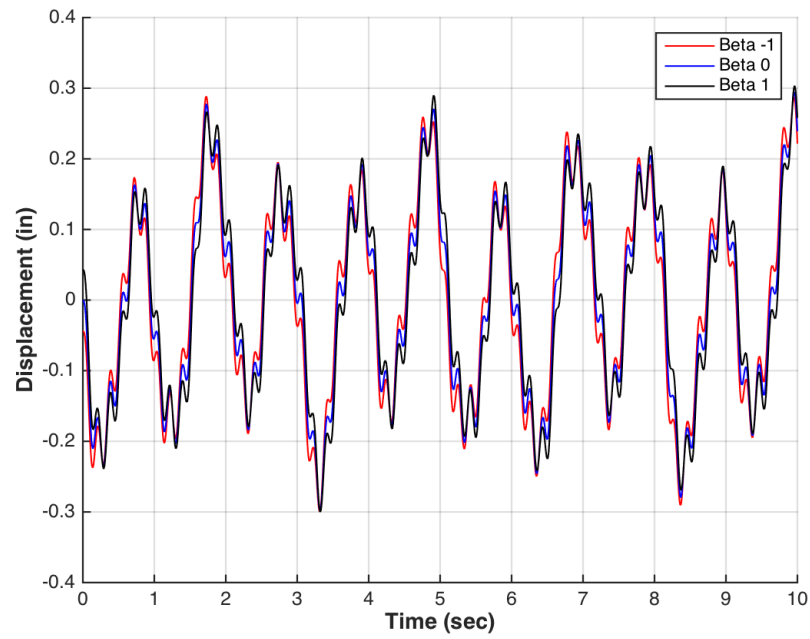


Figure 4.34. Displacement vs time for all three initial conditions with no freeplay and force at the tip of the flap

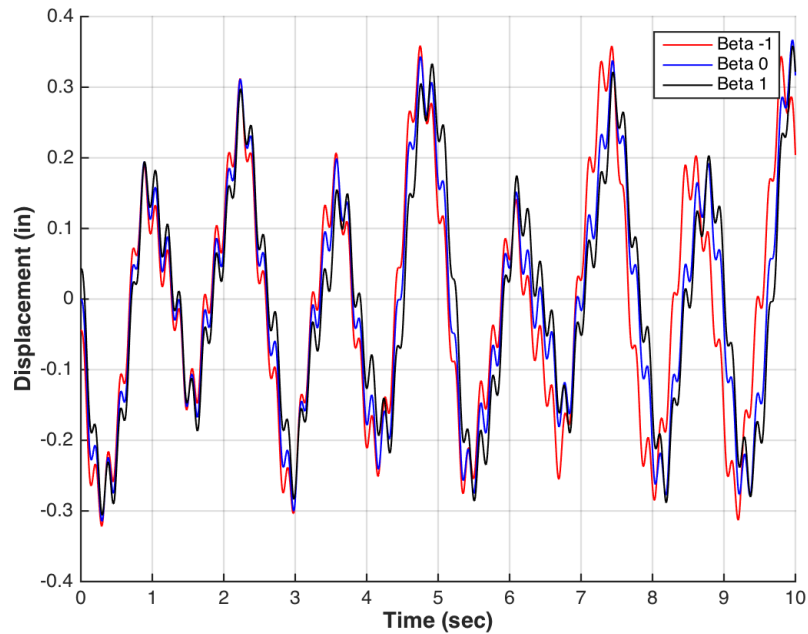
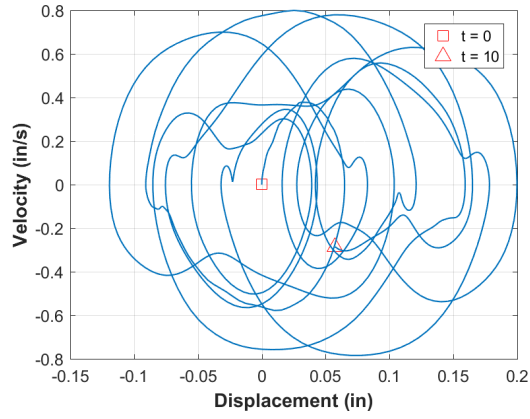


Figure 4.35. Displacement vs time for all three initial conditions with freeplay and force at the tip of the flap

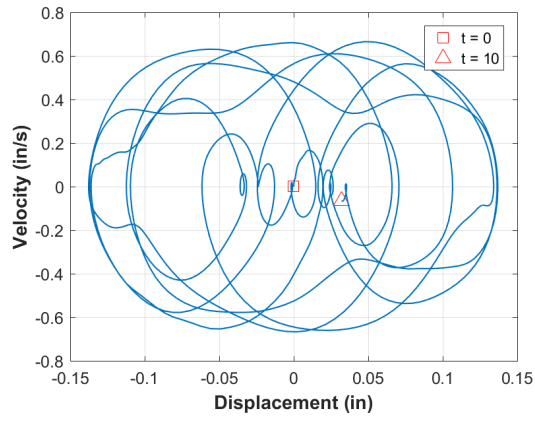
Because displacement is not the only variable to analyze when inspecting for chaos, the phase space plots of velocity vs time are displayed below. Phase space plots should differ vastly when comparing two different initial conditions of a chaotic system, such as a double pendulum. Figure 4.36 shows each phase space plot for each different initial condition when no freeplay is in the system and the force is placed at the aerodynamic center. Generally, they are all different: they reach different peak velocities, they reach different peak displacements, and they do not share any common paths. This can be explained by the large changes in initial conditions. When freeplay is introduced, the results are seen in Figure 4.37. The overall shapes of the phase space plots when β is $\pm 1^\circ$ are similar to each other and different when β is 0° . These differences in the phase space plots can again be explained by the large changes in initial conditions when compared to the largest changes in displacement. It can be seen in all of the phase space plots that an orbit has not been established

for the system. Therefore, the system has not reached steady state movement.

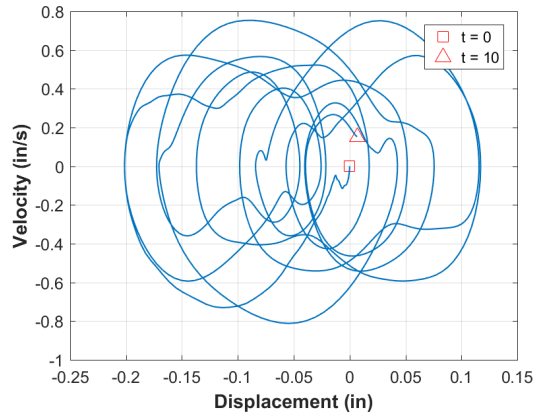
Looking at Figures 4.38 and 4.39, the phase space plots for the system when exciting it at the tip of the flap without and with freeplay, respectively, can be seen. In general, the overall shape of each plot is very similar and the end point on the phase space plot is the same. This statement holds true for when freeplay is in the flap or not. Just as with the displacement vs time plots, the tip excitation phase space plots contradict what the aerodynamic center excitation phase space plots show; therefore, applying the force at the tip of the flap, so near where the displacement is analyzed, is not a good method for determining chaos. Also, the phase space plots for tip excitation have also not reached a repeatable orbit, suggesting that the system has not reached steady state movement.



(a) $\beta = -1^\circ$

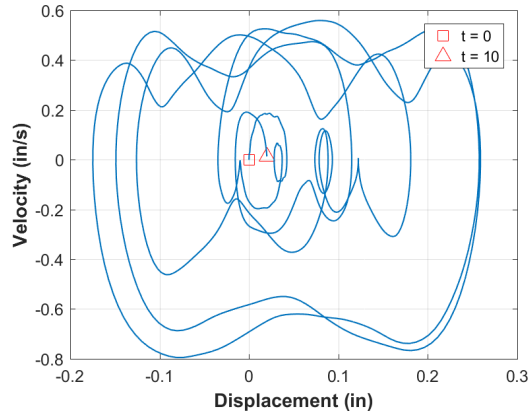


(b) $\beta = 0^\circ$

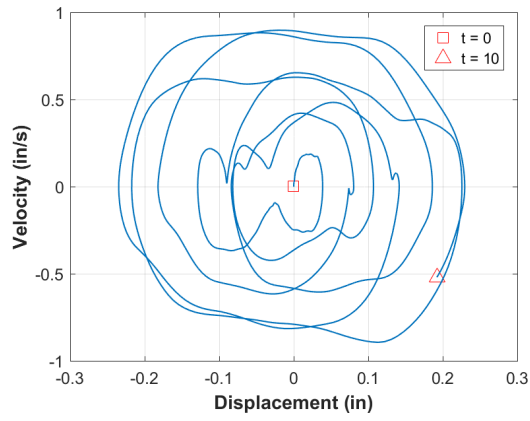


(c) $\beta = 1^\circ$

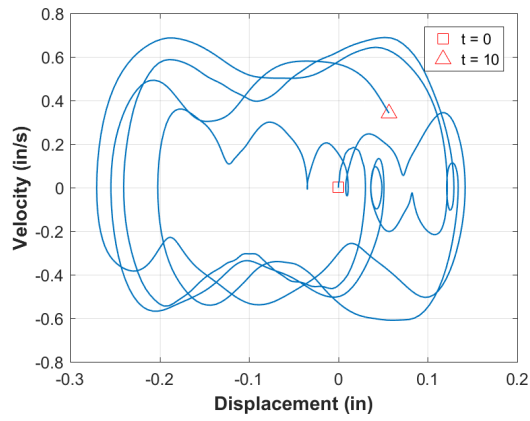
Figure 4.36. Phase space plots for three different initial conditions with no freeplay and force at the aerodynamic center



(a) $\beta = -1^\circ$

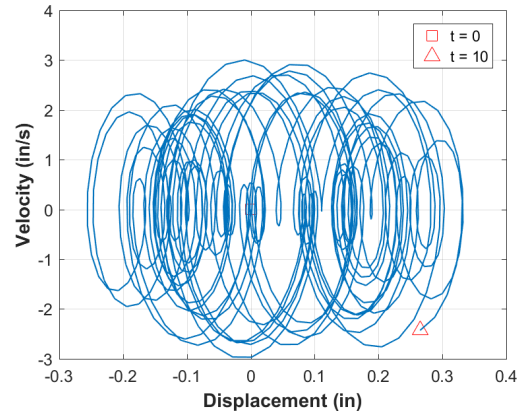


(b) $\beta = 0^\circ$

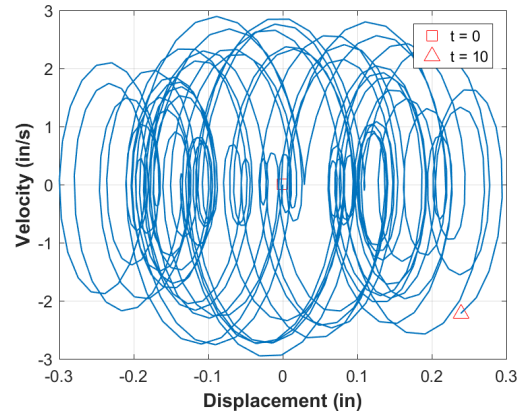


(c) $\beta = 1^\circ$

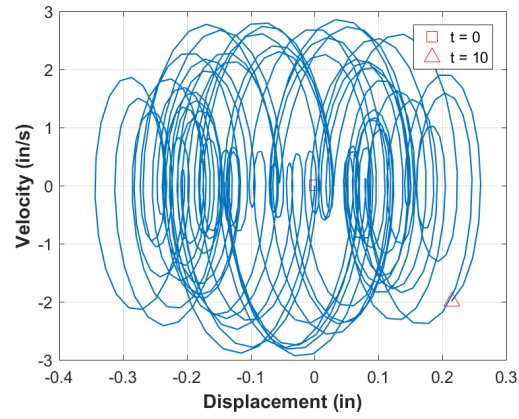
Figure 4.37. Phase space plots for three different initial conditions with freeplay and force at the aerodynamic center



(a) $\beta = -1^\circ$

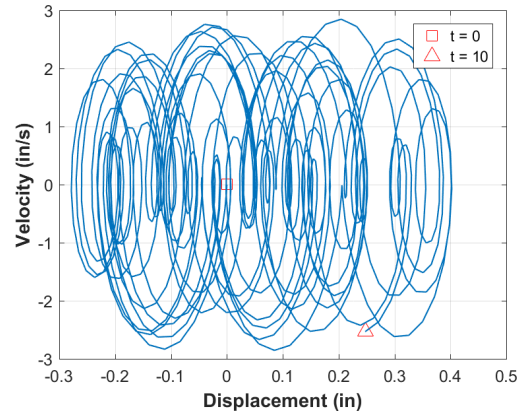


(b) $\beta = 0^\circ$

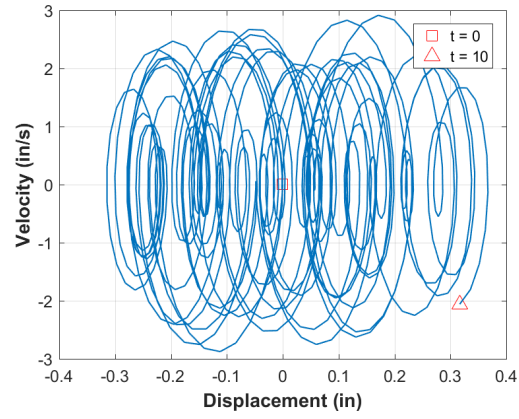


(c) $\beta = 1^\circ$

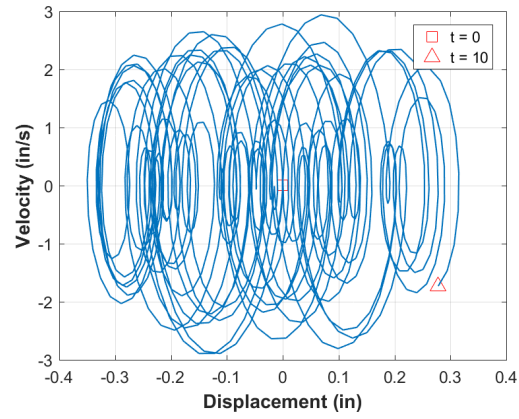
Figure 4.38. Phase space plots for three different initial conditions with no freeplay and force at the tip of the flap



(a) $\beta = -1^\circ$



(b) $\beta = 0^\circ$



(c) $\beta = 1^\circ$

Figure 4.39. Phase space plots for three different initial conditions with freeplay and force at the tip of the flap

4.5.4 Lyapunov Exponent

When viewing phase space plots, a major function to evaluate if the system is chaotic or not is to calculate the Lyapunov exponent from the displacement time series [17]. For each simulation the Lyapunov exponent was calculated at each point on the phase space plot, eventually converging to a single number. The Lyapunov exponent represents the systems convergence or divergence from the systems attractor. A positive exponent is required for a chaotic system. A zero or negative exponent is required for a linear system. That converged number for each simulation is listed below in Table 4.3.

Table 4.3. Table of Lyapunov exponents

Aerodynamic Force		Tip Force	
Description	λ (bit/sec)	Description	λ (bit/sec)
$\beta = -1^\circ$, No Freeplay	0.4278	$\beta = -1^\circ$, No Freeplay	1.5301
$\beta = 0^\circ$, No Freeplay	1.0178	$\beta = 0^\circ$, No Freeplay	2.1134
$\beta = 1^\circ$, No Freeplay	0.3944	$\beta = 1^\circ$, No Freeplay	1.6084
$\beta = -1^\circ$, Freeplay	0.5295	$\beta = -1^\circ$, Freeplay	2.4024
$\beta = 0^\circ$, Freeplay	0.4931	$\beta = 0^\circ$, Freeplay	5.6625
$\beta = 1^\circ$, Freeplay	1.0044	$\beta = 1^\circ$, Freeplay	3.2795

Each simulation has a Lyapunov exponent greater than 0, but they are all very close to zero. This shows that either the parameters chosen for the Lyapunov exponent calculation are off or the simulation was not run long enough for the transient response to reduce to zero. There is no correlation between introducing freeplay and the Lyapunov exponent increasing. There is also no correlation between any one initial condition for β and an increasing or decreasing Lyapunov exponent. These observations point to being unable to state that the system is chaotic or not when freeplay is present, and questions the calculation of the Lyapunov exponent until longer simulations are analyzed.

