2018

Optimization of Tow-Steered Composite Wind Turbine Blades for Static Aeroelastic Performance

Stephen Michael Barr
Lehigh University

Optimization of Tow-Steered Composite Wind Turbine Blades for Static Aeroelastic Performance

by

Stephen M. Barr

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Mechanical Engineering and Mechanics

Lehigh University

May 2018

This thesis is accepted and approved in partial fulfilment of the requirements for the degree of Master of Science in Mechanical Engineering.

Date

Dr. Justin W. Jaworski, Thesis Advisor

Prof. D. Gary Harlow, Chairperson of
Department of Mechanical Engineering and Mechanics

Acknowledgements

I would like to thank my advisor, Dr. Justin W. Jaworski, for the guidance and patience he provided to me throughout this research. I am grateful to not only have had his mentorship in this work, but also in future endeavors.

I would also like to thank my employer, Combustion Research and Flow Technology, Inc., and my colleagues who have always been so generous to share their extensive knowledge and experience with me throughout my career and education at Lehigh.

Thank you to my Mom, Dad, and brother for supporting me in the pursuit of my degrees and in everything I do in my life.

Finally, thank you to my wife, Kathleen. She inspires me to always be the best I can be, and my accomplishments mean so much more now that they are shared with her.

Acronyms

AEY	Annual Energy Yield
BEM	Blade Element Momentum
CAD	Computer-Aided Design
CFD	Computational Fluid Dynamics
FEA	Finite Element Analysis
HAWT	Horizontal Axis Wind Turbine
MPI	Message Passing Interface
NURBS	Non-Uniform Rational B-Spline
SLSQP	Sequential Least Squares Quadratic Programming
VAT	Variable-Angle Tow

Contents

List of Tables

List of Figures

Abstract

The concept of passive aeroelastic tailoring is explored to maximize the performance of the NREL 5-MW wind turbine blade in a uniform flow. Variable-angle tow composite materials model the spanwise-variable wind turbine blade design to allow material-adaptive bend-twist coupling under static aerodynamic loading. A constrained optimization algorithm determines the composite fiber angles along the blade span for four inflow conditions ranging from cut-in to rated wind speeds. The computational fluid dynamics solver CRUNCH CFD® and commercial finite element analysis solver Abaqus compute the static aerodynamic loads and structural deformations of the blades, respectively, which are passed iteratively between the solvers until static aeroelastic convergence is achieved. A parallel grid deformation code based on the stiffness method is developed to deform the fluid mesh based on the structural deformation of the blade. The elemental stiffness is set to the inverse of the element volume to preserve the grid quality during grid deformation.

Turbine power extraction is predicted to increase by up to 14% when the blade is optimized near the cut-in wind speed, and by 7% when optimized at rated wind speed. Using the results from optimizations at discrete wind speeds, two blade design strategies are evaluated to determine a single composite layup for the blade that maximizes performance over the range of wind speeds. The first strategy uses the composite layup optimized at the rated wind speed increasing power extraction by 10% near cut-in and 7% at rated conditions, relative to the baseline blade design. The second strategy seeks a new composite layup that most closely matches the optimal blade twist at each wind speed, which results in an increase in power extraction of 14% near cut-in and 3% at rated conditions.

Chapter 1

Introduction

The goal of a wind turbine is to convert the kinetic energy of oncoming wind into electricity. This goal is accomplished by orienting wind turbines such that the wind produces an aerodynamic force on the blade, which generates torque on a rotating shaft. When the shaft is connected to a generator, the mechanical power can be converted into electricity [1]. Aerodynamically, the amount of power extracted by the wind turbine is based on two factors: the amount of power available in the wind passing through the turbine capture area, and the efficiency of the turbine blades at converting the forward motion of the wind into rotational motion. The amount of power available in the wind for a Horizontal Axis Wind Turbine (HAWT) is determined by the following equation,

$$P_{\text{avail}} = \frac{1}{2}\rho V^3 A = \frac{1}{2}\rho V^3(\pi s^2), \tag{1.1}$$

where ρ is the fluid density, V is the wind velocity, A is the capture area of the turbine, and s is the blade span. From this equation, it is clear that wind turbine designers should make the blade span as large as possible in order to maximize the amount of power available for extraction by the wind turbine. While this solution is intuitive when only considering aerodynamics, challenges arise when the problem is thought of as a multi-disciplinary design involving aerodynamics, structural dynamics, and materials science. Large blade spans generate enormous root bending moments, which can cause substantial blade flexure.

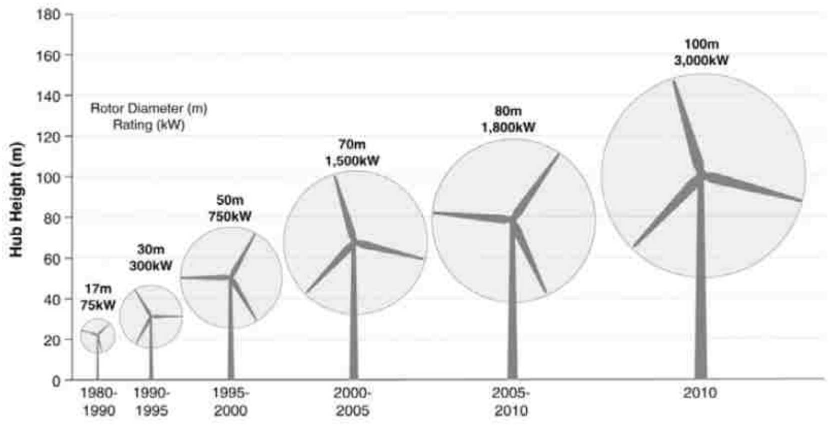


Figure 1.1: Historical trend in wind turbine blade span [3]

Furthermore, as blades deform under aerodynamic loading, their efficiency at extracting power can be altered as their shape changes. Figure 1.1 shows that maximum wind turbine blade spans have consistently increased from the 1980s to present day. Jensen [2] notes that while scaling laws predict that the weight of wind turbine blades should grow as a function of their length cubed, historically, blade weight has grown as a function of length to the power with an exponent of only 2.3. This slower-than-expected increase in blade weight is attributed to advances in wind turbine structural design and materials, including the use of composites. Composite materials not only have a high strength-to-weight ratio, but they also have anisotropic material properties that results in a different stress response in each direction. This anisotropy can be exploited to aeroelastically tailor wind turbine blades as they deform under loads. Novel solutions such as these will need to be used in the future to continue increasing power output from wind turbines, a key factor in reducing the lifetime cost of wind energy. This thesis focuses on the concept of aeroelastically optimizing Variable-Angle Tow (VAT) composite fibers in the blade skin and spars to improve the power capture efficiency of the NREL 5-MW wind turbine blade at multiple points between the cut-in and rated wind speeds.

1.1 Literature Review

Shirk [4] defines aeroelastic tailoring as "the embodiment of directional stiffness into an aircraft structural design to control aeroelastic deformation, static or dynamic, in such a fashion as to affect the aerodynamic and structural performance of that aircraft in a beneficial way." In regard to wind turbine design, aeroelastic tailoring is used to either reduce structural loads on the blades or to increase the aerodynamic performance of the blades by modifying their shape. Wind turbine blade aeroelastic tailoring can be classified as either active or passive. Active aeroelastic tailoring involves changing the blade shape using methods such as adaptive compliant trailing edges [5], flexible airfoil walls that adapt to prevent boundary layer separation [6], and shape change airfoils that modify their aerodynamic shape using piezoelectric materials [7] to mention a few. Active aeroelastic tailoring methods for wind turbine blades have been shown to delay stall, which can increase aerodynamic performance [7], but these methods come with the disadvantages of increased system complexity and power required to operate the control systems. Unlike active aeroelastic tailoring, passive aeroelastic tailoring modifies the blade shape in a beneficial way when the blade deforms under aerodynamic loading without the requirement of control systems or input power. Passive aeroelastic tailoring can be achieved by intelligent shaping of the blade and/or using the anisotropic properties of composite materials. In the subsequent discussion, passive aeroelastic design approaches are generally classified into the two main categories of geometric-adaptive and material-adaptive tailoring concepts.

1.1.1 Geometric-Adaptive Tailoring

Swept Twist Adaptive Rotor Blades

Bend-twist coupling can be achieved by sweeping the blade near the tip [8]. When a swept blade is loaded, a moment is created about the blade spanwise axis that causes the blade to twist at the outboard section; this effect is illustrated in Figure 1.2. The twist unloads the blade at the tip and reduces the root bending moment. By reducing the root bending moment, larger turbine radii can be realized to increase energy capture. Zuteck [8] studied

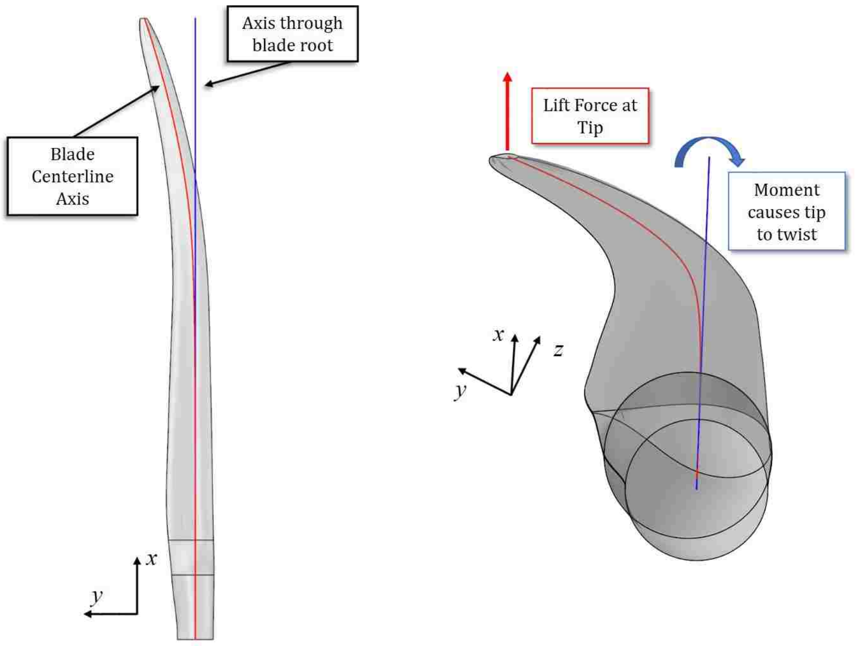
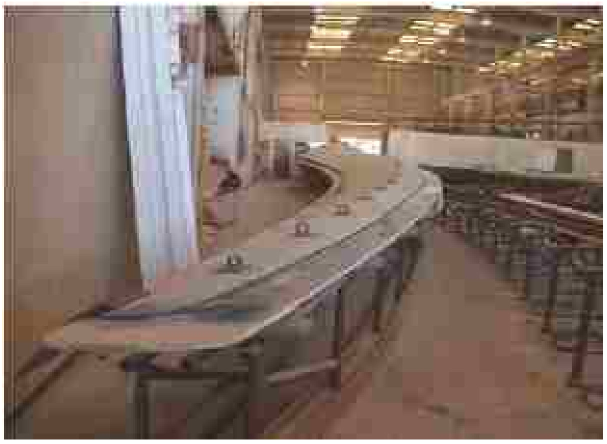
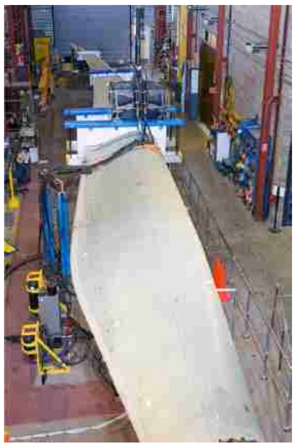


Figure 1.2: Bend-twist coupling can be achieved by sweeping the blade near the tip. Due to the sweep, a lift force at the tip creates a moment about the blade spanwise axis and twists the blade.

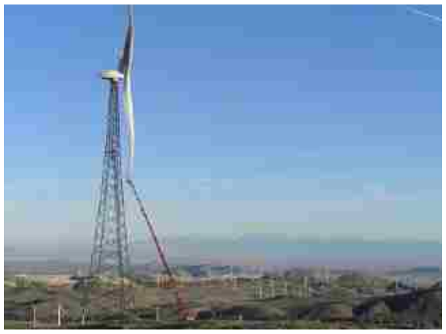
the effect of a swept blade planform on tip twist on a blade with a 30 m span and a sweep curve exponent of 4 in a 10 m/s freestream. He performed structural analysis of the swept blade assuming an inverse triangle thrust distribution in which the thrust varies linearly from zero at the root and maximum at the tip. Zuteck concluded that 4° of tip twist is possible for an all-fiberglass construction blade, and up to 7° of tip twist is possible when the blade is fitted with carbon fiber spar caps. The development of an experimental swept twist adaptive rotor is detailed by Ashwill *et al.* [9, 10], who used the Zond 750 with a 24.5 m blade span as the baseline turbine for their research. A parametric design study of the sweep and cross-sectional variation along the blade was conducted to assess their effects on energy capture. Their study details the fabrication, static load testing, and power output testing of a swept twist adaptive rotor blade, which are shown in Figure 1.3. Ashwill *et al.* [9, 10] concluded that the swept-rotor design passively reduces root bending moments allowing for larger rotor diameters offering 10% to 12% increases in average energy capture compared to the baseline blade.



(a)

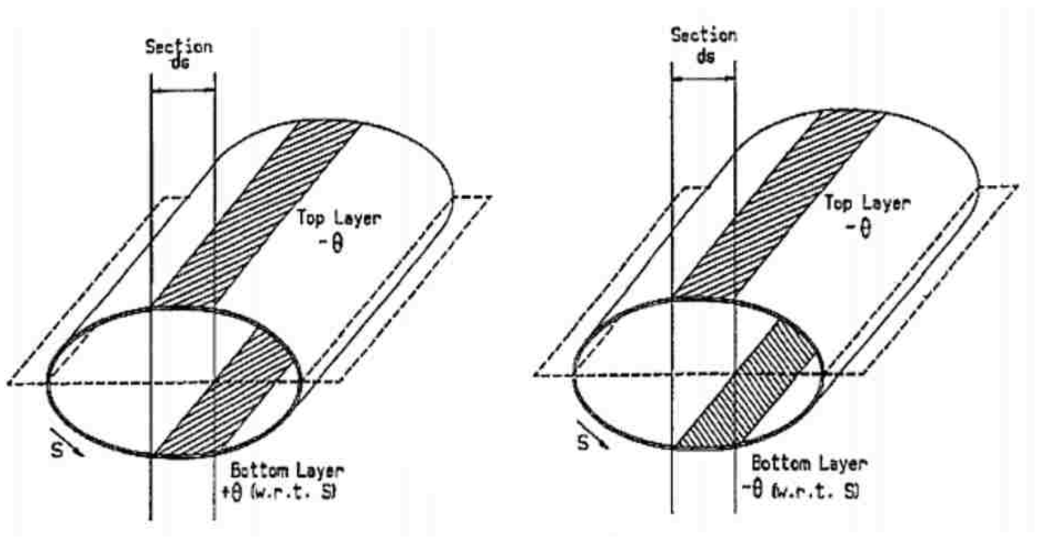


(b)



(c)

Figure 1.3: Sandia swept twist adaptive rotor blade: (a) in fabrication; (b) under static load testing; (c) on the turbine test stand [9].



(a) Bend-twist coupling (b) Extension-twist coupling

Figure 1.4: Illustration of extension-twist and bend-twist coupling. The difference in fiber orientations between the top and bottom surface of the blade determine whether the blade is bend-twist or extension-twist coupled. Bend-twist coupling is created when the fiber directions on the top surface of the blade are the inverse of those on the bottom surface [11].

1.1.2 Material-Adaptive Tailoring

Off-Axis Composite Fibers

Karaolis [11, 12] first introduced the concept of biasing composite fibers away from the bending axis to induce twist coupling. The concept is based on the idea that composite materials have different material properties in each direction. Depending on the direction of the fibers on the top and bottom surfaces of the blade, either bend-twist or extension-twist coupling can be realized; Figure 1.4 illustrates these concepts. With bend-twist coupling, the composite fiber angles on the bottom surface are mirrored from the top surface, which causes the blade to twist in reaction to a bending load. Extension-twist coupled blades use a helical layup to twist the blade in reaction to a tension load (such as centrifugal loads on the turbine blade) as they rotate.

Lobitz and Veers [13] studied the performance benefits of implementing twist-coupling into wind turbine blades. They concluded that bend-twist coupling can be used to mitigate loads by twisting towards feather (unloading the blade) without reducing the average power and results in substantial reductions in blade fatigue by reducing cyclic blade loading. Conversely, twisting towards stall with constant fiber angles along the blade span increases bending fatigue damage loads and induces flutter in either steady or turbulent inflows. Lobitz and Veers [13] estimated that using bend-twist coupling could result in energy production increases of between 5% and 10%.

Maheri [14] utilized design optimization to design an aeroelastically-tailored wind turbine blade. The objective of the optimization was to maximize the average wind power extracted by the turbine. Design variables in the optimization included the blade span, pre-twist distribution, fiber orientation, and the thickness of the blade skin. The fiber orientation was set to be constant along the blade span. The blade aerodynamics were computed using Blade Element Momentum (BEM) theory. From his analysis, Maheri determined increases in energy capture based on site averaged wind speed of about 16% at 5.6 m/s, about 12% at 7 m/s, and about 8% at 9 m/s.

In the work by Battossa [15], the effect of off-axis fibers at various spanwise sections on performance was studied. Among the blades analyzed included a fully-coupled blade with off-axis fibers from root to tip, and five "partially-coupled" configurations with off-axis fibers placed from different spanwise locations to the blade tip. A constrained optimization was conducted on the blade structural members to minimize cost and weight. Again, the aerodynamic loads were computed using BEM. The partially coupled blades were found to reduce damage fatigue while maintaining stiffness in the inboard sections of the blade. One of the conclusions made by the author was that the use of varying fiber angles along the blade span may lead to even higher wind turbine performance.

VAT Composite Fibers

Much of the available literature available on bend-twist coupling of wind turbine blades involves the use of constant fiber-orientations along the blade span. In the work by Capuzzi [16–18], twist-coupling concepts were studied involving the use of VAT composite fibers, whereby the fiber orientations are prescribed and vary over the blade span. Scott and Capuzzi [19] also compared blades constructed with only material-adaptive tailoring with blades including a combination of material-adaptive tailoring and geometric-adaptive tailoring. The blade aerodynamics were computed using BEM theory. Capuzzi [17] used a genetic algorithm to find the blade twist distribution which optimizes the Annual Energy Yield (AEY) of the turbine. Once the optimal twist distribution was found, a spar was designed with VAT composites which causes the blade to conform to the optimal twist distribution when deformed under the aerodynamic loads. Care was also taken to ensure the spar fit into the blade properly. From his work, Capuzzi concluded that bend-twist coupling with VAT composites can be used to reduce root bending moment by about 9% while also increasing AEY by up to 1%.

1.2 Major Unresolved Issues

The survey of related investigations in §1.1 highlights the contemporary need to pursue aeroelastic systems based on a high-fidelity Computational Fluid Dynamics (CFD) solver coupled to an Finite Element Analysis (FEA) solver to optimize the spanwise-variable design of wind turbine blades at and across various operating points. Current research involving the design optimization of wind turbine blades relies principally on lower-order methods such as BEM with ad hoc corrections and a stall model to evaluate the aerodynamics. The use of a CFD solver offers the ability to make more accurate predictions of torque and aerodynamic loading, especially in cases near stall. Additionally, Capuzzi's [17, 18] approach to the optimization of blades with VAT composites was done in two steps: optimize the blade twist distribution to maximize power, and then designed a spar with VAT composites to arrive at that optimal twist distribution. As a consequence, only the elastic behavior of the spar is considered in deforming the blade, and the stiffness of the blade skin is neglected. The approach selected in this thesis aims to improve upon the work by Capuzzi by analyzing the entire blade structure with FEA while optimizing the VAT composite fibers in the spars and skin of the blade.

1.3 Research Questions

The following research questions are considered in this work:

1. Can the composite fibers in the blade skin and spar sections be optimized along the blade span to increase power output without stalling the blade?
2. How does the optimal composite fiber orientation change as a function of wind speed?
3. Can one composite fiber orientation be used to increase the power output over the entire range of operating wind speeds?
4. Is the optimal composite fiber orientation to improve wind turbine power performance unique?
5. Is the optimal blade solution robust with respect to the number and type of design optimization parameters used?

1.4 Technical Approach

While ample research is available showing that off-axis composite fibers can be used in wind turbine blades to twist the blade towards feather and alleviate loads [13], little research has explored the effects of twisting the blade towards stall to increase energy capture.

In this study, the spanwise distribution of VAT composite fibers is optimized to see if power extraction can be optimized between cut-in and rated wind speeds. Optimizations are conducted at individual wind speeds to explore how optimal spanwise fiber orientation distributions vary as a function of wind speed in §3.3.1. The results of those optimizations are then used to design a single composite layup which optimizes the blade over the range of wind speeds in §3.3.2. The optimization solver utilized is the Sequential Least Squares Quadratic Programming (SLSQP) solver in SciPy.

The NREL 5-MW wind turbine blade was selected as the baseline blade for the analysis. Because the internal structure and construction of the blade is not available, a representative blade structure is adopted based on the available published research. For this reason, FEA is performed in Abaqus to deform the blade based on aerodynamic loads; however, computed stresses and strains are not used as performance criteria in the optimization, which represents an area to improve upon the present analysis in the future.

A key difference in this work compared with available research is that the blade aerodynamics used to evaluate each design iteration in the optimization are solved using a high-fidelity, incompressible CFD solver, as opposed to a lower-order model such as BEM theory. Many researchers have avoided the use of CFD in the design optimization of wind turbine blades due to the significant increase in computational cost, yet the increased accuracy possible when utilizing CFD is well-known [20]. The use of a CFD solver in this design optimization was made feasible in part by only considering steady aerodynamics and static deformation of the blade. A grid deformation utility was also developed with Message Passing Interface (MPI) to deform the CFD grid based on the structural analysis. This approach is justified since only conservative estimations of static bend-twist coupling properties of the blade are sought here. Furthermore, the static aeroelastic results are validated in §3.1 against available literature to ensure the simplifying assumptions do not degrade the accuracy of the model. While the evaluation of the unsteady aerodynamics and blade structural dynamics is beyond the scope of this thesis, it represents an opportunity to improve the current model for future work.

Chapter 2

Methods and Models

This chapter details the modeling tools, procedures and assumptions used to carry out the static aeroelastic simulations of rotating composite wind turbine blades. Section 2.1 presents the definition of the NREL 5-MW blade, and the code developed to generate the Computer-Aided Design (CAD) model of the blade based on the cross-sectional definition. Section 2.2 and §2.3 detail the aerodynamic and structural computational models, respectively. A parallel grid deformation tool to deform the CFD mesh was developed with MPI and is detailed in §2.4. The coupling procedure between the aerodynamic and structural models is presented in §2.5. Finally, the constrained optimization solver is discussed in §2.6.

2.1 Blade Geometry

The NREL 5-MW wind turbine blade was selected to conduct the aeroelastic bend-twist coupling optimization analysis. The design of the NREL 5-MW blade was made publicly available in the report by Jonkman *et al.* [21]. The blade design is non-proprietary and has been used in a considerable amount of wind turbine research including the works of Bazilevs [22–24], McWilliam [20], and Capuzzi [16–18]. The NREL 5-MW blade geometry was also selected because of its relatively large span of approximately 63 meters. For a blade with elastic bend-twist coupling, the amount of twist will vary linearly with the tip bending displacement. For this reason, a longer blade will benefit more from bend-twist coupling effects since they have much larger amounts of tip bending displacements than smaller blades.

Table 2.1 lists the airfoil section, twist, and chord along the span of the blade using data from Jonkman *et al.* [21]. A C++ program built using OpenCascade was developed to generate a surface model of the blade by lofting the airfoil cross-section definitions in Table 2.1. OpenCascade is an open-source set of C++ CAD libraries [25]. For each span location, the program reads the appropriate airfoil data file, rotates the airfoil about the aerodynamic centroid by the appropriate twist angle, and scales the airfoil based on the chord length. Two Non-Uniform Rational B-Spline (NURBS) curves are then interpolated through the points: one curve for points defining the top surface, and one curve for the bottom surface. More detail about NURBS curves can be found in Chapter 4 of Piegl [26]. Once all the NURBS curves defining the airfoil sections have been created, a loft function is performed to generate the top and bottom surfaces of the blade. Finally, the program also allows the user to specify any number of shear webs and their locations as a percentage of chord. The final geometry is then exported in the IGES file format, which can be read by many commercial grid generation software packages. The C++ program is very useful because not only does it allow a user to quickly generate the NREL 5-MW wind turbine blade based only on the airfoil section properties in the NREL report, it also allows the user to quickly and easily modify the airfoil shape, twist, or chord length simply by modifying a table in a text file. The source code for the C++ program is available on GitHub [27]. The wetted blade surfaces for the NREL 5-MW blade are shown in Figure 2.1 with the airfoil cross-sections shown in red.

