1-1-2011

Numerical Modeling and Optimization of Power Generation from Shrouded Wind Turbines

Tudor Foote

WASHINGTON UNIVERSITY IN ST. LOUIS
School of Engineering and Applied Science
Department of Mechanical Engineering and Materials Science

Thesis Examination Committee:
Ramesh K. Agarwal, Chair
David Peters
Kenneth Jerina

NUMERICAL MODELING AND OPTIMIZATION OF POWER GENERATION FROM SHROUDED WIND TURBINES

by

Tudor F. Foote

A thesis presented to the School of Engineering
of Washington University in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

December 2011
Saint Louis, Missouri

ABSTRACT OF THE THESIS

Numerical Modeling and Optimization of Power Generation from Shrouded Wind Turbines

by

Tudor F. Foote

Master of Science in Mechanical Engineering

Washington University in St. Louis, 2011

Research Advisor: Professor Ramesh K. Agarwal

In recent years, it has been theoretically suggested by some researchers that the power coefficient of a wind turbine can be increased beyond the Betz limit for a bare turbine by enclosing the turbine with a duct. In this thesis, this potential for increases in power generation by adding a shroud around a turbine is investigated using numerical flow modeling. Two configurations for ducted turbines are considered for numerical simulations and optimization. The first configuration considers shrouds for standard horizontal axis wind turbines. Several turbine diameters, wind speeds, and shroud sizes are considered. The results show that shrouding can almost double the power that would be generated by a bare turbine. The second configuration considers the potential of converting abandoned or unused farm silos into solar chimneys with low cost shape modification of the chimney to further augment the power generation. The most effective, simple and low cost shape modification is found to be a diffuser added to the top of the silo, which can increase the power output by nearly 50%. Increasing the buoyancy effect by heating the air at the base of the silo can further increase the power output by a significant amount, as much as 50%. It should be noted that in this configuration, the effect of cross flow wind at the exit of the diffuser can have a negative effect on the generated wind power; however, it is surmised to be small. Both these configurations are analyzed by employing computational fluid dynamics flow solver, which solves the Reynolds-Averaged Navier-Stokes equations, in conjunction with a two equation k-epsilon turbulence model. The turbine is modeled as an

actuator disk neglecting the rotational effects. The diffuser shapes in both the configurations are optimized using a genetic algorithm. The computations show that the shrouded turbines can generate greater power than that generated by the bare turbines and should be considered for small and medium size turbines. Further investigation is needed in the overall economic benefit considering the initial investment, maintenance and life cycle costs. The technical feasibility of shrouding a turbine and the structural integrity of a shrouded turbine are also major considerations.

Acknowledgments

My infinite gratitude goes to my parents, George and Lisa, for ensuring that all things are possible in my life. Their support means the world to me, and is the foundation of all I build.

Thanks to my friends and members of the CFD lab who have patiently listened and given their thoughts and advice as I worked through the finer points of this project.

Thanks also to my thesis committee members, Dr. David Peters and Dr. Kenneth Jerina, for helping me along the way.

Special thanks go to my advisor, Dr. Ramesh Agarwal, for opening doors for me, and providing consistent guidance and understanding as we worked together these past years. Working with him has been the highlight of my time at Washington University.

Tudor F. Foote

Washington University in St. Louis
December 2011

Contents

List of Tables

List of Figures

Chapter 1

1. Introduction

1.1 Motivation and Objective

The motivation for this work stems from the recent work reported in the literature that has shown that shrouding of wind turbines has the potential of increasing their power production. In the early twentieth century, Betz showed that the maximum power production efficiency of a horizontal axis wind turbine (HAWT) cannot exceed 59.3% of the kinetic energy available in the wind facing the turbine. This maximum possible value of the power coefficient is known as the "Betz limit." It has been shown by Werle and Presz [1] and Hansen et al. [2] among others that enclosing the turbine with a shroud can increase the turbine's efficiency beyond the Betz limit. The goal of this project is to numerically model the performance of shrouded wind turbines and optimize their power production by shape optimization of the shroud using a genetic algorithm.

1.2 Previous Work

Although windmills have existed for centuries, the understanding of their aerodynamics came about only at the beginning of the 20^{th} century. In 1919, Betz developed a model of a wind turbine and determined its performance by applying the principles of fluid conservation. He determined that the maximum possible power that can be extracted

from a fluid stream facing the turbine is 59.3% which is now known in the literature as the Betz limit. Currently, there are many excellent books that discuss and explain the aerodynamics, controls and performance of wind turbines.

Wind turbine development has continued for almost a century. Now, with increased emphasis of renewable energy, it is quickly becoming a highly developed worldwide industry with a rapidly increasing number of turbine installations around the globe. Significant gains have been made in efficiency; modern horizontal axis wind turbines are almost reaching the theoretical limit. Recent researchers are trying to develop alternative configurations to bare HAWT to increase the theoretical limit and thereby bring down the cost of wind energy. One such recent proposal to increase the aerodynamic efficiency of a wind turbine by has been to add a shroud or diffuser surrounding the turbine. Among others, Werle and Presz [1] and Hansen et al. [2] have performed the theoretical analysis of shrouded wind turbines to demonstrate that indeed the power generated by a shrouded HAWT can be greater than that of a bare HAWT and that its power coefficient can exceed the Betz limit. This concept has also been demonstrated in actual installation by the company FloDesign Inc., based in Wilbraham, MA. The data of Werle and Presz [1] is employed in this paper for comparison. It is important to note that there is some debate in the recent literature as to whether the shrouded turbines do in fact surpass the Betz limit. Widnall [3] has argued that they do not if the power coefficient is determined by using the diffuser exit area of the shrouded turbine instead of the turbine area (or throat area). Nevertheless, the total power generated by the shrouded turbine is greater than that of a bare turbine.

1.3 Theoretical Basis and Governing Equations

1.3.1 Betz's Model of a Wind Turbine

The classical formulation of a wind turbine flow field is the Betz model. This model considers an inviscid, incompressible flow past a turbine in a stream tube and applies mass flow continuity and the Bernoulli equation. The inlet flow upstream of the turbine exhibits free stream properties of the wind such as velocity and pressure, and the exit flow downstream of the turbine is assumed to be at atmospheric pressure. From continuity, the velocity in the immediate vicinity of the turbine on both the upstream and downstream sides must be equal, so the turbine must be considered only as a pressure discontinuity as it extracts energy. Then the Bernoulli equation can be applied both upstream of the turbine between the inlet flow and the upstream face of the turbine, and downstream of the turbine between the downstream face of the turbine and the exit flow. This application of the Bernoulli equation can determine the pressure differential across the turbine, which determines the generated power. It can then be shown that the maximum possible power that can be extracted from the stream is 16/27 or 59.3% of the available kinetic energy in the wind stream. Hypothetically, if all the energy was removed, the velocity behind the turbine would become zero, i.e. the flow would stop. This cannot happen in a steady state system; it is a violation of continuity. The power coefficient of 16/27 is called Betz's limit.

1.3.2 Actuator Disk Model

In this work, the turbine is modeled as an actuator disk, a commonly used simplification in the aerodynamic analysis of turbines. The actuator disk models the turbine as a pressure discontinuity in the flow; the pressure drop across the actuator disk represents the energy extracted across the plane of the turbine. This modeling concept is due to the Betz bare (unshrouded) horizontal axis wind turbine formulation mentioned in the previous section. Mikkelsen [4] performs an excellent analysis of this model for wind turbines.

The actuator disk is an idealized model that does not account for any drag or losses at the turbine, or swirl of the turbine wake due to rotational effects. The wake rotation is the single largest source of error in this model. To determine the efficiency of a HAWT, taking into account its rotational speed, the number of blades and blade geometry, a number of empirical relations have been proposed by many researchers in the literature. They are based of combining theoretical considerations with experimental data. One such empirical formula is given in the book by Manwell et al. [5] and presented here as Equation 1.1. It fits the experimental data reasonably well. In equation 1.1, where λ is the tip speed ratio (rotational speed of the turbine/wind speed), n is the number of blades and Cl/Cd is the average lift to drag ratio of the turbine blade. It is important to note from equation 1.1 that the correction to Cp due to n, λ, and Cl/Cd is multiplicative to 16/27 (the ideal Betz limit).

$$(C_{p\,max})_{bare} = \frac{16}{27}\lambda\left[\lambda + \frac{1.32 + \left(\frac{\lambda-8}{20}\right)^2}{n^{\frac{2}{3}}}\right]^{-1} - \frac{0.578\lambda^2}{\frac{C_l}{C_d}\left(\lambda + \frac{1}{2n}\right)} \qquad (1.1)$$

The majority of the currently installed HAWTs have 3 blades and blade Cl/Cd of 25. Then the maximum Cp of these turbines according to Equation 1.1, with a tip speed ratio between 3 and 4, is approximately 0.42. This is 71% of the Betz limit of 0.593. This equation provides a practically useful formula to account for the decrease in power coefficient from the one computed by the actuator disk model. It is suggested (not proven) therefore that the actuator disk model based results in this thesis can be reduced by 29% to provide a more reasonable estimate of the actual power production of a HAWT. This will not provide a precise estimate, however it will provide a conservative downward estimate of the actual Cp of a HAWT.

Chapter 2

2. Modeling of a Shrouded Horizontal Axis Wind Turbine and Shroud Shape Optimization

This chapter describes the modeling of a shrouded horizontal axis wind turbine (HAWT) and shape optimization of its shroud. The flow fields of the shrouded turbine were calculated using a CFD solver described in section 2.1, the turbine was modeled as an actuator disk inside the shroud. The shroud shape was optimized using a Genetic Algorithm (GA) in order to optimize the generated power as described in section 2.3. Results are given in section 2.4, including the comparisons with previous work whenever possible.

2.1 CFD Modeling

The power output of a horizontal axis wind turbine (HAWT) within a shroud was modeled by employing an actuator disk model in the commercial computational fluid dynamics (CFD) solver FLUENT [6]. The geometry was generated around a fixed radius actuator disk; a contoured wall was created that represented the shroud around the turbine. An axi-symmetric geometry for both the actuator disk and the shroud was

considered. Reynolds-Averaged Navier-Stokes (RANS) equations in conjunction with a two equation realizable k – ε model were employed to calculate the turbulent flow.

The geometric models were created in the ANSYS Inc. software "GAMBIT" [7]. A structured mesh was generated in the axi-symmetric models, and the actuator disk was defined as a pressure discontinuity. The pressure drop was calculated for each cell boundary on the actuator disk and was derived from first principles for a shrouded turbine as described below. As with the Betz formulation, the inlet and outlet pressures were assumed to be atmospheric. A schematic of a typical geometric model is shown in Figure 2.1. In Figure 2.1, stations 2 and 3 refer to the upstream and downstream sides of the actuator disk respectively.

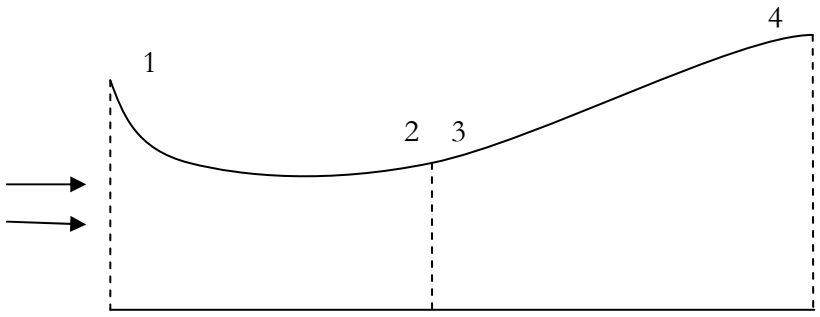

Figure 2.1 Schematic of a shrouded wind turbine modeled as an actuator disk.

Denoting the velocity by "V" and the area by "A", the application of the continuity equation to the geometry shown in Figure 2.1 gives:

$$A_1V_1 = A_2V_2 = A_3V_3 = A_4V_4$$

The application of the Bernoulli equation between stations 1 and 2 and between 3 and 4 gives:

$$p_1 + \tfrac{1}{2}\rho V_1^2 = p_2 + \tfrac{1}{2}\rho V_2^2$$

$$p_3 + \tfrac{1}{2}\rho V_3^2 = p_4 + \tfrac{1}{2}\rho V_4^2$$

Using the above two equations, obtain:

$$p_3 - p_2 = \tfrac{1}{2}\rho[V_4^2 - V_1^2]$$

Or:

$$p_3 - p_2 = \Delta P = \tfrac{1}{2}\rho\left[\left(\frac{A_2}{A_4}\right)^2 - \left(\frac{A_2}{A_1}\right)^2\right]V_2^2 \qquad (2.1)$$

Note that p1 = p4 = 0 as they are assumed to be at atmospheric gage pressure.

Equation 2.1 gives the ideal pressure drop across the turbine based on the actuator disk model. Note that the velocity V_2 is the average local velocity at the turbine face.

2.2 Power Coefficient

In optimization of shroud shape for maximal power extraction from a shrouded turbine, the power coefficient C_p was the parameter that was maximized using the GA. The area-averaged wind velocity at the actuator disk V_t was related to the free stream wind velocity V_o to obtain:

$$C_p = \frac{\Delta P A_t V_t}{\frac{\rho}{2}A_t V_o^3} = \left[\left(\frac{A_2}{A_4}\right)^2 - \left(\frac{A_2}{A_1}\right)^2\right]\frac{V_t^3}{V_o^3}\, C_p = \frac{\frac{\rho}{2}A_t V_t^3}{\frac{\rho}{2}A_t V_o^3} \qquad (2.2)$$

It should be noted that in equation 2.2, the actuator disk area A_t has been employed in the calculation of the power coefficient Cp. This expression can give the value of Cp>0.593 (Betz limit) for a shrouded turbine. Recently, Widnall [3] has pointed out that if instead of A_t, the exit area of the shroud is employed, Cp will not exceed the Betz limit. This issue is addressed later in results section 2.4.7.

2.3 The Genetic Algorithm

2.3.1 Overview

One of the objectives of this project is to find the shroud shape that generates the optimal power production with given constraints. A Genetic Algorithm (GA) is employed to solve this shape optimization problem. The basic concept of a GA is to simulate a process of evolution where many individuals compete to procreate. Each individual is evaluated for fitness. The better ones are allowed to "breed" a new generation in a manner likely to produce individuals of greater fitness, and these replace the less fit individuals from the previous generation. The new generation is then evaluated, and the process repeats. A random mutation process is also included to allow for more significant changes in a single generation, as well as the possibility of creating a radically different individual with better fitness that would not have been found in the normal "breeding" process. The principles of a GA have been explained in greater detail elsewhere [8-10].

The Genetic Algorithm is a piece of software that holds and processes the information about the individuals. It cannot evaluate the fitness of any individual however, so it calls the flow modeling software packages and collects the results. For the purposes of this research, the GA software is written in the JAVA programming language. Compared with C and C++, JAVA has a simpler object-oriented model and fewer low-level facilities. It has the advantage of being general-purpose, concurrent, class-based, and object-oriented. JAVA has been specifically designed to have as few implementation dependencies as possible which render the programming applications to behave as "write once and run anywhere" [11, 12].

2.3.2 Shroud Representation by a Bezier Curve

Each individual shroud shape is represented by a Bezier curve. Pierre Bezier developed the parametric curve in 1962 to describe the complex contours of automobile body shapes in a finite manner. It employs a series of control points that are related by a parametric equation, equation 2.3, where *n* is the number of control points *P*.

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i, \qquad t \in [0,1] \tag{2.3}$$

In this work, seven control points, each with radial and axial coordinates, were used to describe the shroud shapes. The control points for the HAWT shroud were defined such that they were axially ordered with the middle control point within 1 axial foot of the turbine position and three points on either side. For the airfoil shapes generated in this work, the complexity of the curves on either side of the turbine was sufficient to describe the optimized shapes. Limits were set for the maximum dimensions of the control points from practical considerations, in particular to prevent the shroud from becoming very large. Figure 2.2 shows an example curve (Case 1a in Table 2.1 in results section 2.4) and its corresponding control points. Note that the control point on the far right reached both the maximum radius of 8 and the maximum axial distance from the turbine plane of 5, as set by the user prescribed limits.

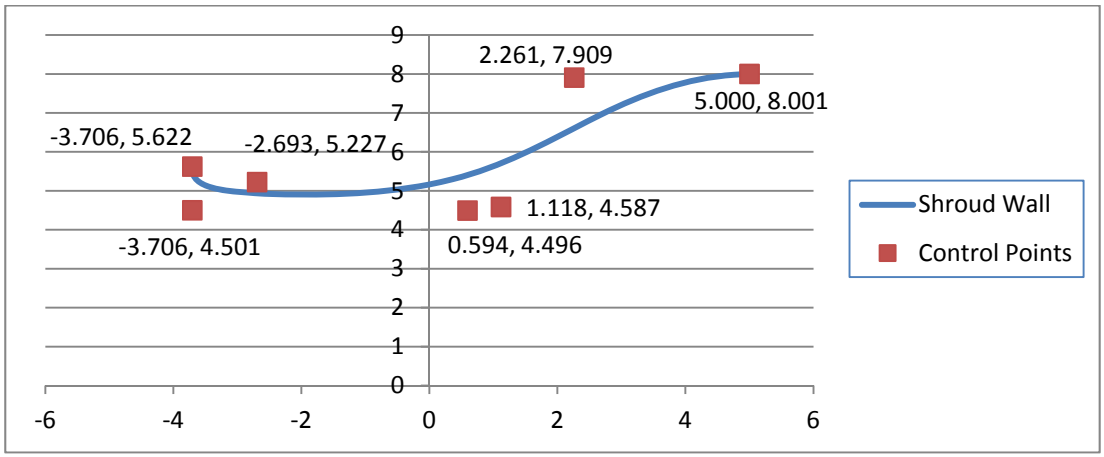

Figure 2.2 Example of an optimized shroud shape with control points (Case 1a in Table 2.1).

2.3.3 CFD Evaluation of an Individual Shroud Shape

Figure 2.3 shows the schematics of the optimization process illustrating the interaction between the GA, the mesh generator, and CFD flow solver.

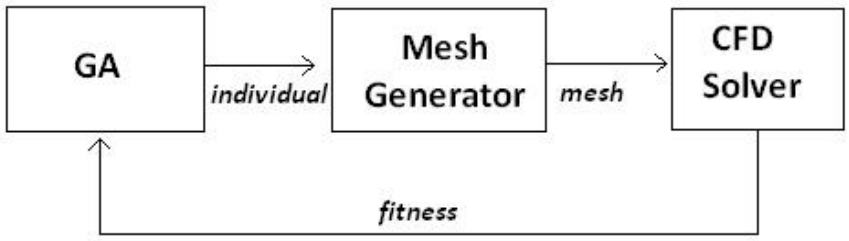

Figure 2.3 Schematic of information flow in the GA optimization process

For any individual, the GA creates a data file from its control points that contains vertices for all of the relevant points of the geometry. The GA then runs a batch file that opens the mesh generator and tells it to run a journal file that takes the geometry data file and creates a mesh. The journal must be robust enough to account for the

possible variation in the geometries and still build a usable mesh. Therefore, a fixed number of nodes are defined on the surfaces of the adjustable shape representing the shroud wall, and the mesh generator fits a fixed number of quadrilateral cells into the geometry based on those nodes. The cells conform to the geometry even though the diffuser in the computational domain may change in geometry during the GA convergence process. Once the GA detects the mesh generator has finished its task, it calls a cleanup batch file to erase unnecessary temporary files, and then calls another batch file that opens the CFD flow solver. This too has a robust journal file that sets the correct parameters and solves the flow in the mesh that was just created. This journal is modified by the GA for each individual before the flow solver runs to ensure that it has the correct pressure drop as determined by Equation 2.1 for that individual. Once the flow solver completes its task, it records the fluid area-averaged flow velocity through the actuator disk. This is then read by the GA and is used to calculate the fitness of the shroud shape. The GA also imposes a penalty function for any control point that moves outside its intended domain by reducing its fitness by a factor of its distance out of bounds. If there is an error such that the flow solver does not return a value, for example if the mesh is physically incorrect or the solver diverges and returns an unreasonable value, the fitness of the individual is set to -1 so that natural selection will remove it from the next generation, but it still contributes to the breeding process explained below.

2.3.4 Advancing to the Next Generation with Crossover and Mutation

In the implementation of the GA for shroud optimization, a generation size of 20 individuals is used with a natural selection rate of 50% with no culling tolerance; that is, no attempt is made to remove similar shrouds. Once the fitness values of all the individuals in a generation are determined, the worst 50% of each generation are replaced through an extrapolation-based crossover scheme. Reproduction is initiated by

randomly selecting two individuals in the generation. The offspring individual is obtained by moving the components of each control point according to equation 2.4. Each new control point is set beyond the more fit parent's control point value from the lesser parent's control point value. The distance it is moved beyond is a random fraction of the distance between the parents' control points.

$$crossover(\mathbf{x}_1, \mathbf{x}_2) = rand(0,1)\cdot(\mathbf{x}_2 - \mathbf{x}_1) + \mathbf{x}_2 \qquad (2.4)$$

Once the new generation of 20 airfoils is created, the mutation procedure occurs. A random number generator uses the mutation rate to decide how many of the 20 individuals to remove and replace with new, randomly generated, individuals. Mutation allows for the possibility of rapid changes and advancement. It also guards against becoming stuck in a relative maximum of fitness and missing the true maximal fitness.

Each configuration in Table 2.1 in section 2.4 was run in the GA for 350 generations, as this number was determined to be sufficient for reasonable convergence without excessive computing time. The number of control points was a large factor in determining the necessary number of generations. The crossover function moves all the control points, so they must be simultaneously optimized. The random guesses made in the first generation and by mutation can accelerate the process significantly. The best way to make the process more rapid is to have small domains for each control point so that the random individuals will be close to the optimum. This however runs the risk of placing the domains in the wrong positions and not obtaining the optimum individual. The control points of the final individual must be checked to see if they are on a boundary condition, since this condition indicates that its optimum position may be beyond the boundary.

2.3.5 Example Convergence History of the Genetic Algorithm

As explained previously, the GA begins with a set of individuals whose control points are randomly generated within the area of the domain following certain constraints which are designed to improve the guessing process and limit the maximum extent of the shroud. As the generations progress, the pool of individuals become more fit. Figure 2.4 shows the gradual improvement of the best individual (shroud shape) in each generation. Due to mutation, the best individual may be removed periodically, but the trend is generally in the correct direction. There are a significant number of poor individuals tested in the process. Figure 2.5 shows all of the ones that returned Cp values within the acceptable range. The following Figures 2.6 through 2.10 show the progression of shroud shapes and flow fields as the GA advanced.

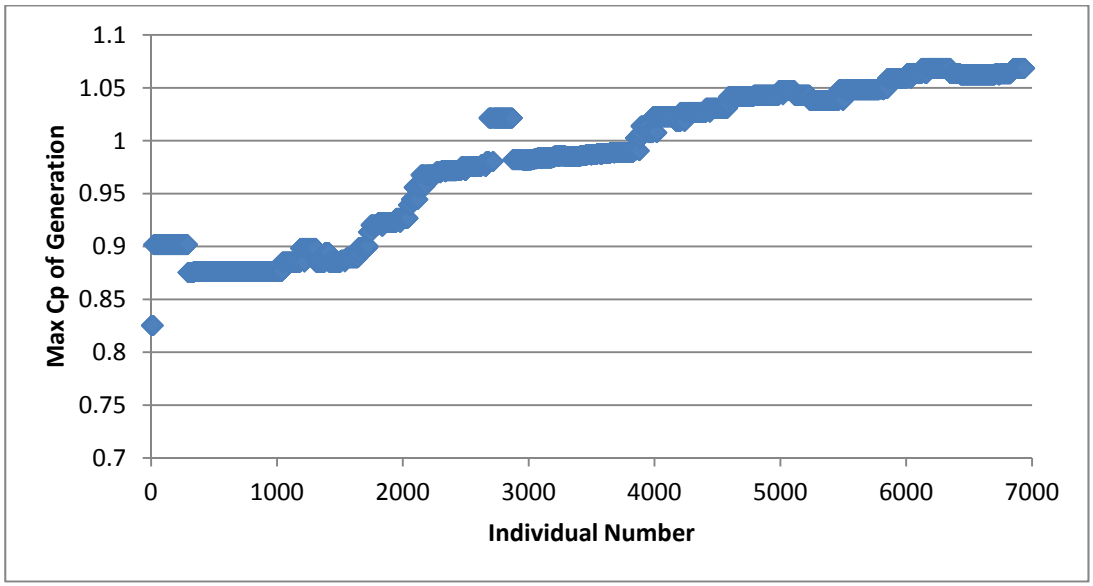

Figure 2.4 Convergence history of GA optimization process (Case 1d of Table 2.1).

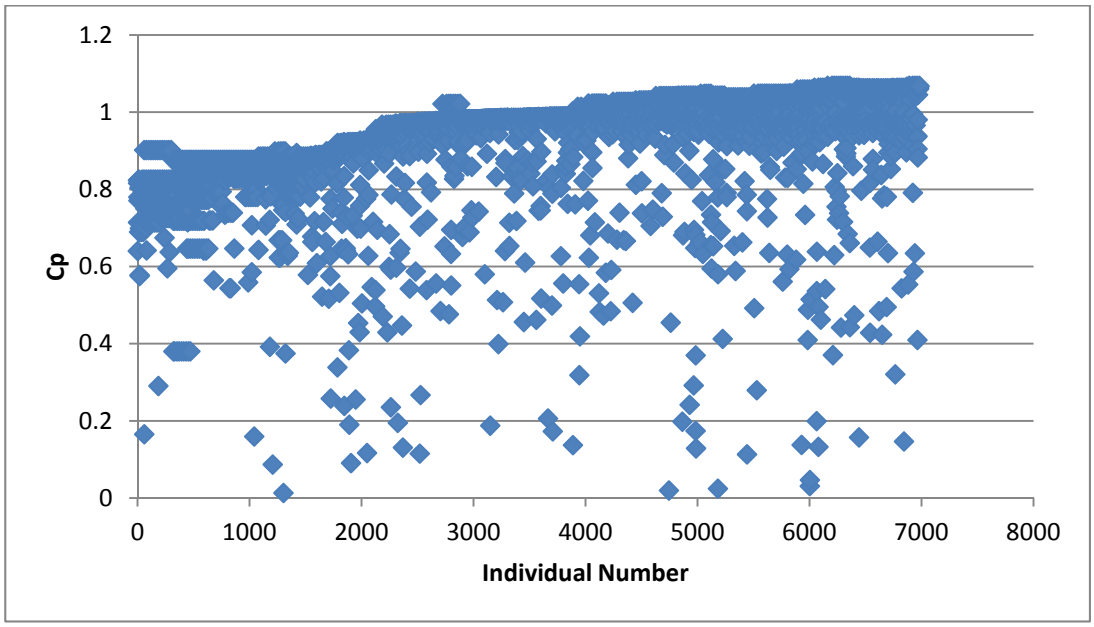

Figure 2.5 Cp of all individuals in the GA convergence history within reasonable bounds (Case 1d of Table 2.1).

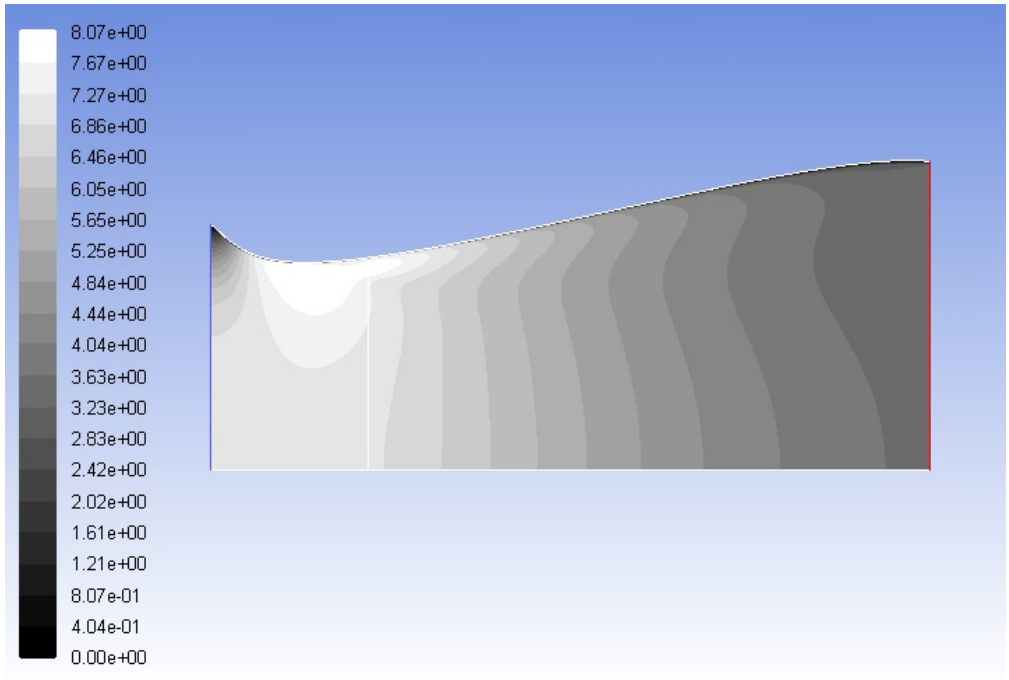

Figure 2.6 Velocity contours (m/s) for the best individual in generation 1 of Case 1d of Table 2.1, Cp=0.825.

