OPTIMIZATION OF A WIND TURBINE ROTOR WITH VARIABLE AIRFOIL SHAPE VIA A GENETIC ALGORITHM

By

Richard W. Vesel, Jr.

A senior honors thesis submitted in partial fulfillment of the requirements for graduating with distinction

Bachelor's Degree

Aeronautical and Astronautical Engineering

The Ohio State University

2009

Senior Honors Thesis Committee:

Professor Jack J. McNamara, Advisor

Professor Joseph H. Haritonidis

ABSTRACT

Two important criteria for wind turbine performance are the power coefficient, defined as the portion of the wind energy passing through the swept rotor area that is captured by the rotor, and the noise level. This thesis accomplishes the optimization of a wind turbine rotor operating in a uniform wind of 10 m/s with respect to these two competing performance measures via a Genetic Algorithm (GA), taking into account the airfoil shape, defined at three locations throughout the rotor span, chord distribution, twist distribution, RPM, and pitch for a 74.6m rotor. Airfoil shape is defined by Bezier curves with 26 control points: two points fixed at the trailing edge and 24 points with fixed *x*-coordinates and variable *y*-coordinates. Twist and chord are each defined at eight spanwise locations. Optimal RPM and pitch are determined in the course of the wind turbine simulation for each rotor, but are not maintained in the GA as part of the genome of a rotor. The open source design codes XFOIL, FAST, and NAFNoise are utilized for aerodynamic, wind turbine performance, and noise calculations, respectively. The GA proves effective in creating diverse solutions with high efficiency across a range of uniform wind speeds as well as greatly diminished noise. Two optimized rotors produced by the GA are analyzed, showing improvement over the GE 1.5sle, an industry standard wind turbine of approximately the same rotor diameter as that examined in this work. The optimized rotor yields a 7.6% increase in annual power production, equating to about $46,000/year at residential rates.

ACKNOWLEDGEMENTS

I would like to thank my undergraduate research advisor, Professor Jack McNamara, for his encouragement and guidance throughout my career as an undergraduate in Aerospace Engineering at The Ohio State University, and particularly for his mentorship throughout the course of this project. Another persona that has colored my undergraduate experience and contributed greatly to my competency as an engineer is that of Professor Joseph Haritonidis, and I thank him for that and also for his participation on my thesis committee. I would like to thank the College of Engineering for their sponsorship over the last year and a half, as well as the university and the National Merit Scholarship Corporation for collectively providing me an affordable and valuable education. Obligatory and heartfelt thanks go to my parents, Theda Bontecou and Richard Vesel, Sr. for the innumerable ways in which they have contributed to my life and experiences. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

NOMENCLATURE

A	Rotor swept area
A_1, A_2	Coefficients used in stall lift and drag calculations
B	Bezier curve
B_1, B_2	Coefficients used in stall lift and drag calculations
bpts	Number of points used to define the airfoil resulting from the Bezier curve
C	Set of Bezier curve control points
c	Bezier curve control point
C_1, C_2	Coefficients used in stall lift and drag calculations
C_d	Airfoil drag coefficient
C_l	Airfoil lift coefficient
C_P	Power coefficient
dB	Decibels
G	Generation number
K	Number of Bezier curve control points minus 1
M	Maximum number of generations
numpoints	Number of Bezier curve control points
P	Population
p	Population member
P_W	Power available from wind
t	Parametric value from 0 to 1 over which a Bezier curve is defined
v	Wind velocity
α	Angle of attack

κ	Parametric curvature
Γ	Integer value represented by a gene of 8 binary values
Ω	Rotational velocity
τ	Rotor torque

Subscripts

m	Genome corresponding to the mother
f	Genome corresponding to the father
i	Index

CHAPTER I

INTRODUCTION AND OBJECTIVES

1.1 Introduction

In 2008, the United States became the nation with the largest wind power generation infrastructure. As wind turbine power generation proliferates, designs are needed which are both efficient and minimally disruptive to surrounding communities, particularly in terms of additions to background noise [16]. Design optimization is therefore needed to resolve the conflicting considerations of maximum power production and minimum noise generation. Horizontal Axis Wind Turbines (HAWTs) have become the predominant configuration for harnessing wind power, exemplified in the General Electric 1.5sle wind turbine, a model rated at 1.5MW, with over 12,000 units in operation as of March, 2009 [9]. The GE 1.5sle is shown in Fig. 1.1.

The wind turbine rotor is the mechanism which interacts directly with the wind in order to convert it into energy, and is also the main contributor to wind turbine noise. Therefore the present work will focus on optimization of the aerodynamic and aeroacoustic properties of the wind turbine rotor, in a similar optimization study to that undertaken by Xuan [17]. The main improvement over that work is the incorporation of

a variable airfoil shape across the rotor and fixed *y*-coordinates of the Bezier curve control points.

1.2 Optimization Scheme

Multiple objective airfoil design problems require conflicting design objectives, in this case maximum power production and minimum noise generation, to be addressed simultaneously. In such cases, there is no single "optimal" solution for which the optimization scheme might search. Instead, the interaction of different objectives produces a set of compromised solutions, largely known as the Pareto-optimal solutions [6]. The goal of the optimization is to produce several Pareto-optimal solutions which are as broadly varied as possible. Then, higher order analysis may take place in order to determine which of the differing approaches is best suited in terms of economy, manufacture, etc.

Genetic Algorithms (GA) have the advantages over classical optimization methods of more efficiently handling problems with multiple possible optimal solutions and with multiple conflicting design considerations where a systematic search of the entire design space is computationally prohibitive [6]. GA are based upon the "survival of the fittest" concept, and mimic some of the processes by which natural evolution takes place, such as mutation, recombination, and selection [3]. In GA, design parameters are represented by "genes," typically a string of binary digits which may be decoded into a decimal value representing one of the design values. All of the genes defining a given solution taken together are that solution's "genome." The genomes are combined and modified throughout the optimization process.

1.3 Design Problem

The wind turbine rotor is the focus of the present work. The parameters to be optimized are: variable airfoil shape across the entire rotor, chord distribution, twist distribution, rotor pitch, and rotational speed. A simple design point was chosen: a uniform wind of 10m/s. The GE 1.5sle wind turbine was used to determine a baseline for the wind turbine radius, hub height, rotational speed, and chord distribution throughout the rotor.

The rotor is represented by a linear interpolation throughout the rotor span of three different airfoil shapes: a root, a mid-section, and a tip airfoil. The rotor is sliced at 15 spanwise locations, each slice defining a unique airfoil at a particular chord and twist. The chord and twist distributions of the rotor are each defined at eight points in the rotor's span, from which a spline curve is defined and used to arrive at the rotor chord and twist values at the 15 spanwise locations. The airfoil shapes are defined by a single Bezier curve, beginning and ending at the trailing edge.

The aerodynamic performance and noise generation of the rotor as well as the power output of the wind turbine are calculated using the open source design codes, XFOIL, NAFNoise, and FAST. XFOIL is used to evaluate the aerodynamic properties of the 15 airfoils. Results from XFOIL are combined with the method described by Tangler [14] to calculated post-stall aerodynamic behavior. This data is input into FAST which then predicts the wind turbine performance. Finally, NAFNoise is used to determine the noise output for each of the 15 airfoil shapes, which are summed on the dB scale to determine an overall noise level for the wind turbine. A fitness value is assigned to

every rotor based primarily on the power coefficient and noise output. The optimization routine is written in MATLAB®, and the design codes are accessed through batch files which are activated via MATLAB®.

1.4 Literature Review

GA have been successfully applied to aerodynamic, aeroacoustic, and aerostructural optimization problems, both generally, and in the context of HAWTs. Tan compares the effectiveness of an evolutionary algorithm with a swarm algorithm in inverse design, single objective, and multiple objective airfoil shape optimization problems, utilizing the PARSEC method for airfoil representation for Mach numbers up to 0.78 [12]. Cases are studied where drag-to-lift ratio is minimized where the lift is either predefined, set as a minimum, or left unconstrained. Tan finds that both methods are well suited to airfoil shape optimization problems. However, many of the airfoil shapes resulting from the unconstrained lift coefficient problem display the unconventional characteristic of having multiple changes in airfoil curvature as exemplified in Fig. 1.2, which includes the corresponding pressure distribution. However, this potentially reduces the attractiveness of these solutions, and therefore a more conventional airfoil shape will be a constraint in this study.

Alpman also applies a GA toward the airfoil shape optimization problem in incompressible flow, but utilizes the NACA 4-digit convention, where the GA acts on the maximum thickness, maximum camber, and location of maximum camber [1]. Large reductions in D/L are achieved. The resulting airfoils are very thin, however, and the

optimized drag to lift ratio calculated from the panel method combined with a boundary layer correction method seems unrealistically small, on the order of 3e-6.

Burger and Hartfield apply a GA to a wind turbine optimization problem which utilizes a vortex lattice method for predicting wind turbine performance [2]. The NACA 4-digit convention is used to define airfoil shape, which is limited to thin airfoils. The GA thus modified the maximum camber and position of maximum camber. The remaining geometric flexibility of the wind turbine rotor includes blade length, blade width at the root and tip, blade sweep, and blade angle of attack as a function of radial position. That work accurately predicts wind turbine performance under the described constraints and recommends for future work the inclusion of variable blade shapes and differing camber through the rotor span.

Kenway and Martins investigate the aerostructural shape optimization of a wind turbine rotor with respect to site-specific winds [9]. The objective function minimized is the Cost of Energy, or COE, defined as C / AEP, where C is the unit cost including materials, fabrication, transportation, construction, and maintenance, and AEP is annual energy production. The design variables are chord, twist, spar thickness, spar location, spar length, airfoil thickness, and rotation rate. The airfoil is a NACA 44XX series with the thickness determined by the SNOPT optimization scheme, which is a gradient based method, as opposed to a GA, based on the sequential quadratic programming approach. XFOIL is used to calculate the aerodynamic coefficients in the attached flow region, while the method described by Tangler, also used in the present work, is utilized to find the aerodynamic coefficients in the stall regions. Wind turbines with a diameter of 5m are optimized at two different locations, with different average wind velocities. As

a result of the optimization, significant increases in power production at each site are achieved. Smaller, but still significant power increases are seen when the optimized wind turbines are modeled at the opposite wind site.

In a study similar to the present work, Xuan et. al optimize the aerodynamic and aeroacoustic behavior of a 0.5MW and a 1.5MW wind turbine using a GA. The rotor is defined by a single airfoil, represented by a Bezier curve, as in the present work, but in their work both the *x* and *y*-coordinates of the control points are variable. More detail on Bezier curve representation is given in subsequent sections. The aerodynamic coefficients are found by XFOIL, and the sound pressure levels are determined by NAFNoise. For the 1.5MW wind turbine, a small reduction in noise generation was achieved in the optimized version, with negligible differences in power production. The airfoil shape was only slightly different from the original, possibly due to the erratic point distribution of the Bezier curve control points. The general approach of the present work is similar to that of Xuan, however, important improvements include the use of 3 independent airfoils to define the rotor shape, and Bezier control points which are constrained in the *x*-direction, such that only the *y*-coordinates are variable.

1.5 OBJECTIVES

The objective of the present work is to utilize a GA to optimize a rotor representative of that used in a 1.5 MW wind turbine for maximum power output and minimum aerodynamic noise. The rotor parameters to be optimized are the airfoil shape at three locations throughout the rotor span, the chord distribution, twist distribution, pitch, and RPM for a given rotor. The GA should produce several viable

solutions with equivalent performance and different physical characteristics. The optimized rotor shapes will be analyzed for their practicality, and their performance will be compared to the power output of the GE 1.5sle wind turbine.

CHAPTER II

FORMULATION OF THE PROBLEM

2.1 Genetic Algorithm

The genetic algorithm is a simplified imitation of evolution of a population by the "survival of the fittest" mechanism. Reproduction, consisting of *crossover* and *mutation*, produces offspring members on which the selection mechanism operates. These terms will be explained later in the section. The general steps of the genetic algorithm are as follows:

1) Initialize population $P(0) = \{p_1(0), p_2(0), p_3(0) \dots p_N(0)\}$ by applying random variations to an initial rotor
2) Calculate the fitness of every member in the population
3) Set generation number G = 1
4) Select two parent members of the population
5) Crossover and mutate the parent genomes to produce two offspring, and calculate the fitnesses of those offspring
6) If the offspring show improved fitness over their parents, replace the parents in the population with the offspring

7) Repeat 4-6 until desired number of parental unions for one generation have taken place
8) Recalculate the average fitness of the population $P(G+1)$
9) Repeat 4-8 until $G = M$ where *M* is the maximum number of generations desired, or until a desired fitness level is achieved

Crossover is the process by which sections of the genomes of the two parents are switched in order to produce unique offspring. Every gene in the genome consists of eight binary digits, where a gene represents one numerical value used to define the shape of the rotor, for example, the *y*-location of a Bezier curve control point. For each gene, crossover operates. Two endpoints in the gene, of values between 1 and 8, are randomly selected. Next, the section of the gene demarcated by the endpoints is swapped between the mother and father genomes, as in the example below, where the bold digits represents the endpoints, of values 2 and 6.

$$p_{mother} = \quad 1\ |\mathbf{0}\ 0\ 1\ 1\ \mathbf{0}|\ 1\ 1$$

$$p_{father} = \quad 0\ |\mathbf{1}\ 0\ 0\ 1\ \mathbf{0}|\ 0\ 0$$

$$offspring_m = 1\ |\mathbf{1}\ 0\ 0\ 1\ \mathbf{0}|\ 1\ 1$$

$$offspring_f = \ 0\ |\mathbf{0}\ 0\ 1\ 1\ \mathbf{0}|\ 0\ 0$$

This operation is carried out for every gene in the genome. Additionally, at the end of each gene's crossover operation, mutation of each digit in the gene occurs at a low rate, such as 2%, where if a digit in the gene is randomly chosen to be mutated, its value switches. If the digit was 0 it becomes 1, and vice versa.

The eight digit gene is converted into an integer value between 0 and 255. This number is then converted to a decimal value, which is the final value of the control point in the rotor definition. The expression relating the integer value to the control point value depends on what part of the rotor the value represents, as explained in the next section.

2.2 Bezier Curve Airfoil Representation

In this work, the airfoil shape is represented by a closed Bezier curve defined by *numpoints* control points, where in this case *numpoints* = 24. The points are represented by

$$C = \{c_1, c_2, c_3 \dots c_{numpoints}\}$$

and each $c_i = (x_i , y_i)$.

Two control points are added to *C*, which are the starting and ending points, located at the trailing edge, at the origin of the coordinate system in which the Bezier curve is defined. The remaining 24 control points are variable and are acted upon by the genetic algorithm. The Bezier curve *B*(t) is defined for *t* of values [0, 1], where [5]

$$B(t) = \sum_{i=1}^{k} C_i * D_i$$

where

$$D_i = \frac{k!}{i! * (k-i)!} * t^i (1 - t^{k-i})$$

and *k* is one less than the total number of control points defining the Bezier curve.

The values of *t* are taken to be from 0 to 1, separated by the interval 1/*bpts*, where *bpts* is the number of points desired to define the actual airfoil shape. Since

three airfoils are needed to define the rotor, three times the number of variable control points, 24*3 = 72 genes in total are required to represent the airfoil shapes.

The Bezier curve control points themselves are each represented by a single gene as described in the previous section, which defines the y_i value for that control point, from the equation

$$y_i = -0.3 + \Gamma * .00253$$

where Γ is the integer value represented by the gene in question. Therefore the control point has *y* values possible from -0.3 to 0.345. The *x* values of the control points are predetermined in order to group the points somewhat closer toward the leading edge of the airfoil to allow more control over that area, and also to reduce the possibility of a chaotic distribution of Bezier control points. The *x* values of the control points are taken as

$$[0 \quad .1632 \quad .2823 \quad .3809 \quad .4677 \quad .5467 \quad .6202 \quad .6896 \quad .7559 \quad .8197 \quad .8814 \quad .9414 \quad 1.0]$$

The distribution of *x* values for the remaining 13 control points is identical to that shown above, beginning at 1.0 and ending at 0.

The Bezier curve method of airfoil representation used allows great variability in the types of airfoils which may be represented, however, there is a tendency to produce shapes with unacceptable geometry, such as self-intersecting airfoils or upper and/or lower airfoil surfaces with 3 or more changes in concavity resulting in a “squiggly” airfoil. Precautions are taken in the code to immediately rule out any design which results in self intersecting airfoils or airfoils with more than 2 changes in concavity in either the upper or lower surfaces, described in the following sections.

2.3 Rotor Representation

Three physical descriptors of the wind turbine rotor are modified by the genetic algorithm: the cross sectional airfoil shape at the rotor root, mid-section and tip; the twist distribution; and the chord distribution. The twist and chord are defined at 8 spanwise locations, in this case, at

$$[4.4 \quad 8.54 \quad 12.65 \quad 16.77 \quad 20.88 \quad 25.00 \quad 29.11 \quad 33.23 \quad 37.34]m$$

where 37.34*m* is the total radius of the rotor. The chord and twist values for each radial points are represented in the genome by one gene each, each gene consisting of eight binary values. If the integer value from 0 to 255 is represented by Γ, the twist in degrees at each radial location is calculated as follows:

$$twist = -15 + \Gamma * \frac{85}{255}$$

Thus the rotor twist is variable from -15° to 70°, in increments of 85°/255 = 0.333°. The chord at each location is calculated by the following equation:

$$chord = 0.8 + \Gamma * \frac{2.7}{255}$$

Thus the chord is variable from 0.8*m* to 3.5*m*, in increments of 2.7*m*/255 = 0.0106*m*.

The physical rotor is constructed by the following steps:

1) A surfaced is created, defined by the root, mid-section, and tip airfoils, where any intermediate point is the linear interpolation of the airfoil shapes. The rotor is defined at 15 spanwise locations. Fig. 2.1 shows a rotor constructed from three NREL large wind turbine airfoils: S817, S818, S828, at the root, middle, and tip of the rotor, respectively [13].
2) The chord distribution is applied to the rotor. This scales the cross section at each spanwise location proportionally in the *x* and *y* directions. The rotor after this scale transformation is shown in Fig. 2.2.
3) The twist distribution is applied to the rotor. This step rotates the cross section at each spanwise location. The baseline twist distribution was arbitrarily chosen as 17° geometric angle of attack everywhere on the rotor, calculated at 15 RPM and at the design wind speed of 10m/s. The baseline twist distribution was calculated from

$$\boldsymbol{twist} = -\arctan\left(\frac{\boldsymbol{v_{des}}}{\boldsymbol{r\Omega_{des}}}\right) - \boldsymbol{\alpha_{des}}$$

where v_{des} is the design velocity, r is radius, and Ω_{des} is initial design angular velocity, in this case 15 RPM. The rotor, after applying the twist distribution, is shown in Fig. 2.3.

In Figs. 2.1-3, the cross-sectional dimensions are shown to scale, but the span is compressed. An example of a complete rotor blade, shown fully to scale, is given in Fig. 2.4.

2.4 Airfoil Treatment and Aerodynamics Calculations

Since the optimization process produces airfoils with random properties, steps must be taken to ensure that invalid or impractical airfoil geometries do not occur. Two checks are performed, the first for intersection of the upper and lower airfoil surfaces, and the second for more than one change in the direction of curvature in either the upper or lower surface of the airfoil. This prevents a "squiggly" airfoil from appearing. Signed curvature is defined as [5]

$$\kappa = \frac{x_i' y_i'' - y_i' x_i''}{(x_i'^2 + y_i'^2)^{3/2}}$$

where x_i and y_i are the coordinates of a point on the airfoil. The derivatives, then, are

$$x_i' = (x_{i+1} - x_i)$$

$$x_i'' = (x_{i+2} - x_{i+1}) - (x_{i+1} - x_i)$$

with the *y*-derivatives defined similarly. If there is more than one sign change in the curvature over either the upper or lower surfaces of an airfoil, it is treated as invalid.