2.2 Fluid Model

2.2.1 Incompressible Flow Solver

The fluid dynamics of the NREL 5-MW turbine blade were simulated using CRUNCH CFD® [28], a parallel unstructured CFD code. Both incompressible and compressible fluid

Table 2.1: Airfoil properties for NREL 5-MW wind turbine blade [21]

Section	Span Location (m)	Twist (°)	Chord (m)	Airfoil Section
1	2.0	13.308	3.542	Cylinder
2	2.8667	13.308	3.542	Cylinder
3	5.6	13.308	3.854	Cylinder
4	8.3333	13.308	4.167	Cylinder
5	11.75	13.308	4.557	DU40 A17
6	15.85	11.48	4.652	DU35 A17
7	19.95	10.162	4.458	DU35 A17
8	24.05	9.011	4.249	DU30 A17
9	28.15	7.795	4.007	DU25 A17
10	32.25	6.544	3.748	DU25 A17
11	36.35	5.361	3.502	DU21 A17
12	40.45	4.188	3.256	DU21 A17
13	44.55	3.125	3.010	NACA64 A17
14	48.65	2.319	2.764	NACA64 A17
15	52.75	1.526	2.518	NACA64 A17
16	56.1667	0.863	2.313	NACA64 A17
17	58.9	0.370	2.086	NACA64 A17
18	61.6333	0.106	1.419	NACA64 A17
19	62.9	0.0	0.7	NACA64 A17

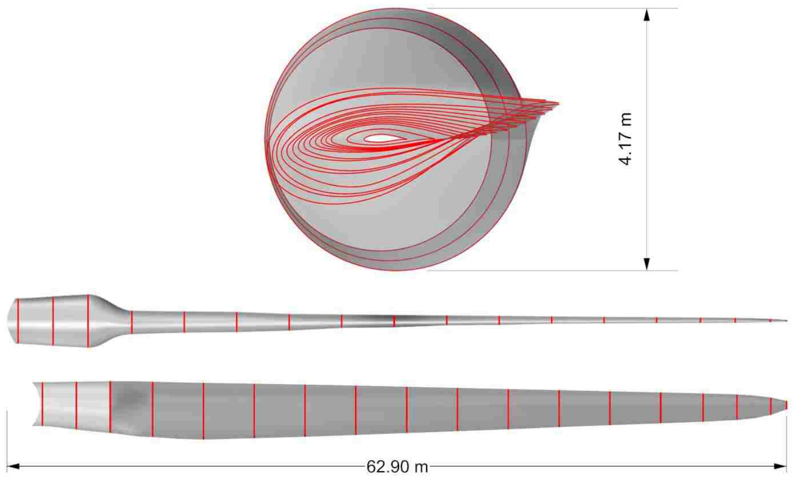


Figure 2.1: NREL 5-MW blade geometry surface model generated using Table 2.1. Airfoil cross-sections shown in red.

dynamics solvers are available within CRUNCH CFD®. For all fluid simulations in this work, the incompressible solver in CRUNCH CFD® was used. Snel [29] states that for wind turbine applications, an incompressible assumption is valid if the tip speed Mach number is below about 0.25. At rated wind speed, the highest wind speed considered in this work, the tip Mach number is about 0.235, thus an incompressible assumption is valid. An incompressible flow assumption is also used in the wind turbine CFD analyses of the NREL 5-MW blade by Bazilevs [22]. The x direction is the inflow direction and corresponds to the axial direction of the turbine, and y and z are in the radial direction of the turbine. The governing equation in the CRUNCH CFD® incompressible pressure-based flow solver is

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{E}}{\partial x} + \frac{\partial \mathbf{F}}{\partial y} + \frac{\partial \mathbf{G}}{\partial z} = \mathbf{S} + \mathbf{D}_v, \tag{2.1}$$

where, $\mathbf{E}$, $\mathbf{F}$, and $\mathbf{G}$ are flux vectors, $\mathbf{S}$ is the vector of source terms related to the phase change of the fluid, $\mathbf{Q}$ is the matrix of incompressible flow variables, and $\mathbf{D}_v$ is the vector of viscous fluxes:

$$\mathbf{Q} = \begin{pmatrix} \rho_m & \rho_m u & \rho_m v & \rho_m w & \rho_g \phi_g & \rho_m h_m & \rho_m k & \rho_m \epsilon \end{pmatrix}^{\mathsf{T}}, \tag{2.2}$$

$$\mathbf{S} = \begin{pmatrix} 0 & 0 & 0 & 0 & m_t & m_t h_{fg} & S_k & S_\epsilon \end{pmatrix}^{\mathsf{T}}, \tag{2.3}$$

$$\mathbf{E} = \begin{pmatrix} \rho_m u & \rho_m u^2 + P & \rho_m uv & \rho_m uw & \rho_g \phi_g u & \rho_m h_m u & \rho_m k u & \rho_m \epsilon u \end{pmatrix}^{\mathsf{T}}, \tag{2.4}$$

$$\mathbf{F} = \begin{pmatrix} \rho_m v & \rho_m vu & \rho_m v^2 + P & \rho_m vw & \rho_g \phi_g v & \rho_m h_m v & \rho_m k v & \rho_m \epsilon v \end{pmatrix}^{\mathsf{T}}, \tag{2.5}$$

$$\mathbf{G} = \begin{pmatrix} \rho_m w & \rho_m wu & \rho_m wv & \rho_m w^2 + P & \rho_g \phi_g w & \rho_m h_m w & \rho_m k w & \rho_m \epsilon w \end{pmatrix}^{\mathsf{T}}, \tag{2.6}$$

where ρ is the fluid density, u, v, and w are the velocity components, P is pressure, k is the turbulent kinetic energy, ϵ is the turbulent dissipation, m_t is the net rate of vapor mass generation, and h_{fg} is the change in enthalpy resulting from phase change. The variables S_k and S_ϵ are the turbulent source terms related to the phase change of the fluid. For details regarding the viscous flux formulation, $\mathbf{D}_v$, the reader is directed to Hosangadi *et al.* [30]. The m subscript denotes a liquid/gas phase mixture quantity, while a g denotes a gas or vapor phase quantity. The first four equations in (2.1) are for mass conservation and the three conservation of momentum equations, respectively. The fifth equation is the mass conservation of the vapor phase, and the sixth equation is the energy equation for the mixture. The seventh and eighth equations describe the turbulent kinetic energy and turbulent dissipation rate. The variable ϕ_g is the volume fraction of the gas phase, and ϕ_l is the volume fraction of the liquid phase. The mixture density and enthalpy for the gas and liquid phases are defined using the following equations.

$$\rho_m = \rho_g \phi_g + \rho_l \phi_l \tag{2.7}$$

$$\rho_m h_m = \rho_g \phi_g h_g + \rho_l \phi_l h_l \tag{2.8}$$

$$\phi_g + \phi_l = 1 \tag{2.9}$$

For wind turbine applications, there is only a gas phase, so $\phi_l = 0$. An assumption is made that the liquid and gas phases are in thermodynamic equilibrium, and thus the thermodynamic properties can be expressed as a function of the saturation temperature. This assumption decouples the energy equation from the differential equations and results in a simpler system of equations when compared to a fully compressible solver. Because (2.1) is very stiff in its given form, the system is preconditioned to increase numerical stability. The preconditioned system is defined as

$$\Gamma \frac{\partial \mathbf{Q}_v}{\partial t} + \frac{\partial \mathbf{E}}{\partial x} + \frac{\partial \mathbf{F}}{\partial y} + \frac{\partial \mathbf{G}}{\partial z} = \mathbf{S} + \mathbf{D}_v, \tag{2.10}$$

where Γ is the preconditioning matrix defined in [28] and $\mathbf{Q}_v = [p, u, v, w, \phi_g, T, k, \epsilon]^\intercal$. The eigenvalues of the system in (2.10) now become,

$$\Lambda = (u + c_m, u - c_m, u, u, u, u, u), \tag{2.11}$$

where c_m is the acoustic wave speed of the mixture.

2.2.2 Computational Fluid Dynamics Grid

The time-accurate fluid motion about wind turbine blades is very complex and attempts to model these flows using CFD often result in large computational grids. The size and complexity of the grid arises from the need to resolve regions of the flow with large velocity gradients, such as viscous boundary layers and blade tip vortices. When fluid simulations are coupled with structural motion, they often need to be simulated for long periods of time to resolve the time scales on the order of the structural response time. Natural frequencies of large wind turbine blades such as the NREL 5-MW blades can be on the order of 0.5-1.0 Hz [21], meaning time accurate fluid simulations must be simulated for at least 1.0-2.0 seconds of physical time just to resolve a single period of vibration. Considering the maximum stable fluid time steps in CRUNCH CFD® (based on the author's experience) typically are on the order of 10^{-4} seconds, it becomes clear that these simulations are complex and demand large computational resources. Use of these fluid models in a design optimization requires the evaluation of potentially hundreds of designs, which can become computationally prohibitive. For this reason, the effort was made here to make the fluid model appropriately inexpensive such that it can be used in an optimization environment while also ensuring no fidelity or accuracy of flow physics is sacrificed. One such simplification is to solve for only the steady fluid dynamics of the blade. The steady flow analysis allows for the use of local time-stepping within CRUNCH CFD®, and much faster convergence when compared to a time-accurate fluid dynamics solution. The use of a steady flow analysis to compute the aerodynamic loading on the blade is justified since we are only concerned in obtaining a static bending displacement of the blade to quantify the amount of bend-twist coupling caused by the anisotropy in the composite layup. A validation of the steady flow analysis will be presented in §3.1.

Rotational fluid dynamics simulations can be solved using one of two methods available within CRUNCH CFD®. The first method solves the fluid equations in the inertial reference frame and the grid can be physically rotated. The disadvantage of using this method is that a constant global time step must be used throughout the simulation, which increases computational time if only the steady fluid dynamics are sought. The second method is to solve for the fluid dynamics in a rotational frame of reference. In this method, the grid is stationary and source terms are added to the fluid governing equations to simulate rotational effects. Local time-stepping is permitted when the fluid is solved in the rotational frame of reference since there is no grid motion. The method of solving the fluid in the rotational reference frame was selected for the numerical simulations in this work.

The computational fluid grid is shown in Figure 2.2. The grid consists of a single blade and a size of about 827,000 cells. Figure 2.3 shows the grid with color-coded boundaries and a legend indicating the corresponding boundary conditions. The side boundaries are positioned 120° from each other in the azimuthal direction and are assigned periodic boundary conditions. The blade surface grid consists of about 20,000 quad elements and was assigned a viscous wall boundary condition. The inflow and outflow boundaries of the grid are positioned 67.5 m (just over one blade span) on either side of the blade along the axis of rotation. The inflow velocity is set to the wind speed on the inflow boundary condition. The pressure on the inflow and outflow boundaries is set to standard sea level pressure. The assumption that inflow and outflow planes spaced sufficiently far from the turbine blades have the same pressure is also used in the derivation of the Betz limit presented in Appendix A. The boundary on the outer circumference of the domain is located at a radius of 200 m from the blade hub and is also assigned an inflow condition. This condition allows flow to either enter or exit through the boundary as necessary based on the local pressure. Finally, the hub along the axis of rotation is assigned an inviscid wall boundary condition. This was done for two reasons; to avoid any boundary layer growth since the length of the hub is not realistic, and the inviscid tag does not require a boundary layer grid on that surface. Unlike in a structured grid where finding the next cell in the normal direction to the wall is trivial, unstructured grids do not have an ordered data structure, which makes finding the next point normal to the wall boundary difficult when computing the wall shear stress. To overcome this challenge, CRUNCH CFD® requires the viscous surfaces of unstructured grids to have at least one full layer of either prism or hexahedral cells grown in the normal direction away from the surface such that the boundary layer

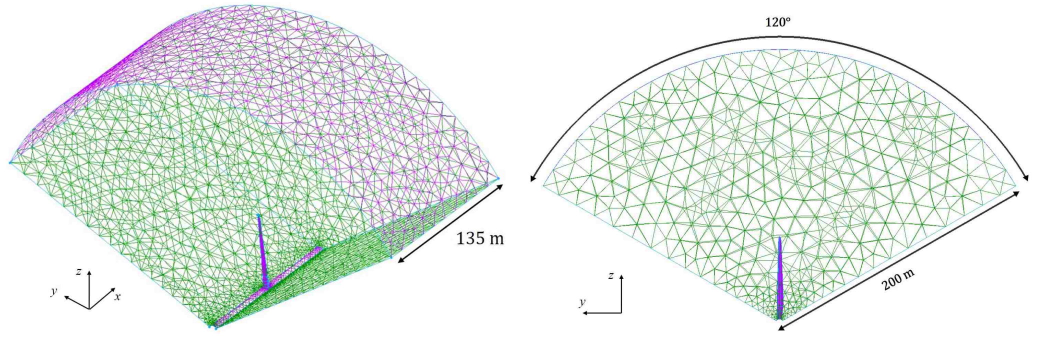


Figure 2.2: CFD grid

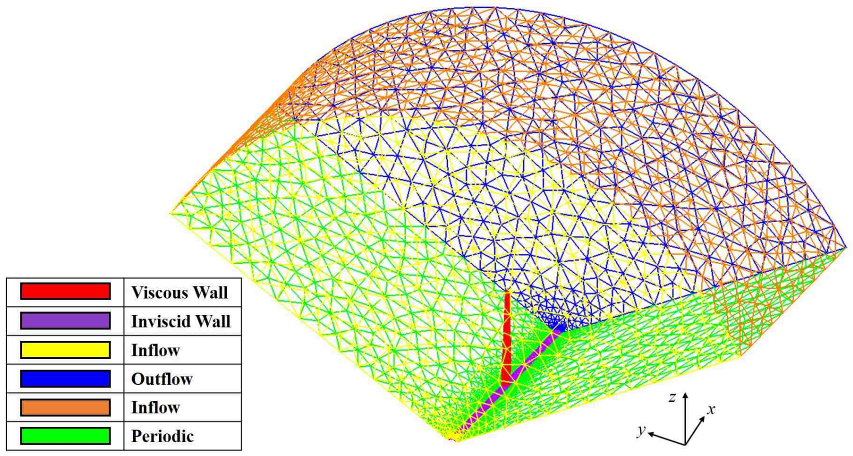


Figure 2.3: CFD grid with color-coded boundaries and a table with the corresponding boundary conditions

region is well resolved. This requirement increases the size of the grid but is avoided here with the inviscid wall assumption at the hub.

The CFD grid was built using Pointwise, a commercial grid generation package with both structured and unstructured meshing capabilities [31]. A detailed image of the hexahedral cells extruded normal to the blade surface can be seen in Figure 2.4 with the blade surface shown in red. The grid was built using Pointwise's anisotropic tetrahedral extrusion capability, called TRex [32]. TRex extrudes right-angled tetrahedral elements normal to the surface and then combines them to create either prism or hexahedral cells. Eleven full layers of hexahedral cells were extruded from the blade surface with an initial wall spacing of 0.1 mm. The hexahedral layers smoothly transition into tetrahedral cells in the

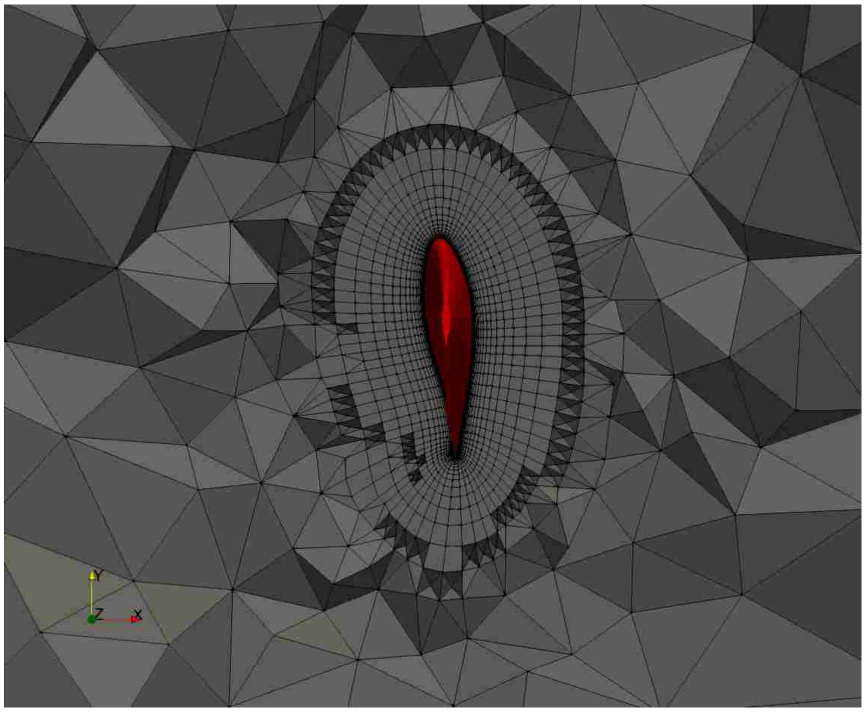


Figure 2.4: Detail of the hexahedral cells extruded normal to the surface of the blade to resolve the boundary layer on the blade, away from the hub

freestream. A dimensionless wall spacing parameter, y^+, is used to gauge the grid's ability to resolve the boundary layer at a viscous wall to accurately compute viscous forces. The y^+ value is defined as

$$y^+ = \frac{u^* y}{\nu}, \tag{2.12}$$

where here $u^* = \sqrt{\tau_w/\rho}$ is the friction velocity, y is the distance between the wall and the first grid point, ν is the kinematic viscosity, τ_w is the shear stress at the wall, and ρ is the fluid density. Generally, y^+ should be less than or equal to 1.0 to accurately resolve the viscous shear stress at the wall. The wall normal spacing of 0.1 mm corresponds to a y^+ value of 20.0 near the tip of the blade at rated wind speed. More detail about the choice of wall normal spacing and its effect on the accuracy of the simulations and the computed torque is presented later in §3.2.

2.2.3 Turbulence Modeling

The standard k-ϵ turbulence model available in CRUNCH CFD® was used with a two-layer near-wall model at the viscous wall boundary condition [33]. Although the details are not shown here, the k-ω SST turbulence model as well as wall functions were also tested for this problem, but the k-ϵ turbulence model with a near-wall model proved the most reliable match with the torque data in the Jonkman report [21] over the range of wind speeds studied. The two-layer near-wall model attempts to predict the dissipation of turbulence in the low-Reynolds number region close to the wall and provide a correction to the k-ϵ turbulence model. A cut-off level separating the inner layer and outer layer is defined based on the turbulent Reynolds number. The turbulent dissipation and eddy viscosity in the areas below the cut-off level are set based on the following equations.

$$\sigma_{\text{alg}} = \frac{k^{1.5}}{C_w y(1 - Re_k/2C_w)} \tag{2.13}$$

$$\mu_{t,\text{alg}} = C_\mu \rho \sqrt{k} \Big\{ C_w y(1 - Re_k/70) \Big\} \tag{2.14}$$

$$C_w = 0.41 C_\mu^{-0.75} \tag{2.15}$$

The inner layer solution is transitioned to the outer layer solution using the blending parameter

$$W = \frac{1}{2}\left\{ 1 + \tanh\left(\frac{Re_k - Re_{k,\text{cutoff}}}{10} \right) \right\}. \tag{2.16}$$

Finally, the turbulent dissipation rate and eddy viscosity are updated

$$\epsilon = W\epsilon + (1 - W)\epsilon_{\text{alg}}, \tag{2.17}$$

$$\mu_t = W\mu_t + (1 - W)\mu_{t,\text{alg}}. \tag{2.18}$$

At each time step, the pressure and viscous forces are calculated and summed for each cell on the blade, and the moments are calculated about the origin located at the turbine hub. The torque is the moment acting on the blade about the turbine's axis of rotation.

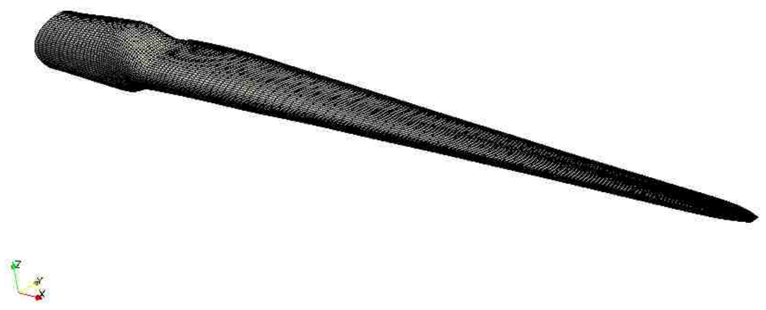


Figure 2.5: Computational structural grid of NREL 5-MW blade

The power extracted by the turbine blade can then be calculated by multiplying the torque about the rotational axis of the turbine by the rotational rate of the blade.

2.3 Structural Model

Static aerodynamic loading of the NREL 5-MW blade including the effect of gravity was simulated using Abaqus, a commercial FEA code [34]. Abaqus includes a graphical user interface (GUI) utility for generating a grid on CAD geometries, setting boundary conditions, and applying loads. This utility was not available to the author, thus the computational grid was generated in Pointwise and converted to the Abaqus input file format using the Python script provided in Appendix B. The grid generated in Pointwise consists of SR4 shell elements for the blade surfaces and SR3 shell elements on the shear web and blade tip surfaces. The structural grid for the blade surfaces matches the CFD blade surface grid point-to-point. The grid matching between the CFD and FEA grids creates an unnecessarily fine FEA grid; however, the grid matching allows for a direct transfer of pressure loads and grid displacements during the grid deformation process, without the need for interpolation. The structural grid for the blade and a cross-sectional cut through the blade can be seen in Figures 2.5 and 2.6. The structural grid has a total of 22,728 elements. The FEA grid was constructed in and exported from Pointwise in the Gmsh file format.

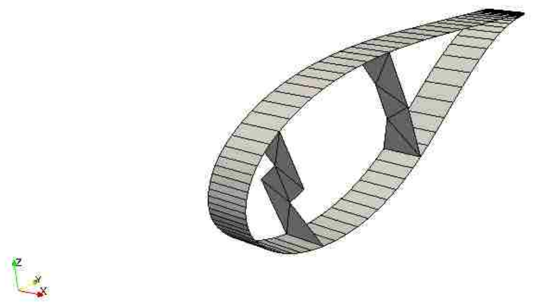


Figure 2.6: Cross-sectional cut of computational structural grid

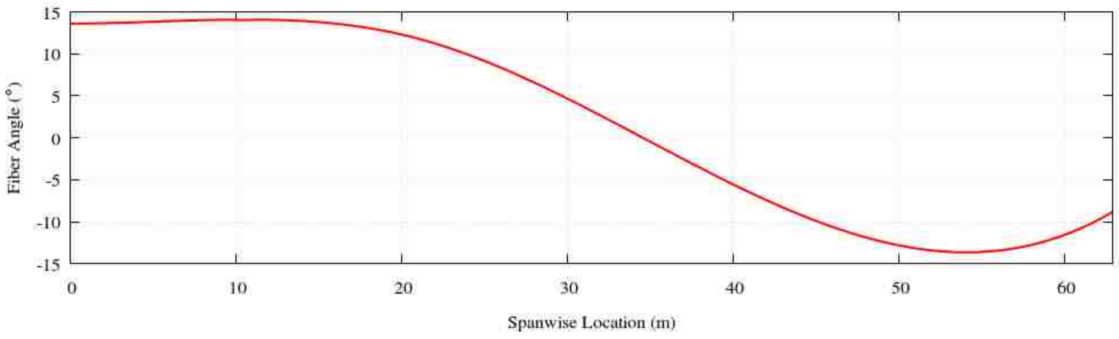
This format was selected because it is an ASCII formatted file which supports the export of two-dimensional shell elements and the specification of boundary and volume conditions. Boundary conditions on two-dimensional elements are applied to line elements. Volume condition specifications are used to tag areas in the grid based on material properties or construction. For example, the blade skin has a different construction than the spar and shear web regions and so they are tagged with different volume conditions. When generating the grid, it is critical to ensure all element normals are pointing in the correct direction. If the normals are not consistent, the pressure loads will be applied in the wrong direction in the Abaqus simulation. The Abaqus input file is a single ASCII formatted file containing all nodal coordinates, element connectivity, material specification, and analysis parameters.

