NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SHAPE FACTOR OPTIMIZATION OF MULTI-STAGE COMPRESSOR BLADE PROFILES

by

Matthew H. Pinney

June 2022

Thesis Advisor: Anthony J. Gannon
Co-Advisor: Christopher S. Clay

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2022	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
SHAPE FACTOR OPTIMIZATION OF MULTI-STAGE COMPRESSOR BLADE PROFILES

5. FUNDING NUMBERS

6. AUTHOR(S) Matthew H. Pinney

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

The purpose of this research was to develop a searchable, expandable database of compressor blade profiles optimized over a range of inlet flow angles for a desired outlet flow angle and to build a selection tool to interpolate new profiles from the database without performing the entire optimization process. Two optimization parameters were analyzed: boundary layer shape factor and blade surface pressure gradient. MATLAB, a matrix-based programming language, was used to perform the geometry generation, flow analysis, and optimization, and ANSYS CFX, a computational fluid dynamics program, was used to perform the fluid flow simulation. Three conclusive accomplishments were made. Firstly, both optimization parameters resulted in blades that turned the inlet flow to the desired outlet angle; however, the pressure-optimized blades resulted in greater total pressure recovery across a cascade and, thus, were chosen for the database. Secondly, the selection tool successfully used Delaunay triangulation to linearly interpolate new blade profiles that closely approximated the shape and performance of an optimized profile at the same set of inlet and outlet conditions. Finally, with sufficient entries in the database, the selection tool provided, at best, a faster method of generating compressor blade profiles or, at worst, initial conditions for the optimization parameters very close to the optimized solution.

14. SUBJECT TERMS compressor, shape factor, optimization, boundary layer, blade, database	**15. NUMBER OF PAGES** 141
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

SHAPE FACTOR OPTIMIZATION OF MULTI-STAGE COMPRESSOR BLADE PROFILES

Matthew H. Pinney
Ensign, United States Navy
BS, United States Naval Academy, 2021

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN AEROSPACE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2022

Approved by: Anthony J. Gannon
Advisor

Christopher S. Clay
Co-Advisor

Garth V. Hobson
Chair, Department of Mechanical and Aerospace Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The purpose of this research was to develop a searchable, expandable database of compressor blade profiles optimized over a range of inlet flow angles for a desired outlet flow angle and to build a selection tool to interpolate new profiles from the database without performing the entire optimization process. Two optimization parameters were analyzed: boundary layer shape factor and blade surface pressure gradient. MATLAB, a matrix-based programming language, was used to perform the geometry generation, flow analysis, and optimization, and ANSYS CFX, a computational fluid dynamics program, was used to perform the fluid flow simulation. Three conclusive accomplishments were made. Firstly, both optimization parameters resulted in blades that turned the inlet flow to the desired outlet angle; however, the pressure-optimized blades resulted in greater total pressure recovery across a cascade and, thus, were chosen for the database. Secondly, the selection tool successfully used Delaunay triangulation to linearly interpolate new blade profiles that closely approximated the shape and performance of an optimized profile at the same set of inlet and outlet conditions. Finally, with sufficient entries in the database, the selection tool provided, at best, a faster method of generating compressor blade profiles or, at worst, initial conditions for the optimization parameters very close to the optimized solution.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Previous Research 2
1.3 Project Objectives 8

2 Numerical Analysis **9**
2.1 Assumptions 9
2.2 Cascade Flow Analysis 9
2.3 Optimization Method. 18

3 Selection Tool Development **23**
3.1 Database Creation 23
3.2 Selection Tool 24

4 Computational Fluid Dynamics Setup **33**
4.1 Geometry 33
4.2 Meshing 35
4.3 Setup and Solution. 43
4.4 Post-Processing 46

5 Profile Validation **53**
5.1 Optimized Profile Performance 53
5.2 Multi-Stage Compressor Application 63

6 Conclusions and Recommendations **69**
6.1 Conclusions 69
6.2 Recommendations 70

Appendix A MATLAB Code **71**
A.1 aerogrid 71
A.2 aerosolve 78
A.3 Cam_opt_fun 84
A.4 camDatabase 87
A.5 camSelectionTool 88
A.6 Find_Cp_Max 91
A.7 fminsearch_2007_constraint 98
A.8 NACA_4_mid_slope 112

Appendix B CFX Report **115**
B.1 Example CFX Report 115

List of References **121**

Initial Distribution List **123**

List of Figures

Figure 1.1 Propeller blade geometry optimized through VLM. Source: [3] . 3

Figure 1.2 Propeller efficiency as a function of thrust coefficient. Source: [3] 3

Figure 1.3 Design variables for planform optimization. Source: [4] 4

Figure 1.4 VLM-optimized wing planforms. Source: [4] 5

Figure 1.5 Drag polar at a Mach number of 0.8. Source: [4] 6

Figure 1.6 Design process flow chart developed by Drayton. Source: [2] . . . 7

Figure 2.1 Cascade flow analysis workflow 10

Figure 2.2 Blade geometry parameter overlay on an example airfoil 12

Figure 2.3 Example blade profile grid 13

Figure 2.4 Example surface velocity profiles 15

Figure 2.5 Example pressure coefficient profiles 16

Figure 2.6 Example shape factor profiles 18

Figure 2.7 Optimization workflow 19

Figure 3.1 Selection tool example for known profile 25

Figure 3.2 Delaunay triangulation of the database 26

Figure 3.3 Enclosing Delaunay tetrahedron for $\beta_{1,min} = 42°$, $\beta_{1,max} = 53°$, and $\beta_2 = 23°$ 28

Figure 3.4 Selection tool interpolation example 29

Figure 3.5 Selection tool interpolation comparison with optimized profile . . 30

Figure 4.1 SolidWorks part generated for the H-optimized blade 34

Figure 4.2 SolidWorks part generated for the P-optimized blade 34

Figure 4.3 DesignModeler geometry for the P-optimized blade 35

Figure 4.4 Named surfaces of the fluid domain 36

Figure 4.5 Single-element-thick mesh generated by the use of a sweep method 37

Figure 4.6 Result of match control on the upper and lower boundaries of the fluid domain 38

Figure 4.7 Inflation layer along the surface of the blade 39

Figure 4.8 Details of the inflation layer 39

Figure 4.9 Representative y^+ profile for the H-optimized blade 40

Figure 4.10 Representative y^+ profile for the P-optimized blade 40

Figure 4.11 Overall mesh for the H-optimized blade profile 41

Figure 4.12 Overall mesh for the P-optimized blade profile 41

Figure 4.13 Mesh details for both the H-optimized (left) and P-optimized (right) profiles 42

Figure 4.14 Representative mass and momentum residuals for the H-optimized blade 45

Figure 4.15 Representative mass and momentum residuals for the P-optimized blade 46

Figure 4.16 Perpendicular lines used to determine the boundary layer 48

Figure 4.17 Example raw data boundary layer profile for the P-optimized blade 49

Figure 4.18 Example laminar boundary layer determined from fluid simulations 51

Figure 4.19 Example separated boundary layer determined from fluid simulations 51

Figure 4.20 Example turbulent boundary layer determined from fluid simulations 52

Figure 5.1 Profile comparison between the H-optimized and P-optimized blades for $\beta_{1,min} = 40°$, $\beta_{1,max} = 55°$, and $\beta_2 = 20°$ 53

Figure 5.2 Velocity contour of the H-optimized blade profile for $\beta_1 = 40°$ and $\beta_2 = 20°$ 54

Figure 5.3 Velocity contour of the H-optimized blade profile for $\beta_1 = 45°$ and $\beta_2 = 20°$ 55

Figure 5.4 Velocity contour of the H-optimized blade profile for $\beta_1 = 50°$ and $\beta_2 = 20°$ 55

Figure 5.5 Velocity contour of the H-optimized blade profile for $\beta_1 = 55°$ and $\beta_2 = 20°$ 56

Figure 5.6 Velocity contour of the P-optimized blade profile for $\beta_1 = 40°$ and $\beta_2 = 20°$ 57

Figure 5.7 Velocity contour of the P-optimized blade profile for $\beta_1 = 45°$ and $\beta_2 = 20°$ 57

Figure 5.8 Velocity contour of the P-optimized blade profile for $\beta_1 = 50°$ and $\beta_2 = 20°$ 58

Figure 5.9 Velocity contour of the P-optimized blade profile for $\beta_1 = 55°$ and $\beta_2 = 20°$ 58

Figure 5.10 Outlet angles achieved by the H- and P-optimized blade profiles . 59

Figure 5.11 Total pressure loss curves for the H- and P-optimized blade profiles 60

Figure 5.12 Shape factor profiles for the H-optimized and P-optimized blades: $\beta_2 = 20°$ 62

Figure 5.13 Query points for interpolation for the simplified three-stage compressor 65

Figure 5.14 Transformation from straight cascade to mixed-flow cascade. Source: [1] 66

Figure 5.15 Design of the simplified three-stage compressor 67

Figure 5.16 Interpolated compressor blade velocity contour 68

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

Table 2.1 List of blade geometry parameters 11

Table 5.1 Required inlet/outlet conditions for simplified three-stage compressor 64

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

CAD	Computer-Aided Design
CFD	Computational Fluid Dynamics
NACA	National Advisory Committee for Aeronautics
NPS	Naval Postgraduate School
RANS	Reynolds-Averaged Navier-Stokes
USN	U.S. Navy
VLM	Vortex Lattice Method

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

1.1 Motivation

The introduction of programming languages, computer-aided design (CAD) tools, and computational fluid dynamics (CFD) programs revolutionized mechanical and aerospace engineering. Gone were the days of free-hand drawings and extensive hand calculations; computers were able to perform all of the necessary calculations in a fraction of the time. The ramifications of varying design parameters could be analyzed almost instantly. Capitalizing on this increase in computational power to rapidly design turbomachinery was inevitable. Through the clever use of each of these technological advances, compressors can be designed and reasonably evaluated without manufacturing a single part. Ultimately, a multi-stage, process gas, axial compressor could be designed through the automated use of commercially available tools by employing optimization techniques. The vortex lattice method (VLM), in particular, is well-suited for optimization due to its minimal computational requirements. VLM can be iterated thousands of times in order to obtain good starting points for aerodynamic designs before resorting to CFD simulations.

The design of a compressor presents a complex optimization challenge. To start, there are a wide range of potential cost functions from which to choose: maximizing the pressure ratio, increasing the stall margin, increasing efficiency, etc. Often, maximizing one of these performance metrics results in the reduction of another. In addition, the optimal solution may change quite drastically for a small change in the inlet/outlet flow of the compressor. Therefore, the entire design process must be repeated in order to achieve an optimal solution for the new inlet conditions. Despite the advances in computational power, conducting optimization processes still requires vast amounts of computational time and resources. A useful compressor design tool must be resistant to the computational requirements of optimization techniques.

Compiling a database of optimized compressor blade profiles would circumvent the computational requirements on the user end. For any set of desired inlet/outlet conditions, a user

could interrogate a database of blade profiles optimized at various predefined conditions. If the optimized blade profile for the given desired inlet/outlet conditions does not exist in the database, it would be interpolated from the optimized profiles that do exist. Thus, a close approximation of the required blade profile could be provided in mere moments, rather than going through the entire optimization process to determine the required blade profile. The vast computational effort of optimization would only be required to populate the "standard" profiles in the database. The process of populating the database would only have to be performed once, provided the interpolated profiles closely approximate their optimized counterparts.

1.2 Previous Research

Design techniques for axial compressors have been thoroughly investigated, most commonly through the application of VLM. Lewis [1] developed an entire design methodology for blade cascades using VLM, which forms the basis of the work performed in this thesis. The particulars of this method will be expanded upon throughout this paper. In addition, the discrete boundary layer method outlined by Grimson [2] provided a framework by which to determine boundary layer characteristics along a blade. The application of this method is outlined further in Section 2.2.

Recently, Du and Kinnas applied the VLM to a numerical optimization technique for the design of cavitating propellers [3]. The goal was to minimize the torque required to produce a given amount of thrust by varying chord length, camber, pitch, and skewness of the propeller blades. The resulting blade geometry is shown in Figure 1.1, while the efficiency of the propeller is plotted against the thrust coefficient in Figure 1.2.

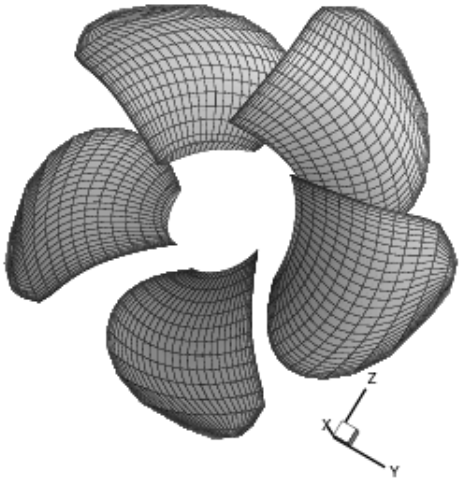


Figure 1.1. Propeller blade geometry optimized through VLM. Source: [3]

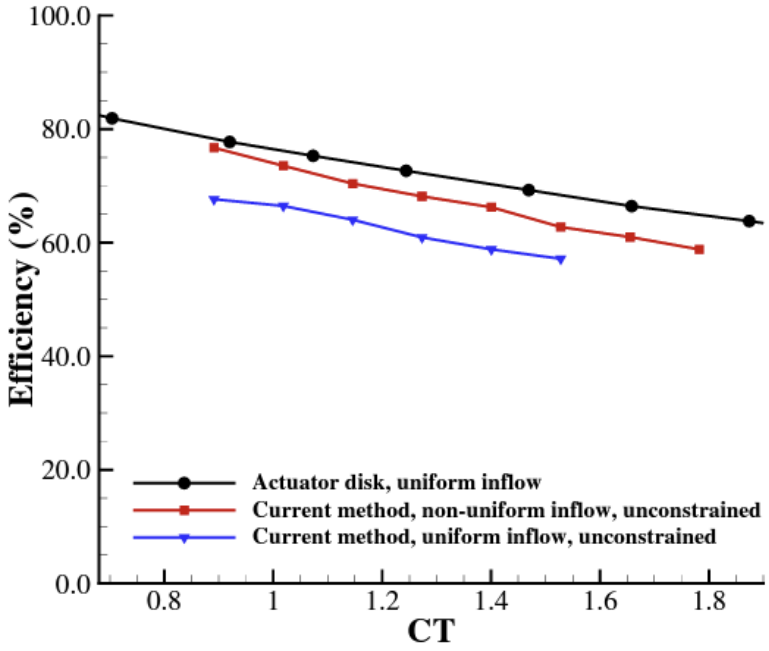


Figure 1.2. Propeller efficiency as a function of thrust coefficient. Source: [3]

The efficiency of the resulting blades compared favorably to the ideal efficiency derived from an actuator disk model of the propeller, especially when considering non-uniform inflow. This suggested that the VLM remains a useful tool in the design and optimization of blade geometries.

Furthermore, Parenteau, Laurendeau, and Carrier performed an aerodynamic optimization of transonic aircraft wings through the application of the VLM and Reynolds-Averaged Navier-Stokes (RANS) sectional data [4]. The goal was to maximize the lift-to-drag ratio of a constant planform area wing by varying root/tip chord, root/tip twist, leading/trailing edge sweep, and kink position (Figure 1.3).

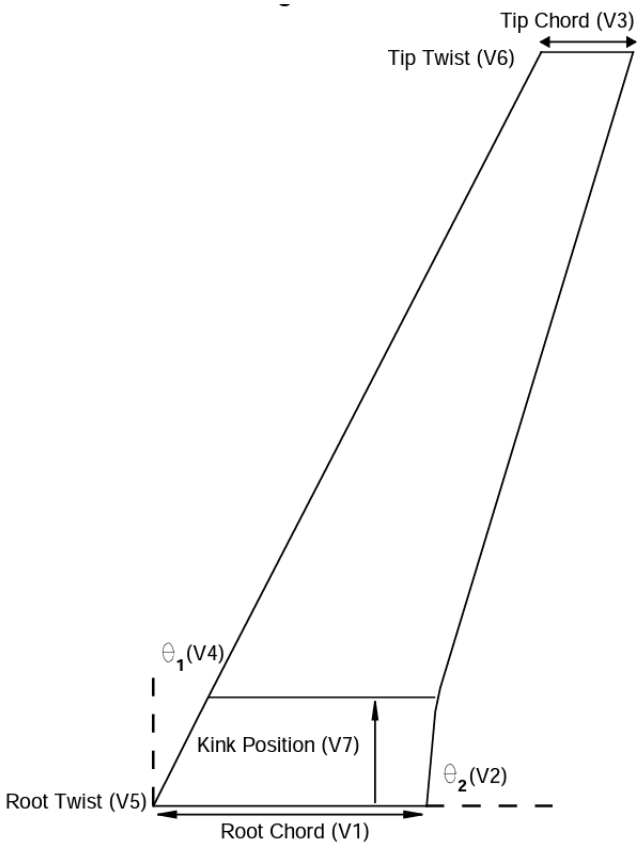


Figure 1.3. Design variables for planform optimization. Source: [4]

The optimization procedure resulted in the planforms shown in Figure 1.4, while the resulting drag polar at a Mach number of 0.8 is shown in Figure 1.5.

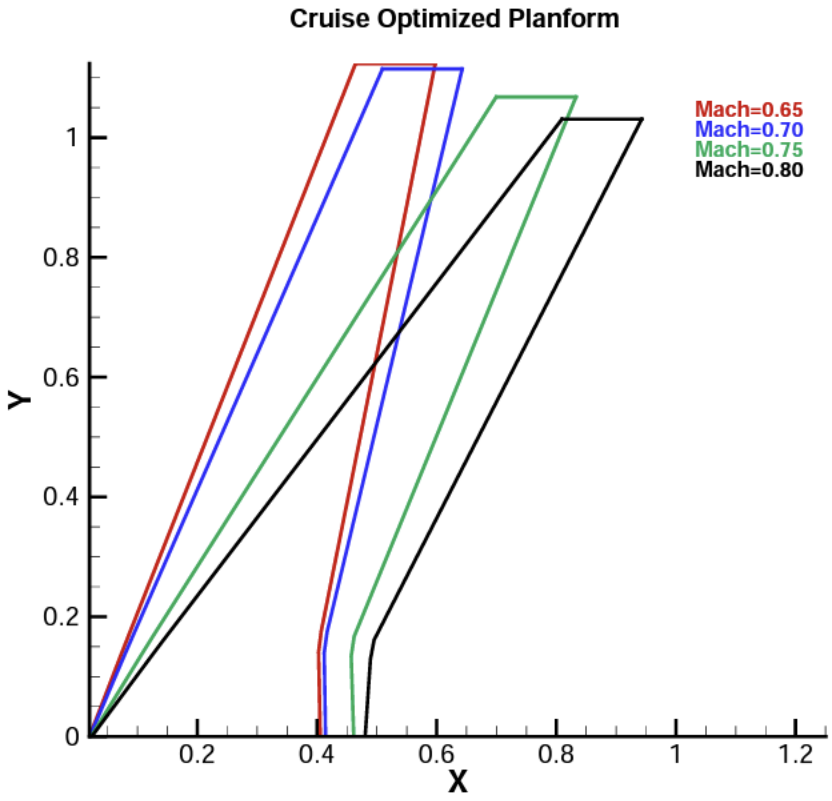


Figure 1.4. VLM-optimized wing planforms. Source: [4]

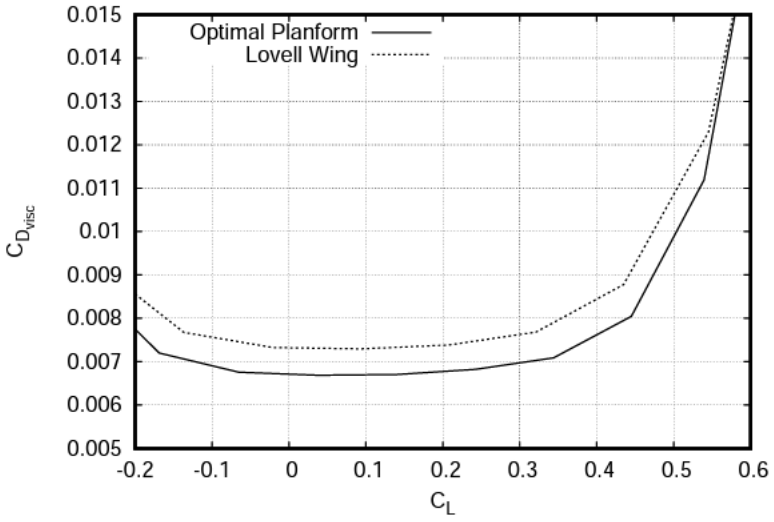


Figure 1.5. Drag polar at a Mach number of 0.8. Source: [4]

From Figure 1.5, it can be seen that the planform optimization reduced the drag of the wing for a given lift coefficient when compared to the reference wing used in the study. This, once again, validated the use of the VLM in numerical optimization methods.

Drayton [5] developed a new design procedure in 2013 for splitter blades in a transonic axial compressor based on the design methodology created by Sanger [6]. This methodology was adapted by Drayton into an automated design process through the integration of three available programs: 1) MATLAB, a math-based programming language specializing in matrix operations, 2) SolidWorks, a CAD tool for solid parts, and 3) ANSYS-CFX, a CFD simulation tool. Drayton used MATLAB to automate the entire design procedure from parameter initialization and geometry generation to the operation of the other two programs. This process allowed Drayton to analyze the expected performance characteristics for a given set of parameters and determine the best blade geometry solution with minimal manual input. This blade geometry was then manufactured for experimental testing. The iterative design process flow chart created by Drayton is given below in Figure 1.6.

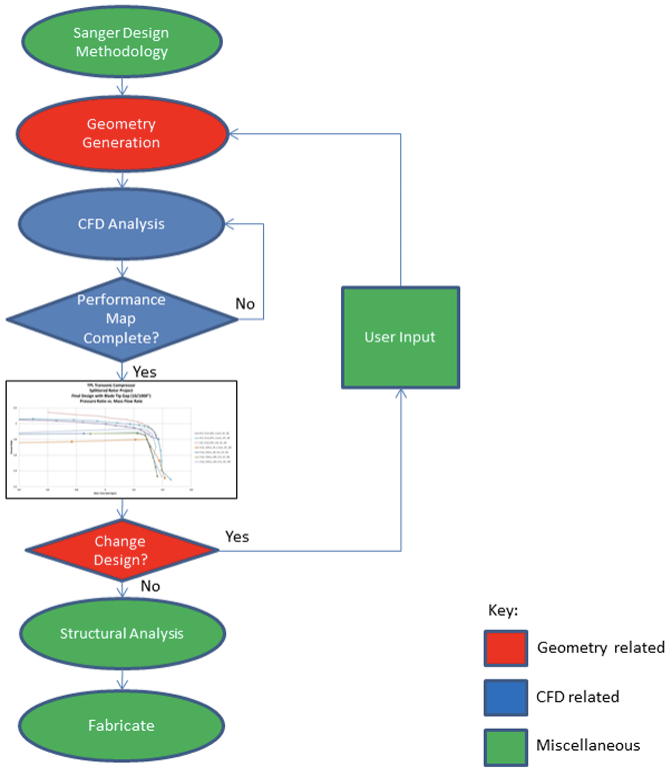


Figure 1.6. Design process flow chart developed by Drayton. Source: [2]

The iterative design process created by Drayton was expanded upon by Brantner [7] through the application of various optimization techniques. Brantner used compressor performance parameters obtained from fluid simulations such as the isentropic efficiency, η, pressure ratio, P_r, and the mass flow, $\dot{m}$, to develop an optimization function (Equation 1.1) for the compressor.

$$Z = \sum_{i=1}^{n} \frac{(\eta_{i-1} P_{r_{i-1}}) - (\eta_i P_{r_i})}{2} \Delta \dot{m}_i \tag{1.1}$$

By iterating blade geometry parameters until Equation 1.1 reached a maximum, Brantner was able to, theoretically, optimize the compressor design process; however, there were

issues with the optimization method. Firstly, the CFD data used in the optimization function was inconsistent, resulting in discrepancies between various expected and calculated values. Second, the optimization method often failed to find the global maximum of the optimization function, settling into local maxima instead. In other words, the optimization function did not optimize certain blades. Thus, a more robust optimization method, able to discriminate between local and global maxima, such as the Nelder-Mead method [8], is critical to the optimization of compressor designs. The specifics of this method are outlined in Section 2.3.

1.3 Project Objectives

The main thrust of this thesis is the creation of a compressor blade profile selection tool that, given a range of desired inlet angles and a fixed outlet angle, interpolates a database of optimized blade profiles, resulting in the generation of a blade profile that satisfies the user input. The objectives that must be met to develop such a selection tool include the following:

1. Implementing VLM and Pohlhausen's method [2] for approximating boundary layers over a discrete set of points constituting a two-dimensional compressor blade profile (Section 2.2)

2. Developing an optimization method that modifies the shape, pitch, and stagger of the blade profile in order to minimize a specified cost function (Section 2.3)

3. Building the database of optimized blade profiles and the interpolation method used by the selection tool to generate new blade profiles (Chapter 3)

4. Validating the optimization results by applying CFD simulations to determine a more realistic estimation of the aerodynamic characteristics and performance of the optimized blades (Chapters 4-5)

CHAPTER 2:
Numerical Analysis

2.1 Assumptions

Real compressor blades are three-dimensional, and blade cross-sections vary radially. For simplicity, only a single cross-section will be considered. In other words, two-dimensional flow will be assumed. In addition, the incoming flow will be assumed to be uniform.

2.2 Cascade Flow Analysis

The process by which the cascade flow analysis was performed involved the use of various interconnected MATLAB scripts, many of which were written by Gannon [9] at the NPS Turbopropulsion Laboratory. The main flow analysis script was "Find_Cp_max.m", in which other functions were called to perform the geometry generation and aerodynamic calculations. Each script used for the cascade flow analysis and their relationship to one another is listed below in Figure 2.1. Scripts written primarily by Gannon [9] are denoted as such. The function and details of each script will be expanded upon in the following sections. The scripts can be found in full in Appendix A.

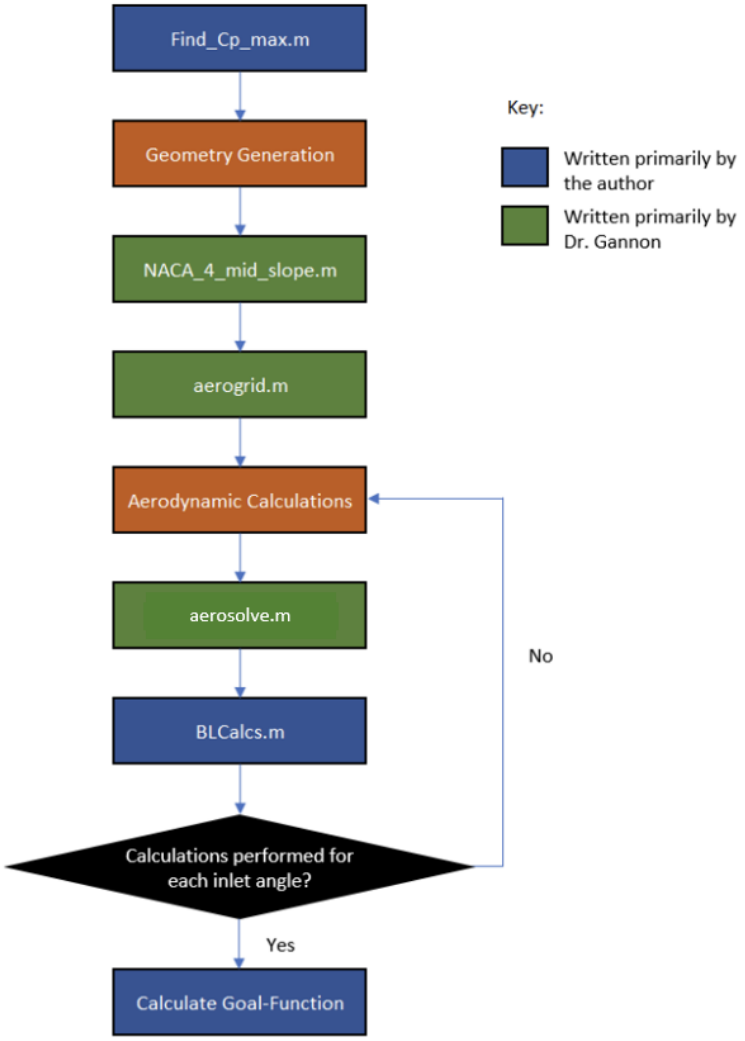


Figure 2.1. Cascade flow analysis workflow

2.2.1 Generating the Blade Geometry

There are multiple ways to define the geometry of an airfoil. For this study, blade geometries were divided into the suction-surface (top) and pressure-surface (bottom) and were defined using fourteen parameters: the twelve required for modified National Advisory Committee

for Aeronautics (NACA) 4-series airfoils presented by Abbott and von Doenhoff [10], along with the pitch between blades and blade stagger. Using modified NACA 4-series airfoils, rather than standard NACA 4-series, introduced the trailing edge slope and nose radius as independent parameters for both the top and bottom surfaces. This increased the degrees of freedom of the blade geometry by four, allowing for more possible geometries to be examined. Thirteen of these parameters (all except for the blade stagger) served as the input argument passed to "Find_Cp_max.m." The full list of parameters is provided in Table 2.1.