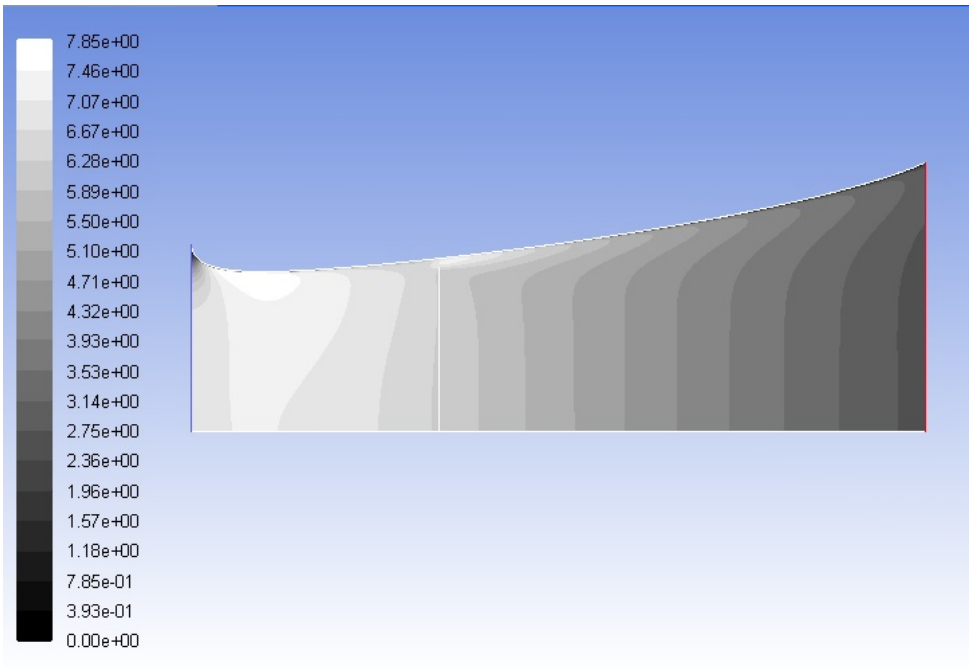

Figure 2.7 Velocity contours (m/s) for the best individual from generation 50 of Case 1d of Table 2.1, Cp=0.877.

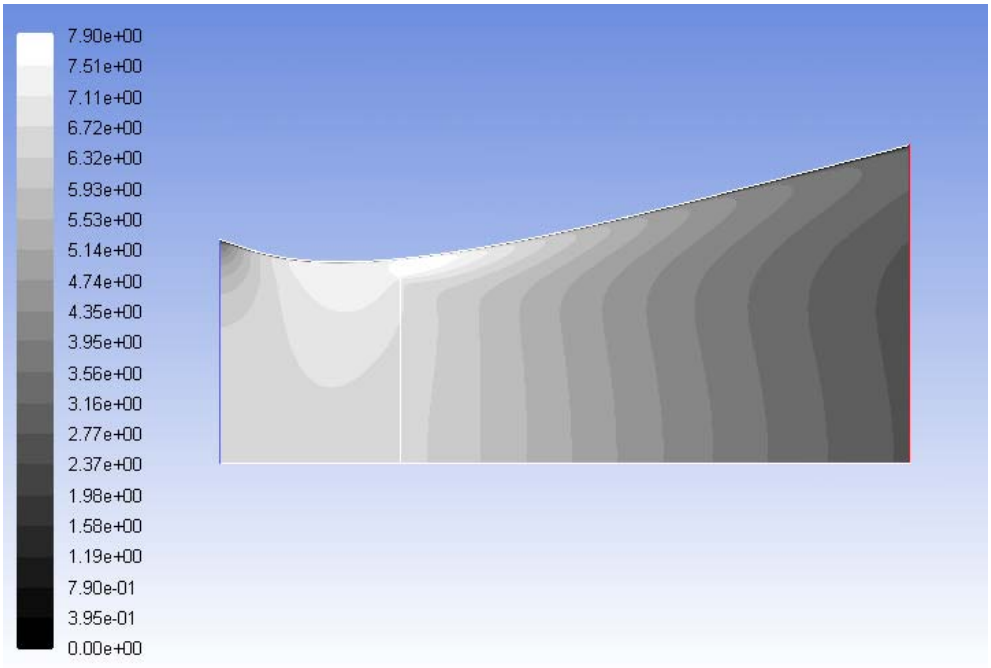

Figure 2.8 Velocity contours (m/s) for the best individual from generation 110 of Case 1d of Table 2.1, Cp=0.957

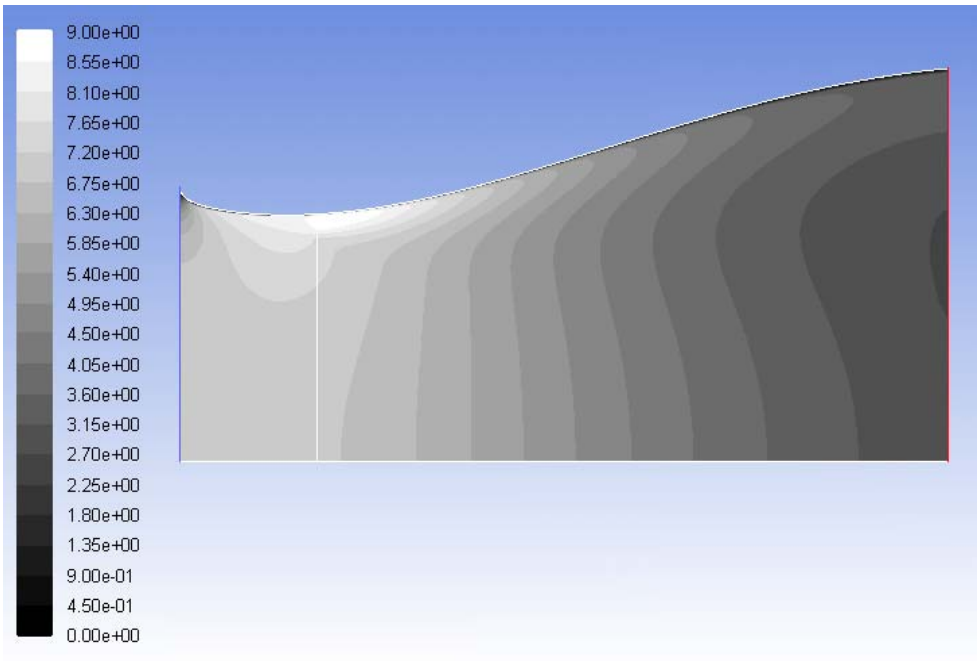

Figure 2.9 Velocity contours (m/s) for the best individual in generation 220 of Case 1d of Table 2.1, Cp=1.027.

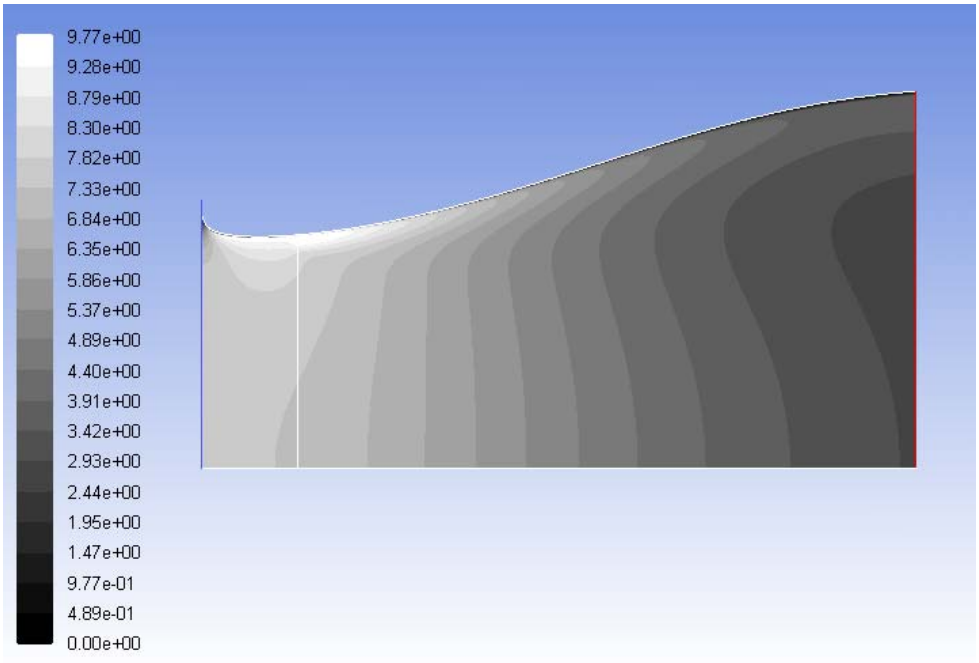

Figure 2.10 Velocity contours (m/s) for best individual in generation 349 of Case 1d of Table 2.1 (converged optimized shroud shape), Cp=1.069.

2.4 Results

In this thesis, thirteen HAWT shrouds were optimized using the combination of a GA and a flow solver. Turbine radius (R_t), wind speed (V_o), shroud exit area to turbine area ratio (R_e^2/R_t^2), and shroud exit length ($L2_{max}$) were all varied. The effects of each of these four variables are discussed in subsequent sections.

2.4.1 Configurations Considered

Figure 2.11 below shows the schematic of the axi-symmetric shrouded wind turbine model.

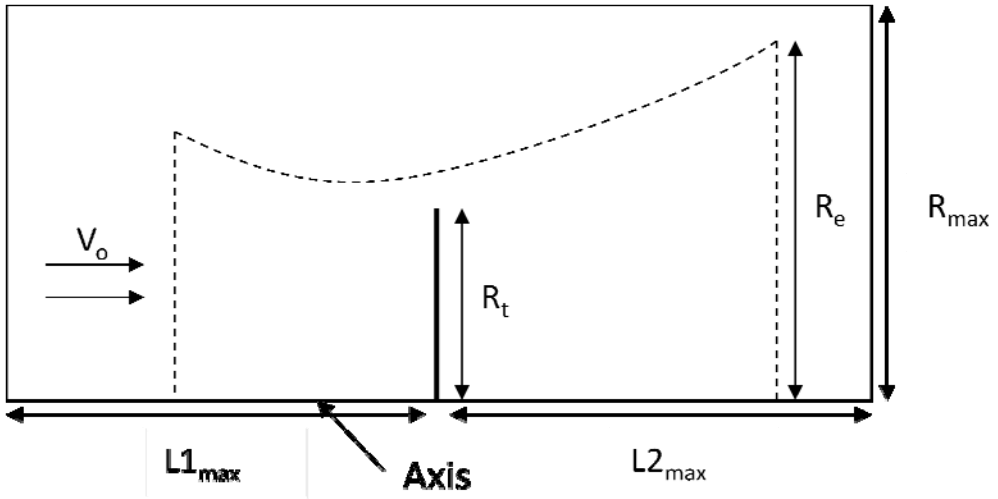

Figure 2.11 Schematic of the axisymmetric HAWT model- actuator disk and shroud dimensions.

Error! Reference source not found. shows all the thirteen cases considered with their corresponding parameters. These cases represent enough variation in various relevant parameters to provide sufficient information to draw reasonable conclusions.

Table 2.1 Parameters used in various shroud configurations.

Case	R_t (ft)				V_o (m/s)		R_e^2/R_t^2			L2 Max./R_t		
	1.5	5.0	8.0	10.0	5.71	7.0	2.0	2.56	3.0	1.0	2.0	3.0
1a		X			X			X		X		
1b		X				X		X		X		
1c		X			X		X				X	
1d		X			X				X			X
1e		X			X				X		X	
1f		X			X				X	X		
1g		X			X		X			X		
2a			X		X			X		X		
2b			X			X		X		X		
3a	X				X			X		X		
3b	X					X		X		X		
4a				X	X		X				X	
4b				X		X	X				X	

Table 2.2 gives the power production data for the optimized shroud cases of **Error! Reference source not found.**. Figure 2.12 shows the optimized shroud shapes scaled and superimposed for a 5 ft turbine radius. For individual figures of each optimized shroud with control points labeled, see section 5.1 in the Appendices.

Table 2.2 Power production and power coefficients for all optimized shroud configurations of Table 2.1.

Case	Power (W)	Cp
1a	717	0.86
1b	1294	0.84
1c	642	0.77
1d	889	1.07
1e	861	1.03
1f	755	0.91
1g	582	0.70
2a	1849	0.87
2b	3321	0.85
3a	65	0.87
3b	117	0.85
4a	2540	0.76
4b	4637	0.76

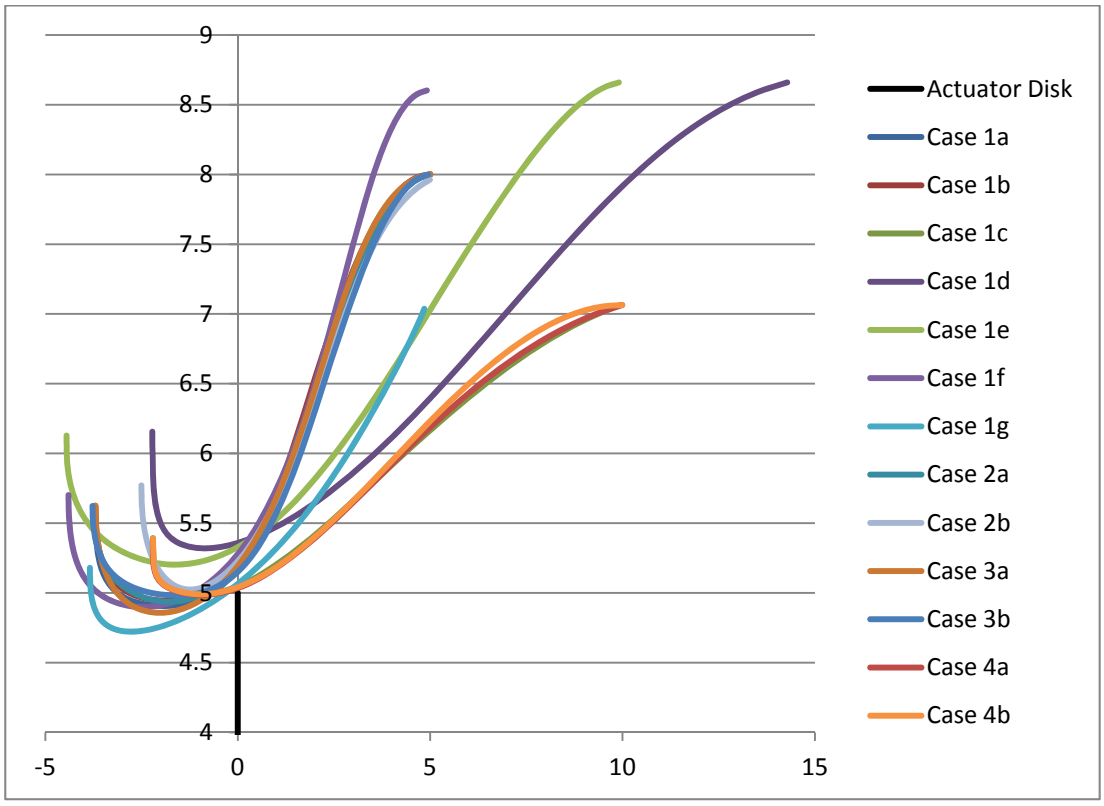

Figure 2.12 Optimized shroud shapes scaled to match a 5 ft radius turbine; these cases correspond to those given in Table 2.1.

2.4.2 Effect of Turbine Radius

The size of the turbine over the range of radii tested does not have a discernible effect on the data computed in this thesis. Two sets of identical tests at two wind speeds show little difference in performance outside the margin of error in the simulations as can be seen in Figure 2.13. Figure 2.14 shows the shroud shapes of these same cases and they are quite similar. This is consistent with the observation that the viscous effects do not become dominant until the size of the turbine becomes very small; much smaller than the turbines analyzed in this work.

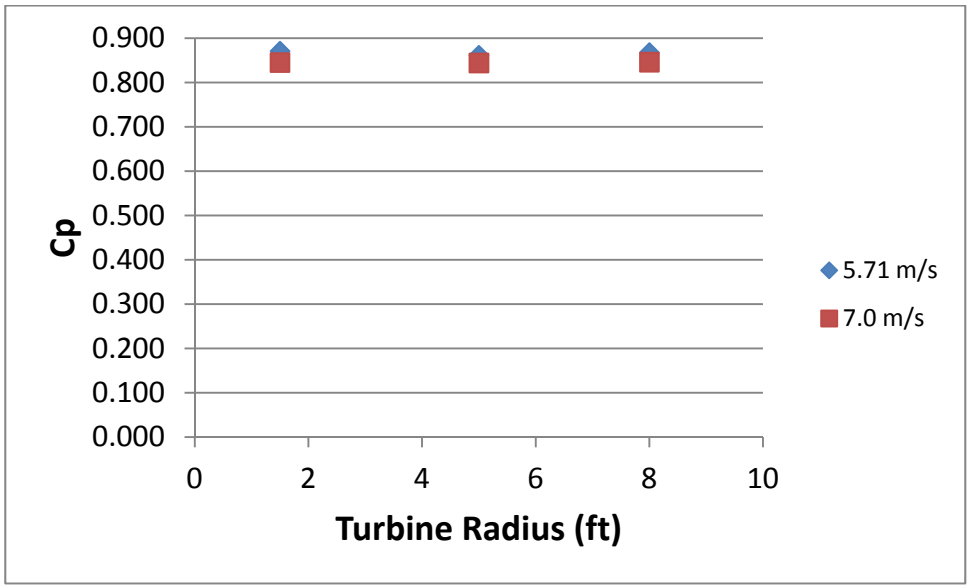

Figure 2.13 Variation of Cp with turbine radius at two free stream wind velocities.

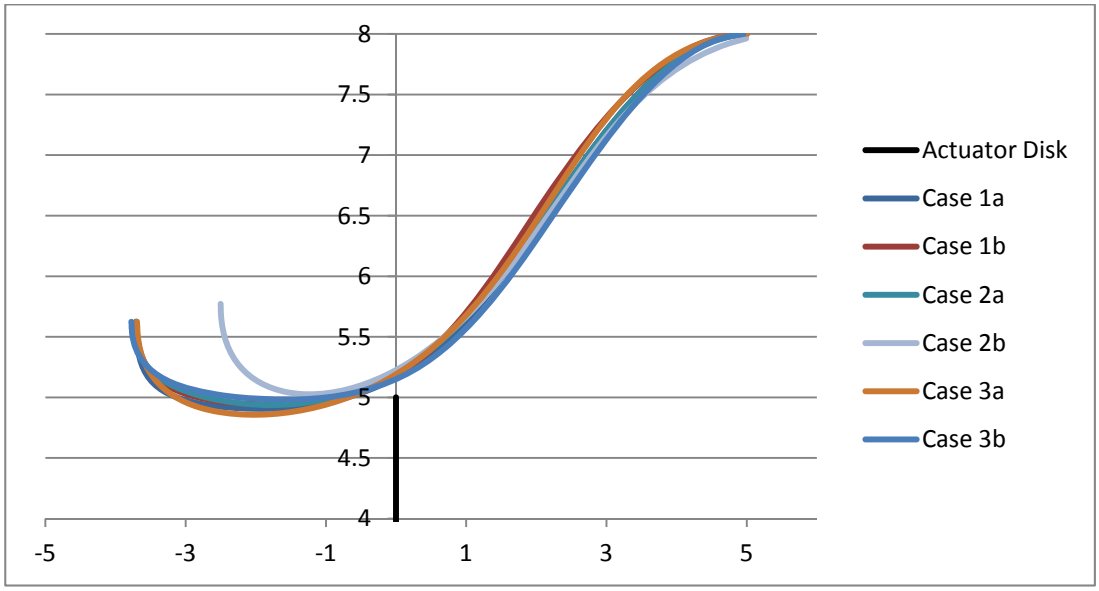

Figure 2.14 Optimized shroud shapes for six cases (1a, 1b, 2a, 2b, 3a, and 3b of Table 2.1). These cases correspond to those shown in Figure 2.13.

2.4.3 Effect of Wind Speed

The effect of wind speed on shroud shape optimization and its performance was also small but consistent. Figure 2.13 shows that at higher wind speed the shrouded turbines were slightly less productive in generating power. Viscous losses due at higher wind speeds are the most likely source of this lesser productivity. There is no consistent trend in the shapes of the shrouds in Figure 2.14.

2.4.4 Effect of Shroud Diffuser Exit to Turbine Area Ratio

The ratio of the shroud exit area to the turbine area was the primary factor in determining shroud performance. A strong positive correlation was observed in all the cases of Table 2.1 as shown in Figure 2.15. The larger size diffusers caused greater flow velocities at the actuator disk, leading to higher power coefficients.

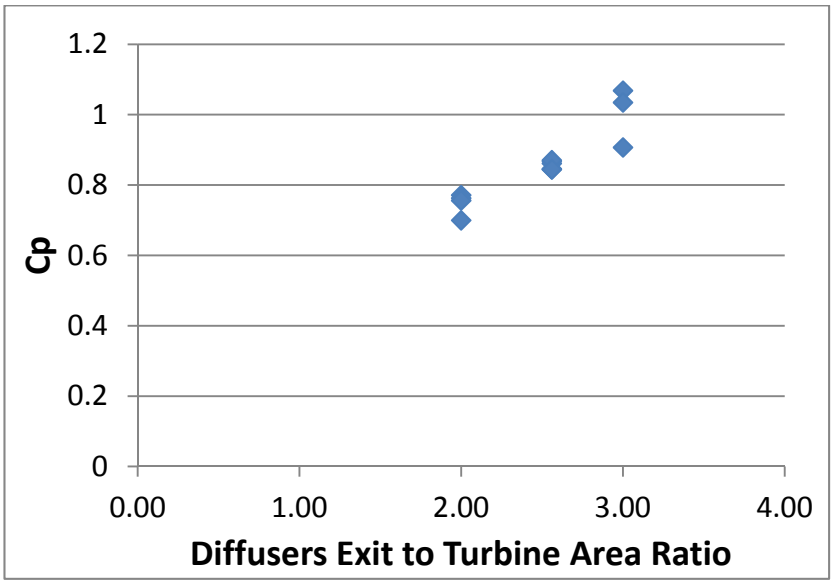

Figure 2.15 The effect of the diffusers' exit area to turbine area ratio on the shrouded turbine Cp.

2.4.5 Effect of Shroud Diffuser Exit Length

The length of the diffuser behind the actuator disk, L2, was not found to be a major factor in determining the performance of a shroud. Figure 2.16 shows significant variation in Cp for both shorter lengths and does not indicate a clear trend beyond the selection of cases considered in this study. However, for the three cases (Case 1d, Case 1e, and Case 1f) that keep other factors equal except length L2, a strong correlation can be seen in the data in Figure 2.17. The primary reason for this trend is the flow separation close to the surface of the diffuser. All three cases have the same exit area which is 3 times the turbine area. The three shroud shapes are shown together in Figure 2.18. Case 1d did not expand to its maximum allowable value of L2, suggesting that it is at an optimum angle. The shorter diffusers create such steep angles that the flow separates and the diffusers become less efficient as can be seen in Figure 2.19.

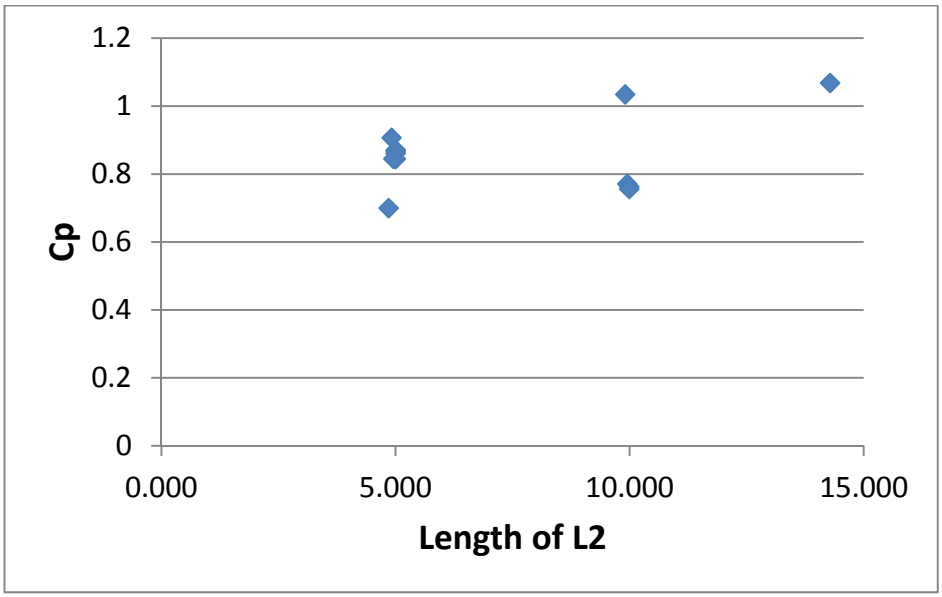

Figure 2.16 Performance of shrouded turbines of various lengths L2.

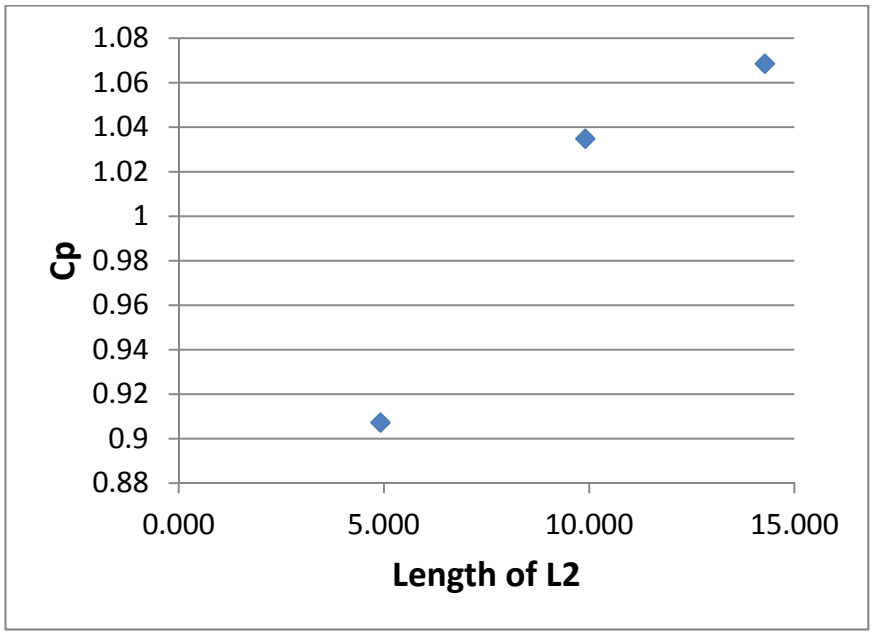

Figure 2.17 Variations of Cp for shrouded turbines of different lengths (cases 1d, 1e, and 1f of Table 2.1).