The material properties and blade construction used in all optimizations presented in this work were based on those used by Cox and Echtermeyer [35] and can be found in Table 2.2. The shear webs consist of a 30-mm-thick core material with four layers of $+45°$ carbon fiber on one side and four layers of -45° carbon fiber on the other. The spar flanges have the same construction as the shear webs, but the carbon fiber on each side of the core material can take on any orientation between $-90°$ and $+90°$. The fiber angles in this layer are what will be optimized to implement the bend-twist coupling in the blade. The blade skin consists of a 2 mm thick layer of carbon fiber and a 1 mm thick layer of fiberglass epoxy

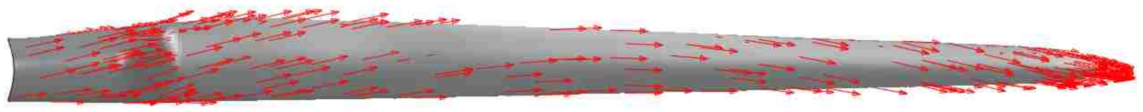
Table 2.2: Blade material properties

Material	E_{xx} (GPa)	E_{yy} (GPa)	G_{xy} (GPa)	ν	ρ (kg/m^3)
Carbon Fiber	135.0	10.0	5.5	0.3	1560
Fiberglass Resin	41.0	9.0	4.1	0.3	1890
Foam Core	0.25	0.25	0.073	0.35	200
Lining	9.65	9.65	3.86	0.3	1670
Gel Coating	3.44	3.44	1.38	0.3	1230

resin each at a particular orientation. On top of the fiberglass epoxy resin is a layer of core material, a 0.38 mm thick layer of lining material, and a 0.51 mm thick layer of gel coating which creates a smooth aerodynamic outer surface on the blade. Each layer in the Abaqus model uses 3 integration points through the thickness of the shell element. The layer of core material varies in thickness along the span of the blade from 15 mm at the root to 5 mm at the tip. Density specification is required to simulate the effects of gravity in Abaqus. The material orientations of the carbon fiber and fiberglass epoxy resin in the blade skins and spars vary smoothly along the span, mimicking VAT composites [36]. VAT composites and variable thickness in the core material are handled in Abaqus on an element-by-element basis using distribution tables. The Python script in Appendix C was written to determine the fiber angle and core thickness for each element based on its spanwise location. As an example, consider a function of fiber angle versus span such as that in Figure 2.7a. The corresponding elemental fiber orientations on the surface of the blade are shown in Figure 2.7b. As discussed in §1.1.2, bend-twist coupling is built into the blade by orienting the fibers on the bottom surface of the blade to the negative angle obtained from the function in Figure 2.7a. For example, if the fiber angle at a spanwise location of 45 m is -10° on the top surface, the fiber angle on the bottom surface at the same spanwise location is 10°. Loads are applied to each element as a distributed load. The pressure is extracted from the corresponding cell on the CFD surface grid and applied to the Abaqus grid as a gauge pressure using the Python script provided in Appendix D. Figure 2.8 shows the gauge pressures extracted from the CFD simulation, which are applied to the blade as elemental distributed loads. As mentioned previously, care was taken in generating the Abaqus grid



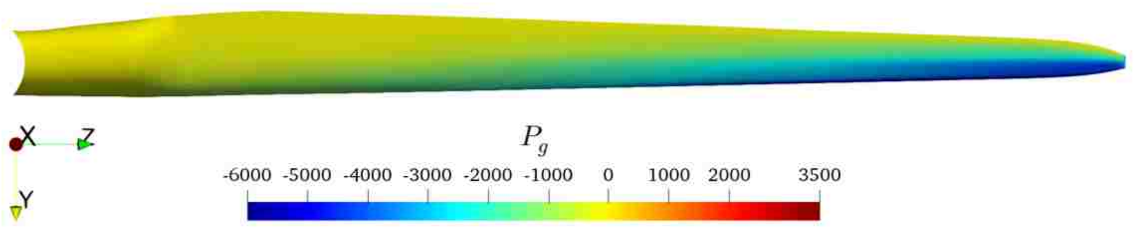
(a)



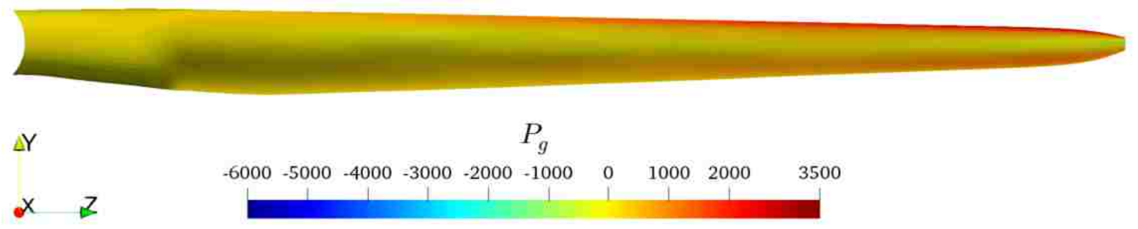
(b)

Figure 2.7: An illustration of the composite fiber specification along the blade span using a smooth function: (a) function of blade fibers versus spanwise element location; (b) corresponding fiber locations along blade span

to ensure that all element normals on the blade surface point away from the blade and into the fluid such that all pressure loads are applied in a consistent manner. An encastre boundary condition was applied to the nodes on the blade root on both the skin and shear webs. This condition forces all degrees of freedom at these nodes to be exactly zero. Gravity is also applied as a separate distributed load. The blade is oriented such that the spanwise direction is aligned with the positive inertial x-axis and the chordwise direction is aligned with the positive inertial y-axis. Gravity is applied in the negative y-direction. Note that this orientation differs from the blade orientation in the fluid calculation seen in Figure 2.8. These orientations simulate loading when the blade spanwise direction is directed to the left when observing the turbine in the direction of the wind along the axis of rotation. This orientation was selected to maximize the effect of blade bending due to gravity, and the corresponding results are presented in detail in the §3.1. Abaqus results are stored in a binary output database file. User-specified variables can also be output in ASCII file format to a data file. For ease of data extraction, the displacement at each node and von Mises stresses at each element were selected to be output to the ASCII data file.



(a)



(b)

Figure 2.8: Surface pressures extracted from CFD simulation for (a) top and (b) bottom surfaces of the NREL 5-MW blade

2.4 Grid Deformation

The coupling between the aerodynamics and structural response in a fluid-structure interaction (FSI) problem can be approached one of two ways: one-way coupled or two-way coupled [37]. In a one-way coupled FSI simulation, the structure responds to the fluid loads, but the fluid domain is not changed based on the structural response. A one-way coupled approach is beneficial when the deformation of the structure is sufficiently small such that the fluid response due to the deformation is small or negligible. When the deformations of the structure are large and have an appropriate effect on the surrounding flow, a two-way coupled FSI approach is adopted in which the fluid responds to changes in the structure by adapting the CFD domain according to the structural deformation. Two-way coupled simulations are more computationally expensive due to the deformation of the CFD grid. In the case of the simulation of wind turbine blades, the tip displacements of the deformed blade can be on the order of several meters and bend-twist coupling effects will create a significant change in aerodynamic loading on the blade. For this reason, the correct approach is to use two-way coupling by deforming the CFD grid and is used in all analyses presented in this work.

A standalone grid deformation utility was developed to adapt the CFD mesh based on the updated blade displacements. The domain of the CFD mesh is treated as a linear-elastic material [38], and deformations are solved for based on the Stiffness Method [39]. The grid deformation utility solves

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{F} \tag{2.19}$$

on an all tetrahedral mesh, hereafter referred to as the deformation grid. Equation (2.19) is a linear system in which the vector $\boldsymbol{u}$ is a vector of the degrees of freedom for each node in the system, $\boldsymbol{K}$ is the stiffness matrix, and $\boldsymbol{F}$ is the vector of reaction forces. The spatial domain of the deformation grid is the same as the CFD grid. The points on the blade in the deformation grid match the CFD grid point-to-point but is coarsened everywhere else. The deformation grid contains only about 100,000 tetrahedral cells and 26,000 nodes while the CFD grid contains over 800,000 tetrahedra, pyramids, prisms and hexes and over 500,000 nodes. The level of refinement in the CFD grid is much too high for what is necessary to solve the spatial deformation problem (2.19) and would increase the computational cost substantially. Once (2.19) is solved on the deformation grid, the solution is interpolated to the CFD grid using a barycentric interpolation [40]. The storage size of the stiffness matrix is of order $O((3n)^2)$ where n is the number of nodes in the deformation grid. Boundary conditions are applied directly as prescribed displacements. At the boundaries of the CFD domain, namely the inflow/outflow, freestream, wind turbine hub, and periodic boundaries, the nodes are held fixed by setting all three degrees of freedom at that node to zero. When displacements at the boundaries are zero, they are treated as homogeneous boundary conditions [39]. A desired consequence of fixed nodes is that the row and column in the linear system corresponding to that degree of freedom can be removed from the stiffness matrix, thus making the dimension of the system of equations smaller. The boundary condition for nodes on the blade surface are still applied as a prescribed displacement, but they are non-homogeneous since the displacements are non-zero. In the case of a non-homogeneous boundary condition, the row and column of that degree of freedom can be removed, similar to homogeneous boundary conditions, but the reaction force must be solved for at each node and moved to the right-hand side of the equation.

The standalone grid deformation utility is written in Fortran using MPI to decrease the computational time to solve the system and reduce the size of memory required to store the stiffness matrix. The code consists of two main parts: the construction of the stiffness matrix, and the solution of the resultant linear system. A stiffness matrix is constructed for each element using

$$\boldsymbol{K}^e = \int_{\omega^e} \boldsymbol{B}^T \boldsymbol{\xi} \boldsymbol{B} = V \boldsymbol{B}^T \boldsymbol{\xi} \boldsymbol{B}, \tag{2.20}$$

where $\boldsymbol{K^e}$ is the stiffness matrix of the element and $\boldsymbol{\xi}$ is the stiffness of the element [40]. The matrix $\boldsymbol{B}$ relates the displacements at the nodes to the elemental strains and is defined as

$$\boldsymbol{B} = \frac{1}{6V} \begin{bmatrix} \boldsymbol{J}^{-1}_{12} & 0 & 0 & \boldsymbol{J}^{-1}_{22} & 0 & 0 & \boldsymbol{J}^{-1}_{32} & 0 & 0 & \boldsymbol{J}^{-1}_{42} & 0 & 0 \\ 0 & \boldsymbol{J}^{-1}_{13} & 0 & 0 & \boldsymbol{J}^{-1}_{23} & 0 & 0 & \boldsymbol{J}^{-1}_{33} & 0 & 0 & \boldsymbol{J}^{-1}_{43} & 0 \\ 0 & 0 & \boldsymbol{J}^{-1}_{14} & 0 & 0 & \boldsymbol{J}^{-1}_{24} & 0 & 0 & \boldsymbol{J}^{-1}_{34} & 0 & 0 & \boldsymbol{J}^{-1}_{44} \\ \boldsymbol{J}^{-1}_{13} & \boldsymbol{J}^{-1}_{12} & 0 & \boldsymbol{J}^{-1}_{23} & \boldsymbol{J}^{-1}_{22} & 0 & \boldsymbol{J}^{-1}_{33} & \boldsymbol{J}^{-1}_{32} & 0 & \boldsymbol{J}^{-1}_{43} & \boldsymbol{J}^{-1}_{42} & 0 \\ 0 & \boldsymbol{J}^{-1}_{14} & \boldsymbol{J}^{-1}_{13} & 0 & \boldsymbol{J}^{-1}_{24} & \boldsymbol{J}^{-1}_{23} & 0 & \boldsymbol{J}^{-1}_{34} & \boldsymbol{J}^{-1}_{33} & 0 & \boldsymbol{J}^{-1}_{44} & \boldsymbol{J}^{-1}_{43} \\ \boldsymbol{J}^{-1}_{14} & 0 & \boldsymbol{J}^{-1}_{12} & \boldsymbol{J}^{-1}_{24} & 0 & \boldsymbol{J}^{-1}_{22} & \boldsymbol{J}^{-1}_{34} & 0 & \boldsymbol{J}^{-1}_{32} & \boldsymbol{J}^{-1}_{44} & 0 & \boldsymbol{J}^{-1}_{42} \end{bmatrix}, \tag{2.21}$$

where the entries of $\boldsymbol{B}$ are found by inverting the Jacobian of the tetrahedral element. The Jacobian is defined as

$$\boldsymbol{J} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix}, \tag{2.22}$$

where x_i, y_i, and z_i are the coordinates of the tetrahedral element's vertices [40]. Once the stiffness matrix is constructed for the element, it is added to the global stiffness matrix in (2.19) [39]. The resulting linear system in (2.19) is solved iteratively using a conjugate gradient method with a Jacobi preconditioner [41]. The conjugate gradient method is a

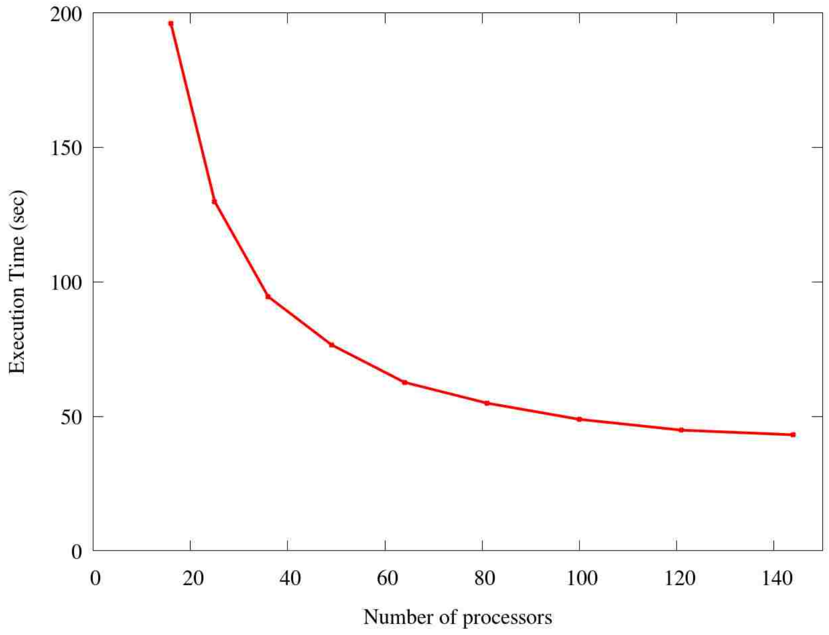


Figure 2.9: Execution time versus number of processors for parallel grid deformation tool

Krylov subspace method with the benefit that only matrix-vector operations are required instead of much more computationally intensive matrix-matrix operations. A requirement for the use of the conjugate gradient method is that the stiffness matrix must be symmetric and positive-definite, both of which are properties of the global stiffness matrix. Figure 2.9 shows the execution time of the parallel grid deformation tool versus number of processors, and Figure 2.10 shows the speedup of the utility versus number of processors. Speedup is defined here as

$$S = \frac{T(1)}{T(N)} \tag{2.23}$$

where S is the speedup, and $T(n)$ is the execution time when n processors are used. It can be seen that the speedup trend with respect to the number of processors is nearly linear up to about 40 processors, after which the speedup curve begins to fall away from the ideal speedup trend. The falloff in the speedup curve indicates that the code efficiency begins to decrease with increasing number of processors. The fall in efficiency can be attributed to the increased number of messages being passed between processors.

Once (2.19) is solved on the deformation grid, the displacements are transferred to the

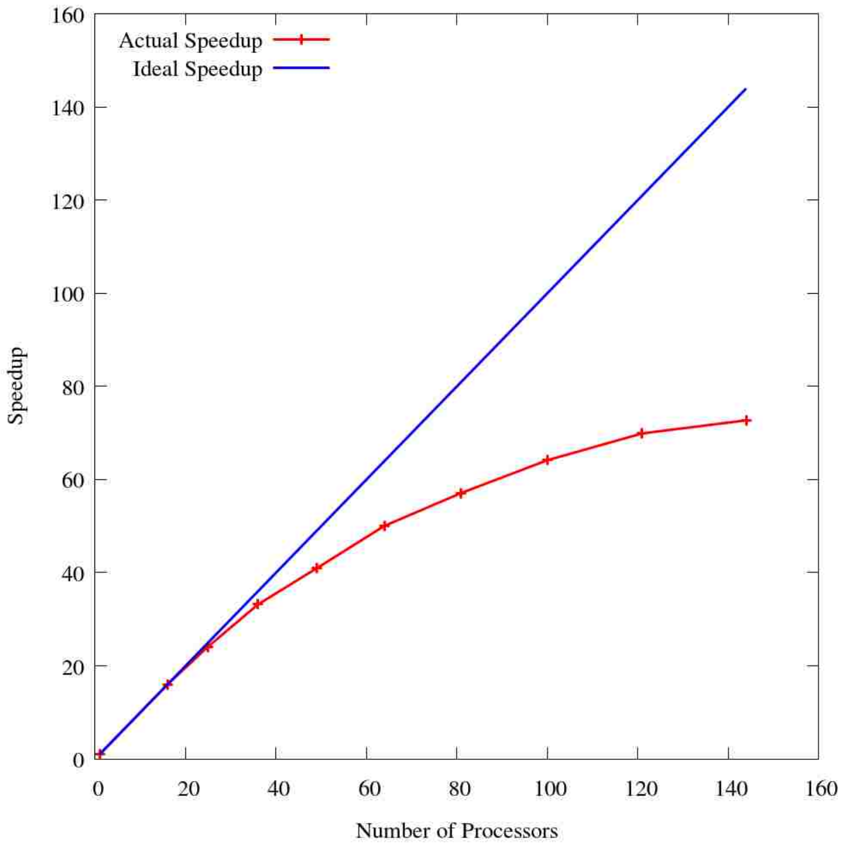


Figure 2.10: Speedup versus number of processors for parallel grid deformation tool

finer CFD grid using a barycentric interpolation described here [40]. Since the solution to Equation (2.19) is stored at every node, the solution at any point on or within the boundaries of a given cell can be linearly interpolated using the element's natural coordinates. The deformation grid is made up of entirely tetrahedral cells to simplify this process. The first step in the interpolation is to determine which cell in the deformation grid contains the point of interest in the CFD grid. A KDTree nearest-node search of the cell centers of the deformation grid is performed to determine the possible cells that may contain the point of interest [42]. For each cell, the natural coordinates for the point are computed using

$$\boldsymbol{\zeta} = \boldsymbol{J}^{-1}\boldsymbol{x} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ x \\ y \\ z \end{bmatrix}, \tag{2.24}$$

where x, y, and z is the point in the CFD grid. If all four natural coordinates are greater than or equal to zero, the point is located on or within the cell boundaries. Once the containing cell is found, the barycentric interpolation can be performed to find the displacements at the point in the CFD grid by solving (2.25).

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} u_{x1} & u_{x2} & u_{x3} & u_{x4} \\ u_{y1} & u_{y2} & u_{y3} & u_{y4} \\ u_{z1} & u_{z2} & u_{z3} & u_{z4} \end{bmatrix} \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \\ \zeta_4 \end{bmatrix}, \tag{2.25}$$

where u_{xi}, u_{yi}, and u_{zi} are the displacements at the cell vertices and ζ_i are the cell natural coordinates [40].

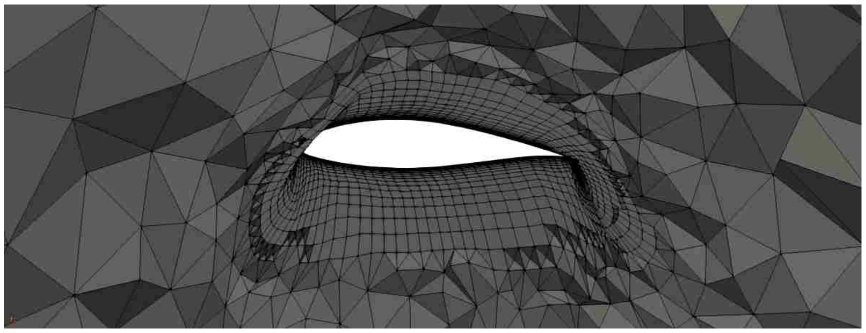
The element stiffness, $\boldsymbol{\xi}$, is a diagonal 6×6 matrix. The first three entries along the diagonal are the moduli along the principal directions. The element is modeled as an isotropic medium; therefore, all three elements are equal. The last three entries along the diagonal are the shear moduli and are set to zero. Initially, a modulus of elasticity of 1.0

was used for every cell in the deformation grid. This caused cells in areas of high curvature, such as near the leading and trailing edge of the blade, to experience high skew and result in negative cell volumes. A negative cell volume is caused when the Jacobian of the cell is negative and must be resolved in order to perform CFD on the grid. A slice through the CFD grid at a spanwise location of 60 m deformed with a constant stiffness of 1.0 is shown in Figure 2.11a. The elements in the grid near the leading and trailing edges of the blade experience high skew, and some elements result in negative cell volumes. In addition, it can be seen that the orthogonality of the cells in the boundary layer is degraded near the midchord. This can affect the accuracy of the computation of the viscous stresses on the blade which are critical in the calculation of torque. Figure 2.11b shows a slice through the CFD grid at the same spanwise location after the grid was deformed with element stiffnesses equal to the inverse of the cell volume. This modification results in a deformed grid with much higher grid quality in the boundary layer region around the blade. By setting the element stiffness equal to the inverse of the cell volume, small cells in the boundary layer region are very stiff and resistant to deformation preserving their shape. Larger cells in the freestream become less stiff and more compliant which is desirable since those regions of the fluid are much more resilient to changes in cell size and shape.

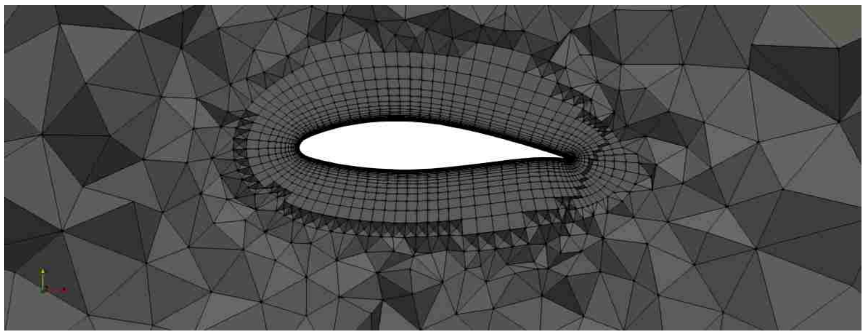
Figure 2.12 compares the convergence of the stiffness method when the element stiffness in each of the three principal directions is set to 1.0 and the inverse of the cell volume. The solution converges over 2.5 times faster with a much smoother convergence history when an element stiffness of 1.0 is used. While the convergence is much better with an element stiffness of 1.0, it cannot be used since it results in CFD meshes with poor mesh quality.

2.5 Fluid-Structure Coupling

This section details the procedure utilized to couple the deformations of the structure with the aerodynamic model. For a given wind condition, a steady CFD simulation is conducted on a rigid, non-deforming blade to convergence. The pressure loading from the rigid CFD is then used for each subsequent deforming blade simulation and the solution is used as



(a)



(b)

Figure 2.11: Comparison of slices through the deformed CFD grid when the element stiffness in the three principal directions is set to (a) 1.0 and (b) the inverse of the cell volume. Slices are located at a spanwise section of 60 m.