Table 2.1. List of blade geometry parameters

#	Parameter
1	Max. Camber (top)
2	Max. Camber (bottom)
3	Position of Max. Camber (top)
4	Position of Max. Camber (bottom)
5	Max. Thickness (top)
6	Max. Thickness (bottom)
7	Position of Max. Thickness (top)
8	Position of Max. Thickness (bottom)
9	Trailing Edge Slope (top)
10	Trailing Edge Slope (bottom)
11	Nose Radius (top)
12	Nose Radius (bottom)
13	Pitch
14	Stagger

The top surface parameters, along with the pitch and stagger, are overlaid onto an example airfoil in Figure 2.2.

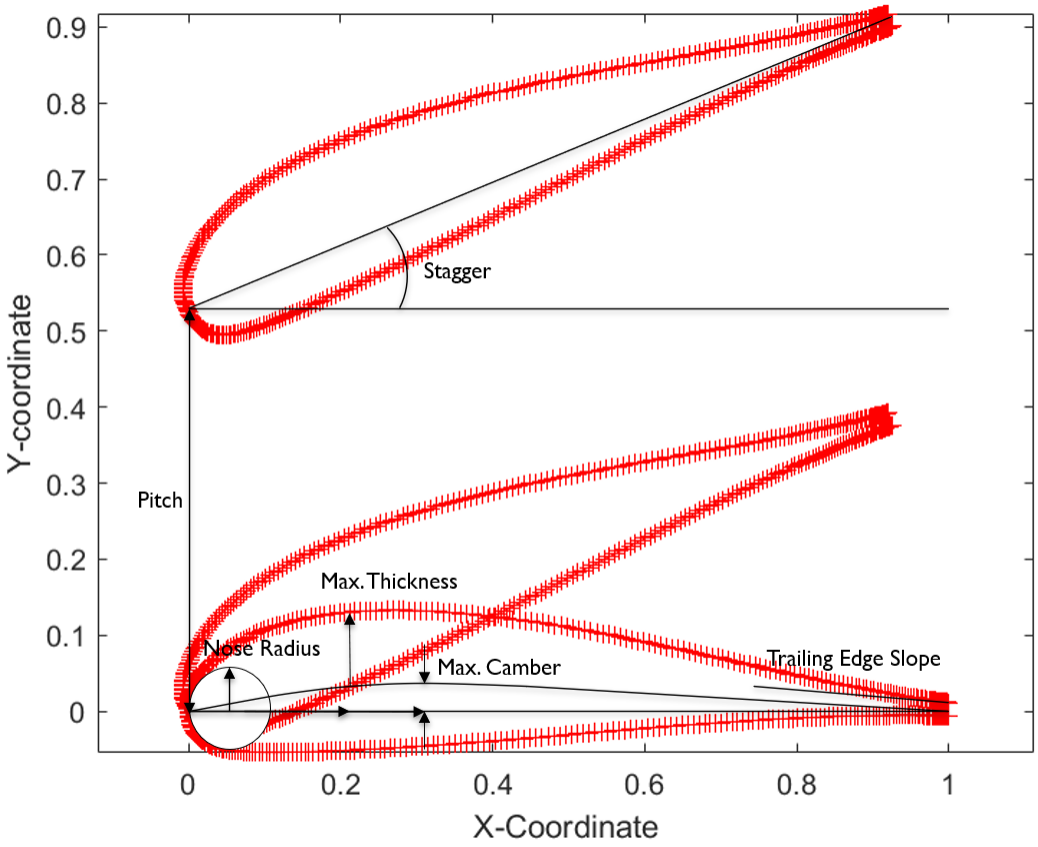


Figure 2.2. Blade geometry parameter overlay on an example airfoil

Within "Find_Cp_max.m," the input parameters are passed to "NACA_4_mid_slope.m." This function calculates the leading and trailing edge thickness polynomial coefficients that define the surface of a blade for the given parameters. Because the top and bottom surfaces are defined independently, their respective parameters are passed to "NACA_4_mid_slope.m" separately, and individual leading/trailing edge polynomials are calculated for each surface. The thickness polynomial coefficients, along with the camber polynomial coefficients (which can be calculated directly from the parameters), are then passed to "aerogrid.m" to generate the grid points around the blade profile. "aerogrid.m" uses the polynomial coefficients and a desired number of grid points to generate the Cartesian coordinates for the specified number of points along the surface of the blade. These calculations are also performed separately between the top and bottom surfaces of the blade.

The x- and y-coordinates for the top and bottom surfaces are then combined into their respective arrays. The final result is a collection of x- and y-coordinates defining a discretized blade surface, which closely approximates a smooth surface for a sufficient number of grid points. Figure 2.3 shows an example of a blade surface generated using the method described in this section, with the leading/trailing edge polynomials differentiated by color.

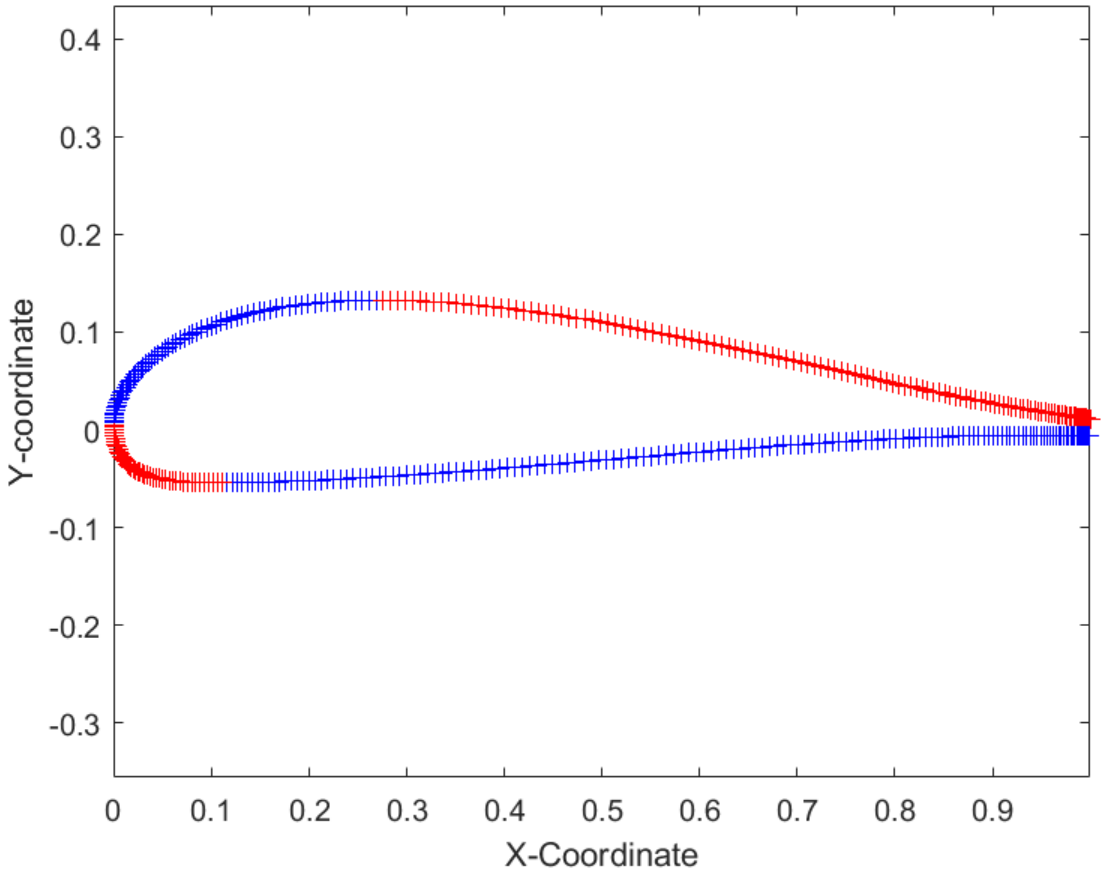


Figure 2.3. Example blade profile grid

2.2.2 Aerodynamic Calculations

In order to meet the outlet flow angle requirement, the blade geometry generated above must, at every inlet angle within the inlet angle range, turn the incoming flow the proper amount. To achieve the required flow turning, the stagger of the blade was adjusted until the aerodynamic calculations performed in "aerosolve.m" resulted in a predicted outlet flow angle within a specified tolerance of the required outlet flow angle. For a given blade stagger, the predicted outlet flow angle was calculated at each inlet angle. If the predicted outlet flow angle of any inlet angle was outside of the tolerance, the stagger was adjusted (increased if predicted angle was too low and decreased if predicted angle was too high). The proper blade stagger was determined only when every predicted outlet flow angle was within the tolerance.

The flow analysis performed in "aerosolve.m" was the VLM for a blade cascade developed from the method outlined by Lewis [1]. The inputs for this function were the x- and y-coordinates of the unstaggered blade determined from "aerogrid.m," the pitch between two consecutive blades, the blade stagger, and the inlet flow angle. The outputs included the staggered blade profile grid points, the distance traveled along the blade from the stagnation point, the flow velocity profile along the blade, the pressure coefficient profile along the blade, and the predicted outlet flow angle. Within the function, the grid points were, first, oriented to the stagger angle via a rotation. These grid points were then used to find the vortex elements (line segments) connecting consecutive grid points. The midpoints of these elements served as the locations at which the vortices were centered. A coupling matrix was then calculated. The coupling matrix determined the flow characteristics at each vortex location, accounting for the effects of a blade cascade on the flow around each blade. The coupling matrix was a function of locations of the vortex elements and the pitch between consecutive blades in the cascade.

From the coupling matrix and the vortex element slopes, the circulation, x-velocity, and y-velocity at each point along the blade was calculated. The predicted outlet flow angle was then determined from the total circulation of the blade vortices and the inlet flow angle. The x- and y-velocities were used to determined other flow characteristics around the blade including total surface velocity, surface velocity gradient, pressure coefficient, and pressure coefficient gradient. The surface velocity and pressure coefficient profiles for the example

blade shown in Figure 2.3 are depicted in Figures 2.4 and 2.5. For these profiles, the inlet angle range was $\beta_{1,min} = 40°$ to $\beta_{1,max} = 55°$ with an outlet angle $\beta_2 = 20°$.

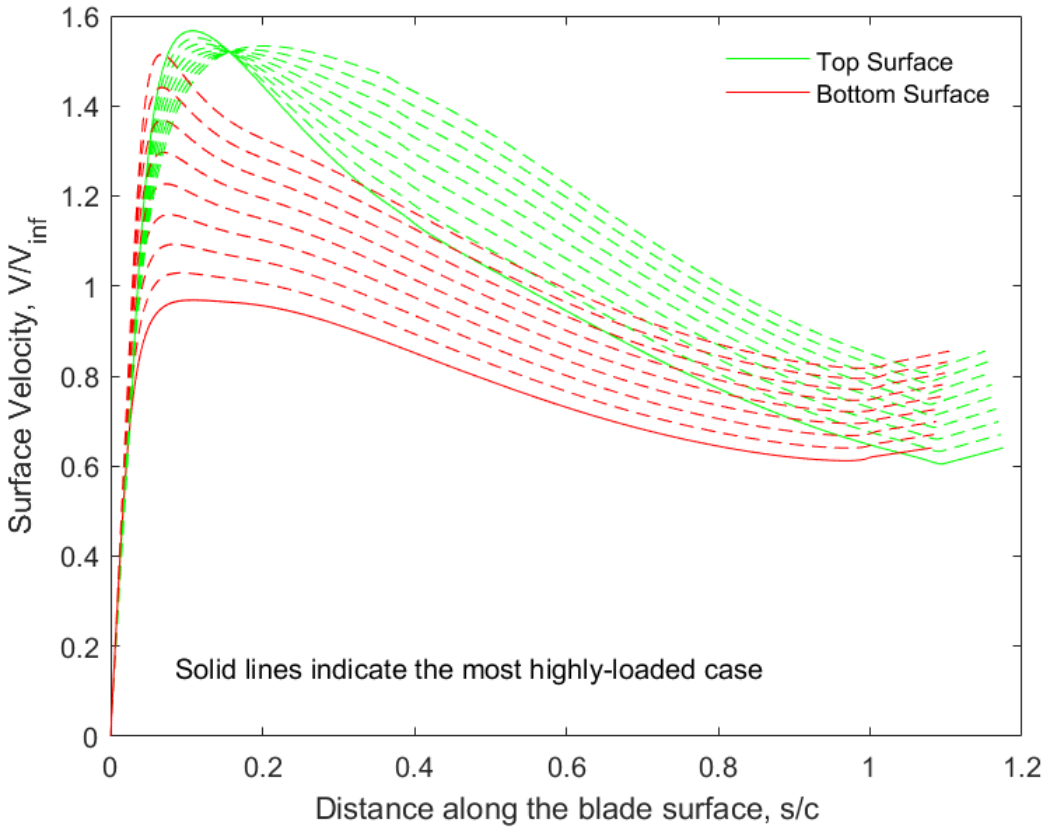


Figure 2.4. Example surface velocity profiles for the blade shown in Figure 2.3: $\beta_{1,min} = 40°$, $\beta_{1,max} = 55°$, and $\beta_2 = 20°$

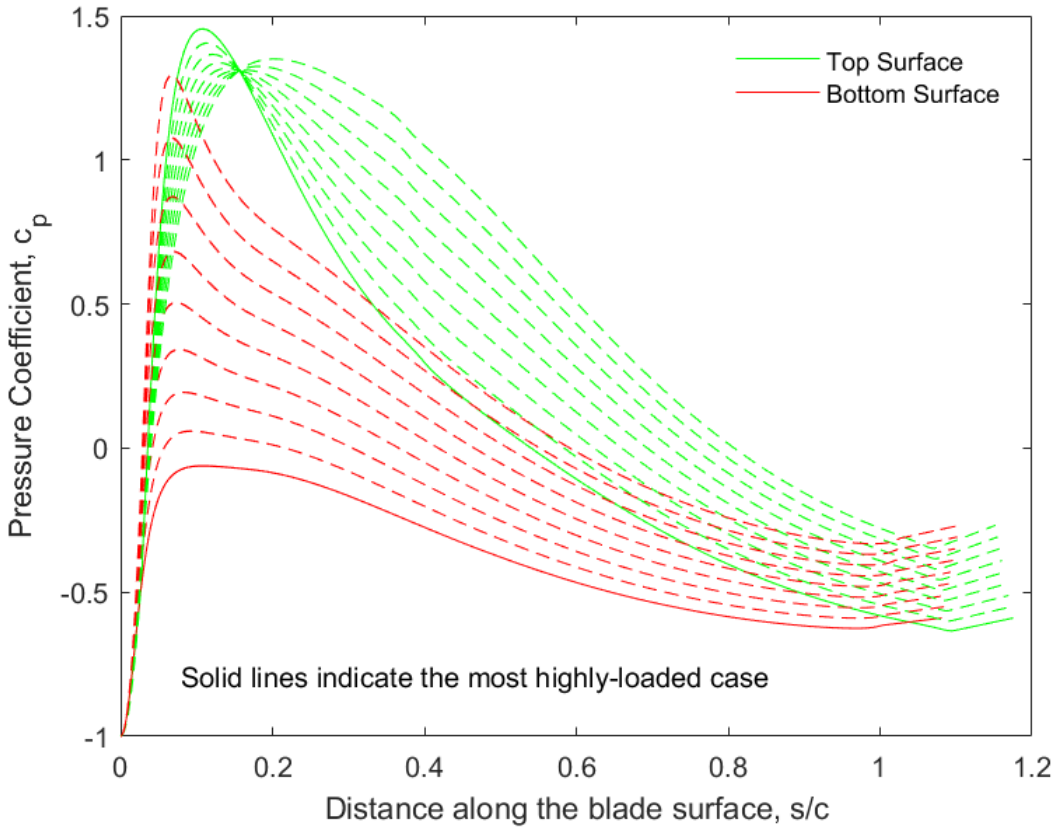


Figure 2.5. Example pressure coefficient profiles for the blade shown in Figure 2.3: $\beta_{1,min} = 40°$, $\beta_{1,max} = 55°$, and $\beta_2 = 20°$

As expected, there is a strong positive velocity gradient following the stagnation point on the both the top and bottom surface of the blade. The flow then gradually decelerates on both surfaces; however, the velocity on the top surface consistently remains higher than on the bottom surface, which is typical of an upper surface of a blade generating lift. The pressure coefficient follows the same trend as the surface velocity. There is a strong favorable pressure gradient immediately after the stagnation point, which gives way to an adverse pressure gradient. It is worth noting that the top and bottom surfaces change length for different inlet angles because the stagnation point moves depending on the inlet angle. As the inlet flow angle increases, the stagnation points moves down, causing the top surface to increase in length and the bottom surface to decrease in length.

After the stagger angle was determined, the boundary layer calculations for each inlet angle was performed in "BLCalcs.m" using the velocity profiles calculated in "aerosolve.m." "BLCalcs.m" employed Pohlhausen's discrete method for determining boundary layer characteristics, following the procedure created by Grimson [2]. This method assumed two-dimensional laminar flow over an airfoil, where the velocity distribution in the boundary layer has two components. The first component is the distribution associated with zero-pressure gradient flow, whereas the second component accounts for the effects of a non-zero pressure gradient. The shape factor for a given boundary layer profile could then be interpreted as the magnitude of the non-zero pressure gradient component of the velocity distribution. "BLCalcs.m" uses the flow velocities and velocity gradients along the surface of the blade to determine this shape factor at each point on the blade. The shape factor calculations were performed at each inlet flow angle in order to determine the shape factor profile for each case. The shape factor profiles for the example blade (Figure 2.3) are shown in Figure 2.6 for the same inlet and outlet conditions as above. At the stagnation point, the shape factor is always zero, since there is no flow tangent to the blade surface. Thus, the stagnation point is omitted from Figure 2.6 in order to reduce the scale of the vertical axis.

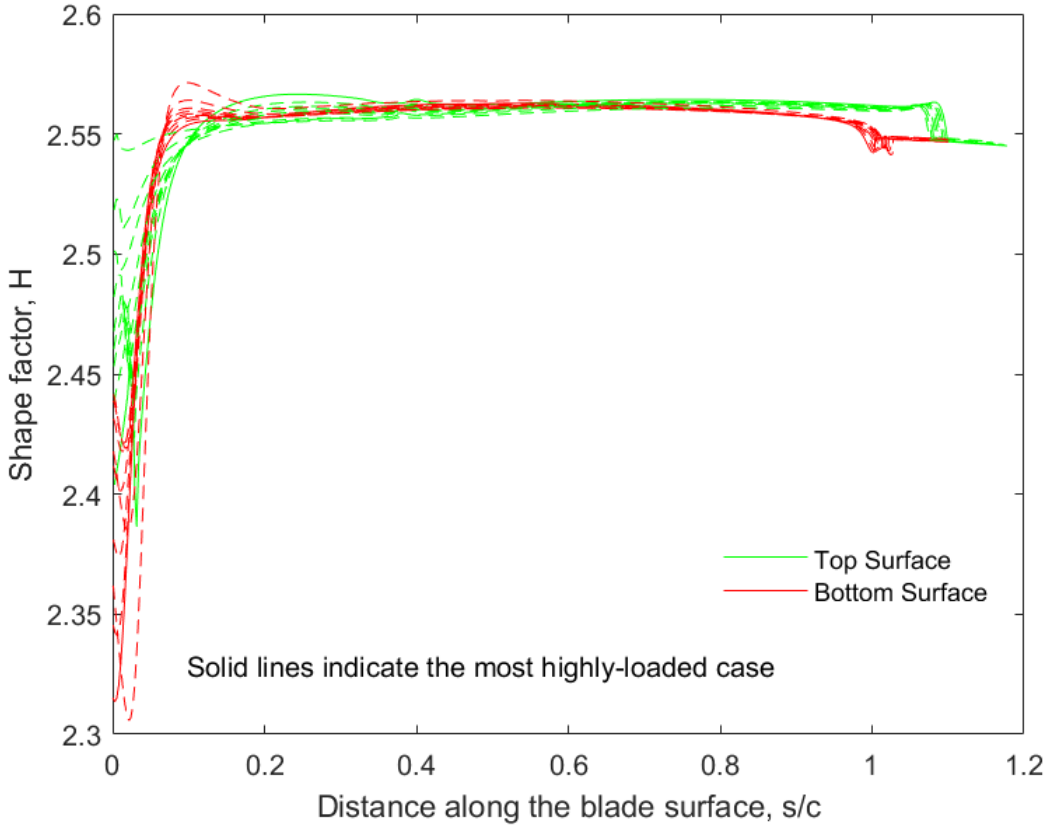


Figure 2.6. Example shape factor profiles for the blade shown in Figure 2.3: $\beta_{1,min} = 40°$, $\beta_{1,max} = 55°$, and $\beta_2 = 20°$

The shape factor near the leading edge of the blade quickly rises from zero at the stagnation point and then begins to level off. This occurs at roughly the same point on the blade where the velocity gradients shift from positive to negative. The vast majority of the rise in the shape factor occurs during the favorable pressure gradient. In the adverse pressure gradient, the shape factor remains relatively constant, fluctuating only slightly.

2.3 Optimization Method

The goal of the optimization method was to determine the blade profile parameters that minimized a specified cost function. Like the cascade flow analysis, the optimization

method used multiple scripts within MATLAB. The over-arching optimization function was "Cam_opt_fun.m," which called "Find_Cp_max.m" and "fminsearch_2007_constraint.m" in order to determine the optimized blade profile. The workflow of these scripts is shown in Figure 2.7, and each one will be examined in further detail in this section. Each of these scripts can be found in Appendix A.

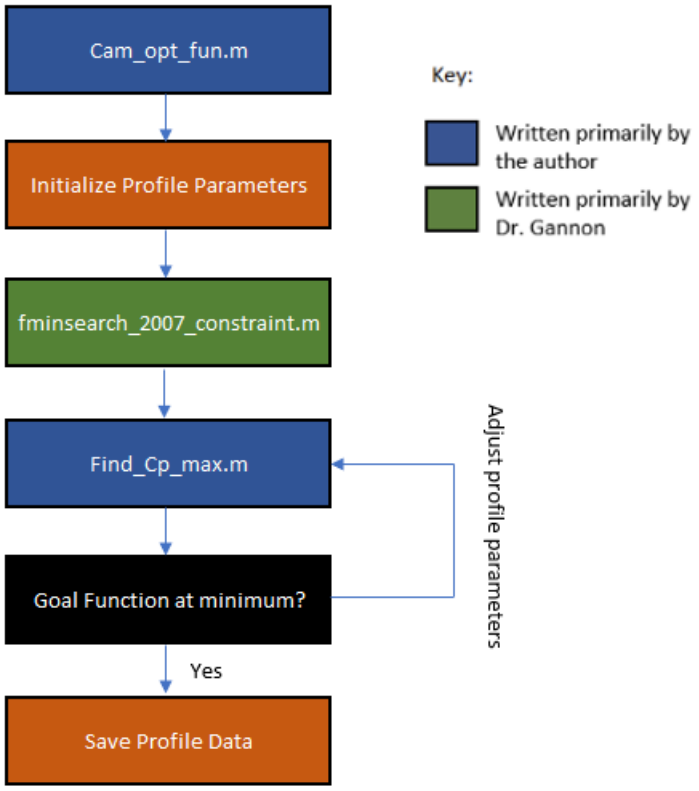


Figure 2.7. Optimization workflow

2.3.1 The Cost Function

The cost function was the output of "Find_Cp_max.m" and was calculated using the results of the cascade flow analysis above. In this project, two cost functions were considered: one based on shape factor and one based on the pressure gradient. Shape factor was selected

as the basis of the first cost function because it is an indicator of the proximity of the boundary layer to separation. The pressure gradient was selected as the basis of the second cost function because strong adverse pressure gradients decelerate flow, often to the point of separation. The shape factor cost function was the sum of the maximum shape factors along the top and bottom blade surfaces at each inlet flow angle, as shown in Equation 2.1, where n represents the number of inlet flow angles within the given range used to perform the optimization. By summing the maximum shape factors at each inlet flow angle for the cost function, the optimization produces blade profiles that work over the entire range of inlet angles, not just at one inlet angle.

$$F_H = \sum_{i=1}^{n} \left(H_{\max,\text{top}_i} + H_{\max,\text{bottom}_i} \right) \tag{2.1}$$

Because the shape factor was only calculated at discrete points along the blade, it is possible that the "true" maximum shape factor was not contained within the points on the blade grid. Therefore, a quadratic function was fit to each set of three consecutive points on the blade. In effect, the maximum of this quadratic function approximated the largest shape factor that could be expected in the regions between the grid points by making the goal function continuous even though it is based on a finite grid over the blade. The largest maximum shape factor found using the quadratic approximations was taken to be the "true" maximum shape factor.

The pressure gradient goal function was the sum of the maximum pressure gradient along the top and bottom blade surfaces at each inlet flow angle, as shown in Equation 2.2.

$$F_P = \sum_{i=1}^{n} \left(\frac{dp}{dx}_{\max,\text{top}_i} + \frac{dp}{dx}_{\max,\text{bottom}_i} \right) \tag{2.2}$$

Before calculating the pressure gradient goal function, a weighting factor was applied to each point along the blade surface. This weighting factor artificially increased the pressure gradient near the leading edge of the blade in order to delay the transition to an adverse pressure gradient. Delaying this transition assists in keeping the flow accelerating near the leading edge, which prevents flow separation.

2.3.2 Minimizing the Cost Function

The optimized blade profile for a given set of inlet and outlet conditions is the set of blade shape parameters that minimized the chosen cost function. The script "Cam_opt_fun.m" accepts desired inlet and outlet conditions as its input and outputs the optimized blade shape parameters. First, the script initializes all of the blade shape parameters (except for stagger, which is determined through "aerosolve.m") by using a standard set of values. The blade shape parameters are then passed to "fminsearch_2007_constraint.m," which performs the optimization.

"fminsearch_2007_constraint.m" is a version of MATLAB's built-in "fminsearch" function modified to be able to constrain the values to which the blade parameters are allowed to change. The "fminsearch" function uses a simplex to perform a multi-dimensional non-linear minimization of a specified function based on the Nelder-Mead method [8]. Using "Find_Cp_max.m" as the function to which the optimization method is applied, "fminsearch_2007_constraint.m" will attempt to minimize the chosen cost function from Section 2.3.1 by tweaking the blade shape parameters. After a sufficiently large number of iterations, the optimization method reaches convergence, indicating that the cost function is at its minimum value. In this case, "fminsearch_2007_constraint.m" returns the set of blade shape parameters that produced the minimum value of the cost function. These parameters are then passed to "Find_Cp_max.m" one more time in order to determine the required blade stagger and associated velocity, pressure, and shape factor profiles. The blade shape parameters and the stagger angle are then saved to a ".mat" file so that the resulting blade profiles can be added to the database explored in the following chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Selection Tool Development

The optimization method developed in Section 2.3, while much faster than even a two-dimensional simulation, is still time-consuming and computationally intensive. A single profile optimization for a specific β_1 range and desired β_2 takes roughly three hours. Each case only needs to be optimized once if the resulting profile is stored in a searchable, expandable profile database. With access to such a database and a selection tool, a user could simply "look up" the optimized profile for the desired inlet and outlet conditions in a matter of moments with no need for the entire optimization process. However, the likelihood that the user's desired inlet and outlet conditions exactly match with the contents of the database is very small. Therefore, the selection tool must be able to generate new profiles without going through the tedious optimization process. By interpolating between optimized profiles in the database for the conditions nearest to the desired conditions, the selection tool would be able to quickly create new profiles that closely approximate the "ideal" optimized profile for the desired conditions.