The lift and drag coefficients for each of the fifteen airfoils created by the interpolation of the three defining airfoils must be calculated. An example of the resulting group of airfoils is shown in Fig. 2.5. The coordinates for each airfoil are saved and loaded by XFOIL, which calculates the zero lift angle of attack, and lift and drag values for angles of attack from -10° to 45°. The viscous flow model in XFOIL is utilized, with the Reynolds number based on the velocity of the radial element at each airfoil's location. Flow transition was forced near the leading edge of the upper and lower surfaces to prevent reliance on laminar flow in the results and also to account for the inevitable dirtying of wind turbine rotors which occurs over time and increases the

likelihood of turbulent flow. The values for lift and drag in the stall region are calculated by the method described in Tangler [14], summarized below.

$$C_{d_{max}} = 1.11 + 0.018AR \qquad (\alpha = 90°)$$

$$C_d = B_1 sin^2\alpha + B_2 cos\alpha \qquad (\alpha = 15° \; to \; 90°)$$

where:

$$B_1 = C_{d_{max}}$$

$$B_2 = \frac{C_{d_{stall}} - C_{d_{max}} sin^2\alpha_{stall}}{cos\ \alpha_{stall}}$$

$$C_L = A_1 sin2\ \alpha + A_2 \frac{cos^2\alpha}{sin\alpha} \qquad (\alpha = 15° \; to \; 90°)$$

where:

$$A_1 = \frac{B_1}{2}$$

$$A_2 = \left(C_{1_{stall}} - C_{d_{max}} sin\alpha_{stall} cos\alpha_{stall}\right) \frac{sin\alpha_{stall}}{cos^2\alpha_{stall}}$$

The lift and drag coefficients calculated by this method for the root airfoil from Fig. 2.5 are shown in Fig. 2.6.

2.5 Fitness Calculation

After the aerodynamic data for each of the 15 airfoils has been calculated as described in the previous section, the overall fitness value of the rotor is calculated by the following steps:

1) FAST is run, using the aerodynamic data for the airfoil sections calculated in the above steps. The rotational speed is optimized by a simple scheme which, typically in 15 or less calls to FAST, determines the optimal rotational

speed for the given rotor within 0.2 RPM. Each call to FAST runs the rotor across a range of pitch angles such that the optimal pitch for each rotational speed is found based on the maximum power coefficient output by FAST for the simulation. The rotor torque is extracted from FAST and converted into a power coefficient based on the wind speed, where available wind power is

$$P_w = \frac{1}{2}\rho A v^3$$

where ρ is air density, *A* is rotor swept area, and *v* is the wind velocity. The power coefficient is defined as

$$C_P = \frac{\tau\Omega}{P_w}$$

where τ is the rotor torque and Ω is the rotor's angular velocity.

2) The aerodynamic noise is calculated by NAFNoise for each of the 15 airfoils. The span input to NAFNoise for each airfoil is the section of the rotor span the airfoil represents. The freestream velocity and angle of attack at which the noise is calculated are the geometric values found from the wind speed, rotational velocity, optimal pitch angle, and rotor twist angle at the given spanwise location. The incorporation of induced velocity into the angle of attack calculation would be an improvement over the present model. The output noise from all airfoils in dB is added from

$$dB_{total} = 10log_{10}\sum_{1}^{n} 10^{dB_i/10}$$

where *n* is the number of sound sources, in this case 15, one for each airfoil.

3) The fitness is calculated from the maximum power coefficient found for the rotor, adjusted by a factor incorporating the rotational rate in RPM if RPM fall outside of the 10 < RPM < 21 range given for the operation of the GE 1.5sle wind turbine upon which the initial design was based. Calculation of the correction factor, F_{RPM}, is described below.

$$\begin{cases} F_{rpm} = 1.2^{(21-RPM)} & RPM > 21 \\ F_{rpm} = 1.9^{(RPM-10)} & RPM < 10 \end{cases}$$

4) The fitness is adjusted by a factor incorporating the total dB level, F_{dB}, as shown below, such that a penalty is incurred for total noise levels above 82 dB, and a reward is given for levels below 82 dB.

$$F_{dB} = .005(82 - dB)$$

5) Finally, the fitness is calculated by *Fitness* = $C_P * F_{rpm} + F_{dB}$.

CHAPTER III

RESULTS

3.1 Performance of the Genetic Algorithm

A 37.34m radius wind turbine was optimized using the described scheme at a design point of a uniform 10 m/s wind. The rotor providing the basis for the population was created by using the NREL S818, S827, and S828 airfoils as the root, middle, and tip airfoils, respectively. The rotor was then scaled according to the initial chord distribution approximated from drawings of the GE 1.5sle rotor, and twisted in order to give 17° geometric angle of attack at 15 RPM and a wind speed of 10 m/s. The resulting rotor is identical to that shown in Fig. 2.4. The genome for this rotor was created and copied 35 times to fill the population. Then, each genome was "vibrated," where the values in the genome for chord and twist at every location, as well as the *y*-coordinate for each Bezier control point, were changed by a small random value. This creates a population with some diversity upon which the GA will act.

The population was evolved over about 450 generations. The algorithm is very computationally expensive, and, under the given parameters, running on a 3.73 GHz Pentium® 4 desktop with 3.25 GB of RAM, each generation can take up to around 30

minutes, where the number of parental unions which occur is equal to half the size of the population. Many of the offspring created, however, have invalid geometry and are immediately discarded, which results in many generations which have very few offspring whose aerodynamic performance must be analyzed. This approach fails to take advantage of the highly parallel nature of the GA; parallelization would greatly reduce the time needed to perform the optimization.

3.2 Optimized Rotor Population

The performance of the GA in terms of the increase in fitness of the population is summarized in Figs. 3.1 and 3.2, which are the average fitness vs. generation and maximum fitness vs. generation, respectively. A relatively steady increase in the average population fitness can be observed, while the maximum fitness increases in steps interspaced by long periods of stagnancy. These trends show that the GA is working to improve the population on a consistent basis; however, many iterations are required to produce a new superior rotor.

In the initial population, after vibration, the rotor with the highest efficiency had a power coefficient of 0.419 and an aerodynamic noise level of 107.2 dB. After the evolution, the maximum power coefficient obtained was 0.477, coupled with a noise level of 90.8 dB. The rotor with the second highest fitness achieved a power coefficient of 0.476 at a noise level of 91.2 dB. The initial power coefficient and noise level versus the optimized values for the entire population is shown in Fig. 3.3. For the best rotor, the increase in maximum power coefficient was 13.8%, accompanied by a noise level decrease of 16.4 dB. This decrease in sound can be thought of as the difference

between the volume produced by a lawn mower at a distance of 3 ft compared to that of a train passing at a distance of 100' [8]. It is important to note that this power is produced at 15.38 RPM, well under the operating limit for the GE 1.5sle of 21 RPM; and also that there is an obvious correlation between RPM and noise level for the optimized rotors, as illustrated in Fig. 3.4. Thus it can be expected that wind turbines extracting wind energy at the lowest possible rotational speed have the potential to produce the least noise. The two best optimized rotors are shown to scale in Figs. 3.5 and 3.6, and the twist distributions are shown in Figs. 3.7 and 3.8.

3.3 Optimized Rotor Properties

The airfoil shapes differ significantly between the best solution, from here on referred to as Solution A, and second best solution, referred to as Solution B. The initial and final airfoil shapes for solutions A and B are shown in Figs. 3.9 and 3.10, respectively. The middle and tip airfoils from Solution A are somewhat reminiscent of a supercritical airfoil, examples of which are shown in Fig. 3.11, where the bottom surface curves sharply toward the trailing edge. This result suggests that those airfoils may have characteristics well-suited to wind turbine applications. The root airfoil of Solution A is little changed from the original, having a slightly rounder leading edge. The middle airfoil is slightly thinner than the initial version and has quite a different shape from the original. The final tip airfoil bears also little resemblance to the original, becoming thicker and shaped more like the middle airfoil.

For Solution B, the root airfoil has a similar shape to the original but with the upper surface substantially lowered. The middle airfoil is quite thick, appears nearly

symmetric, and does not share a likeness with the middle or tip airfoils from Solution A or the supercritical airfoils mentioned. The tip airfoil has a rather unusual shape, and actually does have an extra change in curvature of the upper surface near the trailing edge. Though this appears relatively minor, inaccuracies in the aerodynamics and noise calculations resulting from this feature may be significant. The defect escaped detection due to the change in curvature occurring at the last point prior to the trailing edge in the output airfoil coordinates, for which the curvature was not calculated. This is a coding issue that would need to be resolved for continued work with the optimization scheme.

The chord distribution for Solution A is conventional in that it is generally thicker nearer the root, although the chord does show some fluctuation. Solution B also shows fluctuations, but has the troubling characteristic of being relatively small at the root compared to the rest of the rotor, which has a large chord compared to the original chord distribution. It is unlikely that a design with so much weight out toward the end of the rotor would be practical due to the high bending stresses that would result at the root. A remaining question, then, is whether the chord distributions achieved for Solutions A and B were essential to their performance, and whether such improvements in performance could be achieved with predefined, conventional chord distributions.

The Bezier control points for the airfoils from Solutions A and B are shown in Figs. 3.12 and 3.13, respectively. The control points do not exhibit the erratic behavior of those in Fig. 1.3, partly due to the fact that the *x*-coordinates of the points are predefined.

3.4 Optimized Rotor Performance

The operating point for the optimization was a uniform wind of 10 m/s. The power production over the rest of the operational wind speeds is of interest. Therefore, simulations for the best initial rotor, Solution A, and Solution B were run through a range of wind speeds from 3.5 m/s to 12 m/s. These power curves are compared with the power curve provided by the manufacturer for the GE 1.5sle in Fig. 3.14. The power curves for the initial and optimized rotors are corrected by two factors. The first scales up the results for the modeled rotor for the slightly larger swept area of the GE 1.5sle, and is given by the ratio of swept areas:

$$\left(\frac{R_{GE\ 1.5sle}}{R_{modeled}}\right)^2 = \left(\frac{38.5}{37.34}\right)^2 = 1.063$$

The second is simply a coefficient to represent losses due to generator efficiency, in this case taken to be a constant 0.93 across all wind speeds.

As expected, the optimized rotors show significant and consistent improvement over the initial rotor throughout the range of wind speeds. The two optimized rotors are very close in performance, with Solution B slightly edging out Solution A toward the higher wind speeds. This suggests that rotors with different properties, but optimized for the same wind speed, are likely to show similar relative performance at other wind speeds. It may be possible, however, that a rotor optimized for a different speed, for example 7 m/s, would show further improvement near that wind speed over the solutions optimized at 10 m/s.

The power curve for the GE 1.5sle serves as a good basis for comparison with the results achieved through the optimization. The power curves for the initial and

optimized rotors are similar in shape and value to the 1.5sle. The optimized rotors show a performance superior to the 1.5sle by around 10% for most wind speeds up to about 9 m/s, after which the 1.5sle power curve begins tapering off due to the control system, which was not modeled for the optimized rotors. In retrospect, it would have been more appropriate to choose a wind speed for optimization where the 1.5sle was still producing the maximum power available from the rotor, such as 7 or 8 m/s. Also, in comparing the optimized rotors to the 1.5sle's performance, one must acknowledge that there are likely to be losses in addition to the generator efficiency before GE arrives at their final power output value, and also that the accuracy of the results are only as good as the aerodynamic calculations and BEM-based model used to predict the wind turbine power output.

To further explore the performance gain of the optimized rotors versus the GE 1.5sle, a modified power curve for Solution B was produced, which tapers and caps power production in a similar manner to the 1.5sle. The resulting power curve is shown in Fig. 3.15. The annual wind speed data for White Deer, Texas at an altitude of 50m [18] was used for comparison, shown in Fig. 3.16. To avoid the overprediction of performance near cut-in speed, wind speeds from 4.5 m/s to 20.5 m/s were considered. The resulting total energy output for Solution B is 6.9369 GW-hours, compared to 6.4708 GW-hours for the GE 1.5sle, an improvement of 7.6%, equating to $46,610 worth of electricity at residential rates. This economic benefit demonstrates the potential value of this type of optimization.

CHAPTER IV

CONCLUDING REMARKS

4.1 Conclusions

This optimization study was conducted with the aim of producing at least two distinct optimized wind turbine rotors for a 1.5 MW wind turbine, where the shape of the rotor was given as much freedom as possible. It is hoped that through such a method, new designs and ideas, unconstrained by conventionality, might materialize for wind turbine applications. In the course of the study, the following conclusions have been reached:

1) A Genetic Algorithm utilizing a partially constrained Bezier curve model that still allowed for large variability in airfoil shape was successful in producing unconventional rotor configurations and airfoil designs which still resulted in excellent predicted performance.
2) The open source design code XFOIL, combined with another open source design code, FAST, yield results for the performance of an optimized wind turbine rotor that correlate well to the performance of an industry standard model, the GE 1.5sle.
3) The efficacy of Genetic Algorithms in wind turbine optimization has the potential for vast improvement by parallelization, where multiple offspring can

be evaluated simultaneously in a given generation, greatly reducing the computation time required.

4.2 Recommendations for Future Research

Several recommendations for future work on this project have been made which are summarized as follows:

1) Optimization studies should be performed for rotors with a constrained chord distribution, or a simplified set of chord parameters, ensuring that impractical chord distributions do not result.
2) Rotors should be optimized at a lower wind speed operating point, such as 7 m/s, or under more realistic wind conditions which vary in time and space.
3) The GA stands to greatly benefit from parallel processing.
4) Higher accuracy models for both airfoil aerodynamics and wind turbine performance would add validity and usefulness to the findings.
5) Experimental testing should be carried out for the most attractive rotor configurations.
6) The use of “surrogate” aerodynamic performance functions should be investigated to expedite the GA.

FIGURES

Figure 1.1: GE 1.5sle horizontal axis wind turbine, diameter 77m [9].

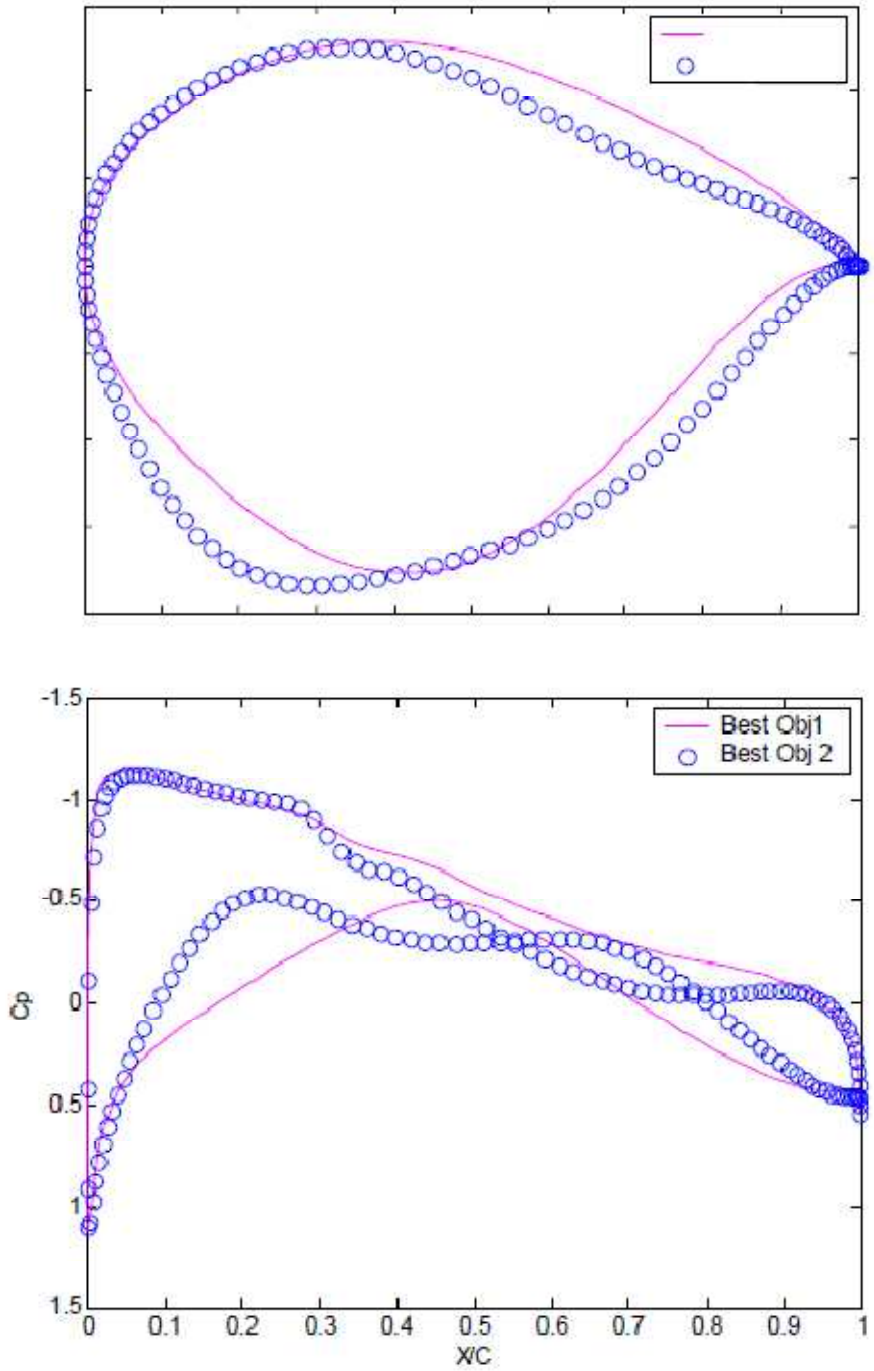

Figure 1.2: Optimized airfoil shape and corresponding C_p distribution for C_D / C_L^2 minimization problem with no constraints on lift coefficient [12].

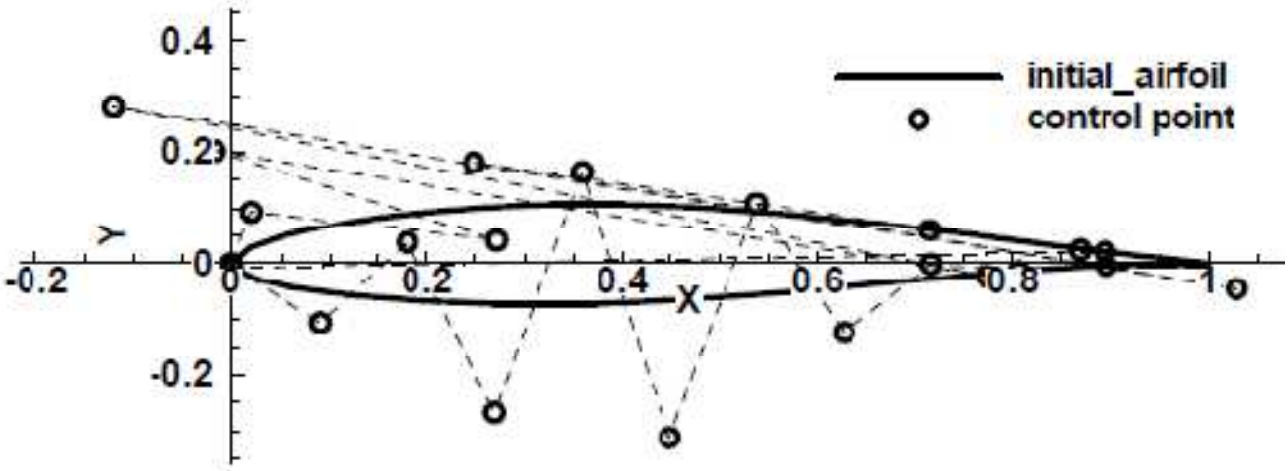

Figure 1.3: Bezier control points and resulting airfoil shape [17]. In the author's experience, such erratic point distributions result in the suitability of the airfoil shape becoming highly sensitive to changes to a single control point, and are to be avoided.