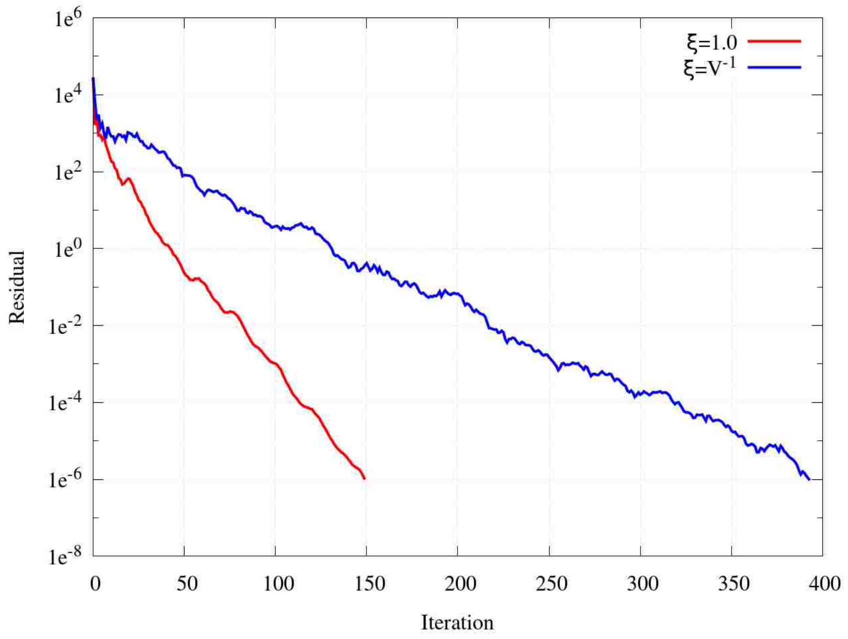


Figure 2.12: Comparison of stiffness method convergence when the element stiffness in the three principal directions is set to 1.0 and the inverse of the cell volume

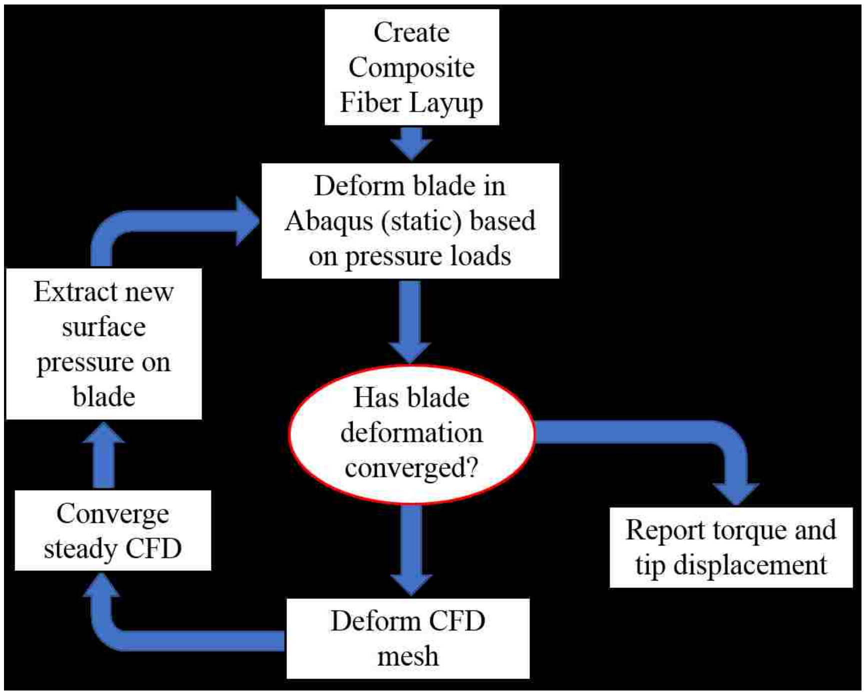

Create Composite Fiber Layup
Deform blade in Abaqus (static) based on pressure loads
Extract new surface pressure on blade
Has blade deformation converged?
Report torque and tip displacement
Converge steady CFD
Deform CFD mesh

Figure 2.13: Information flow chart of the two-way fluid-structure coupling. Blade deformation convergence is defined as less than a 0.1% change in blade tip displacement between successive iterations.

a restart file to expedite design evaluations. Abaqus is invoked at the beginning of the design evaluation to deform the blade based on the pressure loading from the rigid blade CFD simulation. Displacements from the Abaqus solution are then extracted and used as boundary conditions to solve the stiffness method on the deformation grid. Displacements are transferred between the structural, deformation, and fluid mesh directly without the need for interpolation, since the fluid and structural grids match point-to-point at their interface. Displacements at the interior of the CFD mesh are mapped from the deformation grid using a barycentric interpolation, as described in §2.4. After the CFD grid has been deformed, a steady CFD simulation is restarted and run until the computed torque converges. After several iterations of this process, the structural deformation of the blade will reach a converged state, in which the aerodynamic forces are balanced by the stiffness of the blade. At that point, performance criteria such as the torque and blade tip displacement are extracted and reported to the optimizer. Figure 2.13 illustrates the design evaluation loop.

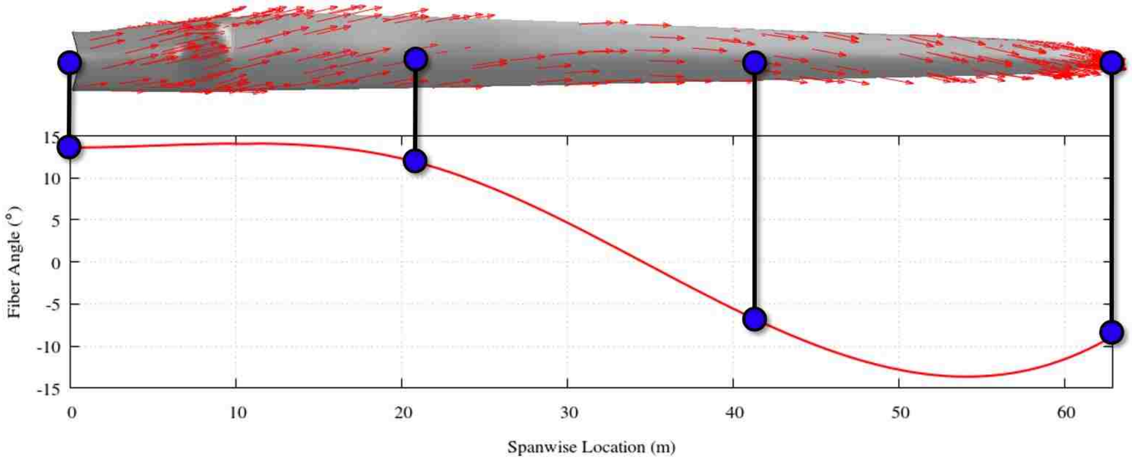


Figure 2.14: Illustration of the four design variables in the optimization. Design variables, represented as the blue dots, are used to specify the fiber orientation at discrete locations along the blade span.

2.6 Optimization

The goal of each optimization is to maximize the torque produced by the wind turbine blade. That objective is achieved by varying the composite fiber layup in the skin and spar sections along the span of the blade. The fiber orientations are specified at some number of points equally spaced along the span. These points where the fiber orientation is explicitly defined are the design variables in the optimization. The design variables are illustrated as the blue dots in Figure 2.14. A cubic spline interpolation is performed between the points were the fiber orientation is explicitly defined to get the fiber orientation at any point along the span. A constraint is enforced to ensure the blade tip displacement does not exceed 70% of the distance between the undeformed blade tip and the wind turbine tower. The optimization problem is formally stated as

$$\begin{aligned} \min \quad & \frac{T_{\text{baseline}} - T}{T} \\ \text{s.t.} \quad & 0.7c - d_{\text{tip}} \geq 0, \end{aligned} \tag{2.26}$$

where T_{baseline} is the torque from the rigid blade CFD simulation, T is the torque of the deformed blade, d_{tip} is the blade tip displacement, and c is the clearance between the blade tip and the tower. Most optimizers work by minimizing the objective function. An objective function can be maximized with a minimizer by returning its negative value. In the case of this optimization, if a design produces higher torque, the objective function will become more negative, and the minimizer will seek the most negative value which corresponds to the largest percent increase in torque.

Optimizer selection is a critical step in setting up an optimization problem. Various optimization solvers are available in the SciPy Python library [43]. Each solver is generally developed to solve a specific class of optimization problem. Two general classifications of optimizers are global and local, or gradient-based, solvers. Global optimizers, such as genetic algorithms, are capable of finding the global extrema in a design space but come with the disadvantage that they are generally slow to converge. Gradient-based solvers use the first and second derivatives of the objective function with respect to the design variables to optimize the objective function, but they will only converge to the closest local extrema and are not guaranteed to find the global extrema if the two are not equal. Generally, gradient based optimizers converge much more rapidly than genetic algorithms.

For this particular problem, a gradient-based optimizer was selected. This decision was made because the gradient of the objective function is expected to behave smoothly as a function of the design variables. Additionally, an assumption was made that only one minimum exists in the design space, as opposed to many local sub-optimal minima.

The optimization problem can be classified as a constrained nonlinear minimization. SLSQP is a nonlinear constrained optimization solver in SciPy which can handle the problem in (2.26). At each design iteration, SLSQP solves a subproblem to determine a search direction and performs a line search [44]. The subproblem is defined as

$$\begin{aligned} &\text{minimize} && g_k^T + \frac{1}{2} p^T B_k p \\ &\text{s.t.} && A_k p \geq -c(x), \end{aligned} \tag{2.27}$$

where g_k is the gradient of the objective function with respect to the design variables, p is the search direction, A_k is the Jacobian of all the constraints with respect to the design variables, and B_k is the Hessian of the Lagrangian. The Lagrangian is a single expression relating the objective function and the inequality constraints defined as

$$\mathcal{L}(x, \lambda) = f(x) - \lambda^T c(x), \tag{2.28}$$

where x is the design variable, $f(x)$ is the objective function, $c(x)$ is the constraint function, and λ is the Lagrangian multiplier. Once a search direction is found, a line search is performed. Nemhauser [44] defines a line search as a march of the solution in the direction which decreases $f(x)$. Since the gradient and Hessian of both the objective function and constraints are unknown analytically, they must be computed numerically. This computation is done by perturbing one design variable at a time by a small amount and evaluating the objective function and constraints. Constructing the gradient with respect to the design variables represents a significant cost. For example, in a case with four design variables, a single design iteration consists of four function evaluations to construct the gradient and at least one more function evaluation to evaluate the resulting search direction, p. The step size in the direction of p is reduced until the objective function is improved. If the objective function is not improved greater that a certain tolerance, in this case 0.01%, the optimizer stops and returns the solution with the most optimal objective function.

Chapter 3

Results

The first section of this chapter details the validation of the fluid and structural models of the wind turbine blade. A study is also presented which explores the effect of wall spacing on the blade in the CFD grid on the computed torque values in §3.2. Next, the results from four optimizations at discrete design points between cut-in and rated wind speeds are presented, including a discussion of the trend observed in optimal composite fiber angle variation along the blade span as a function of wind speed in §3.3.1. Based on the results of the four optimizations, two composite fiber orientations are designed to maximize the power extraction of the turbine blade for all wind speeds considered between cut-in and rated operation in §3.3.2. Finally, a study is conducted to determine the sensitivity of the optimal solution to the selection of optimization parameters such as the number of design variables and their initial conditions in §3.3.3.

3.1 Model Validation

In general, refinement of the computational models necessary to be able to capture and resolve greater physical detail and complexity increases computational cost. Typically, designing the computational model involves a trade-off between what physics are important to capture to effectively analyze the problem, and the amount of computational resources available such as time and computing power. In the case of a design optimization, one

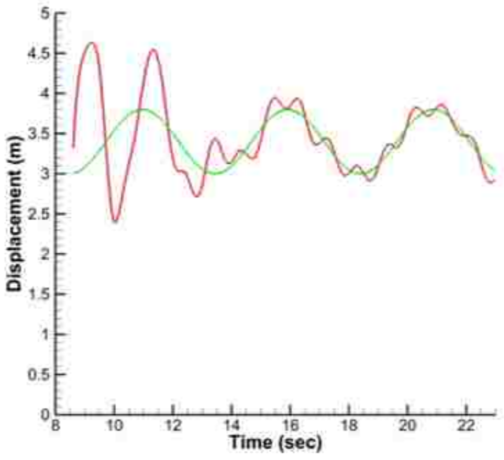
Table 3.1: Comparison of steady-state torque in kNm at rated wind speed (11.4 m/s) with average torque from Hsu and Bazilevs time-accurate simulations [24]

	Hsu & Bazilevs	Barr	% Difference
Rigid	3735.0	3799.28	1.69%
Flexible	3749.0	3855.09	2.75%

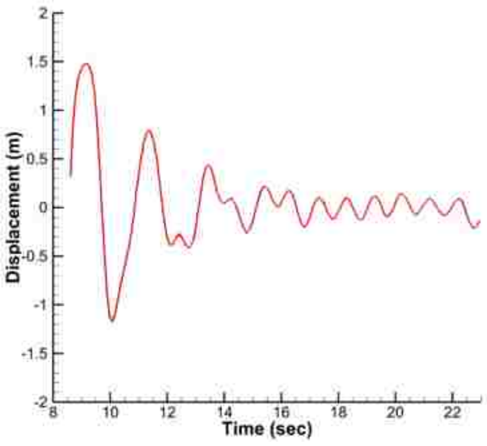
must consider not only the computational cost of a single CFD and structural simulation, but rather the cost of potentially hundreds of them. For this reason, the complexity of the computational model for both the fluid dynamics and structural dynamics must be carefully considered.

With the goal of performing design optimization of wind turbine blades using CFD, it is desirable to solve for the steady-state fluid dynamics and consider only the static loading of the blade in the structural solver. Since we are only concerned with optimizing the degree of bend-twist coupling in a blade with composite materials subject to static loading, neglecting the unsteady dynamics of the fluid can be justified. These simplifications reduce the computational cost of analyzing the wind turbine blade and make design optimization using CFD feasible, but the model must be validated to ensure the necessary level of physics is still captured to analyze the problem accurately.

A validation case was conducted at the rated wind speed (11.4 m/s) of the NREL 5-MW wind turbine matching the flow conditions of the Hsu and Bazlievs's [24] wind turbine case in which they consider all three turbine blades at once in their simulations. The results from the validation case correspond to only a single blade, thus the torque is multiplied by three to make a comparison of the representative torque from the full wind turbine. Table 3.1 shows a comparison of torque with Hsu and Bazilevs's results at rated wind speed for rigid (non-deforming) or flexible rotor blades. The key difference in the comparison of the two solutions is that Hsu and Bazilevs simulation was time-accurate, and all torque values shown are average values. The torque values obtained with CRUNCH CFD$^{®}$ are within 1.7% and 2.75% for the rigid and flexible blades, respectively. Hsu and Bazilevs [24] simulated all three wind turbine blades rotating in the inertial reference frame including the wind turbine hub. Hsu and Bazilevs also included the unsteady effects of the fluid



(a)



(b)

Figure 3.1: Frequency analysis of Hsu and Bazilevs unsteady blade tip displacement data. The red line in (a) is the blade tip displacement over time computed by Hsu and Bazilevs [24]; the green line is a sinusoidal function matching the phase and amplitude of the blade tip displacement curve of a frequency equal to the blade passage frequency. The residual displacement signal when the green line is subtracted from the red line is shown in (b).

flow and the blade structural dynamics. Their results of blade tip displacement over time are shown in Figure 3.1. The tip deflection of the blade presented by Hsu and Bazilevs shows an unsteady behavior. However, if a sinusoidal function with a frequency equal to the blade passage frequency and an amplitude equal to the maximum tip displacement is subtracted from the tip displacement curve, the blade passage frequency mode is removed and the result is shown in Figure 3.1b. The sinusoidal function is shown as the green line in Figure 3.1a. The curve in Figure 3.1b has a much higher frequency and smaller amplitude. This proves that only two modes are responsible for perturbing the blade tip displacement from the average value in Hsu and Bazilevs's results. The first mode has a frequency equal to the blade passage frequency and is likely caused by the effect of gravity changing the bending load as the blade rotates through different angular positions. The residual signal is much higher in frequency and smaller in amplitude and can be reasonably neglected.

The results of Hsu and Bazilevs [24] show that the periodic change in the tip-displacement due to gravity as the blade rotates is significant. Therefore, a validation of the steady-state aerodynamics and static blade deformation was performed to ensure that this effect is accounted for in the model. Steady-state aerodynamics were simulated using CRUNCH

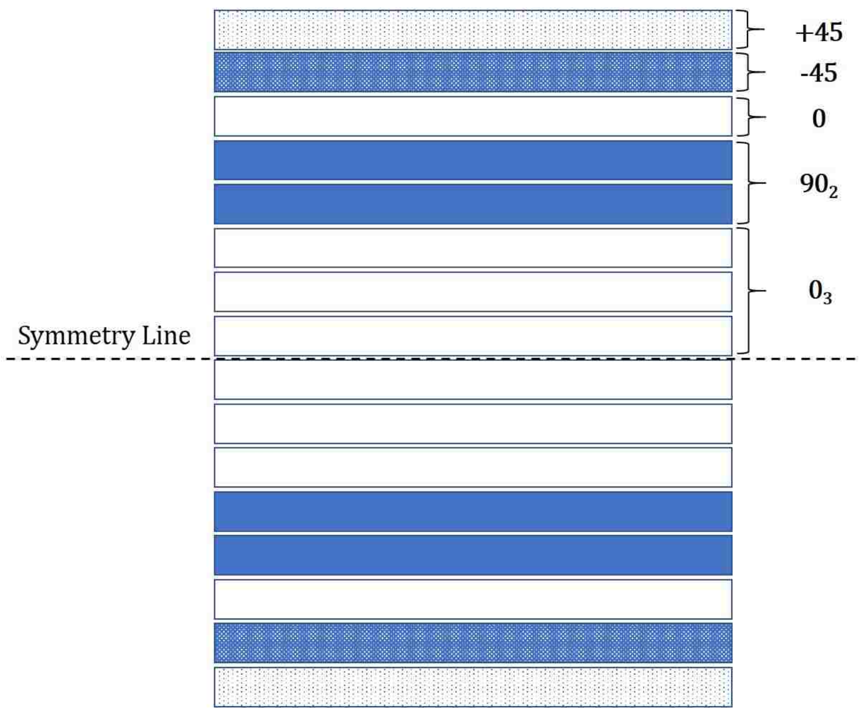


Figure 3.2: An illustration of the composite layup used in the blade skin used by Bazilevs *et al.*[1]

CFD® and static blade displacements were solved for with Abaqus including gravitational effects at four different angular blade positions and compared with Hsu and Bazilevs's results.

For the purposes of validating the structural model, the composite layup was modified to match that used by Bazilevs; specifically, the internal blade spars were removed and a symmetric composite layup was used for the blade skin[1]. The stacking sequence notation of the composite layup was defined as $[\pm 45/0/90_2/0_3]_s$. Each number in the stacking sequence notation denotes the orientation of a unidirectional layer of composite material, and the subscripts denote the number of those layers stacked consecutively. An illustration of the composite layup is shown in Figure 3.2. Note that the $0°$ fibers in Bazilevs's analysis were oriented in the chordwise direction, as opposed to the validation case where $0°$ corresponds to a spanwise orientation. The material properties are provided in Table 3.2.

Figure 3.3b shows the resulting blade tip displacements at the four blade angular po-

[1]This composite structural model was only used for the purposes of validation. Refer to §2.3 for details about the structural model used in all optimizations presented in this chapter.

Table 3.2: Material properties of composite layup in blade skin used by Bazilevs *et al.* [23][1]

E_1 (GPa)	E_2 (GPa)	G_{12} (GPa)	ν_{12}	ρ ($\frac{kg}{m^3}$)
39.0	8.6	3.8	0.28	2100

sitions depicted in Figure 3.3a plotted with Hsu and Bazilevs's tip displacement results. It is seen that at 90° and 270° orientations the tip displacement is nearly equal to the average tip displacement, but at 0° the effect of gravity decreases the blade bending and is increased at 180°. Since blade bending is maximum at a blade orientation of 180°, static blade loading will be considered at that blade orientation in all subsequent analyses to provide a conservative numerical estimate in the design optimizations.

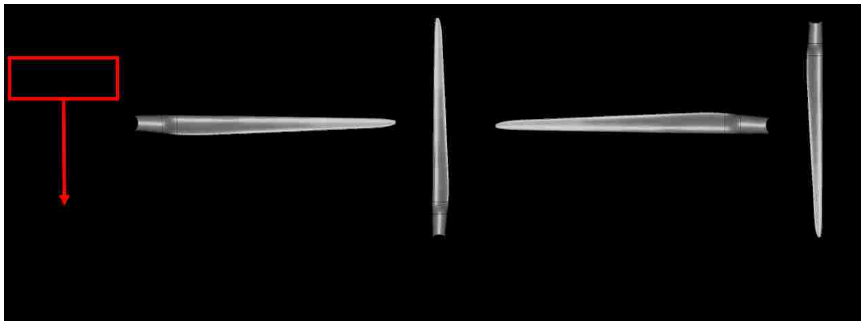
3.2 Wall Spacing Effects on Torque

The power extracted by a wind turbine blade is linearly proportional to the torque produced about the turbine's axis of rotation,

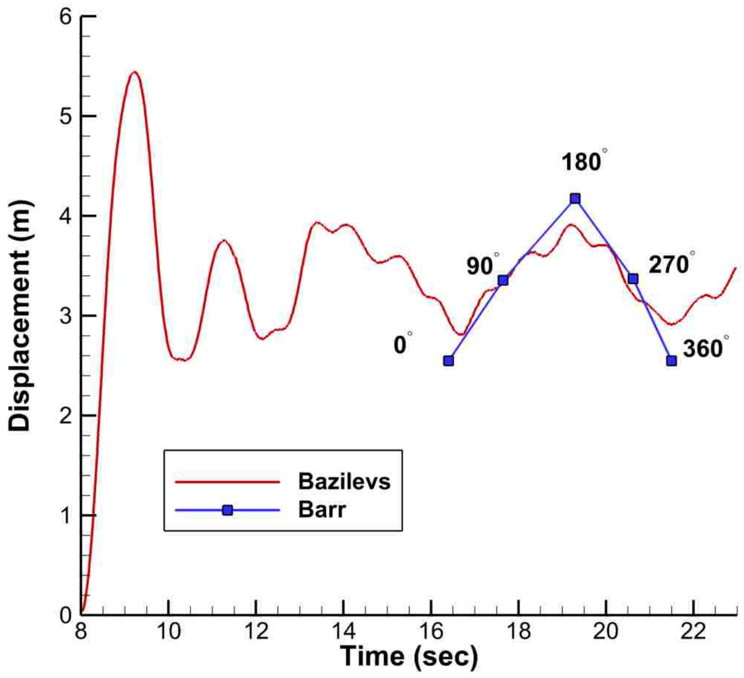
$$P = T\dot{\theta}, \tag{3.1}$$

where P is the power extracted by the turbine, T is the torque about the axis of rotation, and $\dot{\theta}$ is the rotational speed of the turbine blade in radians per second. The torque is the moment perpendicular to the rotor plane acting about the rotational axis of the turbine. The force vector at each point is the sum of the pressure force acting normal to the surface, and the viscous force acting tangent to the surface.

Accurate calculation of the viscous forces acting on the blade using CFD requires a grid with a very small wall normal spacing. If the first grid point off the wall is too large, the boundary layer will not be resolved correctly, and the wall shear stress will be underestimated. Generally speaking, the smaller the wall normal spacing, the larger the cell count of the CFD grid and more computational resources are necessary. The wall normal spacing should be set such that $y^+ \leq 1.0$ in order to accurately compute the shear stress at a viscous wall. In a rotational machine, such as a wind turbine, y^+ is not constant along the span of the blade due to the increase in the local wind speed and Reynolds



(a)



(b)

Figure 3.3: Gravitational effect on blade tip displacement as a function of angular position. (a) Four blade positions tested in relation to the force of gravity, and (b) Comparison of tip displacement with Hsu and Bazilevs's results [24]. Static steady-state tip displacements computed at four discrete angular positions are compared with Hsu and Bazilevs's unsteady simulations.