3.1 Database Creation

The database of optimized blade profiles is a collection of the fourteen optimization parameters (listed in Section 2.3) that characterize the blade shape, pitch, and stagger for each set of inlet and outlet conditions. The optimization function automatically saves this data, along with the desired inlet and outlet conditions, to ".mat" files with a standard naming convention. The naming convention incorporates both the inlet range and the outlet angle in order to distinguish between different profiles. For example, a blade optimization for β_1 from 40° to 55° with a β_2 of 20° would result in the following file name: "Inlet_55to40_Outlet_20.mat." The naming convention is not required for proper functioning of the database, but serves as an aid to the user. The inlet/outlet conditions are stored within the files and read upon opening the file. There are no limitations on which blade optimizations can be added to the database. Blades with non-integer inlet/outlet angles can be added to the database just like blades with integer inlet/outlet angles.

The database is created in the "camDatabase.m" script (Appendix A). The script begins by searching the current directory for all of the files that conform to the specified naming convention. Only the data in files with the naming convention will be added to the database. Next, the script initializes a matrix with zeros in every location. The number of rows in this matrix is dependent on the number of ".mat" files that are in the current directory, while the number of columns is always seventeen. The seventeen columns are drawn from the three inlet/outlet conditions and the fourteen blade parameters. To populate each element of the matrix, a for-loop iterates through each file, pulling the inlet/outlet conditions and optimization parameters from the file and placing them into the proper element. When the for-loop is finished, the resulting array is saved into a separate file, "camDatabase.mat." This file now contains the inlet/outlet conditions and blade parameters for every optimized blade. Instead of needing all of the different blade profile files, a user would only need the single "camDatabase.mat" file to have access to all of the same information. In addition, new blades can be added to the database by simply running the script again, so long as the new blade profiles satisfy the naming convention. This allows for data to be added to the database near points of interest for more accurate interpolation, or for better resolution of edge cases in the data.

3.2 Selection Tool

The selection tool is essentially a look-up table that either returns known optimized profiles or interpolates new profiles based on the desired inlet/outlet conditions specified. If the desired inlet/outlet conditions fall outside of the scope of the database (see Section 3.1), then the selection tool provides a warning that the new profile was extrapolated and, thus, may not approximate the optimized blade profile.

The selection tool is the "camSelectionTool.m" script (Appendix A). The script begins by receiving inputs for the desired inlet/outlet conditions. The database from "camDatabase.mat" is loaded into the selection tool, and the tool immediately checks whether the optimized profile for the desired inputs already exists. If the optimized profile exists, the selection tool chooses the appropriate row of blade shape, pitch, and stagger parameters from the database and calculates the x- and y-coordinates for a blade associated with those parameters. The tool then plots the un-staggered, un-pitched blade profile along with the staggered, un-pitched blade profile and staggered, pitched blade profile as shown in Figure 3.1.

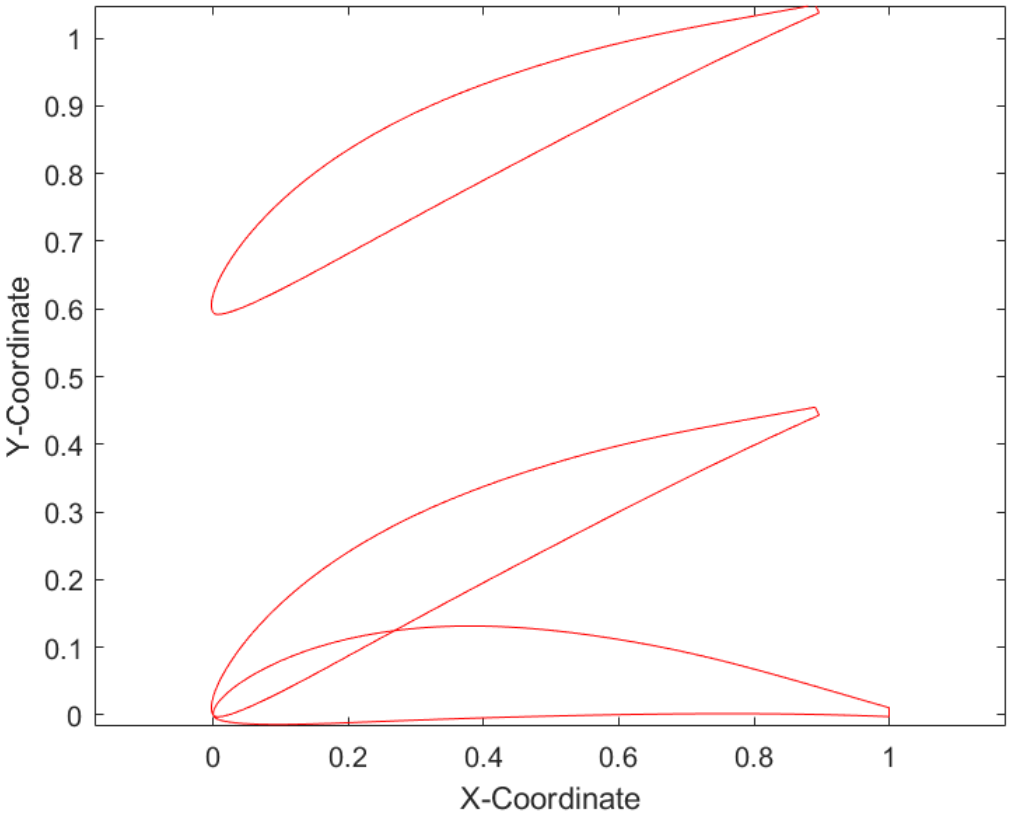


Figure 3.1. Selection tool example for known profile: $\beta_{1,min} = 40°$, $\beta_{1,max} = 55°$, and $\beta_2 = 20°$

If the inputs to the selection tool do not correspond to an existing profile but remain within the scope of the database, then the selection tool interpolates a new profile for the desired inputs from the parameters of known optimized profiles using the built-in MATLAB function "griddata." The interpolation is performed on the parameters, rather than the coordinates of the blade, because knowing the parameters of the interpolated profile provides a great starting point for an optimization at the desired inputs. This had the added benefit that the interpolated profiles remained consistent with the definition of a modified NACA 4-series airfoil, including trailing edge slope and nose radius. The "griddata" function interpolates scattered data by using Delaunay triangulation to determine the nearest known data points and the parameter values at those points. Since there are three inlet/outlet conditions, the

Delaunay triangulation creates tetrahedra, using the inlet/outlet conditions in the database, that satisfy the empty circumsphere criterion. This criterion requires that no other known data point lies within the sphere that passes through four data points. This avoids the creation of sliver tetrahedra, which can produce poor interpolation results. An example of the Delaunay triangulation performed on the database is shown in Figure 3.2.

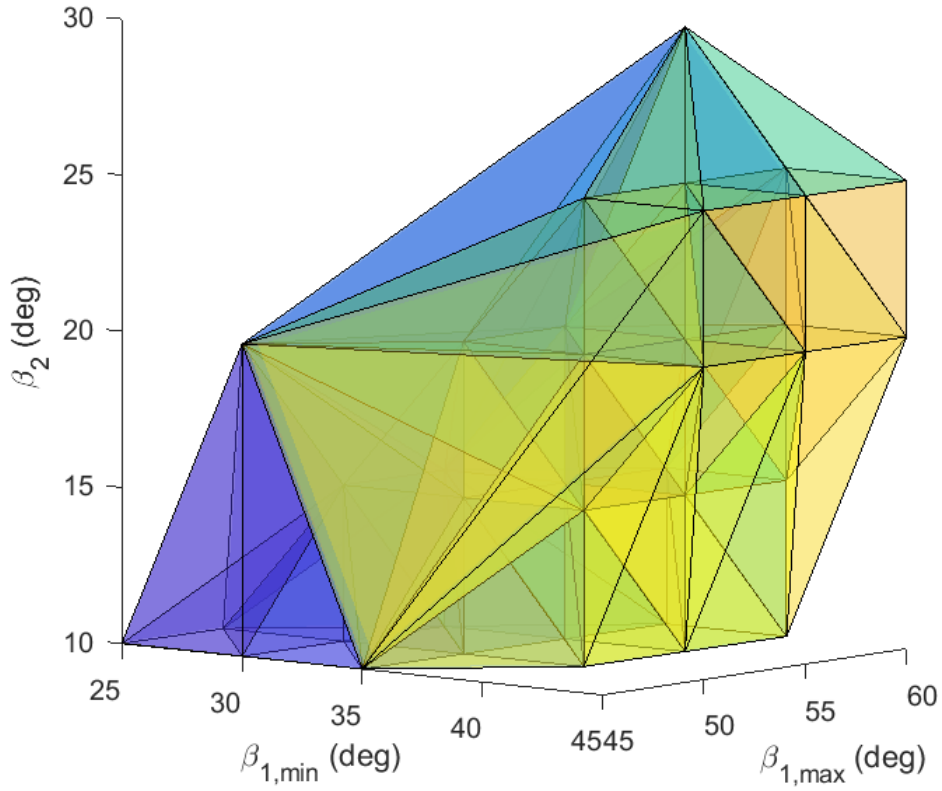


Figure 3.2. Delaunay triangulation of the database

The vertices of the tetrahedra shown in Figure 3.2 are points associated with the existing blade profiles in the database. Where known points are dense, the tetrahedra more closely approximate regular tetrahedra, which is desirable. However, on the edges of the known data, the tetrahedra created by the triangulation become elongated, which is indicative of

regions where the interpolation may be less predictive of the true values. For an arbitrary point of interest, the "griddata" function will interpolate from the vertices of the tetrahedron that contains the query point. Since this is a form of linear interpolation, the estimation of the parameter value will be closer to the "true" parameter value when there are many closely spaced known data points in the vicinity of the query point. Overall, the "griddata" function is called fourteen times, once for each of the blade parameters. The interpolated blade profile is then plotted, in the same manner as above, along with the profiles used for interpolation. An example of an interpolated profile was generated for the conditions $\beta_{1,min} = 42°$, $\beta_{1,max} = 53°$, and $\beta_2 = 23°$. To perform the interpolation, the selection tool used the following optimized profiles: 1) $\beta_{1,min} = 40°$, $\beta_{1,max} = 55°$, and $\beta_2 = 25°$, 2) $\beta_{1,min} = 45°$, $\beta_{1,max} = 50°$, and $\beta_2 = 20°$, 3) $\beta_{1,min} = 40°$, $\beta_{1,max} = 55°$, and $\beta_2 = 20°$, and 4) $\beta_{1,min} = 40°$, $\beta_{1,max} = 50°$, and $\beta_2 = 25°$. Figure 3.3 shows the enclosing tetrahedron along with the query point for the interpolated profile.

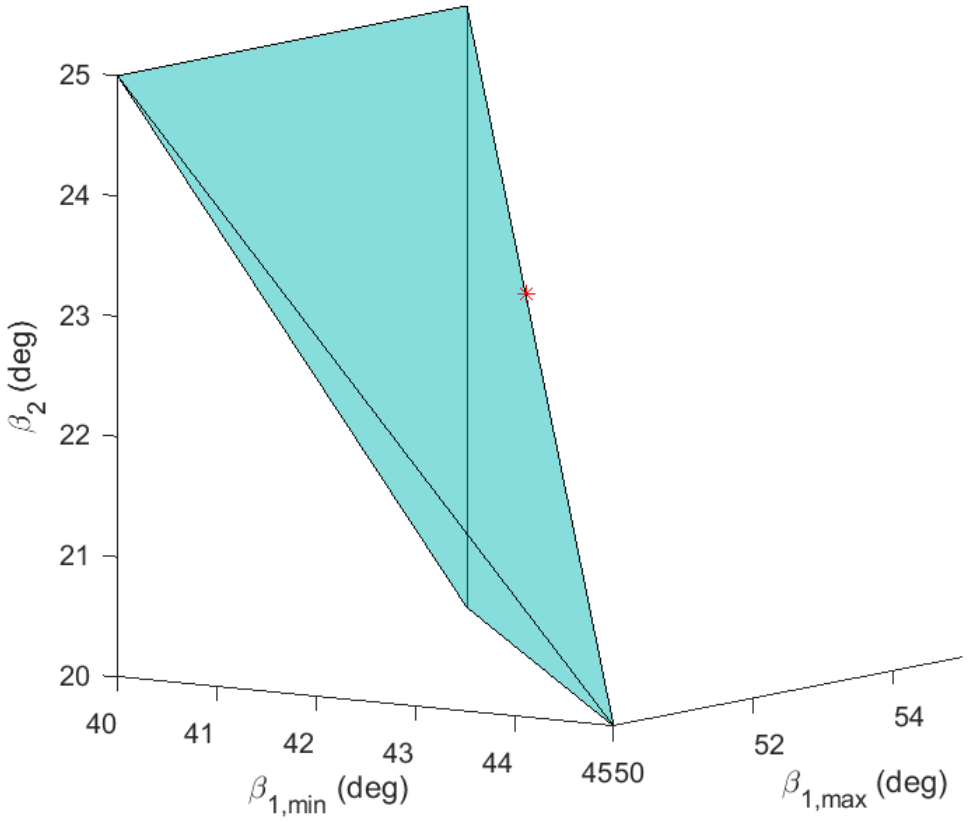


Figure 3.3. Enclosing Delaunay tetrahedron for $\beta_{1,min} = 42°$, $\beta_{1,max} = 53°$, and $\beta_2 = 23°$

The new profile, along with the optimized profiles from which it was interpolated, is plotted in Figure 3.4.

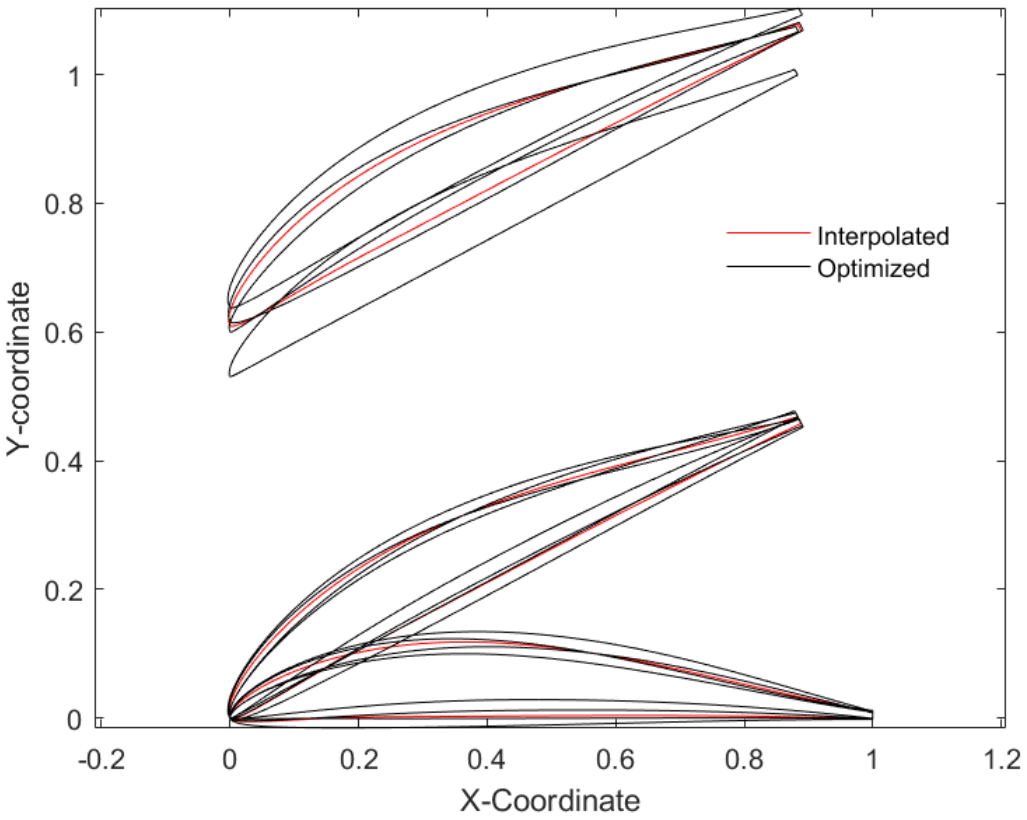


Figure 3.4. Selection tool interpolation example for $\beta_{1,min} = 42°$, $\beta_{1,max} = 53°$, and $\beta_2 = 23°$

In order to qualitatively determine how closely the interpolated profiles approximate the optimized profile at the given inlet/outlet conditions, a profile was both interpolated from the database and optimized for the conditions $\beta_{1,min} = 42°$, $\beta_{1,max} = 53°$, and $\beta_2 = 23°$. The interpolated and optimized profiles are shown in Figure 3.5.

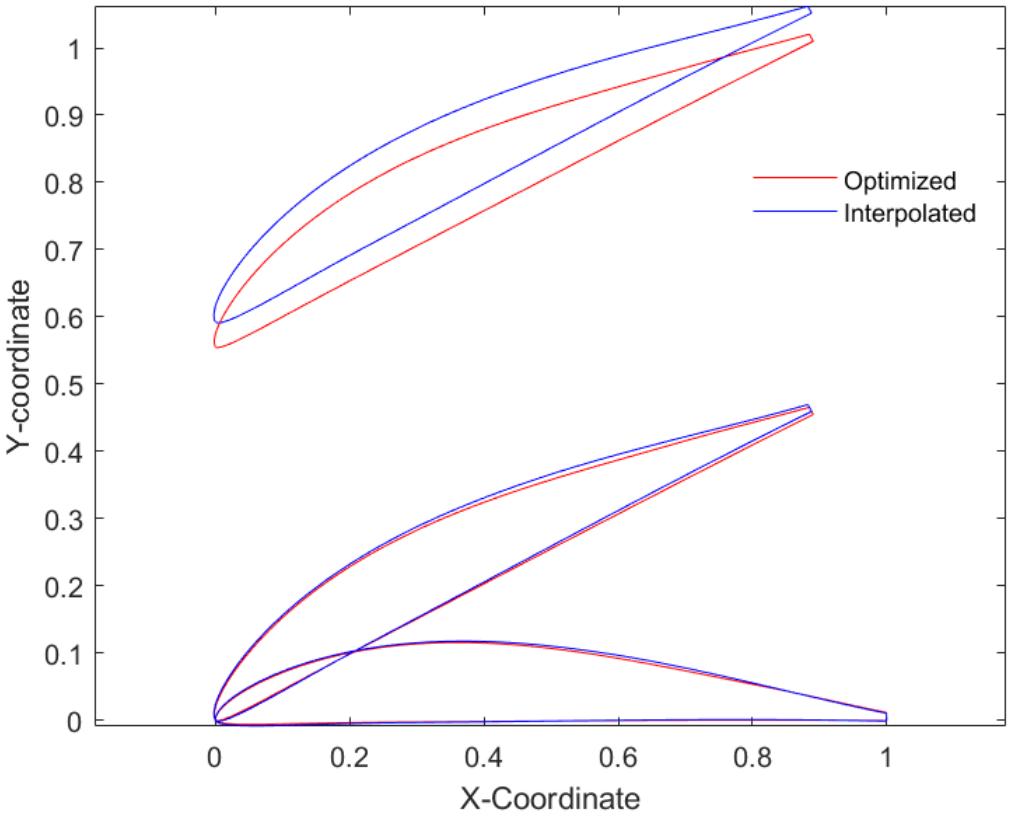


Figure 3.5. Selection tool interpolation comparison with optimized profile for $\beta_{1,min} = 42°$, $\beta_{1,max} = 53°$, and $\beta_2 = 23°$

Based on Figure 3.5, the shape of the interpolated blade closely matched the shape of the optimized blade. The only parameter where there was an appreciable difference was the blade pitch. This was because one of the profiles used for the interpolation was from a vertex far from the query point. However, the difference in pitch between the interpolated

and optimized profiles was only 7.14% of the blade chord, much smaller than the pitch difference between blade cross section at the hub and at the casing of a real compressor. Therefore, this difference was assumed to be negligible. In addition, as more profiles are added to the database, these discrepancies will become even smaller, making this assumption more valid.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Computational Fluid Dynamics Setup

The CFD program used in this thesis was ANSYS CFX. From this program, a more realistic estimation of the aerodynamic characteristics and performance of the blades could be determined by accounting for turbulent effects. The primary objective of the CFD analysis was to validate the optimized profiles by: 1) calculating the boundary layer profiles at various points along the blade, 2) verifying that the blades do not stall within the designed operational range, and 3) determining the total pressure loss coefficient across a single-stage cascade. For the CFD analysis, two optimized blade profiles were considered: the H-optimized and P-optimized profiles for an inlet angle range of 40° to 55° with a desired outlet angle of 20°. The geometry generation, meshing, and setup methods outlined in the following subsections remained consistent between two profiles. For this reason, where applicable, examples will only be shown for one profile. Examples for both profiles will be included where differences necessitate. A complete example solution report for the P-optimized blade at an inlet angle of 45° is provided in Appendix B.

4.1 Geometry

A SolidWorks part file for a unit-chord (1 [m]) blade was generated based on the optimized blade profiles. For each profile examined, the fluid domain extended one chord-length upstream and downstream from the leading and trailing edges of the blade, respectively. The appropriate upper and lower boundaries of the domain were chosen based on the pitch of the blade cascade. Since the flow analysis to be considered was two-dimensional, the width of the domain was irrelevant; however, the width was still chosen to be a small fraction (one-hundredth) of the chord length to approximate a two-dimensional domain. An example of a SolidWorks part generated for the H-optimized blade is shown in Figure 4.1, while the part generated for the P-optimized blade is shown in Figure 4.2.

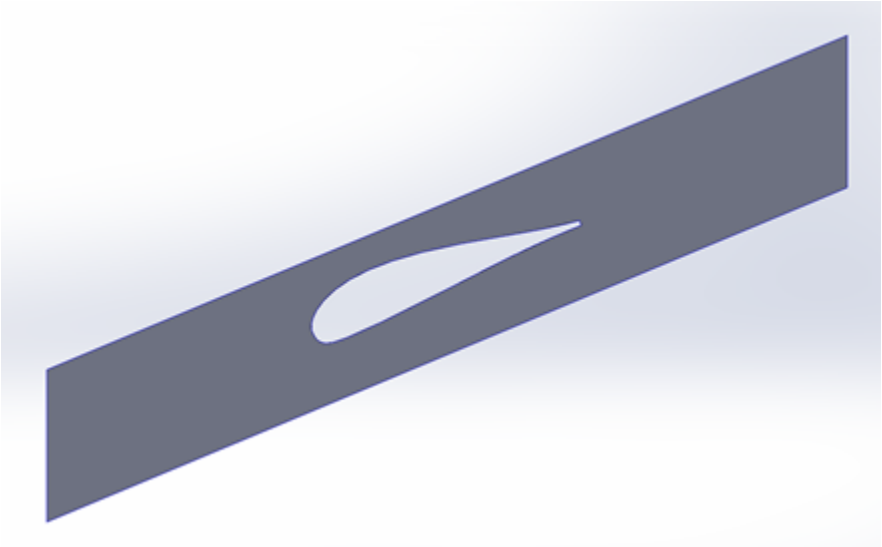


Figure 4.1. SolidWorks part generated for the H-optimized blade

Figure 4.2. SolidWorks part generated for the P-optimized blade

The SolidWorks part was then imported as a parasolid into DesignModeler, the parametric solid modeler native to ANSYS. Importing the SolidWorks part to DesignModeler ensured that the geometry used in the CFD analysis was identical to the profiles generated from the optimization code. The same *P*-optimized blade from Figure 4.2 is shown as a DesignModeler geometry in Figure 4.3.

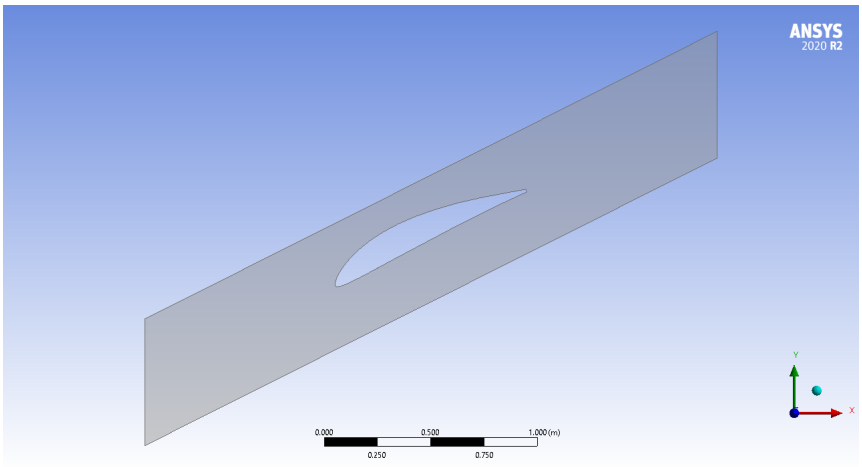


Figure 4.3. DesignModeler geometry for the *P*-optimized blade

4.2 Meshing

Prior to generating a mesh, each surface was named according to its function. Referencing the co-ordinate system in Figure 4.3: the negative-x face was named "Inlet," the positive-x face was named "Outlet," the negative-y face was named "BottomWall," the positive-y face was named "TopWall," the negative-z and positive-z faces were together named "SideWalls," and the blade surface was named "Blade." Naming each surface allowed for various meshing controls and setup constraints to be applied to the appropriate surface as necessary. Figure 4.4 shows the solid model with each of these named surfaces.

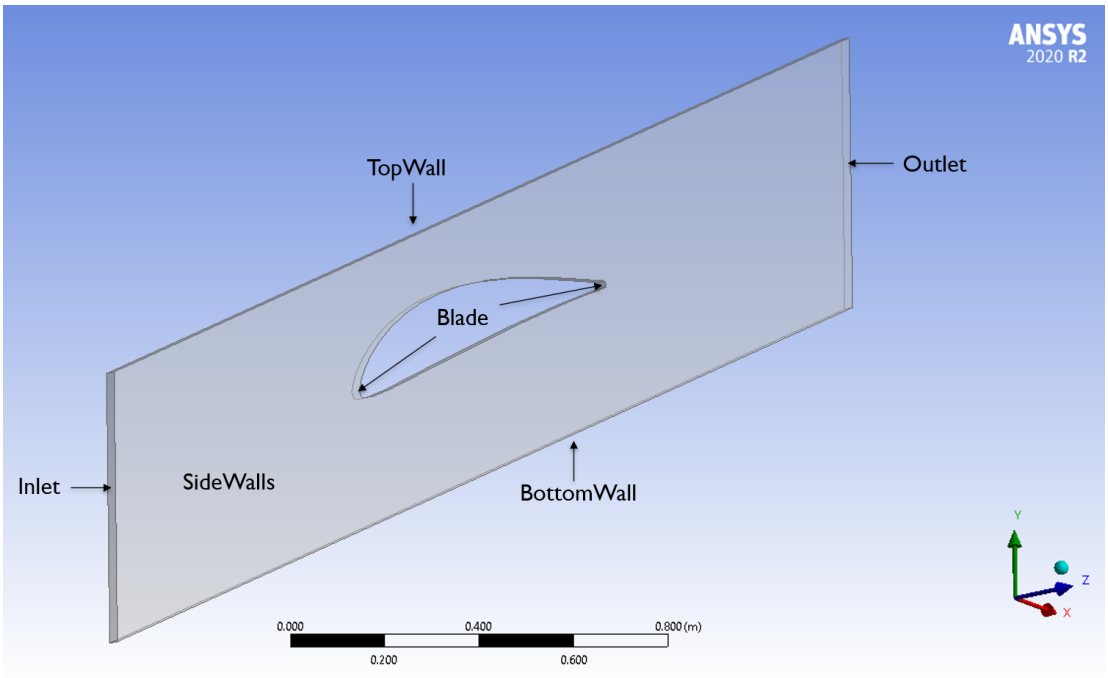


Figure 4.4. Named surfaces of the fluid domain

The first meshing control applied was a sweep method. Using a sweep method allowed for the number of mesh elements across the width of the fluid domain to be predetermined. By setting the number of mesh elements, or divisions, across the width of the domain equal to one, any mesh generated would only be one element-thick. A one element-thick mesh was desirable because it does not allow for any flow along the z-axis, which caused the fluid simulation to be two-dimensional. A close-up of the single-element-thick mesh for the *P*-optimized profile is shown below in Figure 4.5.