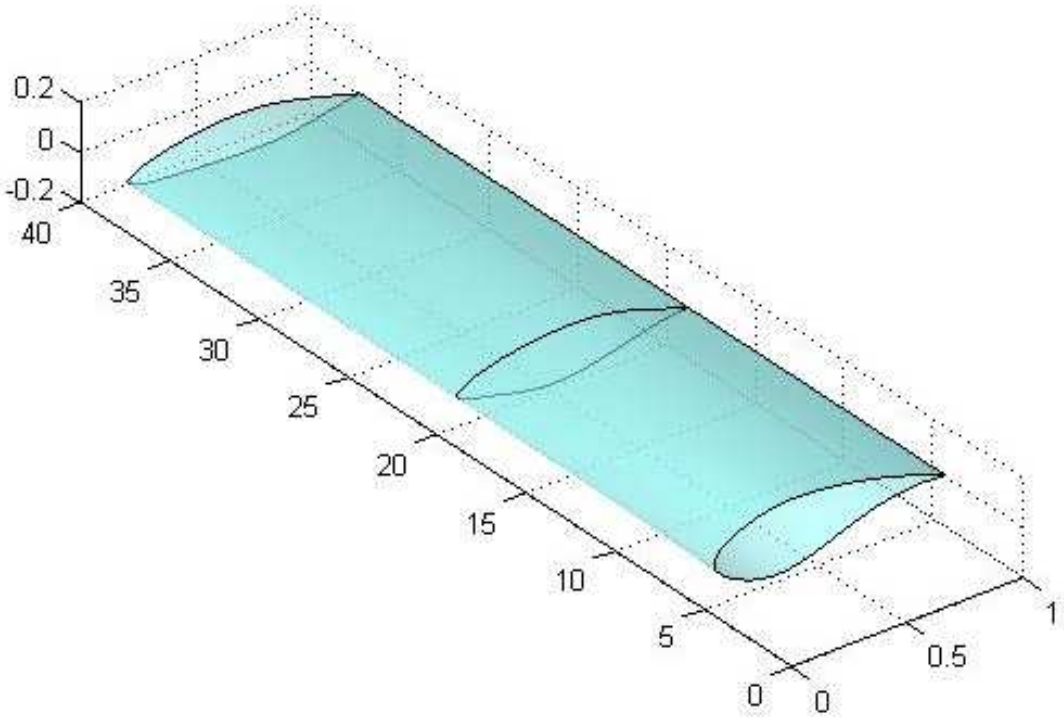

Figure 2.1: Example rotor as defined by three airfoil shapes, outlined in black.

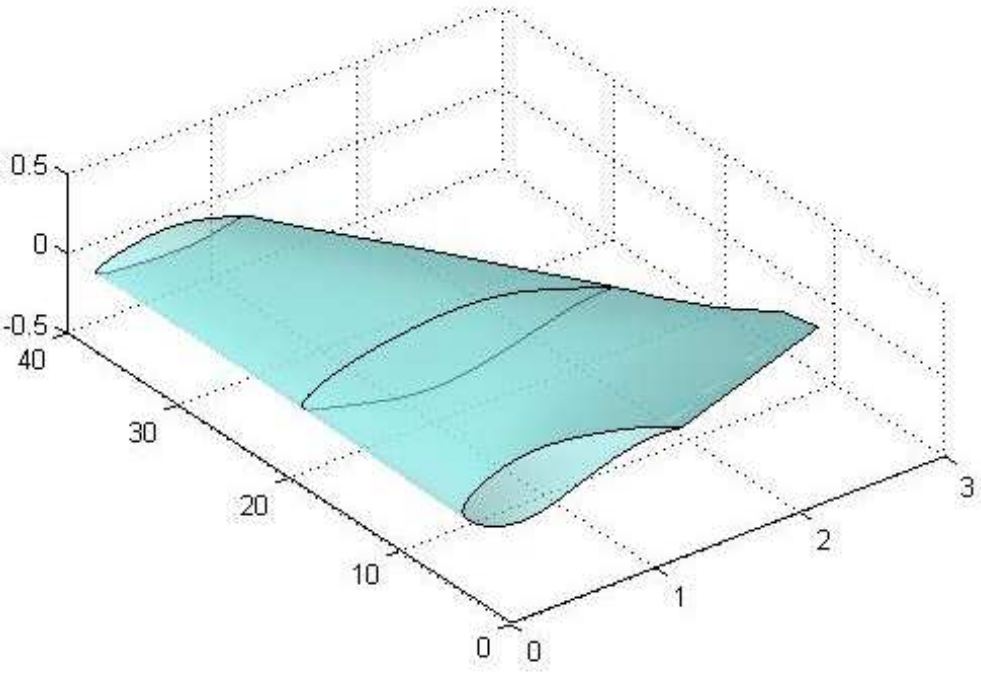

Figure 3.2: Example rotor after distributed scale transformation.

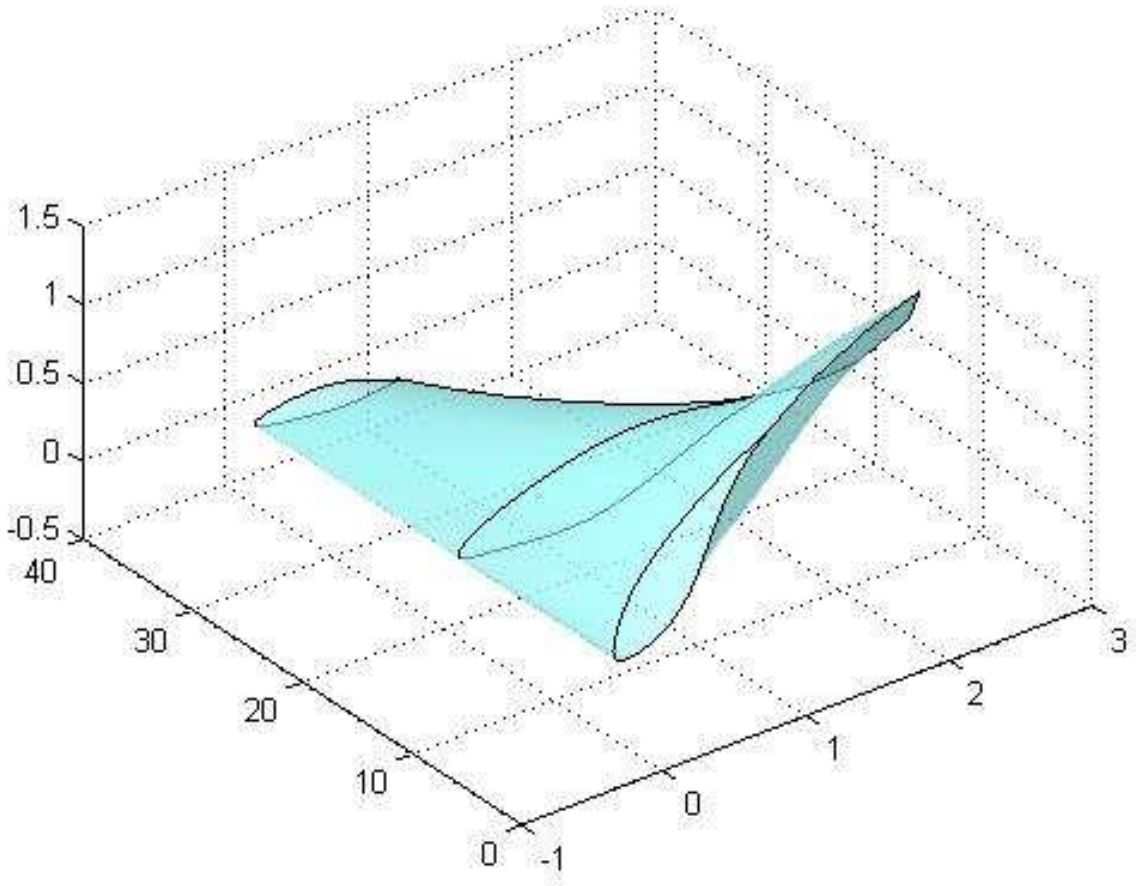

Figure 2.4: Example scaled and twisted rotor.

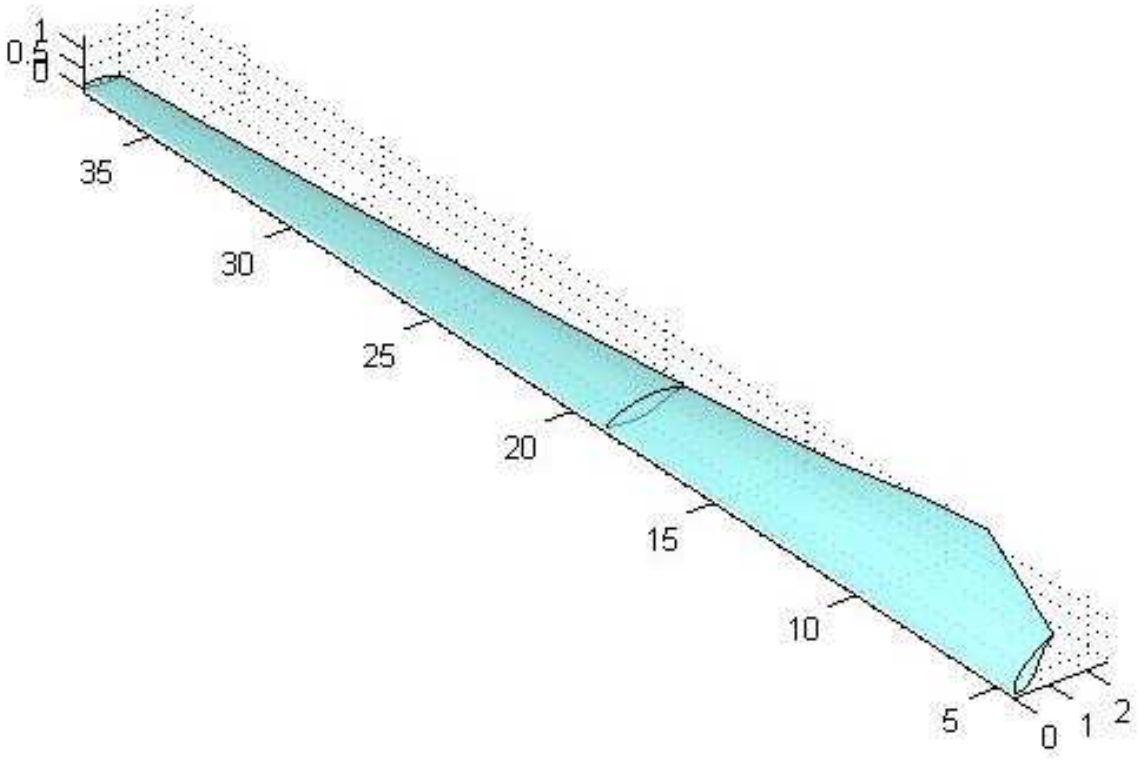

Figure 2.4: Example rotor shown to scale.

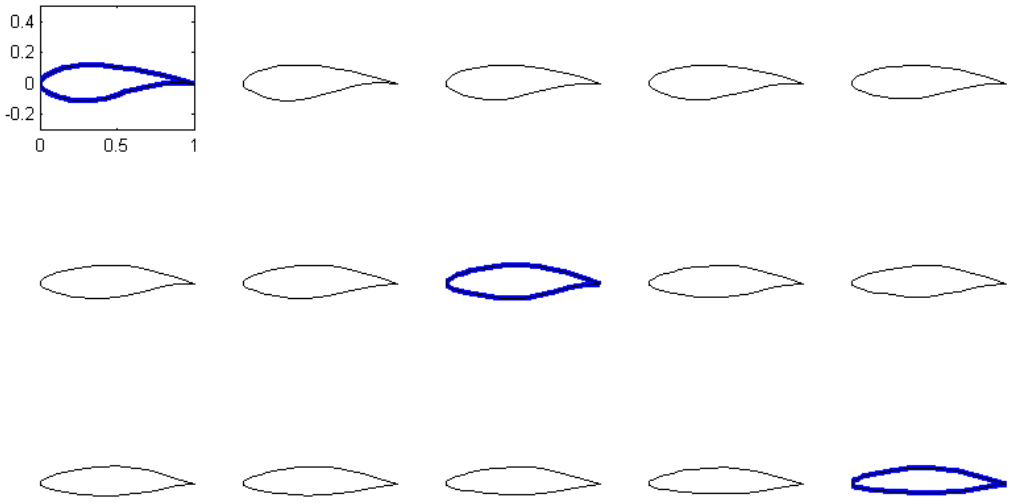

Figure 2.5: Example of the 15 airfoils, created by interpolating the root, middle, and tip airfoils, in this case NREL S818, S827, and S828, from upper left to lower right.

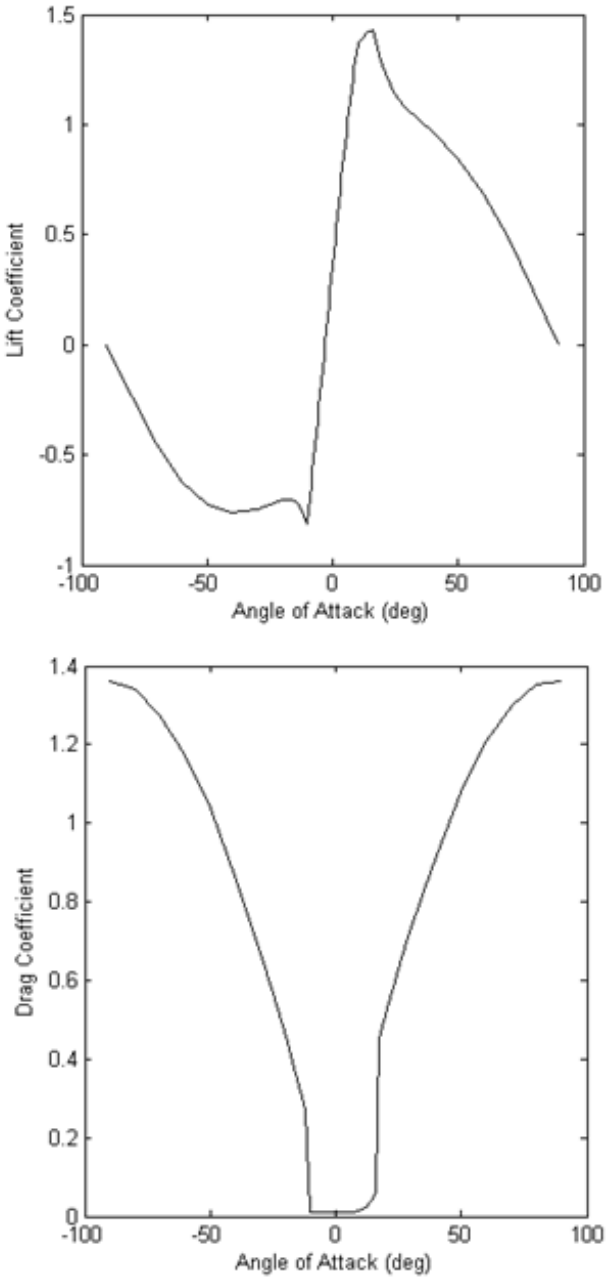

Figure 2.6: Calculated lift and drag coefficients for NREL S818 airfoil.

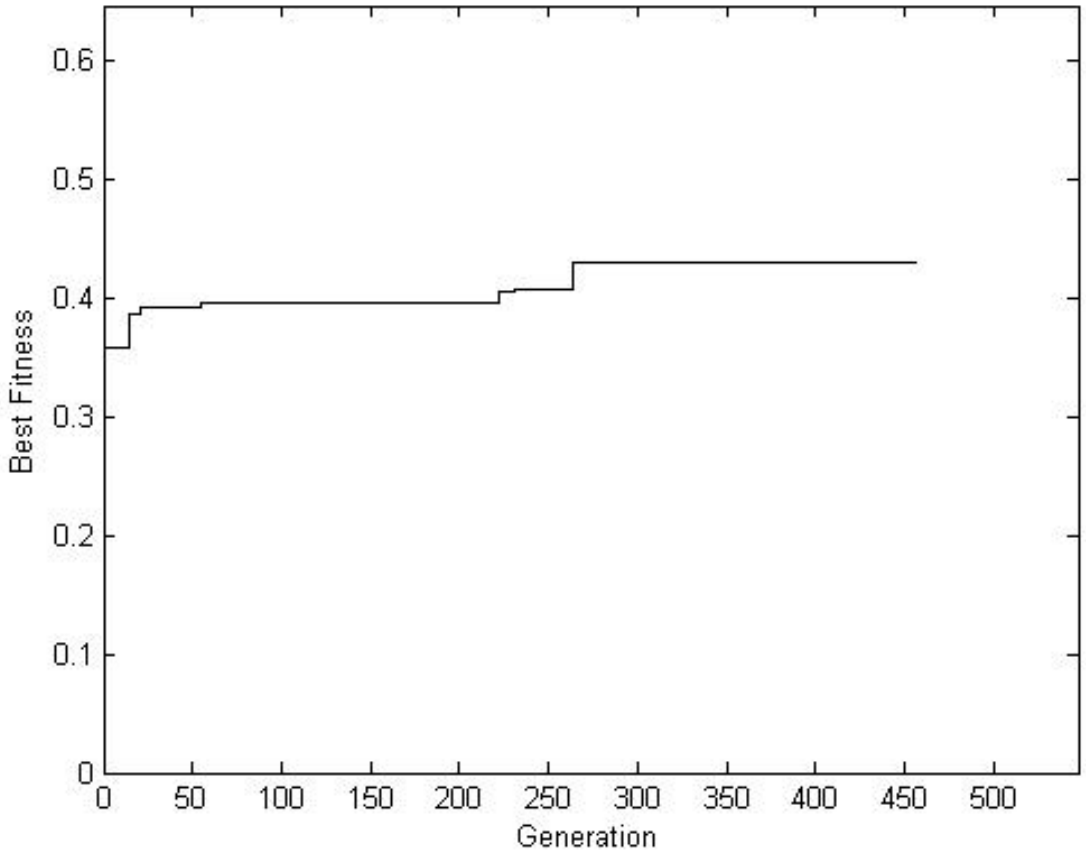

Figure 3.1: Best fitness vs. generation.