Table 3.3: Table of torque values for $y^+ = 1.0$ and $y^+ = 20.0$ and NREL report [21]

Wind Speed (m/s)	NREL Torque (kNm)	$y^+ = 1.0$ Torque (kNm)	$y^+ = 1.0$ Percent Difference	$y^+ = 20.0$ Torque (kNm)	$y^+ = 20.0$ Percent Difference
4.4	374.27	409.73	-8.65%	352.28	6.24%
6.7	1286.55	1306.71	-1.54%	1184.50	8.62%
9.0	2526.32	2546.72	-0.80%	2287.58	10.44%
11.4	4210.53	4213.61	-0.07%	3799.28	10.82%

number from root to tip. It is not common practice to vary the wall spacing along the span of the blade, and so the wall spacing along the entire span is set by the y^+ value at the blade tip. At rated wind speed, the tip speed of the blade is about 80 m/s, and the wall spacing necessary to ensure $y^+ \leq 1.0$ is approximately 5.0 μm. This distance is an extremely small wall-normal spacing and makes grid deformation and design optimization very difficult. A limit on the wall spacing, obtained through trial and error, which still makes grid deformation and design optimization feasible is about $y^+ = 20.0$ at the blade tip and a wall normal spacing of 0.1 mm. Two grids were generated and simulated at four wind speeds, 4.4 m/s, 6.7 m/s, 9.0 m/s, and 11.4 m/s (rated wind speed). The first grid maintained a $y^+ = 1.0$ at the blade tip, and the second blade was generated with $y^+ = 20.0$ at the tip. CFD simulations were conducted on each grid using a rigid blade in order to compare computed toque values. The steady-state torque values presented in the NREL report were obtained using the BEM model within AeroDyn, an aerodynamics code for wind turbines [45]. The results are compared with torque data at four different wind speeds simulated using CRUNCH CFD® with a rigid blade. For the CRUNCH CFD® formulation and conditions used in these simulations, the reader is referred to §2.2. The results of these simulations are shown in Figure 3.4, and the computed torque values can be found in Table 3.3. It is evident from Table 3.3 that while the torque is under predicted using a $y^+ = 20.0$, the predicted value is consistently about 5-10% lower than the NREL torque value at each wind speed. In some cases a wall function can be used to capture the viscous effects for walls on which it is not possible to generate a suitably small boundary layer mesh. The wall function in CRUNCH CFD® was tested on this case, but torque

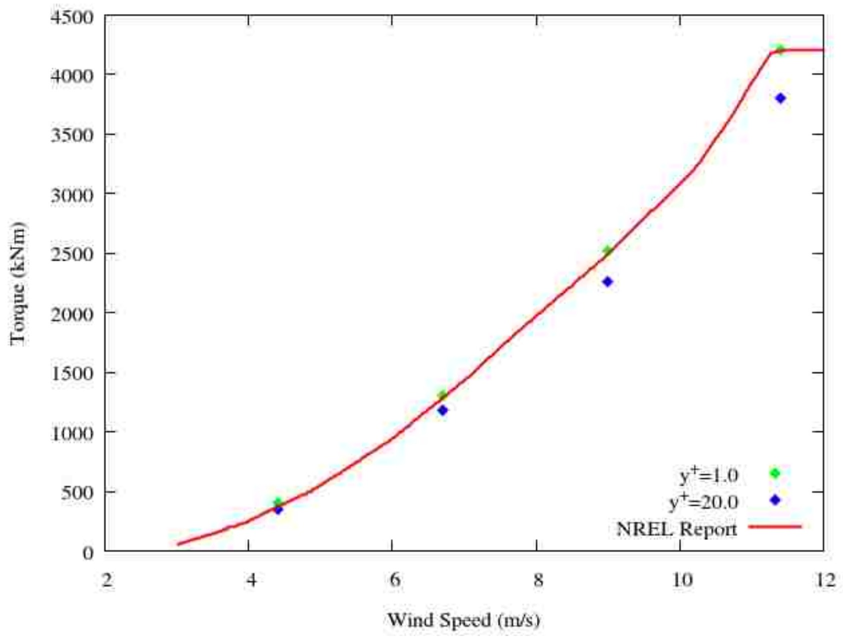


Figure 3.4: Comparison of computed torque values for $y^+ = 1.0$ and $y^+ = 20.0$ against data in the NREL report using AeroDyn [21]

values were over-predicted by about 20%, regardless of the y^+ value. From these results it was decided to use a viscous wall boundary condition without a wall function and a y^+ value of 20.0 at the tip in this design optimization. This method will make the grid deformation and optimization feasible, and is valid since the effect of y^+ on torque is well understood.

3.3 Optimization Results

This section presents the results from each of the optimizations, all of which seek to maximize the useful torque produced by the wind turbine blade. The baseline for each optimization is the NREL 5-MW blade with the composite fibers oriented along the spanwise direction at each point along the span. During the optimization, the undeformed baseline shape of the blade remains unchanged and the composite fiber directions are varied along the span to improve the power capture of the turbine. Section 3.3.1 details the results from the composite fiber optimizations at four wind speeds of 4.4 m/s, 6.7 m/s, 9.0 m/s, and 11.4 m/s. Section 3.3.2 shows the results of the optimization of a single composite fiber

Table 3.4: Rotational speed of wind turbine blades for each of the four wind speeds chosen to conduct optimizations

Wind Speed (m/s)	RPM
4.4	7.31
6.7	8.285
9.0	10.43
11.4	12.1

orientation which maximizes performance over all four wind speeds. Finally, §3.3.3 studies the dependence of the optimal solution on the number of design variables used.

3.3.1 Optimization at Discrete Wind Speeds

Four separate optimizations were conducted at wind speeds of 4.4 m/s, 6.7 m/s, 9.0 m/s, and 11.4 m/s. The corresponding rotational speed of the wind turbine blades for each wind speed is shown in Table 3.4 based on the report by Jonkman *et al.* [21]. Each optimization consisted of four design variables, defined as the composite fiber orientations at four equally spaced locations along the blade span from root to tip. The bounds on the fiber orientation angles, α, were set to $-90° \leq \alpha \leq 90°$. All design variables were initialized to 0° at the first design iteration. An angle of 0° corresponds to an orientation along the blade span. When fibers are oriented along the blade span, flapwise bending stiffness is maximized. The objective function is set to maximize the percent change in torque,

$$\min \quad \frac{T_{\text{baseline}} - T}{T}. \tag{3.2}$$

If the torque in a given design iteration is larger than the baseline torque, (3.2) will yield a negative number, hence the minimization. The optimizer will attempt to find the design with the most negative objective function value, which will represent the largest percent increase in torque. A constraint was placed on the optimization to ensure the tip displacement does not grow large enough to risk a tower strike. The tower clearance of the undeformed blade was estimated to be about 13 m, and it was deemed the tip displacement should not exceed 70% of that distance to safely avoid a tower strike. A similar constraint was enforced by Cox [35]. The mathematical form of the constraint is defined as

$$d_{\text{tip}} \leq 0.7c, \tag{3.3}$$

where d_{tip} is the blade tip displacement and c is the tip-tower clearance.

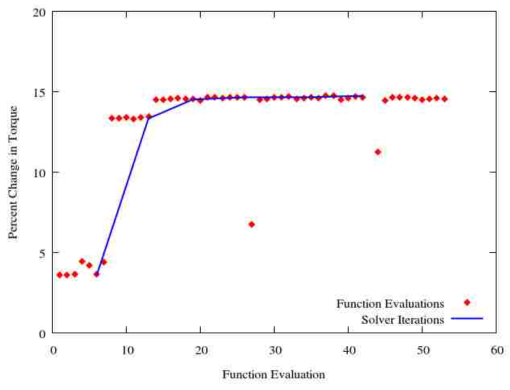
The objective function history for the optimizations at the four wind speeds are shown in the plots in Figure 3.5. The diamonds represent each function evaluation, and the lines represent the objective function history of the optimizer. The SLSQP optimizer is a gradient-based, derivative-free optimizer, meaning the user is not required to specify the first or second derivative functions with respect to the design variables (Jacobian or Hessian). While it is not required to specify the analytical derivative functions of the objective function, the optimizer still needs to know the gradient of the objective function with respect to each design variable at the current evaluation; the calculation of the gradients is performed numerically. Specifically, the optimizer will perturb each design variable in one direction and compute a gradient. Based on this gradient, a line search is performed in a direction defined by the gradients to try to improve the solution. An initial step is made in the direction of the line search. If the objective function is improved, the line search is ended and the gradient is recomputed. If there is no improvement in the objective function, the step size in the direction of the line search is reduced until an improvement is found. Each perturbation of the design variables and the line search are included in the total number of function evaluations. For further details on the mathematics in the SLSQP method, the reader is referred to §2.6. When the objective function history is reported by SLSQP, the function evaluations performed in the gradient computation are neglected, explaining the difference in resolution between the red diamonds and the blue line.

Figure 3.5 shows that each optimization results in an increase in rotor torque, which is proportional to power capture. The optimization at a wind speed of 4.4 m/s results in about a 15% increase in torque. The optimizations at wind speeds of 6.7 m/s and 9.0 m/s each result in a 10% increase in torque, and the optimization at the rated wind speed yields about a 7% increase in torque. Table 3.5 shows a comparison of efficiency

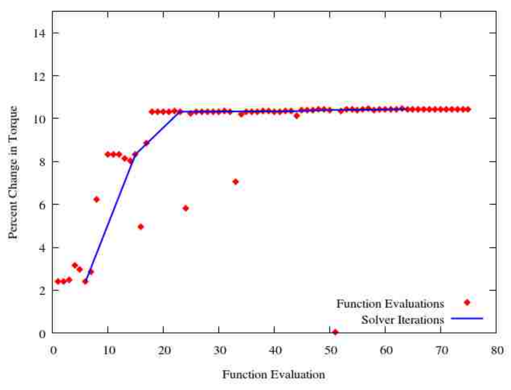
Table 3.5: Comparison of efficiency between baseline and optimal blade designs at each wind speed. Efficiency is defined as power extraction compared with the Betz limit.

Wind Speed (m/s)	Baseline Efficiency	Optimized Efficiency
4.4	70.19%	82.36%
6.7	75.55%	84.36%
9.0	74.69%	83.20%
11.4	72.04%	77.31%

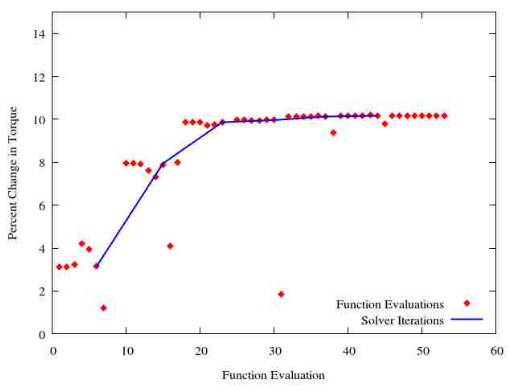
between the baseline and optimal blade designs. All reported efficiencies are compared to the Betz limit which states the ideal power extraction of HAWT is equal to 59.26% of the power available in the wind, shown in Equation (1.1). A derivation of the Betz limit is presented for reference in Appendix A. Generally, the optimizer makes a large improvement on the first two solver iterations followed by much smaller improvements until it finally converges. It is interesting to note that as the wind speed is increased, more function evaluations are required to converge on an optimal solution. The optimization at 4.4 m/s converges to within 2.0% of the final objective function value in only 2 solver iterations, and after 6 solver iterations and 42 objective function evaluations the objective function changes by only 0.01%. The optimization at 6.7 m/s also converges to within 2.0% of the final objective function value in 2 solver iterations, but requires 8 solver iterations and 64 function evaluations to converge the objective function within 0.01%. The optimizations at 9.0 m/s and 11.4 m/s require 4 and 5 solver iterations to converge the objective function to within the 2.0% threshold, respectively. Figure 3.6 shows the twist distributions along the blade span which resulted from the optimizations at each of the four wind speeds. At the lowest wind speed, the optimal solution is simply to increase the twist uniformly across the blade. As the wind speed increases, a uniform increase in blade twist along the span will cause the blade to stall near the tip. As the wind speed is increased even further, a larger percentage of the outboard section of the blade will stall. The twist distributions in Figure 3.6 show that the optimal solutions at the higher wind speeds increase twist near the midspan where local velocities are relatively low, and decrease twist at the tip where local velocities are higher. Figure 3.7 shows the optimal composite fiber orientation angle



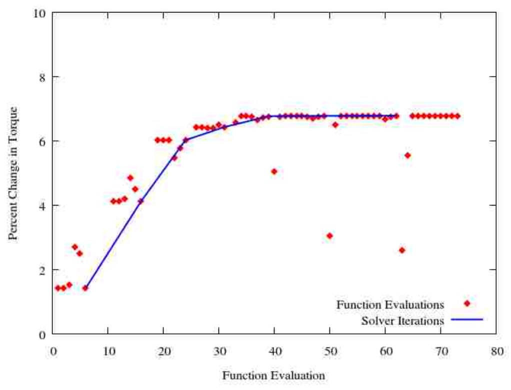
(a) 4.4 m/s



(b) 6.7 m/s



(c) 9.0 m/s



(d) 11.4 m/s

Figure 3.5: Objective function history for optimizations at each wind speed. The points represent all function evaluations in the optimization including the numerical calculation of the objective function gradient with respect to each design variable. The solid line represents the history of each line search performed by the optimizer.

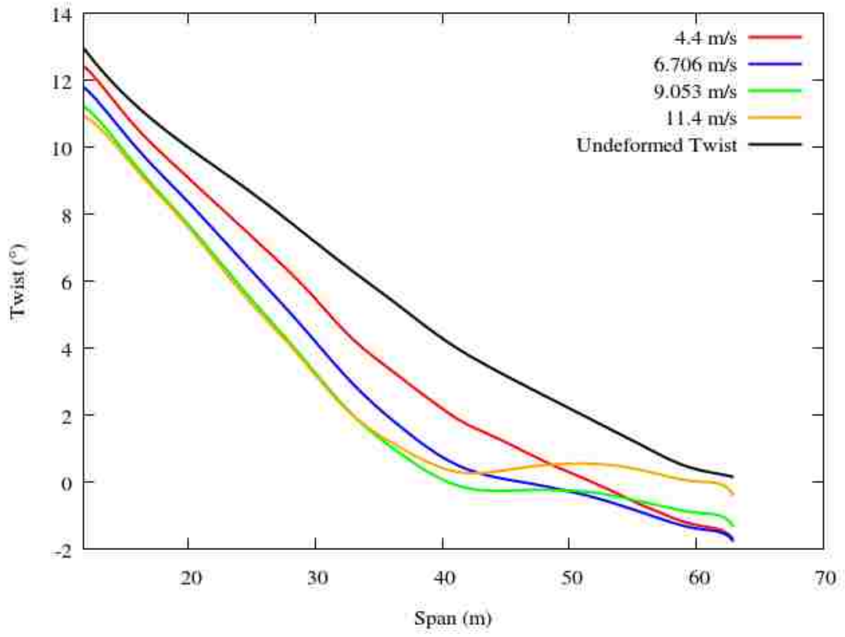


Figure 3.6: Optimal twist distribution for each wind speed

versus span for each wind speed. The optimal fiber orientation along the span at a wind speed of 4.4 m/s is 90° (chordwise) along the entire span to maximize blade twist at each station. For this reason, the solution at 4.4 m/s is not plotted in Figure 3.7. The trend for each wind speed is the same: positive fiber angles on the top of the blade near the root transitioning to negative fiber angles on the top of the blade towards the tip with the transition approximately at the mid-span. The positive fiber angle towards the root causes an elastic bend-twist coupling effect, which increases the blade pitch, and the negative fiber angle at the tip reverses the bend-twist coupling to feather (twist down) the blade. A clear trend is seen near a span location of about 20 m where the positive fiber angle decreases linearly with increasing wind speed. In addition, the slope of the fiber orientation curve between span locations of about 20 m and 45 m also decreases proportionally with the increasing wind speed. There is no discernible trend in the fiber orientation near the tip, and those design variables do not have as large of an effect on the optimal solution. The results in Figure 3.7 underscores the need for variable fiber angles along the span of the blade.

Figures 3.8 through 3.11 show pressure coefficient contour comparisons between the

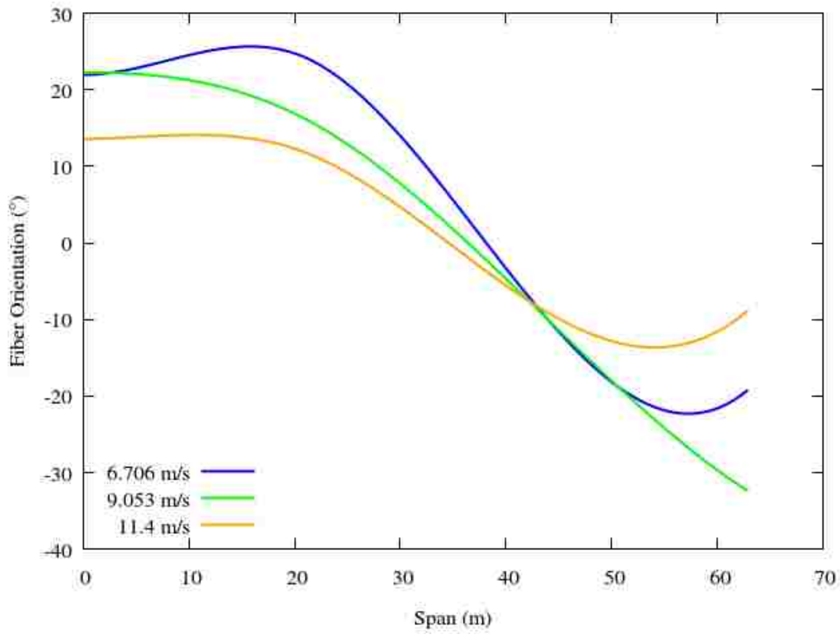


Figure 3.7: Optimal composite fiber orientation angles as a function of span for each wind speed. The optimal composite fiber orientation distribution at 4.4 m/s resulted in a constant fiber angle of 90° and is not plotted here so that the y axis range can be set to $-40 \geq y \geq 30$.

baseline and optimal solutions for each of the four optimizations. Pressure coefficient is defined as

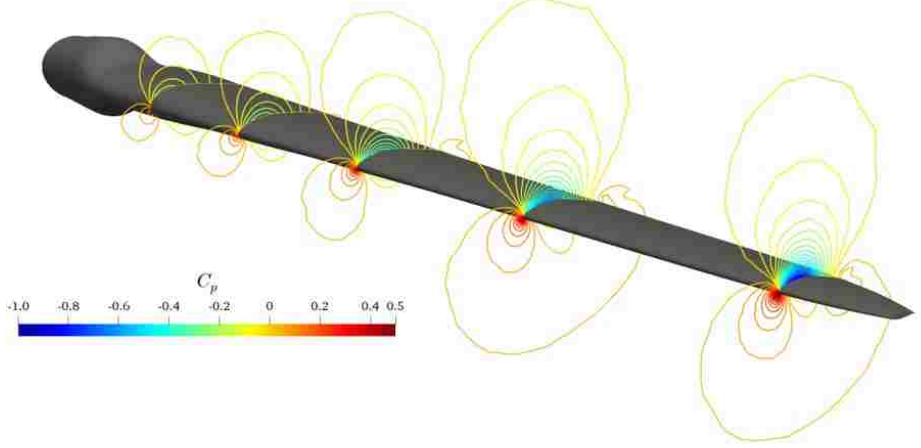
$$C_p = \frac{P - P_\infty}{\frac{1}{2}\rho V_{\text{tip}}^2}, \tag{3.4}$$

where V_{tip} is the blade tip speed. The pressure coefficient contours for each case are created at four locations along the span of the blade. Comparing the plots for each wind speed, it can be seen that the pressure coefficient contour lines for the optimized blade are generally larger than those for the baseline blade. Figure 3.12 through 3.15 shows comparisons of nondimensional thrust force over the span between the baseline and optimized blades for each wind speed. The thrust force is nondimensionalized by the product of the freestream dynamic pressure and the blade planform area. The points in the plot represent the resultant thrust force calculated at each blade station and a curve is fit through those data points. For each simulation, the thrust calculation at the span location of 42 m resulted in a deviation from the trend line. While inconsequential, the cause of this deviation does not seem to be caused by any discrepancy in the blade geometry or CFD simulation, but rather the post-processing technique used to calculate thrust at each span section.

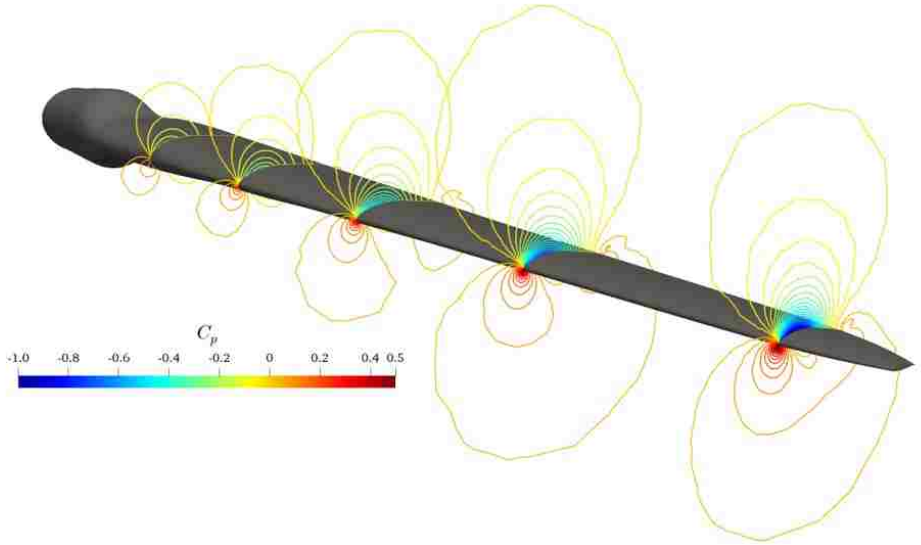
3.3.2 Design of Single Composite Layup for the Range of Wind Speeds

The results from four independent optimizations of the blade at different wind speeds offer valuable insight into the coupling between the blade structural behavior with variable fiber angles along the span and the blade aerodynamics; however, these results do not produce an immediately practical solution. Ideally, it is desirable to find a composite layup that improves the blade performance at all wind speeds and not just at a single wind speed. While a composite layup which improves blade performance across the whole range of wind speeds is a more functional solution, it will likely perform suboptimally when compared with the optimal composite layups at the individual wind speeds.