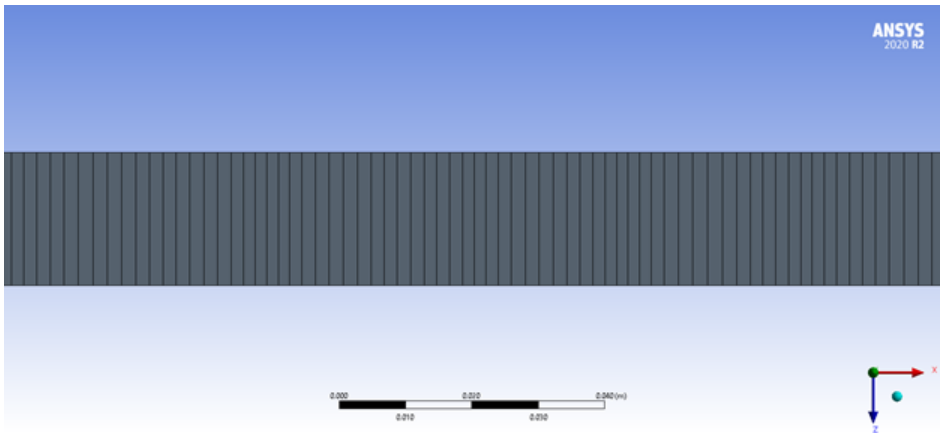


Figure 4.5. Single-element-thick mesh generated by the use of a sweep method

The second meshing control applied was match control. Match control forces the mesh elements on two surfaces to line up with each other; however, the surfaces must be identical to each other geometrically. Since the blade cascade had translational periodicity, the flow exiting the domain from the "TopWall" would re-enter the domain from the "BottomWall." Using match control on the "TopWall" and the "BottomWall" caused the elements on both surfaces to line up which, when combined with the translational periodicity of the blade cascade, minimized the potential for flow discontinuities across the boundary. The image in Figure 4.6 demonstrates match control in the *P*-optimized profile. The elements along the "TopWall" are shown on top and those of the "BottomWall" on bottom. The images were taken by only rotating the mesh view. Thus, each element matches identically to its counterpart on the opposite image.

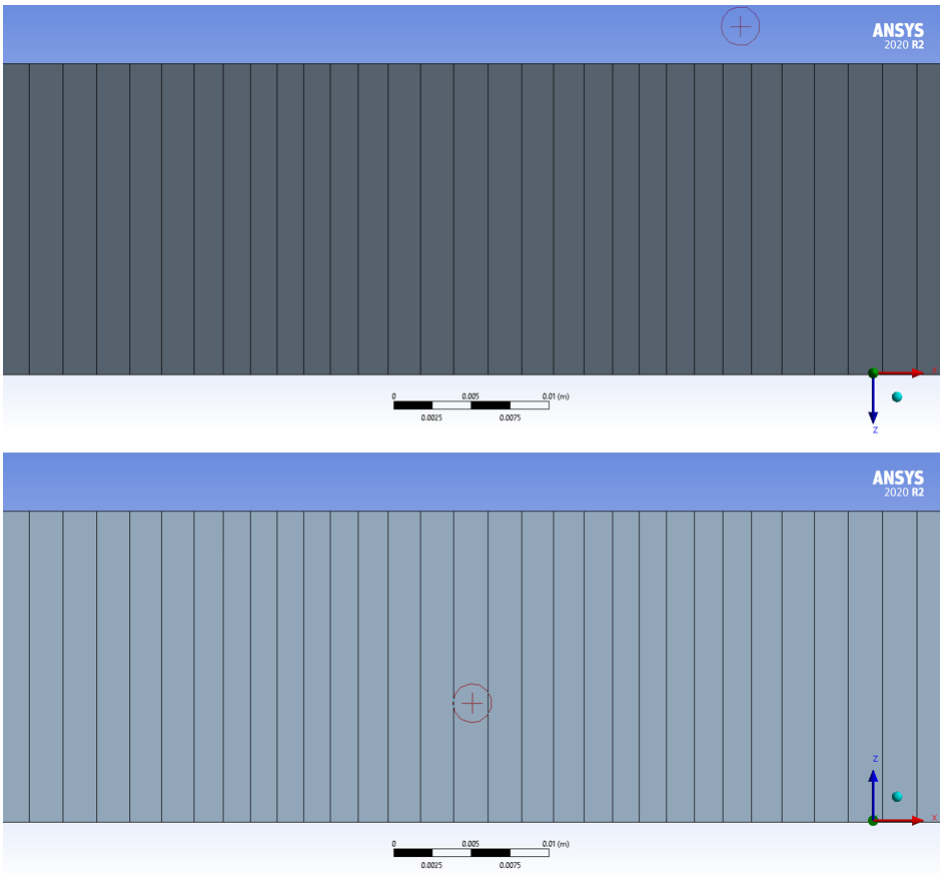


Figure 4.6. Result of match control on the upper and lower boundaries of the fluid domain. The TopWall is represented by the top image, while the BottomWall is represented by the bottom image.

The final meshing control applied was inflation. Mesh inflation reduces the size of elements near a designated surface. This allows for more precise resolution of the boundary layer. Thus, it was critical to apply an inflation layer to the blade surface. In order to accurately capture the boundary layer, including the laminar sublayer, the first-layer thickness, y, of the inflation layer must be small enough relative to the kinematic viscosity, ν, and frictional velocity, u_T. An appropriate selection for the first-layer thickness and growth rate can be determined by considering Equation 4.1 for the dimensionless wall distance, y^+.

$$y^+ = \frac{yu_T}{\nu} \tag{4.1}$$

For a growth rate close to one (1.1 - 1.2), the resolution of the boundary layer is generally deemed acceptable for values of y^+ below one. For each mesh, the first-layer thickness of the inflation layer was determined to be 10 [μm] with a growth rate of 1.15. The maximum number of layers was chosen to be 40. Figure 4.7 depicts the inflation layer along the surface of the P-optimized blade, while Figure 4.8 enumerates the inflation details.

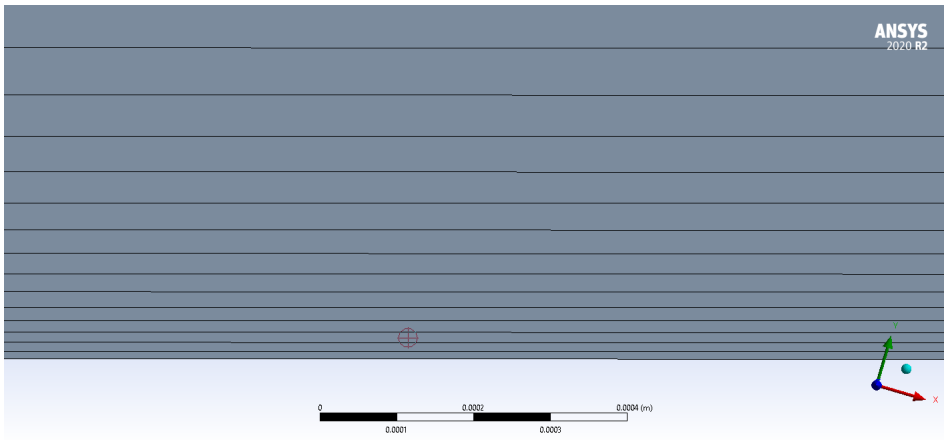


Figure 4.7. Inflation layer along the surface of the blade

Details of "Inflation" - Inflation

Scope	
Scoping Method	Geometry Selection
Geometry	1 Face
Definition	
Suppressed	No
Boundary Scoping Method	Geometry Selection
Boundary	2 Edges
Inflation Option	First Layer Thickness
First Layer Height	1.e-005 m
Maximum Layers	40
Growth Rate	1.15
Inflation Algorithm	Pre

Figure 4.8. Details of the inflation layer

Representative y^+ profiles along H-optimized and P-optimized blades are depicted in Figures 4.9-4.10. Each blade maintained a y^+ value below one along the entirety of the blade. Therefore, it can be assumed that size of the inflation layer is satisfactory to resolve the laminar sublayer of the boundary layer.

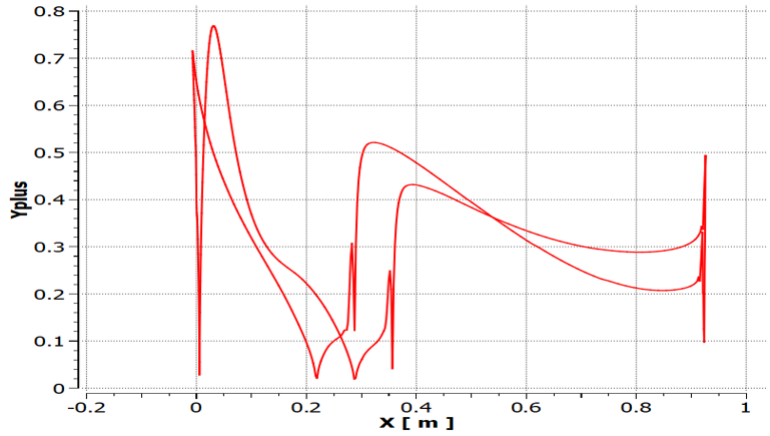


Figure 4.9. Representative y^+ profile for the H-optimized blade

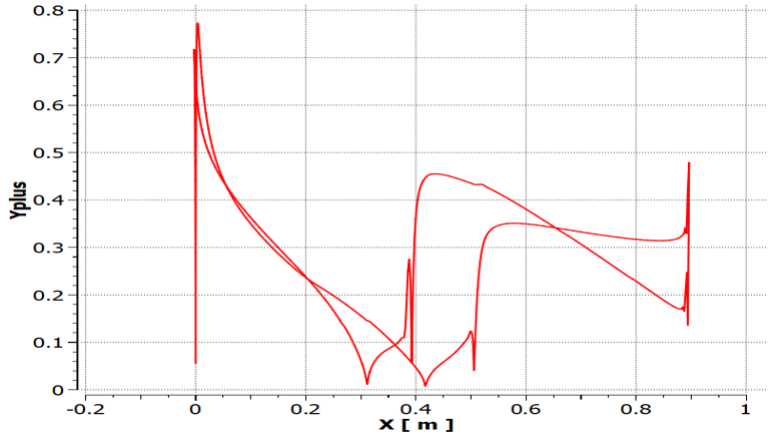


Figure 4.10. Representative y^+ profile for the P-optimized blade

The final meshes were a combination of all of the meshing controls above as well as the global maximum element sizing, which was determined to be 2.5 [mm]. The meshes used for the H-optimized and P-optimized profiles are shown below in Figures 4.11 and 4.12, respectively. The global mesh details, including the number of nodes and elements in each mesh, are provided in Figure 4.13.

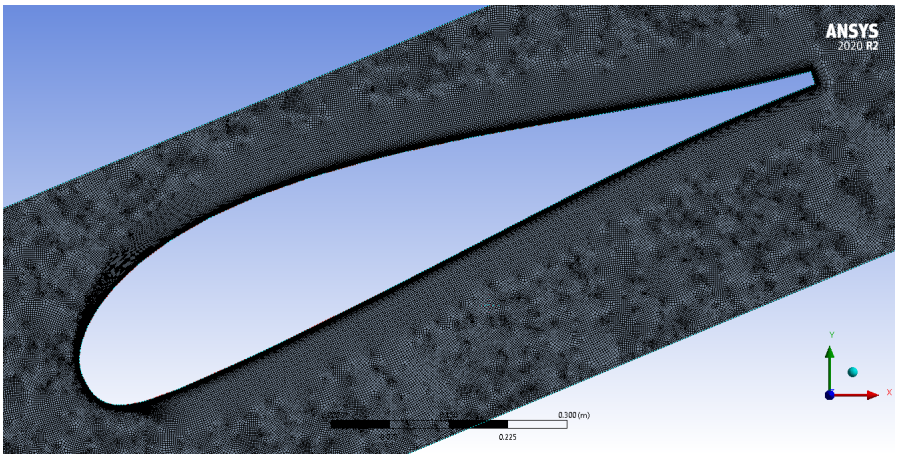


Figure 4.11. Overall mesh for the H-optimized blade profile

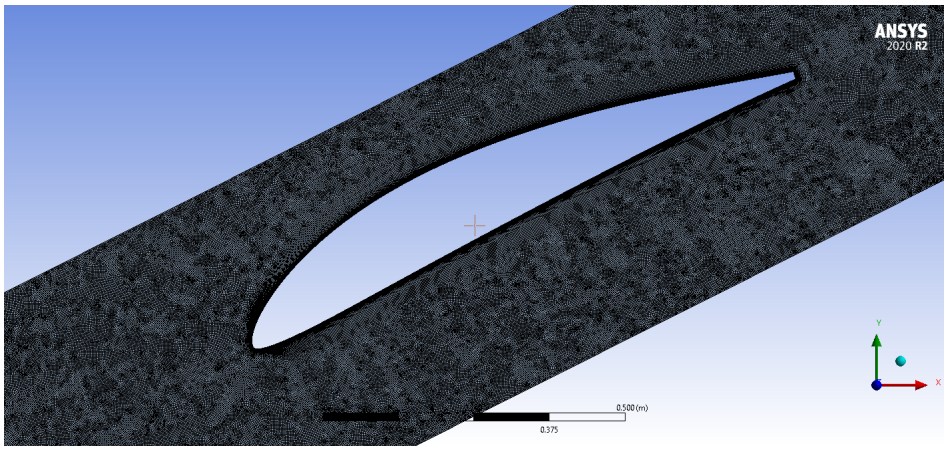
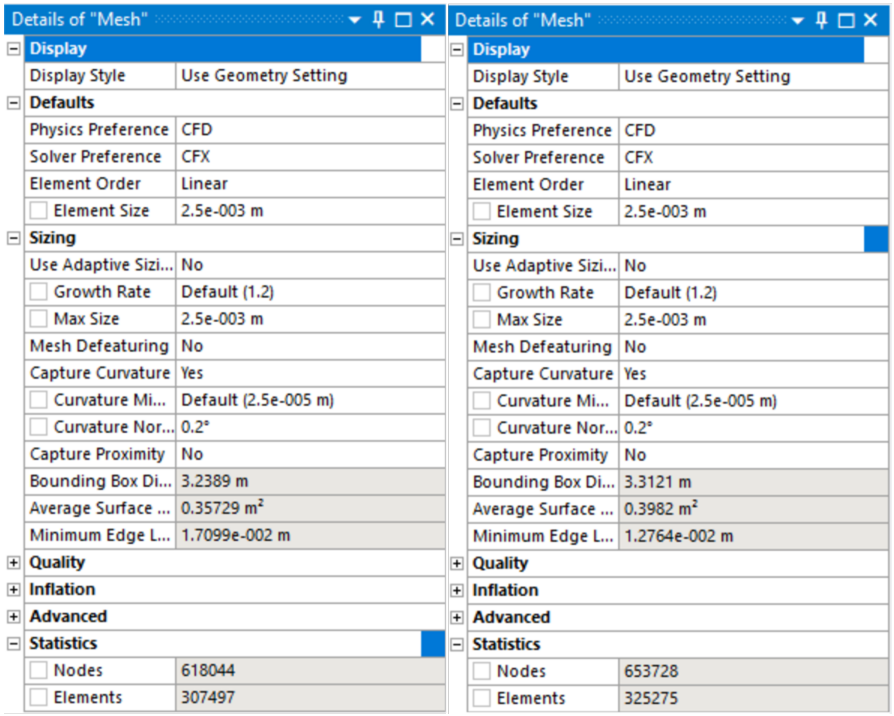


Figure 4.12. Overall mesh for the P-optimized blade profile



Details of "Mesh"	
Display	
Display Style	Use Geometry Setting
Defaults	
Physics Preference	CFD
Solver Preference	CFX
Element Order	Linear
Element Size	2.5e-003 m
Sizing	
Use Adaptive Sizi...	No
Growth Rate	Default (1.2)
Max Size	2.5e-003 m
Mesh Defeaturing	No
Capture Curvature	Yes
Curvature Mi...	Default (2.5e-005 m)
Curvature Nor...	0.2°
Capture Proximity	No
Bounding Box Di...	3.2389 m
Average Surface ...	0.35729 m²
Minimum Edge L...	1.7099e-002 m
Quality	
Inflation	
Advanced	
Statistics	
Nodes	618044
Elements	307497

Details of "Mesh"	
Display	
Display Style	Use Geometry Setting
Defaults	
Physics Preference	CFD
Solver Preference	CFX
Element Order	Linear
Element Size	2.5e-003 m
Sizing	
Use Adaptive Sizi...	No
Growth Rate	Default (1.2)
Max Size	2.5e-003 m
Mesh Defeaturing	No
Capture Curvature	Yes
Curvature Mi...	Default (2.5e-005 m)
Curvature Nor...	0.2°
Capture Proximity	No
Bounding Box Di...	3.3121 m
Average Surface ...	0.3982 m²
Minimum Edge L...	1.2764e-002 m
Quality	
Inflation	
Advanced	
Statistics	
Nodes	653728
Elements	325275

Figure 4.13. Mesh details for both the H-optimized (left) and P-optimized (right) profiles

Because the meshing controls are identical between the two profiles, the 5.77% difference in the number of nodes and elements in the two meshes can be explained by the size of the fluid domain. The domain of the H-optimized blade had a volume of 0.0267 [m^3], while the domain of the P-optimized blade had a volume of 0.0300 [m^3], a 12.36% increase in volume. However, this 12.36% volume increase overestimates the increase in the number of nodes and elements from mesh to mesh. The rest of the discrepancy can be accounted for by the fact that the inflation layer of the P-optimized blade mesh took up a smaller percentage of the total mesh than that of the H-optimized blade mesh. This decreased the mesh density of the P-optimized blade, resulting in only a 5.77% total increase in nodes and elements from the H-optimized blade mesh to the P-optimized blade mesh.

4.3 Setup and Solution

The CFD simulation setup was performed using CFX-Pre. All setup options were constant from simulation to simulation, except for the inlet angle of the incoming flow. The fluid domain was initialized as a continuous fluid using "Air at 25 C" from the Material Library with a reference pressure of 1 [atm]. The fluid model was a total energy, shear stress transport (SST), gamma-theta model including the viscous work term. The total energy model accounts for the transport of enthalpy and kinetic energy in the flow. The SST model is a turbulence model that incorporates the k-omega turbulence model in the boundary layer and the k-epsilon turbulence model in the freestream. This results in a more robust model, especially in the case of adverse pressure gradients. The gamma-theta model is a transitional turbulence model that helps to determine the effects of transitioning from a laminar boundary layer to a turbulent boundary layer.

The various named selections were initialized according to their functions. The "Inlet" was designated as a subsonic inlet with a total inlet velocity, V_T, of 15 [m/s]. This value of V_T, along with the density and dynamic viscosity of air at 298.15 [K], corresponds to a Reynolds number, Re, of 950,000 ($\approx 1,000,000$), which is roughly consistent with the Re of a typical compressor. The Cartesian component inlet velocities were calculated from V_T and β_1 through expressions with the appropriate trigonometric functions. The expressions for the velocity in the x-direction, u, and y-direction, v, are given below in Equations 4.2 and 4.3. Since the simulation was two-dimensional, the velocity in the z-direction was always 0 [m/s].

$$u = V_T \cos \beta_1 \tag{4.2}$$

$$v = V_T \sin \beta_1 \tag{4.3}$$

The inlet turbulence was set to Medium, corresponding to a turbulence intensity of 5%, and the static temperature of the inlet flow was set to 298.15 [K]. The "Outlet" (refer to Figure 4.4 was designated as a subsonic outlet constrained by an average relative pressure over the whole outlet of 0 [Pa]. The "Blade" was designated as a smooth, adiabatic, no-slip wall.

The no-slip condition causes the flow near the surface of the blade to approach 0 [m/s] as the distance from the blade decreases. In practice, this condition allows for the creation of the boundary layer flow. The "SideWalls" were designated as a symmetry, in effect creating slip walls on the sides of the domain. Since the domain is only a single element thick, there could be no flow across the side walls. Both the "TopWall" and "BottomWall" were designated as interfaces with conservative interface flux for mass, momentum, turbulence, and heat transfer. Setting these two boundaries as interfaces allowed them to interact with one another with translational periodicity (i.e. the flow exiting the top boundary would re-enter the domain through the bottom boundary). Employing translational periodicity in this manner simulates the flow field of a blade cascade instead of a single blade; however, the numerical calculations only had to be performed for a single blade domain, greatly reducing the required computational time and resources.

Solutions were run at double precision and initialized to previous solutions, if possible. Otherwise, solutions were initialized to the initial conditions. For each simulation, the convergence criteria was set to root-mean-squared residuals of 10^{-8}, although none of the simulations ultimately met this criteria. However, the simulations were still able to achieve steady-state residuals with reductions of three to four orders of magnitude. Representative mass and momentum residuals plots for the H-optimized and P-optimized blades are shown in Figures 4.14-4.15.

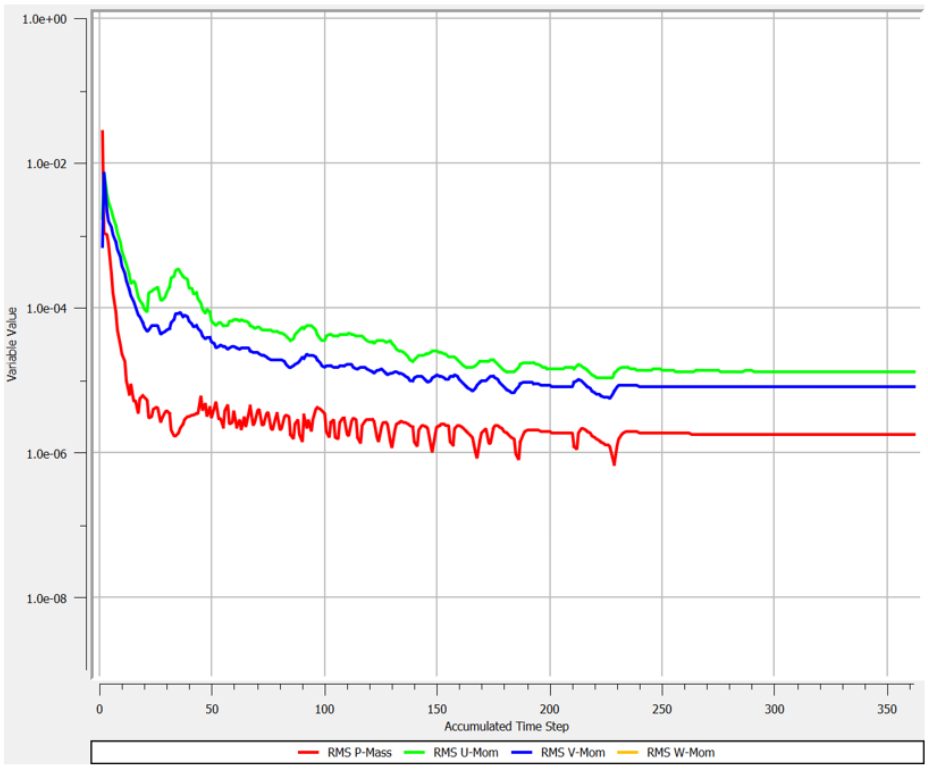


Figure 4.14. Representative mass and momentum residuals for the H-optimized blade

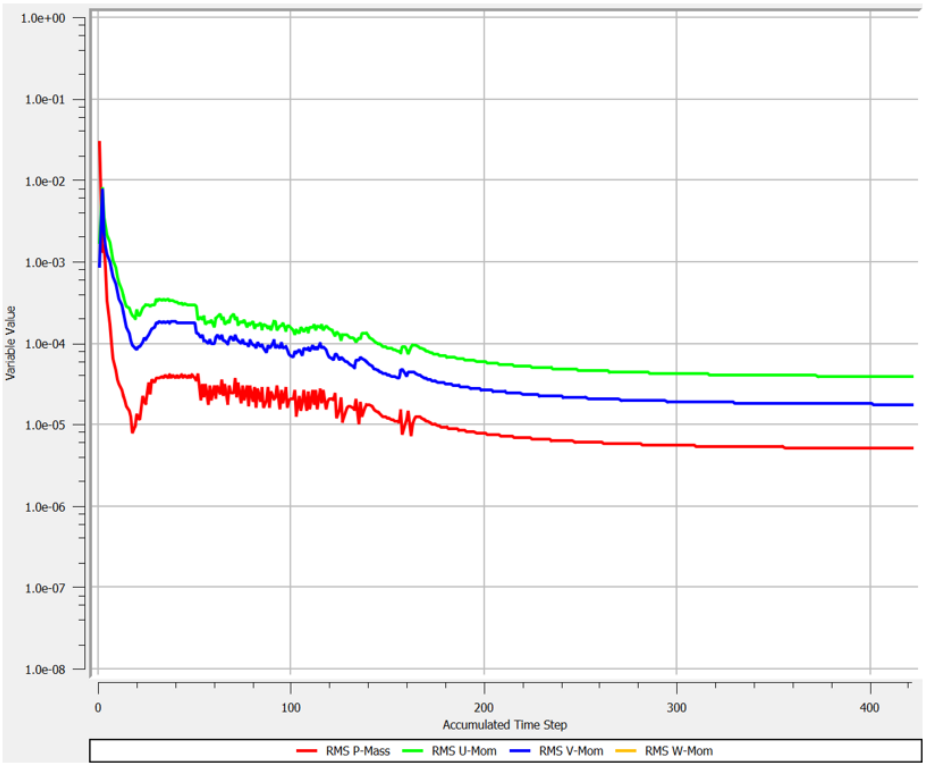


Figure 4.15. Representative mass and momentum residuals for the P-optimized blade

4.4 Post-Processing

There were a number of post-processing steps that were critical to examining the performance of the optimized blade profiles. First, the flow outlet angle was calculated using an expression in order to determine if the blades were actually turning the flow to the desired outlet angle. This expression is shown in Equation 4.4, where u_{out} and v_{out} are the mass-flow-averaged x- and y-velocities, respectively, at the outlet of the domain.

$$\beta_2 = \arctan\left(\frac{v_{out}}{u_{out}}\right) \tag{4.4}$$

The total pressure loss coefficient across the blade cascade was also calculated as a measure of the efficiency of the blade profile. The expression used for this calculation is shown in Equation 4.5, where $p_{t,in}$ and $p_{t,out}$ are the mass-flow-averaged total pressure at the inlet and outlet, respectively, ρ is the air density at the inlet, and V_{in} is the mass-flow-averaged velocity at the inlet [1].

$$\Delta C_{p_t} = \frac{p_{t,in} - p_{t,out}}{0.5\rho V_{in}^2} \tag{4.5}$$

In order to determine the boundary layer profiles and, thus, shape factor, at various points along the blade, a number of perpendicular lines had to be added to the domain as locations where the flow parallel to the blade could be calculated. The technique used to determine the locations of these lines was similar to that developed by Johnson [11], with a few key differences. Johnson found perpendicular lines by using SolidWorks to find the line perpendicular to a smooth curve of an airfoil at a given point. However, the blades used here are a collection of discrete points. Therefore, the perpendicular lines were determined as 90° rotations of the lines connecting consecutive points on the blade. For a point "A" and the consecutive point "B," the perpendicular line at point "A" is defined by two points: point "A" and the 90° rotation of point "B" about point "A." The x- and y-coordinates of this rotated point (point "A"') are given by Equations 4.6 and 4.7, where k is a scaling factor used to ensure that the entire boundary layer is captured by the perpendicular line.

$$x_{A'} = x_A - k\frac{y_B - y_A}{\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}} \tag{4.6}$$

$$y_{A'} = y_A + k\frac{x_B - x_A}{\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}} \tag{4.7}$$

Figure 4.16 shows all of the perpendicular lines calculated for the P-optimized blade using this method. The lines grow from leading to trailing edge as the boundary layer grows. The lines are more dense near the leading and trailing edges because the points defining the profile are more dense near the leading and trailing edges.