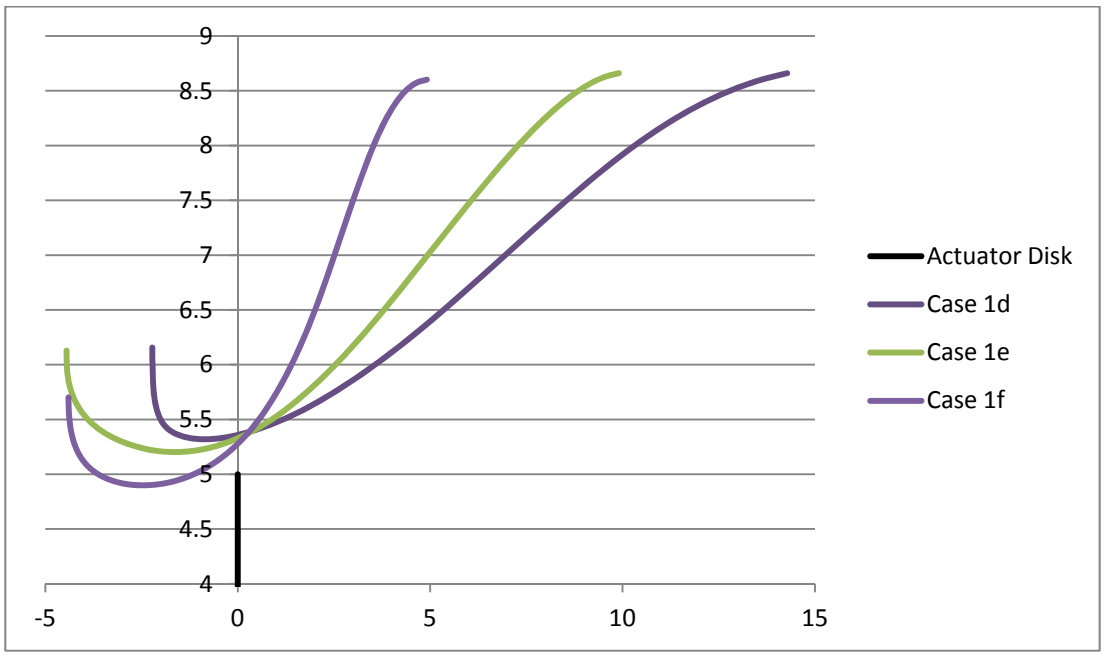

Figure 2.18 Optimized shroud shapes for cases 1d, 1e, and 1f of Table 2.1.

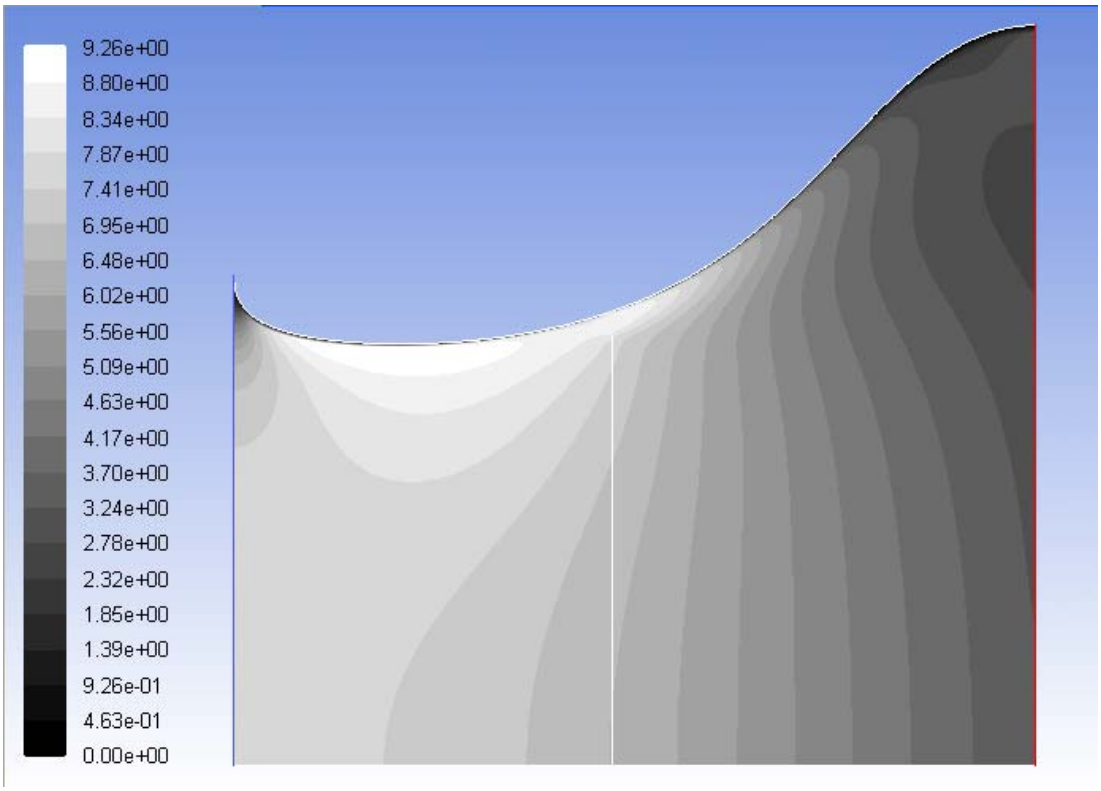

Figure 2.19 Velocity contours (m/s) for the flow field of Case 1f of Table 2.1 showing flow separation at the outer portion of the diffuser.

2.4.6 Comparison of Present Results with the Results of Werle and Presz [1]

Werle and Presz [1] have computed the power coefficient of a shrouded turbine employing inviscid theoretical analysis. Hansen et al. [13] have computed the power coefficient employing viscous CFD analysis. Werle and Presz presented a figure with their calculations compared to those of Hansen et al. that related the power coefficient to the thrust coefficient for an example shrouded turbine. This figure is reproduced as Figure 2.20.

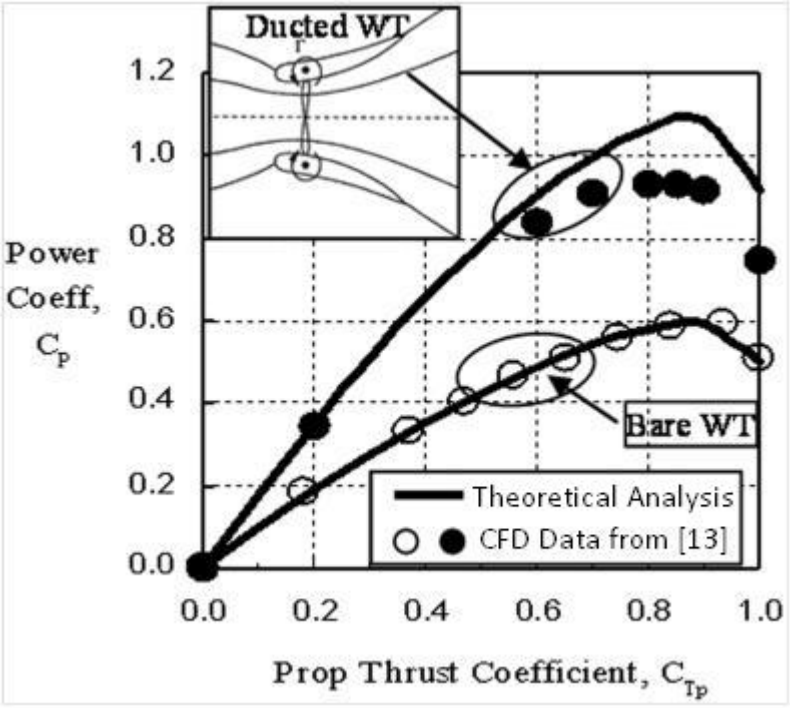

Figure 2.20 Comparison of Cp for a bare and shrouded turbine with thrust coefficient using theoretical inviscid analysis and viscous CFD analysis. (Figure from Werle and Presz [1], CFD data from Hansen et al. [13])

The power coefficient was previously formulated in Equation 2.2, but the comparison requires the thrust coefficient C_T. The thrust coefficient is the nondimentionalized form of the thrust force on the turbine normalized by the momentum in the free stream for an equivalent area; it is given by Equation 2.5 below.

$$C_T = \frac{T_t}{\frac{1}{2}\rho A_t V_o^2} \tag{2.5}$$

Equation 2.6 defines the thrust force on the turbine as the pressure differential across the actuator disk multiplied by the surface area of the turbine. The pressure differential has been previously defined in equation 2.1. Combining Equation 2.1 with Equation 2.5 gives Equation 2.7 for the thrust coefficient.

$$T_t = (\Delta P_t)A_t \qquad (2.6)$$

$$C_T = \left[\left(\frac{A_2}{A_4}\right)^2 - \left(\frac{A_2}{A_1}\right)^2\right]\frac{V_t^2}{V_o^2} \qquad (2.7)$$

It is worth noting that equation 2.7 is different from the way C_T was calculated in Betz's analysis. In equation 2.7, the velocity at the turbine is a known quantity. In Betz's analysis, the wind velocity at the turbine is calculated from the inlet and outlet velocities. The formulation here is more direct because in the CFD environment, the flow properties can simply be read out without the necessity of inference.

When the data for the optimized shrouded wind turbines generated in this thesis are plotted as Cp vs. C_T similar to Figure 2.20, the results can be compared with those of Werle and Presz [1]. This is shown in Figure 2.21.

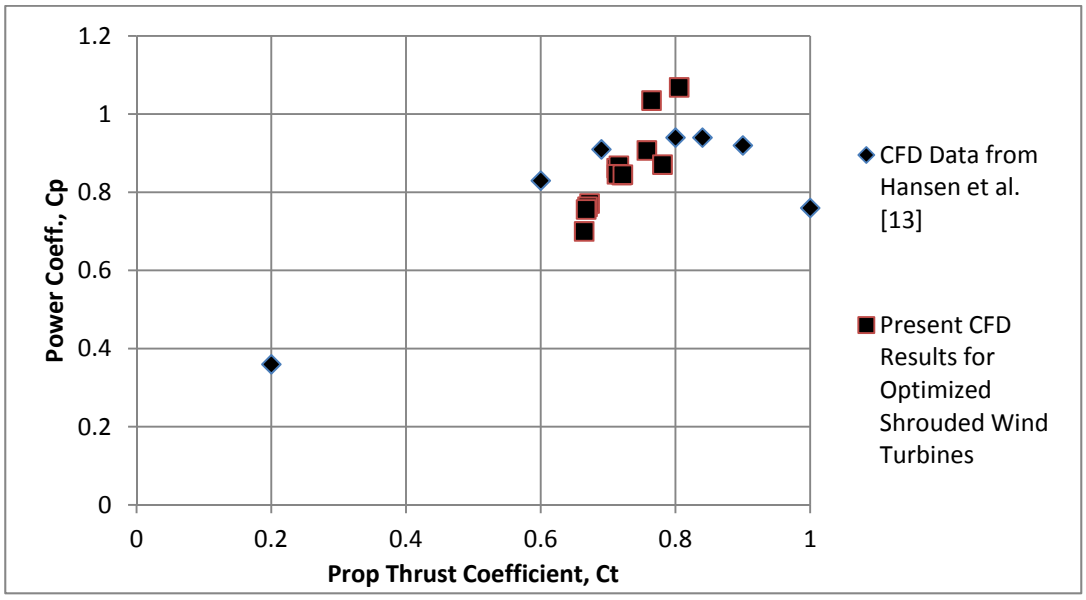

Figure 2.21 Comparison of optimized shroud results to data for a shrouded wind turbine from Hansen et al. [13] as presented in Werle and Presz [1].

The comparisons in Figure 2.21 show that the present results are in reasonably good agreement with those reported previously in the literature. However, it is important to

note that the present results are for optimized shrouded wind turbines in contrast with those reported by Werle and Presz [1].

Since the GA finds the optimum pressure coefficient at the actuator disk, the full range of thrust coefficients could not be shown by the optimized shrouded wind turbine data in Figure 2.21. In order to demonstrate the trend shown by the CFD data in Werle and Presz's [1] work, the pressure coefficient was varied for the optimized shrouded wind turbine (Case 2aof Table 2.1) from 0.05 to 1.5 and then Cp and Ct were computed. Figure 2.22 shows the close agreement between the present results with those of Werle and Presz. For Case 2a of Table 2.1, the maximum Cp occurs at a point using the pressure coefficient calculated by the GA. The deviation at high values of thrust coefficient is due to the tip gap present in this work and not in Werle and Presz's. As the actuator disk exerts more back pressure on the flow, a greater portion of it escapes through the gap resulting in a loss of pressure and power generation.

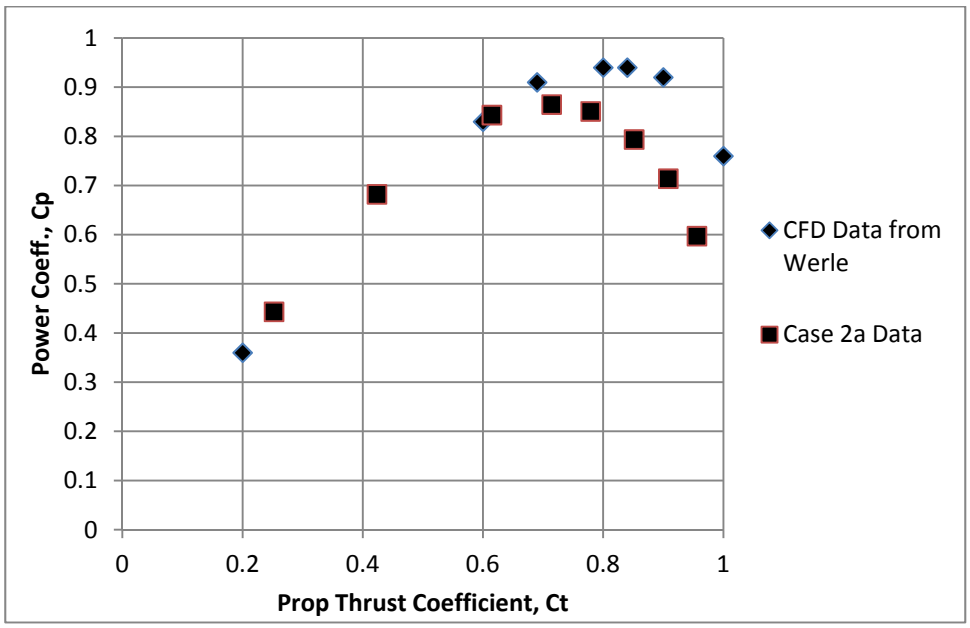

Figure 2.22 Variation of Cp with Ct for an optimized shrouded wind turbine (Case 2a of Table 2.1).

2.4.7 Dependence of the Power Coefficient on the Area Used for Non-Dimentionalization

The intent of a shrouded wind turbine is to change the contours of the stream tube in order to improve the power generation potential. By widening the inlet and the outlet relative to the turbine, the velocity of the wind at the throat increases, providing more energy production potential.

The Betz limit states that a maximum of 16/27 of the momentum in a given flow can be extracted by a turbine. The unshrouded or bare HAWT is modeled as an actuator disk inside a stream tube and the conservation equations of fluid dynamics are applied. The normalization area for power generation calculation is the swept area of the turbine.

In a 2009 paper considering potential flow in an axisymmetric ducted wind turbine, Widnall [3] argued that, "if the power extraction of a ducted turbine as compared to the Betz model is referred to the exit area of the duct rather than the throat area, no power augmentation occurs" beyond the Betz limit. The optimized shrouded wind turbines considered in this paper, in all thirteen cases, confirm her argument.

The debate centers on how to normalize the power coefficient for the shrouded case. If the normalization area remains the turbine area, then the increased power due to the diffuser can cause the Cp to exceed the Betz limit. However, because the shroud has increased the cross-sectional area of the free stream that is disturbed, Widnall argues that the Betz limit is in fact not violated. Cp simply needs to be normalized to the shroud exit area, which is a more reasonable approximation of the total energy available in the flow.

The calculations presented in this thesis for the optimized shrouded wind turbines support Widnall's claim that with exit area normalization, the Betz limit is not violated. Figure 2.23 shows the power coefficients obtained by nondimentionalizing it by the turbine area and by the shroud exit area.

When C_p is nondimentionalized by the turbine area, it exceeds the Betz limit of approximately 0.59 for all cases, but when it is nondimentionalized by the shroud exit area, the coefficients are significantly lower than the Betz limit.

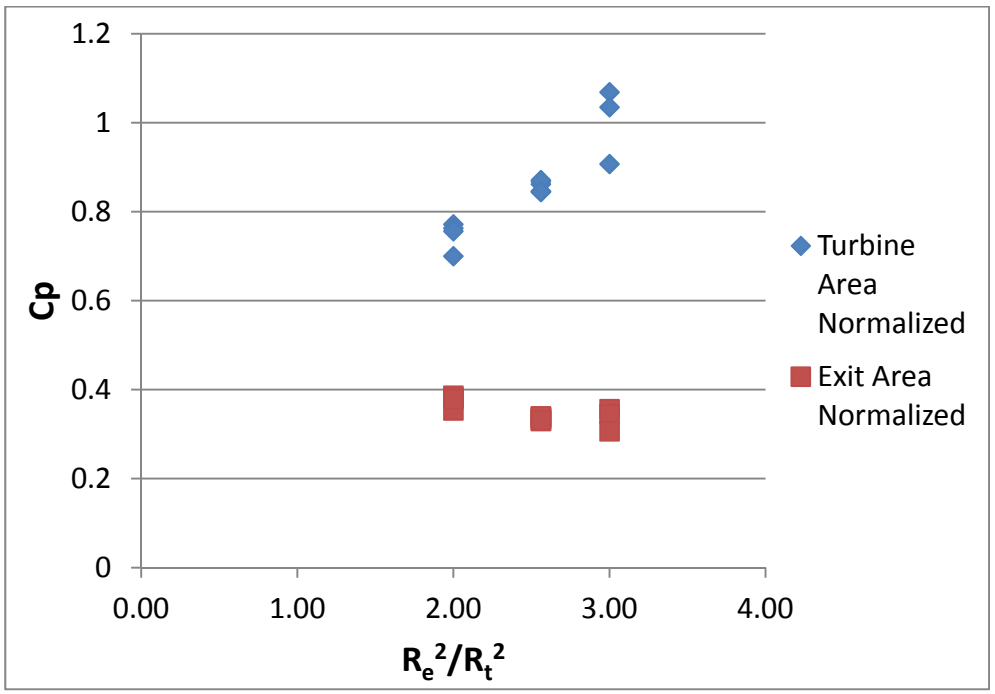

Figure 2.23 Cp of optimized shrouded wind turbines obtained by nondimentionalizing it by the turbine area and the shroud exit area.

In Figure 2.23, R_e^2/R_t^2 demonstrates a marginally inverse relationship with Cp. This can be attributed to the increasing viscous losses due to higher fluid velocities as well as due to greater tip gap losses. Additional testing would be necessary to quantify these effects more clearly and effectively.

The most significant result of Figure 2.23 is that the Betz limit is not violated using Widnall's argument.

2.5 Future Work

Several aspects of the work performed in this thesis require further investigation. There are several significant elements that should be considered to advance this work. (1) Improve the actuator disk model by including rotational effects through blade element momentum theory. This will result in more precise estimates of Cp. (2) Model both internal and external flow around the shroud so that more accurate optimal aerodynamic shapes can be developed. (3) Full 3-D CFD modeling of the blades' interactions within the shroud are needed to develop viable real turbine systems. (4) Determine the effects of shrouds on the turbine wakes within wind turbine farms. It needs to be understood how shrouded HAWT would fit into the commercial wind farm structure. (5) Perform an analysis to determine how shrouded turbines compare to the standard HAWT in cost and performance. (6) Perform full 3-D CFD optimization of shroud shape. (7) Construct and test physical models. The CFD data is purely academic until experimental data can validate the true performance of these systems.

Chapter 3

3. Modeling of Wind Turbines in a Solar Chimney and Shape Optimization of a Diffuser at the Top of the Chimney

3.1 Introduction

In past several years, several studies have shown that shrouded wind turbines can generate greater power than bare turbines. A solar chimney not only generates an upward draft of the wind inside the solar tower but also creates a shroud around the wind turbine. A large number of silos stand empty on farms, especially in the mid-western U.S. The objective of this study is to determine the potential of these silos in generating wind-power by installing a wind turbine inside the silo. Figure 3.1 shows a typical silo enclosed by the staves (blocks). For the purposes of this thesis, a typical silo is 70 ft. in height and 15 – 18 ft. in outside diameter. The dimensions of a stave or block are 10 inch horizontal x 30 inch vertical x 3.5 inch thick. The staves are offset vertically by 6 inch as shown in Figure 3.1. These staves can be removed from various locations around the periphery of the silo and can serve as inlets for the outside air. The air drawn in through these inlets will move upward through the silo (like in a chimney) due to temperature differential between the lower and upper part of the silo (the air at lower part of the silo being at higher temperature). At the top, the silo will be open to the

outside atmosphere. A wind turbine can be installed inside the silo at a suitable height from the ground to extract the kinetic energy of the wind flowing upward from the inlets due to natural convection. The potential of this concept for wind power generation is evaluated by numerical simulation.

3.2 Technical Approach

The performance of a wind turbine inside a vertical silo is modeled by creating an actuator disk model in the CFD commercial solver FLUENT [5]. The geometries of the three configurations studied are shown in Figures 3.2(a), 3.2(b) and 3.2(c). In Figure 3.2(a), the silo is modeled as a circular cylinder with top open to the atmosphere. In Figure 3.2(b), a venturi (a converging – diverging nozzle) is created around the turbine inside the silo. In Figure 2(c), a diverging diffuser section is placed on top of the silo; this section need not be made of staves and can be of a light weight structurally strong material to withstand the forces due to class – 3 wind. Various parameters shown in Figure 3.2 can be easily varied in the computer program to determine their effect on wind power generation.

The geometric models of Figure 3.2 are created in the software "GAMBIT" [6]. A structured mesh is generated in the axisymmetric models. The incompressible Navier-Stokes equations solver in FLUENT, with Boussinesq approximation and a two equation realizable k – ε model is employed in all the calculations.

Figure 3.1 A typical silo on a farm

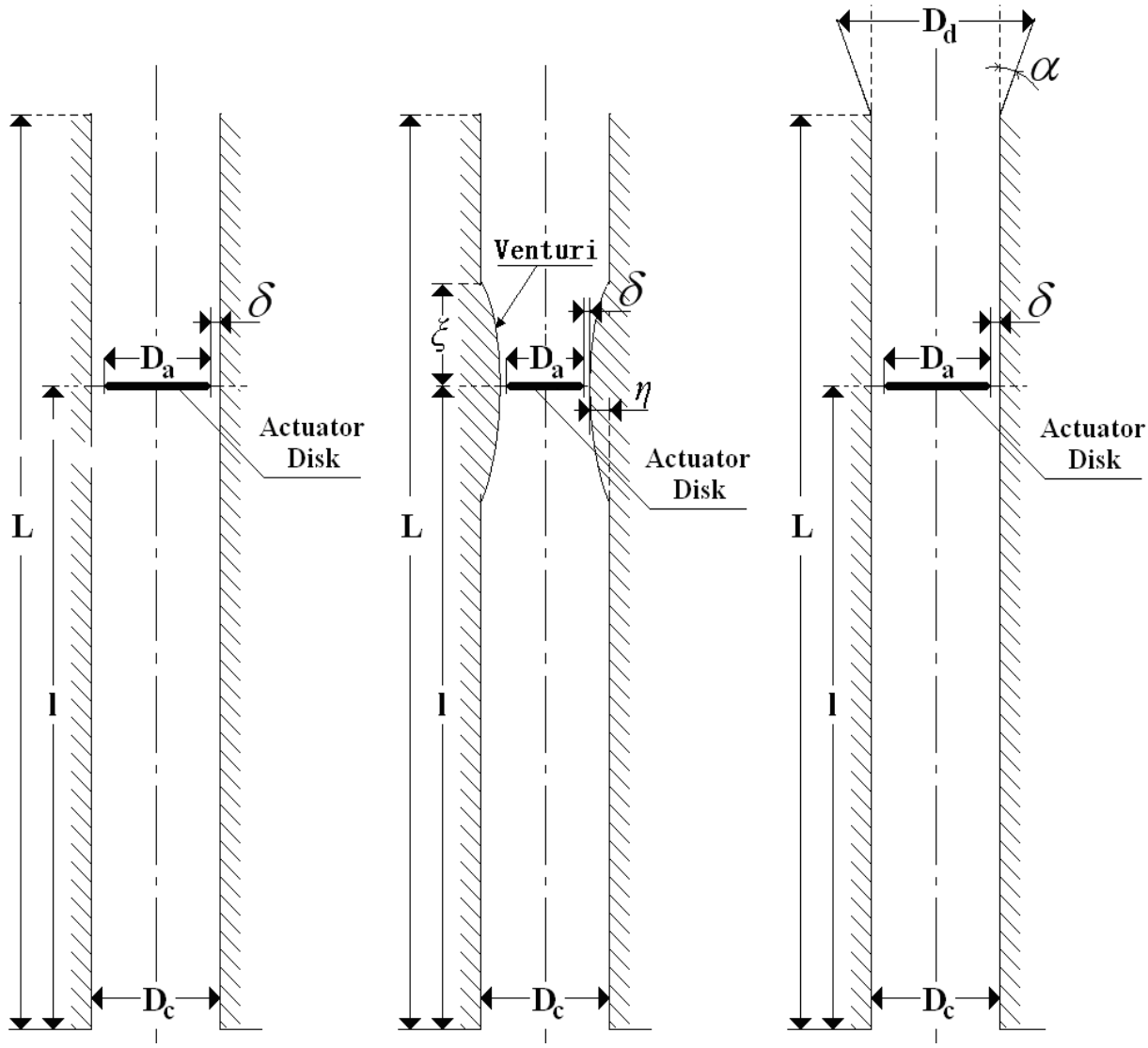

Figure 3.2 (a) Computational geometry of a cylindrical silo (without diffuser), (b) Computational geometry of a cylindrical silo with a venturi around the turbine, (c) Computational geometry of a cylindrical silo with diffuser on the top; L = Height of the silo, Dc = Interior Diameter of the Cylinderical Silo, Da = Diamer of the Turbine modeled as an Actuator Disc, l = Height of the Turbine from the Ground, δ = Clearance between the Turbine and the Silo Wall, Dd = Diameter of the Exit Section of the Diffuse, α = Diffuser Angle, ξ and η define the Parameters related to Venturi.

3.3 Results

As mentioned in the "Introduction" section, first the mathematical model (actuator disk) model was validated by computing the pressure coefficient (Cp) for a bare turbine; Cp value close to Betz's limit (~0.59) was obtained. In the validation, the flow was assumed to be inviscid, incompressible and irrotational (potential flow). After the validation, calculations were performed for five cases using the dimensions of two typical silos of different diameters geometrically modeled as shown in Figure 3.2 and assuming class - 3 wind velocity: (a) bare turbine (without enclosing silo), (b) turbine enclosed by a cylindrical silo, (c) the turbine enclosed by the cylindrical silo with a diffuser at the top of the silo, (d) turbine surrounded by a converging – diverging venturi inside a cylindrical silo, and (e) turbine surrounded by a converging – diverging venturi inside a cylindrical silo with a diffuser at the top of the silo. The calculations for cases (b) – (e) were performed with a temperature differential between the ground and the top of the silo. Cp and generated power were calculated for the five cases. These results are described below:

3.3.1 Configuration 1

Configuration 1A (Figure 3.2(a)): L= 70 ft, l = 50 ft, Dc = 11 ft, Da = 10 ft, δ = 0.5 ft, ΔT = 0, 2, 4, 6, and 8 deg. The wind velocity of class 3 wind was assumed to be V = 5.6m/s facing normal to the actuator disk (turbine).

Configuration 1B (Figure 3.2(a)): L= 70 ft, l = 50 ft, Dc = 17.6 ft, Da = 16 ft, δ = 0.8 ft, ΔT = 0, 2, 4, 6, and 8 deg. The wind velocity of class 3 wind was assumed to be V = 5.6m/s facing normal to the actuator disk (turbine).

The difference between configuration 1A and 1B is in the diameters of the two silos Dc and diameters of the turbines Da.

It should be noted that the wind is sucked in through the openings created by removal of staves. We will call these openings as open-staves. Several effects can occur depending upon the placement of open-staves. If the open-staves are too close to the ground, the wind velocity will be significantly less than 5.6m/s. If they are approximately 4 -6 meters (15 – 20 ft.) above the ground, the wind velocity will be close to that in the atmosphere for that particular day. However, the wind will enter the silo nearly perpendicular to the cylindrical surface. As it moves upward in the silo, it creates a region of separation near the entrance of the open-staves. We studied this effect by assuming a cylindrical opening of 30 inch height at a distance of 5 meters (16 ft.) above the ground. It turns out that within a distance of less than 1 meter (~ 3 ft.), the flow becomes attached again. Therefore, assuming a wind of velocity V facing the turbine at a height of 50 ft from the ground is a reasonable and good approximation for the calculations.