Two strategies were used to find a composite layup which optimizes performance over the entire range of wind speeds. Strategy #1 was to simply use the optimal composite layup found at the rated wind speed of 11.4 m/s and estimate its performance at each of the other three wind speeds. Strategy #2 was to create a new blade with the optimal total twist distribution from the optimization at 4.4 m/s built into the pre-twist, or twist before the blade is loaded, and conduct an optimization to find a new optimal composite layup which matches the optimal twist distribution at the other wind speeds. The pre-twist distribution of the blade in strategy #2 is shown in Figure 3.16. Starting from the modified blade shown in Figure 3.16, a new optimization was designed to find a single optimal composite layup to most closely match the results from the four optimizations at different wind speeds. In order to optimize a single layup for all four wind speeds, the objective function must take into account the fitness of all four conditions. Using CFD to evaluate each design and compute the torque will be 4 times more expensive than the previous optimization because four separate CFD solutions must be evaluated to convergence. While this does not necessarily render the problem infeasible, a much more computationally efficient approach was adopted. Specifically, the pressure distributions from the optimizations at the four wind speeds were stored and then reused to deform

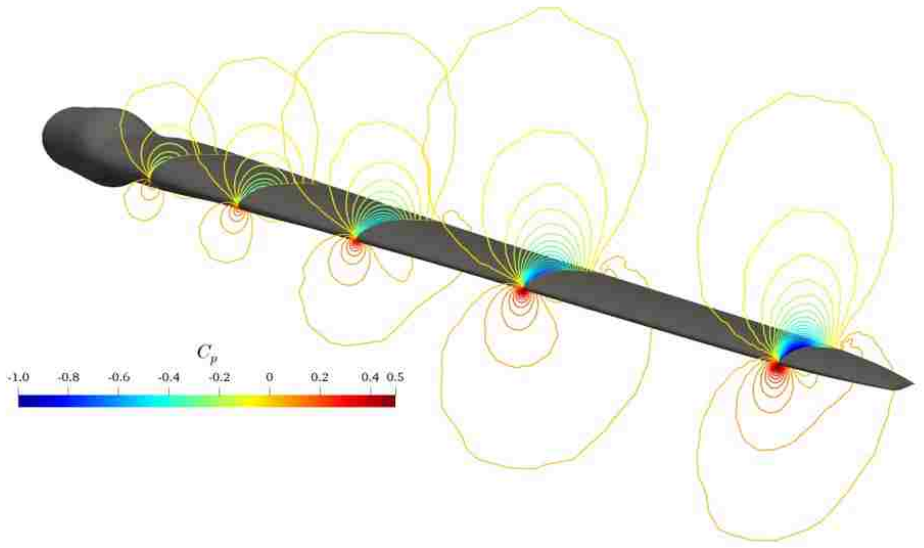


(a) Baseline

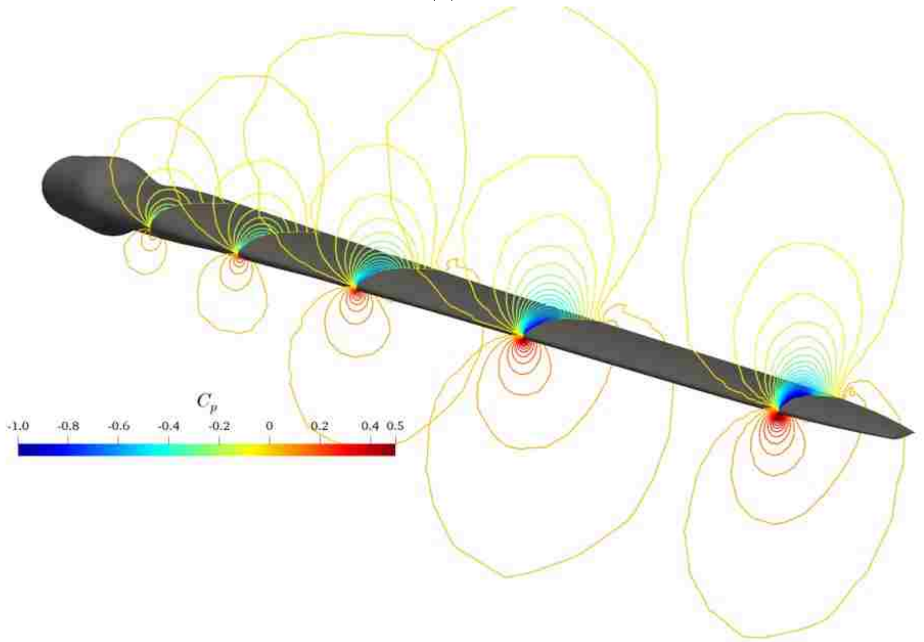


(b) Optimized

Figure 3.8: Contours of pressure coefficient for baseline and optimized blade design at a wind speed of 4.4 m/s

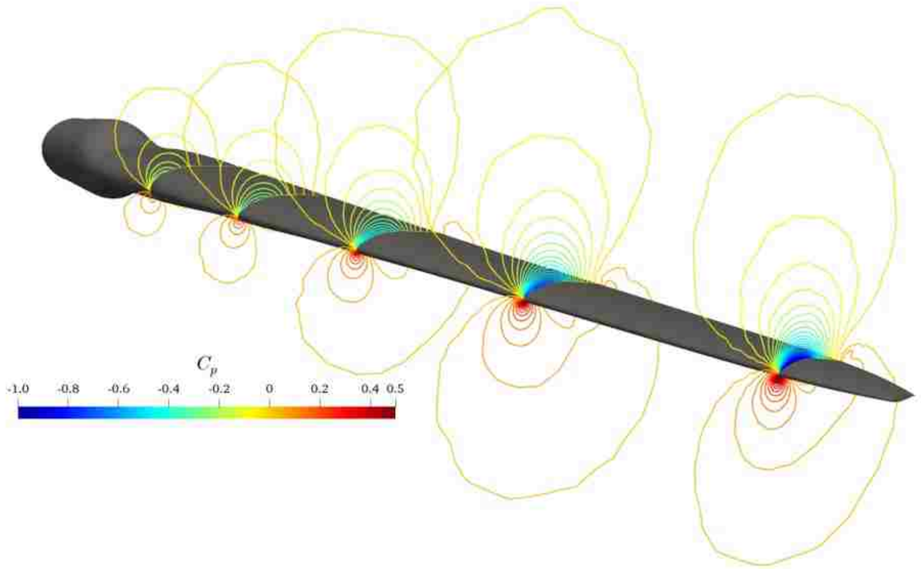


(a) Baseline

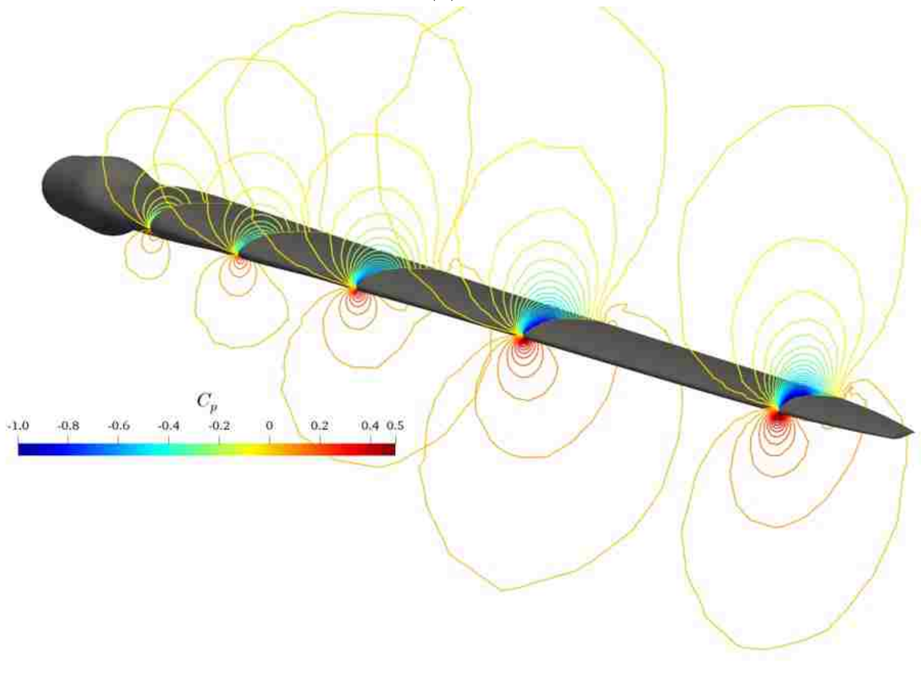


(b) Optimized

Figure 3.9: Contours of pressure coefficient for baseline and optimized blade design at a wind speed of 6.7 m/s

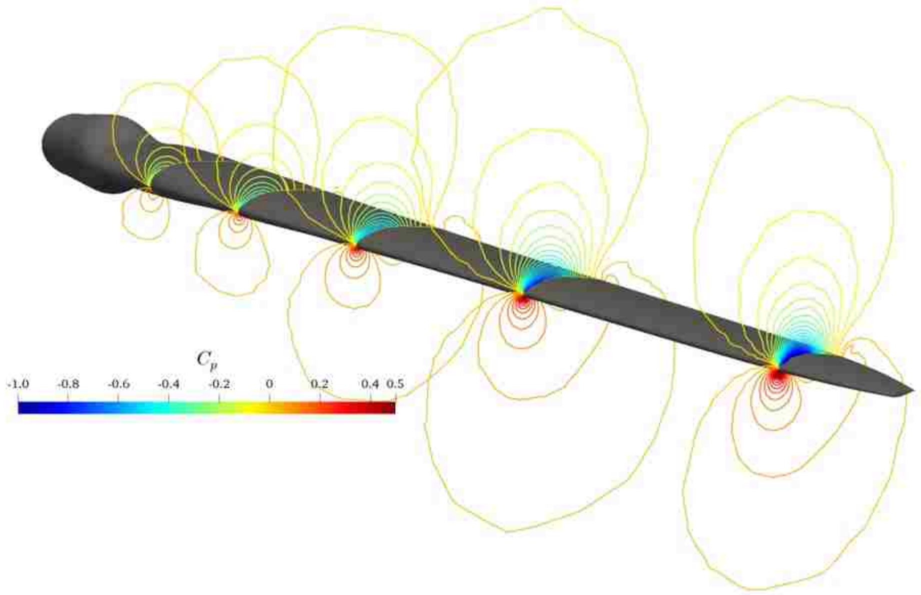


(a) Baseline

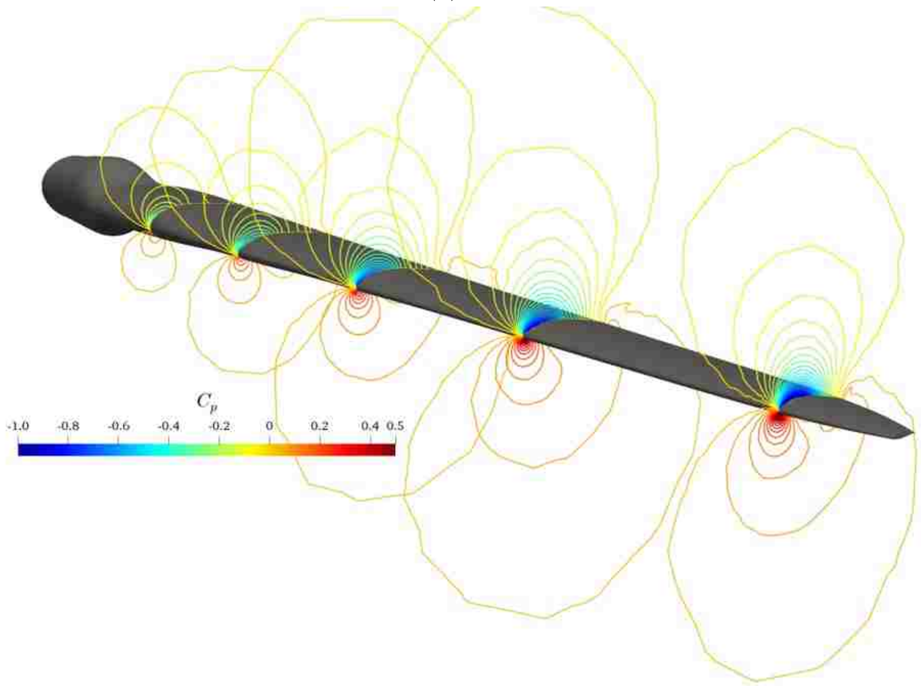


(b) Optimized

Figure 3.10: Contours of pressure coefficient for baseline and optimized blade design at a wind speed of 9.0 m/s



(a) Baseline



(b) Optimized

Figure 3.11: Contours of pressure coefficient for baseline and optimized blade design at a wind speed of 11.4 m/s

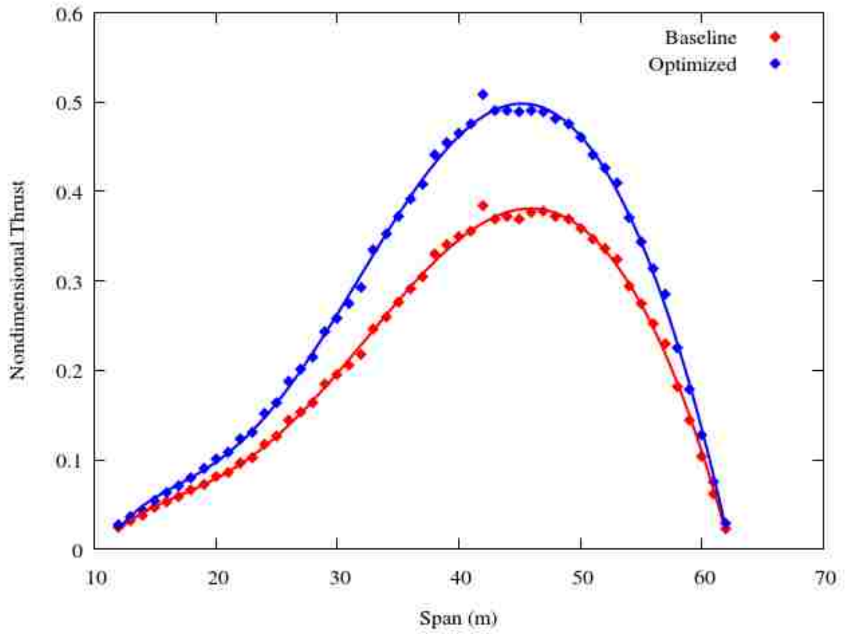


Figure 3.12: Comparison of thrust force over span between baseline and optimized blades at a wind speed of 4.4 m/s

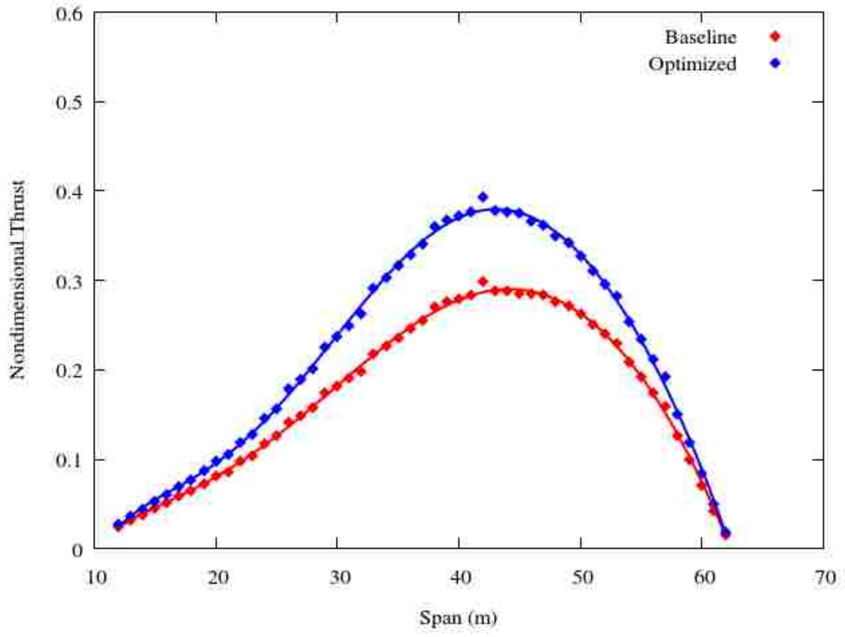


Figure 3.13: Comparison of thrust force over span between baseline and optimized blades at a wind speed of 6.7 m/s

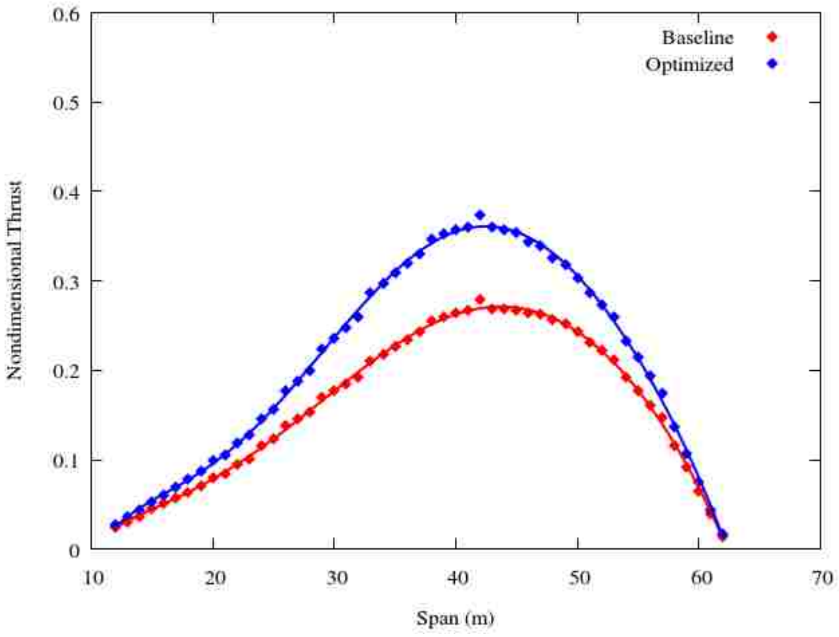


Figure 3.14: Comparison of thrust force over span between baseline and optimized blades at a wind speed of 9.0 m/s

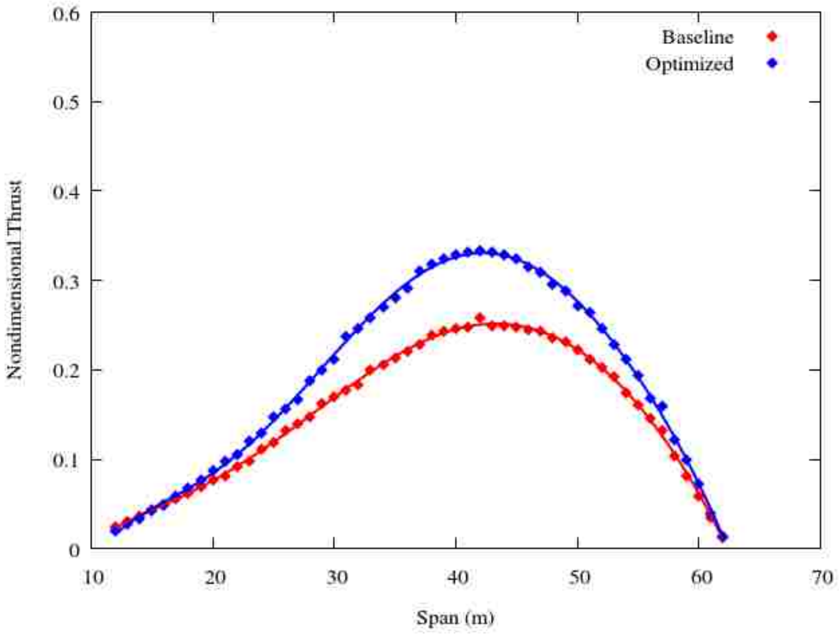


Figure 3.15: Comparison of thrust force over span between baseline and optimized blades at a wind speed of 11.4 m/s

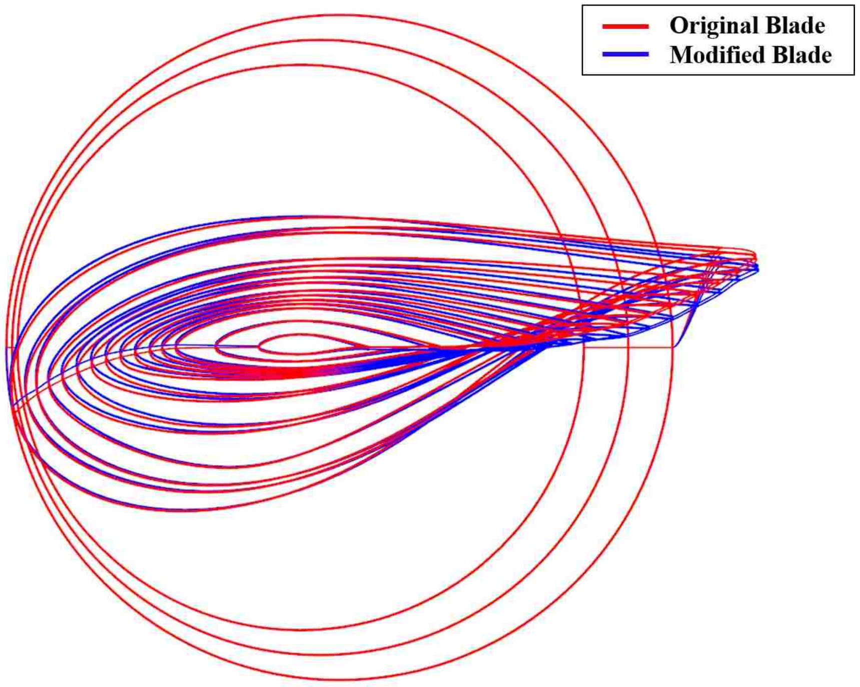


Figure 3.16: Comparison of original blade with blade modified for maximum torque at a wind speed of 4.4 m/s

the modified blade in Abaqus without rerunning the CFD solver. The objective of the new optimization is simply to match the optimal twist distribution for each wind speed as closely as possible. Once an optimal composite layup is found, the result can then be validated using CFD.

The comparison of power increase for both strategies is plotted in Figure 3.17 along with the power increase from each of the optimizations at the four individual wind speeds. The two strategies have competing advantages. The advantage of strategy #1 is that rated power is optimal. The disadvantage is that at lower wind speeds where bending is lower, the bend-twist coupling has a smaller effect and the performance increase is lower. Conversely, strategy #2 will result in a higher performance increase near the cut-in wind speed, but performance increases are lower at rated wind speed. This plot shows that the blade resulting from strategy #1 will perform better in areas with high wind speeds, and the blade found using strategy #2 will perform better in areas of lower average wind speed.

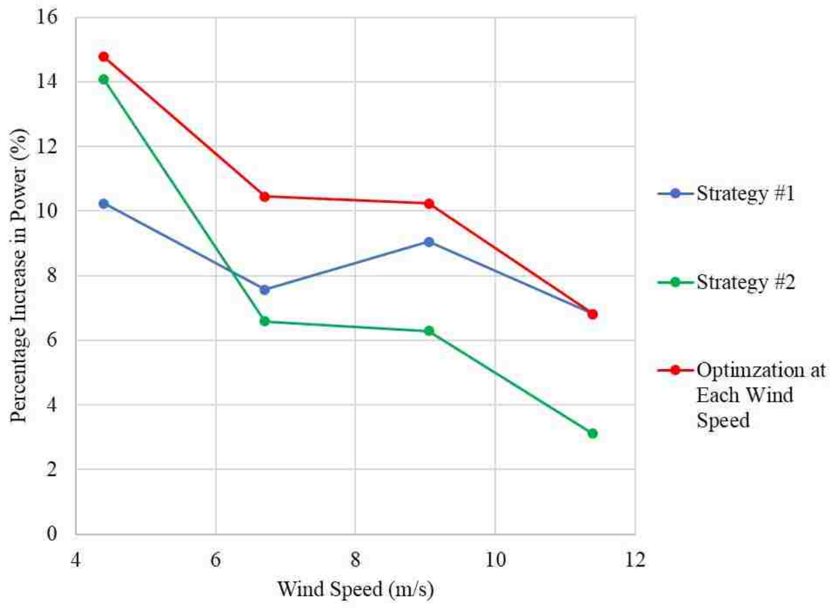


Figure 3.17: Comparison of power increases for Strategy #1 and Strategy #2 with optimal solution from optimizations at each wind speed

3.3.3 Solution Dependency on Optimization Parameters

A study was conducted to understand if there is any dependency of the optimal solution on the number of design variables used in the optimization. Design variables in this context are the number of spanwise locations at which the composite layup is optimized, between which points the fiber orientations are smoothly interpolated. In all previous optimizations, the fiber orientation angle was specified at four locations equally spaced along the span of the blade. The angles at these four points were the design variables in the optimization. A cubic spline interpolation was performed on these points to generate a smooth variation in fiber angle from root to tip.

In the first study, two additional optimizations were conducted at rated wind speed with 5 and 6 design variables, respectively. The design optimization with 5 design variables explicitly specified the fiber orientations at 5 equally spaced locations between the root and tip, and with 6 design variables the fiber orientations were explicitly specified at 6 equally spaced locations between root and tip. The fiber angles were initialized along the blade span direction just as was done in previous optimizations. The resulting optimal fiber

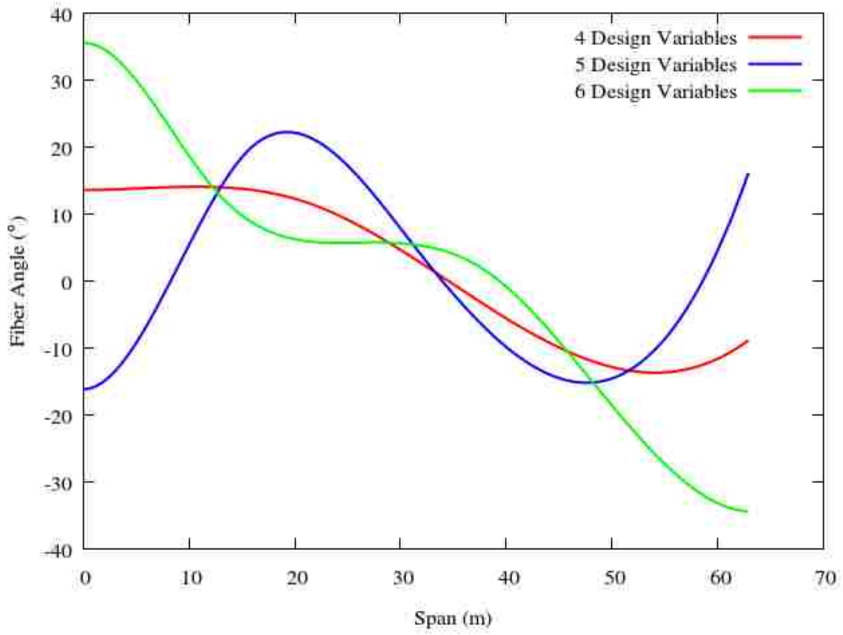


Figure 3.18: Optimal fiber orientation angle versus blade span resulting from optimizations using 4, 5, and 6 design variables. Fiber orientations are initialized from root to tip. All optimizations conducted at rated wind speed.

orientation versus span curve for each optimization is shown plotted against the result from the optimization using four design variables in Figure 3.18. It is clear that the optimal solution is not driving towards a single fiber orientation, and the optimal solutions for each optimization vary greatly with no clear trend. Figure 3.19 shows that while the objective function value is decreased in each optimization, the optimizations with 5 and 6 design variables never reach the objective function value found when using 4 design variables. One possible explanation is that when the number of design variables are increased, the complexity of the cubic spline that defines the fiber angles over the blade span increases, and the optimizer is not as effective at finding an optimal solution with the more complex function. It is similar in effect to using a polynomial function with a much higher order to define the fiber angle over the blade span.