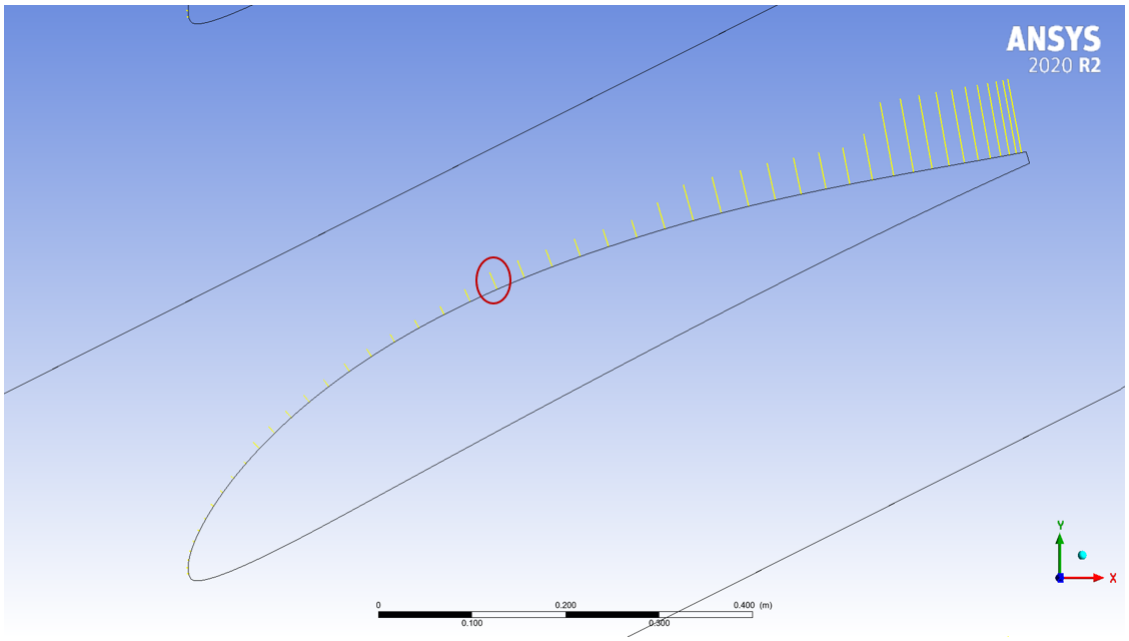


Figure 4.16. Perpendicular lines used to determine the boundary layer for the P-optimized blade

For each point, the component of velocity parallel to the surface of the blade, V_p, was calculated using Equations 4.8 and 4.9, where θ is the angle each blade segment makes with the horizontal axis.

$$\theta = \arctan\left(\frac{y_B - y_A}{x_B - x_A}\right) \tag{4.8}$$

$$V_p = u\cos\theta + v\sin\left(90^\circ - \theta\right) \tag{4.9}$$

Each perpendicular line and corresponding parallel velocity component were used to calculate the velocity at 1,000 points within the boundary layer. To calibrate the distance from the surface, δ, at every point along each line, the distance between each of the 1,000 points and point "A" had to be determined. This distance is given by Equation 4.10.

$$\delta = \sqrt{(x - x_A)^2 + (y - y_A)^2} \tag{4.10}$$

This data for δ and V_p for each line was then exported to a ".csv" file for the calculation of the shape factor. An example of the raw boundary layer data calculated through CFX-Post is shown in Figure 4.17. This particular boundary layer profile is for the perpendicular line circled in red in Figure 4.16.

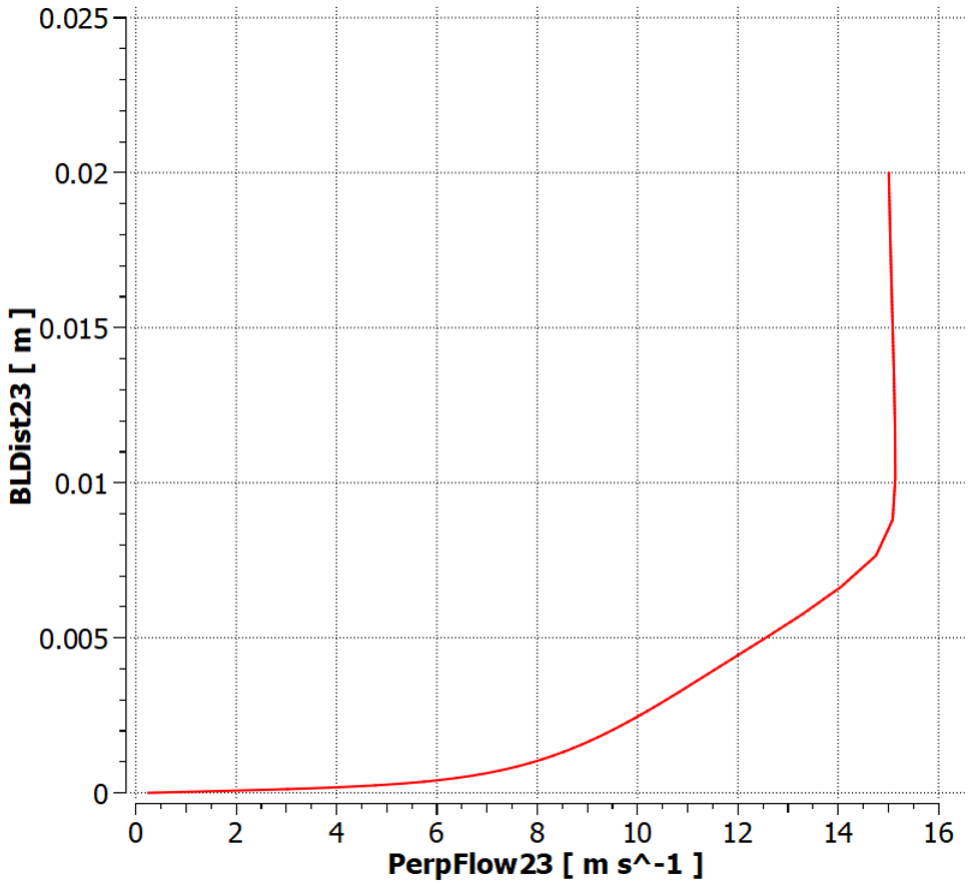


Figure 4.17. Example raw data boundary layer profile for the P-optimized blade

The data shown in Figure 4.17 exemplified the key attributes of a boundary layer profile. At the surface of the blade, the velocity of the flow parallel to the surface was zero, which is consistent with the no-slip condition. As the distance from the surface increased, the velocity parallel to the surface increased rapidly. The parallel velocity eventually reached a maximum value, $V_{p,max}$. The distance from the surface where the parallel velocity was 99% of $V_{p,max}$ was considered to be the boundary layer height, $\delta_{0.99}$. The corresponding velocity was denoted as $V_{0.99}$. With this information, the trapezoidal method of numerical integration was performed to approximate Equations 4.11 and 4.12 for the displacement thickness, δ^*, and momentum thickness, θ, respectively, as described in White [12].

$$\delta^* = \int_0^{\delta_{0.99}} \left(1 - \frac{V_p}{V_{0.99}}\right) \tag{4.11}$$

$$\theta = \int_0^{\delta_{0.99}} \frac{V_p}{V_{0.99}} \left(1 - \frac{V_p}{V_{0.99}}\right) \tag{4.12}$$

The shape factor was defined as the ratio of the displacement thickness to the momentum thickness, as shown in Equation 4.13.

$$H = \frac{\delta^*}{\theta} \tag{4.13}$$

Three types of boundary layer profiles were observed from the computational data: laminar (Figure 4.18), separated (Figure 4.19), and turbulent (Figure 4.20).

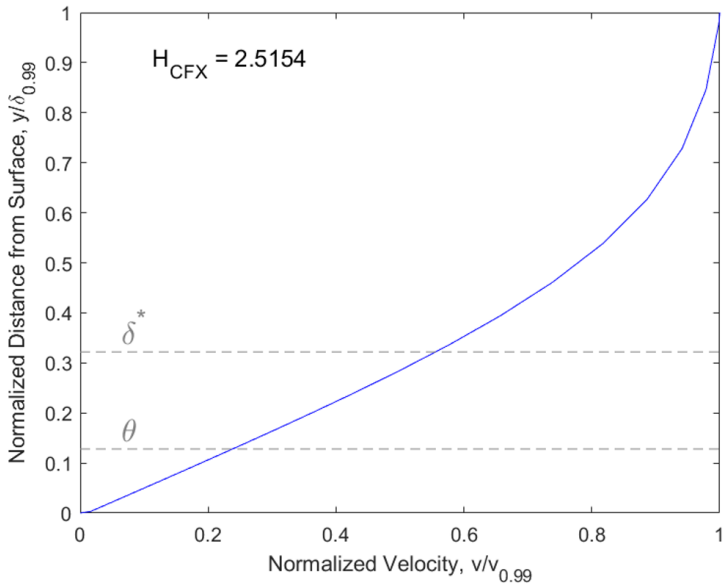


Figure 4.18. Example laminar boundary layer determined from fluid simulations

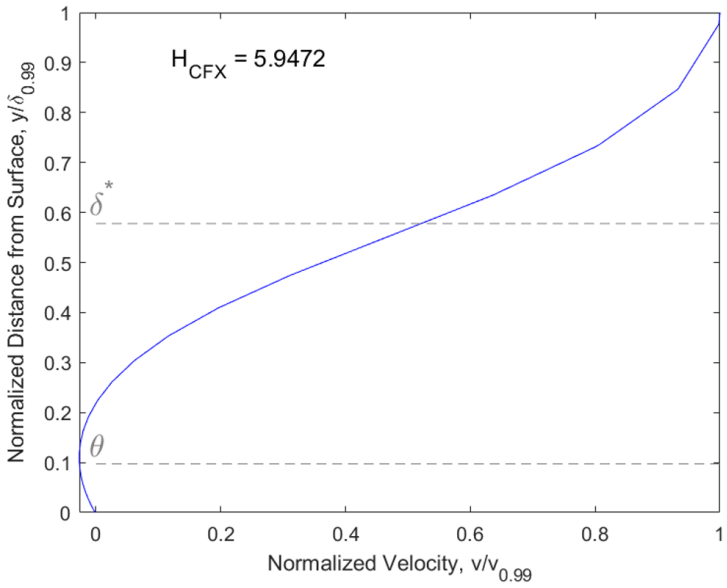


Figure 4.19. Example separated boundary layer determined from fluid simulations

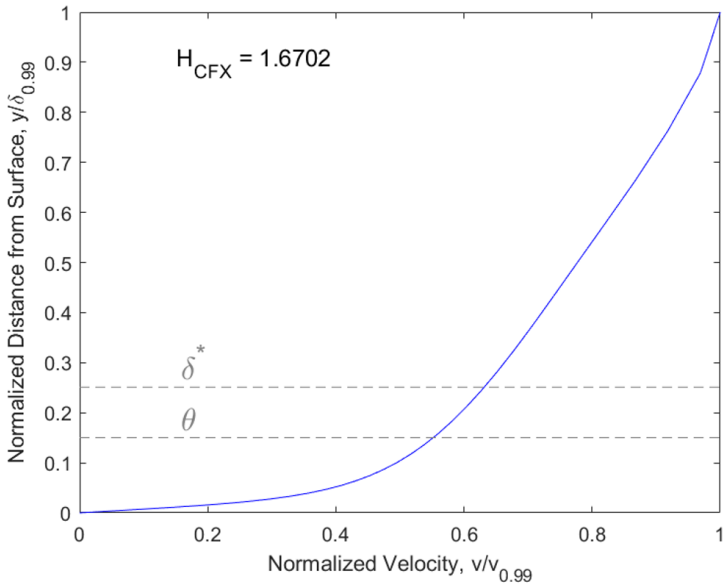


Figure 4.20. Example turbulent boundary layer determined from fluid simulations

Laminar boundary layer profiles, such as the example in Figure 4.18, demonstrated a constant velocity gradient throughout a significant portion of the boundary layer. In the upper portion of the boundary layer, the velocity gradient gradually decreased to a value of approximately zero at $\delta_{0.99}$. Laminar boundary layers corresponded to H values ranging from 2 to 3.5. Separated boundary layer profiles, such as the example in Figure 4.19, were characterized by a negative velocity gradient (reverse flow) at the surface of the blade and were associated with H values above 3.5. Turbulent boundary layer profiles, such as the example in Figure 4.20, consisted of a laminar sublayer and a turbulent sublayer. In the laminar sublayer, near the blade surface, the velocity increased rapidly at a constant rate. As the distance from the surface increased, the velocity gradient transitioned to a smaller value, which persisted for the majority of the boundary layer. Turbulent boundary layers close to the laminar-turbulent boundary layer transition were characterized by H values below 2, whereas fully turbulent boundary layers were characterized by H values ranging from 2 to 3.5. Even though the H values for laminar and turbulent boundary layers overlapped, the type of boundary layer was still distinguishable by its position relative to the laminar-turbulent transition. Provided the boundary layers were attached, boundary layers prior to the transition had to be laminar, while boundary layers after the transition had to be turbulent.

CHAPTER 5:
Profile Validation

5.1 Optimized Profile Performance

The results of the CFD analysis on an H-optimized blade profile and a P-optimized blade profile were gathered in order to compare the relative performance of each cost function in three areas: minimal flow separation on the suction surface, meeting the outlet angle requirement at all inlet angles, total pressure loss across the cascade, and shape factor along the blade. The cost function that produced the profiles with better performance in these metrics was the cost function used to populate blade profiles in the database. The inlet/outlet conditions for the test case blades were an inlet range from 40° to 55° with an outlet angle of 20°. The H-optimized and P-optimized profiles for these conditions are shown below in Figure 5.1.

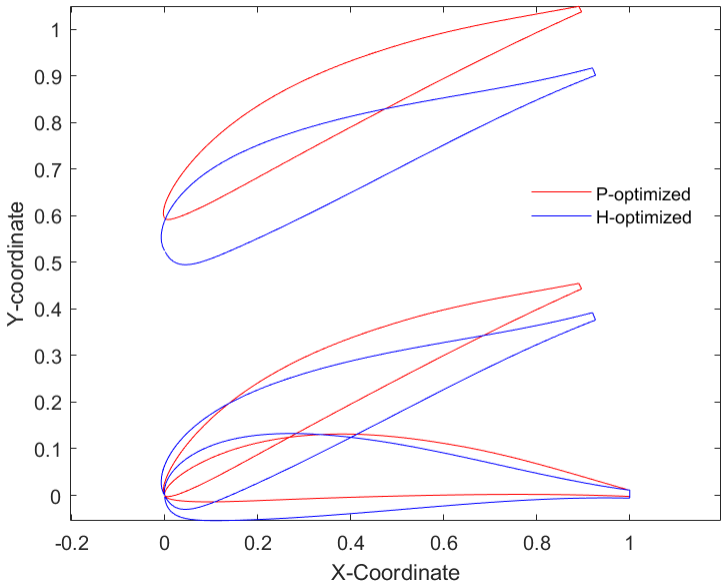


Figure 5.1. Profile comparison between the H-optimized and P-optimized blades for $\beta_{1,min} = 40°$, $\beta_{1,max} = 55°$, and $\beta_2 = 20°$

The two cost functions produced vastly differing blades for these inlet/outlet conditions. The H-optimization resulted in a blade with a bulbous leading edge, a cambered pressure surface, a low stagger angle, and a small pitch. The P-optimization, however, resulted in a blade with a sharp leading edge, a flat pressure surface, a higher stagger angle, and a larger pitch. The performance differences between these two blades are the result of their geometric differences.

A qualitative analysis of flow separation was performed by inspection of the velocity contours of the two profiles. Flow separation was identified as regions of stagnant flow (colored blue) near the blade surface. The velocity contours for the H-optimized blade at each inlet angle tested are shown in Figures 5.2-5.5.

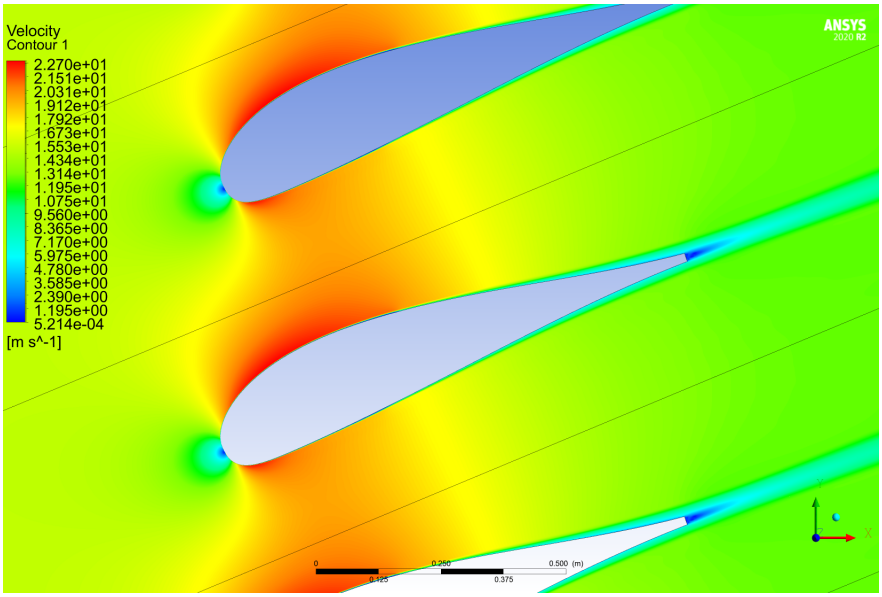


Figure 5.2. Velocity contour of the H-optimized blade profile for $\beta_1 = 40°$ and $\beta_2 = 20°$

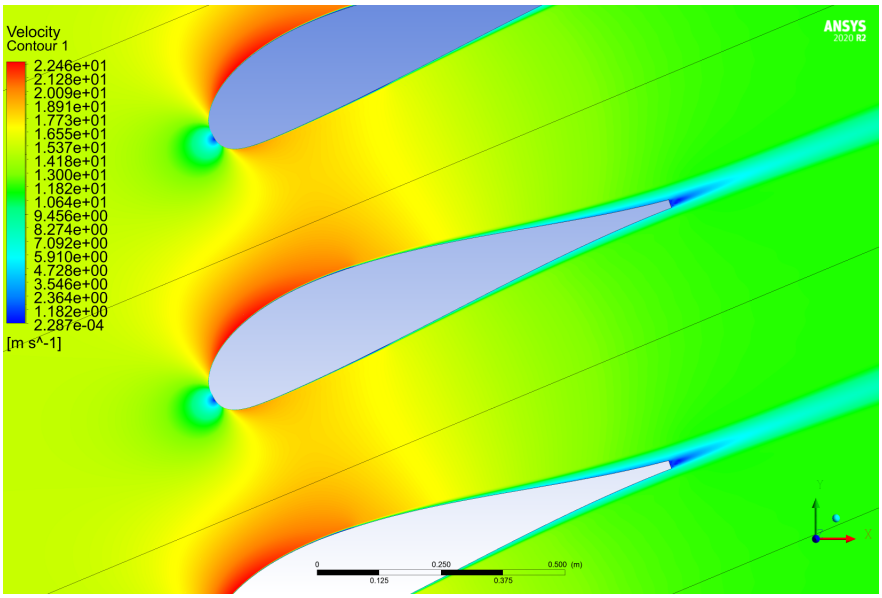


Figure 5.3. Velocity contour of the H-optimized blade profile for $\beta_1 = 45°$ and $\beta_2 = 20°$

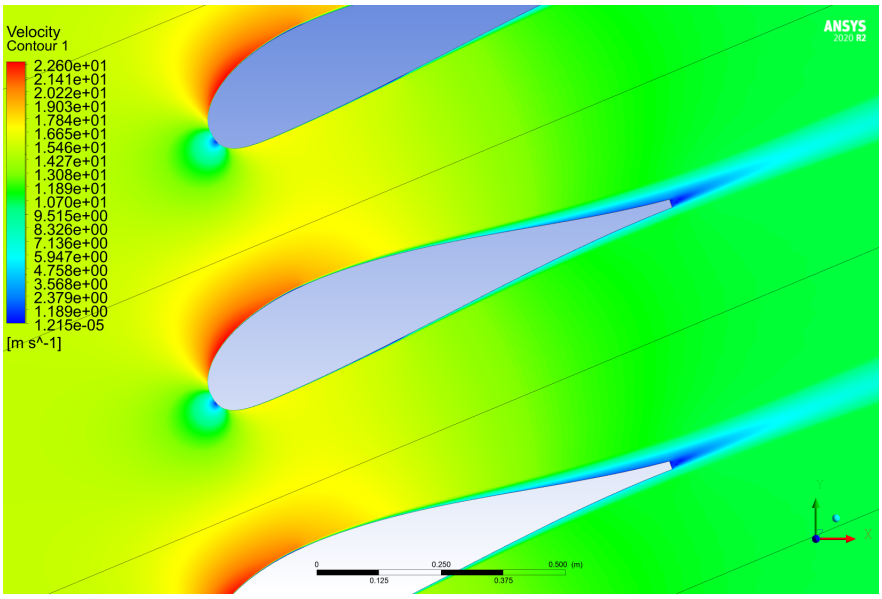


Figure 5.4. Velocity contour of the H-optimized blade profile for $\beta_1 = 50°$ and $\beta_2 = 20°$

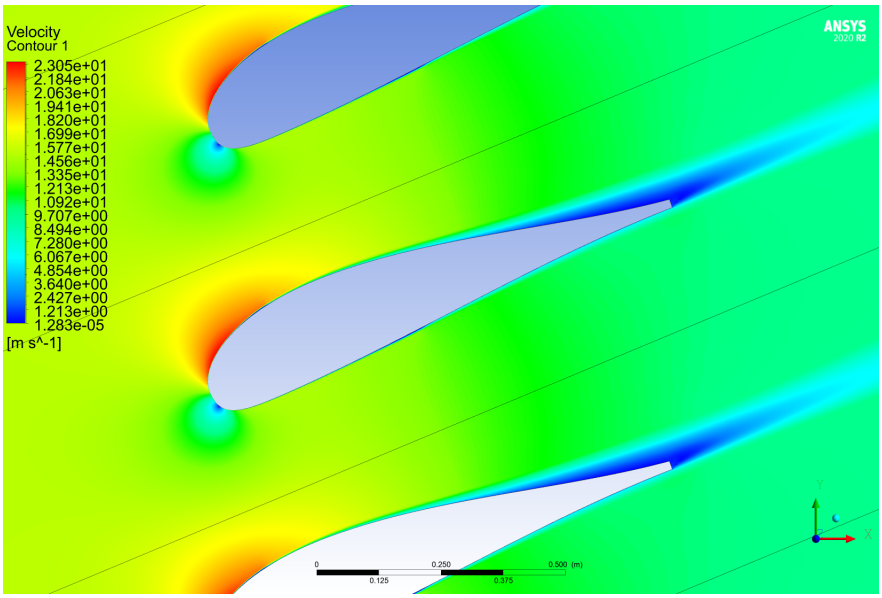


Figure 5.5. Velocity contour of the H-optimized blade profile for $\beta_1 = 55°$ and $\beta_2 = 20°$

Flow separation can be seen to varying degrees at each inlet angle. Ignoring the stagnation point at the leading edge, small blue regions of flow separation are present along the top surface (separation bubble) and downstream of the blunt trailing edge in each contour. The amount of flow separation increased with the increasing inlet angle, culminating in substantial separation in the $\beta_1 = 55°$ case. In addition, the onset of flow separation began to occur further upstream as the inlet angle increased. Similar trends can be seen in the velocity contours of the P-optimized blades, shown in Figures 5.6-5.9.