The wind velocity in a vertical silo is also a function of the temperature differential between the ground and top of the silo. Since the silo height is only 70 ft, this temperature differential ΔT is very small; it is 0.025deg F (based on 0.00357deg.F /ft). The buoyancy effect increasing the speed of the upward flowing air is therefore very small. This effect can be increased by increasing the temperature of the sucked-in air near the ground by some means such as electric heaters appropriately placed or by placing solar panels near the ground to create a solar chimney configuration. The effect of increasing ΔT on increasing the upward wind velocity and therefore the turbine power is also calculated.

Table 3.1 provides the power generated by the turbine and the turbine efficiency for the cylindrical silo geometry of Configuration 1A (Figure 2(a)) and class -3 wind speed of 5.6m/s for various values of ΔT.

Table 3.1 Power Generated by the Turbine in Cylinderical Silo (Configuration 1A) of Figure 2(a) and Turbine Efficiency

ΔT (°C)	Power (W)	Wind Energy (W)	Cp
0	290	785	0.37
2	328	785	0.42
4	367	785	0.47
6	408	785	0.52

From Table 3.1, it can be seen that the increase in ΔT increases the generated power as well as the turbine efficiency. Figure 3.3 shows the entire computational domain employed for the case of the turbine inside the cylindrical silo. The silo has been rotated by 90deg. clockwise in this figure. It shows the velocity vectors indicating the direction and movement of the flow for ΔT = 0.

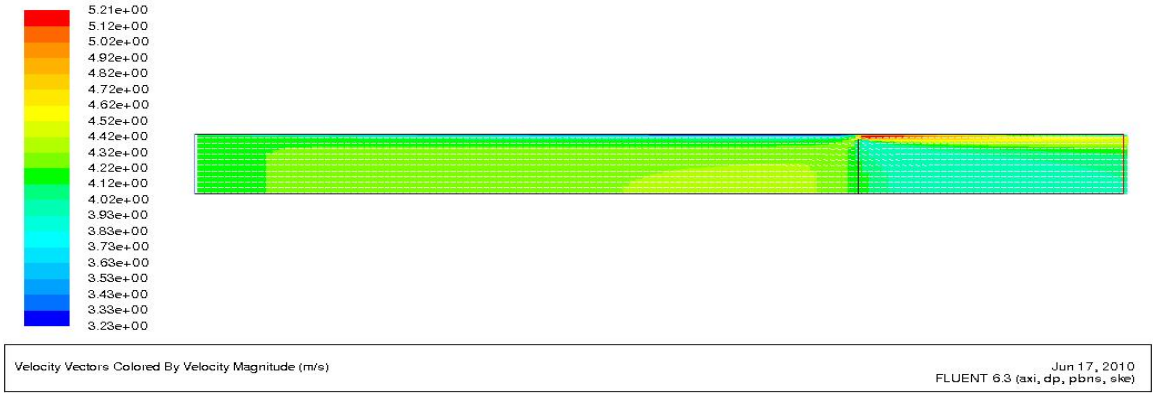

Figure 3.3 Computational domain for flow inside the silo enclosing the turbine: velocity vectors and their magnitude. The silo has been rotated by 90deg. clockwise in this Figure.

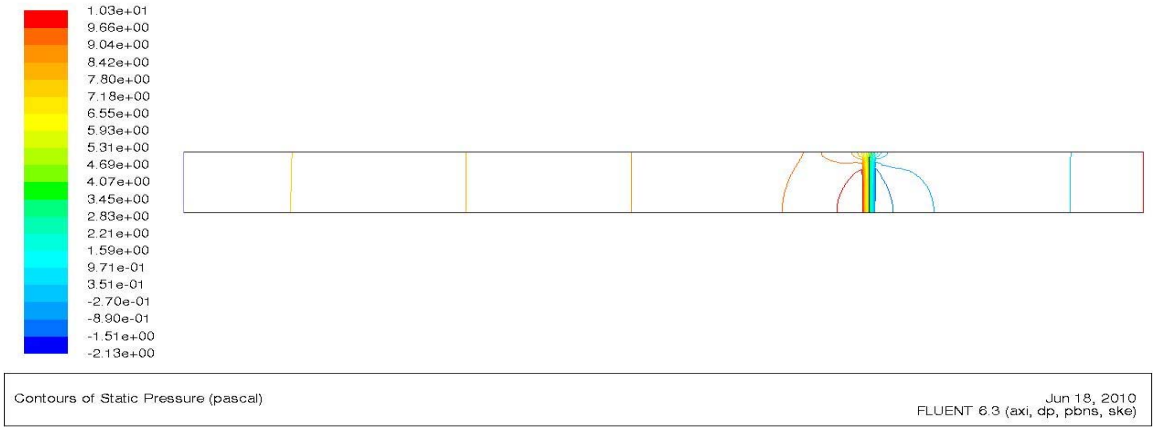

Figure 3.4 Computational domain for flow inside the silo enclosing the turbine: static pressure contours and their magnitude. The silo has been rotated by 90deg. clockwise in this Figure.

Figure 3.4 shows the static pressure contours in the field (in Pascals) for $\Delta T = 0$. It shows that most of the pressure variation is confined to the region near the actuator disk; the pressure upstream of the disk is greater than that downstream as expected. Velocity and pressure contours similar to Figures 3.3 and 3.4 are obtained for other values of ΔT with minor changes. Table 3.2 provides the power generated by the turbine and the turbine efficiency for the cylindrical silo geometry of Configuration 1B (Figure 2(a)) and class -3 wind speed of 5.6m/s for various values of ΔT.

Table 3.2 Power Generated by the Turbine in Cylinderical Silo (Configuration 1B) of Figure 2(a) and Turbine Efficiency

ΔT (°C)	Power (W)	Wind Energy (W)	Cp
0	748	2016	0.37
2	907	2016	0.45
4	1075	2016	0.53
6	1253	2016	0.62
8	1439	2016	0.72

From Table 3.2, it can be seen that larger diameter silo and turbine generate greater wind power as expected. Also, like configuration 1A, the increase in ΔT increases the generated power as well as the turbine efficiency. However, there is little difference in the turbine efficiency between configuration 1A and configuration 1B as expected. Figures analogous to Figures 3.3 and 3.4 are not presented for this configuration because they are qualitatively similar in flow pattern and contours.

3.3.2 Configuration 2

Configuration 2A (Figure 3.2(c)): L= 70 ft, l = 50 ft, Dc = 11 ft, Da = 10 ft, δ = 0.5 ft, α = 20 deg, Dd = 1.5x Dc, ΔT = 0, 2, 4, 6, and 8 deg. The wind velocity of class 3 wind was assumed to be V = 5.6m/s facing normal to the actuator disk (turbine).

Configuration 2B (Figure 3.2(c)): L= 70 ft, l = 50 ft, Dc = 17.6 ft, Da = 16 ft, δ = 0.8 ft, α = 20 deg, Dd = 1.5x Dc, ΔT = 0, 2, 4, 6, and 8 deg. The wind velocity of class 3 wind was assumed to be V = 5.6m/s facing normal to the actuator disk (turbine).

Table 3.3 provides the power generated by the turbine and the turbine efficiency for the cylindrical silo geometry (configuration 2A) with a diffuser on the top (Figure 3.2(c)) and class -3 wind speed of 5.6m/s for various values of ΔT.

Table 3.3 Power Generated by the Turbine in Cylinderical Silo (Configuration 2A) with a Diffuser at top (Figure 2(c)) and Turbine Efficiency

ΔT (°C)	Power (W)	Wind Energy (W)	Cp
0	560	785.00	0.71
2	642	785.00	0.82
4	727	785.00	0.93
6	816	785.00	1.04
8	909	785.00	1.16

From Table 3.3, it can be seen that the diffuser on the top has a significant effect in increasing the turbine power as well as the turbine efficiency which further increases with increase in ΔT. Figure 3.5 shows the zoomed-in view of the computational domain near the actuator disk employed for the case of the turbine inside the cylindrical silo with a diffuser on the top. The silo has been rotated by 90deg. clockwise in this figure. It shows the velocity vectors indicating the direction and movement of the flow for ΔT = 6. Figure 3.6 shows the zoomed-in-view of static pressure contours in field (in Pascals) for ΔT = 6.

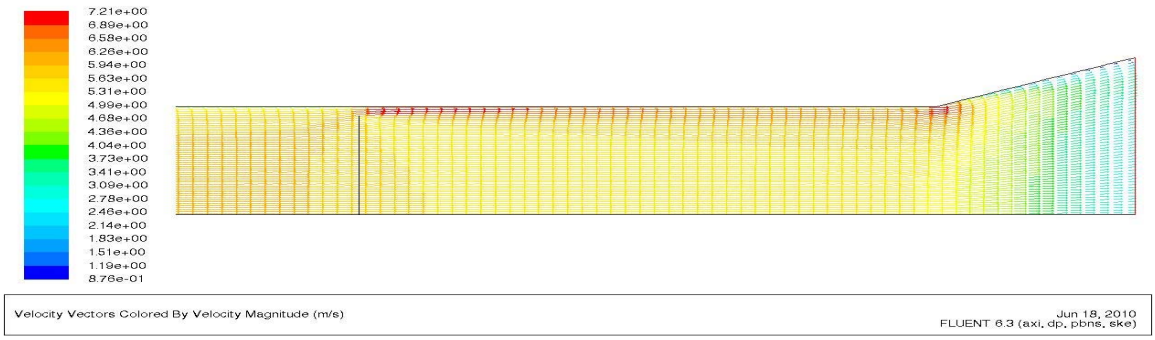

Figure 3.5 Zoomed-in-View of computational domain for flow inside the silo enclosing the turbine with a diffuser on the top: Velocity vectors and their magnitude. The silo has been rotated by 90deg. clockwise in this figure.

Figure 3.6 Zoomed-in-View of computational domain for flow inside the silo enclosing the turbine with a diffuser on the top: static pressure contours and their magnitude. The silo has been rotated by 90deg. clockwise in this figure.

Table 3.4 provides the power generated by the turbine and the turbine efficiency for the cylindrical silo geometry (configuration 2B) with a diffuser on the top (Figure 3.2(c)) and class -3 wind speed of 5.6m/s for various values of ΔT.

Table 3.4 Power Generated by the Turbine in Cylinderical Silo (Configuration 2B) with a Diffuser at top (Figure 3.2(c)) and Turbine Efficiency

ΔT (°C)	Power (W)	Wind Energy (W)	Cp
0	1463	2016	0.73
2	1815	2016	0.90
4	2176	2016	1.08
6	2580	2016	1.28
8	2984	2016	1.48

From Table 3.4, it can be seen that the diffuser on the top has a significant effect in increasing the turbine power as well as the turbine efficiency which further increases with increase in ΔT. Also, it can be seen that larger diameter silo and turbine generate greater wind power as expected. Furthermore, like before for configuration 2A, the increase in ΔT increases the generated power as well as the turbine efficiency. Figures analogous to Figures 3.5 and 3.6 are not presented for this configuration because they are qualitatively similar in flow pattern and contours.

3.3.3 Configuration 3

Configuration 3A (Figure 3.2(b)): L= 70 ft, l = 50 ft, Dc = 11 ft, Da = 8 ft, ξ = 4 ft., η = 1.1 ft., δ = 0.4 ft, ΔT = 0, 2, 4, and 6 deg. This configuration is similar to the configuration in Figure 2(a) with a venturi surrounding the turbine.

Configuration 3B (Figure 2(b)): L= 70 ft, l = 50 ft, Dc = 17.6 ft, Da = 12.8 ft, ξ = 4 ft., η = 1.76 ft., δ = 0.64 ft, ΔT = 0, 2, 4, and 6 deg. This configuration is similar to the configuration in Figure 2(a) with a venturi surrounding the turbine.

Table 3.5 provides the power generated by the turbine and the turbine efficiency for the cylindrical silo geometry (configuration 3A) with a converging-diverging venturi (Figure 2(b)) and class -3 wind speed of 5.6m/s for various values of ΔT.

Table 3.5 Power Generated by the Turbine in a Cylinderical Silo (Configuration 3A) with a Venturi Surrounding it (Figure 3.2(b)) and Turbine Efficiency

ΔT (°C)	Power (W)	Wind Energy (W)	Cp
0	297	502	0.59
2	336	502	0.67
4	376	502	0.75
6	418	502	0.83
8	462	502	0.92

In Table 3.5, it should be noted that the available wind power in this case is less compared to that in Tables 3.1 and 3.2 because the wind turbine diameter in this case is Da = 8 ft. compared to that in configurations 1A and 2A where the wind turbine diameter Da = 10ft. The diameter of the turbine had to be less in this case because of the installation of the venturi. Nevertheless, Table 3.5 shows that the venturi surrounding the turbine (even for a smaller turbine) has a significant effect in increasing the turbine power as well as the turbine efficiency which further increases with increase in ΔT.

Figure 3.7 shows the zoomed-in view of the computational domain near the actuator disk employed for the case of the turbine surrounded by a converging – diverging venturi inside the cylindrical silo. The silo has been rotated by 90deg. clockwise in this figure. It shows the velocity vectors indicating the direction and movement of the flow for ΔT = 6. Figure 3.8 shows the zoomed-in-view of static pressure contours in the field (in Pascals) for ΔT = 6.

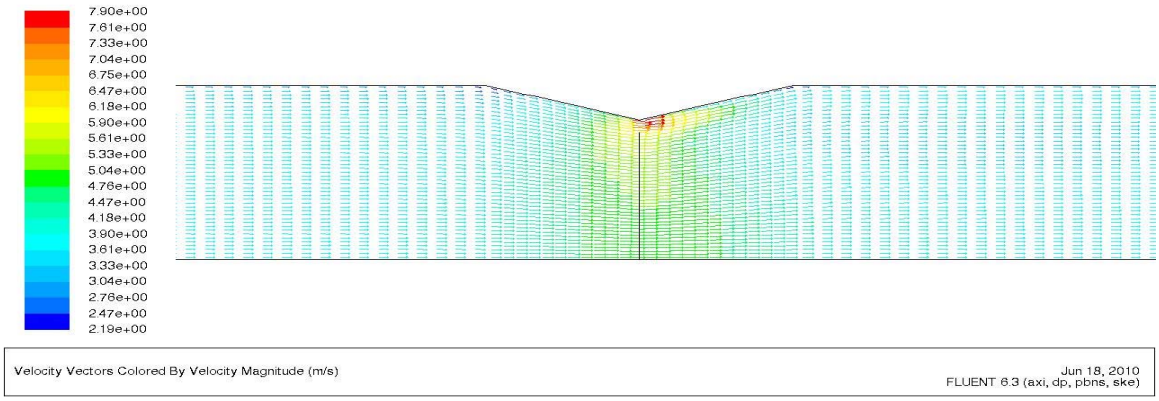

Figure 3.7 Zoomed-in-View of computational domain for flow inside the silo with a turbine surrounded by the venturi (Configuration 3A) (Figure 2(b)): Velocity vectors and their magnitude. The silo has been rotated by 90deg. clockwise in this figure.

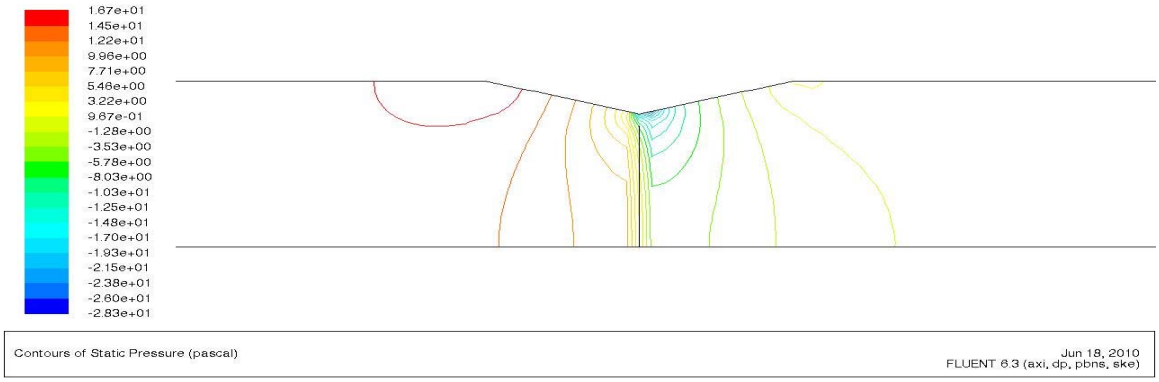

Figure 3.8 Zoomed-in-View of computational domain for flow inside the silo with a turbine surrounded by the venturi: static pressure contours and their magnitude. The silo has been rotated by 90deg. clockwise in this figure.

Table 3.6 provides the power generated by the turbine and the turbine efficiency for the cylindrical silo geometry (configuration 3B) (Figure 3.2(b)) and class -3 wind speed of 5.6m/s for various values of ΔT.

Table 3.6 Power Generated by the Turbine in a Cylinderical Silo (Configuration 3A) with a Venturi Surrounding it (Figure 2(b)) and Turbine Efficiency

ΔT (°C)	Power (W)	Wind Energy (W)	Cp
0	768	1285	0.60
2	931	1285	0.72
4	1104	1285	0.86
6	1286	1285	1.00
8	1478	1285	1.15

From Table 3.6, it can be seen that the converging-diverging venturi surrounding the turbine has a significant effect in increasing the turbine power as well as the turbine efficiency which further increases with increase in ΔT. Also, it can be seen that larger diameter silo and turbine generate greater wind power as expected. Furthermore, like before for configuration 3A, the increase in ΔT increases the generated power as well as the turbine efficiency. Figures analogous to Figures 3.7 and 3.8 are not presented for this configuration because they are qualitatively similar in flow pattern and contours.

3.3.4 Configuration 4

Configuration 4A (Figure 3.2(b) with a Diffuser on the top): L= 70 ft, l = 50 ft, Dc = 11 ft, Da = 8 ft, ξ = 4 ft., η = 1.1 ft., δ = 0.4 ft, α = 20 deg, Dd = 1.5 Dc, ΔT = 0, 2, 4, and 6 deg. This configuration is similar to the configuration 3 with a diffuser on the top.

Configuration 4B (Figure 3.2(b) with a Diffuser on the top): L= 70 ft, l = 50 ft, Dc = 17.6 ft, Da = 12.8 ft, ξ = 4 ft., η = 1.76 ft., δ = 0.64 ft, α = 20 deg, Dd = 1.5 Dc, ΔT

= 0, 2, 4, and 6 deg. This configuration is similar to configuration 3 with a diffuser on the top.

Table 3.7 provides the power generated by the turbine and the turbine efficiency for the cylindrical silo geometry with a converging-diverging venturi (Figure 2(b)) surrounding the turbine and a diffuser at the top of the silo (configuration 4A), and class -3 wind speed of 5.6m/s for various values of ΔT.

Table 3.7 Power Generated by the Turbine in Cylinderical Silo with a Venturi Surrounding it (Figure 3.2(b)) and a Diffuser at the Top of the Silo (Configuration 4A) and Turbine Efficiency

ΔT (°C)	Power (W)	Wind Energy (W)	Cp
0	422	502	0.84
2	483	502	0.96
4	548	502	1.09
6	615	502	1.22
8	684	502	1.36

In Table 3.7, it should be noted that similar to Table 3.5, the available wind power in this case is less compared to that in Tables 3.1 and 3.3 because the wind turbine diameter in this case is Da = 8 ft. compared to that in configurations 1A and 2A where the wind turbine diameter Da = 10ft. The diameter of the turbine had to be less in this case because of the installation of the venturi. Nevertheless, Table 3.7 shows that diffuser at the top of the silo has a significant effect in increasing the turbine power as well as the turbine efficiency (compare the results with those in Table 3.5), which further increases with increase in ΔT.

Figure 3.9 shows the zoomed-in view of the computational domain near the actuator disk employed for the case of the turbine surrounded by a converging – diverging venturi inside the cylindrical silo with a diffuser on the top. The silo has been rotated by 90deg. clockwise in this figure. It shows the velocity vectors indicating the direction and movement of the flow for $\Delta T = 6$. Figure 3.10 shows the zoomed-in-view of static pressure contours in field (in Pascals) for $\Delta T = 6$.

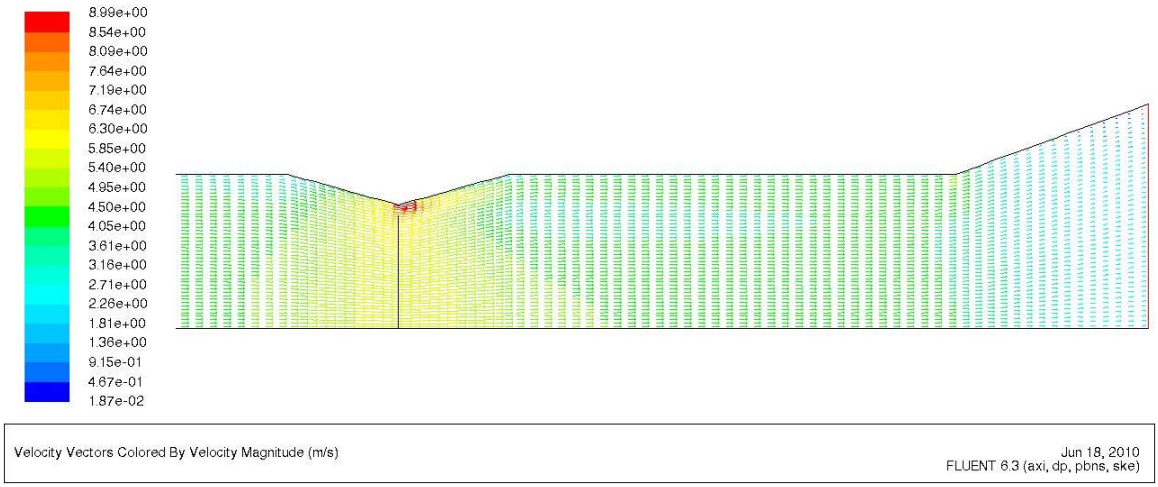

Figure 3.9 Zoomed-in-View of computational domain for flow inside the silo with a turbine surrounded by the venturi and a diffuser on the top: Velocity vectors and their magnitude. The silo has been rotated by 90deg. clockwise in this figure.

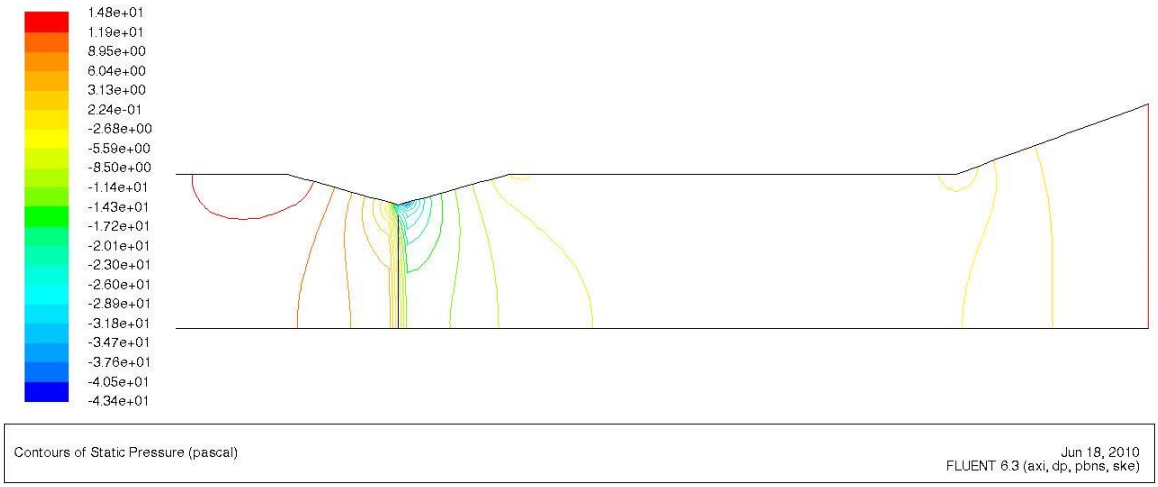

Figure 3.10 Zoomed-in-View of computational domain for flow inside the silo with a turbine surrounded by the venturi and a diffuser at the top: static pressure contours and their magnitude. The silo has been rotated by 90deg. clockwise in this figure.

Table 3.8 provides the power generated by the turbine and the turbine efficiency for the cylindrical silo geometry with a converging-diverging venturi (Figure 3.2(b)) surrounding the turbine and a diffuser at the top of the silo (configuration 4B), and class -3 wind speed of 5.6m/s for various values of ΔT.

Table 3.8 Power Generated by the Turbine in a Cylinderical Silo with a Venturi Surrounding it (Figure 3.2(b)) and a Diffuser at the Top of the Silo (Configuration 4B) and Turbine Efficiency

ΔT (°C)	Power (W)	Wind Energy (W)	Cp
0	1108	1285	0.86
2	1369	1285	1.06
4	1649	1285	1.28
6	1946	1285	1.51
8	2260	1285	1.76

In Table 3.8, it should be noted that similar to Table 3.6, the available wind power in this case is less compared to that in Tables 3.2 and 3.4 because the wind turbine diameter in this case is Da = 12.8 ft. compared to that in configurations 1B and 2B where the wind turbine diameter Da = 16ft. The diameter of the turbine had to be less in this case because of the installation of the venturi. Nevertheless, Table 3.8 shows that diffuser at the top of the silo has a significant effect in increasing the turbine power as well as the turbine efficiency (compare the results with those in Table 3.6), which further increases with increase in ΔT. Figures analogous to Figures 3.9 and 3.10 are not presented for this configuration because they are qualitatively similar in flow pattern and contours.

3.4 Diffuser Optimization

From the modeling of the four configurations in the previous section, the diffuser element was shown to be the dominant factor increasing the power output of the solar chimney. It was however modeled only as a simple cone shape. This presented an

opportunity to optimize the shape of the diffuser for the greatest power output by the included wind turbine. Using a modified version of the GA used for the work presented in Chapter 2 and the same CFD software, Configurations 2A and 2B were optimized for the five temperature differentials modeled previously.

3.4.1 CFD and Genetic Algorithm Implementation

The modeling and optimization techniques used for the solar chimney diffuser derive from the methods previously explained. The CFD model was the same as in Configuration 2, except that the diffuser shape was defined by four control points. The important elements of the model remain the actuator disk's pressure discontinuity and the Boussinesq Approximation that introduces buoyancy due to the temperature differential with the exterior. The first control point was fixed at the top of the straight silo side where the conic diffuser began in the previous work. To achieve results that were directly comparable to the non-optimized configuration, the optimized diffuser was limited to the same maximum radius and length as the diffusers in Configurations 2A and 2B. The genetic algorithm functions in the same manner as described in section 2.2, but due to the smaller number of control points and less complex shape, it only required 150 generations to converge on a solution.

3.5 Results

The optimization of the solar chimney diffuser worked effectively and resulted in improvement in the power coefficient on the order of 7% for all cases. The specifics are elucidated below after an explanation of the convergence history of one case.

3.5.1 Genetic Algorithm Convergence

Figure 3.11 shows the "evolution" of the Configuration 2A diffuser shape with a 4K temperature difference in the implementation of the GA. The initial randomly generated curves have poor fitness, but within a few generations, good diffuser shapes emerge. The third inset figure (clockwise from bottom left) is from generation 5. The next significant improvement comes at generation 55 shown in the fourth inset figure. This has a power coefficient of 0.9916, while the best result shown in the fifth inset from generation 148 has a power coefficient of 0.9945. The improvement is less than 1 percent, but it makes up the majority of the processing time required by the GA.