4.5.5 Effects of Frequency on Dynamics

For all of the simulations presented above, the excitation frequency was held constant at ω_f , which is equal to 5.47 Hz. This was an attempt to prove that flutter equated to chaos. To fully investigate this, a different frequency must be applied. For these simulations, an excitation frequency of 0.8 Hz was utilized. Figure 4.40 shows the displacement vs time curves for no freeplay and freeplay when β is initially set to 0° . Very little differences can be noted between the two plots. When freeplay is in the flap spring, the displacement of the flap is greater at the peak displacements.

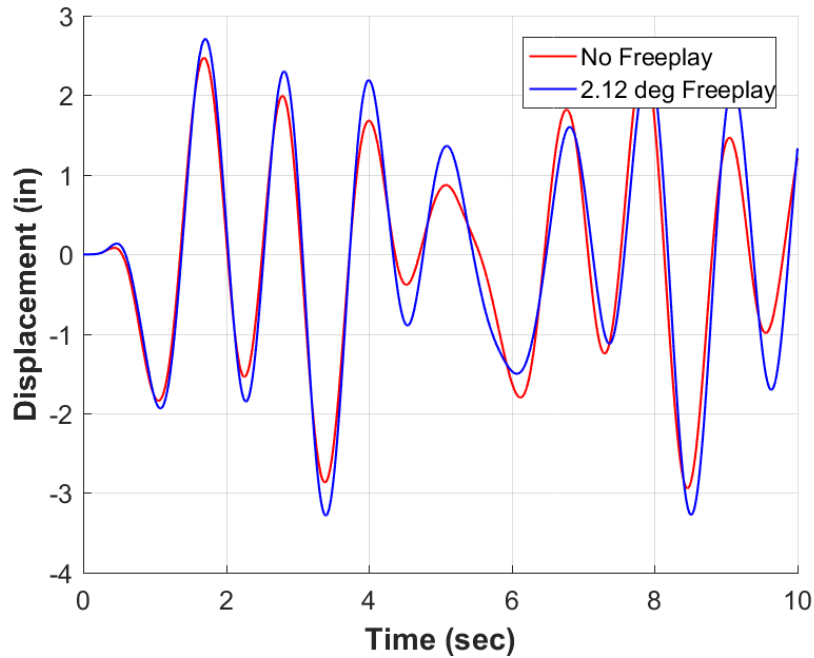
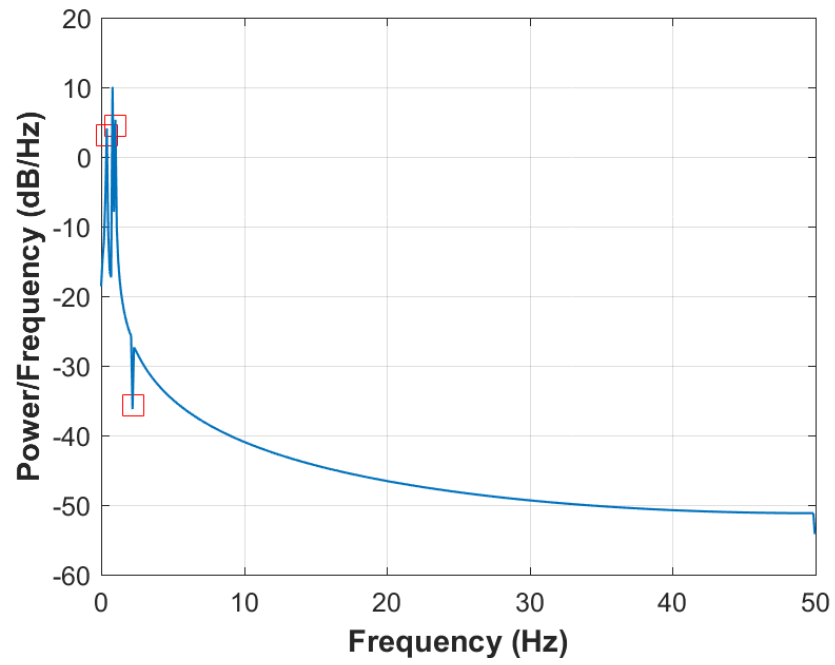


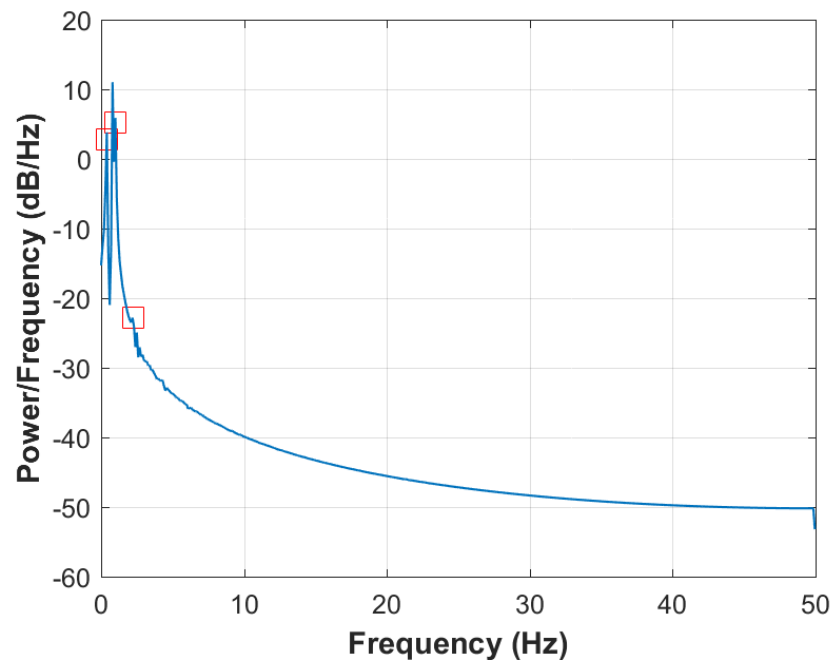
Figure 4.40. Displacement vs time for $\beta = 0^\circ$, with and without freeplay, excited at 0.8 Hz

Next the displacement time series are fed into the PSD MATLAB script and Figure 4.41 is generated. Without freeplay, the system has an antiresonance at mode three, which is intriguing. The linear system does however have clearly defined peaks, while the system with freeplay is slightly erratic. This proves that the system with freeplay is nonlinear while the system without freeplay is linear. Finally, the phase space plots

are compared between the two simulations. Both plots appear to be similar and do not enter into a steady state orbit, suggesting that the simulation length of time was not long enough. Calculating the Lyapunov exponent only further solidifies this result. For the excitation frequency of 0.8 Hz and no freeplay in the flap spring, λ is 12.8857 bits per second. When freeplay is included in the flap spring, λ decreases to 11.2698. As stated previously, the system without freeplay is a completely linear system and therefore cannot have a positive Lyapunov exponent. Either the simulation was not long enough, or the parameters used to calculate the Lyapunov exponent are incorrect.

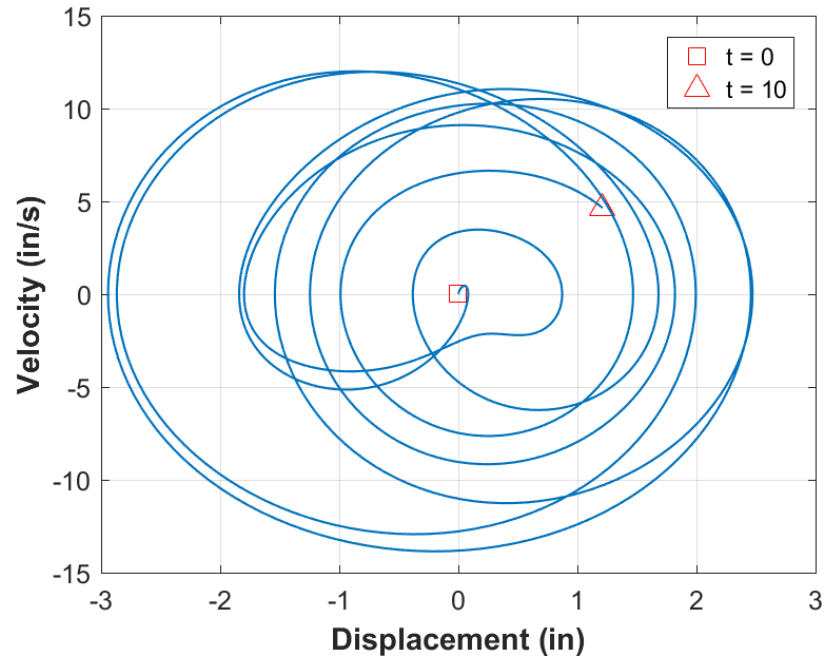


(a) No Freeplay

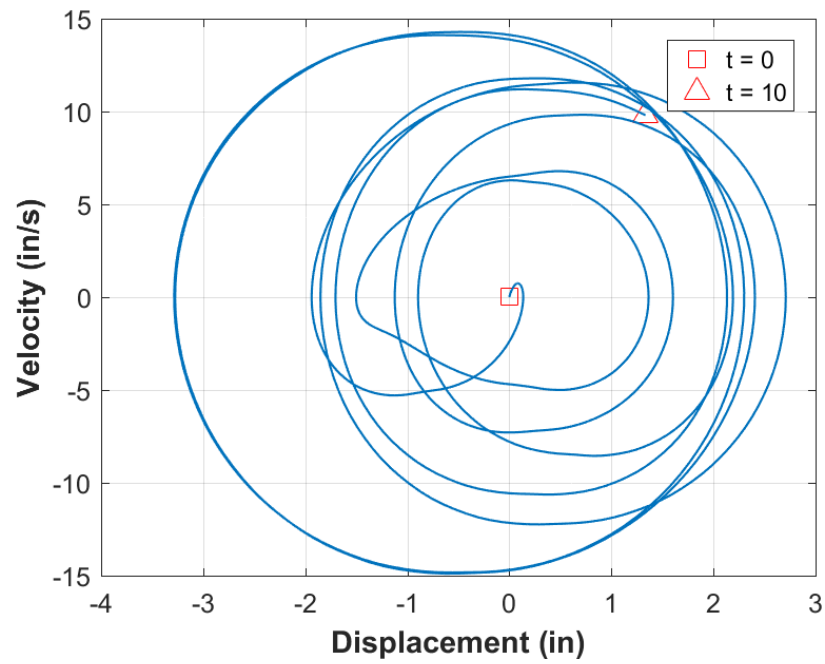


(b) 2.12° Freeplay

Figure 4.41. PSD plots when $\beta = 0^\circ$ initially and the input frequency is 0.8 Hz



(a) No Freeplay



(b) 2.12° Freeplay

Figure 4.42. PSD plots when $\beta = 1^\circ$ initially and the input force is located at the tip of the flap

4.5.6 Longer Simulations

From all of the results above, no statement of a chaotic system can be made. Each simulation was run for only 10 seconds, which turned out to be not long enough for the system to reach a steady state value. To test that this is true, simulations of the airfoil were made out to 100 and 300 seconds. The entire definition of the simulations can be seen in Chapter 3.

When freeplay is not introduced into the system, the system is completely linear. Figures 4.43, 4.44, and 4.45 show the displacement plot, the PSD, and the phase space plot respectively, when the simulation is extended to 100 seconds. The PSD shows that the system is in fact linear, because of the clearly defined peaks. The phase space plots shows that the system is beginning to enter into a steady state orbit. The Lyapunov exponent calculated from the displacement time series is 0.2912. This is severely decreased from the 1.0178 Lyapunov exponent calculated for the 10 second simulation. This suggests that the linear system, when transient motion is removed, has a Lyapunov exponent of 0 and is not chaotic. This would also imply that the parameters chosen for the calculation of the Lyapunov exponent were accurate, because a linear system cannot be chaotic.

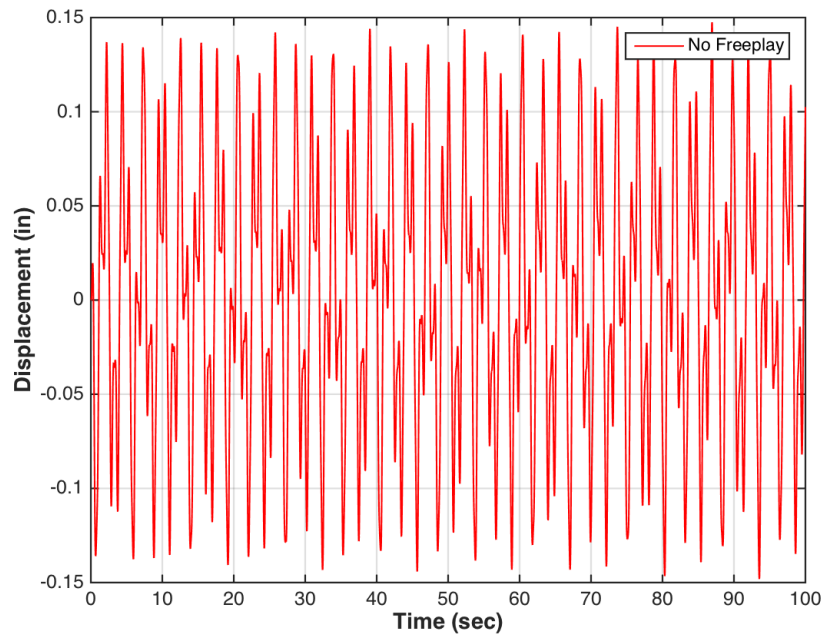


Figure 4.43. Displacement vs time for $\beta = 0^\circ$, without freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 100 seconds

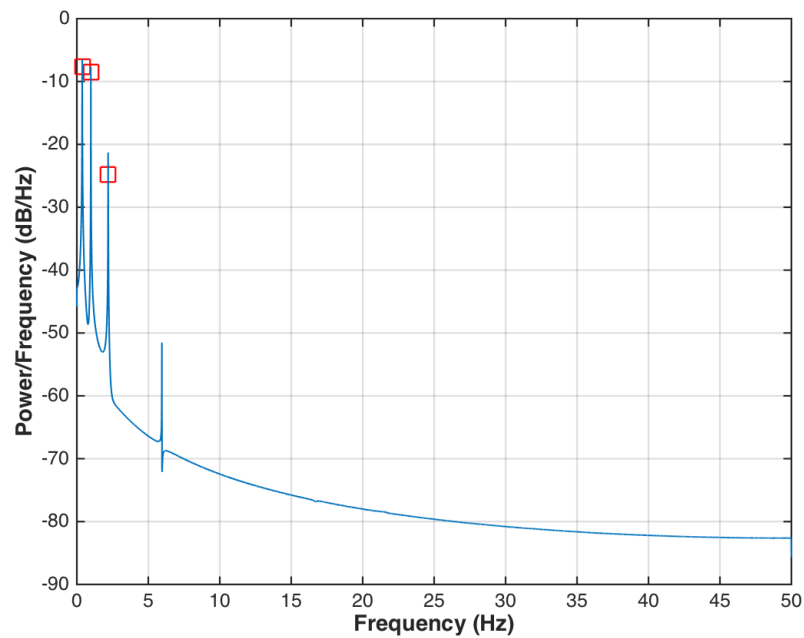


Figure 4.44. PSD for $\beta = 0^\circ$, without freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 100 seconds

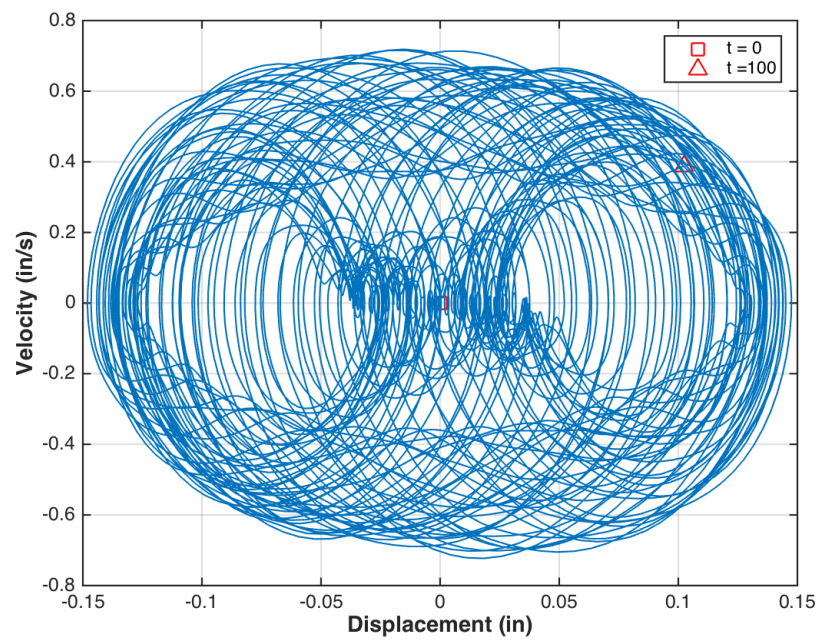


Figure 4.45. Phase space plot for $\beta = 0^\circ$, without freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 100 seconds

To further investigate if the parameters for calculating the Lyapunov exponent are accurate, the simulation for the system was extended to 300 seconds. The results for this simulation can be seen in Figures 4.46, 4.47, and 4.48. Figure 4.46 shows the displacement of the tip of the flap over the 300 seconds. Figure 4.47 shows the PSD, which has clearly defined peaks and smooth valleys. This shows that the system is linear. Finally, Figure 4.48 shows the phase space plot for the duration of the simulation. It can be seen that the orbit of the 100 seconds duration appears to be filled in further. With the longer simulation, the likelihood that the system is operating at steady state conditions is greater. The Lyapunov exponent calculated for this simulation is 0.1598. This only further solidifies that the exponent for the linear system is converging to zero as time goes on. This means that the parameters used to calculate the Lyapunov exponent are valid, if the simulation duration is great enough.