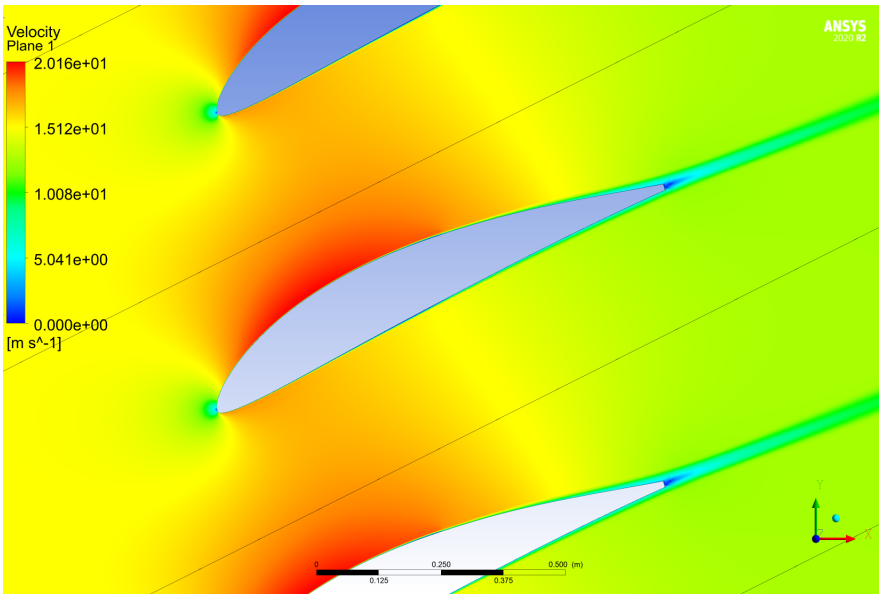


Figure 5.6. Velocity contour of the P-optimized blade profile for $\beta_1 = 40°$ and $\beta_2 = 20°$

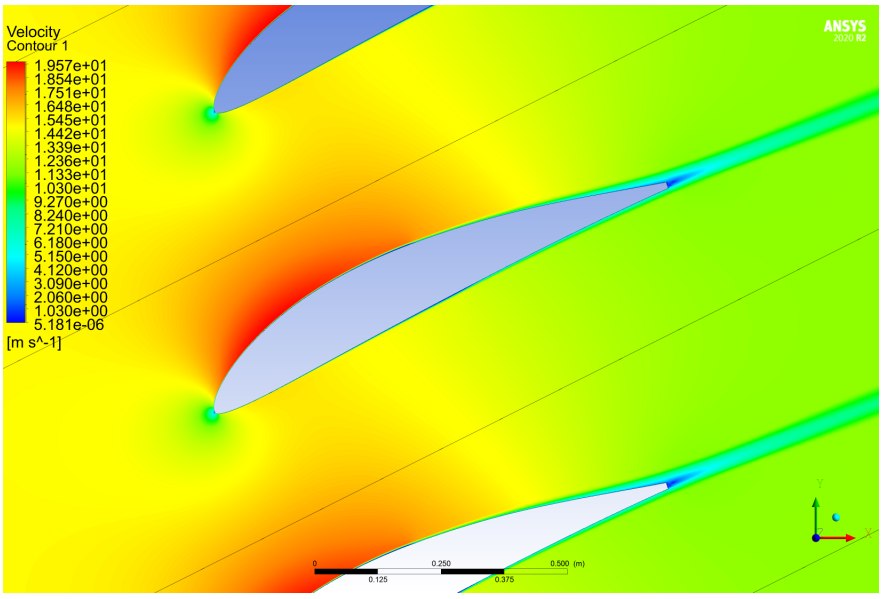


Figure 5.7. Velocity contour of the P-optimized blade profile for $\beta_1 = 45°$ and $\beta_2 = 20°$

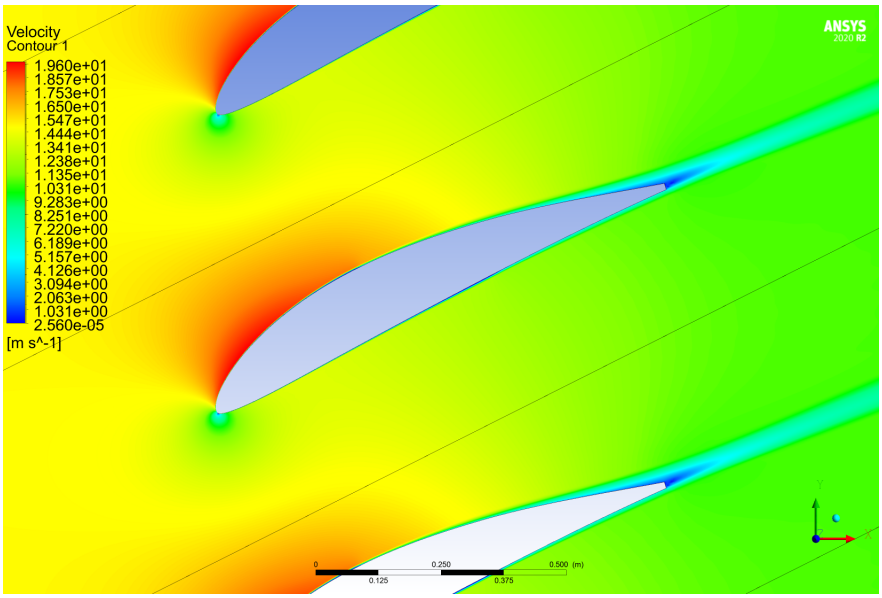


Figure 5.8. Velocity contour of the P-optimized blade profile for $\beta_1 = 50°$ and $\beta_2 = 20°$

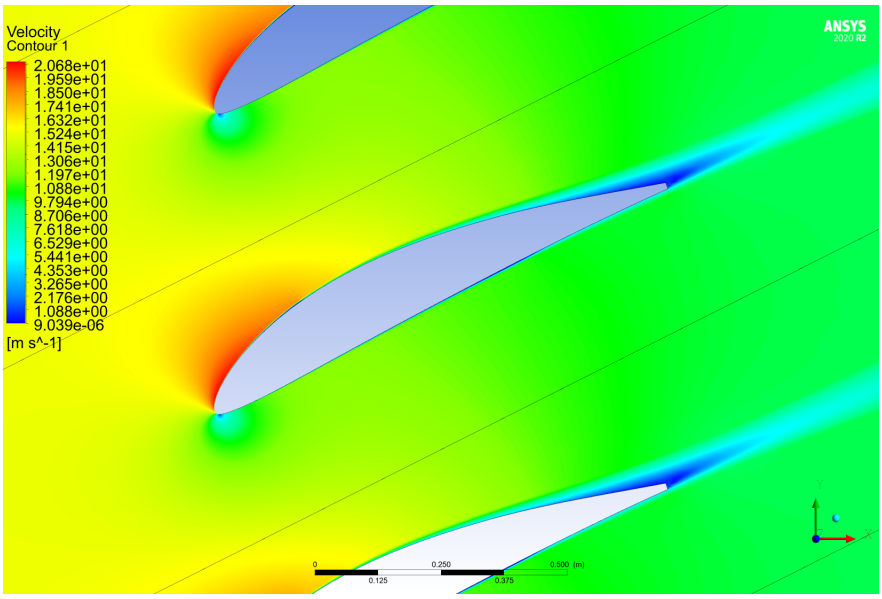


Figure 5.9. Velocity contour of the P-optimized blade profile for $\beta_1 = 55°$ and $\beta_2 = 20°$

Although the amount and onset of flow separation follows a similar trend for the P-optimized blade, the absolute amount of separation shows a stark contrast between the blades. For each inlet angle, the amount of flow separation is much smaller for the P-optimized blade than for the H-optimized blade. Substantial flow separation is not present for the P-optimized blade until the maximum inlet angle of 55°, whereas it was present for the H-optimized blade by an inlet angle of 50°. Substantial flow separation prevents the affected blade surface from producing lift and, therefore, turning the flow any further. Thus, the effects of flow separation manifest in the outlet flow angle achieved by the blade. The outlet flow angles achieved by both the H-optimized and P-optimized blades are depicted in Figure 5.10 along with the desired outlet angle of 20°.

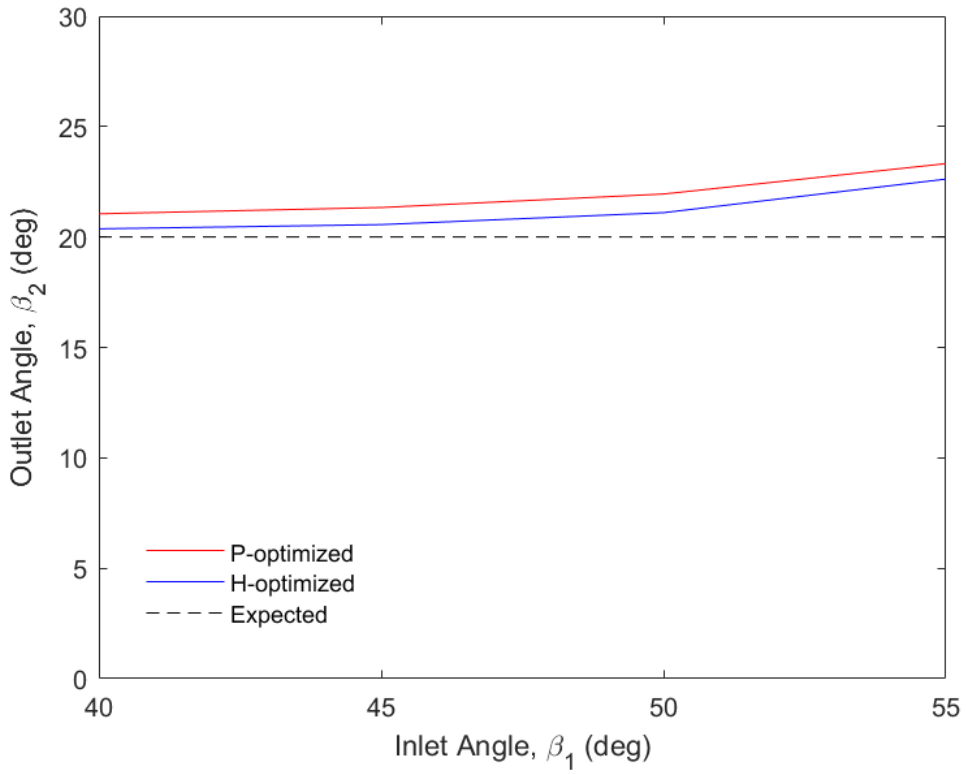


Figure 5.10. Outlet angles achieved by the H- and P-optimized blade profiles

Neither the H-optimized blade nor the P-optimized blade fully achieved the amount of flow turning necessary to reach the desired outlet angle at any of the inlet angles. However,

the amount of flow turning that was achieved was acceptable for both profiles; though, the H-optimized blade produced more flow turning than the P-optimized blade at every inlet angle. At the lowest inlet angle, the percent difference between the achieved and desired outlet angles was only 1.85% for the H-optimized profile and 5.26% for the P-optimized blade. These values increased to a maximum of 13.11% and 16.61%, respectively, at the maximum inlet angle of 55°. The increase in the achieved outlet angle as the inlet angle increased can be explained by the increasing amount of flow separation, as described above. Overall, both blades were successful in turning the flow to the desired outlet angle.

Flow separation also manifested in total pressure loss across the cascade. The total pressure loss curves for the H-optimized and P-optimized blades are plotted in Figure 5.11.

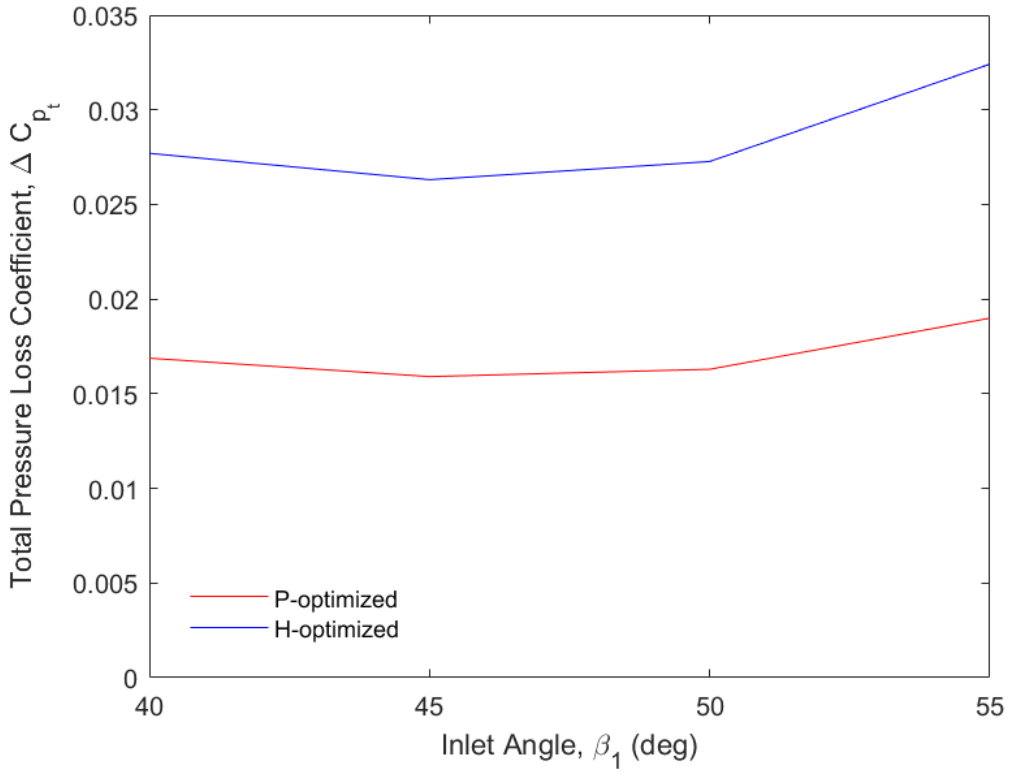


Figure 5.11. Total pressure loss curves for the H- and P-optimized blade profiles

Loss of total pressure was indicative of energy dissipation, or frictional losses, in the flow. Ideally, total pressure loss should be minimal to maximize the work potential of the flow. As shown in Figure 5.11, the total pressure loss coefficient of both blades reached a minimum near the center of the inlet angle range. The inlet angle at which the coefficient was minimized represents the most efficient operating inlet angle for that blade. With only four data points for each blade, however, the exact inlet angle where the coefficient was minimized cannot be resolved. From Figure 5.11, it is also known that the P-optimized blade recovered more total pressure than the H-optimized blade at every inlet angle. Across the entire inlet angle range, the total pressure loss coefficient of the P-optimized blade was roughly 60% of the coefficient for the H-optimized blade (at the same inlet angle), suggesting that the P-optimized blade was far more efficient than the H-optimized blade.

The boundary layer characterizations obtained from the fluid simulations in Section 4.4 resulted in shape factor profiles at each inlet angle for the H-optimized and P-optimized blades. Figure 5.12 plots the shape factor profiles as a function of the distance traveled along each blade, s, where $s = 0$ was defined to be the stagnation point of the flow field at the leading edge.

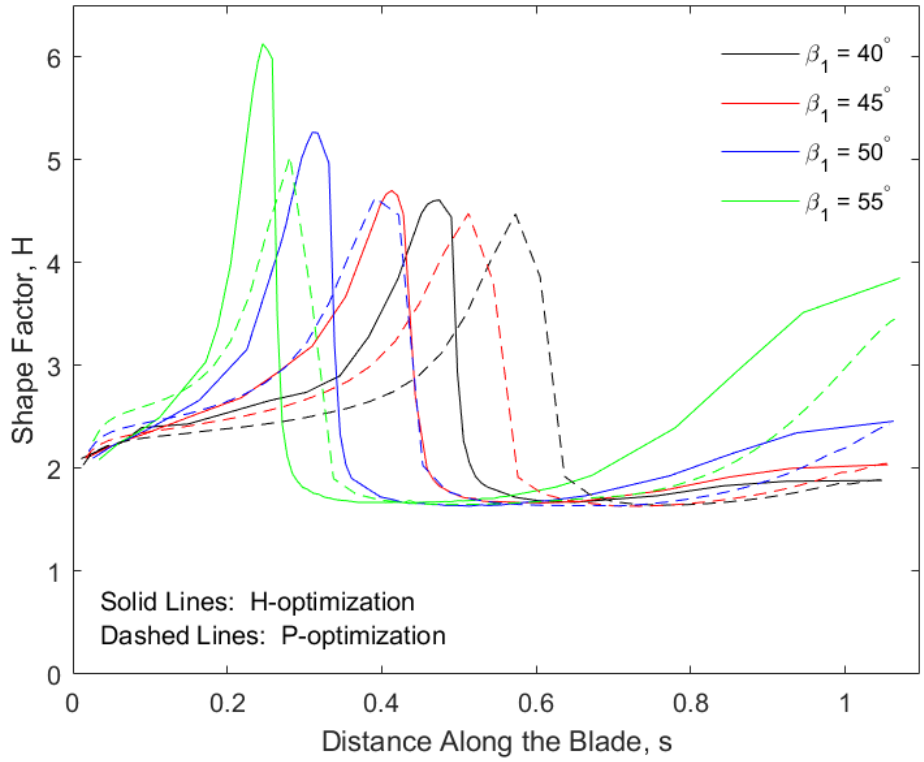


Figure 5.12. Shape factor profiles for the H-optimized and P-optimized blades: $\beta_2 = 20°$

Every shape factor profile in Figure 5.12 followed a similar pattern. At the leading edge, the shape factor was approximately equal to 2 (laminar). Travelling along the blade, the shape factor gradually increased, remaining in the region associated laminar boundary layers, until the boundary layer detached at $H = 3.5$, forming the start of a separation bubble. The separation bubble continued through the region where $H \geq 3.5$. The sharp decrease in shape factor, seen in each profile, was where the separation bubble ended and the boundary layer reattached to the surface of the blade. From that point on, the boundary layers were turbulent, with lower shape factors near the point of boundary layer reattachment.

The effects of increasing the inlet flow angle were present in both the H-optimized and P-optimized blade. As the inlet flow angle increased, the location of boundary layer detachment travelled further upstream, causing the separation bubble to form nearer to the leading

edge of the blade. In addition, the maximum shape factor increased with increasing inlet flow angles, indicating that the boundary layer separation was more severe. Although the boundary layer reattached to the surface for all inlet angles within the optimization range ($\beta_1 = 40° - 55°$), the trend of increasing maximum shape factor suggests that for inlet angles above 55°, the boundary layer may not reattach. Such a scenario would cause the blade to stall and be unable to turn the flow any further.

Given the same inlet flow angle, the separation bubble formed further downstream, with a lower maximum shape factor, for the *P*-optimized blade compared to the *H*-optimized blade. The combination of these factors suggested that the flow permanently detaches from the *P*-optimized blade at a higher inlet angle than for the *H*-optimized blade. Thus, the *P*-optimized blade likely had a greater stall margin than the *H*-optimized blade. Although both blades functioned throughout the entire optimization range, the larger stall margin of the *P*-optimized blade provides the potential for the blade to operate outside of the optimization range while maintaining functionality.

Overall, the *P*-optimized blade outperformed the *H*-optimized blade. Since both blades were able to achieve the flow turning requirement, the superior performance of the *P*-optimized blade in terms of total pressure loss and stall margin was enough to determine that the *P*-optimization cost function designed a blade more suitable to application in a compressor. Therefore, the *P*-optimized blades were used to populate the database described in Section 3.1.

5.2 Multi-Stage Compressor Application

As a culmination of the optimization process, the database of *P*-optimized blade profiles and the selection tool were used to design a simplified multi-stage compressor. Raw data was provided for the required inlet/outlet conditions along five streamlines (from the hub to the casing) for a three-stage axial compressor, totaling thirty unique sets of inlet/outlet conditions. Interpolating the required blade profiles allowed for the compressor to be designed as quickly as possible, with no need to individually optimize each profile. Table 5.1 lists the inlet/outlet conditions required for each streamline at each of the rotors and stators in the compressor.

Table 5.1. Required inlet/outlet conditions for simplified three-stage compressor

	Rotor 1			Stator 1		
	$\beta_{1,min}$	$\beta_{1,max}$	β_2	$\beta_{1,min}$	$\beta_{1,max}$	β_2
(Hub) 1	39.11	53.72	15.49	27.13	49.62	10.00
2	39.61	54.69	19.96	26.72	48.61	10.00
3	40.10	55.58	21.51	26.32	47.66	10.00
4	40.58	56.41	22.66	25.94	46.77	10.00
(Casing) 5	41.04	57.18	23.75	25.57	45.93	10.00

	Rotor 2			Stator 2		
	$\beta_{1,min}$	$\beta_{1,max}$	β_2	$\beta_{1,min}$	$\beta_{1,max}$	β_2
(Hub) 1	34.27	51.93	10.00	34.15	51.66	10.00
2	34.88	53.05	11.31	33.82	50.88	10.00
3	35.46	54.08	12.58	33.51	50.16	10.00
4	36.03	55.03	13.80	33.21	49.47	10.00
(Casing) 5	36.59	55.91	14.97	32.92	48.82	10.00

	Rotor 3			Stator 3		
	$\beta_{1,min}$	$\beta_{1,max}$	β_2	$\beta_{1,min}$	$\beta_{1,max}$	β_2
(Hub) 1	34.27	51.93	10.00	34.15	51.68	10.00
2	34.84	52.96	11.23	33.85	50.97	10.00
3	35.39	53.91	12.43	33.55	50.30	10.00
4	35.93	54.79	13.58	33.27	49.66	10.00
(Casing) 5	36.45	55.62	14.69	33.00	49.05	10.00

*All values given in degrees

The thirty sets of unique inlet/outlet conditions were input as query points into the selection tool in order to interpolate the blade profiles from the database of known optimized profiles. The query points are depicted in Figure 5.13 as red stars within the Delaunay triangulation.

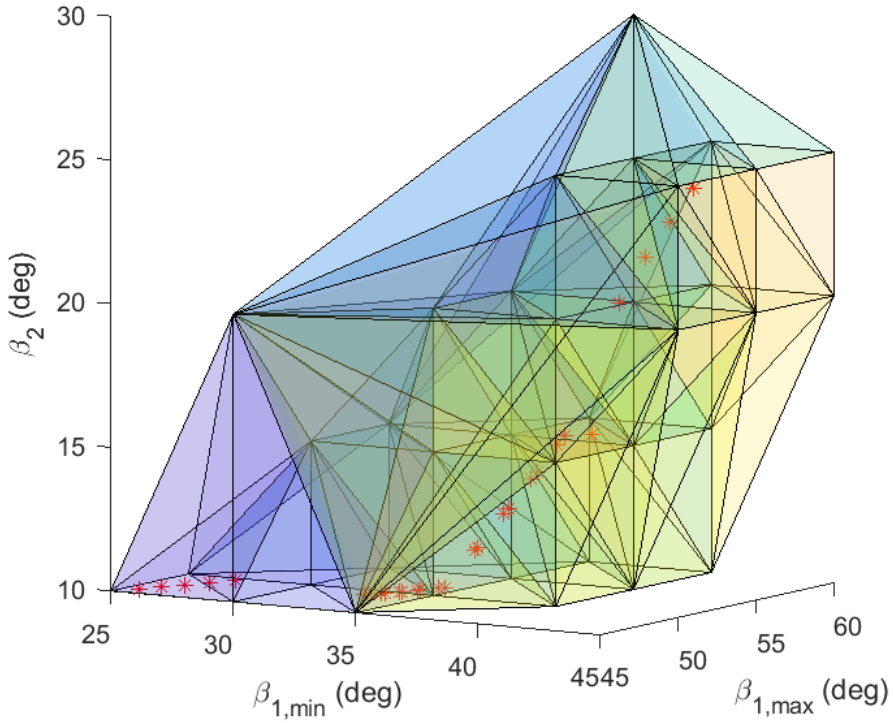


Figure 5.13. Query points for interpolation for the simplified three-stage compressor

For each query point, the required blade profile was interpolated from the profiles in the database corresponding to the points at the vertices of the tetrahedron containing the query point. The resulting blade profiles defined the design of the rotors and stators in a straight cascade. In order to turn the straight cascade into an axial, mixed-flow cascade, the transformation procedure outlined in Lewis [1] was reversed. Figure 5.14 shows the mixed-flow cascade transformation, where η and ζ are transformed to r and θ by Equations 5.1-5.2 [1], with γ representing the local cone angle.

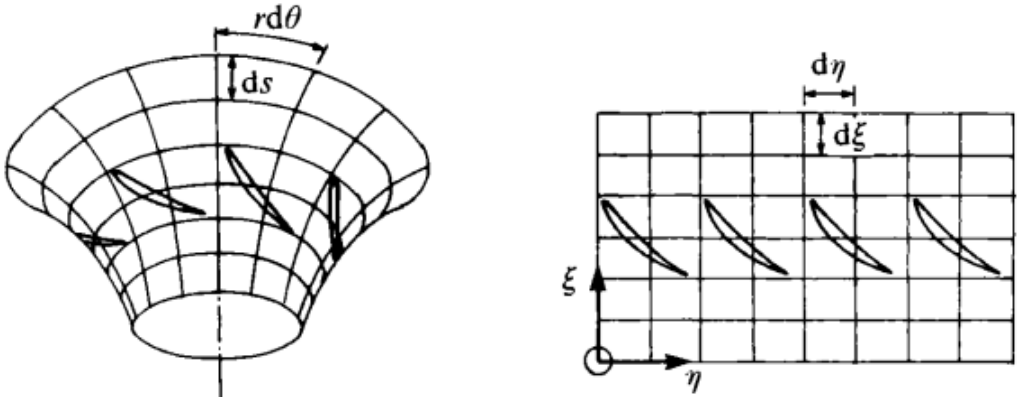


Figure 5.14. Transformation from straight cascade to mixed-flow cascade. Source: [1]

$$d\zeta = \frac{1}{\sin\gamma}\frac{dr}{r} \tag{5.1}$$

$$d\eta = d\theta \tag{5.2}$$

Figure 5.15 depicts the three-stage compressor design, with each blade profile placed according to its respective blade row and station. Blade cross-sections closest to the hub were colored black, while cross-sections near the casing were colored in the lightest gray.

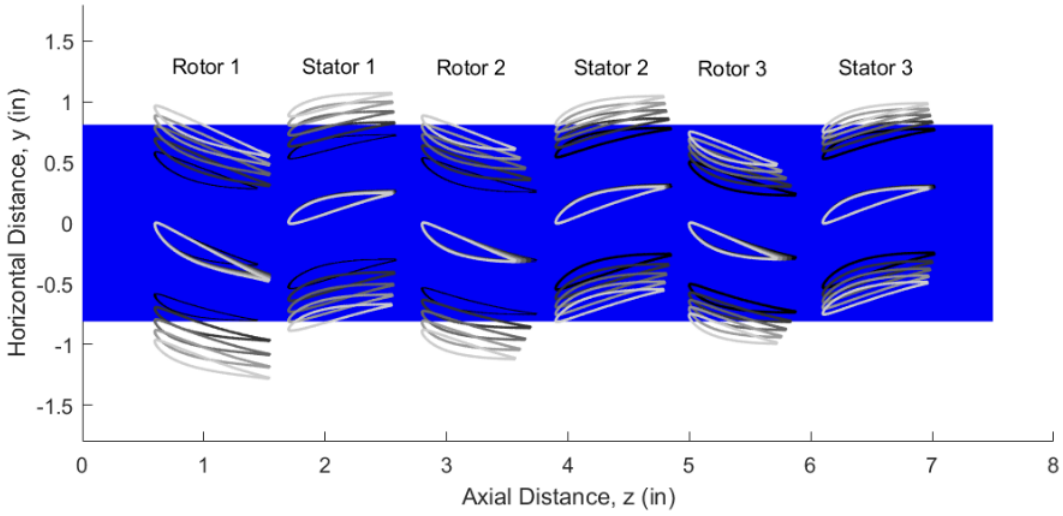


Figure 5.15. Design of the simplified three-stage compressor

In order to maintain the solidity required by the pitch of each profile as the radial distance increased, the chord length was adjusted, resulting in mild blade shortening in the radial direction. In addition, the chord length of each successive row of rotors was shorter than that of the previous row, which is consistent with the design of real compressors.

A validation test was performed on the most highly-loaded blade profile in the three-stage compressor to ensure that the interpolated profiles used for the compressor design were actually able to achieve the required flow turning. The most highly-loaded interpolated blade profile was the profile at the hub of Rotor 2, with a $\beta_{1,max} = 51.93°$ and $\beta_2 = 10°$. These conditions amounted to 41.93° of flow turning required, which was larger than the maximum flow turning required in the initial optimized profile validation tests. The procedure outlined in Chapter 4 was used to run a fluid simulation for the interpolated blade profile. The resulting velocity contour in shown below in Figure 5.16.

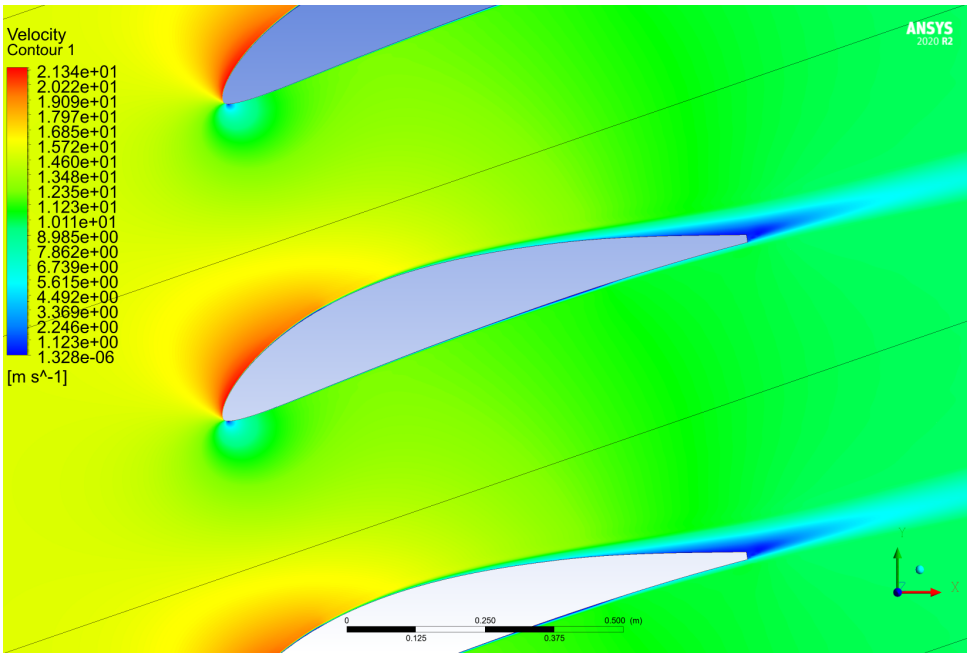


Figure 5.16. Interpolated compressor blade velocity contour: $\beta_1 = 51.93°$, $\beta_2 = 10°$

In Figure 5.16, the interpolated blade profile showed considerable flow separation at the trailing edge and a small separation bubble along the top surface of the blade. Both of these characteristics were consistent with those present in the fluid simulations of the optimized blade profiles. The outlet flow angle achieved by the blade was 13.24°, amounting to 38.69° of flow turning, 92.27% of the desired flow turning. This value is consistent with the flow turning achieved in the optimized profile validation tests. Assuming this holds true for the interpolated blade profiles with less loading, there was no significant loss in blade performance due to the use of interpolation, thus validating the used of the selection tool for the design of new blade profiles and multi-stage compressors.