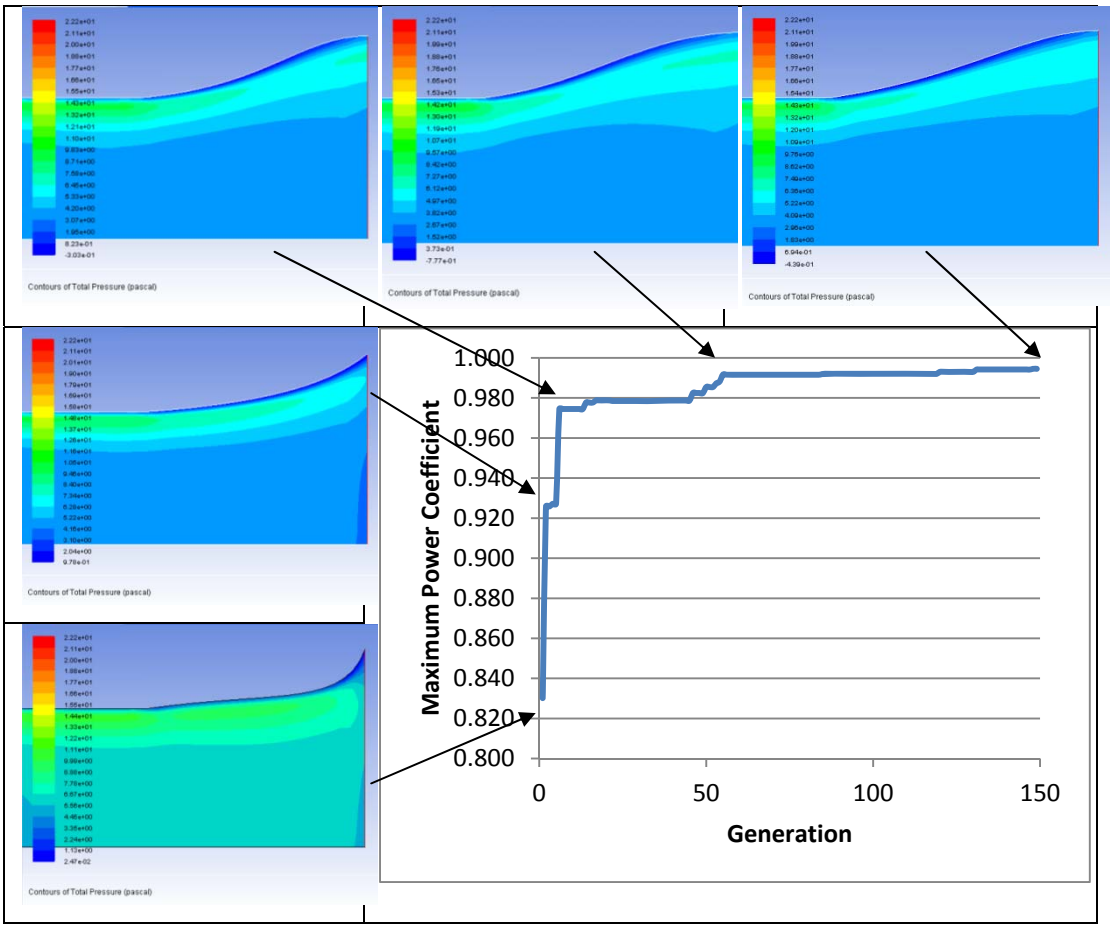

Figure 3.11 Convergence history of the GA for optimized diffuser Configuration 2A with 4K temperature differential. The insets show total pressure contours and their magnitude in Zoomed-in-Views of the computational domain for flow inside the diffuser on top of the silo. The geometries are the individuals with the highest fitness value at the indicated generation. The silo has been rotated by 90 deg. clockwise in these figures.

3.5.2 Data and Figures Related to Diffuser Optimization

The specific results of the diffuser optimization are tabulated in Tables 3.9 and 3.10. For comparison, the power generation and power coefficients of the cone-diffuser are reproduced to illustrate the increases due to optimization.

Table 3.9 Configuration 2A

ΔT (deg.)	Power Generation Cone-Diffuser (W)	Power Generation Optimized Diffuser (W)	Cp (Cone-Diffuser)	Cp (Optimized Diffuser)
0	560	600	0.71	0.76
2	642	688	0.82	0.88
4	727	781	0.93	0.99
6	816	877	1.04	1.12
8	909	987	1.16	1.26

Table 3.10 Configuration 2B

ΔT (deg.)	Power Generation Cone-Diffuser (W)	Power Generation Optimized Diffuser (W)	Cp (Cone-Diffuser)	Cp (Optimized Diffuser)
0	1463	1557	0.73	0.77
2	1815	1938	0.90	0.96
4	2176	2342	1.08	1.17
6	2580	2768	1.28	1.38
8	2984	3237	1.48	1.61

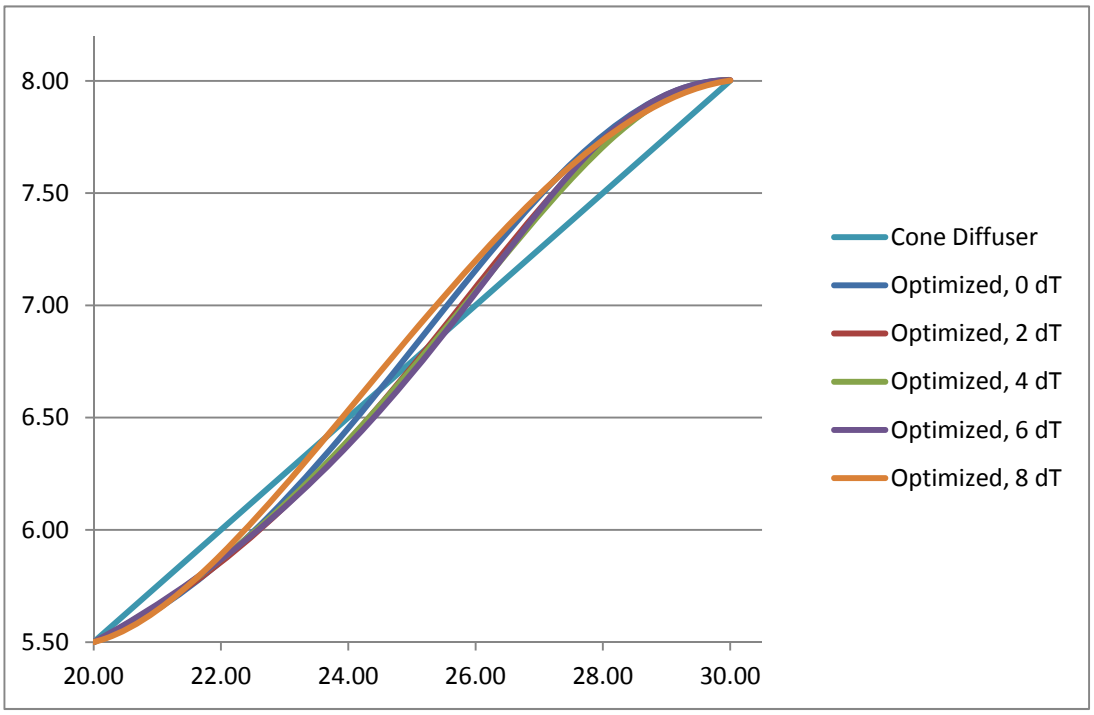

Figure 3.12 GA optimized diffuser shapes for Configuration 2A compared to the original cone diffuser for various temperature differentials.

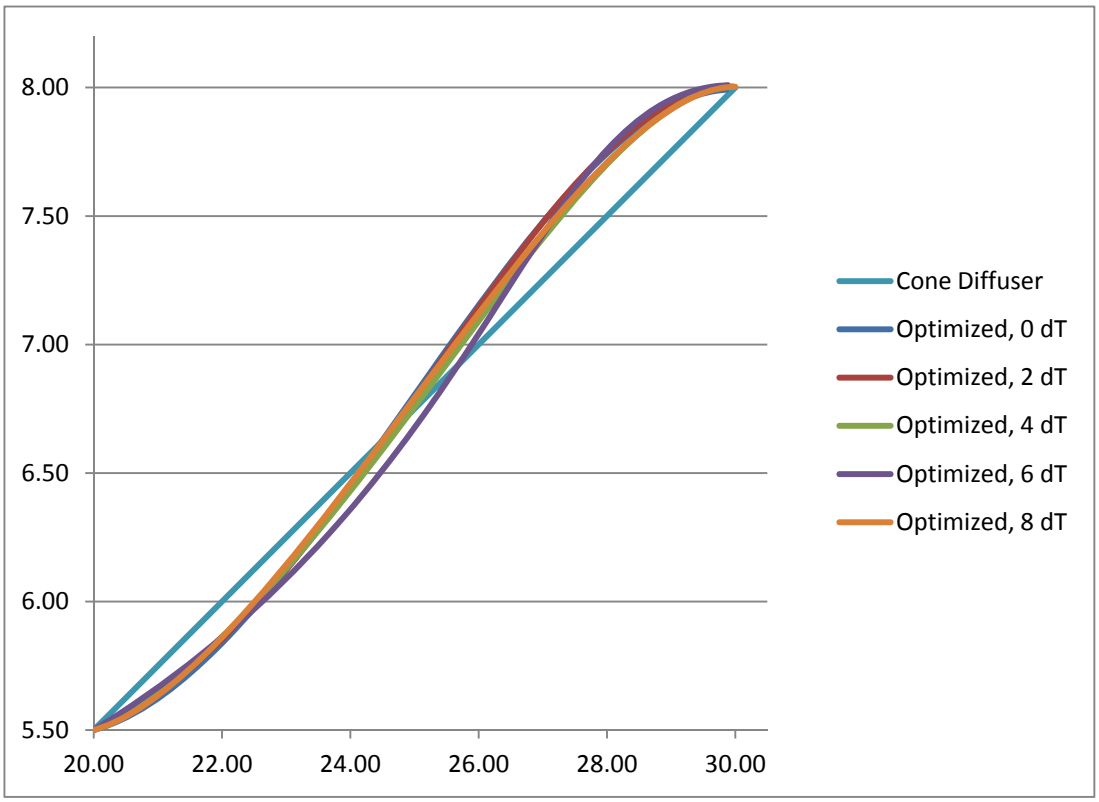

Figure 3.13 GA optimized diffuser shapes for Configuration 2B compared to the original cone diffuser for various temperature differentials.

3.6 Future Work

There are many issues that require further investigation. (a) *The proper placement of stave-openings to draw wind inside the silo*: If the stave-openings are very close to the ground, the wind velocity becomes very small because of the atmospheric boundary layer. Even if the stave-openings are at a reasonably higher level from the ground, their placement at a proper distance from the turbine is critical since the wind stream will separate away from the walls of the silo near the entrance of the stave-openings. (b) *The increase in the updraft velocity of the wind because of temperature stratification*: For an average height of about 70ft, the temperature differential ΔT between the bottom and top of the silo is very

small; it is 0.025deg F (based on 0.00357deg.F/ft). It is not enough to create any significant updraft and increase in wind velocity. The only way to increase the wind velocity upward is to provide heat at the lower level near the stave-openings. The calculations show that that larger ΔT increases the wind velocity and therefore the turbine power. However, the energy needed for heating the ground level air for a desired value of ΔT should be estimated and compared with the additional energy generated by the wind turbine (due to increase in wind speed) in order to determine the benefit of ground level heating. (c) *The venturi shape* (the axial length and boundary shape) should be optimized by using an optimization algorithm (e.g. a genetic algorithm) to extract maximum power from the wind. (d) *The diffuser shape* (the axial length and boundary shape) should be optimized by using an optimization algorithm (e.g. a genetic algorithm) to extract additional power from the wind while accounting for the cross-flow of the free stream wind at the exit. (e) *The full scale 3D CFD simulations* should be performed by including the knowledge from items (a) – (d) for a 3-bladed rotor with optimized blade designs for various blade tip to wind speed ratios ($\lambda = \Omega D/2V$, where Ω = rotational speed of the rotor, D is the rotor diameter and V is the wind velocity facing the rotor).

Addressing these four issues will help in determining the best possible configuration for generating maximum wind power from wind turbines enclosed by silos. Once this configuration is determined, the cost estimates for initial investment and return on investment (ROI) should be conducted taking into account the number of such installations as well as installation and maintenance issues.

Chapter 4

4. Conclusions

In this thesis, we have investigated the potential of shrouded wind turbines for increased power generation, compared to bare turbines, by CFD simulations. Two configurations for ducted turbines were considered for numerical simulations and optimization. Both of the configurations were analyzed by employing a computational fluid dynamics flow solver, which solves the Reynolds-Averaged Navier-Stokes equations in conjunction with a two equation k-epsilon turbulence model. The turbine was modeled as an actuator disk neglecting the rotational effects. The first configuration considered shrouds for standard horizontal axis wind turbines. Several turbine diameters, wind speeds, and shroud sizes were considered. It was found that shrouding can almost double the power that would be generated by a bare turbine. The second configuration considered the potential of converting abandoned or unused farm silos into solar chimneys with low cost shape modification of the chimney to further augment the power generation. The most effective, simple and low cost shape modification was found to be a diffuser added to the top of the silo, which increased the power output by nearly 50%. Increasing the buoyancy effect by heating the air at the base of the silo further increased the power output by a significant amount (as much as 50%). The diffuser shapes for both the configurations were optimized using a genetic algorithm. In summary, the computations showed that the shrouded turbines can generate greater power than that generated by the bare turbines and therefore should be considered for small and medium size turbines. Further investigation is needed in the overall economic benefit considering the initial investment, maintenance

and life cycle costs. The technical feasibility of shrouding a turbine and the structural integrity of a shrouded turbine are also major considerations.

A. Appendices

A.1 Optimized Shrouded Wind Turbines: Table of Results

Table A.1 Results for optimized shrouded HAWT.

Case	Vt:	Ct:	Fan Coeff.	Power (W)	Cp	Cp_exit
1a	6.9061	0.7126	0.2984	717	0.862	0.337
1b	8.2067	0.7202	0.3209	1294	0.844	0.330
1c	6.5517	0.6724	0.3128	642	0.772	0.387
1d	7.5762	0.8054	0.2802	889	1.069	0.356
1e	7.7299	0.7644	0.2555	861	1.035	0.345
1f	6.8397	0.7574	0.3233	755	0.907	0.306
1g	6.0161	0.6644	0.3666	582	0.700	0.353
2a	6.9236	0.7159	0.2983	1849	0.868	0.339
2b	8.3053	0.7132	0.3103	3321	0.846	0.334
3a	6.3712	0.7806	0.3840	65	0.871	0.340
3b	8.1974	0.7218	0.3224	117	0.845	0.330
4a	6.5091	0.6694	0.3155	2540	0.763	0.382
4b	7.9329	0.6673	0.3183	4637	0.756	0.379

A.2 Optimized Shroud Figures with Control Points

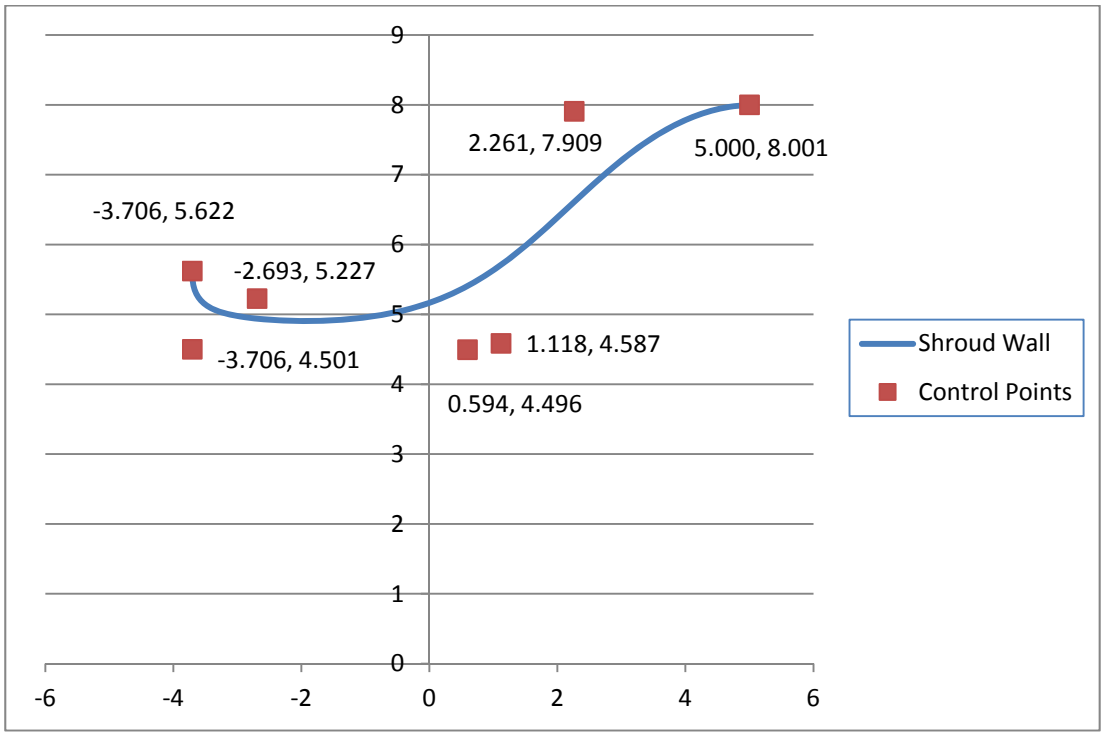

Figure A.1 Case 1a: optimized HAWT shroud with control points.

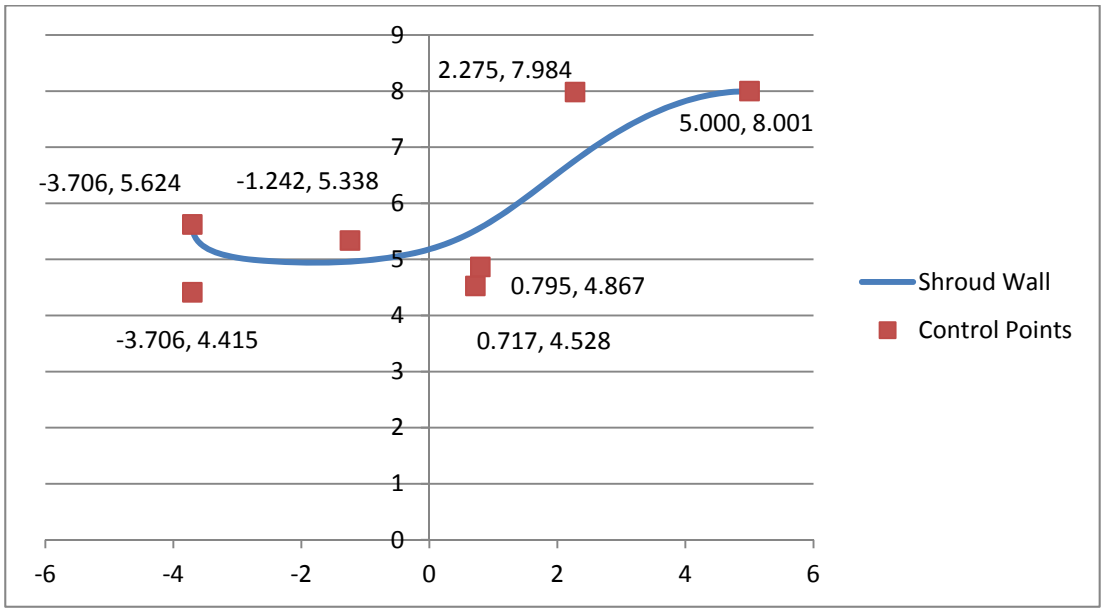

Figure A.2 Case 1b: optimized HAWT shroud with control points.

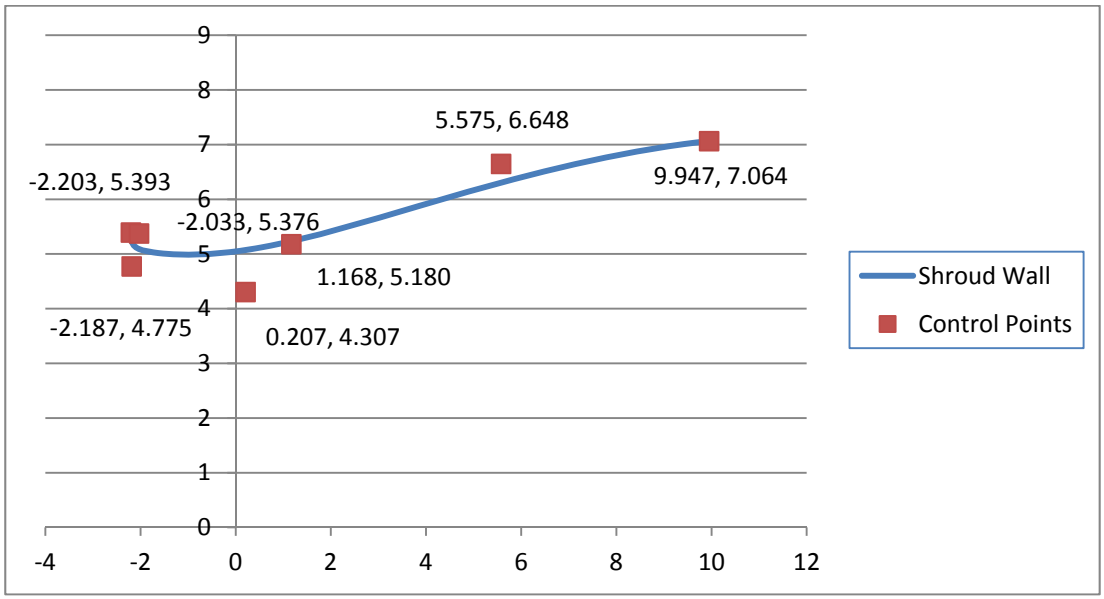

Figure A.3 Case 1c: optimized HAWT shroud with control points.

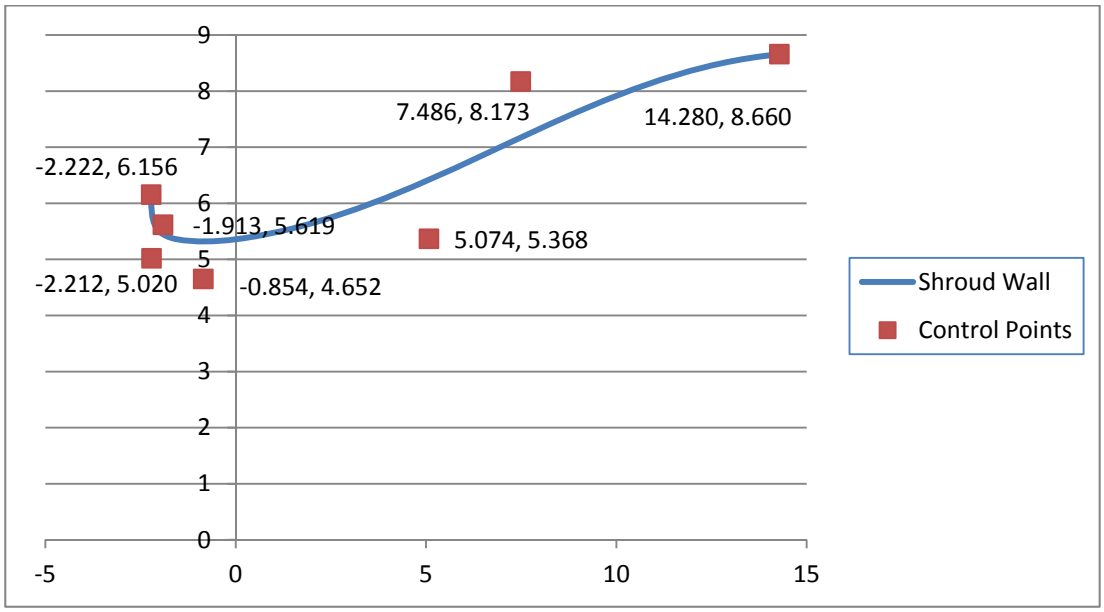

Figure A.4 Case 1d: optimized HAWT shroud with control points.

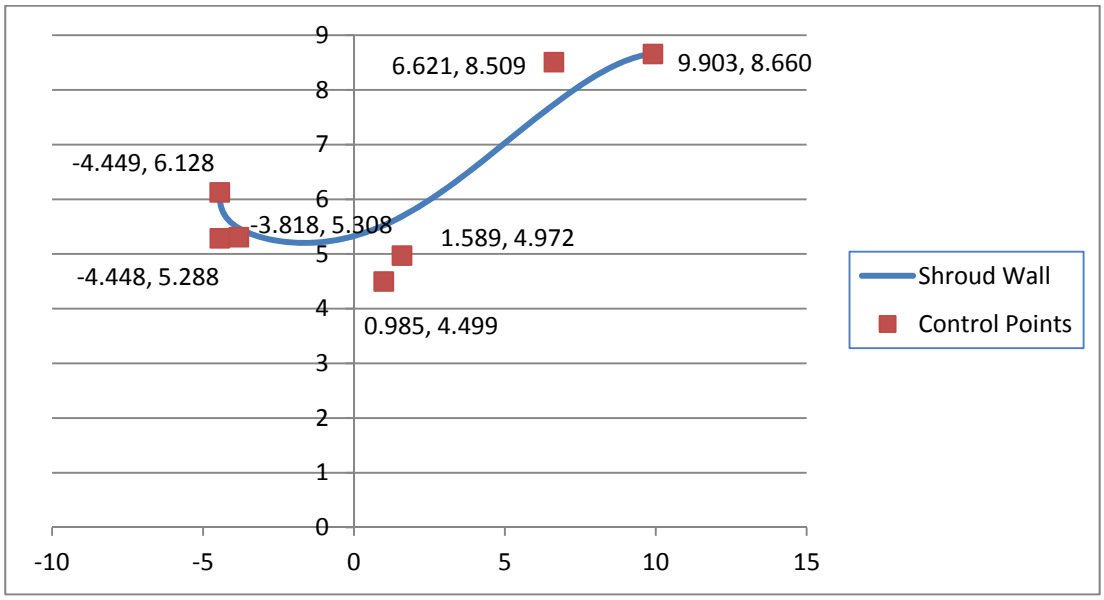

Figure A.5 Case 1e: optimized HAWT shroud with control points.

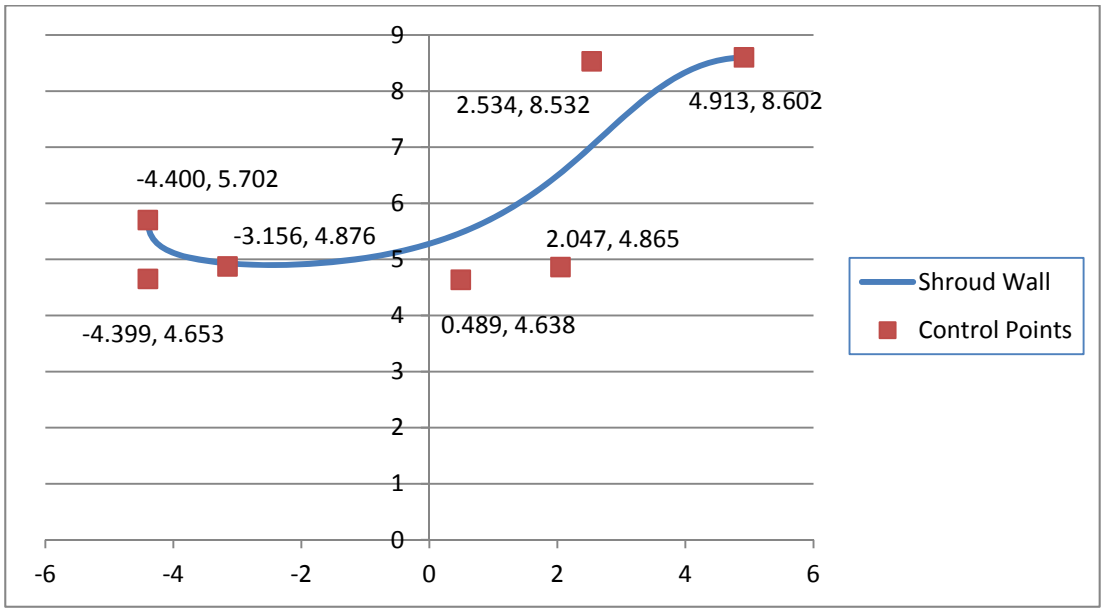

Figure A.6 Case 1f: optimized HAWT shroud with control points.