Another study was conducted using 5 and 6 design variables, but instead initializing the design variables to the optimal solution found using 4 design variables. If the solution found using 4 design variables is truly optimal, the solution should not change when the number of design variables is increased. Figure 3.20 shows the objective function history

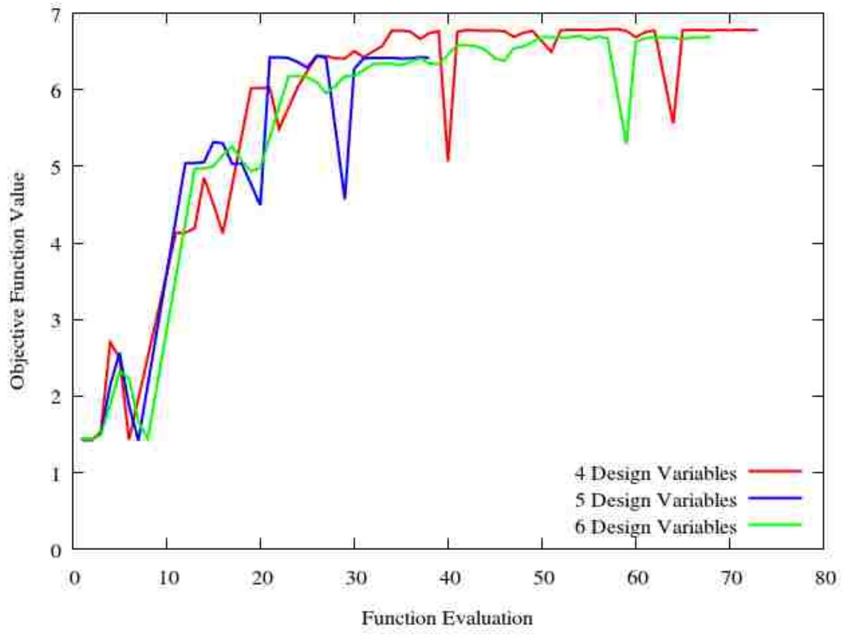


Figure 3.19: Objective function history for optimizations using 4, 5, and 6 design variables. Fiber orientations are initialized from root to tip. All optimizations are conducted at rated wind speed.

for the two optimizations using 5 and 6 design variables. Both optimizations oscillate about the initial condition before exiting, meaning that the optimizer was unsuccessful in finding a better performing design. This study shows that the design is not improved any further by increasing only the number of points along the blade span where the fiber angle is specified.

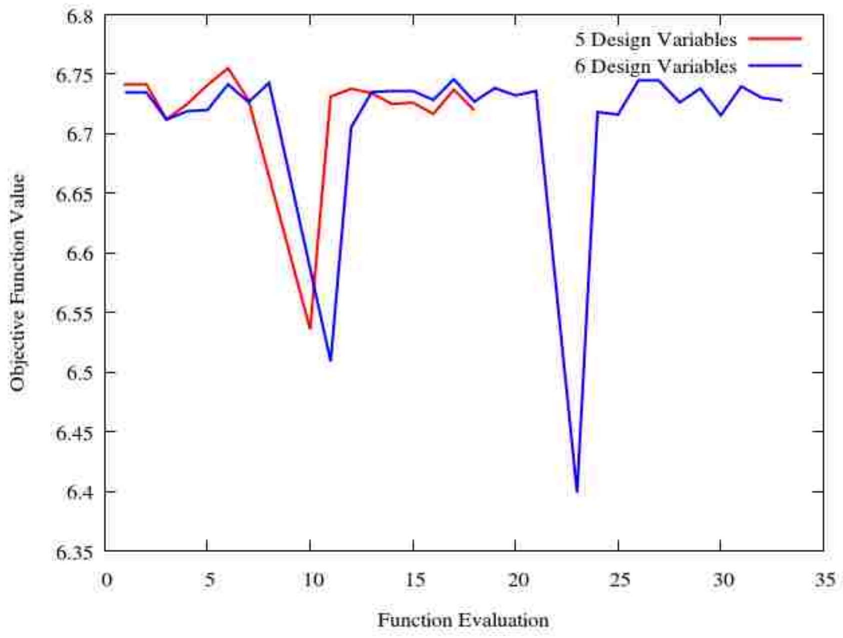


Figure 3.20: Objective function history using 5 and 6 design variables initialized to the optimal solution found using 4 design variables

Chapter 4

Conclusions

This work presents a unique strategy for including a CFD solver in the multi-disciplinary aeroelastic design optimization of wind turbine blades. This aeroelastic optimization is made possible by considering the steady aerodynamics and static structural loading of the blade, and the use of a multi-processor grid deformation tool based on the stiffness method. The following conclusions are made based on the results of this work.

- The use of off-axis composite fibers was found to increase power at all wind speeds between cut-in and rated and offered higher performance gains at lower wind speeds.
- The optimal solution for each wind speed is to use fiber angles directed in positive angles from the spanwise axis inboard and negative angles near the tip. This creates a twist towards stall on the inboard sections of the blade while feathering the blade at the tip avoiding stall.
- When fiber orientations were optimized at individual wind speeds, increases in power were found of 15% at 4.4 m/s, 10% at 6.7 m/s and 9.0 m/s, and 7% at rated wind speed.
- Two different strategies were used to find a single composite fiber orientation that improves blade performance at multiple wind speeds. The first strategy used the optimal composite fiber orientation solved for at rated wind speed. The second strategy

modified the blade pre-twist based on the optimal solution at 4.4 m/s and found a new optimal fiber orientation to match the deformed twist distribution at all other wind speeds. The first strategy increased performance by 10% at the lowest wind speed and 7% at rated wind speed, and the second strategy increased performance by 14% at the lowest wind speed and by 3% at rated.

- When the fiber angles were initialized to run along the spanwise direction, optimizations using different numbers of design variables arrived at different optimal fiber orientations along the span, with the most optimal objective function reached using the fewest design variables (four). When the optimizations using five and six design variables were initialized using the optimal solution solved using four design variables, the solution did not change. From this study, it seems that multiple local extrema exist although their objective function values are fairly close. It also can be concluded that fewer design variables are more likely to find the global minimum. These results may be dependent on the method used to interpolate the fiber angles between the points of specification along the span (cubic spline interpolation), but additional work is necessary to investigate this dependence.

4.1 Future Work

While the blade structural model used in this work was representative of modern wind turbine blades and validated against existing literature, the sophistication of the present computational framework could be improved by using a realistic internal blade structure with well-defined material properties. This adjustment would allow for the inclusion of structural criteria into the optimization of the blade such as peak stresses, material strain limitations, and buckling. In order to qualify the design further, it is important to test the feasibility of the design between rated and cut-off wind speeds, and in response to gust loads.

Bibliography

[1] J. Manwell, J. McGowan, and A. Rogers. *Wind Energy Explained: Theory, Design and Application*. John Wiley and Sons, Inc., 2010.

[2] P. Brondsted and R. Nijssen. *Advances in Wind Turbine Blade Design and Materials*. Woodhead Publishing Limited, 2013.

[3] E. Lantz, M. Hand, and R. Wiser. The past and future cost of wind energy. In *2012 World Renewable Energy Forum*. Paper NREL/CP-6A20-54526, May 2012.

[4] M. Shirk, T. Hertz, and T. Weisshaar. Aeroelastic tailoring - theory, practice, and promise. *Journal of Aircraft*, 23(1):6–18, 1986.

[5] X. Lachenal, S. Daynes, and P. Weaver. Review of morphing concepts and materials for wind turbine blade applications. *Wind Energy*, 16:283–307, 2013.

[6] F. Boria, B. Stanford, W. Bowman, and P. Ifju. Evolutionary optimization of a morphing wing with tunnel hardware-in-the-loop. In *47th AIAA Aerospace Sciences Meeting*. Paper AIAA-2009-1460, January 2009.

[7] D. Berg, S. Johnson, and C. van Dam. Active load control techniques for wind turbines. Technical Report SAND2008-4809, Sandia National Laboratory, 2008.

[8] M. Zuteck. Adaptive blade concept assessment: Curved planform induced twist investigation. Technical Report SAND2002-2996, Sandia National Laboratory, 2002.

[9] T. Ashwill, G. Kanaby, K. Jackson, and M. Zuteck. Development of the swept twist adaptive rotor (STAR) blade. In *48th AIAA Aerospace Sciences Meeting*. Paper AIAA-2010-1582, January 2010.

[10] T. Ashwill, P. Veers, J. Locke, I. Contreras, D. Griffin, and M. Zuteck. Concepts for adaptive wind turbine blades. In *ASME 2002 Wind Energy Symposium*. American Society of Mechanical Engineers, Paper No. WIND 2002-28, 2002.

[11] N. Karaolis, P. Mussgrove, and G. Jeronimidis. Active and passive aerodynamic power control using asymmetric fibre reinforced laminates for wind turbine blades. *Wind Energy Conversion 1988, Proceedings of 10th British Wind Energy Association Conference*, pages 163–172, 1988.

[12] N. Karaolis, G. Jeronimidis, and P. Mussgrove. Composite wind turbine blades: Coupling effects and rotor aerodynamic performance. *EWEC 1989, European Wind Energy Conference*, pages 244–248, 1989.

[13] D. Loibitz, P. Veers, R. Eisler, D. Laino, P. Migliore, and G. Bir. The use of twist-coupled blades to enhance the performance of horizontal axis wind turbines. Technical Report SAND2001-1303, Sandia National Laboratory, 2001.

[14] A. Maheri, S. Noroozi, and J. Vinney. Application of combined analytical/FEA coupled aero-structure simulation in design of wind turbine adaptive blades. *Renewable Energy*, 32(12):2011–2018, 2007.

[15] C. Bottasso, F. Campagnolo, A. Croce, and C. Tibaldi. Optimization-based study of bend-twist coupled rotor blade for passive and integrated passive/active load alleviation. *Wind Energy*, 16:1149–1166, 2013.

[16] M. Capuzzi, A. Pirrera, and P. Weaver. Structural design of a novel aeroelastically tailored wind turbine blade. *Thin-Walled Structures*, 95:7–15, 2015.

[17] M. Capuzzi, A. Pirrera, and P. Weaver. A novel adaptive blade concept for large-scale wind turbines. Part I: Aeroelastic behavior. *Energy*, 73:15–24, 2014.

[18] M. Capuzzi, A. Pirrera, and P. Weaver. A novel adaptive blade concept for large-scale wind turbines. Part II: Structural design and power performance. *Energy*, 73:25–32, 2014.

[19] S. Scott, M. Capuzzi, D. Langston, E. Bossanyi, G. McCann, P. Weaver, and A. Pirrera. Effects of aeroelastic tailoring on performance characteristics of wind turbine systems. *Renewable Energy*, 114:887–903, 2017.

[20] M. McWilliam. *Towards Multidisciplinary Design Optimization Capability of Horizontal Axis Wind Turbines.* PhD thesis, University of Victoria, 2015.

[21] J. Jonkman, S. Butterfield, W. Musial, and G. Scott. Definition of a 5-MW reference wind turbine for offshore system development. Technical Report NREL/TP-500-38060, National Renewable Energy Laboratory, 2009.

[22] Y. Bazilevs, M. Hsu, I. Akkerman, S. Wright, K. Takizawa, B. Henicke, T. Spielman, and T. Tezduyar. 3D simulation of wind turbine rotors at full scale. Part I: Geometry modeling and aerodynamics. *International Journal for Numerical Methods in Fluids*, 65:207–235, 2011.

[23] Y. Bazilevs, M. Hsu, J. Kiendl, R. Wuchner, and K. Bletzinger. 3D simulation of wind turbine rotors at full scale. Part II: Fluid-strucutre interation modeling with composite blades. *International Journal for Numerical Methods in Fluids*, 65:236–253, 2011.

[24] M. Hsu and Y. Bazilevs. Fluid-structure interaction modeling of wind turbines: Simulating the full machine. *Computational Mechanics*, 50:821–833, 2012.

[25] Open CASCADE Technology (OCCT), a software development platform providing services for 3D surface and solid modeling. `https://www.opencascade.com/`. Accessed: 2017-03-01.

[26] L. Piegl and W. Tiller. *The NURBS Book.* Springer, 1 edition, 1995.

[27] S. M. Barr. szb5100/bb3d v1.0.1, March 2018.

[28] V. Ahuja, A. Hosangadi, and S. Aruanajatesan. Simulations of cavitating flows using hybrid unstructured meshes. *Journal of Fluids Engineering*, 123(2):331–340, 2001.

[29] H. Snel. Review of aerodynamics for wind turbines. *Wind Energy*, 6:203–211, 203.

[30] A. Hosangadi, R. Lee, P. Cavallo, N. Sinha, and B. York. Hybrid, viscous, unstructured mesh solver for propulsive applications. In *34th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit.* Paper AIAA-98-3153, 1998.

[31] Release 18, Pointwise user manual, Pointwise Inc., 2017.

[32] J. Steinbrenner and J. Abelanet. Anisotropic tetrahedral meshing based on surface deformation techniques. In *45th AIAA Aerospace Sciences Meeting and Exhibit.* Paper AIAA-2007-554, 2007.

[33] Combustion Research and Flow Technology, Inc. *CRUNCH CFD User Manual and Input Guide*, 2017.

[34] M. Smith. *ABAQUS/Standard User's Manual, Version 6.9.* Simulia, 2009.

[35] K. Cox and A. Echtermeyer. Structural design and analysis of a 10 MW wind turbine blade. *Energy Procedia*, 24:194–201, 2012.

[36] B. Kim, K. Potter, and P. Weaver. Continuous tow steering for manufacturing variable angle tow composites. *Composites: Part A*, 43:1347–1356, 2012.

[37] M. Ahamed, S. Atique, M. Munshi, and T. Koiranen. A concise description of one way and two way coupling methods for fluid-structure interaction problems. *Imperial Journal of Interdisciplinary Research*, 3(3), 2017.

[38] R. Dwight. Robust mesh deformation using the linear elasticity equations. In H. Deconinck and E. Dick, editors, *Computational Fluid Dynamics 2006*, pages 401–406, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[39] C. Felippa. The assembly process. `https://www.colorado.edu/engineering/CAS/courses.d/IFEM.d/IFEM.Ch25.d/IFEM.Ch25.index.html`.

[40] C. Felippa. The linear tetrahedron. `https://www.colorado.edu/engineering/CAS/courses.d/AFEM.d/AFEM.Ch09.d/AFEM.Ch09.index.html`.

[41] M. Hegland and P. Saylor. Block Jacobi preconditioning of the conjugate gradient method on a vector processor. *International Journal of Computational Mathematics*, 44:71–89, 1992.

[42] M. B. Kennel. KDTREE 2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional Euclidean space. *Institute for Nonlinear Science, University of California*, August 2004. `https://arxiv.org/abs/physics/0408067`.

[43] E. Jones, T. Oliphant, and P. Peterson. SciPy: Open source scientific tools for Python. `https://www.scipy.org/`, 2001–. [Online].

[44] G. Nemhauser, A. Rinnooy Kan, and M. Todd. *Handbooks in Operations Research and Management Science: Optimization*, volume 1. North-Holland, 1989.

[45] P. Moriarty, A. Hansen, and P. Moriarty. AeroDyn theory manual.

Appendix A

Betz Limit

Wind turbine efficiency is calculated by comparing the energy extraction of a realistic turbine with an ideal turbine of the same size subject to the same wind speed. One way of quantifying the energy extraction of an idealized wind turbine would be to assume it extracts all the available kinetic energy in the oncoming wind. If a wind turbine were able to extract all of the kinetic energy stored in the wind, the velocity of the wind would be reduced to exactly zero behind the turbine. Since even an ideally designed turbine would still not be capable at reducing the wind velocity to zero, engineers use a different method to determine the performance of an ideal turbine when calculating efficiency. In 1962, Albert Betz devised a mathematical model, which is called the Betz limit, to determine the maximum power extraction allowable for an ideal wind turbine. The derivation in [1] is summarized below.

Consider the graphic in Figure A.1, where the wind turbine is modeled as an actuator disc operating in a streamtube. The model assumes incompressible flow and no frictional losses. The thrust force acting on the actuator disc can be expressed as

$$T = m\frac{du}{dt} = \frac{dm}{dt}\Delta u = \dot{m}(u_1 - u_4) = \rho u_2 A(u_1 - u_4). \tag{A.1}$$

The thrust force can also be expressed as the product of the pressure difference on either

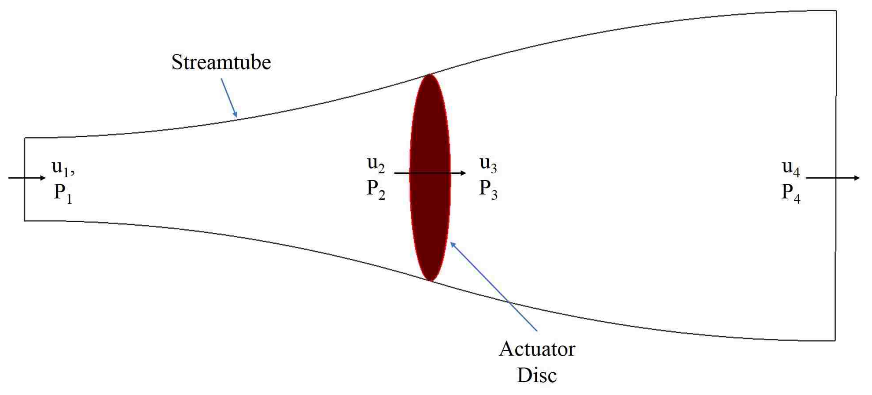


Figure A.1: Illustration of the streamtube entering the wind turbine

side of the disc acting on its area,

$$T = A(P_2 - P_3). \tag{A.2}$$

Using Bernoulli's equation on either side of the disc,

$$\begin{aligned} P_1 + \frac{1}{2}\rho u_1^2 &= P_2 + \frac{1}{2}\rho u_2^2, \\ P_3 + \frac{1}{2}\rho u_3^2 &= P_4 + \frac{1}{2}\rho u_4^2. \end{aligned} \tag{A.3}$$

We can assume $u_2 = u_3$, and if the inflow and outflow of the streamtube are located sufficiently far from the disc, we can further assume $P_1 = P_4$. Solving for $(P_2 - P_3)$,

$$\begin{aligned} P_2 - P_3 &= \frac{1}{2}\rho(u_3^2 - u_2^2) + \frac{1}{2}\rho(u_1^2 - u_4^2), \\ P_2 - P_3 &= \frac{1}{2}\rho(u_1^2 - u_4^2). \end{aligned} \tag{A.4}$$

Substituting (A.4) into (A.2) and setting (A.1) equal to (A.2), we obtain

$$\begin{aligned} \rho u_2 A(u_1 - u_4) &= \frac{1}{2}\rho A(u_1^2 - u_4^2), \\ u_2(u_1 - u_4) &= \frac{1}{2}(u_1 - u_4)(u_1 + u_4), \\ u_2 &= \frac{u_1 + u_4}{2}. \end{aligned} \tag{A.5}$$

Thus, the velocity at the actuator disc is the average of the velocity at the inflow and outflow of the streamtube. If a new variable, a, is introduced called the induced velocity as

$$a = \frac{u_1 - u_2}{u_1}, \tag{A.6}$$

then u_2 and u_4 can be expressed in terms of u_1 and a as follows:

$$\begin{aligned} u_2 &= u_1(1-a), \\ u_4 &= u_1(1-2a). \end{aligned} \tag{A.7}$$

The power extracted by the turbine can be expressed as

$$P = \frac{1}{2}\rho A u_2(u_1^2 - u_4^2) = \frac{1}{2}\rho A u_2(u_1 - u_4)(u_1 + u_4). \tag{A.8}$$

Substituting the expressions in (A.7) into (A.8) yields,

$$P = \frac{1}{2}\rho A(2u_1 a)(2u_1 - 2u_1 a)u_1(1-a). \tag{A.9}$$

Simplifying,

$$P = \frac{1}{2}\rho u_1^3 A(4a - 8a^2 + 4a^3). \tag{A.10}$$

The value of a at which maximum power occurs can be found by differentiating (A.10) and setting the expression equal to zero,

$$\frac{\partial P}{\partial a} = \frac{1}{2}\rho u_1^3 A(4 - 16a + 12a^2) = 0. \tag{A.11}$$

Solving for a, it is found the maximum power occurs when $a = \frac{1}{3}$. Substituting that value back into (A.10),

$$P = \frac{1}{2}\rho u_1^3 A\frac{16}{27}. \tag{A.12}$$

The value of 16/27 equates to approximately 59.259%; therefore, regardless of the aerodynamic efficiency of the blades or the mechanical efficiency of the generator, a wind turbine can only recover a maximum of 59.259% of the power in the wind. This value is referred to as the Betz limit. Wind turbine efficiency is computed by dividing the power extraction of the turbine to the Betz limit.