CHAPTER 6:
Conclusions and Recommendations

6.1 Conclusions

The goal of this thesis was to develop a searchable, expandable database of functional blade profiles optimized over a range of inlet flow angles for a desired outlet flow angle. The VLM and Pohlhausen's method were both successfully implemented over blade profiles consisting of a discrete set of two-dimensional points, which allowed for the determination of aerodynamic and boundary layer characteristics. The resulting pressure gradient and shape factor profiles were implemented in a Nelder-Mead optimization method that modified the shape, pitch, and stagger of the blade profiles. The performances of the blades designed by the two cost functions were simulated using ANSYS-CFX. Both the H- and P-optimization cost functions designed blades that turned the range of inlet flow angles to the desired outlet flow angle, with diminishing flow turning as the inlet flow angle increased due to flow separation. However, the P-optimized blades were far more efficient than the H-optimized blades due to the achieved total pressure recovery across the cascade. For this reason, the P-optimized blades were chosen for the database.

The database was constructed as an array of optimized blade profile parameters and the associated inlet/outlet conditions. The selection tool was built using Delaunay triangulation in order to linearly interpolate scattered three-dimensional data points. The interpolated blade profiles closely approximated the optimized profile for a given set of inlet/outlet conditions. The pitch between consecutive blades demonstrated the largest difference between the optimized and interpolated profiles; although, this difference was smaller than the pitch differential between a single blade at the hub and the casing of a real compressor.

The selection tool was applied to a set of inlet/outlet conditions in order to design a three-stage compressor through the interpolation of optimized blade profiles. A fluid simulation was performed on the most highly-loaded blade profile to ensure flow turning was achieved in the interpolated profiles. Despite the formation of a small separation bubble and flow separation at the trailing edge, the interpolated blade profile performed similarly to the

optimized blade profiles with respect to flow turning. Ultimately, the selection tool successfully interpolated new profiles from the database of optimized profiles. Thus, the use of the selection tool greatly reduced the time required to design a blade profile that satisfied the desired inlet/outlet conditions. If further optimization was desired, the output of the selection tool provided initial values for the blade shape parameters close to the optimal solution.

6.2 Recommendations

Further research should be conducted in three areas of this project. First, different blade shape polynomials should be examined. Increasing the degrees of freedom for blade parameterization would allow for otherwise unobtainable blade shapes to be defined. A simple way to achieve this would be to implement a similar geometry generation method for modified NACA 5-series, 6-series, or 7-series airfoils. In addition, a combined H-P-optimization cost function should be investigated. Such a cost function could result in blade designs that achieve flow turning closer to that of the H-optimization, while maintaining a level of efficiency near that of the P-optimization. Finally, the fluid simulations should be continued in order to define the flow turning and total pressure loss curves with greater resolution. Inlet angles outside of the design inlet range should also be considered in order to determine start-up performance and stall characteristics.

APPENDIX A:
MATLAB Code

A.1 aerogrid

aerogrid.m is a MATLAB function that uses camber and thickness polynomials to generate discrete grid points along the surface of an airfoil.

```
% M-file to generate the grid points around an aerofoil.
%    where ouputs are
%       X_grid     = x coordinates of segment ends
%       Y_grid     = y coordinated of segment ends
%       Y_t        = section thickness at each chord point
%       X_c        = x coordinates along chord line
%       Y_c        = y coordinates along chord line
%       DY_DX      = slope along chord line
%
%    where inputs are
%       grid_no    = number of internal aerofoil stations
%       m_thick    = fraction of chord where thickness equation changes
%       leadthick  = a0, a1, a2, etc for yt =
%                    a0*(x/c)^0.5 + a1 + a2*(x/c) + a3*(x/c)^2 etc
%       m_cam      = fraction of chord where camber equation changes
%       leadcam    = b0, b1, b2, etc for yc =
%                    b0*(x/c)^0.5 + b1 + b2*(x/c) + b3*(x/c)^2 etc
%       trailthick = a0, a1, a2, etc for yt =
%                    a0*(1-x/c)^0.5 + a1 + a2*(1-x/c) + a3*(1-x/c)^2 etc
%       trailcam   = b0, b1, b2, etc for yc =
%                    b0*(1-x/c)^0.5 + b1 + b2*(1-x/c) + b3*(1-x/c)^2 etc
%       edgethick  = trailing edge thickness as a fraction of the chord
%
%       Note trailthick, trailcam can be left out and internal functions
%       will ensure nth order continuity from m_thick and m_cam to
%       trailing edge edgethick, if left out a zero trailing edge
%       thickness will be assumed

```

```
function [X_grid,Y_grid,Y_t,X_c,Y_c,DY_DX] = ...
   aerogrid(grid_no,m_thick,leadthick,m_cam,leadcam,trailthick,...
   trailcam,edgethick)

    % Assigns an edge thickness of zero if not specified
    if nargin < 8
        edgethick = 0;
    elseif isempty(edgethick)
        edgethick = 0;
    end

    % Constants for trailing edge thickness
    if nargin < 6
       temp = 1;
    elseif isempty(trailthick)
       temp = 1;
    else
       temp = 0;
    end

    if (m_thick < 1) && (temp)
       trailthick = trail_const(m_thick,leadthick,edgethick);
    elseif m_thick == 1
       trailthick = 0;
    end

    % Constants for trailing edge camber
    if nargin < 7
       temp = 1;
    elseif isempty(trailcam)
       temp = 1;
    else
       temp = 0;
    end

    if (m_cam < 1) && (temp)
       trailcam = trail_const(m_cam,leadcam);
    elseif m_cam == 1
       trailcam = 0;
```

```
68      end
69
70      % Define number of grid points, the trailing edge grid point, and
71      % the angles for circle grid (see Lewis)
72      M_no  = 2*(grid_no+1);
73      M_te  = M_no/2;
74      psi   = (0:M_te)*(2*pi/(M_no));
75
76      % Points along camber line
77      X_c   = 0.5*(1-cos(psi));
78
79      % Leading edge: camber points and thickness distribution calculations
80      temp        = find(X_c<=m_cam);
81      Y_c(temp)   = y_lead(X_c(temp),leadcam);
82      DY_DX(temp) = dy_dx_lead(X_c(temp),leadcam);
83      temp        = find(X_c<=m_thick);
84      Y_t(temp)   = y_lead(X_c(temp),leadthick);
85
86      % Trailing edge: camber points and thickness distribution calculations
87      temp        = find(X_c>m_cam);
88      Y_c(temp)   = y_trail(X_c(temp),trailcam);
89      DY_DX(temp) = dy_dx_trail(X_c(temp),trailcam);
90      temp        = find(X_c>m_thick);
91      Y_t(temp)   = y_trail(X_c(temp),trailthick,edgethick);
92      temp      = find(Y_t<0);
93      Y_t(temp) = abs(Y_t(temp));
94
95      % Combine leading and trailing edges for x-coordinate
96      X_grid(1:M_te+1)    = X_c - Y_t.*sin(DY_DX);
97      X_grid(M_te+2:M_no) = X_c(M_te:-1:2) + ...
98                            Y_t(M_te:-1:2).*sin(DY_DX(M_te:-1:2));
99      X_grid(M_te+1)      = 1;
100
101     % Combine leading and trailing edges for y-coordinate
102     Y_grid(1:M_te)      = Y_c(1:M_te) + Y_t(1:M_te).*cos(DY_DX(1:M_te));
103     Y_grid(M_te+2:M_no) = Y_c(M_te:-1:2) - ...
104                           Y_t(M_te:-1:2).*cos(DY_DX(M_te:-1:2));
105     Y_grid(M_te+1)      = 0;
106
```

```
107     % Blunt trailing edge is removed
108     if edgethick~=0
109         m1 = ((Y_grid(M_te)-Y_grid(M_te-1))/...
110               (X_grid(M_te)-X_grid(M_te-1)));
111         m2 = ((Y_grid(M_te+2)-Y_grid(M_te+3))/...
112               (X_grid(M_te+2)-X_grid(M_te+3)));
113         c1 = Y_grid(M_te) - m1*X_grid(M_te);
114         c2 = Y_grid(M_te+2) - m2*X_grid(M_te+2);
115         X_grid(M_te+1) = (c2-c1)/(m1-m2);
116         Y_grid(M_te+1) = m1*X_grid(M_te+1) + c1;
117     end
118 
119 %---------------------------------------------------------------------
120 % Function to calculate the y coordinate of the polynomial function
121 % used for aerofoils for the leading section of the blade
122 % y = a*x^0.5 + bx + c*x^2 + d*x^3 + etc
123 function [output] = y_lead(x,coeff)
124 
125     output = coeff(1)*x.^0.5;
126 
127     if length(coeff)>1
128         for i = 1:length(coeff)-1
129             output = output + coeff(i+1)*x.^i;
130         end
131     end
132 end
133 
134 %---------------------------------------------------------------------
135 % Function to calculate the y coordinate of the polynomial function
136 % used for aerofoils for the trailing section of the blade
137 % y = a*(1-x)^0.5 + b*(1-x) + c*(1-x)^2 + d*(1-x)^3 + etc
138 function [output] = y_trail(x,coeff,edgethick)
139 
140     if nargin < 3
141         edgethick = 0;
142     elseif isempty(edgethick)
143         edgethick = 0;
144     end
145 
```

```
146     output = coeff(1)*(1-x).^0.5;
147 
148     if length(coeff)>1
149         for i = 1:length(coeff)-1
150             output = output + coeff(i+1)*(1-x).^i;
151         end
152     end
153 
154     output = output + edgethick;
155 end
156 
157 %-------------------------------------------------------------------------
158 % Function to calculate the slope of the polynomial function used
159 % for aerofoils leading edge
160 % y = a*x^0.5 + b + c*x + d*x^2 + etc
161 function [output] = dy_dx_lead(x,coeff)
162 
163     output = zeros(size(x));
164 
165     if length(coeff)>1
166         for i = 2:length(coeff)
167             output = output + (i-1)*coeff(i)*x.^(i-2);
168         end
169     end
170 
171     output(2:length(x)) = output(2:length(x)) + 0.5*coeff(1)./...
172                           (sqrt(x(2:length(x))));
173     output = atan(output);
174 
175     if coeff(1) ~=0
176         output(1) = pi/2;
177     end
178 end
179 
180 %-------------------------------------------------------------------------
181 % Function to calculate the slope of the polynomial function used
182 % for aerofoils trailing edge
183 % y = a*x^0.5 + b + c*x + d*x^2 + etc
184 function [output] = dy_dx_trail(x,coeff)
```

```
185
186     output = zeros(size(x));
187
188     if length(coeff)>1
189         for i = 2:length(coeff)
190             output = output - (i-1)*coeff(i)*(1-x).^(i-2);
191         end
192     end
193
194     output(1:length(x)-1) = output(1:length(x)-1) - 0.5*coeff(1)./...
195                             (sqrt(1-x(1:length(x)-1)));
196     output = atan(output);
197
198     if coeff(1) ~=0
199         output(length(x)) = pi/2;
200     end
201 end
202
203 %---------------------------------------------------------------------
204 % Function to calculate the trailing edge constants of the aerofoil shape
205 function [output] = trail_const(m,coeff,edgethick)
206
207     if nargin < 3
208         edgethick = 0;
209     elseif isempty(edgethick)
210         edgethick = 0;
211     end
212
213     LHS = zeros(length(coeff)); RHS = zeros(length(coeff));
214
215     % First order continuity
216     LHS(1,1) = sqrt(1-m);
217     RHS(1,1) = sqrt(m);
218
219     if size(coeff,2)>1
220         for j = 2:length(coeff)
221             LHS(1,j) = (1-m)^(j-1);
222             RHS(1,j) = (m)^(j-1);
223         end
```

```
224
225         % Sqrt exponent is dealt with first
226         temp = 1;
227         for i = 2:length(coeff)
228             temp = temp*(-i+2.5);
229             LHS(i,1) = ((-1)^(i-1))*temp*((1-m)^(-i+1.5));
230             RHS(i,1) = temp*((m)^(-i+1.5));
231         end
232
233         temp = ones(1,length(coeff)-1);
234         for i = 2:length(coeff)
235             for j = i:length(coeff)
236                 temp(j-1) = temp(j-1)*(j-1-i+2);
237                 LHS(i,j) = temp(j-1)*((-1)^(i-1))*((1-m)^(j-1-i+1));
238                 RHS(i,j) = temp(j-1)*((m)^(j-1-i+1));
239             end
240         end
241     end % if if size(coeff,2)>1
242
243     RHS    = RHS*coeff';
244     RHS(1) = RHS(1) - edgethick;
245
246     % Number of constants is reduced by number of zero input constants
247     temp = find(coeff~=0);
248     if ~isempty(temp)
249         if temp(1) == 1
250             temp = temp(2:length(temp));
251         end
252     end
253     LHS  = LHS(:,temp);
254     LHS  = LHS(1:length(temp),:);
255     RHS  = RHS(1:length(temp));
256
257     output         = zeros(length(coeff),1);
258     output(temp,1) = LHS\RHS;
259
260     output = output';
261 end
```

A.2 aerosolve

aerosolve.m is a MATLAB function that performs the VLM on discrete grid points along the surface of an airfoil, producing various aerodynamic results including: surface velocity profiles, pressure gradient profiles, circulation, etc.

```
% M-file to perform flow analysis on aerofoil cascade.
%    where ouputs are
%       x_grid     = x coordinates of segment midpoints
%       y_grid     = y coordinated of segment midpoints
%       V_surf     = Surface velocity at midpoints
%       Coeff_pres = Coefficient of pressuse on surface
%       dp_dx_top  = Pressure gradient on top surface
%       dp_dx_bot  = Pressure gradient on bottom surface
%
%    where inputs are
%       X_grid     = x coordinates of segment ends
%       Y_grid     = y coordinated of segment ends
%       pitch      = spacing between blades
%       Beta_inlet = Inlet flow angle
%       stagger    = stagger angle
function [x_grid,y_grid,X_grid,Y_grid,V,ds,Coeff_pres,...
      Gam_circ,Beta_outlet,Del_s,Coeff_L,Del_S_cen, W_inf] = ...
      aerosolve(X_grid,Y_grid,pitch,Beta_inlet,stagger)

Beta_inlet = Beta_inlet*(pi/180);
M_no       = size(X_grid,2);    % No. of grid points
M_te       = M_no/2;            % Trailing edge grid point

% Segment midpoints x & y coord
x_grid     = [0.5*(X_grid(1:M_no-1)+X_grid(2:M_no))
              0.5*(X_grid(M_no)+X_grid(1))];
y_grid     = [0.5*(Y_grid(1:M_no-1)+Y_grid(2:M_no))
              0.5*(Y_grid(M_no)+Y_grid(1))];

% Grid is modified for stagger angle
grid_r     = sqrt(x_grid.^2 + y_grid.^2);
grid_angle = atan2(y_grid,x_grid);
x_grid     = grid_r.*cos(grid_angle + stagger*(pi/180));
```

```
y_grid     = grid_r.*sin(grid_angle + stagger*(pi/180));

grid_r     = sqrt(X_grid.^2 + Y_grid.^2);
grid_angle = atan2(Y_grid,X_grid);
X_grid     = grid_r.*cos(grid_angle + stagger*(pi/180));
Y_grid     = grid_r.*sin(grid_angle + stagger*(pi/180));

% Element lengths
Del_x       = [X_grid(2:M_no)-X_grid(1:M_no-1) X_grid(1)-X_grid(M_no)];
Del_y       = [Y_grid(2:M_no)-Y_grid(1:M_no-1) Y_grid(1)-Y_grid(M_no)];
Del_s_sqr   = Del_x.^2+Del_y.^2;
Del_s       = sqrt(Del_x.^2+Del_y.^2);

% Distance between centre points
Del_X_cen   = [x_grid(2:M_no)-x_grid(1:M_no-1) x_grid(1)-x_grid(M_no)];
Del_Y_cen   = [y_grid(2:M_no)-y_grid(1:M_no-1) y_grid(1)-y_grid(M_no)];
Del_S_cen   = sqrt(Del_X_cen.^2+Del_Y_cen.^2);

% Element slope
Beta_n       = atan_gannon(Del_x,Del_y);
temp         = find(abs(Beta_n(1:M_te))>pi);  % Correction for top surface
Beta_n(temp) = Beta_n(temp)+2*pi;
temp         = M_te+(find(Beta_n(M_te+1:M_no)>0)); % Correction for bottom
Beta_n(temp) = Beta_n(temp)-2*pi;

% Basic coupling matrix
Coup_K            = zeros(M_no,M_no);    % Memory allocation

% Generic elements of coupling matrix
for i = 1:M_no
   for j = 1:M_no
       dx = x_grid(j)-x_grid(i);
       dy = y_grid(j)-y_grid(i);
       Coup_K(i,j) = (Del_s(j)/(2*pitch))*...
           (sin((2*pi/pitch)*(-dy))*cos(Beta_n(i))-...
           sinh((2*pi/pitch)*(-dx))*sin(Beta_n(i)))/...
           (cosh((2*pi/pitch)*(-dx))-cos((2*pi/pitch)*(-dy)));
   end
end
```

```
clear dx dy

% Diagonal self coupling coefficients
Coup_K(1,1)       = -0.5-(Beta_n(2)-Beta_n(M_no)-2*pi)/(8*pi);
Coup_K(M_no,M_no) = -0.5-(Beta_n(1)-Beta_n(M_no-1)-2*pi)/(8*pi);
for i = 2:(M_no-1)
      Coup_K(i,i) = -0.5-(Beta_n(i+1)-Beta_n(i-1))/(8*pi);
end


% Back diagonal correction of coupling matrix
for i = 1:M_no
   temp = 0;
   for j = 1:M_no
      if (j~=(M_no-i+1))
         temp = temp - Coup_K(j,i)*Del_s(j);
      end
   end
   Coup_K((M_no-i+1),i) = temp/Del_s(M_no-i+1);
end

% Kutta condition
Coup_K(M_te+1,:) = zeros(size(Coup_K(M_te+1,:)));
Coup_K(M_te+1,M_te) = 1; Coup_K(M_te+1,M_te+1) = 1;

% Zero gradient conidition, upper surface
Coup_K(M_te,:) = zeros(size(Coup_K(M_te+1,:)));
Coup_K(M_te,M_te)   = -1;
Coup_K(M_te,M_te-1) = (Del_S_cen(M_te-2)+Del_S_cen(M_te-1))/...
                       Del_S_cen(M_te-2);
Coup_K(M_te,M_te-2) = 1-Coup_K(M_te,M_te-1);

% Zero gradient conidition, lower surface
Coup_K(M_te+2,:) = zeros(size(Coup_K(M_te+2,:)));
Coup_K(M_te+2,M_te+1) = -1;
Coup_K(M_te+2,M_te+2) = (Del_S_cen(M_te+2)+Del_S_cen(M_te+1))/...
                         Del_S_cen(M_te+2);
Coup_K(M_te+2,M_te+3) = 1-Coup_K(M_te+2,M_te+2);

```

```
112
113 % Inverse of coupling matrix
114 Coup_K_inv       = Coup_K\eye(size(Coup_K));
115
116 % Unit solutions
117 RHS_unit_circ         = ones(length(Coup_K),1);
118 RHS_unit_circ(M_te+1) = 0; % Kutta
119 RHS_unit_circ(M_te)   = 0; % Zero gradient
120 RHS_unit_circ(M_te+2) = 0; % Zero gradient
121 Unit_circ             = Coup_K_inv*RHS_unit_circ;
122
123 RHS_unit_U            = -cos(Beta_n)';
124 RHS_unit_U(M_te+1)    = 0; % Kutta
125 RHS_unit_U(M_te)      = 0; % Zero gradient
126 RHS_unit_U(M_te+2)    = 0; % Zero gradient
127 Unit_U                = Coup_K_inv*RHS_unit_U;
128
129 RHS_unit_V            = -sin(Beta_n)';
130 RHS_unit_V(M_te+1)    = 0; % Kutta
131 RHS_unit_V(M_te)      = 0; % Zero gradient
132 RHS_unit_V(M_te+2)    = 0; % Zero gradient
133 Unit_V                = Coup_K_inv*RHS_unit_V;
134
135 % Unit circulations
136 Gamma_U = sum(Del_s'.*Unit_U);
137 Gamma_V = sum(Del_s'.*Unit_V);
138
139 %[Del_s(M_te)*Unit_U(M_te)+Del_s(M_te+1)*Unit_U(M_te+1)]
140 %[Del_s(M_te)*Unit_V(M_te)+Del_s(M_te+1)*Unit_V(M_te+1)]
141 %pause
142
143 % Outlet angle and infinite flow angle
144 temp1 = 1 - Gamma_V/(2*pitch); temp2 = 1 + Gamma_V/(2*pitch);
145 Beta_outlet = atan((temp1/temp2)*tan(Beta_inlet) - ...
146               (2/temp2)*Gamma_U/(2*pitch));
147 Beta_inf    = atan(0.5*(tan(Beta_inlet)+tan(Beta_outlet)));
148
149 % Velocities
150 W_inf  = cos(Beta_inlet)/cos(Beta_inf);
```

```
U_inf  = W_inf*cos(Beta_inf);
V_inf = W_inf*sin(Beta_inf);
V_surf = U_inf*Unit_U + V_inf*Unit_V;

% Coefficient of pressure on surface
Coeff_pres      = 1-V_surf.^2;

% Output variable stored in records
V.surf    = V_surf;
V.surfcum = circshift(V.surf,M_te); % Shifted to start and end at TE

ds.circ   = circshift(Del_s',M_te);
ds.circ   = [0; (1/2)*ds.circ(1:end-1)+(1/2)*ds.circ(2:end)];
ds.cum    = cumsum(ds.circ); % cumulative distance from the trailing edge

% Stagnation point added in by interpolation
if max(V.surf) ~= 0
    ds.cum = [ds.cum(V.surfcum<0);...
        interp1([V.surfcum(sum(V.surfcum < 0)) ...
                 V.surfcum(sum(V.surfcum < 0)+1)]',...
                 [ds.cum(sum(V.surfcum < 0)) ...
                 ds.cum(sum(V.surfcum < 0)+1)]',0);...
        ds.cum(V.surfcum>0)];
    V.surfcum = [V.surfcum(V.surfcum < 0); 0; V.surfcum(V.surfcum > 0)];
end % if max

% Coefficient of pressure on surface
V.Cp = 1-V.surfcum.^2;

% Top and bottom profiles are separated for simplicity
V.top     = V.surfcum(V.surfcum>=0);
V.Cp_top  = V.Cp(V.surfcum>=0);
V.bot     = flipud(-V.surfcum(V.surfcum<=0));
V.Cp_bot  = flipud(V.Cp(V.surfcum<=0));

ds.top    = ds.cum(V.surfcum>=0); ds.top = ds.top-ds.top(1);
ds.bot    = ds.cum(V.surfcum<=0); ds.bot = flipud(ds.bot(end)-ds.bot);

% Pressure gradients
```

```
V.dCp_top = gradient(V.Cp_top)./gradient(ds.top);
V.dCp_bot = gradient(V.Cp_bot)./gradient(ds.bot);

% Circulation
Gam_circ   = sum(Del_s'.*V_surf);

% Changed to degree
Beta_outlet = (180/pi)*Beta_outlet;

% Coeff of lift
Coeff_L = 2*Gam_circ/(1*W_inf);
end


% Atan function
% a robust atan function that returns the angle for the vortex method.
function theta = atan_gannon(x,y)
    if nargin == 1
       theta = atan(x);
    end

    if nargin == 2
       r = sqrt(x.^2+y.^2);
       theta_sin = asin(y./r);
       theta_cos = acos(x./r);
       % 1st quadrant
       i = find(x>=0);
       j = find(y>=0);
    %   intersect(i,j)
       theta(intersect(i,j)) = theta_sin(intersect(i,j));
       % 2nd quadrant
       i = find(x<0);
       j = find(y>=0);
    %   intersect(i,j)
       theta(intersect(i,j)) = pi - theta_sin(intersect(i,j));
       % 3rd quadrant
       i = find(x<0);
       j = find(y<0);
    %   intersect(i,j)
```

```
229         theta(intersect(i,j)) = -pi-theta_sin(intersect(i,j));
230         % 4th quadrant
231         i = find(x>=0);
232         j = find(y<0);
233     %    intersect(i,j)
234         theta(intersect(i,j)) = theta_sin(intersect(i,j));
235 
236         % Correction into +x direction
237         i = find(x<0);
238         j = find(theta>0);
239         theta(intersect(i,j)) = theta(intersect(i,j))-2*pi;
240     end
241 
242     if length(theta) ~= length(x)
243         disp('problem')
244         theta
245         x
246         y
247     end
248 end
```

A.3 Cam_opt_fun

Cam_opt_fun.m is a MATLAB function that optimizes a blade cascade over a range of inlet flow angles for a desired outlet flow angle.

```
1  % M-file to optimize an aerofoil for a range of inlet flow angles and
2  % one outlet angle
3  %
4  % Inputs are:
5  % inletMaxBeta = maximum inlet angle of the inlet range (degrees)
6  % inletMinBeta = minimum inlet angle of the inlet range (degrees)
7  % outletBeta   = desired outlet angle (degrees)
8  % preSolution  = 0 if no previous solution
9  %              = 1 if there is a previous solution
10 %
11 % Outputs are:
12 %            X = blade shape parameters (except for stagger angle)
```

```
%stagger_final = stagger angle (degrees)
%     EXITFLAG = 0 if the optimization did not converge
%              = 1 if the optimization did converge
function [X, stagger_final, EXITFLAG] = Cam_opt_fun(inletMaxBeta, ...
          inletMinBeta, outletBeta, preSolution)

    % Function name
    FUN = 'Find_Cp_max';

    % Calculation constants
    global grid_no
    grid_no = 200;

    global Max_vel_pos Beta_inlet Beta_outlet Beta_tol edgethick

    % Flow constants
    % Furtherest foward that max velocity may be, 0.1 recommended
    Max_vel_pos = 0.1;

    % Design Inlet flow angles
    % More make for smoother optimization but of course slower
    Beta_inlet  = linspace(inletMaxBeta, inletMinBeta, 9);
    Beta_outlet = [outletBeta]; % Required outlet flow angles
    Beta_tol    = 0.0001;  % Angle tolerance
    edgethick    = 0.05;    % Thickness of trailing edge

    % Initial profile constants, could be loaded from previous file
    m_ord_top       = 0.0250; % Starting amount of camber
    m_ord_bot       = 0.0150; % Starting amount of camber
    p_cam_top       = 0.2500; % Position of max camber
    p_cam_bot       = 0.5000; % Position of max camber
    m_thick_top     = 0.2500; % Position of maximum thickness
    m_thick_bot     = 0.2500; % Position of maximum thickness
    trail_slope_top = 0.4000; % Amount of slope at the trailing edge
    trail_slope_bot = 0.2000; % Amount of slope at the trailing edge
    mid_slope_top   = 0.0;    % Top slope joining leading/trailing edge
    mid_slope_bot   = 0.0;    % Bottom slope joining leading/trailing edge
    thickness_top   = 0.1300; % Blade thickness on top
    thickness_bot   = 0.1600; % Blade thickness on bottom
```

```
I_nose_top      = 6.0000; % Nose radius bottom (6 = standard)
I_nose_bot      = 5.0000; % Nose radius bottom (6 = standard)
pitch           = 0.6500; % Starting amount of pitch

%Create array of blade parameters
XO = [m_ord_top m_ord_bot p_cam_top p_cam_bot m_thick_top ...
      m_thick_bot trail_slope_top trail_slope_bot...
      thickness_top thickness_bot I_nose_top I_nose_bot pitch]';

%Loads previous solution for blade parameters if they exist
%for the given inlet/outlet conditions
if preSolution == 1
    nameString = ['P_Inlet_',num2str(inletMaxBeta),'to', ...
         num2str(inletMinBeta),'_Outlet_',num2str(outletBeta),'.mat'];
    data = load(nameString);
    XO = data.X;
end

%Set iteration and function evaluation limits
options = optimset('display','iter','MaxIter',2500,'MaxFunEvals',2500);

%Perform optimization
[X,FVAL,EXITFLAG,OUTPUT] = fminsearch_2007_constraint(FUN,XO,options);

%Determine required stagger angles
[F_min,g, stagger_final] = Find_Cp_max(X,1);

% Generate SolidWorks geometry for CFX
GeomCascade([1 1]);

%Save blade parameter data to file with naming convention
inletMax    = num2str(inletMaxBeta);
inletMin    = num2str(inletMinBeta);
outletAngle = num2str(outletBeta);
InletMaxA   = inletMaxBeta;
InletMinA   = inletMinBeta;
outletB     = outletBeta;
camFileName = ['P_Inlet_',inletMax,'to',inletMin,'_Outlet_',...
               outletAngle];
```

```
91      save(camFileName,'InletMaxA','InletMinA','outletB','X','stagger_final')
92  end
```