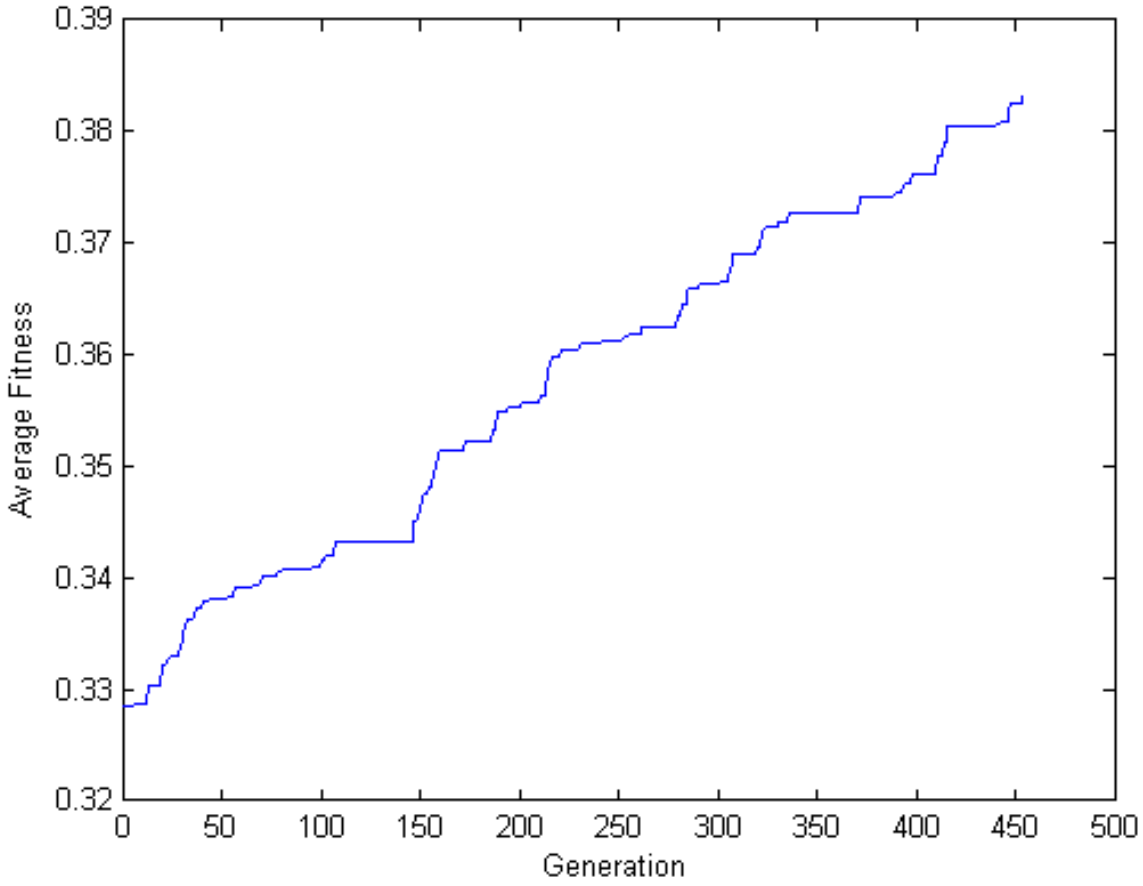

Figure 3.2: Average fitness vs. generation.

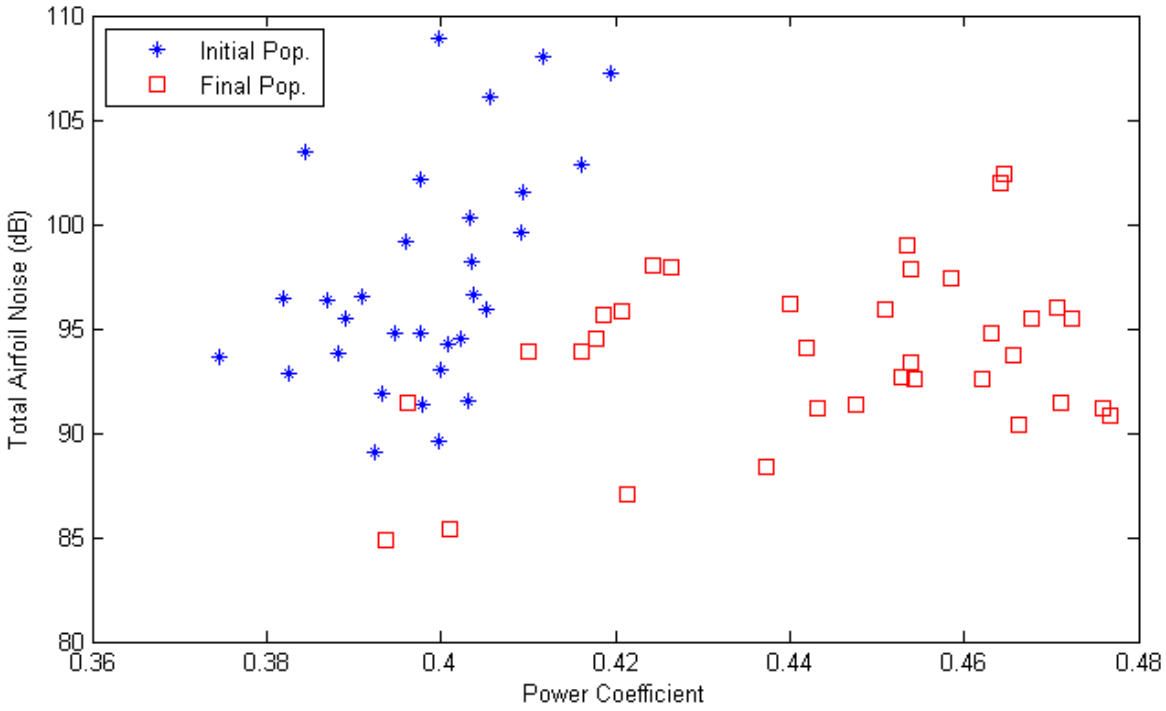

Figure 3.3: Initial and final power coefficients and airfoil noise levels for population of 35 rotors.

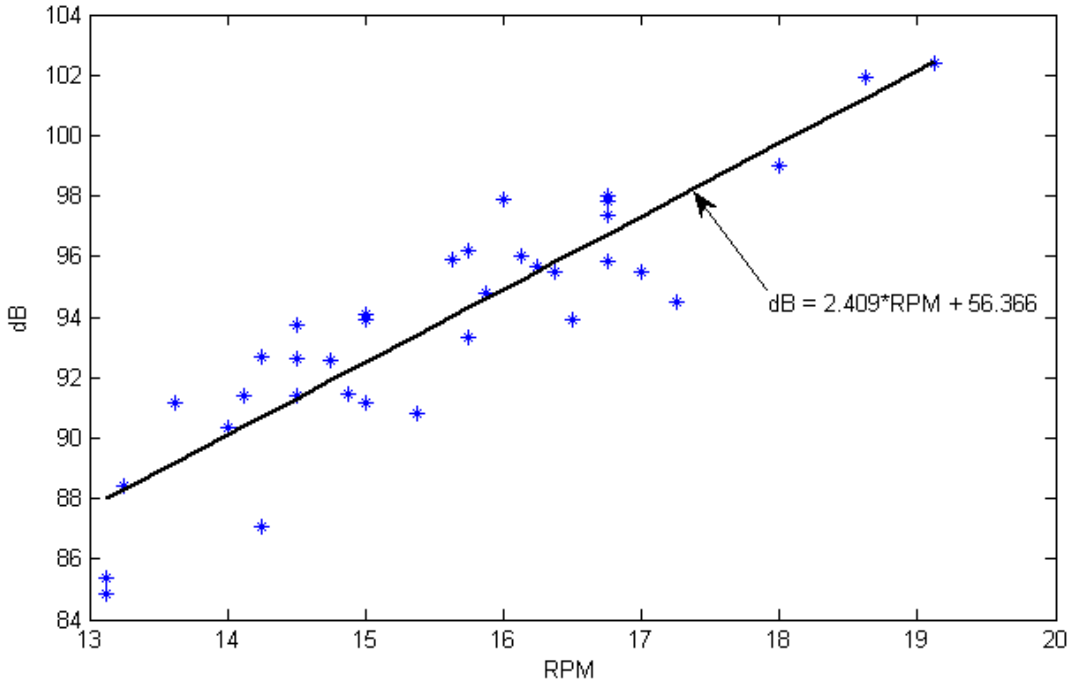

Figure 3.4: Noise level versus RPM for all of the optimized rotors in the final population. The line of best fit is shown in black.

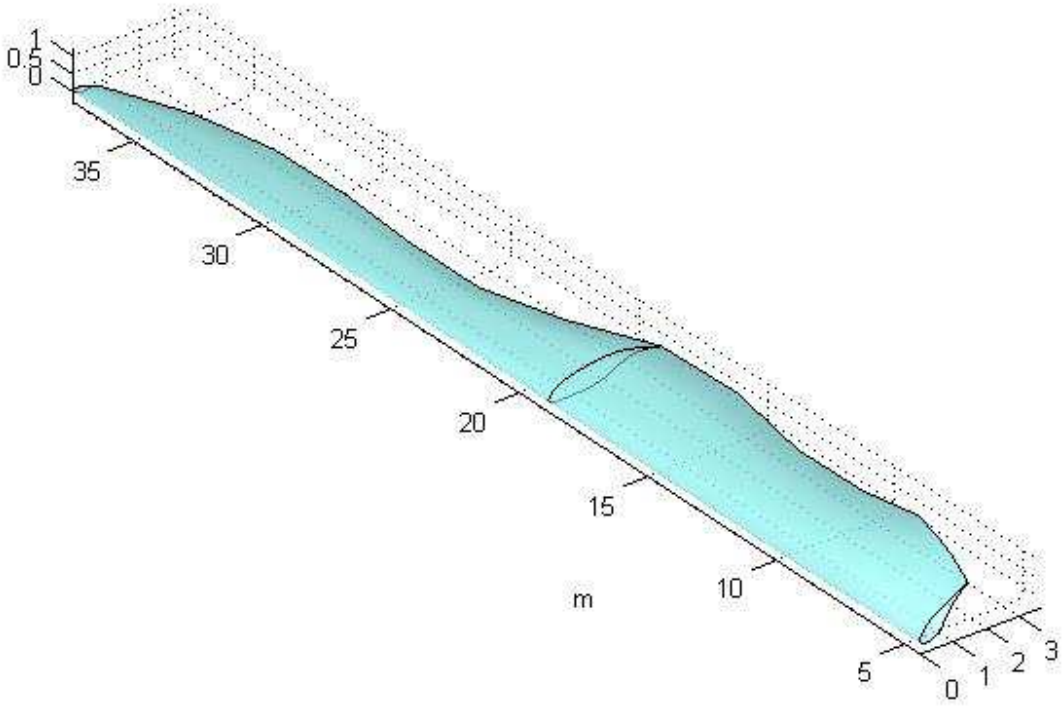

Figure 3.5: Optimized rotor with highest fitness (Solution A).

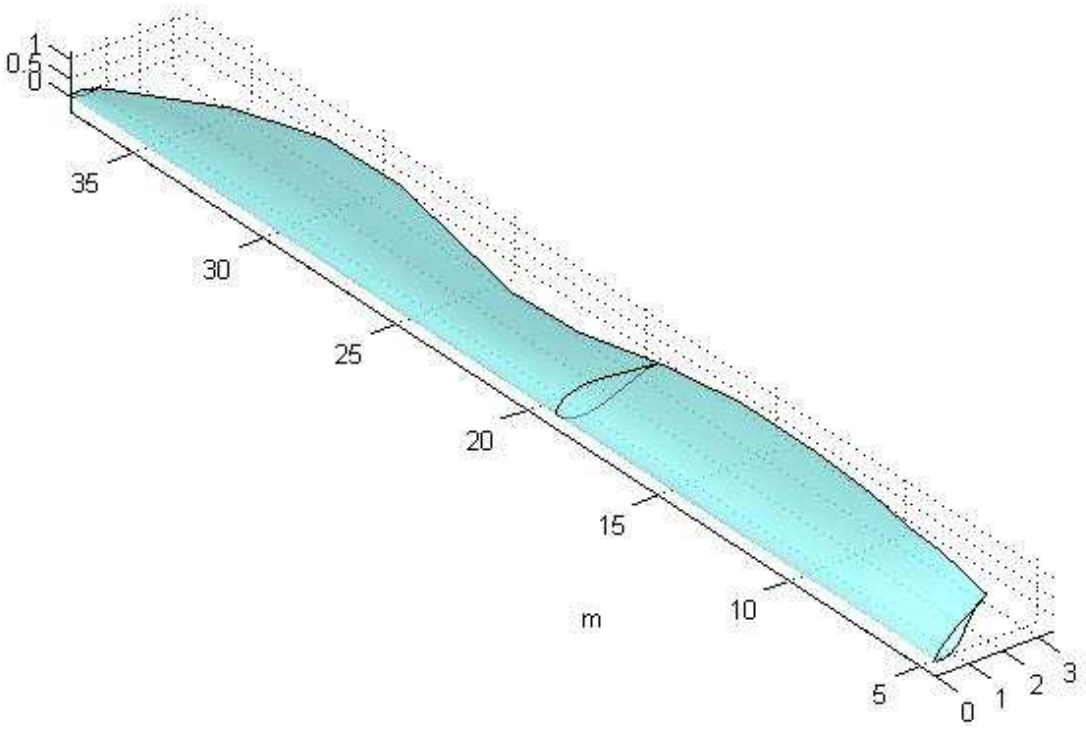

Figure 3.6: Optimized rotor with second highest fitness (Solution B).

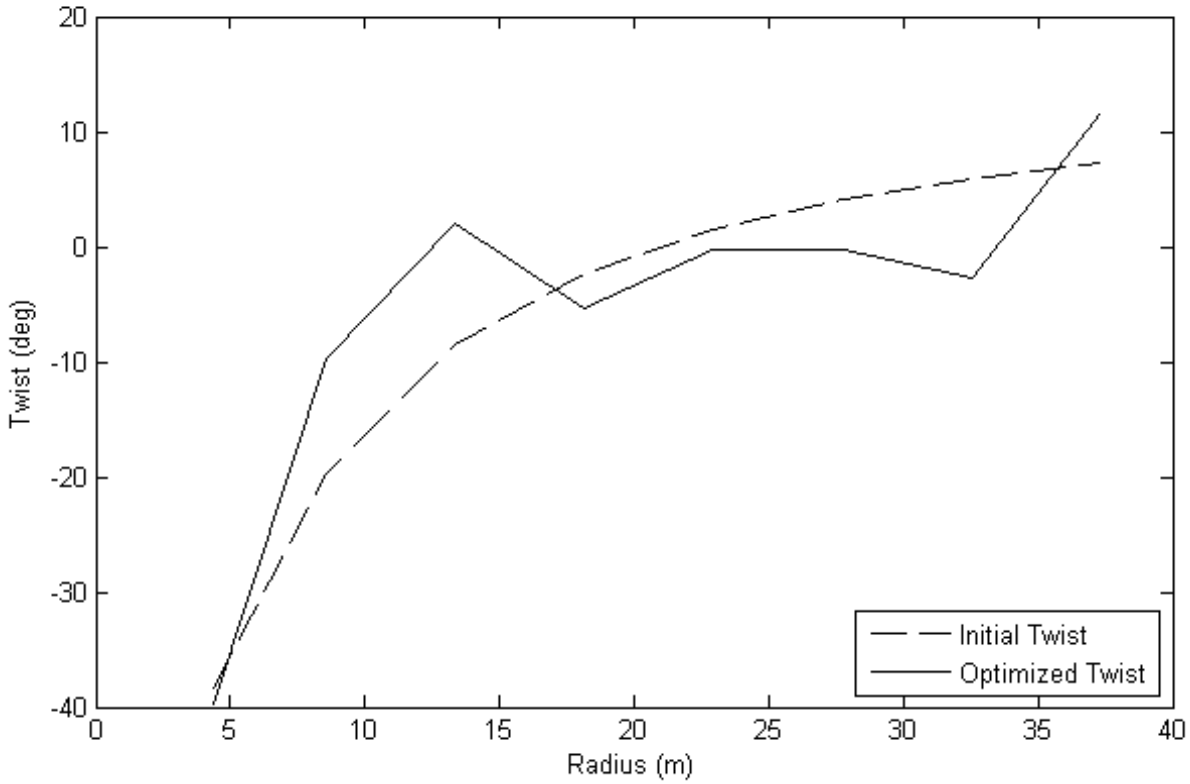

Figure 3.7: Solution A twist distribution.

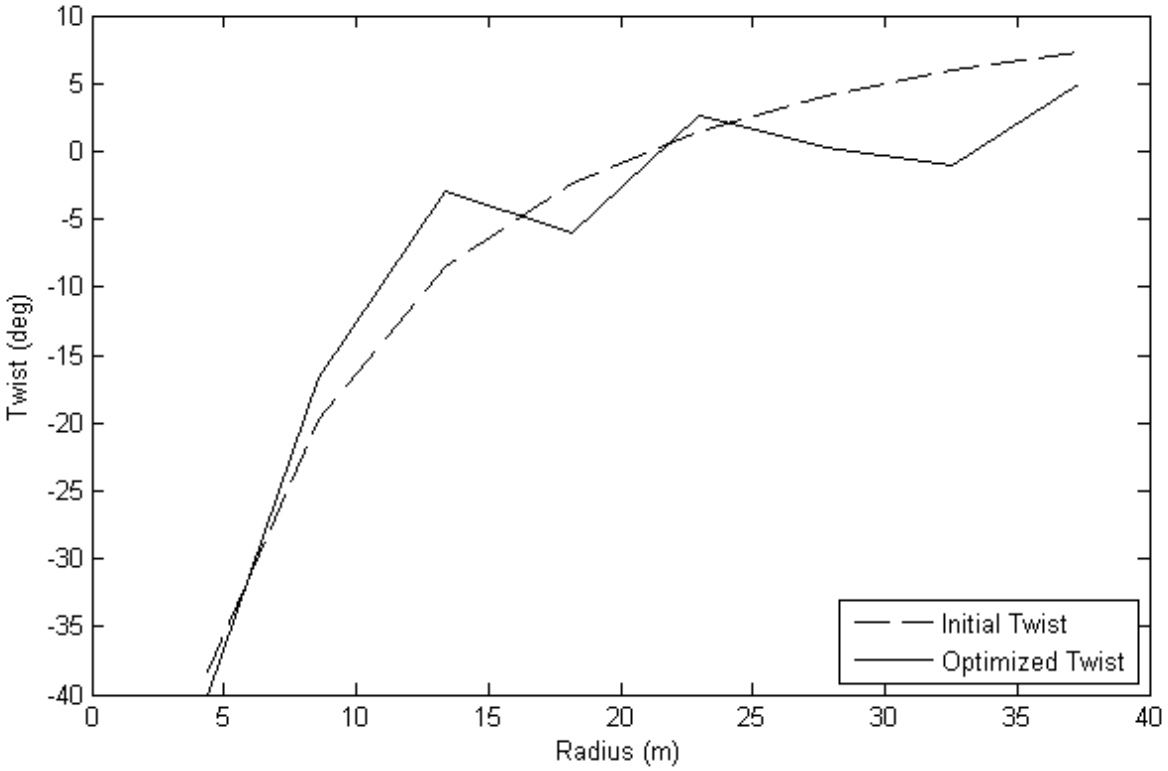

Figure 3.8: Solution B twist distribution.

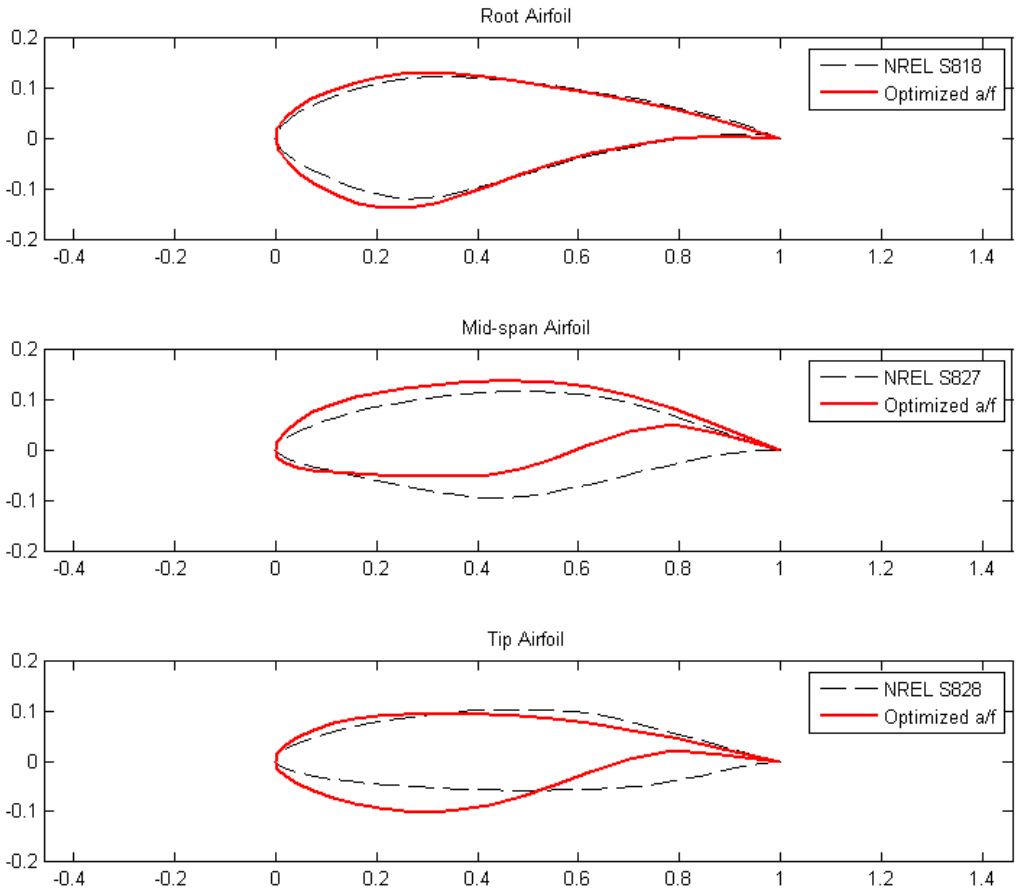

Figure 3.9: Initial and optimized airfoil shapes for Solution A.