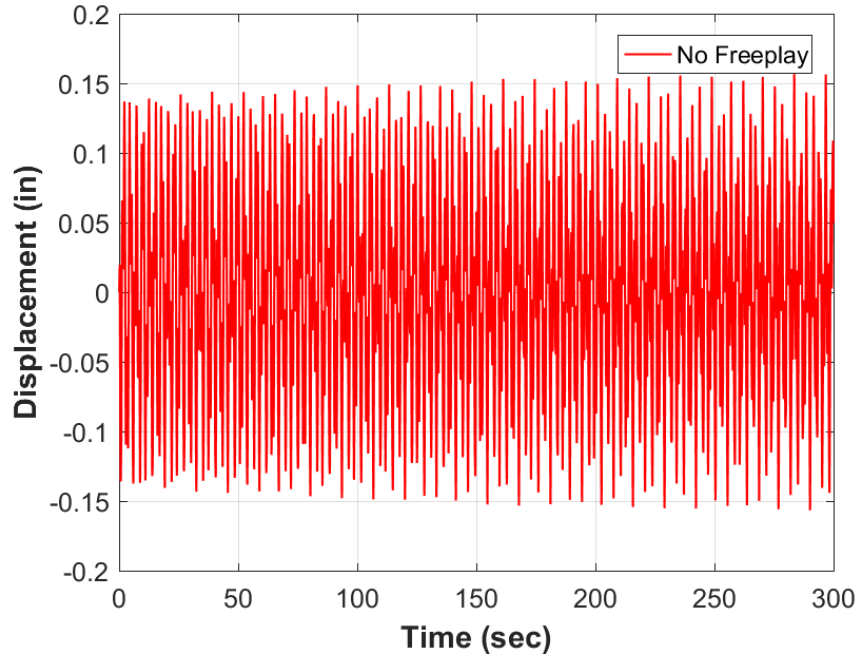


Figure 4.46. Displacement vs time for $\beta = 0^\circ$, without freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 300 seconds

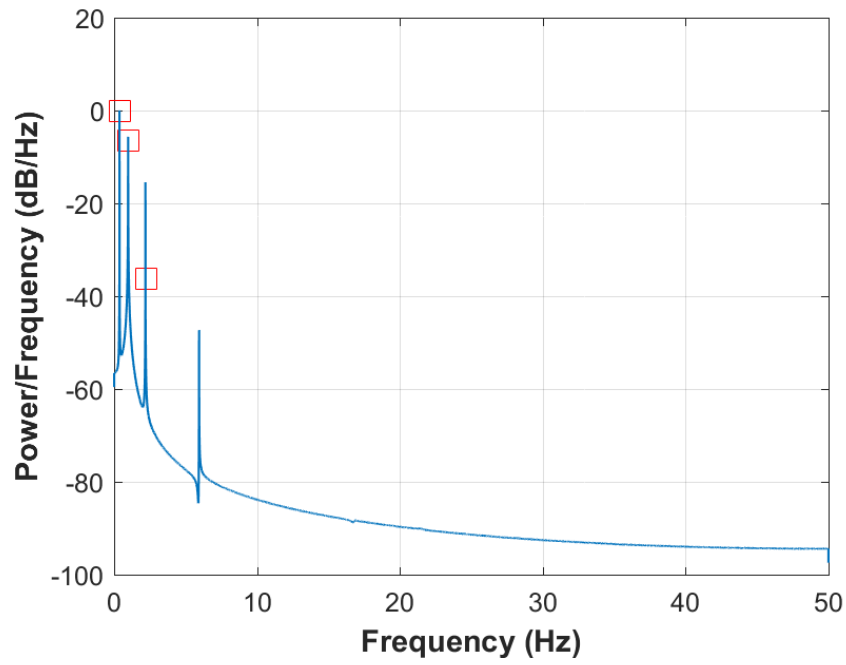


Figure 4.47. PSD for $\beta = 0^\circ$, without freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 300 seconds

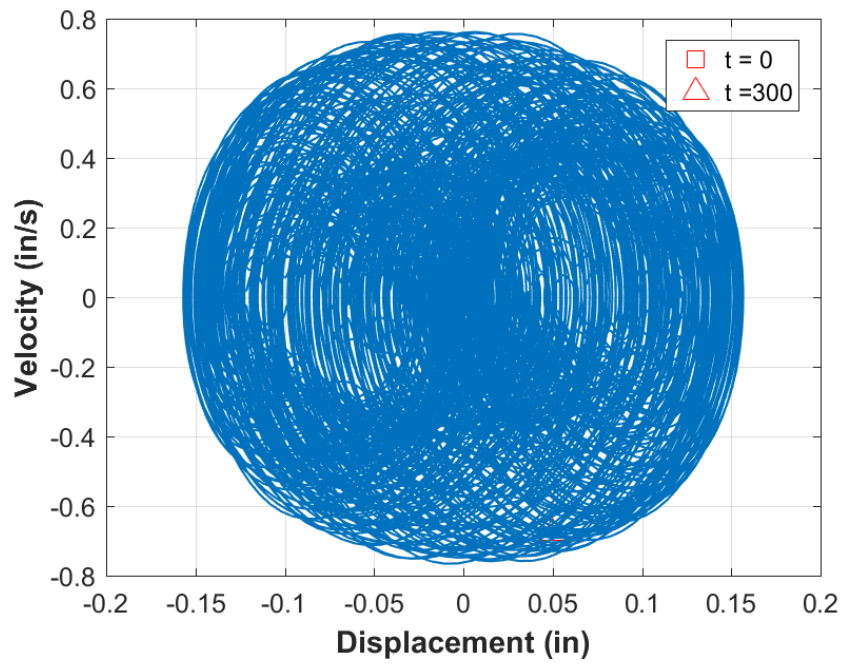


Figure 4.48. Phase space plot for $\beta = 0^\circ$, without freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 300 seconds

Extending the lessons learned above to the system with freeplay, the nonlinear system can be analyzed with longer simulations. Figures 4.49, 4.50, and 4.51 show the displacement plot, the PSD, and the phase space plot for the nonlinear system when the simulation is extended to 100 seconds. The displacement plot shows that as time simulation progresses past 20 seconds the response of the tip of the flap begins to change. The same can be said for the system at 60 and 80 seconds. The PSD is extremely erratic, showing that the system, with freeplay in the flap spring, is in fact nonlinear. The phase space plot shows that the system may have begun to enter an orbit, but it is not entirely clear. That can also be said from the calculated Lyapunov exponent for the simulation, 1.4409. This increase from 0.4903 suggests that the system has potential to be classified as chaotic, but a longer simulation is needed to verify this belief.

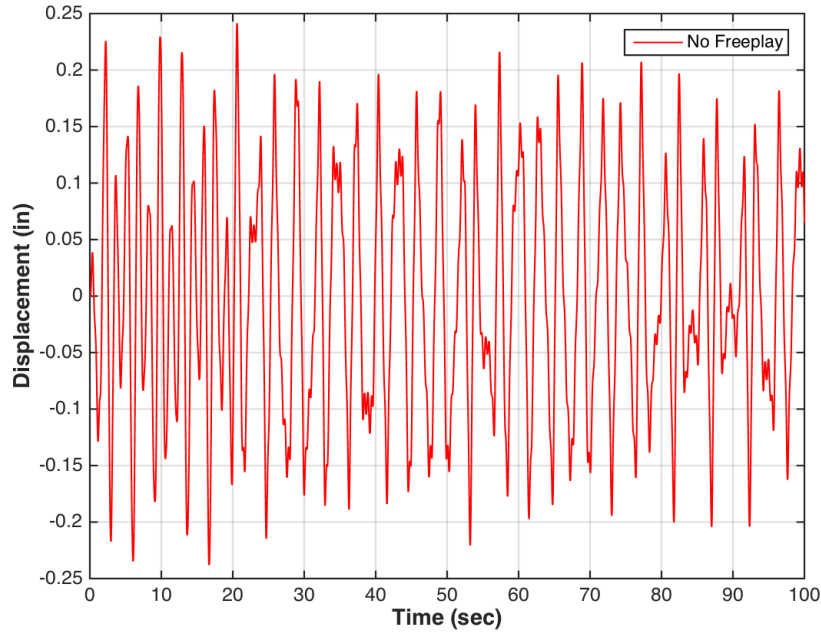


Figure 4.49. Displacement vs time for $\beta = 0^\circ$, with freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 100 seconds

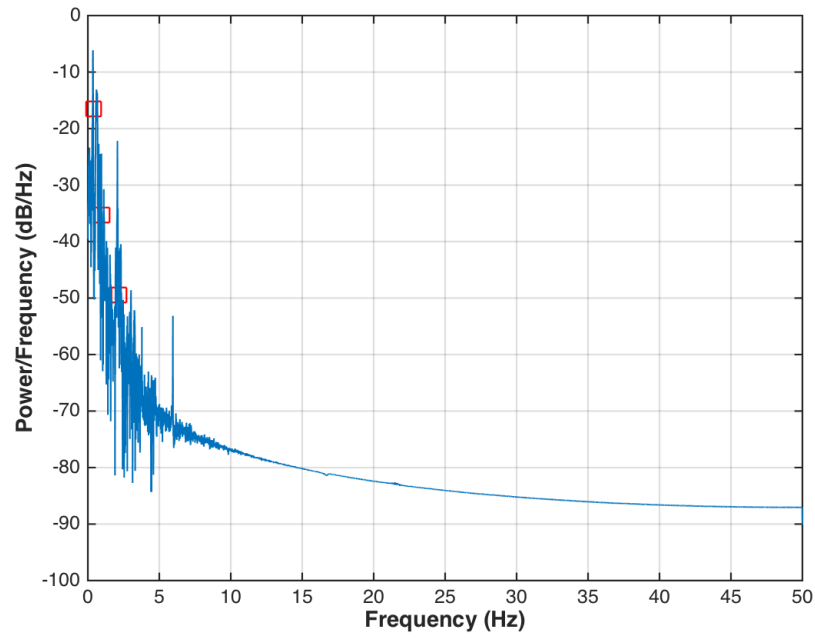


Figure 4.50. PSD for $\beta = 0^\circ$, with freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 100 seconds

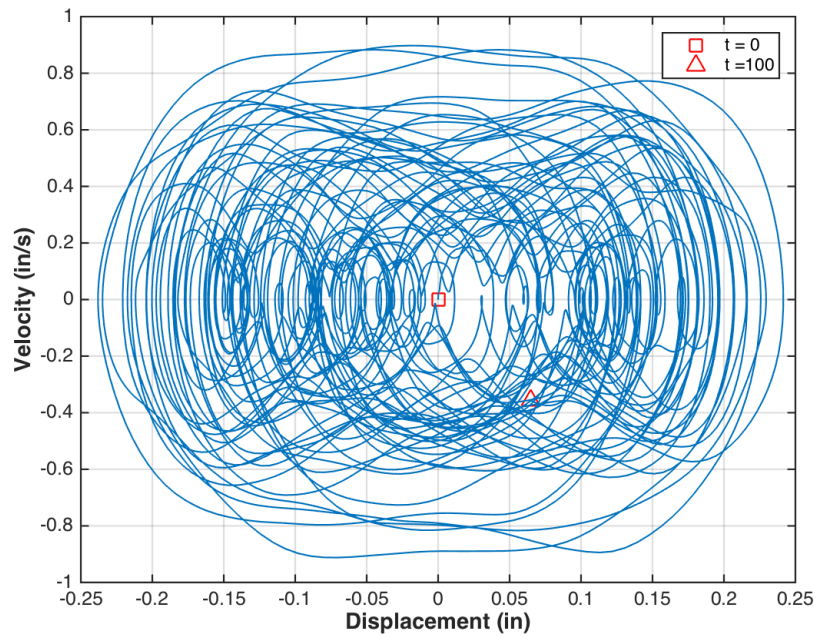


Figure 4.51. Phase space plot for $\beta = 0^\circ$, with freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 100 seconds

Because the previous simulation showed an increase in the Lyapunov exponent, a 300 second simulation was run to further allow the exponent to converge on a final number and the system to enter a more steady state movement. Figure 4.52 shows the displacement of the system for the duration of the simulation. It can be seen that the system continues to shift when analyzing the response of the tip of the flap. This either indicates that the system has failed to reach steady state, or the system is unpredictable. Figure 4.53 shows the PSD of the system. It is erratic and indicates the system is nonlinear, a prerequisite for any chaotic system. Figure 4.54 shows the phase space plot for the system. This plot shows that the system is in fact entering into an orbit for its attractor, but does not suggest that the system has reached steady state. The Lyapunov exponent for this simulation was calculated to be 0.7981. That is a decrease from the 100 second simulation. This means that the simulation duration was too short and did not allow the system to reach steady state. The Lyapunov exponent could be converging to zero, just as it did with the linear system, or it could continue to converge to a positive value, indication a chaotic system. Neither conclusion could be drawn.