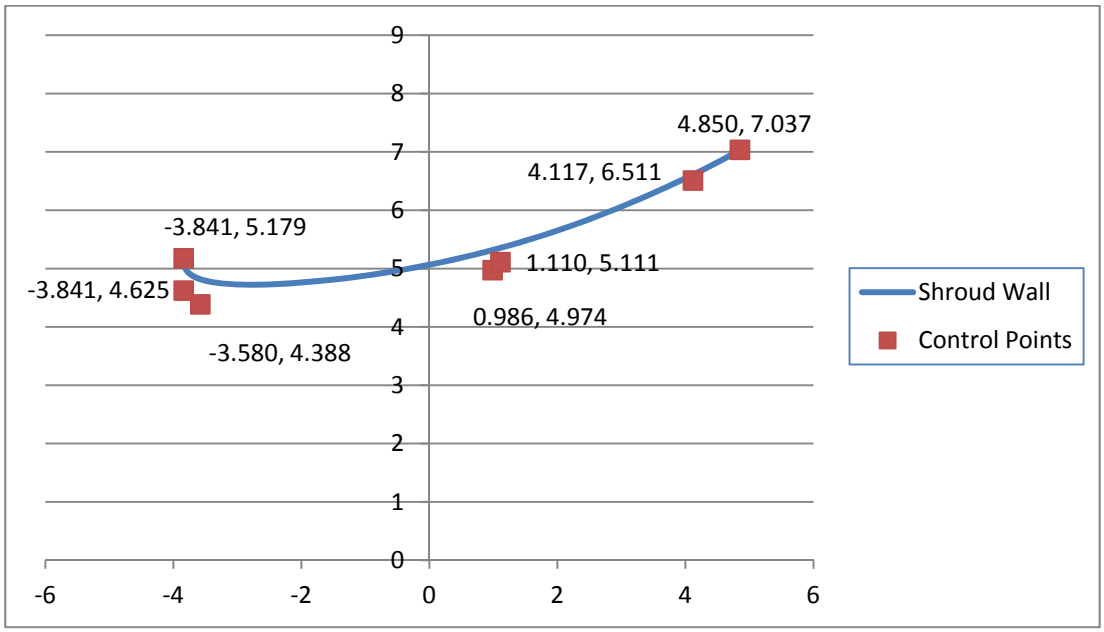

Figure A.7 Case 1g: optimized HAWT shroud with control points.

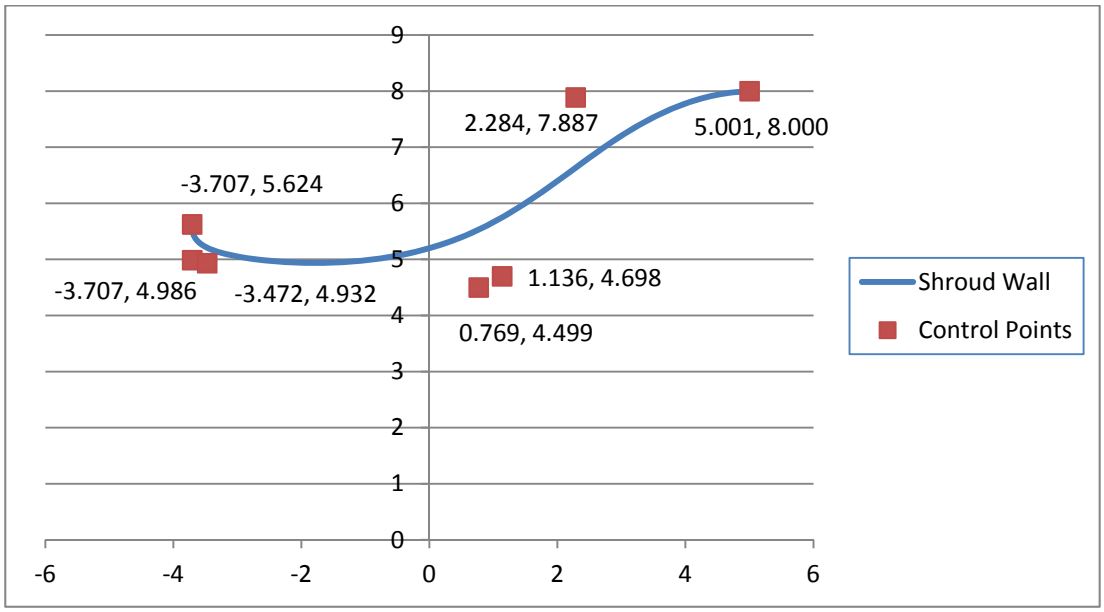

Figure A.8 Case 2a: optimized HAWT shroud with control points.

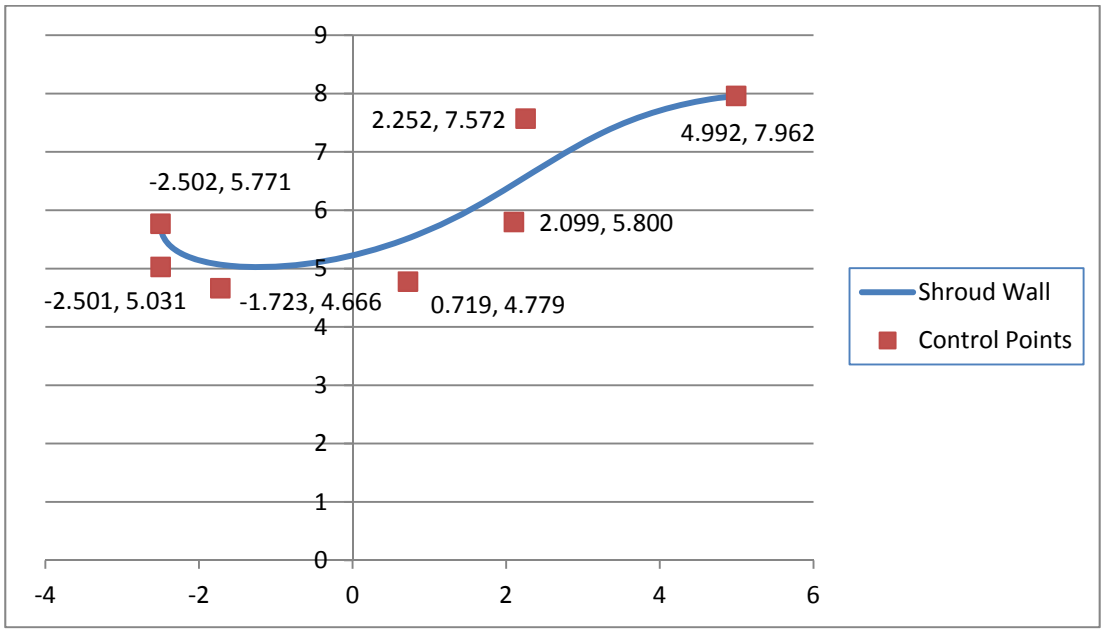

Figure A.9 Case 2b: optimized HAWT shroud with control points.

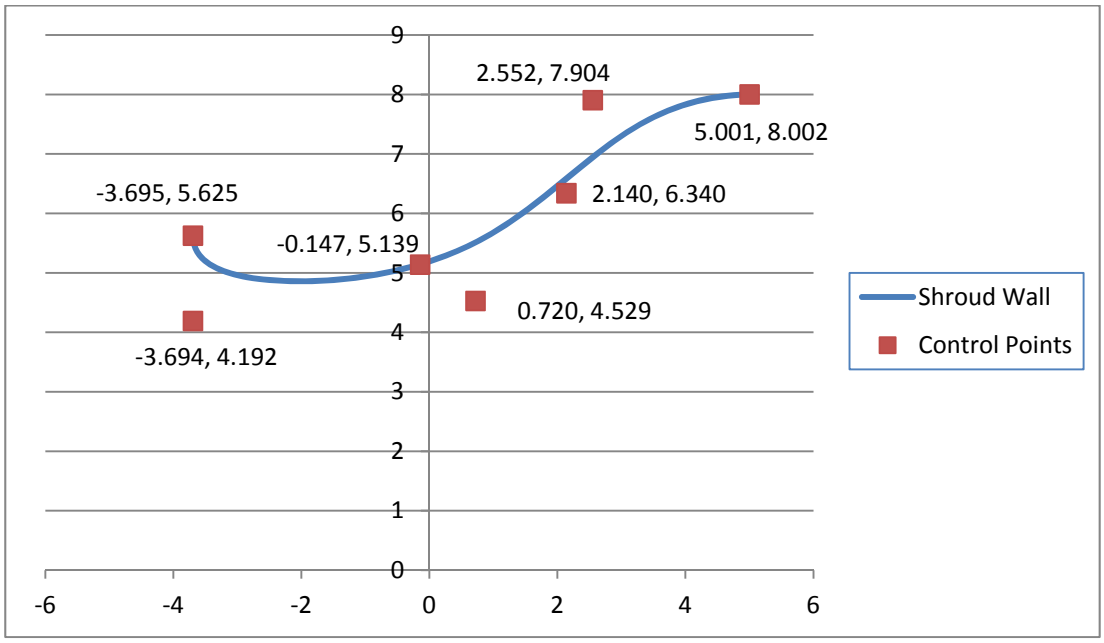

Figure A.10 Case 3a: optimized HAWT shroud with control points.

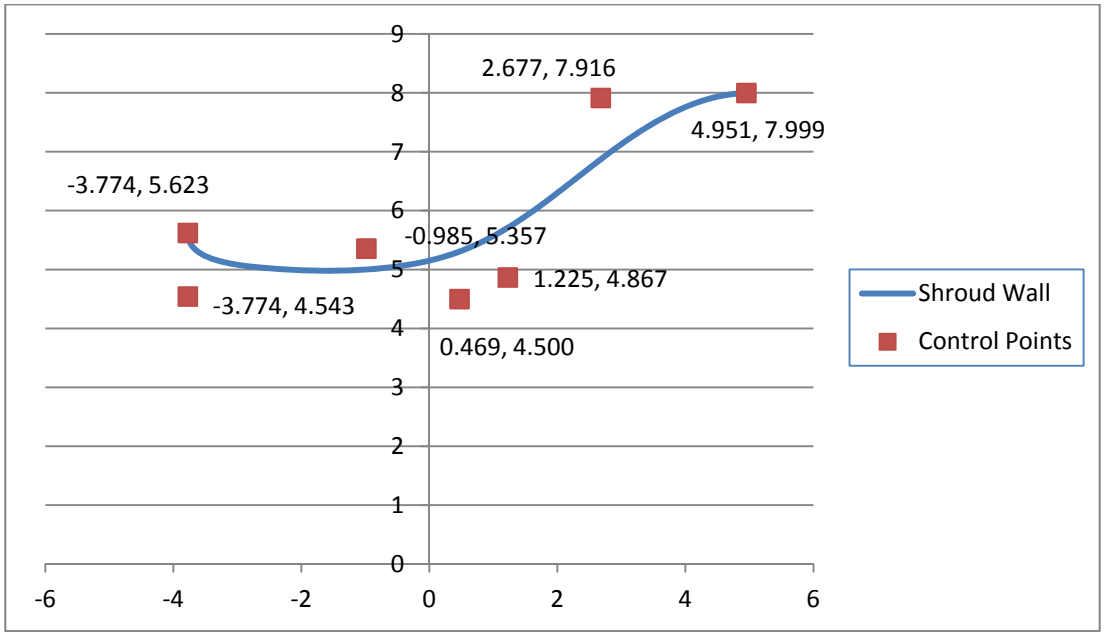

Figure A.11 Case 3b: optimized HAWT shroud with control points.

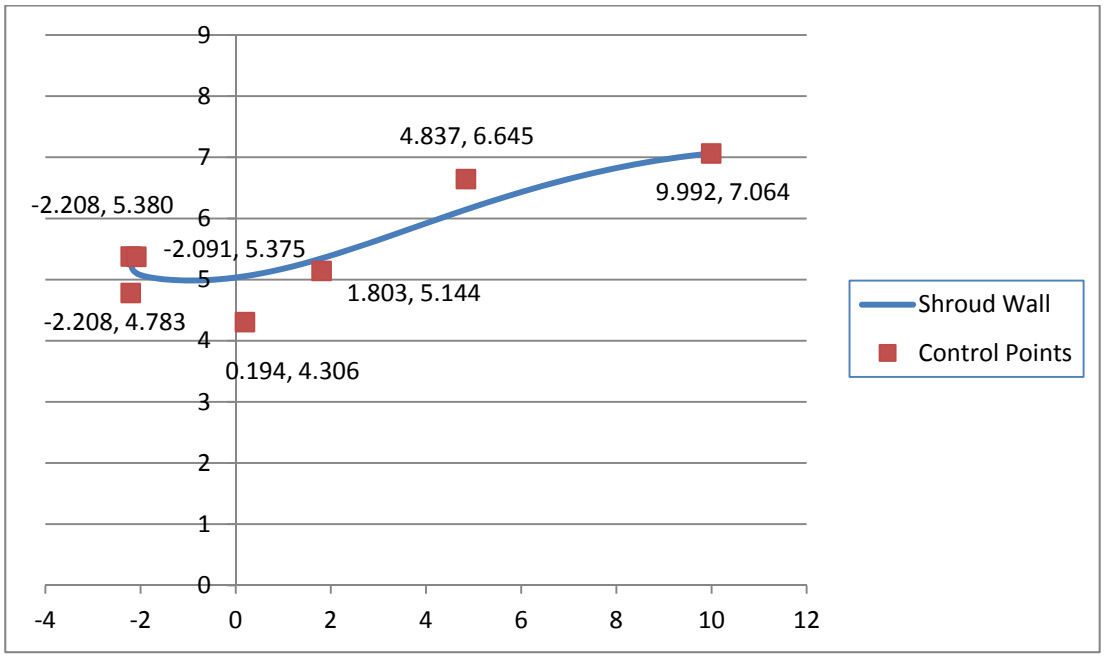

Figure A.12 Case 4a: optimized HAWT shroud with control points.

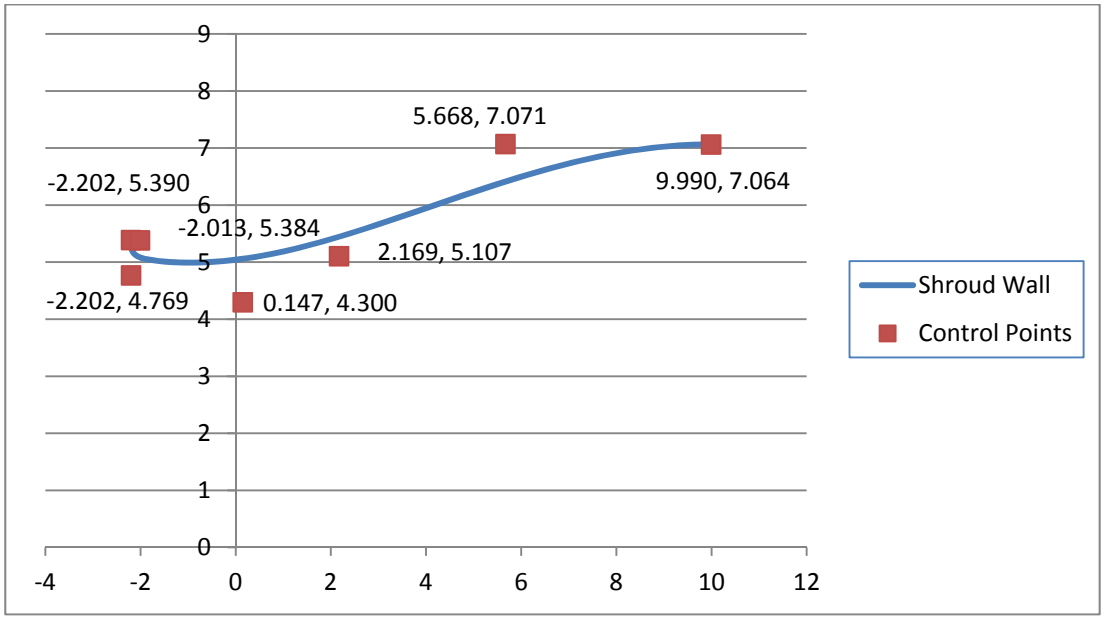

Figure A.13 Case 4b: optimized HAWT shroud with control points.

A.3 Genetic Algorithm Code with Ancillary Files

This is the complete text of the Genetic Algorithm Java files, as well as the text of the external files called in the process of running the code. Section headings are the file names.

A.3.1 wing.java

```
package wing;


	public class wing {
		private int genSize, numGens, E, existingGenerations;
		private double removePercentage, mutRate;
		public static Airfoil bestAirfoil;
		private generation one;
		double t;


		public wing(int genSize, int numGens, double
removePercentage, double mutRate){
			this.genSize = genSize;
			this.numGens = numGens;
			this.removePercentage = removePercentage;
			E = (int) Math.round(genSize *
this.removePercentage);
			this.mutRate = mutRate;

		}

		public void runOptimization(){
			//one = manyIndividuals();
			one = existingIndividuals(1);
			bestAirfoil = new Airfoil();

			for (int i=1; i<numGens; i++){
				one.determineFitness(i);

				/**Find airfoil with highest coefficient of
lift */
				System.out.println("***** Generation " + i +
"******");
				bestAirfoil = one.getBestAirfoil();
				System.out.println("Best airfoil Cp = " +
bestAirfoil.getFitness());
```

```
                        /**Create new generations to find best
airfoil*/
                        advanceGen();
                  }

            }

      public Airfoil generateIndividual(){

            Airfoil airfoil = new Airfoil();

            double X6=AirfoilModifier.minX6+Math.random() *
(AirfoilModifier.maxX6-AirfoilModifier.minX6);
            double X5=AirfoilModifier.minX5+Math.random() * (X6-
AirfoilModifier.minX5);
            double X4=AirfoilModifier.minX4+Math.random() * (X5-
AirfoilModifier.minX4);
            double X3=AirfoilModifier.minX3+Math.random() * (X4-
AirfoilModifier.minX3);

            double X7=X6+                   Math.random() *
(AirfoilModifier.maxX7-X6);
            double X8=X7+                   Math.random() *
(AirfoilModifier.maxX8-X7);
            double X9=X8+                   Math.random() *
(AirfoilModifier.maxX9-X8);

            double Y3=AirfoilModifier.minY3+Math.random() *
(AirfoilModifier.maxY3-AirfoilModifier.minY3);
            double Y4=AirfoilModifier.minY4+Math.random() *
(AirfoilModifier.maxY4-AirfoilModifier.minY4);
            double Y5=AirfoilModifier.minY5+Math.random() *
(AirfoilModifier.maxY5-AirfoilModifier.minY5);
            double Y6=AirfoilModifier.minY6+Math.random() *
(AirfoilModifier.maxY6-AirfoilModifier.minY6);
            double Y7=AirfoilModifier.minY7+Math.random() *
(AirfoilModifier.maxY7-AirfoilModifier.minY7);
            double Y8=AirfoilModifier.minY8+Math.random() *
(AirfoilModifier.maxY8-AirfoilModifier.minY8);
            double Y9=AirfoilModifier.minY9+Math.random() *
(AirfoilModifier.maxY9-AirfoilModifier.minY9);

            Airfoil airfoilgen = new
Airfoil(X3,Y3,X4,Y4,X5,Y5,X6,Y6,X7,Y7,X8,Y8,X9,Y9);
            AirfoilModifier.modAirfoil(airfoilgen);
            airfoil = airfoilgen;

            return airfoil;
         }


      private generation manyIndividuals(){
```

```
            //new generation of airfoils
            generation airfoils = new generation(genSize);

            //loop to generate genSize airfoils
            for(int i=0;i<genSize;i++){
                  airfoils.addAirfoil(generateIndividual());
            }

            return airfoils;
      }

      /**create generation with existing wings**/
      private generation existingIndividuals(int
existingGenerations){
            //Input the generation
            this.existingGenerations = existingGenerations;

            //new generation of wings
            generation airfoils = new generation(genSize);

            //add existing wings manually
            airfoils.addAirfoil(new Airfoil());
            airfoils.addAirfoil(new Airfoil());
            airfoils.addAirfoil(new Airfoil());
            airfoils.addAirfoil(new Airfoil());
            airfoils.addAirfoil(new Airfoil());
            airfoils.addAirfoil(new Airfoil());
            airfoils.addAirfoil(new Airfoil());
            airfoils.addAirfoil(new Airfoil());
            airfoils.addAirfoil(new Airfoil());
            airfoils.addAirfoil(new Airfoil());

            airfoils.addAirfoil(FromRecord(-2.203148738270498,
5.393070851734786, -2.187450061351378, 4.774690986521157, -
2.033185488350599, 5.375622047373333, 0.2071159316701668,
4.30693343712174, 1.1675404372777267, 5.180456711640752,
5.575394178619769, 6.6483017687088175, 9.947163978226866,
7.064226770987885));
            airfoils.addAirfoil(FromRecord(-
2.2031563688838394,5.38974204644828,-
2.187519415256461,4.765324729035233,-
2.032589081851058,5.375624069892875,0.20688284397129159,4.3079542981
57818,1.1285085316013237,5.181228539118404,6.0919799017645575,6.6481
80476693175,9.9471154690912,7.064225956209522));
            airfoils.addAirfoil(FromRecord(-
2.203242972318233,5.428435754003545,-
2.188946662658091,4.604890879280909,-
2.0333691081836083,5.375020138070308,0.18670313501311064,4.338337757
39219,2.0413081291653565,5.170381945315839,5.481685780565558,6.66198
243600454,9.943601873269163,7.064199988243365));
            airfoils.addAirfoil(FromRecord(-
2.203292492284962,5.431043102577696,-
2.189234769990262,4.568255531386719,-
2.0333597700161934,5.374848000148967,0.1676513091246766,4.3514245564
```

```
236465,2.2299930120556,5.161370564983122,4.339165577643721,6.6685315
885925185,9.943385743272596,7.06418144283336));
            airfoils.addAirfoil(FromRecord(-
2.203139636591628,5.44889706697271,-
2.188055966731389,4.610273765570841,-
2.0353139875454733,5.375566255298646,0.19562973119910243,4.326836201
697043,1.909620439894519,5.180671986463295,5.7233578223442425,6.6613
15078035779,9.943343999324249,7.064228186980254));
            airfoils.addAirfoil(FromRecord(-
2.2031765431713892,5.396705637675992,-
2.1874099772398274,4.732792807576367,-
2.036036217855667,5.375659926189331,0.2057733558680029,4.31612789687
25005,0.7377918367789627,5.179930277102026,6.312642602467313,6.65890
31113421,9.946514730895467,7.064225583498222));
            airfoils.addAirfoil(FromRecord(-
2.2031704188617858,5.384780920287789,-
2.187730676813248,4.755287684620443,-
2.031772219468423,5.375627016797401,0.20604102424041873,4.3104138896
22369,0.9196178284377317,5.182528953205678,6.67000510253117,6.647289
6671883,9.94704965965121,7.0642239092848085));
            airfoils.addAirfoil(FromRecord(-
2.203179101171859,5.3864499679385744,-
2.187866144622038,4.763366721000551,-
2.028965610797605,5.375827135265614,0.20263297829501306,4.3093444987
88545,0.6656225169940884,5.178866525861145,6.72903414911511,6.643479
935572796,9.947107658312234,7.064225195401249));
            airfoils.addAirfoil(FromRecord(-
2.2031836399469773,5.389258883922361,-
2.190455421742542,4.769493146820115,-
2.028763066478957,5.3760501342603355,0.19969930819437232,4.309831528
6345095,0.5131890481462964,5.176460218382226,6.723057870108455,6.642
657034026507,9.947491599837884,7.064228815380099));
            airfoils.addAirfoil(FromRecord(-
2.2031561361060077,5.389638787450879,-
2.1883829390327487,4.740473422102478,-
2.0329156073865047,5.376058609198227,0.20374769247375832,4.321783712
62854,0.9370794692122187,5.182079140667373,6.578434495018177,6.65234
8109639632,9.946286188354772,7.064224247053876));

            return airfoils;
      }



      public void advanceGen(){
            System.out.println("Begin advance gen");
            generation nextGen = new generation(genSize);

            /**Create E number of new airfoils by using coordinates
from two random airfoils from the previous generation*/
            int count = 0;

            while(count<E){
```

```
                  Airfoil airfoil1 =
one.getAirfoil((int)Math.floor(Math.random()*genSize));
                  Airfoil airfoil2 =
one.getAirfoil((int)Math.floor(Math.random()*genSize));
                  if(airfoil1 != airfoil2){
                        Airfoil
airfoilCrossover=crossover(airfoil1,airfoil2);
                        nextGen.addAirfoil(airfoilCrossover);
                        count++;

                        }
                  }

                  /**Check to make sure it works*/
                  System.out.print("***Next generation of
airfoils***");
                  nextGen.outputAirfoils();

            /**Remove E number of airfoils from the previous
generation with the highest Cp*/
            naturalSelection();

            /**Add surviving airfoils to nextGen of airfoils*/
            for(int i=0;i<one.getAirfoilVectorSize();i++){
                  nextGen.addAirfoil(one.getAirfoil(i));
            }


            /**Mutate*/
            mutate(nextGen);

            /**Check to make sure it works*/
            System.out.println("***Next generation of airfoils with
survivors (and mutants)***" + "\r");
            nextGen.outputAirfoils();

            /**Original generation becomes nextGen*/
            one = nextGen;


      }

      /**Take coordinates of two airfoils and combine them to get a
new airfoil*/
      private Airfoil crossover(Airfoil airfoil1, Airfoil airfoil2){

            System.out.println("Begin crossover");
            //double x0_1,x0_2;
            double x3_1,x3_2;
            double y3_1,y3_2;
            double x4_1,x4_2;
            double y4_1,y4_2;
```

```
double x5_1,x5_2;
double y5_1,y5_2;
double x6_1,x6_2;
double y6_1,y6_2;
double x7_1,x7_2;
double y7_1,y7_2;
double x8_1,x8_2;
double y8_1,y8_2;
double x9_1,x9_2;
double y9_1,y9_2;


if(airfoil1.getFitness() > airfoil2.getFitness()){
	x3_2 = airfoil1.X3;
	y3_2 = airfoil1.Y3;
	x4_2 = airfoil1.X4;
	y4_2 = airfoil1.Y4;
	x5_2 = airfoil1.X5;
	y5_2 = airfoil1.Y5;
	x6_2 = airfoil1.X6;
	y6_2 = airfoil1.Y6;
	x7_2 = airfoil1.X7;
	y7_2 = airfoil1.Y7;
	x8_2 = airfoil1.X8;
	y8_2 = airfoil1.Y8;
	x9_2 = airfoil1.X9;
	y9_2 = airfoil1.Y9;

	x3_1 = airfoil2.X3;
	y3_1 = airfoil2.Y3;
	x4_1 = airfoil2.X4;
	y4_1 = airfoil2.Y4;
	x5_1 = airfoil2.X5;
	y5_1 = airfoil2.Y5;
	x6_1 = airfoil2.X6;
	y6_1 = airfoil2.Y6;
	x7_1 = airfoil2.X7;
	y7_1 = airfoil2.Y7;
	x8_1 = airfoil2.X8;
	y8_1 = airfoil2.Y8;
	x9_1 = airfoil2.X9;
	y9_1 = airfoil2.Y9;


} else {
	x3_1 = airfoil1.X3;
	y3_1 = airfoil1.Y3;
	x4_1 = airfoil1.X4;
	y4_1 = airfoil1.Y4;
	x5_1 = airfoil1.X5;
	y5_1 = airfoil1.Y5;
	x6_1 = airfoil1.X6;
	y6_1 = airfoil1.Y6;
```

```
                x7_1 = airfoil1.X7;
                y7_1 = airfoil1.Y7;
                x8_1 = airfoil1.X8;
                y8_1 = airfoil1.Y8;
                x9_1 = airfoil1.X9;
                y9_1 = airfoil1.Y9;

                x3_2 = airfoil2.X3;
                y3_2 = airfoil2.Y3;
                x4_2 = airfoil2.X4;
                y4_2 = airfoil2.Y4;
                x5_2 = airfoil2.X5;
                y5_2 = airfoil2.Y5;
                x6_2 = airfoil2.X6;
                y6_2 = airfoil2.Y6;
                x7_2 = airfoil2.X7;
                y7_2 = airfoil2.Y7;
                x8_2 = airfoil2.X8;
                y8_2 = airfoil2.Y8;
                x9_2 = airfoil2.X9;
                y9_2 = airfoil2.Y9;
            }

            /**Create new airfoils coordinates with crossover of two
old airfoils biasing towards airfoil with smaller Cp*/
            //double x0 = Math.random()*(x0_2-x0_1) + x0_2;
            double x3 = Math.random()*(x3_2-x3_1) + x3_2;
            double y3 = Math.random()*(y3_2-y3_1) + y3_2;
            double x4 = Math.random()*(x4_2-x4_1) + x4_2;
            double y4 = Math.random()*(y4_2-y4_1) + y4_2;
            double x5 = Math.random()*(x5_2-x5_1) + x5_2;
            double y5 = Math.random()*(y5_2-y5_1) + y5_2;
            double x6 = Math.random()*(x6_2-x6_1) + x6_2;
            double y6 = Math.random()*(y6_2-y6_1) + y6_2;
            double x7 = Math.random()*(x7_2-x7_1) + x7_2;
            double y7 = Math.random()*(y7_2-y7_1) + y7_2;
            double x8 = Math.random()*(x8_2-x8_1) + x8_2;
            double y8 = Math.random()*(y8_2-y8_1) + y8_2;
            double x9 = Math.random()*(x9_2-x9_1) + x9_2;
            double y9 = Math.random()*(y9_2-y9_1) + y9_2;


            Airfoil airfoil = new
Airfoil(x3,y3,x4,y4,x5,y5,x6,y6,x7,y7,x8,y8,x9,y9);
            AirfoilModifier.modAirfoil(airfoil);

            return airfoil;

      }

      private void naturalSelection(){
            System.out.println("Begin natural selection");
            BubbleSorter.sort(one);
            one.removeAirfoils(genSize-E);
```

```
	}

	private void mutate(generation nextGen){
		for(int i=0;i<nextGen.getGenSize();i++){
			if(Math.random() <= mutRate) {

				Airfoil airfoiladd = new Airfoil();
				Airfoil airfoilremove = new Airfoil();


				Airfoil airfoil = nextGen.getAirfoil(i);

				double
x6=AirfoilModifier.minX6+Math.random() * (AirfoilModifier.maxX6-
AirfoilModifier.minX6);
				double
x5=AirfoilModifier.minX5+Math.random() * (x6-AirfoilModifier.minX5);
				double
x4=AirfoilModifier.minX4+Math.random() * (x5-AirfoilModifier.minX4);
				double
x3=AirfoilModifier.minX3+Math.random() * (x4-AirfoilModifier.minX3);

				double x7=x6+
Math.random() * (AirfoilModifier.maxX7-x6);
				double x8=x7+
Math.random() * (AirfoilModifier.maxX8-x7);
				double x9=x8+
Math.random() * (AirfoilModifier.maxX9-x8);

				double
y3=AirfoilModifier.minY3+Math.random() * (AirfoilModifier.maxY3-
AirfoilModifier.minY3);
				double
y4=AirfoilModifier.minY4+Math.random() * (AirfoilModifier.maxY4-
AirfoilModifier.minY4);
				double
y5=AirfoilModifier.minY5+Math.random() * (AirfoilModifier.maxY5-
AirfoilModifier.minY5);
				double
y6=AirfoilModifier.minY6+Math.random() * (AirfoilModifier.maxY6-
AirfoilModifier.minY6);
				double
y7=AirfoilModifier.minY7+Math.random() * (AirfoilModifier.maxY7-
AirfoilModifier.minY7);
				double
y8=AirfoilModifier.minY8+Math.random() * (AirfoilModifier.maxY8-
AirfoilModifier.minY8);
				double
y9=AirfoilModifier.minY9+Math.random() * (AirfoilModifier.maxY9-
AirfoilModifier.minY9);

				Airfoil mutant = new
Airfoil(x3,y3,x4,y4,x5,y5,x6,y6,x7,y7,x8,y8,x9,y9);
				AirfoilModifier.modAirfoil(mutant);
```

```
                              airfoiladd = mutant;
                              airfoilremove=airfoil;


                        nextGen.removeAirfoil(airfoilremove);
                        System.out.println("Airfoil removed for
mutation");
                        nextGen.addAirfoil(airfoiladd);
                        System.out.println("Airfoil added for
mutation");
                  }
            }
      }

      public void EvaIndividual(double X3, double Y3, double X4,
double Y4, double X5, double Y5, double X6, double Y6, double X7,
double Y7, double X8, double Y8, double X9, double Y9){
            double timeStep=0.02;
            Airfoil af=new Airfoil( X3,  Y3,  X4,  Y4,  X5,  Y5,
X6,  Y6,  X7,  Y7,  X8,  Y8,  X9,  Y9);
            gambitAirfoils gt=new gambitAirfoils(af);
            gt.buildAirfoil(timeStep);


            gt.publishFile("diffuser.dat");
            gt.setCoefficient();
      }

      public Airfoil FromRecord(double X3, double Y3, double X4,
double Y4, double X5, double Y5, double X6, double Y6, double X7,
double Y7, double X8, double Y8, double X9, double Y9) {
            Airfoil foil= new Airfoil( X3,  Y3,  X4,  Y4,  X5,  Y5,
X6,  Y6,  X7,  Y7,  X8,  Y8,  X9,  Y9);
            return foil;
      }

      public static void main(String[] args){
            wing firstTry = new wing(20,350,0.5,0.04);
//(Generation size, number of Gens, selection rate, mutation rate)
            //firstTry.EvaIndividual(-
3.707035639192833,5.624203134918806,-
3.706662087140752,4.985808753989674,-
3.4723243280406546,4.931716006530891,0.7693404695050218,4.4994021759
60706,1.1356468799647224,4.697972648066941,2.2836468994641668,7.8867
90558583122,5.001192838767047,7.999725825385547);
            firstTry.runOptimization();

      }
```

```
}
```