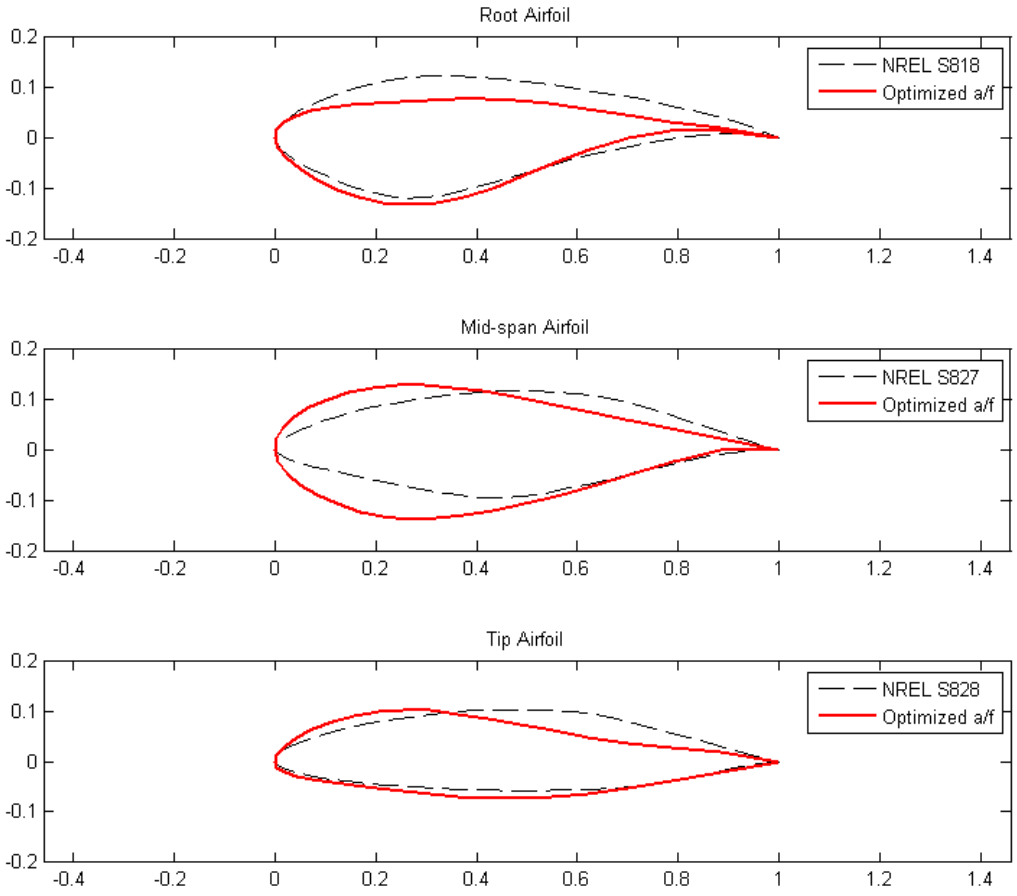

Figure 3.10: Initial and final airfoils for Solution B.

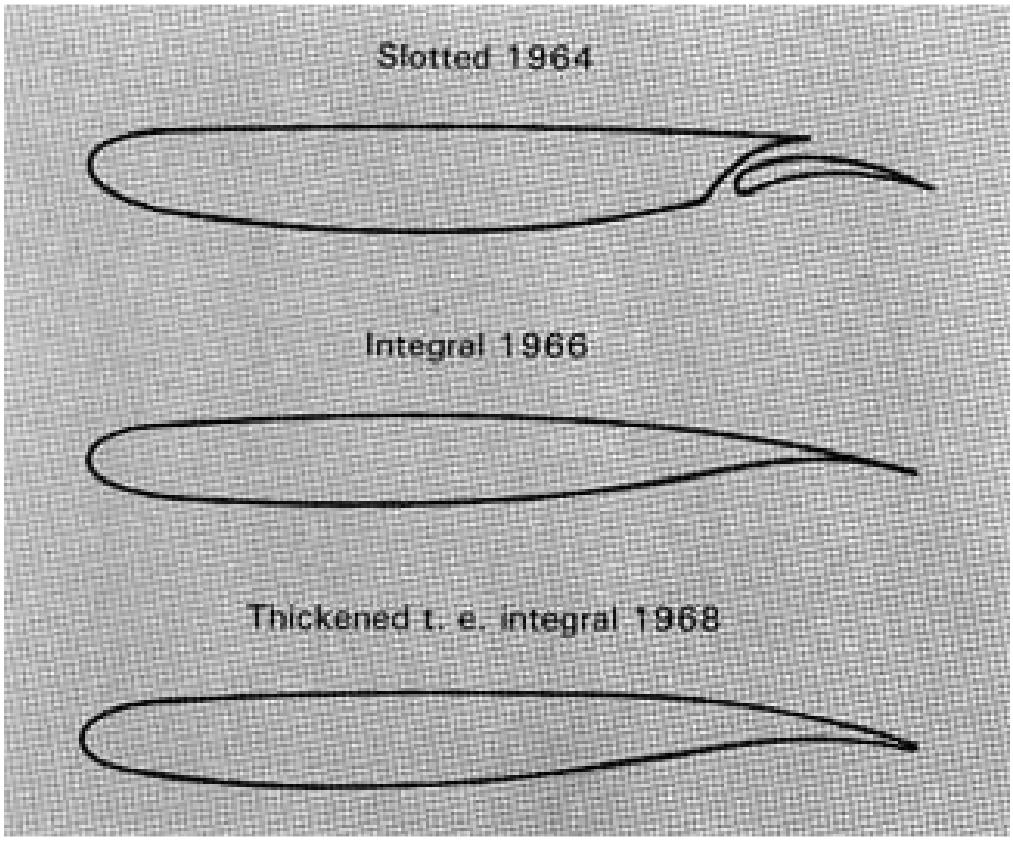

Figure 3.11: Supercritical airfoils [4].

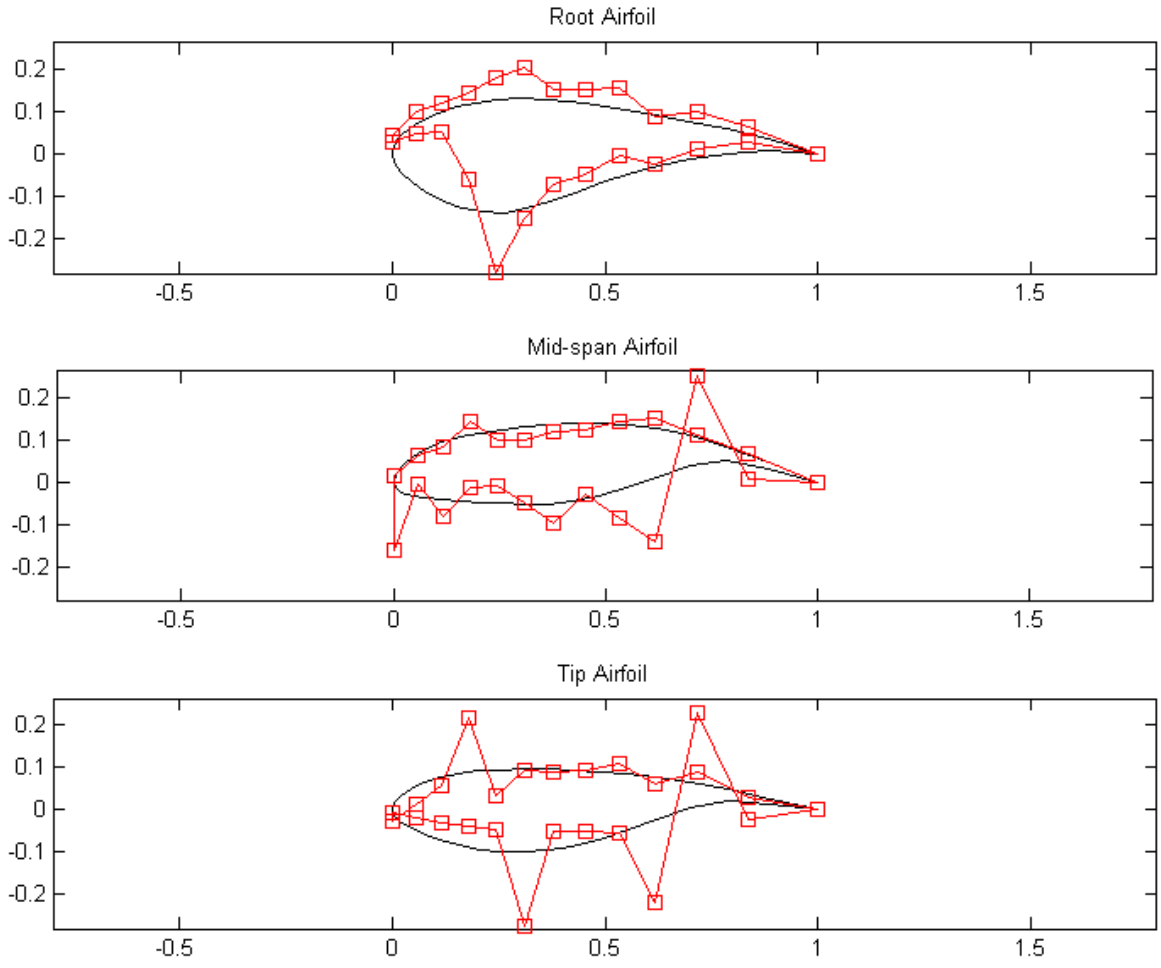

Figure 3.12: Bezier control points and resulting airfoils for Solution A.

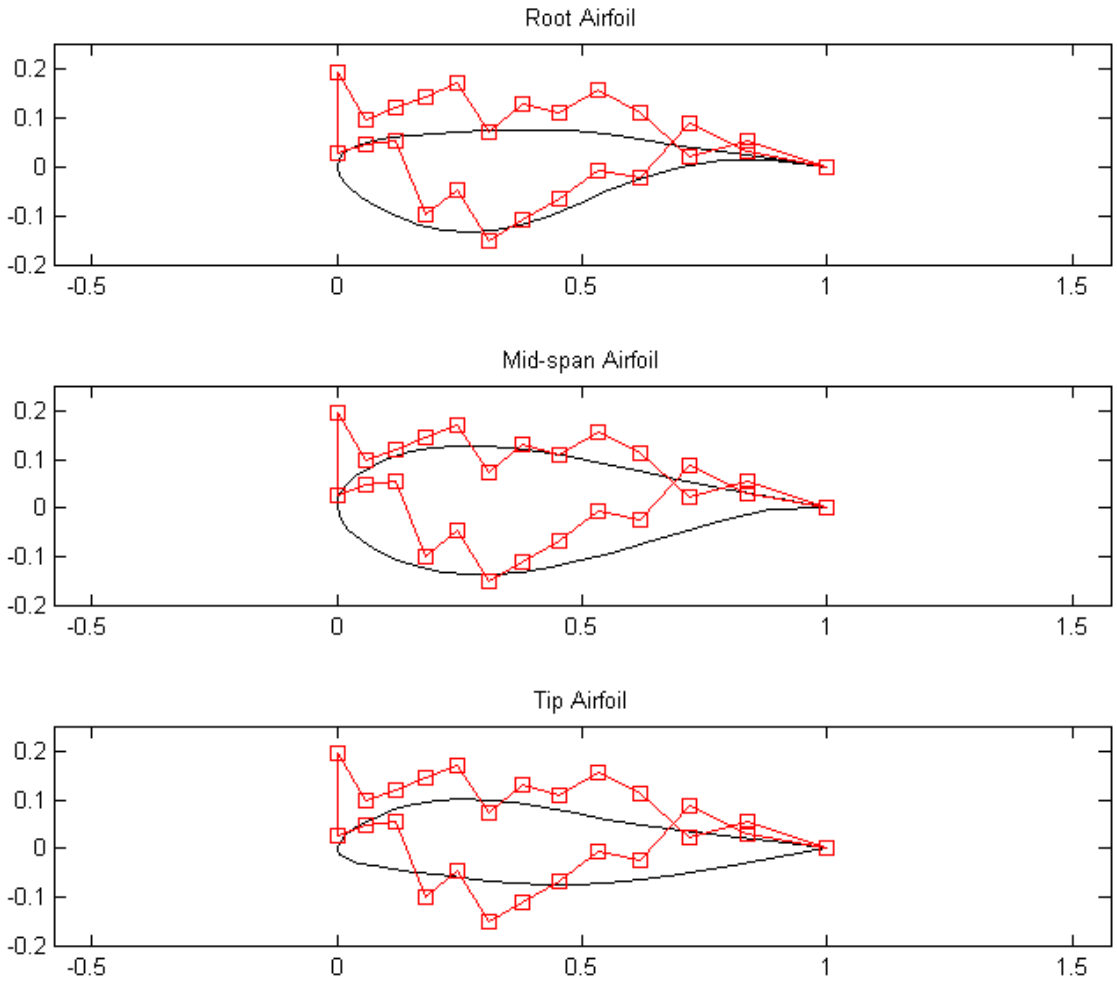

Figure 3.13: Bezier control points and resulting airfoils for Solution B.

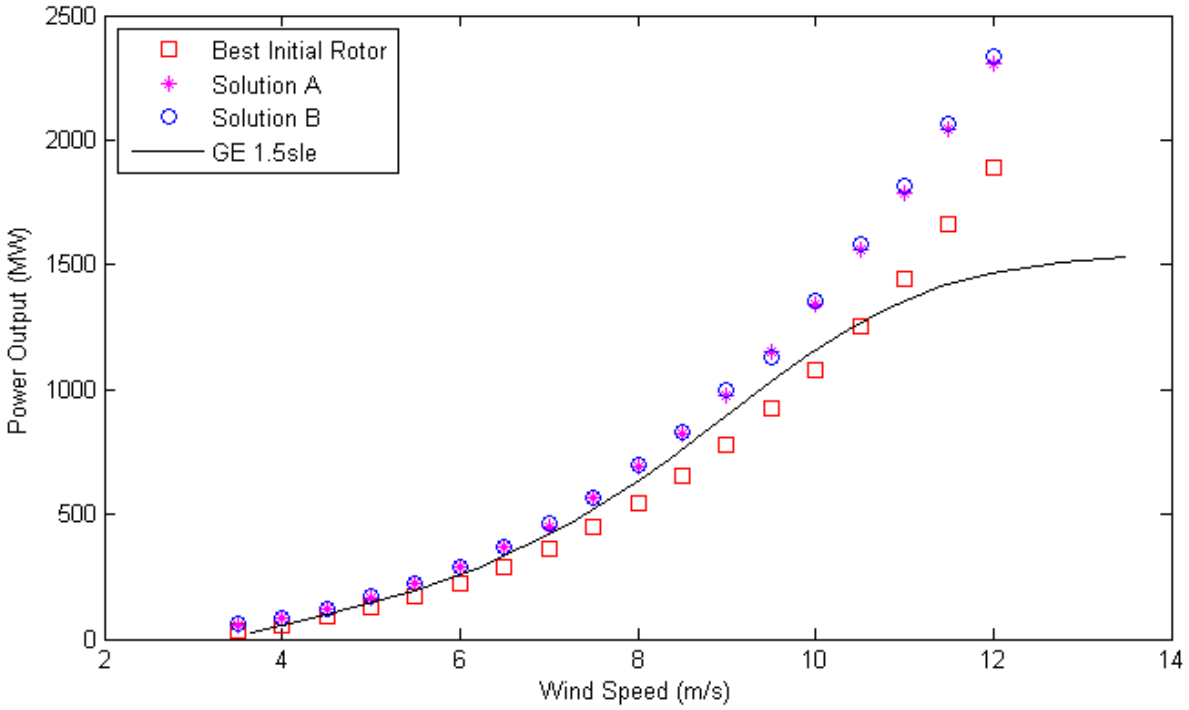

Figure 3.14: Comparison of initial best power curve to Solutions A and B, as well as the data provided for the GE 1.5sle wind turbine, adapted from [9].

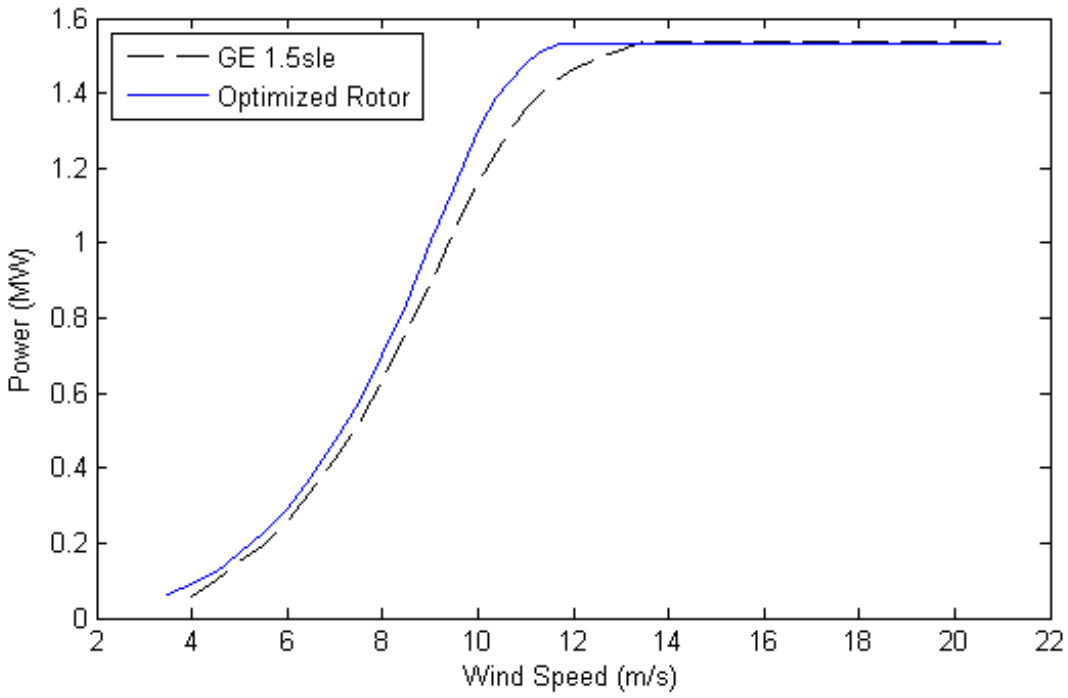

Figure 3.15. Modified power curve for Solution B versus power curve for the GE 1.5sle.