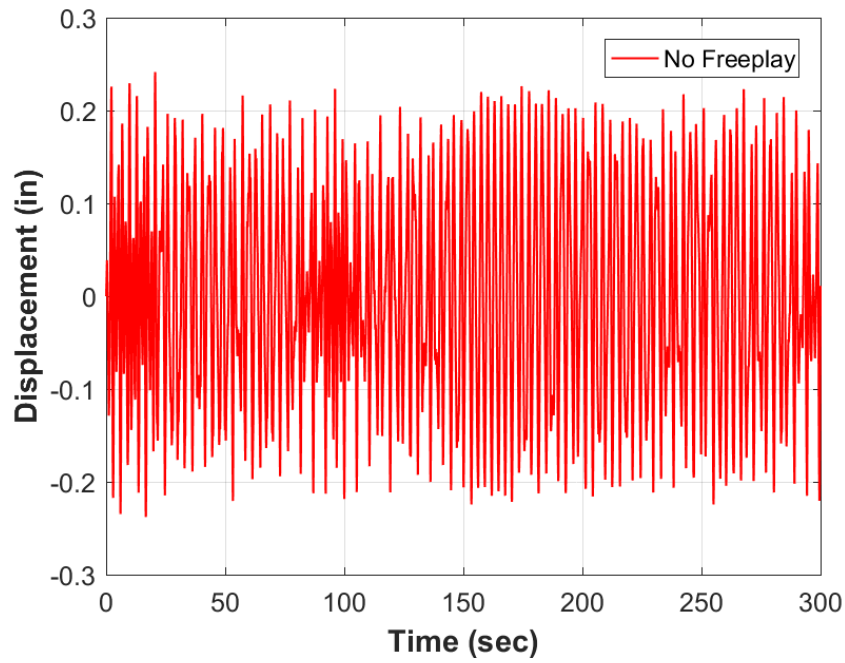


Figure 4.52. Displacement vs time for $\beta = 0^\circ$, with freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 300 seconds

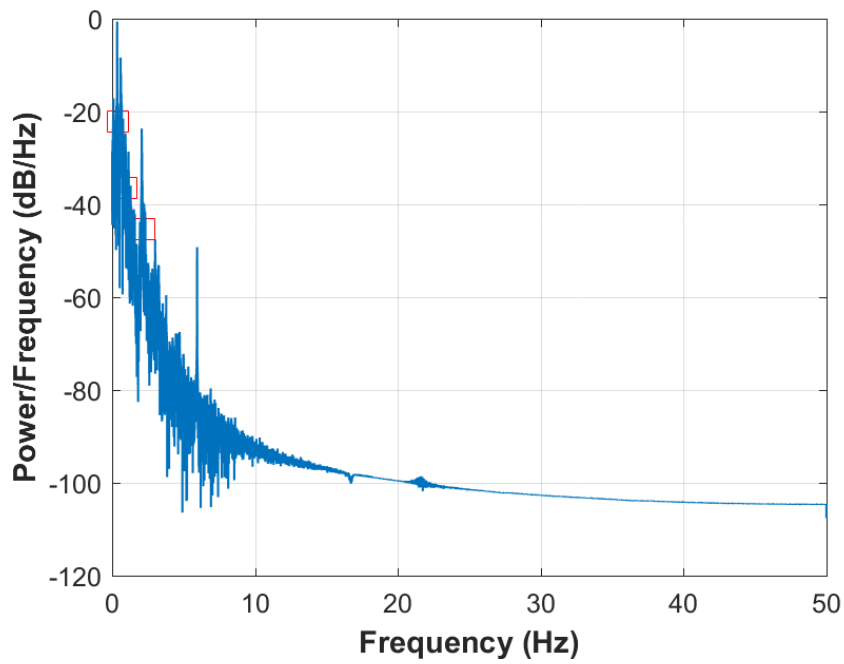


Figure 4.53. PSD for $\beta = 0^\circ$, with freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 300 seconds

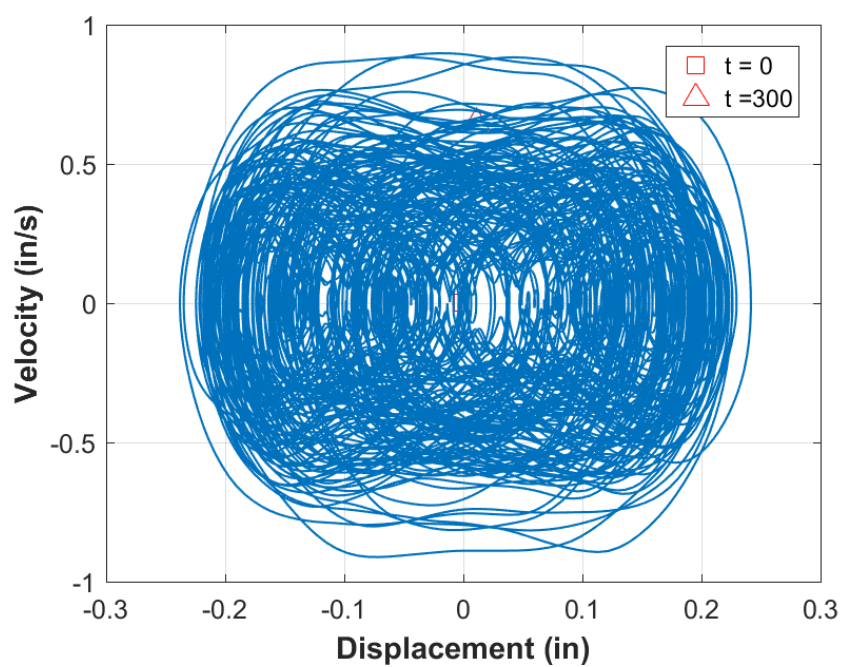


Figure 4.54. Phase space plot for $\beta = 0^\circ$, with freeplay, excited at the aerodynamic center at 5.47 Hz, simulated to 300 seconds

V. Conclusions and Recommendations

5.1 Chapter Overview

In previous chapters this paper has presented the motivation behind the research being performed, discussed the theory needed to understand the work, explained previous research that has led to this point, broken down the methods utilized, and examined the results of the investigation. This chapter aims to present the overall findings of this research, the relevance of the findings, any lessons learned, and any future research possibilities.

5.2 Conclusions of Research

- Topology optimization of an airfoil required the constraint of the center of gravity; without it, the structure varied greatly. When using Optistruct, the center of gravity lower bound had to be that of the actual airfoil, because Optistruct attempted to place it as close to the lowest compliance point as possible.
- At subsonic speeds, it was determined that while deformation of the National Advisory Committee for Aeronautics (NACA) 0012 internal structure and skin changed the coefficient of lift value by 39.5%, the actual lift created by this change is still negligible in magnitude. That is to say the assumption of a rigid airfoil structure at subsonic speeds for aeroelasticity purposes holds true.
- The two dimensional representation of the average wing structure was more stiff than the span wise spring stiffness of the wing. This was seen in the modal analysis of the overall structure by analyzing the mode shapes of each natural frequency.
- When freeplay was introduced into the system, the Power Spectral Density (PSD) became erratic. This indicated that the PSD function was not defined for this

system and that the system is therefore nonlinear.

- Analyzing the chaos of the system requires the transient response of the system to stop and the steady state response to dominate. This required a minimum of 100 seconds for the linear system and 300 seconds for the nonlinear system.
- Utilizing code written by Wolf to calculate the Lyapunov exponent from a time series, the Lyapunov exponents were calculated for each simulation [17]. When the duration of the simulation was continuously increased, the linear system had a Lyapunov exponent that was converging to zero, indicating the correct parameters were utilized. For the nonlinear system, or the system with freeplay, the Lyapunov exponent did not appear to converge to a single value, not allowing the the chaos of the system to be commented on.

5.3 Significance of Research

The internal structure of an airfoil was developed and optimized, the deformation of the skin with an internal structure was analyzed for changes in flow over it, and the structure was placed in a dynamic model representing an aeroelastic airfoil and analyzed for chaotic motion. The methods and procedures developed can be utilized to analyze other systems for chaos, as well as potentially present new ways for modeling aeroelastic systems. The results have also shown that long duration simulation must be performed to properly analyze a system for chaos.

5.4 Lessons Learned

- The simulation duration is crucial to the chaos analysis performed, because the system must be operating at steady state.

- When investigating the systems sensitivity to initial conditions, smaller changes should have been used. The smaller the change, the better.

5.5 Recommendations for Future Research

- Longer duration simulation should be run in order to fully analyze and find any chaos present.
- Changing the initial conditions by 1 degree was significantly too large. Much smaller changes should be made and an analysis of the sensitivity to initial conditions can be done.
- Utilizing a purely sinusoidal excitation force sufficed to excite the chaotic system; however, adding a sinusoidal pitching moment or linking the fluid structure interaction to the input force would provide a more realistic representation of the aerodynamic forces acting on the system.
- Selecting input parameters for the Lyapunov exponent code was purely based on intuition and one example based on a different experiment. A proper parameter study would provide better results when comparing exponents between simulations.

Appendix A. XFoil Command Line Commands

```
% Lt Lee Joshua J
% XFoil input code
% Each command is typed into the command line
%
% Calling the airfoil name
NACA 0012
% Changing the chord length
gdes
scal
10
exec
% return - press enter here
% Changing the number of panels to the max
pane
ppar
n
354
% Setting up the polar
oper
re
110026.61
mach
0.1885
visc
pacc
% return
% return
alfa
0
% Writing the cp data to a file
cpwr
filename.dat
% coefficient data
plis
% pressure vector plot
cpv
% return
quit
```

Appendix B. National Advisory Committee for Aeronautics (NACA) 0012 Defining Points

NACA 0012 AIRFOILS
66. 66.

0.0000000 0.0000000
0.0005839 0.0042603
0.0023342 0.0084289
0.0052468 0.0125011
0.0093149 0.0164706
0.0145291 0.0203300
0.0208771 0.0240706
0.0283441 0.0276827
0.0369127 0.0311559
0.0465628 0.0344792
0.0572720 0.0376414
0.0690152 0.0406310
0.0817649 0.0434371
0.0954915 0.0460489
0.1101628 0.0484567
0.1257446 0.0506513
0.1422005 0.0526251
0.1594921 0.0543715
0.1775789 0.0558856
0.1964187 0.0571640
0.2159676 0.0582048
0.2361799 0.0590081
0.2570083 0.0595755
0.2784042 0.0599102
0.3003177 0.0600172
0.3226976 0.0599028
0.3454915 0.0595747
0.3686463 0.0590419
0.3921079 0.0583145
0.4158215 0.0574033
0.4397317 0.0563200
0.4637826 0.0550769
0.4879181 0.0536866
0.5120819 0.0521620
0.5362174 0.0505161
0.5602683 0.0487619
0.5841786 0.0469124
0.6078921 0.0449802
0.6313537 0.0429778
0.6545085 0.0409174
0.6773025 0.0388109
0.6996823 0.0366700

0.7215958 0.0345058
0.7429917 0.0323294
0.7638202 0.0301515
0.7840324 0.0279828
0.8035813 0.0258337
0.8224211 0.0237142
0.8405079 0.0216347
0.8577995 0.0196051
0.8742554 0.0176353
0.8898372 0.0157351
0.9045085 0.0139143
0.9182351 0.0121823
0.9309849 0.0105485
0.9427280 0.0090217
0.9534372 0.0076108
0.9630873 0.0063238
0.9716559 0.0051685
0.9791229 0.0041519
0.9854709 0.0032804
0.9906850 0.0025595
0.9947532 0.0019938
0.9976658 0.0015870
0.9994161 0.0013419
1.0000000 0.0012600

0.0000000 0.0000000
0.0005839 -.0042603
0.0023342 -.0084289
0.0052468 -.0125011
0.0093149 -.0164706
0.0145291 -.0203300
0.0208771 -.0240706
0.0283441 -.0276827
0.0369127 -.0311559
0.0465628 -.0344792
0.0572720 -.0376414
0.0690152 -.0406310
0.0817649 -.0434371
0.0954915 -.0460489
0.1101628 -.0484567
0.1257446 -.0506513
0.1422005 -.0526251
0.1594921 -.0543715
0.1775789 -.0558856
0.1964187 -.0571640
0.2159676 -.0582048
0.2361799 -.0590081
0.2570083 -.0595755
0.2784042 -.0599102
0.3003177 -.0600172
0.3226976 -.0599028
0.3454915 -.0595747

0.3686463 -.0590419
0.3921079 -.0583145
0.4158215 -.0574033
0.4397317 -.0563200
0.4637826 -.0550769
0.4879181 -.0536866
0.5120819 -.0521620
0.5362174 -.0505161
0.5602683 -.0487619
0.5841786 -.0469124
0.6078921 -.0449802
0.6313537 -.0429778
0.6545085 -.0409174
0.6773025 -.0388109
0.6996823 -.0366700
0.7215958 -.0345058
0.7429917 -.0323294
0.7638202 -.0301515
0.7840324 -.0279828
0.8035813 -.0258337
0.8224211 -.0237142
0.8405079 -.0216347
0.8577995 -.0196051
0.8742554 -.0176353
0.8898372 -.0157351
0.9045085 -.0139143
0.9182351 -.0121823
0.9309849 -.0105485
0.9427280 -.0090217
0.9534372 -.0076108
0.9630873 -.0063238
0.9716559 -.0051685
0.9791229 -.0041519
0.9854709 -.0032804
0.9906850 -.0025595
0.9947532 -.0019938
0.9976658 -.0015870
0.9994161 -.0013419
1.0000000 -.0012600

Appendix C. Matlab Code

```
1 %% ...
   -----%
2
3 % Author : Lt Lee, Joshua J
4 % Last updated : 13 Feb 2017
5
6 % Must have the following files in the working directory to run this ...
   code.
7 % undeformed_CP.dat
8 % FourthTry2.cdb
9 % deformed3.txt THRU deformed8.txt
10 % coarse_mesh_kw.xy
11 % medium_mesh_ke.xy
12 % medium_mesh_kw.xy
13 % fine_mesh_ke.xy
14 % fine_mesh_kw.xy
15 % very_extra_fine_kw.xy
16 % undeformed.xy
17
18 % Utilized structured variables and cleans up variables that are no ...
   longer
19 % needed after each section.
20
21 %% ...
   -----%
22
23 % Starting the code
24 clear
25 clc
26 close all
27
28 %% ...
   -----%
29
30 % IMPORTING CP DATA FROM XFOIL, INPUTTING CONSTANTS, CALCULATING AIRFOIL
31 % UNDEFORMED SKIN AND PLOTTING ALL THE ABOVE
32
33 % Importing data specific .dat file
34 A = importdata('undeformed_CP.dat');
35
36 % L should be 354 (max number of nodes XFOIL allows)
37 L = length(A.data);
38
39 % Ripping off Cp from the data [ xupper cpupper xlower cplower ]
40 undeformed.Cp = [flipud(A.data(1:L/2,:)) A.data(L/2+1:end,:)];
41
42 % chord length
43 constants.C = 10;
```

```

44
45 % psi (dynamic pressure)
46 constants.Q = 40.6177/144;
47
48 % psi (free stream pressure)
49 constants.Pinf = 1632.93/144;
50
51 % chord vector [ x y ]
52 chord = [undeformed.Cp(:,1) zeros(L/2,1)];
53
54 % airfoil thickness
55 thickness = ...
    5*.12*constants.C*(0.2969*sqrt(chord(:,1)./constants.C)-0.1260*...
56     chord(:,1)./constants.C-0.3516*(chord(:,1)./constants.C).^2+0.2843...
57     *(chord(:,1)./constants.C).^3-0.1015*(chord(:,1)./constants.C).^4);
58
59 % skin vector [ xupper yupper xlower ylower]
60 undeformed.skin = [chord(:,1) (chord(:,2)+thickness) chord(:,1) ...
    (chord(:,2)-thickness)];
61
62 % Clearing unrequired variables
63 clear A L xL thickness chord
64
65 %% ...
    -----%
66
67 % CALCULATING VECTORS THAT ARE NORMAL TO THE AIRFOIL SURFACE, ...
    PLOTTING THE
68 % VECTORS, MULTIPLYING CP AND PRESSURE TIMES THE VECTORS TO PLOT THEM
69 % NORMAL TO THE SURFACE
70
71 % Calculating normal vectors along the airfoil
72 % [ dxupper dupper dxlower dylower ]
73 for a = 1:(length(undeformed.skin)-1)
74     undeformed.normal(a,:) = ...
        [-(undeformed.skin(a+1,2)-undeformed.skin(a,2))...
75         (undeformed.skin(a+1,1)-undeformed.skin(a,1))...
76         (undeformed.skin(a+1,4)-undeformed.skin(a,4))...
77         -(undeformed.skin(a+1,3)-undeformed.skin(a,3))];
78     % normalizing vectors to become unit length and face inward
79     undeformed.normal(a,:) = ...
        -undeformed.normal(a,:)/norm(undeformed.normal(a,:));
80 end
81
82 % Calculating Pressure from Cp and Q (dynamic pressure)
83 % [ xupper Pupper xlower Plower ]
84 undeformed.P(:, :) = undeformed.Cp(:, :);
85 undeformed.P(:, [2 4]) = undeformed.P(:, [2 4])*constants.Q + ...
    constants.Pinf;
86
87 % Clearing unrequired variables
88 clear a xL

```

```

89
90 %% ...
    -----%
91
92 % CODE FOR THE FEM.  READS IN THE .CDB FILE EXPORTED FROM HYPERWORKS TO
93 % GRAB THE X AND Y LOCATIONS OF THE NODES ALONG THE OUTER EDGE OF THE
94 % AIRFOIL, CALCULATE THE PRESSURE ACTING NORMAL TO THE SURFACE AT ...
    THE NODE,
95 % THEN WRITES THE .FEM CODE TO INPUT THE SOLVER DECK WITH FORCES.
96
97 % Reading in the X and Y positions of nodes to calculate and place ...
    pressure forces
98 fid = fopen('FourthTry2.cdb');
99 A = textscan(fid, '%f%f%f%f%f', 'Delimiter', ' ...
    ', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 11);
100 fclose(fid);
101
102 % The node numbers are gotten by displaying the node numbers along a ...
    path
103 % in hyperworks and writing them in the code.  I was unable to ...
    figure out
104 % how to automate this.  It is completely dependent upon them mesh.
105 constants.nodes = [44149 43579:1:43862 36916:1:36925 35551:1:35690 ...
    35320:1:35364 ...
106     24836:1:24838 24550:1:24690 13541:1:13548 13159:1:13344 ...
    14123:1:14126 ...
107     13740:1:13928 25266:1:25270 24982:1:25120 36232:1:36245 ...
    35875:1:36045 ...
108     36948:1:36956 36936 44718:1:44720 44153:1:44433];
109
110 % Ripping off the nodes numbers, x and y locations from textscan ...
    variable
111 % [ nodenumber x y ]
112 undeformed.hyperworksalldata(:,1) = A{1}(:,1); % node numbers
113 undeformed.hyperworksalldata(:,2) = A{1}(:,4); % node x location
114 undeformed.hyperworksalldata(:,3) = A{1}(:,5); % node y location
115
116 % Preallocating to calculate the pressure acting on just the skin nodes
117 % [ nodenumber Px Py ]
118 undeformed.hyperworksP(:, [1 2 3]) = zeros(length(constants.nodes), 3);
119 undeformed.hyperworksP(:,1) = constants.nodes';
120
121 % In this loop, the forces acting on the nodes due to the pressure are
122 % calculated.
123 for a = 1:length(constants.nodes)-1
124     % search all data for the first node on the exterior
125     one = find(undeformed.hyperworksalldata(:,1) == constants.nodes(a));
126     % search for the next node
127     two = find(undeformed.hyperworksalldata(:,1) == ...
    constants.nodes(a+1));
128     % grab the x position of the first node
129     X1 = undeformed.hyperworksalldata(one,2);