A.3.2 Airfoil.java

```
package wing;

public class Airfoil {
	public double
X3,Y3,X4,Y4,X5,Y5,X6,Y6,X7,Y7,X8,Y8,X9,Y9,X10,X11;

	public double fitness;
	public double X1=0, Y1=0, X2=0, Y2=5, Y10=0, Y11=0;

	/**Constructor for copying an existing airfoil**/
	public Airfoil(Airfoil af){

	this(af.X3,af.Y3,af.X4,af.Y4,af.X5,af.Y5,af.X6,af.Y6,af.X7,af.
Y7,af.X8,af.Y8,af.X9,af.Y9 );
		}

	/**Default constructor.  Initialize everything to 0.**/
	public Airfoil(){
		X3=0;
		X4=0;
		X5=0;
		X6=0;
		X7=0;
		X8=0;
		X9=0;

		Y3=0;
		Y4=0;
		Y5=0;
		Y6=0;
		Y7=0;
		Y8=0;
		Y9=0;
		fitness = 0;
	}

	/**Main constructor**/
	public Airfoil(double X3, double Y3, double X4, double Y4,
double X5, double Y5, double X6, double Y6, double X7, double Y7,
double X8, double Y8, double X9, double Y9 ){
		this.X3=X3;
		this.X4=X4;
		this.X5=X5;
```

```
            this.X6=X6;
            this.X7=X7;
            this.X8=X8;
            this.X9=X9;
            this.X10=X9;
            this.X11=X3;

            this.Y3=Y3;
            this.Y4=Y4;
            this.Y5=Y5;
            this.Y6=Y6;
            this.Y7=Y7;
            this.Y8=Y8;
            this.Y9=Y9;
            this.fitness = 1000000;
      }

      public Airfoil(double X3, double Y3, double X4, double Y4,
double X5, double Y5, double X6, double Y6, double X7, double Y7,
double X8, double Y8, double X9, double Y9, double fitness ){
            this.X3=X3;
            this.X4=X4;
            this.X5=X5;
            this.X6=X6;
            this.X7=X7;
            this.X8=X8;
            this.X9=X9;
            this.X10=X9;
            this.X11=X3;

            this.Y3=Y3;
            this.Y4=Y4;
            this.Y5=Y5;
            this.Y6=Y6;
            this.Y7=Y7;
            this.Y8=Y8;
            this.Y9=Y9;

    this.fitness = fitness;
      }



      public double getX3(){
            return X3;
      }
      public double getY3(){
            return Y3;
      }
      public double getX4(){
            return X4;
      }
      public double getY4(){
            return Y4;
```

```
	}
	public double getX5(){
		return X5;
	}
	public double getY5(){
		return Y5;
	}
	public double getX6(){
		return X6;
	}
	public double getY6(){
		return Y6;
	}
	public double getX7(){
		return X7;
	}
	public double getY7(){
		return Y7;
	}
	public double getX8(){
		return X8;
	}
	public double getY8(){
		return Y8;
	}
	public double getX9(){
		return X9;
	}
	public double getY9(){
		return Y9;
	}
	public double getFitness(){
		return fitness;
	}

}
```

A.3.3 generation.java

```
package wing;


import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.util.Vector;
```

```
public class generation {
	private Vector<Airfoil> airfoils;
	private int genSize;
	private double timeStep = 0.02;

	public generation(int genSize){
		this.genSize = genSize;
		this.airfoils = new Vector<Airfoil>();
	}

	public void addAirfoil(Airfoil airfoil){
		airfoils.add(airfoil);
	}

	public void replaceAirfoil(Airfoil airfoil, int i){
		airfoils.removeElementAt(i);
		airfoils.insertElementAt(airfoil,i);
	}

	public Airfoil getAirfoil(int i){
		return airfoils.elementAt(i);
	}

	public int getGenSize(){
		return genSize;
	}

	public int getAirfoilVectorSize(){
		return airfoils.size();
```

```
	}

	public void removeAirfoils(int num){
		for(int i=0;i<num;i++){
			airfoils.removeElementAt(0);
		}
	}

	public boolean removeAirfoil(Airfoil airfoil){
		return airfoils.remove(airfoil);
	}

	/**Get the airfoils and their X1,X2,Y1,Y2,M1,M2,M3,N1,N2,N3 values*/
	public void outputAirfoils(){
		for(int i=0;i<airfoils.size();i++){
			Airfoil af = airfoils.elementAt(i);
			System.out.println("Airfoil variables: " + af.X3 + ", " + af.Y3 +
", " + af.X4 + ", " + af.Y4 + ", " + af.X5 + ", " + af.Y5 + ", " + af.X6 + ", " + af.Y6 +
", " + af.X7 + ", " + af.Y7 + ", " + af.X8 + ", " + af.Y8 + ", " + af.X9 + ", " + af.Y9 +
", " + "fitness: " + af.fitness);

		}
	}

	/**create the coordinates for this generation of airfoils and store in vector
"airfoils"*/
	public void determineFitness(int generation){
		for(int i=0;i<airfoils.size();i++){
			Airfoil airfoil = airfoils.elementAt(i);
			gambitAirfoils gt = new gambitAirfoils(airfoil);
```

```
                        gt.buildAirfoil(timeStep);
                        airfoils.elementAt(i).fitness =
gt.getScoreWithFluent(generation,i,airfoils.elementAt(i).fitness);
                        try{
                                BufferedWriter recordWriter =  new
BufferedWriter(new FileWriter(new File("record.txt"), true));
                                recordWriter.write("Generation: " + generation +  "; ("
+ airfoils.elementAt(i).getX3()+ "," + airfoils.elementAt(i).getY3() + "," +
airfoils.elementAt(i).getX4() + "," + airfoils.elementAt(i).getY4()+ "," +
airfoils.elementAt(i).getX5() + "," + airfoils.elementAt(i).getY5() + "," +
airfoils.elementAt(i).getX6() + "," + airfoils.elementAt(i).getY6() + "," +
airfoils.elementAt(i).getX7() + "," + airfoils.elementAt(i).getY7()        + "," +
airfoils.elementAt(i).getX8() +"," + airfoils.elementAt(i).getY8() +"," +
airfoils.elementAt(i).getX9() +"," + airfoils.elementAt(i).getY9()+ ") fitness: " +
airfoils.elementAt(i).fitness + "\r");
                                recordWriter.close();
                        }
                        catch(Exception e){
                                e.printStackTrace();
                        }
                }
        }

        /**Get airfoil of the generation that has the highest coefficient of lift*/
        public Airfoil getBestAirfoil(){
                //Airfoil bestAirfoil = new Airfoil();
                for(int i=0;i<airfoils.size();i++){
                        System.out.println("Currently running airfoil Cp = " +
airfoils.elementAt(i).getFitness());
```

```
                        if(airfoils.elementAt(i).getFitness() >
wing.bestAirfoil.getFitness()){
                                wing.bestAirfoil = airfoils.elementAt(i);
                        }
                }
                System.out.println("New best airfoil Cp = " +
wing.bestAirfoil.getFitness());
                return wing.bestAirfoil;
        }
}
```

A.3.4 gambitAirfoils.java

```
package wing;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.text.DecimalFormat;



public class gambitAirfoils {
        public static final double X1=0, Y1=0, X2=0, Y2=5, Y10=0, Y11=0;
        public static final int extraPoints = 5;
        public double
X3,Y3,X4,Y4,X5,Y5,X6,Y6,X7,Y7,X8,Y8,X9,Y9,X10,X11,Ax,Bx,Cx,Dx,Ex,Fx,Ay,By,C
y,Dy,Ey,Fy,fitness, thickness, penalty, F;
```

```
        public double[] xPoints, yPoints;
        public int iterations,
A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14,A15,A16;
        public double max=0;
        public String coefficient;
        public Double fanCoeff;

        public static final double freeVel = 5.71;
        public static final double rho = 1.225;

        public static final double minX3=-8;
        public static final double maxX3=-.5;
        public static final double minX4=-8;
        public static final double maxX4=-.5;
        public static final double minX5=-4;
        public static final double maxX5=-.1;
        public static final double minX6=-1;
        public static final double maxX6=1;
        public static final double minX7=.1;
        public static final double maxX7=9;
        public static final double minX8=2;
        public static final double maxX8=10;
        public static final double minX9=3;
        public static final double maxX9=10;

        public static final double minY3=5;
        public static final double maxY3=6;
        public static final double minY4=4;
        public static final double maxY4=6;
        public static final double minY5=4;
```

```
public static final double maxY5=6;
public static final double minY6=4.3;
public static final double maxY6=5;
public static final double minY7=4;
public static final double maxY7=7;
public static final double minY8=5;
public static final double maxY8=7.071;
public static final double minY9=7;
public static final double maxY9=7.071;

public gambitAirfoils(Airfoil af){
        this.X3 = af.X3;
        this.Y3 = af.Y3;
        this.X4 = af.X4;
        this.Y4 = af.Y4;
        this.X5 = af.X5;
        this.Y5 = af.Y5;
        this.X6 = af.X6;
        this.Y6 = af.Y6;
        this.X7 = af.X7;
        this.Y7 = af.Y7;
        this.X8 = af.X8;
        this.Y8 = af.Y8;
        this.X9 = af.X9;
        this.Y9 = af.Y9;

        this.X10 = af.X9;
        this.X11 = af.X3;
```

```
this.Fx = getFx();
this.Ex = getEx();
this.Dx = getDx();
this.Cx = getCx();
this.Bx = getBx();
this.Ax = getAx();
this.Fy = getFy();
this.Ey = getEy();
this.Dy = getDy();
this.Cy = getCy();
this.By = getBy();
this.Ay = getAy();


}

public void buildAirfoil(double timeStep){
        iterations = (int) (1/timeStep);
        xPoints = new double[iterations + extraPoints];
        yPoints = new double[iterations + extraPoints];
        //mPoints = new double[iterations + extraPoints];
        //nPoints = new double[iterations + extraPoints];
        double t = 0;

        //bezier curve first
        for(int i=0;i<iterations;i++){
            xPoints[i] = X3*Math.pow(1-t, 6)
                        +X4*6*t*Math.pow(1-t, 5)
                        +X5*15*Math.pow(t,
2)*Math.pow(1-t, 4)
```

```
                                        +X6*20*Math.pow(t,
3)*Math.pow(1-t, 3)
                                        +X7*15*Math.pow(t,
4)*Math.pow(1-t, 2)
                                        +X8*6*(Math.pow(t, 5))*(1-t)
                                        +X9*Math.pow(t, 6);

                    //System.out.println(xPoints[i]);

                    yPoints[i] = Y3*Math.pow(1-t, 6)
                                        +Y4*6*t*Math.pow(1-t, 5)
                                        +Y5*15*Math.pow(t,
2)*Math.pow(1-t, 4)
                                        +Y6*20*Math.pow(t,
3)*Math.pow(1-t, 3)
                                        +Y7*15*Math.pow(t,
4)*Math.pow(1-t, 2)
                                        +Y8*6*(Math.pow(t, 5))*(1-t)
                                        +Y9*Math.pow(t, 6);
                    //System.out.println(yPoints[i]);



                    /** xPoints[i] = (Ax*Math.pow(t,6)) +
(Bx*Math.pow(t,5)) + (Cx*Math.pow(t,4)) + (Dx*Math.pow(t,3)) + (Ex*Math.pow(t,2))
+ Fx*t + X3;
                    yPoints[i] = (Ay*Math.pow(t,6)) + (By*Math.pow(t,5)) +
(Cy*Math.pow(t,4)) + (Dy*Math.pow(t,3)) + (Ey*Math.pow(t,2)) + Fy*t + Y3;
                    System.out.println(yPoints[i]);
                    **/
```

```
                        //mPoints[i] = (Dm*Math.pow(t,4)) +
(Em*Math.pow(t,3)) + (Fm*Math.pow(t, 2)) + Gm*t+X0;
                        //nPoints[i] = (Dn*Math.pow(t,4)) +
(En*Math.pow(t,3)) + (Fn*Math.pow(t, 2)) + Gn*t+Y0;
                        t = t + timeStep;
                    }

                    //Add tail points
                    for(int j=iterations;j<iterations+extraPoints;j++){
                        if(j==iterations){
                            xPoints[iterations] = X9;
                            yPoints[iterations] = Y9;
                        }if(j==iterations+1){
                                xPoints[iterations+1] = X1;
                                yPoints[iterations+1] = Y1;
                        }if(j==iterations+2){
                                    xPoints[iterations+2] = X2;
                                    yPoints[iterations+2] = Y2;
                        }if(j==iterations+3){
                                        xPoints[iterations+3] =
X10;
                                        yPoints[iterations+3] =
Y10;
                        }if(j==iterations+4){

        xPoints[iterations+4] = X11;

        yPoints[iterations+4] = Y11;
                                        }
```

```
                }
        }


        public double getFitness(){
                return fitness;
        }
        public synchronized double getScoreWithFluent(int generation, int iteration,
double oldfit){
                //System.out.println("Fitness: " + oldfit);
                if (oldfit != 1000000){
                        return oldfit;
                }

                else
                publishFile("diffuser.dat");
                //System.out.println("Ymiddle = " + yPoints[iterations/2+1]);


                setCoefficient();

                if (fanCoeff<0.1) {
                        return -1;
                }

                try{
                        Process gambitProc =
Runtime.getRuntime().exec("gambitTest.bat");
                        gambitProc.waitFor();
```

```
                        System.out.println("gambitTest Ran");
                        Process cleanupProc =
Runtime.getRuntime().exec("cleanup.bat");
                        cleanupProc.waitFor();
                        System.out.println("cleanup Ran");
                        File mesh = new File("diffuser.msh");
                        if(!mesh.exists()){
                                System.out.println("no mesh");
                                return -1;
                        }
                        Process fluentProc =
Runtime.getRuntime().exec("fluentTest.bat");
                        File transcript = new File("trans.jou");
                        System.out.println("Fluent Ran");
                        //wait and check for Fluent's return at one second
intervals
                        for(int s=0; s<180; s++){
                                wait(1000);
                                if(transcript.exists()){
                                        break;
                                }
                        }

                        //if Fluent did not return...
                        if(!transcript.exists()){
                                publishFile("trans.jou");
                                System.out.println("FLUENT DID NOT
RETURN A RESULT FOR THIS CASE: #" + iteration);
                                Process fluentKill =
Runtime.getRuntime().exec("fluentKill.bat");
```

```
                fluentKill.waitFor();
                return -1;
            }

            BufferedReader mo1Input =  new BufferedReader(new
FileReader(new File("surf-mon-1.out")));
            //BufferedReader mo2Input =  new
BufferedReader(new FileReader(new File("monitor-2.out")));
            //System.out.println("mo1Input= "+mo1Input);
            double mo1 = 100000;
            //double mo2=  100000;
            String linemo1;
            //String linemo2;

            //skip the first two lines

            for(int i=0;i<1001;i++){
                mo1Input.readLine();
                //mo2Input.readLine();
            }
            //System.out.println("mo1Input="+mo1Input);
              linemo1 = mo1Input.readLine();
              //linemo2 = mo2Input.readLine();
              //System.out.println("linemo1"+linemo1);
                linemo1 = linemo1.substring(5);
                //System.out.println("linemo1= "+linemo1);
                //linemo2 =
linemo2.substring(linemo2.indexOf("\t")+5);
                mo1 = Double.parseDouble(linemo1);
                //System.out.println("mo1"+mo1);
```

```
                    //mo2 = Double.parseDouble(linemo2);
                    double
Cp=fanCoeff*mo1*mo1*mo1/(.5*rho*freeVel*freeVel*freeVel);


                    System.out.println("FanVel = " + mo1);
                    //System.out.println("p2 = " + mo2);
                    System.out.println("Cp = " + Cp);
                    //return Cp;

                    A1=0; A2=0; A3=0; A4=0; A5=0; A6=0; A7=0;
A8=0; A9=0; A9=0; A10=0; A11=0; A11=0; A12=0; A13=0; A14=0; A15=0; A16=0;

                    if(X3 < minX3)
                              A1=-1;
                    else if(X3 > maxX3)
                              A1=1;

                    if(X4 < minX4)
                              A2=-1;
                    else if(X4 > maxX4)
                              A2=1;

                    if(X5 < minX5)
                              A3=-1;
                    else if(X5 > maxX5)
                              A3=1;


                    if(X6 < minX6)
```

```
        A4=-1;
else if(X6 > maxX6)
        A4=1;

if(X7 < minX7)
        A5=-1;
else if(X7 > maxX7)
        A5=1;

if(X8 < minX8)
        A6=-1;
else if(X8 > maxX8)
        A6=1;

if(X9 < minX9)
        A7=-1;
else if(X9 > maxX9)
        A7=1;


if(Y3 < minY3)
        A8=-1;
else if(Y3 > maxY3)
        A8=1;

if(Y4 < minY4)
        A9=-1;
else if(Y4 > maxY4)
        A9=1;
```

```
if(Y5 < minY5)
        A10=-1;
else if(Y5 > maxY5)
        A10=1;

if(Y6 < minY6)
        A11=-1;
else if(Y6 > maxY6)
        A11=1;

if(Y7 < minY7)
        A12=-1;
else if(Y7 > maxY7)
        A12=1;

if(Y8 < minY8)
        A13=-1;
else if(Y8 > maxY8)
        A13=1;

if(Y9 < minY9)
        A14=-1;
else if(Y9 > maxY9)
        A14=1;

        System.out.println(" X3:" + A1 + " X4:" + A2 +
" X5:" + A3+ " X6:" + A4+ " X7:" + A5+ " X8:" + A6+ " X9:" +A7 +
                                        " Y3:" + A8 + " Y4:" + A9 + "
Y5:" + A10+ " Y6:" + A11+ " Y7:" + A12+ " Y8:" + A13 + " Y9:" + A14);
    //+ "A12 = " + A12 +"A13 = "+ A13+ "A14 = " +A14 );
```

penalty=20*(

A1*A1*(A1*((X3-minX3)+(X3-maxX3))-(maxX3-minX3))*(A1*((X3-minX3)+(X3-maxX3))-(maxX3-minX3))

+A2*A2*(A2*((X4-minX4)+(X4-maxX4))-(maxX4-minX4))*(A2*((X4-minX4)+(X4-maxX4))-(maxX4-minX4))

+A3*A3*(A3*((X5-minX5)+(X5-maxX5))-(maxX5-minX5))*(A3*((X5-minX5)+(X5-maxX5))-(maxX5-minX5))

+A4*A4*(A4*((X6-minX6)+(X6-maxX6))-(maxX6-minX6))*(A4*((X6-minX6)+(X6-maxX6))-(maxX6-minX6))

+A5*A5*(A5*((X7-minX7)+(X7-maxX7))-(maxX7-minX7))*(A5*((X7-minX7)+(X7-maxX7))-(maxX7-minX7))

+A6*A6*(A6*((X8-minX8)+(X8-maxX8))-(maxX8-minX8))*(A6*((X8-minX8)+(X8-maxX8))-(maxX8-minX8))

+A7*A7*(A7*((X9-minX9)+(X9-maxX9))-(maxX9-minX9))*(A7*((X9-minX9)+(X9-maxX9))-(maxX9-minX9))

+A8*A8*(A8*((Y3-minY3)+(Y3-maxY3))-(maxY3-minY3))*(A8*((Y3-minY3)+(Y3-maxY3))-(maxY3-minY3))

+A9*A9*(A9*((Y4-minY4)+(Y4-maxY4))-(maxY4-minY4))*(A9*((Y4-minY4)+(Y4-maxY4))-(maxY4-minY4))

+A10*A10*(A10*((Y5-minY5)+(Y5-maxY5))-(maxY5-minY5))*(A10*((Y5-minY5)+(Y5-maxY5))-(maxY5-minY5))

+A11*A11*(A11*((Y6-minY6)+(Y6-maxY6))-(maxY6-minY6))*(A11*((Y6-minY6)+(Y6-maxY6))-(maxY6-minY6))

+A12*A12*(A12*((Y7-minY7)+(Y7-maxY7))-(maxY7-minY7))*(A12*((Y7-minY7)+(Y7-maxY7))-(maxY7-minY7))

```
                                                    +A13*A13*(A13*((Y8-
minY8)+(Y8-maxY8))-(maxY8-minY8))*(A13*((Y8-minY8)+(Y8-maxY8))-(maxY8-
minY8))
                                                    +A14*A14*(A14*((Y9-
minY9)+(Y9-maxY9))-(maxY9-minY9))*(A14*((Y9-minY9)+(Y9-maxY9))-(maxY9-
minY9))
                                                );

                                        System.out.println("penalty = "+ penalty);

                                F=Cp-penalty;
                                if (Cp>2.5){
                                        System.out.println("FLUENT RETURNED
SPURIOUS RESULT FOR THIS CASE: #" + iteration);
                                        return -1;
                                }
                                System.out.println("fitness = "+ F);
                                return F;

                }

                        catch(Exception e){
                                e.printStackTrace();
                                return -1;
                        }
                }

        public static String truncate(String value, int length)
        {
          if (value != null && value.length() > length)
```

```
    value = value.substring(0, length);
  return value;
}

        public void setCoefficient() {

                double coefficientLong = -0.5*rho*(
(yPoints[24]*yPoints[24]*yPoints[24]*yPoints[24]/(yPoints[50]*yPoints[50]*yPoints[50]*
yPoints[50])) -
(yPoints[24]*yPoints[24]*yPoints[24]*yPoints[24]/(yPoints[0]*yPoints[0]*yPoints[0]*yP
oints[0])) );

                fanCoeff = coefficientLong;
                coefficient = truncate(Double.toString(coefficientLong), 8);
                System.out.println("Fan Coefficient: " + coefficient);


                try {
                        BufferedReader r = new BufferedReader(new
FileReader("diffuserTemplate.jou"));
                        BufferedWriter w = new BufferedWriter(new
FileWriter("diffuserFluent.jou"));
                        String line;
                        while((line = r.readLine()) != null) {
                                line = line.replaceFirst("xxxx", coefficient);
                                w.write(line);
                                w.newLine();
                        }
                        r.close();
                        w.close();
```

```
            } catch(Exception e) {
                    e.printStackTrace();
            }
}

        public boolean publishFile(String filename){
                try{
                        // Create file
                        FileWriter fstream = new FileWriter(filename);
                        BufferedWriter out = new BufferedWriter(fstream);
                        out.write(2*xPoints.length+ " 2\n");
                        for(int i=0;i<xPoints.length;i++){
                              out.write(xPoints[i]+ " " + yPoints[i] + " 0\n");
                        }



                        out.close();
                        return true;
                }
                catch (Exception e){
                        System.err.println("Error: " + e.getMessage());
                        return false;
                }
        }

        //coefficients of 7 ctrl pts
        public double getFx(){
                return -6*X3+X4;
        }
```

```
public double getEx(){
        return 15*X3-5*X4+X5;
}

public double getDx(){
        return -20*X3+10*X4-4*X5+X6;
}

public double getCx(){
        return 15*X3-10*X4+6*X5-3*X6+X7;
}

public double getBx(){
        return (-6*X3+5*X4-4*X5+3*X6-2*X7+X8);
}

public double getAx(){
        return X3-X4+X5-X6+X7-X8+X9;
}

public double getFy(){
        return -6*Y3+Y4;
}

public double getEy(){
        return 15*Y3-5*Y4+Y5;
}

public double getDy(){
```

```
        return -20*Y3+10*Y4-4*Y5+Y6;
}

public double getCy(){
        return 15*Y3-10*Y4+6*Y5-3*Y6+Y7;
}

public double getBy(){
        return (-6*Y3+5*Y4-4*Y5+3*Y6-2*Y7+Y8);
}

public double getAy(){
        return Y3-Y4+Y5-Y6+Y7-Y8+Y9;
}

//followings are coefficient for 2 ctrl pts
/**public double getCx(){
        return 3*(X4-X3);
}

public double getBx(){
        return 3*(X5-X4)-3*(X4-X3);
}

public double getAx(){
        return X6-X3-3*(X4-X3)-3*(X5-X4)+3*(X4-X3);
}

public double getCy(){
        return 3*(Y4-Y3);
```

```
	}

	public double getBy(){
		return 3*(Y5-Y4)-3*(Y4-Y3);
	}

	public double getAy(){
		return Y6-Y3-3*(Y4-Y3)-3*(Y5-Y4)+3*(Y4-Y3);
	}
	**/

}
```