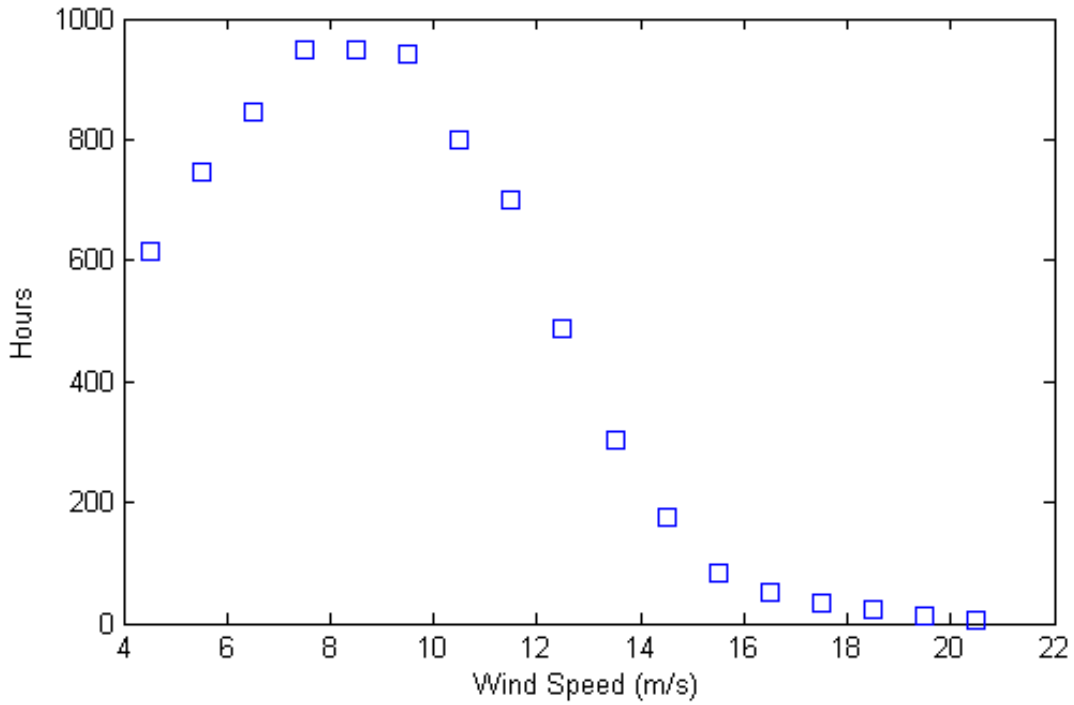

Figure 3.16: Annual wind speed histogram at 50m altitude for White Deer, TX, 1996-1999 [18].

REFERENCES

[1] Alpman, Emre. “Airfoil Shape Optimization Using Evolutionary Algorithms”. Aerospace Engineering Department, Pennstate University.

[2] Burger, Christoph, and Roy Hartfield. "Wind Turbine Airfoil Performance Optimization using the Vortex Lattice Method and a Genetic Algorithm." Proc. of 4th AIAA Energy Conversion Conference, San Diego, CA.

[3] Chambers, Lance. *Practical Handbook of Genetic Algorithms*. Vol. I. New York: CRC, Inc., 1995

[4] "Chapter 7 - The Post-Sputnik Renaissance of Aeronautics The Role of Wind Tunnels in Modern Aeronautical Research." *Wind Tunnels of NASA*. Web. 30 Oct. 2009. <http://history.nasa.gov/SP-440/ch7-3.htm>.

[5] "Curvature -- from Wolfram MathWorld." *Wolfram MathWorld: The Web's Most Extensive Mathematics Resource*. Web. Fall 2009. <http://mathworld.wolfram.com/Curvature.html>.

[6] Deb, Kalyanmoy. “Evolutionary Algorithms for Multi-Criterion Optimization in Engineering Design”. Kanpur Genetic Algorithms Laboratory, Department of Mechanical Engineering, Indian Institute of Technology Kanpur, Kanpur, India, 1998.

[7] "Decibel Level Comparison Chart." *Netwell Noise Control*. Web. Fall 2009. <http://www.esoundproof.com/Screens/Basics/Academy/Sound%20Measurement/Decibels/dBChart.aspx>.

[8] *Environmental Noise*. Environmental Protection Department. Web. Fall 2009. <http://www.epd.gov.hk/epd/noise_education/web/ENG_EPD_HTML/m1/intro_5.html>.

[9] *GE Energy 1.5 MW Wind Turbine*. General Electric, 2009.

[10] Kenway, Gaetan, and Joaquim R. R. A. Martins. "Aerostructural Shape Optimization of Wind Turbine Blades Considering Site-Specic Winds." Proc. of 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Victoria, British Columbia, Canada. Toronto, Ontario, Canada: University of Toronto Institute for Aerospace Studies, 2008.

[11] Kenway, Gaetan, and Joaquim R. R. A Martins. “Aerostructural Shape Optimization of Wind Turbine Blades Considering Site-Specic Winds.” Proc. of 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, British Columbia, Canada, Victoria.

[12] Tan, C.M. "A Comparative Study of Evolutionary Algorithm and Swarm Algorithm for Airfoil Shape Optimization Problems." Proc. of 41st Aerospace Sciences Meeting and Exhibit, Reno, Nevada.

[13] Tangler, J. L. "NREL Airfoil Families for HAWTs." National Renewable Energy Laboratory.

[14] Tangler, J. and Kocurek, J. D., "Wind turbine Post-Stall Airfoil Performance Characteristics Guidelines for Blade-Element Momentum Methods," *43rd AIAA Aerospace Sciences Meeting and Exhibit,* Reno, Nevada, 10-13 January 2005.

[15] Weisstein, Eric W. "Curvature." *MathWorld*--A Wolfram Web Resource. Fall 2009. <http://mathworld.wolfram.com/Curvature.html>.

[16] *Wind turbine Noise 2009*. 2 July 2009. Web. Fall 2009. <http://www.windturbinenoise2009.org/>.

[17] Xuan, Hao et. al. "Aerodynamic and Aeroacoustic Optimization of a Wind Turbine Blade by a Genetic Algorithm". China Academy of Aerospace Aerodynamics. Beijing, 2008.

[18] Nelson, Vaughn. *Wind Energy: Renewable Energy and the Environment*. Boca Raton, FL: CRC, 2009.

APPENDIX A

ea.m

```
%this version of eaairfoil is written with the intent that only the y
%coordinates of the points of the bezzier curve will be mobile

fclose('all')
close all
clear all
clc

runtype = 1;  %input('Saved (1) or random (2) population?  ');
gencount = 1;

%GA parameters
numpoints = 24; %EVEN NUMBER ONLY - number of Bezier curve control points
genesize = 8;   %number of bits in one gene
popsize = 35;   %members in population
genmax = 100;  %max number of generations
numgenes = numpoints*3+8+8; %airfoil control points, chord, twist
genelength = genesize*numgenes;  %length of the entire genome
fitstop = 1;    %fitness level at which EA stops
bpts = 36;      %EVEN NUMBER ONLY: number of bezzier points
mutprob = 0.04; %probability of mutation for pop vector randomization NOT for
breeding; that parameter is found in breed()
popfrac = .5;   %how much of population to breed step by step each generation

%design parameters relavent to twist
vdes = 10;  %design wind velocity
rpmdes = 15;  %approximate rpm value (value at which rpm optimization begins)
omegades = rpmdes*2*pi/60;  %rpmdes in rad/sec
aoades = 17;
turbparams = [aoades, omegades, vdes];

%ROTOR PARAMS
radii = [4.40 linspace(8.54,37.34,7)]';  %vector of radial locations where
the chord and twist will be defined
chord = [1.51 2.77 2.48 2.19 1.90 1.61 1.31 1.02]';  %initial chord values

%creates twist that accurately draws the rotor blade, but must be of
%opposite sign to function correctly in FAST
twistfunc = inline('-(atand(vdes./(radius*omegades))-aoades)');
twist = twistfunc(aoades,omegades,radii,vdes);

%read saved values for population and GA parameters
if runtype == 1  %this value was set above -- if runtype == 1, a previously
saved population will be loaded, as defined in the following lines
    oname = 'backup/r2_evolve3';  %name of file containing saved population -
- .dat will be added later in the program
    sname = 'junk';  %filename to save the population

    %FILENAME CONVENTION
    %
    % the filename under which to save the population is stored in sname.
```

```
    % Three other files are created by appending 'initial', 'fit', and
    % 'param' onto the end of sname.
    %  - snameinitial saves the population before it begins evolving
    %  - snamefit saves a table of the fitness values versus generation
number.
    %  - snameparam saves numpoints, genesize, genelength, and prints out
    %    the average fitness of each generation.
    %
    %The data saved in snamefit and snameparam is read in if selected in
    %the section titled 'READ PARAMETERS AND AVERAGE FITNESS' below.


    %check to vibrate population
    ifvib =.3;  %if this value is nonzero (positive only) the population will
be "vibrated" (see "VIBRATE AND RECHECK FITNESS" below)

    %check to recalculate all fitnesses by running XFOIL and FAST
    if ifvib == 0
        fitsetchk = 'n';  %checks whether to recalculate fitnesses for all
population members
        if fitsetchk == 'y'
            fitset = 1;  %fitset is input into the function getfitness --
            %             0:  only airfoil geometry will be checked for
validity by getfitness
            %             1:  getfitness will run XFOIL, FAST, and NAFNoise
            %             and calculate the total fitness of a population
            %             member
        else
            fitset = 0;
        end
    else
        fitset = 0;
    end

    %read in population matrix from file with saved population
    fid = fopen([oname '.dat'],'r');
    pop = fscanf(fid, '%f',[genelength+5 inf]);
    pop = pop';
    fclose(fid);

    %READ PARAMETERS AND AVERAGE FITNESS
    oldchk = 'y';  %check to determine whether to import population data and
average fitness vs generation
    if oldchk == 'y'
        %import population parameters
        fid = fopen([oname 'param.dat'],'rt');
        params = fscanf(fid,'%f')';
        fclose(fid);
        avgfit = params(4:length(params));

        %import max fitness vs generation data
        fid = fopen([oname 'fit.dat'],'r');
        maxfit1 = fscanf(fid, '%f');
        maxfit(1,:) = maxfit1(1:length(maxfit1)/2);
        maxfit(2,:) = maxfit1(length(maxfit1)/2+1:length(maxfit1));
        fclose(fid);
```

```
        gencount = maxfit(1,length(maxfit(1,:)));    %find generation number
        mfcount = length(maxfit(1,:));  % the generation number of the most
recent increase in maximum fitness
    else
        mfcount = 1;  %sets max fitness generation number to 1
    end

    %create vector of x-coordinates for Bezier control points
    x1 = 1:2*9/numpoints:10;
    x2 = log(x1);
    x3 = 0:2*max(x2)/numpoints:max(x2);
    x4 = (x2+x3)/2/max(x2);
    xlocs = [x4, fliplr(x4)]';

    %get fitnesses for pop, delete members w/neg fitness
    i=1;

    %The section below calculates the fitness for each member of the
    %population.  If the fitness comes back negative, that member is
    %invalid due to unacceptable geometry or a problem calculating the
    %aerodynamic or noise values.  If a member is invalid, it is deleted
    %from the population.  After all members have been checked, the empty
    %spots in the population are all filled with the most fit member.

    while i<=length(pop(:,1))
        [popmem pointsfit] =
getfitness(genelength,xlocs,bpts,radii,genesize,numpoints,pop(i,:),fitset);
        if pointsfit<0
            pop(i,:) = [];
            fprintf('Deleted %d\n',i)
        else
            pop(i,:) = popmem;
            i = i+1;
        end
        savepop(genelength,pop,sname);
    end
    pop = sortrows(pop,genelength+1);

    %fill population if too few members
    poplen = length(pop(:,1));
    if poplen<popsize
        for i = poplen:popsize-1
            pop(i+1,:) = pop(poplen,:);
        end
    end

    savepop(genelength,pop,sname);

    %VIBRATE AND RECHECK FITNESSES----------------------------------------
    if ifvib ~= 0
        %vVIBRATE - each of the points in the twist and chord vectors, as
        %well as the y-coordinates for all the Bezier control points, are
        %randomly adjusted a small amount as described in vibrate.m

        newpop = vibrate(ifvib,popsize,pop,numpoints,genesize,genelength);
```

```
        pop = recfits(newpop,genelength,popsize);

        %The fitnesses of every member of the population are rechecked to
        %remove members for which the vibration created an invalid rotor.
        i=1;
        while i<=length(pop(:,1))
            [popmem pointsfit] =
getfitness(genelength,xlocs,bpts,radii,genesize,numpoints,pop(i,:),1);
            if pointsfit<0
                disp(pointsfit)
                pop(i,:) = [];
                fprintf('Deleted %d after vibrate\n',i)
            else
                pop(i,:) = popmem;
                i = i+1;
            end
            savepop(genelength,pop,sname);
        end
    end
    pop = sortrows(pop,genelength+1);

    %fill population if too few members
    poplen = length(pop(:,1));
    if poplen<popsize
        for i = poplen:popsize-1
            pop(i+1,:) = pop(poplen,:);
        end
    end

    %save population
    savepop(genelength,pop,sname)
    disp('vibrated')
    %     pop(:,length(pop)-2) = recfits(pop,genelength,popsize); %correct
for
    %     similarity to most fit member.

    %FINISHED VIBRATING AND RECHECKING FITNESSES--------------------------

    %calculate first average fitness value if not importing population data
    %also set the first value for variable tracking the maximum fitness
    if oldchk == 'n'
        avgfit(1) = mean(pop(:,genelength+1));
        maxfit(1:2,:) = [1; max(pop(:,genelength+1))];
    end

    %begin the section dedicated to creating a new population from scratch
    %THIS SECTION MAY NO LONGER FUNCTION PROPERLY -- For the latter stages
    %of the project, a preset population was used, ie population type 1, so
    %so as the functionality of the code was tweaked this section was not
    %correspondingly updated as there was no need.
else
    sname = 'rotor_evolve5';%input('File name to save population matrix (no
spaces):  ','s');
    %create x vector of positions of bezzier points
    x1 = 1:2*9/numpoints:10;
    x2 = log(x1);
```

```
    x3 = 0:2*max(x2)/numpoints:max(x2);
    x4 = (x2+x3)/2/max(x2);
    xlocs = [x4, fliplr(x4)]';

    %initialize population
    %create initial shape
    initaf(1:numpoints/2) = 45;
    initaf(numpoints/2+1:numpoints) = 80;
    for i = 1:numpoints
        binpt = dec2bin(2*initaf(i));
        for j = 1:length(binpt)
            tempop(genesize*(i-1)+j+genesize-length(binpt)) =
str2num(binpt(j));
        end
    end
    %airfoil Bezzier template
    for i = 1:popsize
        pop(i,1:3*length(tempop)) = [tempop tempop tempop];
    end

    %initialize random population
    maxfit = 0;
    mfcount = 1;
    for i = 1:popsize
        disp(i)
        fitcheck = -1;
        while fitcheck<0
            %randomize the genome
            for j = 1:genelength
                if rand < mutprob
                    pop(i,j) = pop(i,j)+1-2*pop(i,j);
                end
            end
            %get bezzier airfoil curve
            [junk fitcheck] =
getfitness(genelength,xlocs,bpts,radii,genesize,numpoints,pop(i,:),1);
            if fitcheck>0
                pop(i,genelength+1) = fitcheck;
            end
        end %randomize genome
    end  %create random population
end
%end of the section in which a new population is created from scratch

%create a vector of the x-coordinates for the Bezier curve control points
%this groups the points slightly closer in the region of the leading edge
x1 = 1:2*9/numpoints:10;
x2 = log(x1);
x3 = 0:2*max(x2)/numpoints:max(x2);
x4 = (x2+x3)/2/max(x2);
xlocs = [x4, fliplr(x4)]';

pop = sortrows(pop,genelength+1);  %sort population vector by fitness
fprintf('The best initial fitness is %f.\n',maxfit(2,length(maxfit(2,:))))
savepop(genelength,pop,[sname 'initial']);  %save initial population
```

```
%The below while loop is the actual Genetic Algorithm section of the code,
%where genomes are combined and offspring created and selected from.  This
%is run until genmax is reached, or some nominal fitness value is achieved.

disp(pop(:,genelength+1))

%three lines below initiate the figure window used to display the three
%airfoil shapes which define up to 20 members of the population in a
%subplotted window
figure
screen_size = get(0, 'ScreenSize');
set(gcf, 'Position', [0 100 screen_size(3) screen_size(4)*.8 ] );

while gencount<genmax && maxfit(2,mfcount)<fitstop %EVOLVE
    %display superpositions of the three airfoils for selected members of
    %the population
    drawfirst = 30;  %first member of population to be plotted
    try
        for j = drawfirst:drawfirst+19  %displays 20 members
            [points chord rtwist] = decodeaf(genesize,numpoints,pop(j,:));
            bcurve = bezzier(bpts,[xlocs, points]);
            subplot(3,2,j+1-drawfirst)
            plot(bcurve(:,1),bcurve(:,2),'r','LineWidth',3)
            hold on
            plot(bcurve(:,3),bcurve(:,4),'b','LineWidth',2)
            plot(bcurve(:,5),bcurve(:,6),'m','LineWidth',1)
            axis equal
            hold off
            title(num2str(j))
            set(gcf,'Color','w')
            axis off
        end
        drawnow
    catch
        disp('could not draw airfoils')
    end

    gencount = gencount+1;  %count up to the next generation
    for i=1:round(popsize*popfrac)     %popfrac is fraction of population
which gets mated every generation: default = 0.5
        tic
        %step down through population starting at most fit
        %randomly choose pop member to breed, weighted with higher fitness
        mom = popsize+1-i;
        dad = mom;

        %create "roulette wheel" type selection of mate for mom
        totfit = sum(pop(:,genelength+1));
        for j = 1:popsize
            wheel(j) = sum(pop(1:j,genelength+1))/totfit;
        end

        %select dad - moves through roulette wheel until the value chosen
        %for dad is >= the value in the roulette wheel.  Also prevents the
```

```
        %situation mom = dad.
        while dad == mom
            dad = rand;
            for j = 1:popsize
                if dad<wheel(j)
                    dad = j;
                    break
                end
            end
        end

        %create a daughter (kidA), which is the mom's genome with pieces from
the
        %dad inserted, and a son (kidB), which is the dad's genome with the
        %corresponding pieces from the mom inserted
        [kidA, kidB] = breed(numgenes,genesize,pop(mom,:),pop(dad,:));

        %get fitness for children
        [kidA kidAfit] =
getfitness(genelength,xlocs,bpts,radii,genesize,numpoints,kidA,1);
        [kidB kidBfit] =
getfitness(genelength,xlocs,bpts,radii,genesize,numpoints,kidB,1);

        %easier variable names for parents' fitnesses
        momfit = pop(mom,genelength+1);
        dadfit = pop(dad,genelength+1);

        %if daughter is better than mom, she replaces mom.  if son is
        %better than dad, he replaces dad.  if a one child is better than
        %one parent and the other is worse than both, the former replaces
        %the worse parent.
        if kidAfit > momfit
            pop(mom,:) = kidA;
        end
        if kidBfit > dadfit
            pop(dad,:) = kidB;
        end
        if kidAfit < momfit && kidAfit < dadfit && (kidBfit > dadfit ||
kidBfit > momfit)
            if kidBfit > momfit
                mom = kidB;
            else
                dad = kidB;
            end
        elseif kidBfit < momfit && kidBfit < dadfit && (kidAfit > dadfit ||
kidAfit > momfit)
            if kidAfit > momfit
                mom = kidA;
            else
                dad = kidA;
            end
        end

        %save population
        savepop(genelength,pop,sname);
        toc
```

```
    end   %end breed

    fprintf('Best fitness after %.0f generations is
%.5f.\n',gencount,maxfit(2,mfcount))

    %update maxfit - create step plot
    maxfit(:,mfcount+1) = [gencount;maxfit(2,mfcount)];
    if maxfit(2,mfcount) < max(pop(:,genelength+1));
        disp('Update all fitnesses')
        %update fitness counting
        mfcount = mfcount+2;
        maxfit(1,mfcount) = gencount;
        maxfit(2,mfcount) = max(pop(:,genelength+1));
        %recalculate all fitnesses
        newpop = recfits(pop,genelength,popsize);
    end

    %resort population by fitness
    pop = sortrows(pop,genelength+1);

    %update avgfit and save population
    avgfit(gencount-1) = mean(pop(:,genelength+1));
    savepopdata(numpoints,genesize,genelength,avgfit,maxfit,sname);
toc
end   %evolve

%save population and GA parameters
savepop(genelength,pop,sname);
savepopdata(numpoints,genesize,genelength,avgfit,maxfit,sname)

fprintf('Final population size = %.0f.\n',popsize) %present final data

%display best airfoils (for code description see getfitness.m)
for i = popsize:-1:33
    [points chord rtwist] = decodeaf(genesize,numpoints,pop(i,:));
    bcurve = bezzier(bpts,[xlocs, points]);

    %build rotor
    rootaf = [-bcurve(:,1)+1, bcurve(:,2)];
    midaf = [-bcurve(:,3)+1, bcurve(:,4)];
    tipaf = [-bcurve(:,5)+1, bcurve(:,6)];
    n = 15;
    rootsegs = 4;  %number of segments of innermost rotor section
    X(length(rootaf),n) = 0;
    Y(length(rootaf),n) = 0;
    Z(length(rootaf),n) = 0;
    for j = 1:length(rootaf)
        X(j,:) =
interp1(1:3,[rootaf(j,1),midaf(j,1),tipaf(j,1)],linspace(1,3,n));
        Y(j,:) = [linspace(radii(1),radii(2),rootsegs)
linspace((radii(3)+radii(2))/2,radii(8),n-rootsegs)];
        Z(j,:) =
interp1(1:3,[rootaf(j,2),midaf(j,2),tipaf(j,2)],linspace(1,3,n));
    end
```

```
    %create smoothed chord vector, apply scale transformation to blade
    %segments
    smchord = spline(radii,chord);
    scale = [interp1(radii,chord,linspace(radii(1),radii(2),rootsegs)),
ppval(smchord,Y(1,rootsegs+1:n))];
    for j = 1:length(X(1,:))
        X2(:,j) = X(:,j)*scale(j);
        Z2(:,j) = Z(:,j)*scale(j);
    end

    %apply twist to rotor - create smoothed twist vector as above
    smtwist = spline(radii,rtwist);
    sectwist = ppval(smtwist,Y(1,:));
    for j = 1:length(X2(1,:))
        rotpoint = [0 0];  %rotates about leading edge
        xsec = [X2(:,j)-rotpoint(1),Z2(:,j)-rotpoint(2)];
        rotmat = [cosd(sectwist(j)) -sind(sectwist(j)); sind(sectwist(j))
cosd(sectwist(j))];
        xsec = xsec*rotmat;
        X2(:,j) = xsec(:,1)+rotpoint(1);
        Z2(:,j) = xsec(:,2)+rotpoint(2);
    end

    %shaded surface + outlines
    figure
    h = surf(X2,Y,Z2,'FaceColor',[.5 .8 .8],'EdgeColor','none');
    hold on
    camlight left; lighting phong
    set(gca,'DataAspectRatio',[1 10 1])
    set(h,'AmbientStrength',.8);
    plot3(X2(1,:),Y(1,:),Z2(1,:),'k')
    plot3(X2(:,1),Y(:,1),Z2(:,1),'k')
    plot3(X2(:,n),Y(:,n),Z2(:,n),'k')
    plot3(X2(:,round(n/2)),Y(:,round(n/2)),Z2(:,round(n/2)),'k')
    % plot3(s827(:,1),Y(1:length(s818),n/2),s827(:,2))
    alpha(0.6)
    grid on
end

%plot maxmium fitness versus generation
figure
plot(maxfit(1,:),maxfit(2,:),'k')
xlabel('Generation')
ylabel('Best Fitness')
% title('Best Fitness vs Generation')
axis([0 gencount*1.2 0 max(maxfit(2,:))*1.5])

%plot avg fitness versus generation
figure
plot(avgfit)
xlabel('Generation')
ylabel('Average Fitness')
title('Average Fitness vs Generation')
```