Appendix B

Converter for Gmsh to Abaqus Input File

The following Python script was used to convert a 2D Gmsh grid file generated in Pointwise to an Abaqus input file. The script reads the Gmsh file which is provided as a command line argument. The output of the code is the main Abaqus input file, *blade.inp*.

```python
import os, sys, math

def rotate(pnt, center ,theta):
  theta_r = theta*math.pi/180.0
  x = (pnt[0]*math.cos(theta_r))-(pnt[1]*math.sin(theta_r))
  y = (pnt[1]*math.cos(theta_r))+(pnt[0]*math.sin(theta_r))
  return [x,y,pnt[2]]
class elem:
  def __init__(self, num, typ, grp, ind, dim):
    self.typ = typ
    self.grp = grp
    self.num = num
    self.dim = dim
    if typ == 11:
      face_def = [[0,1,2],[0,3,1],[1,3,2],[2,3,0]]
      sub_face_def = [[0,1,2,4,5,6],
                      [0,3,1,4,7,8],
                      [1,3,2,5,8,9],
                      [2,3,0,6,7,9]]
      self.faces = [[0 for j in range(3)] for i in range(4)]
      self.sub_faces = [[0 for j in range(6)] for i in range(4)]
      self.indicies = [0 for i in range(10)]
      for i in range(10):
        self.indicies[i] = ind[i]
      for i in range(4):
        for j in range(3):
          self.faces[i][j] = ind[face_def[i][j]]
        for j in range(6):
          self.sub_faces[i][j] = ind[sub_face_def[i][j]]
    elif typ == 17:
      face_def = [[0,1,2,3],[4,7,6,5],[0,4,5,1],[1,5,6,2],[2,6,7,3],[3,7,4,0]]
      sub_face_def = [[0,1,2,3,8,9,10,11],
                      [4,5,6,7,11,13,14,15],
                      [0,1,4,5,8,11,16,17],
                      [1,2,5,6,9,13,17,18],
                      [2,3,6,7,10,14,18,19],
                      [0,3,4,7,11,15,16,19]]
      self.faces = [[0 for j in range(4)] for i in range(6)]
      self.sub_faces = [[0 for j in range(8)] for i in range(6)]
      self.indicies = [0 for i in range(20)]
      for i in range(20):
        self.indicies[i] = ind[i]
      for i in range(6):
        for j in range(4):
          self.faces[i][j] = ind[face_def[i][j]]
        for j in range(8):
          self.sub_faces[i][j] = ind[sub_face_def[i][j]]
    elif typ == 4:
      face_def = [[0,1,2],[0,3,1],[1,3,2],[2,3,0]]
      self.faces = [[0 for j in range(3)] for i in range(4)]
      self.indicies = [0 for i in range(4)]
      for i in range(4):
        self.indicies[i] = ind[i]
      for i in range(4):
        for j in range(3):
          self.faces[i][j] = ind[face_def[i][j]]
    elif typ == 5:
      face_def = [[0,1,2,3],[4,7,6,5],[0,4,5,1],[1,5,6,2],[2,6,7,3],[3,7,4,0]]
      self.faces = [[0 for j in range(4)] for i in range(6)]
      self.indicies = [0 for i in range(8)]
      for i in range(8):
        self.indicies[i] = ind[i]
      for i in range(6):
        for j in range(4):
          self.faces[i][j] = ind[face_def[i][j]]
    elif typ == 7:
      print "Pyramids not supported"
      sys.exit(1)
    elif typ == 1:
      self.indicies = [0 for i in range(2)]
      for i in range(2):
        self.indicies[i] = ind[i]
    elif typ == 8:
      self.indicies = [0 for i in range(3)]
      for i in range(3):
        self.indicies[i] = ind[i]
    elif typ == 2:
      self.indicies = [0 for i in range(3)]
      for i in range(3):
        self.indicies[i] = ind[i]
    elif typ == 3:
      self.indicies = [0 for i in range(4)]
      for i in range(4):
        self.indicies[i] = ind[i]
    elif typ == 9:
      self.indicies = [0 for i in range(6)]
      for i in range(6):
        self.indicies[i] = ind[i]
    elif typ == 16:
      self.indicies = [0 for i in range(8)]
      for i in range(8):
        self.indicies[i] = ind[i]
class group:
  def __init__(self, num, name, typ, dim):
    ## typ --> 1 = Constrained
    ## typ --> 2 = Periodic
    self.num = num
    self.name = name
    self.dim = dim
    self.typ = typ
    self.elems = []
    self.nodes = []
    self.nodes_assoc = []
lines = [line for line in open(sys.argv[1])]
c = len(lines)
for l in reversed(lines):
  if "$PhysicalNames" in l:
    break
  else:
    c = c - 1
num_groups = int(lines[c])
groups = []
c = 0
for n in range(len(lines)-num_groups-1, len(lines)-1):
  num = int(lines[n].split()[1])
  name = lines[n].split()[2].split('"')[1]
  typ = 0
  if name.split("_")[0] == "Contrained":
    typ = 1
  elif name.split("_")[0] == "Periodic":
    typ = 2
  elif name.split("_")[0] == "Unspecified":
    typ = -1
  dimen = int(lines[n].split()[0])
  groups.append(group(num, name, typ, dimen))
  c = c + 1
xlinf = 1.0
nodes = []
num_nodes = int(lines[15].strip())
boundary_nodes = [False for i in range(num_nodes)]
node_set = [False for i in range(num_nodes)]
offset = 16
out = open("plate.inp","w")
out.write("*Heading\n")
out.write("%s\n"%os.getcwd())
out.write("**NUM_NODES %d\n"%num_nodes)
out.write("*NODE\n")
for n in range(0,num_nodes):
  x = float(lines[offset+n].split()[1].strip())*xlinf
  y = float(lines[offset+n].split()[2].strip())*xlinf
  z = float(lines[offset+n].split()[3].strip())*xlinf
  nodes.append([x,y,z])
  out.write("%d, %f, %f, %f\n"%(n+1,x,y,z))
for g in groups:
  g.nodes_assoc = [False for i in range(num_nodes)]
num_elems = int(lines[25+num_nodes])-1
elems_1d = []
elems_2d = []
elems_3d = []
elem_cnt = 0
offset = 26+num_nodes
print num_elems
```

```
for n in range(0,num_elems):
  typ = int(lines[offset+n].split()[1])
  grp = int(lines[offset+n].split()[4])
  if typ == 1 or typ == 8:
    ind = []
    for e in range(5,len(lines[offset+n].split())):
      ind.append(int(lines[offset+n].split()[e]))
    elems_1d.append(elem(0, typ, grp, ind, 1))
  elif typ == 2 or typ == 3 or typ == 9 or typ == 16:
    ind = []
    for e in range(5,len(lines[offset+n].split())):
      ind.append(int(lines[offset+n].split()[e]))
    elems_2d.append(elem(0, typ, grp, ind, 2))
  else:
    ind = []
    for e in range(5,len(lines[offset+n].split())):
      ind.append(int(lines[offset+n].split()[e]))
    elems_3d.append(elem(elem_cnt,typ,grp,ind, 3))
    elem_cnt = elem_cnt + 1
for g in groups:
  if g.dim == 1:
    for e in elems_1d:
      if e.grp == g.num:
        for i in e.indicies:
          if g.nodes_assoc[i-1] == False:
            g.nodes.append(i)
            g.nodes_assoc[i-1] = True
            boundary_nodes[i-1] = True
```

```
hasTie = False
hasBounds = False
alreadyTied = [False for n in range(num_nodes)]
for g in groups:
  if g.name == "ENCASTRE" or g.name == "PINNED":
    hasBounds = True
  elif g.name == "TIE":
    hasTie = True
c = 1
out.write("******* E L E M E N T S *************\n")
for g in groups:
  if g.dim > 1 and not g.name == "Unspecified":
    old_elem = 0
    for e in elems_2d:
      if e.grp == g.num:
        if not e.typ == old_elem:
          if e.typ == 2:
            out.write("*ELEMENT, type=S3R, ELSET=%s\n"%g.name)
          if e.typ == 3:
            out.write("*ELEMENT, type=S4R, ELSET=%s\n"%g.name)
          if e.typ == 9:
            out.write("*ELEMENT, type=S6R, ELSET=%s\n"%g.name)
          if e.typ == 16:
            out.write("*ELEMENT, type=S8R, ELSET=%s\n"%g.name)
        string = "%d, "%c+",".join(map(str, e.indicies))
        out.write(string+"\n")
        c = c + 1
        old_elem = e.typ
```

```python
for g in groups:
  if not g.name == "ENCASTRE" and not g.name == "PINNED" and not g.name == "TIE" and not g.name == "Unspecified" and len(g.nodes) > 0:
    out.write("*NSET, NSET=%s\n"%g.name)
    for n in g.nodes:
      out.write("%d\n"%(n))
if hasTie:
  out.write("*MPC\n")
  for g in groups:
    if g.name == "TIE":
      for e in elems_1d:
        if e.grp == g.num:
          if alreadyTied[e.indicies[0]-1] == False and alreadyTied[e.indicies[1]-1] == False:
            out.write("TIE, %d, %d\n"%(e.indicies[0],e.indicies[1]))
            alreadyTied[e.indicies[0]-1] = True
            alreadyTied[e.indicies[1]-1] = True
out.write('*INCLUDE, INPUT="orientation.inp"\n')
out.write('*SHELL GENERAL SECTION, ELSET=BLADE_CAP, MATERIAL=STEEL\n')
out.write('0.05\n')
out.write('*MATERIAL,NAME=STEEL\n')
out.write('*ELASTIC\n')
out.write('2.1E11, 0.31\n')
out.write('*MATERIAL,NAME=CARBON\n')
out.write('*DENSITY\n')
out.write('1560\n')
out.write('*ELASTIC,TYPE=LAMINA\n')
out.write('135.0E9, 10.0E9, 0.30, 5.0E9, 5.0E9, 5.0E9\n')
out.write('*MATERIAL,NAME=GLASS\n')
out.write('*DENSITY\n')
out.write('1890\n')
out.write('*ELASTIC,TYPE=LAMINA\n')
out.write('41.0E9, 9.0E9, 0.30, 4.1E9, 4.1E9, 4.1E9\n')
out.write('*MATERIAL,NAME=CORE\n')
out.write('*DENSITY\n')
out.write('200\n')
out.write('*ELASTIC,TYPE=LAMINA\n')
out.write('0.25E9, 0.25E9, 0.35, 0.073E9, 0.073E9, 0.073E9\n')
out.write('*MATERIAL,NAME=LINING\n')
out.write('*DENSITY\n')
out.write('1670\n')
out.write('*ELASTIC,TYPE=LAMINA\n')
out.write('9.65E9, 9.65E9, 0.30, 3.86E9, 3.86E9, 3.86E9\n')
out.write('*MATERIAL,NAME=GEL\n')
out.write('*DENSITY\n')
out.write('1230\n')
out.write('*ELASTIC,TYPE=LAMINA\n')
out.write('3.44E9, 3.449, 0.30, 1.38E9, 1.38E9, 1.38E9\n')
out.write('*STEP\n')
out.write('*STATIC\n')
if hasBounds:
  out.write("*BOUNDARY\n")
  for g in groups:
    if g.name == "ENCASTRE":
      for n in g.nodes:
        out.write("%d, ENCASTRE\n"%(n))
out.write('*INCLUDE, INPUT="pload.inp"\n')
out.write('*DLOAD\n')
out.write(' , GRAV, -9.8, 0.0, 1.0, 0.0\n')
out.write("*NODE PRINT\n")
out.write("U,\n")
out.write("*EL PRINT\n")
out.write("MISES,\n")
out.write("*END STEP\n")
out.close()
```

Appendix C

Python Script for Composite Fiber Orientation Specification

The following Python script was used to generate the distribution tables read by Abaqus to specify the composite fiber orientation and thickness for each shell element in the FEA grid. The main Abaqus input file, *blade.inp*, is read by the code and must be present in the directory in which the code is executed. The code generates *dist.inp*, the distribution table for the composite fiber orientation angles and *thick.inp*, the distribution table for the shell thicknesses.

```
import numpy as np
from scipy.interpolate import CubicSpline
import os, sys, math

def is_int(i):
  f = True
  try:
    value = int(i)
  except ValueError:
    f = False
  return f
# Read in Abaqus input file
lines = [line for line in open("blade.inp")]
nlines = 0
nnodes = 0
ntelems = 0
nbelems = 0
tb = 0
readNodes = False
readElems = False
doneNodes = False
for l in lines:
  if readNodes:
    i = l.split(",")[0]
    if not is_int(i):
      readNodes = False
      doneNodes = True
      if l.split(",")[0].strip() == "*NODE":
        if not doneNodes:
          readNodes = True
      elif l.split(",")[0].strip() == "*ELEMENT":
        readElems = True
        if "ELSET" in l.split(",")[2].strip() and "TOP" in l.split(",")[2].
            strip():
          tb = 1
        elif "ELSET" in l.split(",")[2].strip() and "BOTTOM" in l.split(",")
            [2].strip():
```

```
            tb = 2
      else:
        point = l.split(",")[1:3]
        nnodes = nnodes + 1
    elif readElems:
      i = l.split(",")[0]
      if not is_int(i):
        readElems = False
        if l.split(",")[0].strip() == "*NODE":
          if not doneNodes:
            readNodes = True
        elif l.split(",")[0].strip() == "*ELEMENT":
          readElems = True
          if "ELSET" in l.split(",")[2].strip() and "TOP" in l.split(",")[2].strip():
            tb = 1
          elif "ELSET" in l.split(",")[2].strip() and "BOTTOM" in l.split(",")[2].strip():
            tb = 2
      else:
        if tb == 1:
          ntelems = ntelems + 1
        elif tb == 2:
          nbelems = nbelems + 1
    else:
      if l.split(",")[0].strip() == "*NODE":
        if not doneNodes:
          readNodes = True
      elif l.split(",")[0].strip() == "*ELEMENT":
        readElems = True
        if "ELSET" in l.split(",")[2].strip() and "TOP" in l.split(",")[2].strip():
          tb = 1
        elif "ELSET" in l.split(",")[2].strip() and "BOTTOM" in l.split(",")[2].strip():
          tb = 2
  nodes = [[0 for j in range(3)] for i in range(nnodes)]
  top_elems = [[0 for j in range(6)] for i in range(ntelems)]
  bot_elems = [[0 for j in range(6)] for i in range(nbelems)]
  nn = 0
  nte = 0
  nbe = 0
  readNodes = False
  readElems = False
  doneNodes = False
  typ = 0
  for l in lines:
    if readNodes:
      i = l.split(",")[0]
      if is_int(i):
        i = l.split(",")[0]
        point = l.strip().split(",")[1:4]
        nodes[nn][0] = float(point[0])
        nodes[nn][1] = -1.0*float(point[1])
        nodes[nn][2] = float(point[2])
        nn = nn + 1
      else:
        readNodes = False
        doneNodes = True
        if l.split(",")[0].strip() == "*NODE":
          if not doneNodes:
            readNodes = True
        elif l.split(",")[0].strip() == "*ELEMENT":
          etype = l.split(",")[1].strip()
          readElems = True
          if "ELSET" in l.split(",")[2].strip() and "TOP" in l.split(",")[2].strip():
            tb = 1
          elif "ELSET" in l.split(",")[2].strip() and "BOTTOM" in l.split(",")[2].strip():
            tb = 2
          if etype == "type=S4R":
            typ = 4
          elif etype == "type=S3R":
            typ = 3
    elif readElems:
      i = l.split(",")[0]
      if is_int(i):
        if typ == 3:
          te3 = l.strip().split(",")[1:4]
          if tb == 1:
            top_elems[nte][0] = 3
            top_elems[nte][1] = int(l.strip().split(",")[0])
            top_elems[nte][2] = int(te3[0])
            top_elems[nte][3] = int(te3[1])
            top_elems[nte][4] = int(te3[2])
            nte = nte + 1
          elif tb == 2:
            bot_elems[nbe][0] = 3
            bot_elems[nbe][1] = int(l.strip().split(",")[0])
            bot_elems[nbe][2] = int(te3[0])
            bot_elems[nbe][3] = int(te3[1])
            bot_elems[nbe][4] = int(te3[2])
            nbe = nbe + 1
        elif typ == 4:
          te4 = l.strip().split(",")[1:5]
          if tb == 1:
            top_elems[nte][0] = 4
            top_elems[nte][1] = int(l.strip().split(",")[0])
            top_elems[nte][2] = int(te4[0])
            top_elems[nte][3] = int(te4[1])
            top_elems[nte][4] = int(te4[2])
            top_elems[nte][5] = int(te4[3])
            nte = nte + 1
          elif tb == 2:
            bot_elems[nbe][0] = 4
            bot_elems[nbe][1] = int(l.strip().split(",")[0])
            bot_elems[nbe][2] = int(te4[0])
            bot_elems[nbe][3] = int(te4[1])
            bot_elems[nbe][4] = int(te4[2])
            bot_elems[nbe][5] = int(te4[3])
            nbe = nbe + 1
```

```
  else:
    readElems = False
    if l.split(",")[0].strip() == "*NODE":
      if not doneNodes:
        readNodes = True
    elif l.split(",")[0].strip() == "*ELEMENT":
      etype = l.split(",")[1].strip()
      readElems = True
      if "ELSET" in l.split(",")[2].strip() and "TOP" in l.split(",")[2].strip():
        tb = 1
      elif "ELSET" in l.split(",")[2].strip() and "BOTTOM" in l.split(",")[2].strip():
        tb = 2
      if etype == "type=S4R":
        typ = 4
      elif etype == "type=S3R":
        typ = 3
else:
  if l.split(",")[0].strip() == "*NODE":
    if not doneNodes:
      readNodes = True
  elif l.split(",")[0].strip() == "*ELEMENT":
    etype = l.split(",")[1].strip()
    readElems = True
    if "ELSET" in l.split(",")[2].strip() and "TOP" in l.split(",")[2].strip():
      tb = 1
      elif "ELSET" in l.split(",")[2].strip() and "BOTTOM" in l.split(",")[2].strip():
        tb = 2
      if etype == "type=S4R":
        typ = 4
      elif etype == "type=S3R":
        typ = 3
# Get location of each element centroid on the blade
min_x = 0.0
max_x = 0.0
et_cntr = [[0 for j in range(3)] for i in range(ntelems)]
eb_cntr = [[0 for j in range(3)] for i in range(nbelems)]
for i in range(ntelems):
  for j in range(3):
    et_cntr[i][j] = 0.0
    for k in range(top_elems[i][0]):
      et_cntr[i][j] = et_cntr[i][j] + nodes[top_elems[i][k+2]-1][j]
    et_cntr[i][j] = et_cntr[i][j]/top_elems[i][0]
  if et_cntr[i][0] > max_x:
    max_x = et_cntr[i][0]
  if et_cntr[i][0] < min_x:
    min_x = et_cntr[i][0]
for i in range(nbelems):
  for j in range(3):
    eb_cntr[i][j] = 0.0
    for k in range(bot_elems[i][0]):
      eb_cntr[i][j] = eb_cntr[i][j] + nodes[bot_elems[i][k+2]-1][j]
    eb_cntr[i][j] = eb_cntr[i][j]/bot_elems[i][0]
  if eb_cntr[i][0] > max_x:
    max_x = eb_cntr[i][0]
  if eb_cntr[i][0] < min_x:
    min_x = eb_cntr[i][0]
# Perform a cubic spline interpolation based on composite
#   fiber specification
run_num = int(sys.argv[1])
lines = [line for line in open("layup-%d.dat"%run_num)]
b = np.zeros(4)
xi = np.zeros(4)
for i in range(4):
  xi[i] = float(lines[i].split()[0].strip())
  b[i] = float(lines[i].split()[1].strip())
f = CubicSpline(xi,b,bc_type=('clamped','not-a-knot'))
# Write element numbers, fiber orientations, and thicknesses to
#   an Abaqus distribution table based on the element
#   centroid's spanwise location
out = open("dist.inp","w")
out_t = open("thick.inp","w")
out.write("*DISTRIBUTION TABLE, NAME=tableAngle\n")
out.write("ANGLE\n")
out.write("*DISTRIBUTION, NAME=oriAngle1, LOCATION=element, TABLE=tableAngle\n"
    )
out.write("      ,0.0\n")
out_t.write("*DISTRIBUTION TABLE, NAME=tableThick\n")
out_t.write("LENGTH\n")
out_t.write("*DISTRIBUTION, NAME=thick1, LOCATION=element, TABLE=tableThick\n")
out_t.write("      ,0.005\n")
```

```
for i in range(ntelems):
  x = et_cntr[i][0]
  if x > xi[0]:
    out.write("%i,  %f\n"%(top_elems[i][1],f(x)))
  else:
    out.write("%i,  %f\n"%(top_elems[i][1],b[0]))
  if x <= 11.75:
    out_t.write("%i,   %f\n"%(top_elems[i][1],0.015))
  else:
    out_t.write("%i,   %f\n"%(top_elems[i][1],(-0.0002*x)+0.0173))
out.write("*DISTRIBUTION, NAME=oriAngle2, LOCATION=element, TABLE=tableAngle\n")
out.write("      ,0.0\n")
for i in range(nbelems):
  x = eb_cntr[i][0]
  if x > xi[0]:
    out.write("%i,  %f\n"%(bot_elems[i][1],-1.0*f(x)))
  else:
    out.write("%i,  %f\n"%(bot_elems[i][1],-1.0*b[0]))
  if x <= 11.75:
    out_t.write("%i,   %f\n"%(bot_elems[i][1],0.015))
  else:
    out_t.write("%i,   %f\n"%(bot_elems[i][1],(-0.0002*x)+0.0173))
out.close()
```

Appendix D

Python Script for Transferring Pressure Loads Between CFD and FEA Grids

The following Python script was used to transfer the pressure loads computed on the blade surface by CRUNCH CFD® to the FEA grid. The main Abaqus input file, *blade.inp*, is read by the script and must be present in the directory in which the script is executed. The pressure and location of each cell face on the blade surface in the CFD grid must be provided in a comma-separated value file named *press.csv*. An unstructured VTK file named *press.vtk* is generated to visualize the interpolation of pressure onto the FEA grid. The pressures are recorded in *pload.inp* which is included in the main Abaqus input file and read when the Abaqus simulation is executed.

```
from scipy import spatial
import numpy as np
import os, sys, math

class surf:
  def __init__(self, name, typ):
    self.name = name
    self.typ = typ
    self.nelems = 0
    self.elems = []
  def addElem(self, nodes):
    self.elems.append(nodes)
    self.nelems = self.nelems + 1
def isInt(i):
  check = True
  try:
    i = int(i)
  except ValueError:
    check = False
  return check
# Write extracted pressures to an unstructured VTK file for visualization
out = open('press.vtk',"w")
out.write("# vtk DataFile Version 2.0\n")
out.write("untitled, Created by Gmsh\n")
out.write("ASCII\n")
out.write("DATASET UNSTRUCTURED_GRID\n")
# Read node and element data from Abaqus input file
lines = [line for line in open(sys.argv[1])]
num_nodes = int(lines[2].split()[1])
nodes = [[0 for j in range(3)] for i in range(num_nodes)]
out.write("POINTS %d double\n"%num_nodes)
for n in range(num_nodes):
  nodes[n][0] = float(lines[n+4].split(",")[1])
```

```
    nodes[n][1] = float(lines[n+4].split(",")[2])
    nodes[n][2] = float(lines[n+4].split(",")[3])
    out.write("%f %f %f\n"%(nodes[n][0],nodes[n][1],nodes[n][2]))
ce = -1
surfs = []
readElems = False
for l in lines:
    if readElems:
        if l.split(",")[0].strip() == "*ELEMENT":
            name = l.split(",")[2].split("ELSET=")[1].strip()
            typ = l.split(",")[1].split("type=")[1].strip()
            surfs.append(surf(name, typ))
            readElems = True
            ce = ce + 1
        elif not isInt(l.split(",")[0].strip()):
            readElems = False
        else:
            lnodes = []
            if surfs[ce].typ == "S3R":
                lnodes = [int(l.split(",")[1]), int(l.split(",")[2]), int(l.split(",")[3])]
            elif surfs[ce].typ == "S4R":
                lnodes = [int(l.split(",")[1]), int(l.split(",")[2]), int(l.split(",")[3]), int(l.split(",")[4])]
            surfs[ce].addElem(lnodes)
    elif l.split(",")[0].strip() == "*ELEMENT":
        name = l.split(",")[2].split("ELSET=")[1].strip()
        typ = l.split(",")[1].split("type=")[1].strip()
        surfs.append(surf(name, typ))
        readElems = True
        ce = ce + 1
nsurfs = ce + 1
ccp = []
nl = 0
nll = 0
for s in surfs:
    for n in range(s.nelems):
        if s.typ == "S3R":
            nll = nll + 4
        elif s.typ == "S4R":
            nll = nll + 5
        nl = nl + 1
out.write("\n")
out.write("CELLS %d %d\n"%(nl,nll))
for s in surfs:
    for n in range(s.nelems):
        if s.typ == "S3R":
            out.write("3 %d %d %d\n"%(s.elems[n][0]-1,s.elems[n][1]-1,s.elems[n
                ][2]-1))
        elif s.typ == "S4R":
            out.write("4 %d %d %d %d\n"%(s.elems[n][0]-1,s.elems[n][1]-1,s.elems[n
                ][2]-1,s.elems[n][3]-1))
out.write("\n")
out.write("CELL_TYPES %d\n"%nl)
for s in surfs:
    for n in range(s.nelems):
```

```
    if s.typ == "S3R":
      cx = (nodes[s.elems[n][0]-1][0]+nodes[s.elems[n][1]-1][0]+nodes[s.elems[n][2]-1][0])/3.0
      cy = (nodes[s.elems[n][0]-1][1]+nodes[s.elems[n][1]-1][1]+nodes[s.elems[n][2]-1][1])/3.0
      cz = (nodes[s.elems[n][0]-1][2]+nodes[s.elems[n][1]-1][2]+nodes[s.elems[n][2]-1][2])/3.0
      out.write("5\n")
      ccp.append([cx,cy,cz])
    if s.typ == "S4R":
      cx = (nodes[s.elems[n][0]-1][0]+nodes[s.elems[n][1]-1][0]+nodes[s.elems[n][2]-1][0]+nodes[s.elems[n][3]-1][0])/4.0
      cy = (nodes[s.elems[n][0]-1][1]+nodes[s.elems[n][1]-1][1]+nodes[s.elems[n][2]-1][1]+nodes[s.elems[n][3]-1][1])/4.0
      cz = (nodes[s.elems[n][0]-1][2]+nodes[s.elems[n][1]-1][2]+nodes[s.elems[n][2]-1][2]+nodes[s.elems[n][3]-1][2])/4.0
      out.write("9\n")
      ccp.append([cx,cy,cz])
# Read CFD surface pressure data from CSV file
plines = [line for line in open("press.csv")]
ppnt = []
px = []
py = []
pz = []
press = []
c = 0
for p in plines:
  if c > 0:
    # Dimensionalize pressure and subtract the freestream pressure
    press.append((float(p.split(",")[0])*1.225*11.4**2)-79.6005)
    px.append(float(p.split(",")[1]))
    py.append(float(p.split(",")[2]))
    pz.append(float(p.split(",")[3]))
  c = c + 1
# Create KDTree and interpolate surface pressures onto Abaqus FEA grid
out.write("\n")
out.write("CELL_DATA %d\n"%nl)
out.write("SCALARS press float\n")
out.write("LOOKUP_TABLE default\n")
print "Creating KD Tree..."
tree = spatial.KDTree(zip(np.array(px),np.array(py),np.array(pz)))
pout = open("pload.inp","w")
pout.write("*DLOAD\n")
ce = 1
for s in surfs:
  for n in range(s.nelems):
    if s.typ == "S4R":
      q = tree.query(nodes[s.elems[n][0]-1])
      pout.write("%d, P, %e\n"%(ce, -1.0*press[q[1]]))
      out.write("%f\n"%press[q[1]])
    else:
      out.write("0.0\n")
    ce = ce + 1
out.close()
pout.close()
```

Biography

Mr. Barr graduated from The Pennsylvania State University with a bachelor's degree in Aerospace Engineering in 2012. Since graduating, Mr. Barr has been a Research Scientist at Combustion Research and Flow Technology, Inc. in Pipersville, PA. He has experience in areas such as Large Eddy Simulation of aircraft bays, dynamic store release simulations, and design optimization of inlets for air breathing propulsion. Mr. Barr has been a part-time master's student at Lehigh University since 2014.