```

```

130 % grab the y position of the first node
131 Y1 = undeformed.hyperworksalldata(one,3);
132 % grab the x position of the second node
133 X2 = undeformed.hyperworksalldata(two,2);
134 % grab the y position of the second node
135 Y2 = undeformed.hyperworksalldata(two,3);
136 % calculate the distance between the two nodes
137 l = sqrt((X2-X1)^2+(Y2-Y1)^2);
138 % This if statement allows the upper and lower surface to both go
139 % through the same loop. In this loop, the pressure is ...
    calculated at
140 % the midpoint between the two nodes and is later assumed to be
141 % constant over the surface between the nodes. The vector ...
    normal to
142 % the surface at the midpoint is also calculated.
143 if X1>X2
144     P = ...
        interp1(undeformed.P(:,1),undeformed.P(:,2),(X1-X2)/2+X2,'linear','extrap');
145     Normal = [-(Y1-Y2) (X1-X2)];
146 elseif X2>X1
147     P = ...
        interp1(undeformed.P(:,1),undeformed.P(:,2),(X2-X1)/2+X1,'linear','extrap');
148     Normal = [(Y2-Y1) -(X2-X1)];
149 end
150 % making normal vector a unit normal vector
151 Normal = -Normal./norm(Normal);
152 % calculating the equivalent nodal force from the pressure
153 F = P*l/2*Normal;
154 % indexing where in the pressure vector the force data should go
155 one = find(undeformed.hyperworksP(:,1) == constants.nodes(a));
156 % indexing where in the pressure vector the force data should go
157 two = find(undeformed.hyperworksP(:,1) == constants.nodes(a+1));
158 % placing the force vector for the first node
159 undeformed.hyperworksP(one,[2 3]) = ...
    undeformed.hyperworksP(one,[2 3])+F;
160 % placing the force vector for the second node
161 undeformed.hyperworksP(two,[2 3]) = ...
    undeformed.hyperworksP(two,[2 3])+F;
162 end
163
164 % Printing the .fem code to the command line to copy and paste into the
165 % fem. This format is based on previously exported .fem files with ...
    forces.
166 fprintf('FORCE          1%8i          01.0          %-7.4f %-7.4f ...
    0.0\n',undeformed.hyperworksP(:,[1 2 3]));
167
168 % Clearing unrequired variables
169 clear a A ans F fid l Normal one P two X1 X2 Y1 Y2
170 clc
171
172 %% ...
    -----%
```

```

173
174 % CALCULATING THE REQUIRED VOLUME FRACTION FOR THE FEM.  THIS IS ...
    BASED ON
175 % OPTISTRUCTS FORMULATION OF THE VOLUME FRACTION, THE ACTUAL AREA IN THE
176 % FEM USED, THE DENSITY OF ALUMINUM, AND THE DENSITY PER SPAN ...
    PUBLISHED BY
177 % CONNER IN 1997.
178
179 % undeformed.hyperworks.outernodes = [ nodenumber xpos ypos ]
180 % undeformed.hyperworks.innernodes = [ nodenumber xpos ypos ]
181 % undeformed.hyperworks.skinarea    = [ totalairfoilarea ]
182 % undeformed.hyperworks.innerarea   = [ designspacearea ]
183 % undeformed.hyperworks.VF          = [ volumefraction ]
184
185 % Nodes of the upper outer and inner surfaces of the skin
186 undeformed.hyperworksouternodes = [44149 43579:1:43862 36916:1:36925 ...
    35551:1:35690 35320:1:35364 24836:1:24838 24550:1:24690 ...
    13541:1:13548 13159:1:13344]';
187 undeformed.hyperworksinnernodes = [37416:1:37434 36967:1:37232 ...
    36459:1:36468 25773:1:25777 25414:1:25593 14658:1:14660 ...
    14328:1:14468 425:1:432 86:1:271]';
188
189 % Grabbing x and y positions to calculate area
190 for a = 1:length(undeformed.hyperworksouternodes)
191     one = find(undeformed.hyperworksalldata(:,1) == ...
        undeformed.hyperworksouternodes(a));
192     undeformed.hyperworksouternodes(a,[2 3]) = ...
        undeformed.hyperworksalldata(one,[2 3]);
193 end
194 for b = 1:length(undeformed.hyperworksinnernodes)
195     one = find(undeformed.hyperworksalldata(:,1) == ...
        undeformed.hyperworksinnernodes(b));
196     undeformed.hyperworksinnernodes(b,[2 3]) = ...
        undeformed.hyperworksalldata(one,[2 3]);
197 end
198
199 % Calculating area by numerically integrating
200 undeformed.hyperworkssskinarea = ...
    (-2*trapz(undeformed.hyperworksouternodes(:,2),undeformed.hyperworksouternodes(:,3))
    ...
201                                     + ...
    2*trapz(undeformed.hyperworksinnernodes(:,2),undeformed.hyperworksinnernodes(:,3)))
202 undeformed.hyperworksinnerarea = ...
    -2*trapz(undeformed.hyperworksinnernodes(:,2),undeformed.hyperworksinnernodes(:,3))*
203
204 % Calculating area ratio of 1in chord to 10in chord (Lee to Connor) ...
    -- No
205 % longer necessary, Lee model changed to 10 in chord (21 Dec 2016)
206 % C1 = 1;
207 % C2 = 10;
208 % chord1 = linspace(0,C1,1000);
209 % chord2 = linspace(0,C2,1000);

```

```

210 % thickness1 = ...
      5*.12*C1*(0.2969*sqrt(chord1./C1)-0.1260*chord1./C1-0.3516*(chord1./C1).^2+0.2843*(c
211 % thickness2 = ...
      5*.12*C2*(0.2969*sqrt(chord2./C2)-0.1260*chord2./C2-0.3516*(chord2./C2).^2+0.2843*(c
212 % Alee = 2*trapz(chord1,thickness1);
213 % Aconner = 2*trapz(chord2,thickness2);
214
215 % Calculating volume fraction (percent)
216 undeformed.hyperworksVF = ...
      100*(1.558/2712-undeformed.hyperworksskinarea)/undeformed.hyperworksinnerarea;
217
218 % Clearing unrequired variables
219 clear a b one
220
221 %% ...
      -----%
222
223 % CALCULATING THE FORCE ACTING ON THE HIGHEST STRESS BEAM (MOST ...
      SUSEPTIBLE
224 % TO BUCKLING) AS DETERMINED BY OPTISTRUCT, THEN CALCULATING THE ...
      REQUIRED
225 % WIDTH FOR THE BEAM TO NOT BUCKLE.
226
227 % THE DISTANCES WERE DETERMINED BY LOADING A PICTURE OF THE OPTIMIZED
228 % STRUCTURE INTO THE SKETCH PLANE, ENSURING THE CHORD LENGTH IS 10 ...
      INCHES,
229 % AND MEASUREING THE FREE STANDING LENGTH ALONG WITH THE WIDTH OF ...
      THE BEAM.
230 % FOR AN EXAMPLE, PLEASE LOOK AT BEAMBUCKLINGMEASUREMENTS.PNG.
231
232 % Measurements from Solidworks and other constants
233 undeformed.bucklingw = 0.011;
234 undeformed.bucklingl = 0.86;
235 undeformed.bucklingt = 1;
236 constants.Ealum = 10E6;
237
238 % Calculating moment of inertia
239 undeformed.bucklingIz = undeformed.bucklingt*undeformed.bucklingw^3/12;
240 undeformed.bucklingIx = undeformed.bucklingw*undeformed.bucklingt^3/12;
241
242 % The smallest moment of inertia will give the smalled Pcr, or the ...
      smallest
243 % force required to buckle the beam. This is the force we are concerned
244 % with. I am using the hinged hinged boundary condition to calculate ...
      Pcr.
245 if undeformed.bucklingIz > undeformed.bucklingIx
246     undeformed.bucklingPcr = ...
        pi^2*constants.Ealum*undeformed.bucklingIx/(undeformed.bucklingl^2);
247 else
248     undeformed.bucklingPcr = ...
        pi^2*constants.Ealum*undeformed.bucklingIz/(undeformed.bucklingl^2);
249 end

```

```

250 undeformed.bucklingstress = undeformed.bucklingPcr / ...
    (undeformed.bucklingw*undeformed.bucklingt);
251
252 % Clearing unrequired variables
253 clear a
254
255 %% ...
    -----%
256
257 % AT THIS POINT, I EXPORTED THE DEFORMED SHAPE OF THE AIRFOIL AFTER
258 % OPTIMIZATION FROM HYPERVIEW (ABAQUS FORMAT STYLE). THAT FILE IS THE
259 % DEFORMED.TXT FILE IN THE DIRECTORY. FIRST, THE NODE NUMBERS AND
260 % LOCATIONS WILL BE PULLED FROM THE .TXT FILE, THEN BY USING THE OUTER
261 % NODES PREVIOUSLY USED BEFORE, THE DEFORMED OUTER SKIN CAN BE USED ...
    TO SEE
262 % IF THE AERODYNAMICS HAVE CHANGED IN XFOIL.
263
264 % SO BY NOW I REALIZED THAT THE CG IS TOO FAR OFF FROM WHAT WAS ...
    CALCULATED
265 % IN CONNER'S PAPER AND COULD USE COG AS A CONSTRAINT IN THE ...
    OPTIMISATION.
266 % THE FOLLOWING IS COMPARING THE DEFORMED STRUCTURES (SCALED ...
    EQUIVILENTLY)
267 % AND WRITING DEFORMED .TXT FILES FOR XFOIL.
268
269 % 1 - does not exist
270 % 2 - incorrect forces, but file still exists
271
272 % 3 - No CG constraint
273 % Opening and textscanning the file
274 fid = fopen('deformed3.txt');
275 A = ...
    textscan(fid, '%f%f%f', 'Delimiter', ',', 'MultipleDelimsAsOne', true, 'CollectOutput', 1
276 fclose(fid);
277 % Ripping off the data from textscan throw away variable information
278 % [ nodenumber xpos ypos ]
279 deformed.nocg.hyperworksalldata(:,1) = A{1}(:,1);
280 deformed.nocg.hyperworksalldata(:,2) = A{1}(:,2);
281 deformed.nocg.hyperworksalldata(:,3) = A{1}(:,3);
282
283 % 4 - CG constrained from 4.60 to 4.70
284 % Opening and textscanning the file
285 fid = fopen('deformed4.txt');
286 A = ...
    textscan(fid, '%f%f%f', 'Delimiter', ',', 'MultipleDelimsAsOne', true, 'CollectOutput', 1
287 fclose(fid);
288 % Ripping off the data from textscan throw away variable information
289 % [ nodenumber xpos ypos ]
290 deformed.cg460.hyperworksalldata(:,1) = A{1}(:,1);
291 deformed.cg460.hyperworksalldata(:,2) = A{1}(:,2);
292 deformed.cg460.hyperworksalldata(:,3) = A{1}(:,3);
293

```

```

294 % 5 - CG constrained from 4.65 to 4.69
295 % Opening and textscanning the file
296 fid = fopen('deformed5.txt');
297 A = ...
    textscan(fid, '%f%f%f', 'Delimiter', ',', 'MultipleDelimsAsOne', true, 'CollectOutput', 1)
298 fclose(fid);
299 % Ripping off the data from textscan throw away variable information
300 % [ nodenumber xpos ypos ]
301 deformed.cg465.hyperworksalldata(:,1) = A{1}(:,1);
302 deformed.cg465.hyperworksalldata(:,2) = A{1}(:,2);
303 deformed.cg465.hyperworksalldata(:,3) = A{1}(:,3);
304
305 % 6 - CG constrained from 4.00 to 5.00
306 % Opening and textscanning the file
307 fid = fopen('deformed6.txt');
308 A = ...
    textscan(fid, '%f%f%f', 'Delimiter', ',', 'MultipleDelimsAsOne', true, 'CollectOutput', 1)
309 fclose(fid);
310 % Ripping off the data from textscan throw away variable information
311 % [ nodenumber xpos ypos ]
312 deformed.cg400.hyperworksalldata(:,1) = A{1}(:,1);
313 deformed.cg400.hyperworksalldata(:,2) = A{1}(:,2);
314 deformed.cg400.hyperworksalldata(:,3) = A{1}(:,3);
315
316 % 7 - CG constrained from 1.00 to 9.00
317 % Opening and textscanning the file
318 fid = fopen('deformed7.txt');
319 A = ...
    textscan(fid, '%f%f%f', 'Delimiter', ',', 'MultipleDelimsAsOne', true, 'CollectOutput', 1)
320 fclose(fid);
321 % Ripping off the data from textscan throw away variable information
322 % [ nodenumber xpos ypos ]
323 deformed.cg100.hyperworksalldata(:,1) = A{1}(:,1);
324 deformed.cg100.hyperworksalldata(:,2) = A{1}(:,2);
325 deformed.cg400.hyperworksalldata(:,3) = A{1}(:,3);
326
327 % 8 - CG constrained from 4.67 to 4.69
328 % Opening and textscanning the file
329 fid = fopen('deformed8.txt');
330 A = ...
    textscan(fid, '%f%f%f', 'Delimiter', ',', 'MultipleDelimsAsOne', true, 'CollectOutput', 1)
331 fclose(fid);
332 % Ripping off the data from textscan throw away variable information
333 % [ nodenumber xpos ypos ]
334 deformed.cg467.hyperworksalldata(:,1) = A{1}(:,1);
335 deformed.cg467.hyperworksalldata(:,2) = A{1}(:,2);
336 deformed.cg467.hyperworksalldata(:,3) = A{1}(:,3);
337
338 % Using the node numbers to grab just the outer skin
339 for a = 1:length(constants.nodes)
340     one = find(deformed.nocg.hyperworksalldata(:,1) == ...
        constants.nodes(a));

```

```

341     two = find(deformed.cg460.hyperworksalldata(:,1) == ...
342               constants.nodes(a));
343     three = find(deformed.cg465.hyperworksalldata(:,1) == ...
344                 constants.nodes(a));
345     four = find(undeformed.hyperworksalldata(:,1) == ...
346                constants.nodes(a));
347     five = find(deformed.cg400.hyperworksalldata(:,1) == ...
348                 constants.nodes(a));
349     six = find(deformed.cg100.hyperworksalldata(:,1) == ...
350                constants.nodes(a));
351     seven = find(deformed.cg467.hyperworksalldata(:,1) == ...
352                  constants.nodes(a));
353     deformed.nocg.hyperworksskin(a,:) = ...
354         deformed.nocg.hyperworksalldata(one,:);
355     deformed.cg460.hyperworksskin(a,:) = ...
356         deformed.cg460.hyperworksalldata(two,:);
357     deformed.cg465.hyperworksskin(a,:) = ...
358         deformed.cg465.hyperworksalldata(three,:);
359     undeformed.hyperworksskin(a,:) = ...
360         undeformed.hyperworksalldata(four,:);
361     deformed.cg400.hyperworksskin(a,:) = ...
362         deformed.cg400.hyperworksalldata(five,:);
363     deformed.cg100.hyperworksskin(a,:) = ...
364         deformed.cg100.hyperworksalldata(six,:);
365     deformed.cg467.hyperworksskin(a,:) = ...
366         deformed.cg467.hyperworksalldata(seven,:);
367 end
368
369 % Calculating the displacement in the x and y directions and then ...
370 % scaling
371 % them, then adding them back.
372 % .displacement -> [ undeformed+scale*deltaX undeformed+scale*deltaY ]
373 constants.scale = 300;
374 deformed.nocg.hyperworksscaled = ...
375     constants.scale*(deformed.nocg.hyperworksskin(:, [2 3]) - ...
376                     undeformed.hyperworksskin(:, [2 ...
377               3]))+undeformed.hyperworksskin(:, [2 3]);
378 deformed.cg460.hyperworksscaled = ...
379     constants.scale*(deformed.cg460.hyperworksskin(:, [2 3]) - ...
380                     undeformed.hyperworksskin(:, [2 ...
381               3]))+undeformed.hyperworksskin(:, [2 3]);
382 deformed.cg465.hyperworksscaled = ...
383     constants.scale*(deformed.cg465.hyperworksskin(:, [2 3]) - ...
384                     undeformed.hyperworksskin(:, [2 ...
385               3]))+undeformed.hyperworksskin(:, [2 3]);
386 deformed.cg467.hyperworksscaled = ...
387     constants.scale*(deformed.cg467.hyperworksskin(:, [2 3]) - ...
388                     undeformed.hyperworksskin(:, [2 ...
389               3]))+undeformed.hyperworksskin(:, [2 3]);
390
391 % Clearing unrequired variables
392 clear A ans a fid one two three four five six seven