A.3.5 AirfoilModifier.java

```
package wing;

public class AirfoilModifier {
	public static final double minX3=-8;
	public static final double maxX3=-.5;
	public static final double minX4=-8;
	public static final double maxX4=-.5;
	public static final double minX5=-4;
	public static final double maxX5=-.1;
	public static final double minX6=-1;
	public static final double maxX6=1;
	public static final double minX7=.1;
	public static final double maxX7=9;
	public static final double minX8=2;
	public static final double maxX8=10;
	public static final double minX9=3;
	public static final double maxX9=10;

	public static final double minY3=5;
	public static final double maxY3=6;
	public static final double minY4=4;
	public static final double maxY4=6;
	public static final double minY5=4;
	public static final double maxY5=6;
	public static final double minY6=4.3;
	public static final double maxY6=5;
```

```
public static final double minY7=4;
public static final double maxY7=7;
public static final double minY8=5;
public static final double maxY8=7.071;
public static final double minY9=7;
public static final double maxY9=7.071;


public static void modAirfoil(Airfoil af
            ){
      /** make sure all the variables in the right range**/
      /**if(af.X4 < minX4)
            af.X4 = minX4;
      else if(af.X4 > maxX4)
            af.X4 = maxX4;
      else
            af.X4 = af.X4;

      if(af.X5 < minX5)
            af.X5 = minX5;
      else if(af.X5 > maxX5)
            af.X5 = maxX5;
      else
            af.X5 = af.X5;

      if(af.X6 < minX6)
            af.X6 = minX6;
      else if(af.X6 > maxX6)
            af.X6 = maxX6;
      else
            af.X6 = af.X6;

      if(af.Y4 < minY4)
            af.Y4 = minY4;
      else if(af.Y4 > maxY4)
            af.Y4 = maxY4;
      else
            af.Y4 = af.Y4;

      if(af.Y5 < minY5)
            af.Y5 = minY5;
      else if(af.Y5 > maxY5)
            af.Y5 = maxY5;
      else
            af.Y5 = af.Y5;

      if(af.Y6 < minY6)
            af.Y6 = minY6;
      else if(af.Y6 > maxY6)
            af.Y6 = maxY6;
      else
            af.Y6 = af.Y6;**/
```

```
            /**make x1,x2,m1,m2 in order**/
            if(af.X5>af.X6)
                  af.X5=af.X6-Math.random() * (af.X6-minX5);

            if(af.X5<af.X4)
                  af.X4=af.X5-Math.random() * (af.X5-minX4);
            if(af.X4<af.X3)
                  af.X3=af.X4-Math.random() * (af.X4-minX3);

            if(af.X7<af.X6)
                  af.X7=af.X6+Math.random() * (maxX7-af.X6);
            if(af.X8<af.X7)
                  af.X8=af.X7+Math.random() * (maxX8-af.X7);
            if(af.X9<af.X8)
                  af.X9=af.X8+Math.random() * (maxX9-af.X8);

      }
/**
      public static void modgenerateIndividual(Airfoil af
      ){
            gambitAirfoils gt = new gambitAirfoils(af);
            gt.buildAirfoil(0.02);

      System.out.println("maxthickness1="+gt.getmaxThickness());
            if
(gt.getmaxThickness()<=0.35&&gt.getmaxThickness()>=0.30){
                  af.X1 = af.X1;
                  af.Y1 = af.Y1;
                  af.X2 = af.X2;
                  af.Y2 = af.Y2;
                  af.M1 = af.M1;
                  af.M2 = af.M2;
                  af.N1 = af.N1;
                  af.N2 = af.N2;
                  af.N3 = af.N3;}
            else{
                        af=gaflatback.generateIndividual();}
      }**/

}
```

A.3.6 BubbleSorter.java

```
package wing;

import java.util.Vector;

/*
 * @(#)BubbleSortAlgorithm.java     1.6 95/01/31 James Gosling
 *
```

```
 * Copyright (c) 1994 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Permission to use, copy, modify, and distribute this software
 * and its documentation for NON-COMMERCIAL purposes and without
 * fee is hereby granted provided that this copyright notice
 * appears in all copies. Please refer to the file "copyright.html"
 * for further important copyright and licensing information.
 *
 * SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
OF
 * THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
LIMITED
 * TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE
FOR
 * ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
OR
 * DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
 */

/**
 * A bubble sort demonstration algorithm
 * SortAlgorithm.java, Thu Oct 27 10:32:35 1994
 *
 * @author James Gosling
 * @version       1.6, 31 Jan 1995
 *
 * Modified 23 Jun 1995 by Jason Harrison@cs.ubc.ca:
 *   Algorithm completes early when no items have been swapped in
the
 *   last pass.
 *
 *
 */
public class BubbleSorter {
       public static void sort(generation a){
                  for (int i = a.getGenSize(); --i>=0; ) {
                  boolean flipped = false;
                  for (int j = 0; j<i; j++) {

                        if (a.getAirfoil(j).getFitness() >
a.getAirfoil(j+1).getFitness()) {
                              Airfoil T = a.getAirfoil(j);
                              a.replaceAirfoil(a.getAirfoil(j+1),
j);
                              a.replaceAirfoil(T,j+1);
                              flipped = true;
                        }
                  }
                  if (!flipped) {
                        return;
                  }
            }
         }
```

```
}
```

A.3.7 gambitTest.bat

```
DEL "diffuser.msh"
C:\Fluent.Inc\ntbin\ntx86\gambit -inputfile "diffuserGambit.jou"
```

A.3.8 diffuserGambit.jou

```
/ Journal File for GAMBIT 2.4.6, Database 2.4.4, ntx86 SP2007051421
/ Identifier "default_id18208"
/ File opened for write Wed Nov 03 17:00:12 2010.
/ERROR occurred in the next command!
import vertexdata "C:\\FOOTE\\FirstGA\\wing\\diffuser.dat"
edge create straight "vertex.52" "vertex.53"
edge create straight "vertex.55" "vertex.52"
edge create straight "vertex.52" "vertex.54"
edge create straight "vertex.55" "vertex.1"
edge create straight "vertex.54" "vertex.51"
edge create nurbs "vertex.1" "vertex.2" "vertex.3" "vertex.4" "vertex.5" \
  "vertex.6" "vertex.7" "vertex.8" "vertex.9" "vertex.10" "vertex.11" \
  "vertex.12" "vertex.13" "vertex.14" "vertex.15" "vertex.16" "vertex.17" \
  "vertex.18" "vertex.19" "vertex.20" "vertex.21" "vertex.22" "vertex.23" \
  "vertex.24" "vertex.25" "vertex.26" "vertex.27" "vertex.28" "vertex.29" \
  "vertex.30" "vertex.31" "vertex.32" "vertex.33" "vertex.34" "vertex.35" \
  "vertex.36" "vertex.37" "vertex.38" "vertex.39" "vertex.40" "vertex.41" \
  "vertex.42" "vertex.43" "vertex.44" "vertex.45" "vertex.46" "vertex.47" \
  "vertex.48" "vertex.49" "vertex.50" "vertex.51" interpolate
vertex create coordinates 0 9 0
edge create straight "vertex.53" "vertex.56"
edge split "edge.6" tolerance 1e-06 edge "edge.7" connected
edge create straight "vertex.53" "vertex.57"
face create wireframe "edge.4" "edge.6" "edge.9" "edge.1" "edge.2" real
face create wireframe "edge.8" "edge.9" "edge.1" "edge.3" "edge.5" real
undo begingroup
edge picklink "edge.1" "edge.2" "edge.3" "edge.8" "edge.6"
edge mesh "edge.6" "edge.8" "edge.3" "edge.2" "edge.1" successive ratio1 1 \
  intervals 70
undo endgroup
undo begingroup
edge picklink "edge.5" "edge.4"
```

```
edge mesh "edge.4" "edge.5" successive ratio1 1 intervals 90
undo endgroup
undo begingroup
edge picklink "edge.9"
edge mesh "edge.9" successive ratio1 1 intervals 20
undo endgroup
face mesh "face.1" "face.2" map size 1
physics create btype "PRESSURE_INLET" edge "edge.4"
physics create btype "FAN" edge "edge.1"
physics create btype "PRESSURE_OUTLET" edge "edge.5"
physics create btype "AXIS" edge "edge.2" "edge.3"
physics create btype "WALL" edge "edge.6" "edge.8"
physics create btype "INTERIOR" edge "edge.9"
export fluent5 "C:\\FOOTE\\FirstGA\\wing\\diffuser.msh" nozval
```

A.3.9 diffuser.dat

This file is created for each individual and holds the vertex data of the shroud for the journal running in GAMBIT. This is an example of the file from Case 2a.

```
110 2
-3.707035639192833 5.624203134918806 0
-3.705010802206705 5.550952334764735 0
-3.6969338339862823 5.483854984874721 0
-3.6801211082192835 5.422155306186885 0
-3.652452946768152 5.3652073081701 0
-3.6123225502438956 5.312472302602039 0
-3.5585867996455938 5.263515995513405 0
-3.4905189290655025 5.218005157298251 0
-3.4077630694597927 5.17570387099048 0
-3.31029066348492 5.136469358706467 0
-3.1983587513996077 5.100247386253809 0
-3.0724701280324758 5.067067245906265 0
-2.9333353708152723 5.037036317344785 0
-2.7818367388817515 5.010334206764702 0
-2.6189939432321667 4.9872064641490725 0
-2.445931787963385 4.967957878708145 0
-2.263849682564642 4.952945352484972 0
-2.0739930252789134 4.942570352127173 0
-1.8776264575299102 4.937270938824826 0
```

-1.6760089894147072 4.937513376414511 0
-1.4703709962619913 4.943783317649484 0
-1.261893086255942 4.956576568636005 0
-1.0516868391257304 4.976389431435791 0
-0.8407774159006483 5.003708624834629 0
-0.630088039730865 5.039000783277116 0
-0.42042634777380883 5.082701533967543 0
-0.21247261414616736 5.135204152136927 0
-0.006769843941527395 5.196847794476178 0
0.19628426168636945 5.267905310735407 0
0.396443469374743 5.348570633489376 0
0.593617306337263 5.438945746069096 0
0.7878734820839135 5.539027228659554 0
0.9794384390752296 5.648692382563592 0
1.1686960323109097 5.767684932631919 0
1.3561843378528007 5.895600307859273 0
1.5425905902822585 6.031870500146715 0
1.7287442490918812 6.175748501230071 0
1.9156081940116167 6.326292317774511 0
2.104268049269246 6.482348564635279 0
2.2959196367852366 6.642535636284541 0
2.491854558301975 6.805226456404403 0
2.693443906447368 6.968530805646054 0
2.902120104732825 7.130277227555049 0
3.119356876485604 7.287994512662741 0
3.346647342715543 7.438892760743849 0
3.5854802489161584 7.579844021240172 0
3.8373143208001177 7.7073625118504365 0
4.103550748969089 7.817584415286291 0
4.385503802517964 7.906247254194441 0
4.684369571573453 7.968668844244929 0
5.001192838767047 7.999725825385547 0
0.0 0.0 0
0.0 5.0 0
5.001192838767047 0.0 0
-3.707035639192833 0.0 0

A.3.10 cleanup.bat

DEL *default*

DEL "trans.jou"

A.3.11 fluentTest.bat

```
Start C:\Foote\FirstGA\wing\fluent 2ddp -g -i
"C:\Foote\FirstGA\wing\diffuserFluent"
```

A.3.12 fluentTemplate.jou

This is the template file that is read by the GA to create the diffuserFluent.jou file that is the journal file called by fluentTest.bat and used by the flow solver FLUENT to evaluate each individual. To modify the dimentions of the system the scaling factor can be changed, and to modify the free stream velocity, the pressure inlet can be modified to match the stagnation pressure of the velocity desired. This example is from Case 2a.

```
(cx-gui-do cx-activate-item "MenuBar*ReadSubMenu*Mesh...")
(cx-gui-do cx-set-text-entry "Select File*FilterText" "c:\foote\firstga\wing\*")
(cx-gui-do cx-activate-item "Select File*Apply")
(cx-gui-do cx-set-text-entry "Select File*Text" "diffuser.msh")
(cx-gui-do cx-activate-item "Select File*OK")
(cx-gui-do cx-set-toggle-button
"General*Frame1*Table1*Frame2(Solver)*Table2(Solver)*Frame6(2D
Space)*ToggleBox6(2D Space)*Axisymmetric" #f)
(cx-gui-do cx-activate-item
"General*Frame1*Table1*Frame2(Solver)*Table2(Solver)*Frame6(2D
Space)*ToggleBox6(2D Space)*Axisymmetric")
(cx-gui-do cx-activate-item
"General*Frame1*Table1*Frame1(Mesh)*ButtonBox1(Mesh)*PushButton1(Scale)")
(cx-gui-do cx-set-list-selections "Scale
Mesh*Frame2(Scaling)*Table2(Scaling)*DropDownList2(Mesh Was Created In)" '( 5))
(cx-gui-do cx-activate-item "Scale
Mesh*Frame2(Scaling)*Table2(Scaling)*DropDownList2(Mesh Was Created In)")
(cx-gui-do cx-activate-item "Scale
Mesh*Frame2(Scaling)*Table2(Scaling)*PushButton4(Scale)")
(cx-gui-do cx-set-toggle-button "Scale
Mesh*Frame2(Scaling)*Table2(Scaling)*Frame1*ToggleBox1*Specify Scaling Factors"
#f)
```

```
(cx-gui-do cx-activate-item "Scale
Mesh*Frame2(Scaling)*Table2(Scaling)*Frame1*ToggleBox1*Specify Scaling Factors")
(cx-gui-do cx-set-real-entry-list "Scale
Mesh*Frame2(Scaling)*Table2(Scaling)*Frame3(Scaling Factors)*RealEntry1(X)" '( 1.6))
(cx-gui-do cx-set-real-entry-list "Scale
Mesh*Frame2(Scaling)*Table2(Scaling)*Frame3(Scaling Factors)*RealEntry2(Y)" '( 1.6))
(cx-gui-do cx-activate-item "Scale
Mesh*Frame2(Scaling)*Table2(Scaling)*PushButton4(Scale)")
(cx-gui-do cx-activate-item "Scale Mesh*PanelButtons*PushButton1(Close)")
(cx-gui-do cx-activate-item "NavigationPane*Frame1*PushButton3(Models)")
(cx-gui-do cx-set-list-selections "Models*Frame1*Table1*Frame1*List1(Models)" '( 2))
(cx-gui-do cx-activate-item "Models*Frame1*Table1*Frame1*List1(Models)")
(cx-gui-do cx-activate-item "Models*Frame1*Table1*PushButton2(Edit)")
(cx-gui-do cx-set-toggle-button "Viscous
Model*Frame1*Table1*Frame1(Model)*ToggleBox1(Model)*k-epsilon (2 eqn)" #f)
(cx-gui-do cx-activate-item "Viscous
Model*Frame1*Table1*Frame1(Model)*ToggleBox1(Model)*k-epsilon (2 eqn)")
(cx-gui-do cx-set-position "Viscous Model" '(x 140 y 230))
(cx-gui-do cx-activate-item "Viscous Model*PanelButtons*PushButton1(OK)")
(cx-gui-do cx-activate-item "NavigationPane*Frame1*PushButton4(Materials)")
(cx-gui-do cx-set-list-selections "Materials*Frame1*Table1*Frame1*List1(Materials)" '(
1))
(cx-gui-do cx-activate-item "Materials*Frame1*Table1*Frame1*List1(Materials)")
(cx-gui-do cx-activate-item "NavigationPane*Frame1*PushButton7(Boundary
Conditions)")
(cx-gui-do cx-set-list-selections "Boundary
Conditions*Frame1*Table1*Frame1*List1(Zone)" '( 4))
(cx-gui-do cx-activate-item "Boundary
Conditions*Frame1*Table1*Frame1*List1(Zone)")
(cx-gui-do cx-activate-item "Boundary
Conditions*Frame1*Table1*Frame2*Table2*Frame4*Table4*Frame1*ButtonBox1*Pus
hButton1(Edit)")
(cx-gui-do cx-set-real-entry-list "pressure-inlet-8-
1*Frame4*Frame3(Momentum)*Frame1*Table1*Frame5*Table5*RealEntry2(Gauge
Total Pressure)" '( 20))
(cx-gui-do cx-activate-item "pressure-inlet-8-1*PanelButtons*PushButton1(OK)")
(cx-gui-do cx-set-list-selections "Boundary
Conditions*Frame1*Table1*Frame1*List1(Zone)" '( 2))
(cx-gui-do cx-activate-item "Boundary
Conditions*Frame1*Table1*Frame1*List1(Zone)")
(cx-gui-do cx-activate-item "Boundary
Conditions*Frame1*Table1*Frame2*Table2*Frame4*Table4*Frame1*ButtonBox1*Pus
hButton1(Edit)")
(cx-gui-do cx-activate-item "Fan*Frame3*Frame1(Pressure-Jump
Specification)*Frame1*Table1*Frame4*Frame2*PushButton2(Edit)")
```

```
(cx-gui-do cx-set-integer-entry "Polynomial
Profile*Frame1*IntegerEntry3(Coefficients)" 2)
(cx-gui-do cx-activate-item "Polynomial Profile*Frame1*IntegerEntry3(Coefficients)")
(cx-gui-do cx-set-integer-entry "Polynomial
Profile*Frame1*IntegerEntry3(Coefficients)" 3)
(cx-gui-do cx-activate-item "Polynomial Profile*Frame1*IntegerEntry3(Coefficients)")
(cx-gui-do cx-set-real-entry-list "Polynomial
Profile*Frame2(Coefficients)*Table2(Coefficients)*RealEntry3(3)" '(xxxx))
(cx-gui-do cx-activate-item "Polynomial Profile*PanelButtons*PushButton1(OK)")
(cx-gui-do cx-activate-item "Fan*Frame3*Frame1(Pressure-Jump
Specification)*Frame1*Table1*Frame4*Frame2*RealEntry3")
(cx-gui-do cx-set-toggle-button "Fan*Frame3*Frame1(Pressure-Jump
Specification)*Frame1*Table1*Frame1*Table1*CheckButton1(Reverse Fan Direction)"
#f)
(cx-gui-do cx-activate-item "Fan*Frame3*Frame1(Pressure-Jump
Specification)*Frame1*Table1*Frame1*Table1*CheckButton1(Reverse Fan
Direction)")
(cx-gui-do cx-activate-item "Fan*PanelButtons*PushButton1(OK)")
(cx-gui-do cx-activate-item "NavigationPane*Frame1*PushButton12(Solution
Methods)")
(cx-gui-do cx-activate-item "NavigationPane*Frame1*PushButton13(Solution
Controls)")
(cx-gui-do cx-activate-item "NavigationPane*Frame1*PushButton14(Monitors)")
(cx-gui-do cx-activate-item "Monitors*Frame1*Table1*PushButton2(Edit)")
(cx-gui-do cx-set-toggle-button "Residual
Monitors*Frame1*Table1*Frame2*Table2*Frame1(Equations)*Table1(Equations)*Che
ckButton10" #t)
(cx-gui-do cx-activate-item "Residual
Monitors*Frame1*Table1*Frame2*Table2*Frame1(Equations)*Table1(Equations)*Che
ckButton10")
(cx-gui-do cx-set-toggle-button "Residual
Monitors*Frame1*Table1*Frame2*Table2*Frame1(Equations)*Table1(Equations)*Che
ckButton16" #t)
(cx-gui-do cx-activate-item "Residual
Monitors*Frame1*Table1*Frame2*Table2*Frame1(Equations)*Table1(Equations)*Che
ckButton16")
(cx-gui-do cx-set-toggle-button "Residual
Monitors*Frame1*Table1*Frame2*Table2*Frame1(Equations)*Table1(Equations)*Che
ckButton22" #t)
(cx-gui-do cx-activate-item "Residual
Monitors*Frame1*Table1*Frame2*Table2*Frame1(Equations)*Table1(Equations)*Che
ckButton22")
(cx-gui-do cx-set-toggle-button "Residual
Monitors*Frame1*Table1*Frame2*Table2*Frame1(Equations)*Table1(Equations)*Che
ckButton28" #t)
```

```
(cx-gui-do cx-activate-item "Residual
Monitors*Frame1*Table1*Frame2*Table2*Frame1(Equations)*Table1(Equations)*Che
ckButton28")
(cx-gui-do cx-set-toggle-button "Residual
Monitors*Frame1*Table1*Frame2*Table2*Frame1(Equations)*Table1(Equations)*Che
ckButton34" #t)
(cx-gui-do cx-activate-item "Residual
Monitors*Frame1*Table1*Frame2*Table2*Frame1(Equations)*Table1(Equations)*Che
ckButton34")
(cx-gui-do cx-activate-item "Residual Monitors*PanelButtons*PushButton1(OK)")
(cx-gui-do cx-activate-item "NavigationPane*Frame1*PushButton15(Solution
Initialization)")
(cx-gui-do cx-set-real-entry-list "Solution Initialization*Frame1*Table1*Frame4(Initial
Values)*Table4(Initial Values)*RealEntry2(Axial Velocity)" '( 6.3))
(cx-gui-do cx-activate-item "Solution Initialization*Frame1*Table1*Frame4(Initial
Values)*Table4(Initial Values)*RealEntry2(Axial Velocity)")
(cx-gui-do cx-activate-item "Solution
Initialization*Frame1*Table1*Frame6*ButtonBox6*PushButton1(Initialize)")
(cx-gui-do cx-activate-item "NavigationPane*Frame1*PushButton17(Run Calculation)")
(cx-gui-do cx-set-integer-entry "Run
Calculation*Frame1*Table1*IntegerEntry8(Number of Iterations)" 1000)
(cx-gui-do cx-activate-item "Run Calculation*Frame1*Table1*IntegerEntry8(Number
of Iterations)")
(cx-gui-do cx-activate-item "NavigationPane*Frame1*PushButton14(Monitors)")
(cx-gui-do cx-activate-item
"Monitors*Frame1*Table1*Frame4*Table4*PushButton1(Create)")
(cx-gui-do cx-set-list-selections "Surface
Monitor*Frame1*Table1*Frame2*Table2*DropDownList1(Report Type)" '( 6))
(cx-gui-do cx-activate-item "Surface
Monitor*Frame1*Table1*Frame2*Table2*DropDownList1(Report Type)")
(cx-gui-do cx-set-list-selections "Surface
Monitor*Frame1*Table1*Frame2*Table2*DropDownList2(Field Variable)" '( 2))
(cx-gui-do cx-activate-item "Surface
Monitor*Frame1*Table1*Frame2*Table2*DropDownList2(Field Variable)")
(cx-gui-do cx-set-list-selections "Surface
Monitor*Frame1*Table1*Frame2*Table2*DropDownList3" '( 1))
(cx-gui-do cx-activate-item "Surface
Monitor*Frame1*Table1*Frame2*Table2*DropDownList3")
(cx-gui-do cx-set-list-selections "Surface
Monitor*Frame1*Table1*Frame2*Table2*Frame5*Table5*Frame1*List1(Surfaces)" '(
2))
(cx-gui-do cx-activate-item "Surface
Monitor*Frame1*Table1*Frame2*Table2*Frame5*Table5*Frame1*List1(Surfaces)")
```

```
(cx-gui-do cx-set-toggle-button "Surface
Monitor*Frame1*Table1*Frame1*Table1*Frame2(Options)*Table2(Options)*CheckBu
tton4(Write)" #f)
(cx-gui-do cx-activate-item "Surface
Monitor*Frame1*Table1*Frame1*Table1*Frame2(Options)*Table2(Options)*CheckBu
tton4(Write)")
(cx-gui-do cx-activate-item "Surface Monitor*PanelButtons*PushButton1(OK)")
(cx-gui-do cx-activate-item "NavigationPane*Frame1*PushButton17(Run Calculation)")
(cx-gui-do cx-activate-item "Run
Calculation*Frame1*Table1*PushButton18(Calculate)")
(cx-gui-do cx-activate-item "Question*Cancel")
(cx-gui-do cx-activate-item "Warning*OK")
(cx-gui-do cx-activate-item "Information*OK")
(cx-gui-do cx-activate-item "MenuBar*WriteSubMenu*Profile...")
(cx-gui-do cx-set-list-selections "Write Profile*Frame2*List2(Surfaces)" '( 2))
(cx-gui-do cx-activate-item "Write Profile*Frame2*List2(Surfaces)")
(cx-gui-do cx-set-list-selections "Write Profile*Frame3*List3(Values)" '( 9))
(cx-gui-do cx-activate-item "Write Profile*Frame3*List3(Values)")
(cx-gui-do cx-activate-item "Write Profile*PanelButtons*PushButton1(OK)")
(cx-gui-do cx-set-text-entry "Select File*Text" "fanvelocityprofile")
(cx-gui-do cx-activate-item "Select File*OK")
(cx-gui-do cx-activate-item "Write Profile*PanelButtons*PushButton2(Cancel)")
(cx-gui-do cx-activate-item "MenuBar*WriteSubMenu*Stop Transcript")
(cx-gui-do cx-set-text-entry "Select File*FilterText" "c:\foote\firstga\wing\*")
(cx-gui-do cx-activate-item "Select File*Apply")
(cx-gui-do cx-set-text-entry "Select File*Text" "trans.jou")
(cx-gui-do cx-activate-item "Select File*OK")
(cx-gui-do cx-activate-item "MenuBar*WriteSubMenu*Stop Transcript")
(cx-gui-do cx-activate-item "MenuBar*FileMenu*Exit")
(cx-gui-do cx-activate-item "Warning*OK")
```

References

[1] Michael Werle, FloDesign Inc.; Walter Presz, FloDesign Inc. Journal of Propulsion and Power 2008, 0748-4658 vol.24 no.5 (1146-1150). doi: 10.2514/1.37134

[2] Hansen, M.O.L., *Aerodynamics of Wind Turbines*, Earthscan, Sterling, VA, 2000.

[3] Widnall, Sheila. *Potential Flow Calculations of Axisymmetric Ducted Wind Turbines*. Massachusetts Institute of Technology, July 2009.

[4] Mikkelsen, R., "Actuator Disc Methods applied to Wind Turbines," PhD thesis, Technical University of Denmark, Copenhagen, June 2003.

[5] Manwell, J.F., McGowan, J.G. and Rogers, A.L., *Wind Energy Explained*, John Wiley, 2nd edition, 2009.

[6] FLUENT 12.1: Flow Modeling Software, Ansys Inc., 2009.

[7] GAMBIT 6.2: Geometry and Mesh Generation Preprocessor, Ansys Inc., 2007.

[8] Pratihar, D. K., *Soft Computing*. Narosa Publishing House: New Delhi, India, 2008.

[9] Holland, J. H. *Adaptation in Natural and Artificial Systems*, The University of Michigan Press: Ann Arbor, MI, U.S.A., 1975.

[10] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley: Reading, MA, 1989.

[11] "Programming Language Popularity," 2009.

[12] "TIOBE Programming Community Index," 2009.

[13] Hansen, M. O. L., Sorensen, N. N., Flay, R. G. J., *Effect of Placing a Diffuser around a Wind Turbine*. Wind Energy, 2000; 3:207-213 (DOI: 10.1002/we.37).

Vita

Tudor F. Foote

Date of Birth August 15, 1986

Place of Birth Concord, Massachusetts

Degrees B.S. Mechanical Engineering, May 2010
M.S. Mechanical Engineering, December 2011

Publications T. Foote and R. K. Agarwal, "Power Generation from Wind Turbines in a Solar Chimney," ASME Paper ESFuelCell 2011-54085, Proc. of ASME 2011 Energy Sustainability and Fuel Cell Conference, Washington D.C, 7-10 August 2011.

T. Foote and R. K. Agarwal, "Optimization of Power Generation from Shrouded Wind Turbines," submitted for presentation at the AIAA Fluid Dynamics Conference, New Orleans, 25-28 June 2012.

December 2011