vibrate.m

```
%This function adds random variations to a population of rotors, treating
%each design value differently
%
%INPUTS: MA (range of variability for Bezier curve control points)
% popsize (number of members in population)
% pop (population of genomes)
% numpoints (number of variable Bezier curve control points (total number
% of Bezier points - 2))
% genesize (number of bits in one gene)
% genelength (number of bits in an entire genome, plus the values that are
% stored at the end of the genome such as fitness, efficiency, RPM, pitch,
% and noise level
%
%OUTPUTS:  newpop (randomized population based on input population)

function newpop = vibrate(MA,popsize,pop,numpoints,genesize,genelength)
chordvar = .3;  %range of variability for chord
twistvar = 5;  %range of variability for twist

%leaves best two members of population untouched
for i = 1:popsize-2
    [points chord twist] = decodeaf(genesize,numpoints,pop(i,:)); %design
values from genome

    %vibrate control points
    for j = 1:length(points(1,:))
        newpoints(:,j) = points(:,j).*(1+MA*(0.5-rand(length(points),1)));
    end

    %calculate the binary value needed to represent the new control points
    %locations based on the equation used to define control point values in
    %decodeaf()
    points1 = newpoints(2:length(newpoints)-1,:);
    points1 = (points1+.3)/.002353;

    %correct for out of range gene values (greater than 255 or less than 0)
    for m = 1:length(points1)
        if points1(m)<0
            points1(m) = 0;
        elseif points1(m) > 255
            points1(m) = 255;
        end
    end

    %convert to binary and build genome
    bins = dec2bin([points1(:,1)' points1(:,2)' points1(:,3)']);
    for m = 1:length(points(1,:))
        for k=1:numpoints
            for j = 1:genesize
                newpop(i,(m-1)*genesize*numpoints+genesize*(k-1)+j) =
str2num(bins((m-1)*numpoints+k,j));
            end
        end
```

```
end

%calculate vibrated chord, convert to binary
chord = chord.*((1-chordvar/2)+rand(length(chord),1)*chordvar);

%correct for out of range values
for j = 1:8
    if chord(j)<.8
        chord(j) = .8;
    elseif chord(j)>3.5
        chord(j) = 3.5;
    end
end

%convert to binary
bins = dec2bin((chord-.8)/(2.7/255));
space = 8-length(bins(1,:));
if space>0
    bins2(1:8,1:space) = '0';
    bins = [bins2 bins];
end

%construct separate genome just for the chord to be added to the
%overall genome later
for j = 1:8
    for k = 1:8
        chordpop(8*(j-1)+k) = str2num(bins(j,k));
    end
end

%calculate vibrated twist, convert to binary
twist = twist-twistvar/2+twistvar*(rand(length(twist),1));

%correct for out of range values
for j = 1:8
    if twist(j)<-70
        twist(j) = -70;
    elseif twist(j) > 15
        twist(j) = 15;
    end
end

%convert to binary
bins = dec2bin((-twist+15)/(85/255));
space = 8-length(bins(1,:));
if space>0
    bins2(1:8,1:space) = '0';
    bins = [bins2 bins];
end

%create twist genome to be added to overall genome later as in chord
%above
for j = 1:8
    for k = 1:8
        twistpop(8*(j-1)+k) = str2num(bins(j,k));
```

```
        end
    end

    %add randomized genome, built from the genomes created by the control
    %point, chord, and twist sections of the code above, to the new
    %population
    newpop2(i,:) = [newpop(i,:) chordpop twistpop];
end

newpop = newpop2;

%adds the old two best fit members to new population unaltered
newpop(popsize-1:popsize,:) = pop(popsize-1:popsize,1:genelength);

%adds the spaces for the informational columns in the last five places of
%the genome
newpop(:,genelength+1:genelength+5) = 0;
```

breed.m

```
%this function in written with it in mind for only the y points of the
%bezzier curve to be mobile
%
%INPUTS:  numgenes (number of genes, or design values, for the
%optimization), genesize (number of bits per gene), mom and dad (genomes of
%the mother and father)
%
%OUTPUTS:  kidA and kidB (genomes of the two offspring produced by the
%breeding process

function [kidA kidB] = breed(numgenes,genesize,mom,dad) %input mom, dad,
afpoints for comparison, xlocs, numpoints

%initialize kidA as mom and kidB as dad
kidA = mom;
kidB = dad;

%probability of mutation for each individual bit in the genomes
mutchance = .025;

%conducts crossover for each gene (group of eight bits) in the genomes
for i = 1:numgenes
    %determine endpoints for crossover, between 1 and 8
    splicepoints(1) = round(1+(genesize-1)*rand());
    splicepoints(2) = round(1+(genesize-1)*rand());
    splicepoints = sort(splicepoints);

    %switch the selected portion of the gene between mother and father
    kidA((genesize*(i-1)+splicepoints(1)):(genesize*(i-1)+splicepoints(2))) =
dad((genesize*(i-1)+splicepoints(1)):(genesize*(i-1)+splicepoints(2)));
    kidB((genesize*(i-1)+splicepoints(1)):(genesize*(i-1)+splicepoints(2))) =
mom((genesize*(i-1)+splicepoints(1)):(genesize*(i-1)+splicepoints(2)));

    %randomly mutate each bit in the gene with the probability assigned
    %above
    for j = 1:genesize
        mut = rand();
        if mut<mutchance
            %flip bit value
            kidA(genesize*(i-1)+j) = kidA(genesize*(i-1)+j)+(1-
2*kidA(genesize*(i-1)+j));
        end
        mut = rand();
        if mut<mutchance
            %flip bit value
            kidB(genesize*(i-1)+j) = kidB(genesize*(i-1)+j)+(1-
2*kidB(genesize*(i-1)+j));
        end
    end  %end mutation
end  %end crossover loop for all genes
```

savepop.m

```
function [] = savepop(genelength,pop,sname)

%save population
fid = fopen([sname '.dat'],'wt');
for k=1:length(pop(:,1))
    fprintf(fid,'%2.1d ',pop(k,1:genelength));
    fprintf(fid,'%9.7f ',pop(k,genelength+1));
    fprintf(fid,'%5.3f %5.3f %9.7f
%5.3f\n',pop(k,genelength+2:genelength+5));
end
disp('Pop saved')
fclose(fid);
```