```

```

367
368 %% ...
      -----%
369
370 % HERE IS WHERE I STARTED BUILDING THE MODELS AND WANTED TO HAVE ...
      .TXT FILES
371 % OF POINTS FOR THE INNER AND OUTER SURFACES OF THE SKIN.  I'M ...
      TAKING A
372 % SHORTCUT, SINCE IT IS A SYMMETRIC AIRFOIL... I WILL ONLY BE USING ...
      POINTS
373 % FOR THE UPPER SURFACE AND REFLECTING/MIRRORING.
374
375 % Writing input .txt for solidworks // outerskin
376 fid = fopen('undeformedouterskin.txt','w');
377 dlmwrite('undeformedouterskin.txt',[undeformed.hyperworksouternodes(:,[2 ...
      3]) ...
      zeros(length(undeformed.hyperworksouternodes),1)], 'delimiter', ' ')
378 fclose(fid);
379
380 % Writing input .txt for solidworks // innerskin
381 fid = fopen('undeformedinnerskin.txt','w');
382 dlmwrite('undeformedinnerskin.txt',[undeformed.hyperworksinnernodes(:,[2 ...
      3]) ...
      zeros(length(undeformed.hyperworksinnernodes),1)], 'delimiter', ' ')
383 fclose(fid);
384
385 % Clearing unrequired variables
386 clear fid ans
387
388 %% ...
      -----%
389
390 % I SAW HOW JAGGED THE CP OUTPUT WAS FOR DEFORMED CASES SO I LEANED ...
      ON A
391 % FRIEND TO RUN THE CFD IN ANSYS FLUENT.  I HAVE THREE DIFFERENCE GRIDS
392 % WITH TWO DIFFERENCE TURBULENCE MODELS.
393
394 % Writting [x y z] file for pointwise // deformed and undeformed
395 deformed.cg467.hyperworksskin(:,4) = ...
      zeros(length(deformed.cg467.hyperworksskin(:,1)),1);
396 undeformed.hyperworksskin(:,4) = ...
      zeros(length(undeformed.hyperworksskin(:,1)),1);
397 dlmwrite('deformedNACA8.dat',deformed.cg467.hyperworksskin(:,[2 3 ...
      4]), 'delimiter', ' ')
398 dlmwrite('undeformedNACA.dat',undeformed.hyperworksskin(:,[2 3 ...
      4]), 'delimiter', ' ')
399
400 % 1
401 % Coarse Mesh with k-omega turbulence model lower surface
402 fid = fopen('coarse_mesh.kw.xy');
403 A = textscan(fid,'%f%f','Delimiter',' ...
      ','MultipleDelimsAsOne',true,'CollectOutput',1,'HeaderLines',4);

```

```

404 fclose(fid);
405 % Coarse Mesh with k-omega turbulence model upper surface
406 fid = fopen('coarse_mesh_kw.xy');
407 B = textscan(fid, '%f%f', 'Delimiter', ' ...
    ', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 827);
408 fclose(fid);
409 % Saving the data into a structured variable and converting the x ...
    position
410 CFD.coarsekwlower = [ A{1}(:,1)/0.0254 A{1}(:,2) ];
411 CFD.coarsekwupper = [ B{1}(:,1)/0.0254 B{1}(:,2) ];
412
413 % 2
414 % Medium Mesh with k-omega turbulence model lower surface
415 fid = fopen('medium_mesh_kw.xy');
416 A = textscan(fid, '%f%f', 'Delimiter', ' ...
    ', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 4);
417 fclose(fid);
418 % Medium Mesh with k-omega turbulence model upper surface
419 fid = fopen('medium_mesh_kw.xy');
420 B = textscan(fid, '%f%f', 'Delimiter', ' ...
    ', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 1643);
421 fclose(fid);
422 % Saving the data into a structured variable and converting the x ...
    position
423 CFD.mediumkw = [ A{1}(:,1)/0.0254 A{1}(:,2) B{1}(:,1)/0.0254 ...
    B{1}(:,2) ];
424
425 % 3
426 % Medium Mesh with k-epsilon turbulence model lower surface
427 fid = fopen('medium_mesh_ke.xy');
428 A = textscan(fid, '%f%f', 'Delimiter', ' ...
    ', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 4);
429 fclose(fid);
430 % Medium Mesh with k-epsilon turbulence model upper surface
431 fid = fopen('medium_mesh_ke.xy');
432 B = textscan(fid, '%f%f', 'Delimiter', ' ...
    ', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 1643);
433 fclose(fid);
434 % Saving the data into a structured variable and converting the x ...
    position
435 CFD.mediumke = [ A{1}(:,1)/0.0254 A{1}(:,2) B{1}(:,1)/0.0254 ...
    B{1}(:,2) ];
436
437 % 4
438 % Fine Mesh with k-omega turbulence model lower surface
439 fid = fopen('fine_mesh_kw.xy');
440 A = textscan(fid, '%f%f', 'Delimiter', ' ...
    ', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 4);
441 fclose(fid);
442 % Coarse Mesh with k-omega turbulence model upper surface
443 fid = fopen('fine_mesh_kw.xy');
444 B = textscan(fid, '%f%f', 'Delimiter', ' ...

```

```

        ', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 3279);
445 fclose(fid);
446 % Saving the data into a structured variable and converting the x ...
    position
447 CFD.finekw = [ A{1}(:,1)/0.0254 A{1}(:,2) B{1}(:,1)/0.0254 B{1}(:,2) ];
448
449 % 5
450 % Fine Mesh with k-epsilon turbulence model lower surface
451 fid = fopen('fine_mesh_ke.xy');
452 A = textscan(fid, '%f%f', 'Delimiter', ' ...
    ', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 4);
453 fclose(fid);
454 % Fine Mesh with k-epsilon turbulence model upper surface
455 fid = fopen('fine_mesh_ke.xy');
456 B = textscan(fid, '%f%f', 'Delimiter', ' ...
    ', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 3279);
457 fclose(fid);
458 % Saving the data into a structured variable and converting the x ...
    position
459 CFD.fineke = [ A{1}(:,1)/0.0254 A{1}(:,2) B{1}(:,1)/0.0254 B{1}(:,2) ];
460
461 % 6
462 % VERY EXTRA Fine Mesh with k-omega turbulence model lower surface
463 fid = fopen('very_extra_fine_kw.xy');
464 A = textscan(fid, '%f%f', 'Delimiter', ' ...
    ', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 4);
465 fclose(fid);
466 % VERY EXTRA Fine Mesh with k-omega turbulence model upper surface
467 fid = fopen('very_extra_fine_kw.xy');
468 B = textscan(fid, '%f%f', 'Delimiter', ' ...
    ', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 5008);
469 fclose(fid);
470 % Saving the data into a structured variable and converting the x ...
    position
471 CFD.veryfinekw = [ A{1}(:,1)/0.0254 A{1}(:,2)+1 B{1}(:,1)/0.0254 ...
    B{1}(:,2)+1 ];
472
473 % 7
474 % UNDEFORMED VERY EXTRA Fine Mesh with k-omega turbulence model ...
    lower surface
475 fid = fopen('undeformed.xy');
476 A = textscan(fid, '%f%f', 'Delimiter', ' ...
    ', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 4);
477 fclose(fid);
478 % UNDEFORMED VERY EXTRA Fine Mesh with k-omega turbulence model ...
    upper surface
479 fid = fopen('undeformed.xy');
480 B = textscan(fid, '%f%f', 'Delimiter', ' ...
    ', 'MultipleDelimsAsOne', true, 'CollectOutput', 1, 'HeaderLines', 5008);
481 fclose(fid);
482 % Saving the data into a structured variable and converting the x ...
    position

```

```

483 CFD.undeformedveryfinekw = [ A{1}(:,1)/0.0254 A{1}(:,2)+1 ...
    B{1}(:,1)/0.0254 B{1}(:,2)+1 ];
484
485 % Clearing unrequired variables
486 clear A ans B fid
487
488 %% ...
    -----%
489
490 % ANALYZING THE DYNAMIC FEM
491
492 % Uploading Displacement and Velocity Data
493 u4_l = importdata('4_L.U2.dat');
494 v4_l = importdata('4_L.V2.dat');
495 u4_nl = importdata('4_NL.U2.dat');
496 v4_nl = importdata('4_NL.V2.dat');
497
498 u5_l = importdata('5_L.U2.dat');
499 v5_l = importdata('5_L.V2.dat');
500 u5_nl = importdata('5_NL.U2.dat');
501 v5_nl = importdata('5_NL.V2.dat');
502
503 u6_l = importdata('6_L.U2.dat');
504 v6_l = importdata('6_L.V2.dat');
505 u6_nl = importdata('6_NL.U2.dat');
506 v6_nl = importdata('6_NL.V2.dat');
507
508 u14_l = importdata('14_L.U2.dat');
509 v14_l = importdata('14_L.V2.dat');
510 u14_nl = importdata('14_NL.U2.dat');
511 v14_nl = importdata('14_NL.V2.dat');
512
513 u15_l = importdata('15_L.U2.dat');
514 v15_l = importdata('15_L.V2.dat');
515 u15_nl = importdata('15_NL.U2.dat');
516 v15_nl = importdata('15_NL.V2.dat');
517
518 u16_l = importdata('16_L.U2.dat');
519 v16_l = importdata('16_L.V2.dat');
520 u16_nl = importdata('16_NL.U2.dat');
521 v16_nl = importdata('16_NL.V2.dat');
522
523 % Adjusting Displacement Data for change in initial conditions
524 u4_l(:,2) = u4_l(:,2)-0.0436;
525 u4_nl(:,2) = u4_nl(:,2)-0.0436;
526
527 u6_l(:,2) = u6_l(:,2)+0.0436;
528 u6_nl(:,2) = u6_nl(:,2)+0.0436;
529
530 u14_l(:,2) = u14_l(:,2)-0.0436;
531 u14_nl(:,2) = u14_nl(:,2)-0.0436;
532

```

```

533 u16_l(:,2) = u16_l(:,2)+0.0436;
534 u16_n1(:,2) = u16_n1(:,2)+0.0436;
535
536 %% ...
537 -----%
538 % Consolodating plots to one section.
539
540 % Plotting Airfoil
541 figure%('units','normalized','outerposition',[0 0 1 1])
542 plot(undeformed.skin(:,1),undeformed.skin(:,2),'b',undeformed.skin(:,3),undeformed.skin
    % airfoil skin
543 grid minor
544 xL = xlim;
545 line(xL, [0 0], 'Color','k'); % x axis, but also the chord
546 axis([0 constants.C -1 1])
547 axis equal
548 ylabel('Thickness - Y ...
    (in)', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold')
549 xlabel('Chord - X ...
    (in)', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold');
550 legend('UpperSurface', 'LowerSurface')
551 set(gca, 'FontSize', 12)
552
553 % Plotting Cp along the chord
554 figure%('units','normalized','outerposition',[0 0 1 1])
555 plot(undeformed.Cp(:,1),undeformed.Cp(:,2),'b',undeformed.Cp(:,3),undeformed.Cp(:,4),'r
    % Cp
556 set(gca, 'Ydir', 'reverse') % reverse y axis, because that is how cp ...
    is plotted
557 grid minor
558 xL = xlim;
559 line(xL, [0 0], 'Color','k'); % x axis
560 axis([0 constants.C -1 1])
561 ylabel('$C_P$', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold')
562 xlabel('Chord - X ...
    (in)', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold');
563 legend('UpperSurface', 'LowerSurface')
564
565 % Plotting Pressure Vectors Acting on the Airfoil
566 figure%('units','normalized','outerposition',[0 0 1 1])
567 hold on;
568 plot(undeformed.skin(:,1),undeformed.skin(:,2),'b',undeformed.skin(:,3),undeformed.skin
    % airfoil skin
569 quiver(undeformed.skin(1:end-1,1)-undeformed.normal(:,1).*undeformed.P(1:end-1,2),undef
570         undeformed.normal(:,1).*undeformed.P(1:end-1,2),undeformed.normal(:,2).*undeformed
    % vectors on upper surface
571 quiver(undeformed.skin(1:end-1,3)-undeformed.normal(:,3).*undeformed.P(1:end-1,4),undef
572         undeformed.normal(:,3).*undeformed.P(1:end-1,4),undeformed.normal(:,4).*undeformed
    % vectors on lower surface
573 xL = xlim;
574 line(xL, [0 0], 'Color','k'); %x-axis

```

```

575 hold off
576 axis equal
577 grid minor
578 ylabel('Thickness - Y ...
      (in)', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold')
579 xlabel('Chord - X ...
      (in)', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold');
580 legend('UpperSurface', 'LowerSurface')
581 set(gca, 'FontSize', 12)
582
583 % plotting all three outer skins just to see if they look different at
584 % scale 1 (the answer is no)
585 figure%('units', 'normalized', 'outerposition', [0 0 1 1])
586 hold on;
587 plot(deformed.nocg.hyperworksskin(:,2), deformed.nocg.hyperworksskin(:,3), 'r')
588 plot(deformed.cg460.hyperworksskin(:,2), deformed.cg460.hyperworksskin(:,3), 'b')
589 plot(deformed.cg465.hyperworksskin(:,2), deformed.cg465.hyperworksskin(:,3), 'm')
590 plot(deformed.cg467.hyperworksskin(:,2), deformed.cg467.hyperworksskin(:,3), 'g')
591 plot(undeformed.hyperworksskin(:,2), undeformed.hyperworksskin(:,3), 'k')
592 hold off;
593 grid minor
594 axis equal
595 legend('No Constraint', '4.6 - 4.7', '4.65 and - 4.69', '4.67 - ...
      4.69', 'Undeformed Skin', 'Location', 'East')
596 axis([0 2 -1 1]);
597 set(gca, 'FontSize', 12)
598 ylabel('Thickness - Y ...
      (in)', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold')
599 xlabel('Chord - X ...
      (in)', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold');
600
601 % plotting all three outer skins just to see if they look different at
602 figure%('units', 'normalized', 'outerposition', [0 0 1 1])
603 hold on;
604 plot(undeformed.hyperworksskin(:,2), undeformed.hyperworksskin(:,3), 'k')
605 plot(deformed.nocg.hyperworksscaled(:,1), deformed.nocg.hyperworksscaled(:,2), 'r')
606 plot(deformed.cg460.hyperworksscaled(:,1), deformed.cg460.hyperworksscaled(:,2), 'b')
607 plot(deformed.cg465.hyperworksscaled(:,1), deformed.cg465.hyperworksscaled(:,2), 'm')
608 plot(deformed.cg467.hyperworksscaled(:,1), deformed.cg467.hyperworksscaled(:,2), 'g')
609 hold off;
610 axis equal
611 grid minor
612 legend('Undeformed Skin', 'No Constraint', '4.6 - 4.7', '4.65 and - ...
      4.69', '4.67 - 4.69', 'Location', 'East')
613 axis([0 2 -1 1]);
614 set(gca, 'FontSize', 12)
615 ylabel('Thickness - Y ...
      (in)', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold')
616 xlabel('Chord - X ...
      (in)', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold');
617
618 % Comparing coarse, med and fine KW method