getfitness.m

```
function [genome fitness] =
getfitness(genelength,xlocs,bpts,radii,genesize,numpoints,genome,fitset)

%outputs the same genome as input, except with a new fitness, pitch at
%maximum power coefficient, maximum power coefficient, and noise level.
%Also outputs the calculated fitness.

%inputs are the gene length; the x locations of the bezier control points;
%bpts, the number of points on the output airfoil; the radial locations for
%the chord and twist values; genesize, the length of one gene, 8 in this
%case; numpoints, the number of control points; the genome, and fitset
%described below

%fitset = 1 - run all xfoil normally; 2 - only check for self intersection
% 3 - do not run xfoil

%power available = 1/2*A*rho*v^3

rho = 1.246;  %sea level density
windspeed = 10; %windspeed
fitness = -100;  %fitness - initialize output values to -100 so if that value
appears in the input the user knows something did not run or perform as
expected
neweff = -100;  %wind turbine power coefficient
totnoise = -100;  %total calculated dB level
mxangle = -100;  %rotor pitch angle at maximum power coefficient
rotspeed = -100;  %rotor rotational speed at maximum power coefficient

fclose('all');

%delete previously used files to prevent rewrite issues
delete afd*.dat;
delete afpts*.dat;
delete afdata*.dat;
%calculate Reynolds numbers
% Req = inline('1.2*sqrt((omega.*chord).^2+vel^2)/1.78e-
5','omega','vel','chord');
% renums = Req(16.4*2*pi/60,13.5,Y(1,:));

%decode genome
%points = 3 columns of y-values.  Each column represents the control points
%for the bezier curve for one of the airfoils, first column is inner,
%second column is middle, third column is tip.
%chord = 8 element vector of rotor chord value at each radial location
%twist = 8 element vector of rotor twist value at each radial location
%bcurve is a 6 column matrix.  Each pair of columns are the x and y
%coordinates of the points for one of the airfoils.
[points chord twist] = decodeaf(genesize,numpoints,genome);
bcurve = bezzier(bpts,[xlocs, points]);

%check for self intersecting geometry
for i = 1:length(bcurve(1,:))/2
```

```
    %divides each airfoil up into a top and bottom curve
    midrow = (length(bcurve)+1)/2;
    bcbottom = flipud(bcurve(1:midrow,2*i));
    bctop = bcurve(midrow:length(bcurve),2*i);
    %check for self-intersection - the points are spaced the same on the
    %top and bottom, so if subtracting the bottom y-coordinate from the top
    %y-coordinate gives a negative value, the airfoil self-intersects
    if min(bctop(2:midrow-1)-bcbottom(2:midrow-1))<0
        fitness = -100;
        genome(genelength+1) = fitness;
        disp('self intersecting')
        return
    elseif genome(length(genome)-2) > 0
        fitness = genome(length(genome)-2);
    else
        fitness = 0;
    end

% check for wavy airfoils - calculates the curvature
    xvector = bcurve(1:midrow,1); %x points - y points are already stored as
btop and bcbottom
    yp = diff(bctop); %first derivative of y points
    ypp = diff(yp); %second derivative of y points
    xp = diff(xvector);
    xpp = diff(xp);
    xp(1) = []; %deletes first member of x and y vectors so that length(x) =
length(x')
    yp(1) = [];
    ktop = (xp.*ypp - yp.*xpp)./(xp.^2+yp.^2).^1.5; %curvature of top airfoil
surface

    yp = diff(bcbottom);
    ypp = diff(yp);
    xp = diff(flipud(xvector));
    xpp = diff(xp);
    xp(1) = [];
    yp(1) = [];
    kbottom = (xp.*ypp - yp.*xpp)./(xp.^2+yp.^2).^1.5; %curvature of bottom
airfoil surface

    bcount = 0;  %keeps track of changes in curvature direction for top
surface of airfoil
    tcount = 0;  %same as above for bottom surface

    %steps through the curvature vectors, counting each sign switch.
    %The number of sign switches + 1 is the number of signs of curvature
    %for the airfoil surface; if # sign changes > 1 than the airfoil is
    %invalid.
    for j = 1:length(ktop)-1
        if ktop(j)*ktop(j+1)<0
            tcount = tcount+1;
        end
        if kbottom(j)*kbottom(j+1)<0
            bcount = bcount+1;
        end
    end
```

```
    if bcount>1 || tcount>1
        fitness = -100;
        genome(genelength+1) = fitness;
        disp('wavy')
        return
    end
end  %check for self intersecting / wavy airfoils

%if fitset == 0 then code stops after checking for intersection and
%curvature without calculating aerodynamic, noise, and turbine performance
if fitset == 0
    return
end

%build rotor
%extracts root, mid and tip airfoils from bcurve
rootaf = [-bcurve(:,1)+1, bcurve(:,2)];
midaf = [-bcurve(:,3)+1, bcurve(:,4)];
tipaf = [-bcurve(:,5)+1, bcurve(:,6)];

n = 15;  %number of airfoils to be extracted from rotor shape
rootsegs = 4;  %number of segments of innermost rotor section
X(length(rootaf),n) = 0;
Y(length(rootaf),n) = 0;
Z(length(rootaf),n) = 0;
for j = 1:length(rootaf)
    X(j,:) =
interp1(1:3,[rootaf(j,1),midaf(j,1),tipaf(j,1)],linspace(1,3,n));
    Y(j,:) = [linspace(radii(1),radii(2),rootsegs)
linspace((radii(3)+radii(2))/2,radii(8),n-rootsegs)];
    Z(j,:) =
interp1(1:3,[rootaf(j,2),midaf(j,2),tipaf(j,2)],linspace(1,3,n));
end

%create smoothed chord vector, apply scale transformation to blade
%segments
smchord = spline(radii,chord);
scale = [interp1(radii,chord,linspace(radii(1),radii(2),rootsegs)),
ppval(smchord,Y(1,rootsegs+1:n))];
for j = 1:length(X(1,:))
    X2(:,j) = X(:,j)*scale(j);
    Z2(:,j) = Z(:,j)*scale(j);
end

%apply twist to rotor - create smoothed twist vector as above
smtwist = spline(radii,twist);
sectwist = ppval(smtwist,Y(1,:));
for j = 1:length(X2(1,:))
    rotpoint = [0 0];  %rotates about leading edge
    xsec = [X2(:,j)-rotpoint(1),Z2(:,j)-rotpoint(2)];
    rotmat = [cosd(sectwist(j)) -sind(sectwist(j)); sind(sectwist(j))
cosd(sectwist(j))];
    xsec = xsec*rotmat;
    X2(:,j) = xsec(:,1)+rotpoint(1);
    Z2(:,j) = xsec(:,2)+rotpoint(2);
end
```

```
%get headers and from wind file
fid = fopen('windfile.wnd','rt');
windfile = textscan(fid,'%s','delimiter','\n');
fclose(fid);
windfile = windfile{1};
windfile{4} = ['0   ' num2str(windspeed) '     0.0     0.0     0.0     0.0
0.0     0.0'];
windfile{5} = ['999.9   ' num2str(windspeed) '     0.0     0.0     0.0     0.0
0.0     0.0'];

%write wind file
fid = fopen('windfile.wnd','wt');
for q = 1:length(windfile)
    fprintf(fid,'%s\n',windfile{q});
end
fclose(fid);

%gets headers for airfoil files for FAST
fid = fopen('aerofile.dat','rt');
aeheader = textscan(fid,'%s','delimiter','\n');
fclose(fid);
aeheader = aeheader{1};

%get headers for primary AD input file
fid = fopen('primary_AD2.ipt','rt');
ADheader = textscan(fid,'%s','delimiter','\n');
fclose(fid);
ADheader = ADheader{1};

%calculate endpoints of blade segments and locations of the center of
%each segment
numel = length(Y(1,:));
r = Y(1,:);
endpts(1) = r(1)-(r(2)-r(1))/2;
endpts(2:numel) = (r(1:numel-1)+r(2:numel))/2;
endpts(numel+1) = r(numel)+(r(numel)-r(numel-1))/2;
endpts = round2(endpts,.001);
widths = round2(endpts(2:length(endpts))-endpts(1:length(endpts)-1),.01);
%segment widths (spans)
r = (endpts(1:numel)+endpts(2:numel+1))/2;  %segment centers


%write rotor twist, segment spans, chord, and airfoil file number to
%aerodynamic input file
fid = fopen('primary_AD.ipt','wt');
fprintf(fid,'%s\n',ADheader{:});
for i = 1:length(Y(1,:))
    fprintf(fid,'%.4f\t%.2f\t%.4f\t%.4f\t%d\t%s\n',r(i),...
        -sectwist(i),widths(i),scale(i),i,'NOPRINT');
end
fclose(fid);

%writes coordinate files for all 15 airfoils from the rotor definition
for i = 1:15
```

```
    fid  = fopen(['afpts' num2str(i) '.dat'],'wt');
    for j = 1:length(X(:,1))
        fprintf(fid,'%f',X(j,i));  %each column of X contains the x
coordinates for that airfoil
        fprintf(fid,' %f\n',Z(j,i));  %each column of Z contains the y
coordinates for that airfoil
    end
end

%runs all .bat files for running XFOIL
try
    tic
    if fitset == 1  %if fitset ==1, runs XFOIL, FAST, and NAFNoise
        disp('Running XFOIL')

        %runs XFOIL for each airfoil
        for i = 1:numel
            t1 = toc;

            %start XFOIL, wait for 2 seconds before checking whether to
            %kill the operation
            dos(['start /min xfoilrun' num2str(i) '.bat >NUL']); %add >NUL to
eliminate output
            pause(2)
            chk = 1;
            clvalchk2 = 0;
            while chk == 1
                %attempt to open XFOIL output file
                fid = fopen(['afd' num2str(i) '.dat'],'rt');
                fseek(fid,429,'bof');

                %saves the aerodynamic data created by XFOIL as afdatai,
                %where i is the airfoil number from inner to outer
                name{i} = ['afdata' num2str(i)];
                func = [name{i} ' = fscanf(fid,''%f'',[7 inf]);'];
                func2 = [name{i} ' = ' name{i} ''';'];
                eval(func);
                eval(func2);
                fclose(fid);
                eval(['afdata = ' name{i} ';']);

                %ends XFOIL operation if Cl is decreasing after entering
                %positive lift, or if XFOIL has been running longer than 20
                %seconds, or if it has gone past around 30 degrees AOA
                clvalchk = abs(afdata(length(afdata(:,1)),2));
                runtime = toc - t1;
                if length(afdata(:,1))>58   &&   max(afdata(:,2)) - clvalchk
> .01
                    %                                          disp('Killed:
decreasing Cls')
                    dos('taskkill /f /im xfoil.exe >NUL');
                    chk = 0;
                elseif afdata(length(afdata(:,1))) >= 44
                    %                                          disp('Killed:
past 30 degrees')
                    dos('taskkill /f /im xfoil.exe >NUL');
```

```
                    chk = 0;
                elseif runtime > 20
                    %                                           disp('Killed:
runtime > 15 sec')
                    dos('taskkill /f /im xfoil.exe >NUL');
                    chk = 0;
                else
                    clvalchk2 = clvalchk;
                    pause(.3)
                end
            end  %done running XFOIL

            %READS XFOIL OUTPUT
            fid = fopen(['afd' num2str(i) '.dat'],'rt');
            fseek(fid,429,'bof');
            name{i} = ['afdata' num2str(i)];
            func = [name{i} ' = fscanf(fid,''%f'',[7 inf]);'];
            func2 = [name{i} ' = ' name{i} ''';'];
            eval(func);
            eval(func2);
            fclose(fid);
            eval(['afdata = ' name{i} ';']);

            %find Cl peak, make sure it isn't last member
            afdata = sortrows(afdata,1);
            if length(afdata(:,1)) > 25
                [pks pksindex] = findpeaks(afdata(23:length(afdata(:,1)),2));
                if isempty(pks)
                    [pks pksindex] = max(afdata(23:length(afdata(:,1)),2));
                end
            end

            %check for NaNs, return fitness = -100 if any occur in data, or
if Cl doesn't
            %have stall region
            if sum(sum(isnan(afdata))) ~= 0 || (isempty(pks) && pksinex ==
length(afdata(:,1))) ...
                    || length(afdata(:,1))<41
                fitness = -100;
                genome(genelength+1) = fitness;
                afd = afdata;
                return
            end
        end
    end
catch
    %if an error occurs running XFOIL
    disp('XFOIL did not execute properly')
    fitness = -100;
    genome(genelength+1) = fitness;
    return
end
toc
afd = afdata;

try
```

```
%processes airfoil data
%saves airfoil data in AERODYN airfoil data file format
%the code below is tailored to meet the input parameters required in the
%airfoil data files used by AERODYN in FAST
for i = 1:numel
    eval(['afdata = afdata' num2str(i) ';']);

    %create normal force coefficients
    afdata(:,8) =
afdata(:,2).*cosd(afdata(:,1))+afdata(:,3).*sind(afdata(:,1));

    %sort afdata
    afdata(2:length(afdata(:,1)),:) =
sortrows(afdata(2:length(afdata(:,1)),:),1);

    %find first positive lift
    zrow = 2;
    while afdata(zrow,2)<0
        zrow = zrow+1;
    end

    %assume negative stall occurs at the most negative AOA for which there
    %is data.  This is because XFOIL does not seem to do a good job finding
    %negative stall for cambered airfoils.
    minrow = 2;

    %find first maximum cl for positive stall
    [clpks pksindex] = findpeaks(afdata(38:length(afdata(:,1)),2));
    %if max cl occurs twice fix it
    if isempty(clpks)
        [clpks pksindex] = max(afdata(38:length(afdata(:,1)),2));
    end
    mxcl = clpks(1);
    mxclrow = pksindex(1)+37;

    %the slope of the normal force coefficients near zero lift is required
    %by AERODYN in FAST
    cnfit = polyfit(afdata(zrow-2:zrow+3,1),afdata(zrow-2:zrow+3,8),1);
    cnslope = cnfit(1)*180/pi;

    %find max and min Cl and alfa at stall
    clstall = afdata(mxclrow,2);
    astall = afdata(mxclrow,1);
    cdstall = afdata(mxclrow,3);
    anegstall = afdata(minrow,1);
    cdnegstall = afdata(minrow,3);
    clnegstall = afdata(minrow,2);

    %ROTOR PARAMS
    radii = [4.40 8.54 13.34 18.14 22.94 27.74 32.54 37.34]';
    chord = [1.51 2.77 2.48 2.19 1.90 1.61 1.31 1.02]';

    %calculations for Cl and Cd in the positive and negative stall regions
    %are shown below.  See Tangler in list of references for details
    AR = max(radii)/interp1(radii,chord,.8*max(radii));
```

```
    %calculate stall approximation
    alfa2 = [astall+1:2:30 35 40:10:90];
    alfa3 = [anegstall-1:-1:-10 -12:-2:-20 -25 -30:-10:-90];
    alfa = [fliplr(alfa3) afdata(minrow:mxclrow,1)' alfa2]';
    cdmax = 1.11+0.01*AR;
    B1 = cdmax;
    B2 = (cdstall - cdstall*sind(astall).^2)/cosd(astall);
    cd = B1*sind(alfa2)+B2*cosd(alfa2); %alfa > stall
    A1 = B1/2;
    A2 = (clstall -
cdmax*sind(astall)*cosd(astall))*sind(astall)/cosd(astall)^2;
    cl = A1*sind(2*alfa2)+A2*cosd(alfa2).^2./sind(alfa2); %alfa > stall

    %calc for neg stall
    B2 = cdnegstall - cdnegstall*sind(anegstall).^2/cosd(astall);
    cd2 = abs(fliplr(B1*sind(alfa3)+B2*cosd(alfa3))); %alfa < negstall
    A2 = (clnegstall -
cdmax*sind(anegstall)*cosd(anegstall))*sind(anegstall)/cosd(anegstall)^2;
    cl2 = fliplr(A1*sind(2*alfa3)+A2*cosd(alfa3).^2./sind(alfa3)); %alfa <
negstall
    clt = [cl2 afdata(minrow:mxclrow,2)' cl]';
    cdt = [cd2 afdata(minrow:mxclrow,3)' cd]';

    %modify the first several rows of information for the current airfoil
    %data file
    aeheader{5} = [num2str(afdata(mxclrow,1)) '  stall angle'];
    aeheader{9} = [num2str(-cnfit(2)/cnfit(1)) '  AOA zero cn linear fit'];
    aeheader{10} = [num2str(cnslope) '  Cn slope zero lift'];
    aeheader{11} = [num2str(afdata(mxclrow-2,8)) '  Cn pos stall'];
    aeheader{12} = [num2str(afdata(minrow,8)) '  Cn neg stall'];
    [mincd mincdrow] = min(afdata(:,3));
    aeheader{13} = [num2str(afdata(mincdrow,1)) '  AOA min Cd'];
    aeheader{14} = [num2str(afdata(1,3)) '  zero lift drag'];

    %save the lift and drag coefficients to the airfoil data file for FAST
operation
    fid = fopen(['afdata' num2str(i) '.dat'],'wt');
    fprintf(fid,'%s\n',aeheader{:});
    for jline = 1:length(cdt)
        fprintf(fid,'%.2f\t%.4f\t%.4f\n',alfa(jline),clt(jline),cdt(jline));
    end
    fclose(fid);
end
catch
    %check for errors processing airfoil data
    disp('Could not process XFOIL data properly')
    fitness = -100;
    genome(genelength+1) = fitness;
    return
end


try  %run FAST
    disp('Running FAST')
    %READ FAST INPUT FILE
```

```
    fid = fopen('primary.fst','rt');
    fst = textscan(fid,'%s',191,'delimiter','\n');
    fclose(fid);
    fst = fst{1};
    rotspeed = 16;  %initial rotor speed - guess what will be close to the
idea speed to minimize iterations
    direc = 1;  %direction of optimization, positive or negative
    int = 1;  %interval size

    %write rotspeed to FAST input file
    %neweff is the max power coefficient achieved
    %effange is the pitch angle at which the max power coefficient is
    %achieved
    [neweff, effangle] = runfast(rho,fst,rotspeed,windspeed,radii);
    oldeff = neweff; %the change in efficiency will be found from neweff-
oldeff

    %the code below runs FAST to find the optimal RPM to within a given
    %value.  The wind turbine is run across a range of pitch angles each
    %time FAST is run
    while int > .1  %precision to which RPM is optimized
        effect = 1;  %effect measures whether the change in performance at
the new RPM; here it is initialized to 1

        %the optimization scheme first obtains a performance value, then
        %goes to the next RPM.  It repeats this until the performance
        %decreases, in which case the direction of RPM change is reversed,
        %and the interval is halved.  This repeats until the interval
        %reaches the desired value.  This scheme is simple to code but
        %results in some RPM values being tested more than once.
        while effect > 0
            [neweff, effangle] =
runfast(rho,fst,rotspeed+int*direc,windspeed,radii);
            effect = neweff - oldeff;
            if effect > 0
                rotspeed = rotspeed+int*direc; %if RPM improves, calculates
next RPM
                oldeff = neweff;
                mxangle = effangle;
            else
                direc = -direc;
                effect = 1;
                while effect > 0
                    [neweff, effangle] =
runfast(rho,fst,rotspeed+int*direc,windspeed,radii);
                    effect = neweff - oldeff;
                    if effect > 0
                        rotspeed = rotspeed+int*direc;
                        oldeff = neweff;
                        mxangle = effangle;
                    else
                        int = int/2;
                    end
                end
            end
        end
```

```
    end
    %end optimize rpm and pitch angle


    fitness = neweff;
    %summarize
    fprintf('Best overall efficiency achieved is %.3f, at %.2f degrees, %.2f
rpm.\n',neweff,effangle,rotspeed);

    %apply fitness correction to rotor - reduce fitness if ideal RPM is
    %over 21, or under 10
    if rotspeed > 21
        fitness = fitness * 1.2^-(rotspeed - 21);
    elseif rotspeed < 10
        fitness = fitness * 1.9^-(10 - rotspeed);
    end

catch
    %check for errors in running FAST
    disp('FAST did not run correctly')
    fitness = -100;
    genome(genelength+1) = fitness;
    return
end

%calculation fitness adjustment favoring less jumpy bezzier control point
%distributions
jumpcor = 0;
for i = 1:3
    avgjump = sum(abs(points(1:numpoints+1,i)-
points(2:numpoints+2,i)))/(numpoints+1);
    jumpcor = jumpcor +  (1-1.4*avgjump/(avgjump+20));   %correction value
for chopiness of airfoil points
end
jumpcor = jumpcor / 3;
fitness = fitness * jumpcor;

try
%modify NAFNOISE file for each rotor segment, run one at a time
disp('Calculating noise')
totnoise = 0; %total noise in dB
for i = 1:15
    %open NAFNoise input file, modify values to match those for the given
    %airfoil
    fid = fopen(['nafnoise2' '.ipt'],'rt');
    fcont = textscan(fid, '%s','delimiter','\n');
    fclose('all');
    fcont = fcont{1};

    fcont{25} = ['afptsref' num2str(i) '.dat'];
    fcont{15} = [num2str(scale(i)) ' \t chord length'];
    fcont{16} = [num2str(widths(i)) ' \t airfoil span'];
    fcont{17} = [num2str((windspeed^2+(r(i)*rotspeed*2*pi/60)^2)^.5)...
        ' \t freestream velocity'];
    fcont{18} = [num2str(atand(windspeed/(r(i)*rotspeed*2*pi/60))- ...
```

```
    (-sectwist(i)+mxangle)) ' \t angle of attack'];

%find values for law of cosines for calculating solid angle - this is
%the angle formed by the last upper and lower segments at the trailing
%edge
fid = fopen(['afpts' num2str(i) '.dat'],'rt');
afpts = fscanf(fid,'%f',[2 inf]);
afpts = afpts';
p1 = afpts(2,:);
p2 = afpts(1,:);
p3 = afpts(length(afpts)-1,:);
c = sqrt((p1(1)-p3(1))^2+(p1(2)-p3(2))^2);
b = sqrt((p2(1)-p1(1))^2+(p2(2)-p1(2))^2);
a = sqrt((p3(1)-p2(1))^2+(p3(2)-p2(2))^2);
solidangle = acosd((c^2-a^2-b^2)/(-2*a*b));
fcont{20} = [num2str(solidangle) ' \t trailing edge solid angle'];

%print nafnoise2.ipt
fid = fopen('nafnoise2.ipt','wt');
for j = 1:length(fcont)-1
    fprintf(fid,fcont{j});
    fprintf(fid,'\n');
end
fprintf(fid,fcont{j+1});
fclose('all');

%run nafnoise - check for output before continuing with code
tic
dos('start /min nafnoiserun.bat >NUL');
pause(2);
nafchk = 0;
while nafchk == 0
    try
        nafchk = 1;
        fid = fopen('nafnoise2.out','rt');
        fseek(fid,615,'bof');
        noise = fscanf(fid,'%f',[8 inf]);
        noise = noise';
        fclose('all');
        if round(noise(length(noise(:,1)),1)) ~= 20000
            nafchk = 0;
        end
    catch
        nafchk = 0;
        pause(.2);
    end
    if toc > 15
        fitness = -100;
        genome(genelength+1) = fitness;
        disp('nafnoise did not run correctly');
        return
    end
end

%sum total of dB values for each frequency range
bladenoise = 10*log10(sum(10.^(noise(:,8)/10)));
```

```
    %add bladenoise to the running total of total rotor noise, totnoise
    totnoise = 10*log10(sum(10.^([bladenoise,totnoise]/10)));
end

%adjust fitness for noise level
noiseadd = -.005*totnoise+82*.005;
fitness = fitness + noiseadd;

%if totnoise is NaN, NAFNoise could not compute the noise for the given
%airfoil for some reason, possibly odd geometry.  Return negative fitness
if isnan(totnoise)
    fitness = -100;
    genome(genelength+1) = fitness;
    disp('Totnoise NaN')
    pause
end

catch
    %check for errors in running NAFNoise
    disp('NAFNoise could not complete properly')
    fitness = -100;
    genome(genelength+1) = fitness;
    return
end
fclose('all');

%update the genome to include the new fitness, pitch angle at max power
%coefficient, RPM at max power coefficient, max power coefficient, and
%noise level in dB
genome = [genome(1:genelength) fitness mxangle rotspeed neweff totnoise];
```

decodeaf.m

```
%input are the number of points of the bezzier curve, assuming that the y
%coordinates are the only part that's being modified, and the airfoil
%genome

function [afpoints chord twist] = decodeaf(genesize,numpoints,afgenome)
afpoints(numpoints+2,1) = 0;

%extract airfoil control points from genome
for i = 1:3
    for m = 1:numpoints
        % decode into binomial form
        afpoints(m+1,i) = -.3 + .002353*bin2dec(num2str(afgenome((i-
1)*genesize*numpoints+(m-1)*genesize+1:...
            (i-1)*genesize*numpoints+(m-1)*genesize+genesize)));
    end
end

genelength = genesize*numpoints*3;

%extract chord values from genome
%the possible chord values are [0.8 3.5]m in increments of 2.7m/255
for i = 1:8
    chord(i,1) = 0.8+2.7/255*bin2dec(num2str(afgenome(genelength+8*(i-
1)+1:genelength+8*i)));
end

%extract twist values from genome
%the possible twist values are [-15 70]deg in incremements of (85 deg)/255
for i = 1:8
    twist(i,1) = -15+85/255*bin2dec(num2str(afgenome(genelength+64+8*(i-
1)+1:genelength+64+8*i)));
end

%reverses twist - this is due to how MATLAB plots the rotor versus how the
%twist values are interpreted in FAST
twist = -twist;
```

runfast.m

```
%This file runs FAST with the given parameters
%
%
%
%INPUTS:  rho (density), fst (FAST input file), rotspeed (rotor RPM),
%windspeed, radii (vector of radial locations)
%OUTPUTS:  neweff (maximum achieved efficiency), effangle (pitch angle at
%which maximum efficiency is achieved)



function [neweff, effangle] = runfast(rho,fst,rotspeed,windspeed,radii)
fclose('all');
delete primary.out; %deletes old FAST input file
maxtime = 10;  %max simulation time
minangle = -30;  %simulation starting pitch angle
maxangle = 30;  %simulation ending pitch angle
anglerange = maxangle - minangle;
tstep = .004;  %simulation time step
anglestep = 5*anglerange/(maxtime/tstep);  %calculates angle increment
between each time step

%rewrite FAST input file
fst{73} = [num2str(rotspeed) '     RotSpeed']; %add new RPM value
fst{10} = [num2str(maxtime) ' \t total simulation time']; %change simulation
time
for fstrow = 43:45
    fst{fstrow} = [num2str(maxtime) ' \t time to end pitch maneuver'];
end

%write starting pitch angle
for fstrow = 46:48
    fst{fstrow} = [num2str(minangle) ' \t minimum angle'];
end

%write ending pitch angle
for fstrow = 49:51
    fst{fstrow} = [num2str(maxangle) ' \t maximum angle'];
end

%save FAST input file
fid = fopen('primary.fst','wt');
fprintf(fid,'%s\n',fst{:});
fclose(fid);

%run FAST
dos('start /min runfast.bat >NUL');
tic;
pause(1) %wait 1 second for FAST to run
fid = 1;

%this loop waits for FAST to run in 0.2 second intervals, each time
```

```
%checking if primary.out is found, which indicates FAST is done running.
%if it is, it breaks the loop and moves on to the next section of code
while fid ~= -1
    pause(.2)
    fid = fopen('primary.out','rt');
    if fid ~= -1
        fseek(fid,291,'bof');
        fastout = fscanf(fid, '%f', [2 inf])';
        fclose(fid);
        if fastout(length(fastout(:,1)),1) >= maxtime
            break  %stops loop if primary.out is found
        end
    end
    if toc > 10
        dos('taskkill /f /im fast.exe >NUL');
    end
end

%read FAST output
fid = fopen('primary.out','rt');
fseek(fid,291,'bof');
fastout = fscanf(fid, '%f', [2 inf])';
fclose(fid);

%analyze output
%convert rotor torque to power coefficient
fastout(:,3) = 1000*fastout(:,2)*rotspeed*2*pi/60 /
(1/2*pi*max(radii)^2*rho*windspeed^3);

%add column of pitch angles at each time step
fastout(:,4) = (fastout(:,1))*anglerange/maxtime+minangle;

%find maximum efficiency and corresponding angle
[neweff optrow] = max(fastout(:,3));
effangle = fastout(optrow,4);

%in this section, the maximum efficiency is taken as the average efficiency
%within +/-1 degree of the maximum efficiency.  This prevents solutions
%with an extremely narrow peak efficiency from getting high fitness
%ratings.
arange = round(1/anglestep);  %number of rows to get 1 degree of twist in
fastout
effrange = trapz(fastout(optrow-arange:optrow+arange,4), ...
    fastout(optrow-arange:optrow+arange,3));
neweff = effrange / (fastout(optrow+arange,4)-fastout(optrow-arange,4));
```