```

```

619 figure
620 hold on
621 plot(CFD.coarsekwlower(:,1), CFD.coarsekwlower(:,2), 'k');
622 plot(CFD.mediumkw(:, [1]), CFD.mediumkw(:, [2]), 'r');
623 plot(CFD.finekw(:, [1]), CFD.finekw(:, [2]), 'b');
624 set(gca, 'Ydir', 'reverse', 'FontSize', 12)
625 legend('Coarse', 'Medium', 'Fine')
626 grid minor
627 xL = xlim;
628 line(xL, [0 0], 'Color', 'k'); % x axis
629 axis([0 constants.C -1 1])
630 ylabel('$C_P$', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold')
631 xlabel('Chord - X ...
        (in)', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold');
632
633 % Comparing med and fine KE method
634 figure
635 hold on
636 plot(CFD.mediumke(:, [1]), CFD.mediumke(:, [2]), 'r');
637 plot(CFD.fineke(:, [1]), CFD.fineke(:, [2]), 'b');
638 hold off
639 set(gca, 'Ydir', 'reverse', 'FontSize', 12)
640 legend('Medium', 'Fine')
641 grid minor
642 xL = xlim;
643 line(xL, [0 0], 'Color', 'k'); % x axis
644 axis([0 constants.C -1 1])
645 ylabel('$C_P$', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold')
646 xlabel('Chord - X ...
        (in)', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold');
647
648 % Comparing Deformed and Undeformed Very Extra Fine
649 figure
650 hold on
651 plot(CFD.veryfinekw(:, [1]), CFD.veryfinekw(:, [2]), 'r');
652 plot(CFD.undeformedveryfinekw(:, [1]), CFD.undeformedveryfinekw(:, [2]), 'b');
653 hold off
654 set(gca, 'Ydir', 'reverse', 'FontSize', 12)
655 legend('Deformed', 'Undeformed')
656 grid minor
657 xL = xlim;
658 line(xL, [0 0], 'Color', 'k'); % x axis
659 axis([0 constants.C -1 1])
660 ylabel('$C_P$', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold')
661 xlabel('Chord - X ...
        (in)', 'Interpreter', 'LaTeX', 'FontSize', 14, 'FontWeight', 'bold');
662
663 % feeding into plotDVT
664 figure%('units', 'normalized', 'outerposition', [0 0 1 1])
665 plotDVT(u4_l(:, 1), u4_l(:, 2), u4_nl(:, 1), u4_nl(:, 2))
666 print('4_DVT', '-dpng')
667

```

```

668 figure%('units','normalized','outerposition',[0 0 1 1])
669 plotDVT(u5_l(:,1),u5_l(:,2),u5_nl(:,1),u5_nl(:,2))
670 print('5_DVT','-dpng')
671
672 figure%('units','normalized','outerposition',[0 0 1 1])
673 plotDVT(u6_l(:,1),u6_l(:,2),u6_nl(:,1),u6_nl(:,2))
674 print('6_DVT','-dpng')
675
676 figure%('units','normalized','outerposition',[0 0 1 1])
677 plotDVT(u14_l(:,1),u14_l(:,2),u14_nl(:,1),u14_nl(:,2))
678 print('14_DVT','-dpng')
679
680 figure%('units','normalized','outerposition',[0 0 1 1])
681 plotDVT(u15_l(:,1),u15_l(:,2),u15_nl(:,1),u15_nl(:,2))
682 print('15_DVT','-dpng')
683
684 figure%('units','normalized','outerposition',[0 0 1 1])
685 plotDVT(u16_l(:,1),u16_l(:,2),u16_nl(:,1),u16_nl(:,2))
686 print('16_DVT','-dpng')
687
688 % Feeding into plotDVT2
689 figure%('units','normalized','outerposition',[0 0 1 1])
690 plotDVT2(u4_l(:,1),u4_l(:,2),u5_l(:,1),u5_l(:,2),u6_l(:,1),u6_l(:,2))
691 print('L-aero_DVT','-dpng')
692
693 figure%('units','normalized','outerposition',[0 0 1 1])
694 plotDVT2(u4_nl(:,1),u4_nl(:,2),u5_nl(:,1),u5_nl(:,2),u6_nl(:,1),u6_nl(:,2))
695 print('NL-aero_DVT','-dpng')
696
697 figure%('units','normalized','outerposition',[0 0 1 1])
698 plotDVT2(u14_l(:,1),u14_l(:,2),u15_l(:,1),u15_l(:,2),u16_l(:,1),u16_l(:,2))
699 print('L-tip_DVT','-dpng')
700
701 figure%('units','normalized','outerposition',[0 0 1 1])
702 plotDVT2(u14_nl(:,1),u14_nl(:,2),u15_nl(:,1),u15_nl(:,2),u16_nl(:,1),u16_nl(:,2))
703 print('NL-tip_DVT','-dpng')
704
705 % Feeding into plotPSD
706 figure%('units','normalized','outerposition',[0 0 1 1])
707 plotPSD(u4_l(:,1),u4_l(:,2))
708 print('4_L_PSD','-dpng')
709 figure
710 plotPSD(u4_nl(:,1),u4_nl(:,2))
711 print('4_NL_PSD','-dpng')
712
713 figure
714 plotPSD(u5_l(:,1),u5_l(:,2))
715 print('5_L_PSD','-dpng')
716 figure
717 plotPSD(u5_nl(:,1),u5_nl(:,2))
718 print('5_NL_PSD','-dpng')
719

```

```

720 figure
721 plotPSD(u6_l(:,1),u6_l(:,2))
722 print('6_L_PSD','-dpng')
723 figure
724 plotPSD(u6_nl(:,1),u6_nl(:,2))
725 print('6_NL_PSD','-dpng')
726
727 figure
728 plotPSD(u14_l(:,1),u14_l(:,2))
729 print('14_L_PSD','-dpng')
730 figure
731 plotPSD(u14_nl(:,1),u14_nl(:,2))
732 print('14_NL_PSD','-dpng')
733
734 figure
735 plotPSD(u15_l(:,1),u15_l(:,2))
736 print('15_L_PSD','-dpng')
737 figure
738 plotPSD(u15_nl(:,1),u15_nl(:,2))
739 print('15_NL_PSD','-dpng')
740
741 figure
742 plotPSD(u16_l(:,1),u16_l(:,2))
743 print('16_L_PSD','-dpng')
744 figure
745 plotPSD(u16_nl(:,1),u16_nl(:,2))
746 print('16_NL_PSD','-dpng')
747
748 % Feeding into plotPhaseSpace
749 figure
750 plotPhaseSpace(u4_l(:,2),v4_l(:,2))
751 print('4_L_PhaseSpace','-dpng')
752 figure
753 plotPhaseSpace(u4_nl(:,2),v4_nl(:,2))
754 print('4_NL_PhaseSpace','-dpng')
755
756 figure
757 plotPhaseSpace(u5_l(:,2),v5_l(:,2))
758 print('5_L_PhaseSpace','-dpng')
759 figure
760 plotPhaseSpace(u5_nl(:,2),v5_nl(:,2))
761 print('5_NL_PhaseSpace','-dpng')
762
763 figure
764 plotPhaseSpace(u6_l(:,2),v6_l(:,2))
765 print('6_L_PhaseSpace','-dpng')
766 figure
767 plotPhaseSpace(u6_nl(:,2),v6_nl(:,2))
768 print('6_NL_PhaseSpace','-dpng')
769
770 figure
771 plotPhaseSpace(u14_l(:,2),v14_l(:,2))

```

```

772 print('14.L.PhaseSpace', '-dpng')
773 figure
774 plotPhaseSpace(u14_n1(:,2),v14_n1(:,2))
775 print('14.NL.PhaseSpace', '-dpng')
776
777 figure
778 plotPhaseSpace(u15_n1(:,2),v15_n1(:,2))
779 print('15.L.PhaseSpace', '-dpng')
780 figure
781 plotPhaseSpace(u15_n1(:,2),v15_n1(:,2))
782 print('15.NL.PhaseSpace', '-dpng')
783
784 figure
785 plotPhaseSpace(u16_n1(:,2),v16_n1(:,2))
786 print('16.L.PhaseSpace', '-dpng')
787 figure
788 plotPhaseSpace(u16_n1(:,2),v16_n1(:,2))
789 print('16.NL.PhaseSpace', '-dpng')
790
791 % Feeding into lyunpanov
792 for count = 1:12;
793     domFreqs = [0.3996 0.5994 0.3996 0.5994 0.3996 0.3996 0.999 ...
794                 0.7992 0.999 0.7992 0.999 0.7992];
795     varname = {'u4_L' 'u4_nL' 'u5_L' 'u5_nL' 'u6_L' 'u6_nL' 'u14_L' ...
796               'u14_nL' 'u15_L' 'u15_nL' 'u16_L' 'u16_nL'};
797     varname2 = {'4_L' '4_NL' '5_L' '5_NL' '6_L' '6_NL' '14_L' ...
798                '14_NL' '15_L' '15_NL' '16_L' '16_NL'};
799     evalc(strcat('disp = ',varname{count}, '(:,2);'));
800     save Disp_Data.txt disp -ASCII
801     fname = 'Disp_Data.txt';
802     datcnt = length(disp);
803     % taus = [100 100 100 100 100 100 100 100 100 100 100 100 100]./100;
804     tau = 1;
805     ndim = 2;
806     ires = 10;
807     maxbox = 6000;
808
809     db = basgen(fname, tau, ndim, ires, datcnt, maxbox);
810
811     dt = u4_n1(2,1)-u4_n1(1,1);
812     evolve = 1;
813     dismin = 0.001;
814     dismax = 0.1;
815     thmax = 30;
816
817     [out, SUM] = fet(db, dt, evolve, dismin, dismax, thmax);
818
819     makeplot(db, out, evolve, 'NorthWest')
820
821     [exp_bps,exp_bpo] = lyapunov_expEst(domFreqs(count))
822
823     evalc(strcat('print(''',varname2{count}, '_lyupanovEXP'', '-dpng''));'));

```

821 end

```
1 function [ ] = plotDVT(time1,displacement1,time2,displacement2)
2
3 hold on
4 plot(time1,displacement1,'r','LineWidth',1)
5 plot(time2,displacement2,'b','LineWidth',1)
6 hold off
7 set(gca,'FontSize',12)
8 grid on
9 xlabel('Time (sec)')
10 ylabel('Displacement (in)')
11 % axis([0 21000 -inf inf])
12 handxlabel1 = get(gca, 'XLabel');
13 set(handxlabel1, 'FontSize', 14, 'FontWeight', 'bold')
14 handylabel1 = get(gca, 'ylabel');
15 set(handylabel1, 'FontSize', 14, 'FontWeight', 'bold')
16 % figureHandle = gcf;
17 % ...
18     set(findall(figureHandle,'type','text'),'fontSize',14,'fontWeight','bold')
19 legend('No Freeplay','2.12 deg Freeplay')
20 end
```

```
1 function [ ] = plotPSD(time,displacement)
2
3 dt = time(2) - time(1);
4 Fs = 1/dt;
5 x = displacement;
6
7 N = length(x);
8 xdft = fft(x);
9 xdft = xdft(1:N/2+1);
10 psdx = (1/(Fs*N)) * abs(xdft).^2;
11 psdx(2:end-1) = 2*psdx(2:end-1);
12 freq = 0:Fs/length(x):Fs/2;
13
14
15
16 try
17     nattys = [0.37657 0.98170 2.1966];
18     power = interp1(freq,psdx,nattys);
19     plot(nattys(:),10*log10(power),'rs','MarkerSize',12)
20     hold on;
21     plot(freq,10*log10(psdx),'LineWidth',1)
22 catch
23     xdft = xdft(1:round(N/2));
24     psdx = (1/(Fs*N)) * abs(xdft).^2;
25     psdx(2:end-1) = 2*psdx(2:end-1);
26     freq = 0:Fs/length(x):Fs/2;
```

```

27     nattyys = [0.37657 0.98170 2.1966];
28     power = interp1(freq,psdx,nattyys);
29     plot(nattyys(:),10*log10(power),'rs','MarkerSize',12)
30     hold on;
31     plot(freq,10*log10(psd), 'LineWidth',1)
32 end
33
34 hold off
35 set(gca,'FontSize',12)
36 grid on
37
38 xlabel('Frequency (Hz)')
39 ylabel('Power/Frequency (dB/Hz)')
40 % axis([0 21000 -inf inf])
41 handxlabel1 = get(gca, 'XLabel');
42 set(handxlabel1, 'FontSize', 14, 'FontWeight', 'bold')
43 handylabel1 = get(gca, 'YLabel');
44 set(handylabel1, 'FontSize', 14, 'FontWeight', 'bold')
45 % figureHandle = gcf;
46 % ...
    set(findall(figureHandle,'type','text'),'fontSize',14,'fontWeight','bold')
47
48 end

```

```

1 function [ ] = plotPhaseSpace(displacement,velocity)
2
3 plot(displacement(1),velocity(1),'rs','MarkerSize',10)
4 hold on
5 plot(displacement(end),velocity(end),'r^','MarkerSize',10)
6 plot(displacement,velocity,'LineWidth',1)
7 hold off
8 xlabel('Displacement (in)')
9 ylabel('Velocity (in/s)')
10 set(gca,'FontSize',12)
11 handxlabel1 = get(gca, 'XLabel');
12 set(handxlabel1, 'FontSize', 14, 'FontWeight', 'bold')
13 handylabel1 = get(gca, 'YLabel');
14 set(handylabel1, 'FontSize', 14, 'FontWeight', 'bold')
15 grid on;
16 legend('t = 0','t = 10')
17
18
19 end

```

```

1 function db = basgen(fname, tau, ndim, ires, datcnt, maxbox)
2 % Database generator for fet.m function
3 % Taehyeun Park, The Cooper Union, EE'15
4
5 x = fileread(fname);
6 data = zeros(1,datcnt);

```

```

7  trck = 1;
8  start = 1;
9  fin = 0;
10
11  for ii = 1:length(x)
12      if strcmp(x(ii), char(32)) || strcmp(x(ii), char(13)) || ...
          strcmp(x(ii), char(10)) || strcmp(x(ii), char(26))
13          if fin >= start
14              data(trck) = str2num(x(start:fin));
15              trck = trck + 1;
16              if trck > 8*floor(datcnt/8)
17                  break
18              end
19          end
20          start = ii + 1;
21      else
22          fin = ii;
23      end
24  end
25
26  delay = 0:tau:(ndim-1)*tau;
27
28  nxtbox = zeros(maxbox, ndim);
29  where = zeros(maxbox, ndim);
30  datptr = zeros(1,maxbox);
31  nxtdat = zeros(1,datcnt);
32
33  datmin = min(data);
34  datmax = max(data);
35
36  datmin = datmin - 0.01*(datmax - datmin);
37  datmax = datmax + 0.01*(datmax - datmin);
38  boxlen = (datmax - datmin)/ires;
39
40  boxcnt = 1;
41
42  for ii = 1:(datcnt-(ndim-1)*tau)
43      target = floor((data(ii+delay)-datmin)/boxlen);
44      runner = 1;
45      chaser = 0;
46
47      jj = 1;
48      while jj <= ndim
49          tmp = where(runner, jj)-target(jj);
50          if tmp < 0
51              chaser = runner;
52              runner = nxtbox(runner, jj);
53              if runner ~= 0
54                  continue
55              end
56          end
57          if tmp ~= 0

```

```

58         boxcnt = boxcnt + 1;
59
60         if boxcnt == maxbox
61             error('Grid overflow, increase number of box count')
62         end
63
64         for kk = 1:ndim
65             where(boxcnt, kk) = where(chaser, kk);
66         end
67         where(boxcnt, jj) = target(jj);
68         nxtbox(chaser, jj) = boxcnt;
69         nxtbox(boxcnt, jj) = runner;
70         runner = boxcnt;
71     end
72     jj = jj + 1;
73 end
74     nxtdat(ii) = datptr(runner);
75     datptr(runner) = ii;
76 end
77
78 used = 0;
79 for ii = 1:boxcnt
80     if datptr(ii) ~= 0;
81         used = used + 1;
82     end
83 end
84 display(['Created: ', num2str(boxcnt)]);
85 display(['Used: ', num2str(used)]);
86
87 db.ndim = ndim;
88 db.ires = ires;
89 db.tau = tau;
90 db.datcnt = datcnt;
91 % db.datcnt = datcnt-6;
92 db.boxcnt = boxcnt;
93 db.datmax = datmax;
94 db.datmin = datmin;
95 db.boxlen = boxlen;
96
97 db.datptr = datptr(1:boxcnt);
98 db.nxtbox = nxtbox(1:boxcnt, 1:ndim);
99 db.where = where(1:boxcnt, 1:ndim);
100 db.nxtdat = nxtdat(1:datcnt);
101 db.data = data;
102 % db.data = data(1:end-6);

1 function [out, SUM] = fet(db, dt, evolve, dismin, dismax, thmax)
2 % Computes Lyapunov exponent of given data and parameters, generates ...
3 %   output
4 % textfile, exact replica of Fortran 77 version of fet
5 % Taehyeun Park, The Cooper Union, EE'15

```

```

5
6 out = [];
7
8 ndim = db.ndim;
9 ires = db.ires;
10 tau = db.tau;
11 datcnt = db.datcnt;
12 datmin = db.datmin;
13 boxlen = db.boxlen;
14
15 datptr = db.datptr;
16 nxtbox = db.nxtbox;
17 where = db.where;
18 nxtdat = db.nxtdat;
19 data = db.data;
20
21 delay = 0:tau:(ndim-1)*tau;
22 datuse = datcnt-(ndim-1)*tau-evolve;
23
24 its = 0;
25 SUM = 0;
26 savmax = dismax;
27
28 oldpnt = 1;
29 newpnt = 1;
30
31 fileID = fopen('fetout.txt', 'w');
32
33 goto50 = 1;
34 while goto50 == 1;
35     goto50 = 0;
36     [bstpnt, bstdis, thbest] = search(0, ndim, ires, datmin, boxlen, ...
37         nxtbox, where, ...
38         datptr, nxtdat, data, delay, oldpnt, newpnt, datuse, dismin, ...
39         dismax,...
40         thmax, evolve);
41
42     while bstpnt == 0
43         dismax = dismax * 2;
44         [bstpnt, bstdis, thbest] = search(0, ndim, ires, datmin, ...
45             boxlen, nxtbox, where, ...
46             datptr, nxtdat, data, delay, oldpnt, newpnt, datuse, ...
47             dismin, dismax,...
48             thmax, evolve);
49     end
50
51     dismax = savmax;
52     newpnt = bstpnt;
53     disold = bstdis;
54     iang = -1;
55
56     goto60 = 1;

```

```

53     while goto60 == 1;
54         goto60 = 0;
55
56         oldpnt = oldpnt + evolve;
57         newpnt = newpnt + evolve;
58
59         if oldpnt >= datuse
60             return
61         end
62
63         if newpnt >= datuse
64             oldpnt = oldpnt - evolve;
65             goto50 = 1;
66             break
67         end
68
69         p1 = data(oldpnt + delay);
70         p2 = data(newpnt + delay);
71         disnew = sqrt(sum((p2 - p1).^2));
72
73         its = its + 1;
74
75         SUM = SUM + log(disnew/disold);
76         zlyap = SUM/(its*evolve*dt*log(2));
77         out = [out; its*evolve, disold, disnew, zlyap, ...
78             (oldpnt-evolve), (newpnt-evolve)];
79
80         if iang == -1
81             fprintf(fileID, '%-d\t\t\t%-8.6f\t\t\t%-8.6f\t\t\t%-8.6f\n', ...
82                 out(end,1:4)');
83         else
84             fprintf(fileID, ...
85                 '%-d\t\t\t%-8.6f\t\t\t%-8.6f\t\t\t%-8.6f\t\t\t%-d\n', ...
86                 [out(end,1:4), iang]');
87         end
88
89         if disnew <= dismax
90             disold = disnew;
91             iang = -1;
92             goto60 = 1;
93             continue
94         end
95
96         [bstpnt, bstdis, thbest] = search(1, ndim, ires, datmin, ...
97             boxlen, nxtbox, where, ...
98             datptr, nxtdat, data, delay, oldpnt, newpnt, datuse, ...
99             dismin, dismax,...
100             thmax, evolve);
101
102         if bstpnt ~= 0
103             newpnt = bstpnt;
104             disold = bstdis;

```

```

99         iang = floor(thbest);
100         goto60 = 1;
101         continue
102     else
103         goto50 = 1;
104         break;
105     end
106 end
107 end
108 fclose(fileID);

1 function [] = makeplot(db, out, evolve, loc)
2 % Plots 2D or 3D attractor evolution by evolution, 4th parameter is the
3 % location of legend
4 % Taehyeun Park, The Cooper Union, EE'15
5
6 datcnt = db.datcnt;
7 ndim = db.ndim;
8 tau = db.tau;
9 dataplot = [];
10 freerun = 0;
11
12 delay = 0:tau:(ndim-1)*tau;
13 data = db.data;
14
15 for ii = 1:(datcnt-(ndim-1)*tau)
16     dataplot = [dataplot; data(ii+delay)];
17 end
18
19 figure, bar(out(:,1),out(:,3)), hold on;
20 mle = max(dataplot(:)) - min(dataplot(:));
21 plot([0, out(end,1)], [mle, mle], 'r', 'LineWidth', 1.5), hold off;
22 set(gca,'YTick', [0, mle])
23 axis([0, out(end,1), 0, 1.1*mle])
24 title('d.f of evolutions scaled to the maximum linear extent of the ...
    attractor')
25
26 if ndim == 2
27     figure('Position', [100, 100, 800, 500]);
28     plot(dataplot(:,1), dataplot(:,2), '.', 'MarkerSize', 3), hold on;
29     display('To see the next evolution, press enter')
30     display('To clear the screen and then see the next evolution, ...
        type c and press enter')
31     display('To proceed without stopping, type r and press enter')
32     display('To terminate plot generating, type g and press enter')
33
34     for ii = 1:size(out,1)
35         if freerun == 0
36             % RESET = input('Next evolution? ', 's');
37             RESET = 'g';
38             if strcmp(RESET, 'c')

```

```

39         display('Screen cleared')
40         hold off;
41         clf;
42         plot(dataplot(:,1), dataplot(:,2), '.', ...
              'MarkerSize', 3), hold on;
43     elseif strcmp(RESET, 'r')
44         display('Evolving without stopping...')
45         display('Press ctrl+c to terminate')
46         freerun = 1;
47     elseif strcmp(RESET, 'g')
48         display('Plot generating stopped')
49         return;
50     else
51         if ii > 1
52             delete(ann)
53         end
54     end
55 end
56
57 tmpold = out(ii,5);
58 oldpnt = tmpold + evolve;
59 tmpnew = out(ii,6);
60 newpnt = tmpnew + evolve;
61
62 plot(data(tmpold:oldpnt), data((tmpold+tau):(oldpnt+tau)), ...
      'r', 'LineWidth', 1);
63 plot(data(tmpnew:newpnt), data((tmpnew+tau):(newpnt+tau)), ...
      'g', 'LineWidth', 1);
64 for aa = 0:evolve;
65     plot([data(tmpold+aa), data(tmpnew+aa)], ...
          [data(tmpold+aa+tau), data(tmpnew+aa+tau)], ...
          'LineWidth', 1)
66 end
67
68 ann = legend(['Iteration: ', num2str(out(ii,1)), '/', ...
              num2str(out(end,1)), char(10)...
              'd_i:', num2str(out(ii,2)), char(10)...
              'd_f:', num2str(out(ii,3)), char(10)...
              'Current Estimate:' num2str(out(ii,4))], ...
              'location', loc);
74     if freerun == 1
75         drawnow
76     end
77 end
78
79 elseif ndim == 3
80     figure('Position', [100, 100, 800, 500]);
81     plot3(dataplot(:,1), dataplot(:,2), dataplot(:,3), '.', ...
           'MarkerSize', 3), hold on;
82     display('To see the next evolution, press enter')
83     display('To clear the screen and then see the next evolution, ...

```

```

    type c and press enter')
84 display('To proceed without stopping, type r and press enter')
85 display('To terminate plot generating, type g and press enter')
86
87 for ii = 1:size(out,1)
88     if freerun == 0
89         % RESET = input('Next evolution? ', 's');
90         RESET = 'g';
91         if strcmp(RESET, 'c')
92             display('Screen cleared')
93             hold off;
94             clf;
95             plot3(dataplot(:,1), dataplot(:,2), dataplot(:,3), ...
96                 '.', 'MarkerSize', 3), hold on;
97         elseif strcmp(RESET, 'r')
98             display('Evolving without stopping...')
99             display('Press ctrl+c to terminate')
100             freerun = 1;
101         elseif strcmp(RESET, 'g')
102             display('Plot generating stopped')
103             return;
104         else
105             if ii > 1
106                 delete(ann)
107             end
108         end
109
110         tmpold = out(ii,5);
111         oldpnt = tmpold + evolve;
112         tmpnew = out(ii,6);
113         newpnt = tmpnew + evolve;
114
115         plot3(data(tmpold:oldpnt), data((tmpold+tau):(oldpnt+tau)), ...
116             data((tmpold+(2*tau)):(oldpnt+(2*tau))), 'r', ...
117             'LineWidth', 1);
118         plot3(data(tmpnew:newpnt), data((tmpnew+tau):(newpnt+tau)), ...
119             data((tmpnew+(2*tau)):(newpnt+(2*tau))), 'g', ...
120             'LineWidth', 1);
121         for aa = 0:evolve;
122             plot3([data(tmpold+aa), data(tmpnew+aa)], ...
123                 [data(tmpold+aa+tau), data(tmpnew+aa+tau)], ...
124                 [data(tmpold+aa+(2*tau)), data(tmpnew+aa+(2*tau))], ...
125                 'LineWidth', 1)
126         end
127
128         ann = legend(['Iteration: ', num2str(out(ii,1)), '/', ...
129             num2str(out(end,1)), char(10)...
130             'd_i:', num2str(out(ii,2)), char(10)...
131             'd_f:', num2str(out(ii,3)), char(10)...
132             'Current Estimate:' num2str(out(ii,4))], ...

```

```

126             'location', loc);
127         if freerun == 1
128             drawnow
129         end
130     end
131 end

1 function [LyaExp_b_sec,LyaExp_b_orb] = lyapunov_expEst(domFreq)
2
3 close all;
4 % Mean Orbital Period from PSD
5 meanPeriod = 1/domFreq;
6
7 % Output data fetout.txt from lyapunov.m code
8 bitsPerSec_Data = importdata('fetout.txt');
9 % Estimate of lyapunov exponent for each increment
10 bitsPerSec_Exp = bitsPerSec_Data(:,8);
11
12 i = 1;
13 ind = 1;
14 while i <= length(bitsPerSec_Exp)
15     if isnan(bitsPerSec_Exp(i)) ~= 1
16         BPS(ind) = bitsPerSec_Exp(i);
17         ind = ind + 1;
18     end
19     i = i + 1;
20 end
21
22 BPS = BPS';
23 bitsPerOrbit = BPS.*meanPeriod;
24 plot(bitsPerOrbit,'linewidth',2)
25 hold on; plot([1 length(BPS)],[0 0],'-k')
26 grid on;
27 axis([0 length(BPS) -inf inf])
28
29 xlabel('Time \rightarrow')
30 ylabel('Lyapunov Exponent (bits/orbit)')
31 figureHandle = gcf;
32 set(findall(figureHandle,'type','text'),'fontSize',10,'fontWeight','bold')
33
34 LyaExp_b_sec = BPS(end);
35 LyaExp_b_orb = bitsPerOrbit(end);
36
37 end

```

Bibliography

- [1] M. Conner *et al.*, “Nonlinear behavior of a typical airfoil section with control surface freeplay: a numerical and experimental study,” *Journal of Fluids and Structures*, vol. 11, pp. 89–109, 1997.
- [2] (2015) Chaos. [Online]. Available: Merriam-Webster.com
- [3] L. W. Just, “Dynamic response analysis of an icosahedron shaped lighter than air vehicle,” Master’s thesis, Air Force Institute of Technology, Wright Patterson AFB, Ohio, 2015.
- [4] G. L. Baker and J. P. Gollub, *Chaotic Dynamics: An Introduction*, 1st ed. New York, NY: Cambridge University Press, 1990.
- [5] O. Sigmund, “A 99 line topology optimization code written in matlab,” *Struct Multidisc Optim*, vol. 21, pp. 120–127, 2001.
- [6] R. M. Cummings *et al.*, *APPLIED COMPUTATIONAL AERODYNAMICS: A MODERN ENGINEERING APPROACH*. New York, NY: Cambridge University Press, 2015.
- [7] L. Meirovitch, *Fundamentals of Vibrations*. New York, NY: McGraw-Hill, 2001.
- [8] D. H. Hodges and G. A. Pierce, *Introduction to Structural Dynamics and Aeroelasticity*, 2nd ed. New York, NY: Cambridge University Press, 2013.
- [9] R. L. Bisplinghoff *et al.*, *Aeroelasticity*, 1st ed. New York, NY: Addison-Wesley Publishing Company, 1955.
- [10] J. G. Reid, *Linear System Fundamentals: Continuous and Discrete, Classic and Modern*. New York, NY: McGraw-Hill Inc., 1983.

- [11] R. D. Cook *et al.*, *Concepts and Applications of Finite Element Analysis*, 4th ed. New York, NY: John Wiley & Sons Inc., 2002.
- [12] *Abaqus Analysis User's Manual*, Dassault Systèmes, 2014.
- [13] D. P. Raymer, *Aircraft Design: A Conceptual Approach*, 5th ed. Reston, VA: American Institute of Aeronautics and Astronautics, 2012.
- [14] E. Dowell *et al.*, “Transonic limit cycle oscillation analysis using reduced order aerodynamic models,” *Journal of Fluids and Structures*, vol. 19, pp. 17–27, 2004.
- [15] D. Tang *et al.*, “Limit cycle behavior of an airfoil with a control surface,” *Journal of Fluids and Structures*, vol. 12, pp. 839–858, 1998.
- [16] J. P. Crutchfield *et al.*, “Chaos,” *Scientific American*, vol. 254, no. 12, pp. 46–57, 1986.
- [17] A. Wolf *et al.*, “Determining lyapunov exponents from a time series,” *Physica D: Nonlinear Phenomena*, vol. 16, no. 3, pp. 285–317, 1985.
- [18] (2017). [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/thumb/b/b7/Wing_structure_-_ribs.svg/1012px-Wing_structure_-_ribs.svg.png
- [19] M. Drela, “Xfoil: An analysis and design system for low reynolds number airfoils,” *MIT Dept. of Aeronautics and Astronautics*.
- [20] *ANSYS Fluent Theory Guide*, ANSYS, 2013.
- [21] A. S. Saada, *Elasticity: Theory and Applications*, 2nd ed. Ft Lauderdale, FL: J. Ross Publishing Inc., 2009.

REPORT DOCUMENTATION PAGE					<i>Form Approved</i> OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY)		2. REPORT TYPE		3. DATES COVERED (From — To)		
23 March 2017		Master's Thesis		Sept 2016 — March 2017		
4. TITLE AND SUBTITLE				5a. CONTRACT NUMBER		
STUDY OF CHAOTIC BEHAVIOR IN THE NONLINEAR DYNAMIC RESPONSE OF AN AIRFOIL WITH A TRAILING EDGE FLAP				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S)				5d. PROJECT NUMBER		
Lee, Joshua J., 1st Lieutenant				JON 17Y292		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)					8. PERFORMING ORGANIZATION REPORT NUMBER	
Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765					AFIT-ENY-MS-17-M-272	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)					10. SPONSOR/MONITOR'S ACRONYM(S)	
AFRL/RW Att: Dr. Crystal L. Pasiliao 101 West Eglin Blvd, Suite 134 Eglin AFB, FL 32542 Email: crystal.pasiliao@us.af.mil					AFRL/RW	
					11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT						
DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT						
Even though flutter is a linear phenomenon, no connections between LCO, a nonlinear version of flutter, and chaos have been made. To make this connection, an internal structure for an airfoil was optimized to maximize stiffness, any changes in the aerodynamics of the optimized structure were calculated, and finally, a Finite Element model was developed to excite the aeroelastic structure. The results show that the simulation must run for an extended period of time, that the Lyapunov exponent parameters used were acceptable for the linear system, and that the chaos of the nonlinear system could not be commented on until further analysis is performed.						
15. SUBJECT TERMS						
Structures, Optimization, Computational Fluid Dynamics, Finite Element Analysis, Nonlinear Dynamics						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT		18. NUMBER OF PAGES	
a. REPORT	b. ABSTRACT	c. THIS PAGE			19a. NAME OF RESPONSIBLE PERSON	
U	U	U	UU		Dr. A. Palazotto, AFIT/ENY	
					19b. TELEPHONE NUMBER (include area code)	
					(937) 255-3636, x4599; Anthony.Palazotto@afit.edu	