A.4 camDatabase

camDatabase.m is a MATLAB function that compiles all of the optimized profile data into a single ".mat" file.

```
1
2   %Function to create the database of optimized blade profiles
3   function [] = camDatabase()
4
5       %Choose all files that conform to the naming convention
6       camStruct = dir("P_Inlet_*.mat");
7
8       %Memory allocation
9       camData = zeros(length(camStruct), 17);
10
11      %Loop through each file
12      for i = 1:length(camStruct)
13          %Load data
14          load(camStruct(i).name);
15
16          %Fill inlet/outlet conditions
17          camData(i, 1)    = InletMinA;
18          camData(i, 2)    = InletMaxA;
19          camData(i, 3)    = outletB;
20
21          %Fill blade shape parameters
22          for j = 1:length(X)
23              camData(i, j+3) = X(j);
24          end
25
26          %Fill stagger angle
27          camData(i, 17)   = stagger_final;
28      end
29
30      %Save final array to camDatabase.mat
```

```
    save('camDatabase.mat', 'camData')
end
```

A.5 camSelectionTool

camSelectionTool.m is a MATLAB function that either returns known optimized profiles from the database or interpolates new profiles using Delaunay triangulation.

```
%Function to interpolate new blade profiles for an existing database
%
% Inputs are:
%       inletMax = maximum angle of the desired inlet range (degrees)
%       inletMin = minimum angle of the desired inlet range (degrees)
%         outlet = desired outlet angle (degrees)
%
% Outputs are:
%              X = blade shape parameters (except for stagger angle)
% stagger_final = stagger angle (degrees)
function [X, stagger_final] = camSelectionTool(inletMax, inletMin, outlet)

    %Redesignate desired inlet/outlet conditions
    beta1min = inletMin;
    beta1max = inletMax;
    beta2    = outlet;

    %Load the database of optimized profiles
    load('camDatabase.mat');

    %Inlet/outlet conditions of the optimized profiles
    minInlets = camData(:, 1);
    maxInlets = camData(:, 2);
    Outlets   = camData(:, 3);

    %Determine if the profile has already been optimized
    isHere = 0;
    for i = 1:length(minInlets)
        if beta1min == minInlets(i) && beta1max == maxInlets(i) && ...
           beta2 == Outlets(i)
```

```
            X = camData(i, 4:16);
            stagger_final = camData(i, 17);
            isHere = 1;
        end
    end

    %If the desired profile is not in the database, interpolate
    if isHere == 0
        X = zeros(13, 1);
        %Interpolate over each of the blade parameters
        for i = 1:length(X)
            X(i) = griddata(minInlets, maxInlets, Outlets, ...
                   camData(:, i+3), beta1min, beta1max, beta2);
        end
        stagger_final = griddata(minInlets, maxInlets, Outlets, ...
                        camData(:, 17), beta1min, beta1max, beta2);
    end

    %Display whether the profile was interpolated/drawn from database
    if isHere == 1
        disp('This Profile exists in the database')
    else
        disp('This Profile was interpolated')
    end

    %Plot Interpolated Profile along with the profiles used
    m_ord_top       = X(1);
    m_ord_bot       = X(2);
    p_cam_top       = X(3);
    p_cam_bot       = X(4);
    m_thick_top     = X(5);
    m_thick_bot     = X(6);
    trail_slope_top = X(7);
    trail_slope_bot = X(8);
    thickness_top   = X(9);
    thickness_bot   = X(10);
    I_nose_top      = X(11);
    I_nose_bot      = X(12);
    pitch           = X(13);
```

```
70
71      % Calculation constants
72      grid_no = 200;
73      M_no  = 2*(grid_no+1);
74      M_te  = M_no/2; % Trailing edge grid point
75      edgethick = 0.05;
76
77      % Top Leading and trailing edge thicknes for a modified NACA profile
78      [leadthick,trailthick] = ...
79          NACA_4_mid_slope(thickness_top,m_thick_top,I_nose_top,edgethick,...
80          trail_slope_top,mid_slope_top);
81      % NACA 4 digit constants for leading edge polynomial
82      leadcam  = [0 2*m_ord_top/p_cam_top -m_ord_top/(p_cam_top^2)];
83      trailcam = [];
84
85      % Top Aerofoil grid is set up
86      [X_grid_top,Y_grid_top] =...
87          aerogrid(grid_no,m_thick_top,leadthick,p_cam_top,leadcam,...
88          trailthick,trailcam,edgethick*thickness_top);
89
90      % Bottom Leading and trailing edge thicknes for a modified NACA profile
91      [leadthick,trailthick] = ...
92          NACA_4_mid_slope(thickness_bot,m_thick_bot,I_nose_bot,edgethick,...
93          trail_slope_bot,mid_slope_bot);
94      % NACA 4 digit constants for leading edge polynomial
95      leadcam  = [0 2*m_ord_bot/p_cam_bot -m_ord_bot/(p_cam_bot^2)];
96      trailcam = [];
97
98      % Bottom Aerofoil grid is set up
99      [X_grid_bot,Y_grid_bot] =...
100         aerogrid(grid_no,m_thick_bot,leadthick,p_cam_bot,leadcam,...
101         trailthick,trailcam,edgethick*thickness_bot);
102
103     Y_grid              = zeros(size(Y_grid_top));
104     X_grid              = zeros(size(X_grid_top));
105     Y_grid(1:M_te)      = Y_grid_top(1:M_te);
106     Y_grid(M_te+1:end)  = Y_grid_bot(M_te+1:end);
107     X_grid(1:M_te)      = X_grid_top(1:M_te);
108     X_grid(M_te+1:end)  = X_grid_bot(M_te+1:end);
```

```
109
110     [xi, yi] = line_intersection([X_grid_top(M_te) Y_grid_top(M_te) ...
111         X_grid_top(M_te+1) Y_grid_top(M_te+1)], [X_grid_bot(M_te+1) ...
112         Y_grid_bot(M_te+1) X_grid_bot(M_te+2) Y_grid_bot(M_te+2)]);
113     X_grid(M_te+1) = xi;
114     Y_grid(M_te+1) = yi;
115
116     % Aerofoil shape
117     X_grid_plot = X_grid.*(X_grid <= 1);
118     Y_grid_plot = [0 Y_grid(X_grid_plot ~= 0)];
119     X_grid_plot = [0 X_grid_plot(X_grid_plot ~= 0)];
120
121     % Add stagger
122     staggeredX = cosd(stagger_final)*(X_grid_plot) - ...
123         sind(stagger_final)*(Y_grid_plot);
124     staggeredY = sind(stagger_final)*(X_grid_plot) + ...
125         cosd(stagger_final)*(Y_grid_plot);
126
127     %Plot blade profile
128     figure(10);
129     plot(X_grid_plot,Y_grid_plot,'r-')
130     hold on;
131     plot(staggeredX, staggeredY, 'r-')
132     plot(staggeredX, staggeredY+pitch, 'r-')
133     axis equal;
134     hold off;
135 end
```

A.6 Find_Cp_Max

Find_Cp_Max.m is a MATLAB function that calculates the cost function for a given blade geometry over the stated inlet angle range for a desired outlet angle.

```
1 % m-file to calculate the required stagger angle for a required exit
2 % angle for a compressor type cascade; this file also calculates the
3 % goal function for the optimization of the blade
4 function [F_min,g,stagger_final] = Find_Cp_max(X, plot_or_not)
5
```

```
6       %Designates shape factor or pressure for the goal function
7       H_or_P = 1;
8
9       %Initialize the blade parameters
10      m_ord_top       = X(1);
11      m_ord_bot       = X(2);
12      p_cam_top       = X(3);
13      p_cam_bot       = X(4);
14      m_thick_top     = X(5);
15      m_thick_bot     = X(6);
16      trail_slope_top = X(7);
17      trail_slope_bot = X(8);
18      thickness_top   = X(9);
19      thickness_bot   = X(10);
20      I_nose_top      = X(11);
21      I_nose_bot      = X(12);
22      pitch           = X(13);
23      mid_slope_top   = 0;
24      mid_slope_bot   = 0;
25
26      % Calculation constants
27      global grid_no
28      M_no  = 2*(grid_no+1);          % No. of grid points
29      M_te  = M_no/2;                 % Trailing edge grid point
30
31      global Max_vel_pos Beta_inlet Beta_outlet Beta_tol edgethick
32
33      % Stagger is stored
34      persistent stagger_old
35      if isempty(stagger_old)
36          stagger_old = 20;
37          stagger     = stagger_old;
38      else
39          stagger     = stagger_old;
40      end
41
42      % Top Leading and trailing edge thicknes for a modified NACA profile
43      [leadthick,trailthick] = NACA_4_mid_slope(thickness_top,m_thick_top,...
44          I_nose_top,edgethick,trail_slope_top,mid_slope_top);
```

```
45
46      % NACA 4 digit constants for leading/trailing edge polynomial (top)
47      leadcam   = [0 2*m_ord_top/p_cam_top -m_ord_top/(p_cam_top^2)];
48      trailcam  = [];
49
50      % Top Aerofoil grid is set up
51      [X_grid_top,Y_grid_top] = aerogrid(grid_no,m_thick_top,leadthick,...
52          p_cam_top,leadcam,trailthick,trailcam,edgethick*thickness_top);
53
54      % Bottom Leading and trailing edge thicknes for a modified NACA profile
55      [leadthick,trailthick] = NACA_4_mid_slope(thickness_bot,m_thick_bot,...
56          I_nose_bot,edgethick,trail_slope_bot,mid_slope_bot);
57
58      % NACA 4 digit constants for leading/trailing edge polynomial (bottom)
59      leadcam   = [0 2*m_ord_bot/p_cam_bot -m_ord_bot/(p_cam_bot^2)];
60      trailcam  = [];
61
62      % Bottom Aerofoil grid is set up
63      [X_grid_bot,Y_grid_bot] = aerogrid(grid_no,m_thick_bot,leadthick,...
64          p_cam_bot,leadcam,trailthick,trailcam,edgethick*thickness_bot);
65
66      %Combine the top and bottom aerofoil grids
67      X_grid         = zeros(size(X_grid_top));
68      Y_grid         = zeros(size(Y_grid_top));
69      X_grid(1:M_te) = X_grid_top(1:M_te);
70      X_grid(M_te+1:end) = X_grid_bot(M_te+1:end);
71      Y_grid(1:M_te) = Y_grid_top(1:M_te);
72      Y_grid(M_te+1:end) = Y_grid_bot(M_te+1:end);
73
74      %Correct trailing edge point is generated
75      [xi, yi] = line_intersection([X_grid_top(M_te) Y_grid_top(M_te) ...
76          X_grid_top(M_te+1) Y_grid_top(M_te+1)], [X_grid_bot(M_te+1) ...
77          Y_grid_bot(M_te+1) X_grid_bot(M_te+2) Y_grid_bot(M_te+2)]);
78      X_grid(M_te+1) = xi;
79      Y_grid(M_te+1) = yi;
80
81      % Plot original unstaggered aerofoil
82      if nargin ~=1
83          figure(1); close; figure(1);
```

```
84          plot(X_grid,Y_grid,'+r')
85          axis equal
86      end
87
88      % Recall that the stagger method for aerosolve is an adjustment
89      % Stagger is adjusted until the outlet angle is within tolerance for
90      % all inlet angles
91      check = true;
92      Beta_outlet_calc = Beta_outlet+2*Beta_tol;
93      while (abs(max(Beta_outlet_calc) - Beta_outlet) > Beta_tol || check)
94          %Perform aerodynamic calculations for each inlet angle
95          for i = 1:length(Beta_inlet)
96
97              %Designate plot styles
98              if i >= 2
99                  plotstyle_top = '--g';
100                 plotstyle_bot = '--r';
101             else
102                 plotstyle_top = 'g';
103                 plotstyle_bot = 'r';
104             end
105
106             % Solve for aerodynamic characteristics
107             [x_grid,y_grid,X_grid,Y_grid,V,ds,Gam_circ,...
108                 Beta_outlet_calc(i),Del_s,Coeff_L(i),Del_S_cen,W_inf] = ...
109                 aerosolvetandem(X_grid,Y_grid,pitch,Beta_inlet(i),stagger);
110
111             % Weighting function to keep flow accelerating
112             % over a certain section (first 0.1*c) of the blade
113             Max_vel_weight_top = ...
114                 double(Max_vel_pos*ds.top(end) <= ds.top');
115             Max_vel_weight_top(1:sum(Max_vel_weight_top == 0)) = ...
116                 [linspace(1/sum(Max_vel_weight_top == 0),...
117                 1-1/sum(Max_vel_weight_top == 0),...
118                 sum(Max_vel_weight_top == 0))]';
119
120             Max_vel_weight_bot = ...
121                 double(Max_vel_pos*ds.bot(end) <= ds.bot');
122             Max_vel_weight_bot(1:sum(Max_vel_weight_bot == 0)) = ...
```

```
123                 [linspace(1/sum(Max_vel_weight_bot == 0),...
124                 1-1/sum(Max_vel_weight_bot == 0),...
125                 sum(Max_vel_weight_bot == 0))]';
126 
127             % Apply weighting factors to the pressure gradient
128             dp_dx_top_max(i) = max(V.dCp_top'./Max_vel_weight_top);
129             dp_dx_bot_max(i) = max(V.dCp_bot'./Max_vel_weight_bot);
130 
131             stagger = 0; % Stagger is reset
132 
133             % Check for realistic pressure gradients
134             if min(V.dCp_bot) > 0 || min(V.dCp_top) > 0
135                 disp('There seems to be a problem in Find_Cp_max')
136                 V.dCp_bot
137                 pause
138             end
139 
140             % Perform shape factor calculations if and only if the blade
141             % stagger is correct
142             if ~(abs(max(Beta_outlet_calc) - Beta_outlet) > Beta_tol) ...
143                 && H_or_P == 0
144 
145                 % Function to calculate the properties of the boundary
146                 % layers on the upper and lower surfaces
147                 dstar_Theta_top = (BLCalcs(W_inf, V.top, ds.top,...
148                     V.dUds_top, V.d2Uds2_top))';
149                 dstar_Theta_bot = (BLCalcs(W_inf, V.bot, ds.bot,...
150                     V.dUds_bot, V.d2Uds2_bot))';
151 
152                 % Apply a polynomial approximation for the true maximum of
153                 % shape factor on the upper and lower surfaces
154                 [maxDstar_top, imax] = max(dstar_Theta_top);
155                 if imax == length(dstar_Theta_top) || imax == 1
156                     dstar_Theta_top_max(i) = maxDstar_top;
157                 else
158                     [~, dstar_Theta_top_max(i)] = ...
159                         Polymax(ds.top(imax-1:imax+1), ...
160                         dstar_Theta_top(imax-1:imax+1));
161                 end
```

```
162
163                 [maxDstar_bot, imax] = max(dstar_Theta_bot);
164                 if imax == length(dstar_Theta_bot) || imax == 1
165                     dstar_Theta_bot_max(i) = maxDstar_bot;
166                 else
167                     [~, dstar_Theta_bot_max(i)] = ...
168                         Polymax(ds.bot(imax-1:imax+1), ...
169                         dstar_Theta_bot(imax-1:imax+1));
170                 end
171
172                 % Plot the shape factor data
173                 if nargin ~= 1
174                     figure(5);
175                     plot(ds.top(2:end), dstar_Theta_top(2:end), ...
176                         plotstyle_top)
177                     hold on; grid on;
178                     plot(ds.bot(2:end), dstar_Theta_bot(2:end), ...
179                         plotstyle_bot)
180                     title('Shape Factor')
181                 end
182                 check = false;
183             elseif ~(abs(max(Beta_outlet_calc) - Beta_outlet) > Beta_tol)
184                 check = false;
185             end
186
187             % Plot pressure coefficient and surface velocity
188             if nargin ~=1
189
190                 % Pressure coefficient
191                 figure(2);
192                 plot(ds.top,-V.Cp_top,plotstyle_top)
193                 hold on
194                 plot(ds.bot,-V.Cp_bot,plotstyle_bot)
195                 title('Pressure Coefficient (1-V_s_u_r_f^2)')
196
197                 % Surface velocity
198                 figure(3);
199                 plot(ds.top,V.top,plotstyle_top)
200                 hold on
```

```
201                 plot(ds.bot,-V.bot,plotstyle_bot)
202                 title('Velocity (V_s_u_r_f)')
203             end
204         end
205 
206         %Re-initialize stagger with the new value
207         stagger     = -(max(Beta_outlet_calc) - Beta_outlet);
208         stagger_old = stagger_old + stagger;
209         stagger_final = stagger_old;
210     end
211 
212     % Sum the maximum pressure gradient for each inlet angle
213     dp_dx_max = sum([dp_dx_top_max dp_dx_bot_max]);
214 
215     % Sum the maximum shape factor for each inlet angle
216     if H_or_P == 0
217         dstar_Theta_max = sum([dstar_Theta_top_max dstar_Theta_bot_max]);
218     end
219 
220     % Determine the value of the goal function
221     if H_or_P == 1
222         F_min = dp_dx_max;
223     else
224         F_min = dstar_Theta_max;
225     end
226 
227     g = -1;
228 
229     % Display certain performance characteristics of the aerofoil
230     if nargin ~=1
231 
232         % Save blade coordinates to a file
233         temp_xyz = [X_grid' Y_grid' zeros(size(Y_grid'))];
234         temp_xyz = [temp_xyz; temp_xyz(1,:)];
235         temp_xyz = [temp_xyz temp_xyz(:,1)+pitch temp_xyz(:,2:3)];
236         save('xyz.txt', 'temp_xyz' , '-ascii')
237 
238         % Plot the final blade coordinates
239         figure(1); close; figure(1);
```

```
240         hold on
241         plot(X_grid,Y_grid,'g',X_grid,Y_grid,'+r')
242         plot(X_grid,Y_grid+pitch,'g',X_grid,Y_grid+pitch,'+r')
243         axis equal
244 
245         % Display theoretical maximum shape factors, lift coefficient,
246         % and stagger of the blade profile
247         if H_or_P == 0
248             disp(['Max Shape Factor H for all blade angles ...' ...
249                 '(Suction surface)'])
250             dstar_Theta_top_max
251             disp(['Max Shape Factor H for all blade angles ...' ...
252                 '(Pressure surface)'])
253             dstar_Theta_bot_max
254         end
255         disp('Coefficient of lift all blade angles')
256         Coeff_L
257         disp('Blade stagger angle (degrees)')
258         stagger_final
259     end
260 end
261 
262 % Function to calculate the the maximum of a quadratic polynomial
263 % passing through three point
264 function [Xmax, Ymax] = Polymax(x, y)
265 
266     parms = [x.^2  x  ones(size(x))]\y; %Quadratic polynomial values
267     Xmax = -parms(2)./(2*parms(1));
268     Ymax = sum(parms.*[Xmax^2; Xmax; 1]);
269 end
```

A.7 fminsearch_2007_constraint

fminsearch_2007_constraint.m is a MATLAB function that employs the Nelder-Mead method to find the minimum of a given function.

```
1 function [x,fval,exitflag,output] = ...
2     fminsearch_2007_constraint(funfcn,x,options,varargin)
```

```
% Modified version of fminsearch (Nelder-Mead) to handle constraint and
% also move into feasible region before optimizing.

defaultopt =struct('Display','notify','MaxIter','200*numberOfVariables',...
    'MaxFunEvals','200*numberOfVariables','TolX',1e-4,'TolFun',1e-4, ...
    'FunValCheck','off','OutputFcn',[]);

% If just 'defaults' passed in, return the default options in X
if nargin==1 && nargout <= 1 && isequal(funfcn,'defaults')
    x = defaultopt;
    return
end

if nargin < 2,
    error('MATLAB:fminsearch:NotEnoughInputs',...
        'FMINSEARCH requires at least two input arguments');
end

if nargin<3, options = []; end

% Check for non-double inputs
if ~isa(x,'double')
  error('MATLAB:fminsearch:NonDoubleInput', ...
         'FMINSEARCH only accepts inputs of data type double.')
end

n = numel(x);
numberOfVariables = n;

printtype = optimget(options,'Display',defaultopt,'fast');
tolx = optimget(options,'TolX',defaultopt,'fast');
tolf = optimget(options,'TolFun',defaultopt,'fast');
maxfun = optimget(options,'MaxFunEvals',defaultopt,'fast');
maxiter = optimget(options,'MaxIter',defaultopt,'fast');
funValCheck=strcmp(optimget(options,'FunValCheck',defaultopt,'fast'),'on');

% In case the defaults were gathered from calling: optimset('fminsearch'):
if ischar(maxfun)
    if isequal(lower(maxfun),'200*numberofvariables')
```

```
42          maxfun = 200*numberOfVariables;
43      else
44          error('MATLAB:fminsearch:OptMaxFunEvalsNotInteger',...
45              ['Option ''MaxFunEvals'' must be an integer value' ...
46              'if not the default.'])
47      end
48  end
49  if ischar(maxiter)
50      if isequal(lower(maxiter),'200*numberofvariables')
51          maxiter = 200*numberOfVariables;
52      else
53          error('MATLAB:fminsearch:OptMaxIterNotInteger',...
54           'Option ''MaxIter'' must be an integer value if not the default.')
55      end
56  end
57
58  switch printtype
59      case 'notify'
60          prnt = 1;
61      case {'none','off'}
62          prnt = 0;
63      case 'iter'
64          prnt = 3;
65      case 'final'
66          prnt = 2;
67      case 'simplex'
68          prnt = 4;
69      otherwise
70          prnt = 1;
71  end
72  % Handle the output
73  outputfcn = optimget(options,'OutputFcn',defaultopt,'fast');
74  if isempty(outputfcn)
75      haveoutputfcn = false;
76  else
77      haveoutputfcn = true;
78      xOutputfcn = x; % Last x passed to outputfcn; has the input x's shape
79      % Convert to function handle as needed.
80      outputfcn = fcnchk(outputfcn,length(varargin));
```

```
81 end
82
83 header = ' Iteration   Func-count     min f(x)     min g(x)     Procedure';
84
85 % Convert to function handle as needed.
86 funfcn = fcnchk(funfcn,length(varargin));
87 % Add a wrapper function to check for Inf/NaN/complex values
88 if funValCheck
89     % Add a wrapper function, CHECKFUN, to check for NaN/complex values
90     % without having to change the calls that look like this:
91     % f = funfcn(x,varargin{:});
92     % x is the first argument to CHECKFUN, then the user's function,
93     % then the elements of varargin. To accomplish this we need to add the
94     % user's function to the beginning of varargin, and change funfcn to be
95     % CHECKFUN.
96     varargin = {funfcn, varargin{:}};
97     funfcn = @checkfun;
98 end
99
100 n = numel(x);
101
102 % Initialize parameters
103 rho = 1; chi = 2; psi = 0.5; sigma = 0.5;
104 onesn = ones(1,n);
105 two2np1 = 2:n+1;
106 one2n = 1:n;
107
108 % Set up a simplex near the initial guess.
109 xin = x(:); % Force xin to be a column vector
110 v = zeros(n,n+1); fv = zeros(1,n+1);
111 v(:,1) = xin;    % Place input guess in the simplex!
112 x(:) = xin;    % Change x to the form expected by funfcn
113 [fv(:,1),gv(:,1)] = funfcn(x,varargin{:});
114 func_evals = 1;
115 itercount = 0;
116 how = '';
117
118 % Initialize the output function.
119 if haveoutputfcn
```

```
120     [xOutputfcn, optimValues, stop] = callOutputFcn(outputfcn,x,...
121         xOutputfcn,'init',itercount, ...
122         func_evals, how, fv(:,1), gv(:,1),varargin{:});
123     if stop
124         [x,fval,exitflag,output] =cleanUpInterrupt(xOutputfcn,optimValues);
125         if  prnt > 0
126             disp(output.message)
127         end
128         return;
129     end
130 end
131 
132 % Print out initial f(x) as 0th iteration
133 if prnt == 3
134     disp(' ')
135     disp(header)
136     disp(sprintf(' %5.0f        %5.0f     %12.6g     %6.6g          %s', ...
137         itercount, func_evals, fv(1), gv(1), how));
138 elseif prnt == 4
139     clc
140     formatsave = get(0,{'format','formatspacing'});
141     format compact
142     format short e
143     disp(' ')
144     disp(how)
145     v
146     fv
147     func_evals
148 end
149 % OutputFcn call
150 if haveoutputfcn
151     [xOutputfcn, optimValues, stop] = callOutputFcn(outputfcn,x,...
152         xOutputfcn,'iter',itercount, ...
153         func_evals, how, fv(:,1), gv(:,1),varargin{:});
154     if stop  % Stop per user request.
155         [x,fval,exitflag,output] =cleanUpInterrupt(xOutputfcn,optimValues);
156         if  prnt > 0
157             disp(output.message)
158         end
```

```
159         return;
160     end
161 end
162 
163 usual_delta = 0.05;             % 5 percent deltas for non-zero terms
164 zero_term_delta = 0.00025;      % Even smaller delta for zero elements of x
165 for j = 1:n
166     y = xin;
167     if y(j) ~= 0
168         y(j) = (1 + usual_delta)*y(j);
169     else
170         y(j) = zero_term_delta;
171     end
172     v(:,j+1) = y;
173     x(:) = y; [f,g] = funfcn(x,varargin{:});
174     fv(1,j+1) = f;
175     gv(1,j+1) = g;
176 end
177 
178 % sort so v(1,:) has the lowest function value
179 [fv,j] = sort(fv);
180 v = v(:,j);
181 gv = gv(:,j);
182 
183 how = ' initial simplex';
184 itercount = itercount + 1;
185 func_evals = n+1;
186 if prnt == 3
187     disp(sprintf(' %5.0f        %5.0f     %12.6g     %12.6g     %s', ...
188         itercount, func_evals, fv(1), how))
189 elseif prnt == 4
190     disp(' ')
191     disp(how)
192     v
193     fv
194     func_evals
195 end
196 % OutputFcn call
197 if haveoutputfcn
```

```
198     [xOutputfcn, optimValues, stop] = callOutputFcn(outputfcn,...
199         x,xOutputfcn,'iter',itercount, ...
200         func_evals, how, fv(:,1), gv(:,1),varargin{:});
201     if stop  % Stop per user request.
202         [x,fval,exitflag,output] =cleanUpInterrupt(xOutputfcn,optimValues);
203         if  prnt > 0
204             disp(output.message)
205         end
206         return;
207     end
208 end
209 exitflag = 1;
210 
211 % Main algorithm
212 % Iterate until the diameter of the simplex is less than tolx
213 %   AND the function values differ from the min by less than tolf,
214 %   or the max function evaluations are exceeded. (Cannot use OR instead of
215 %   AND.)
216 while func_evals < maxfun && itercount < maxiter
217     if max(max(abs(v(:,two2np1)-v(:,onesn)))) <= tolx && ...
218             max(abs(fv(1)-fv(two2np1))) <= tolf
219         break
220     end
221 
222     v_old  = v;  % Old simplex points are stored
223     fv_old = fv; % Old functions are stored
224     gv_old = gv; % Old constraints are stored
225 
226     % Alternative reflection points
227     for k = 1:n
228         if n > 1 & k~=n & k < n+1
229             xbar = sum([v(:,1:k) v(:,k+2:n+1)],2)/...
230                 sum([length(1:k) length(k+2:n+1)]);
231             xr = (1 + rho)*xbar - rho*v(:,k+1);
232         elseif k == n % if n > 1
233             % classic xbar = average of the n (NOT n+1) best points
234             xbar = sum(v(:,one2n),2)/n;
235             xr = (1 + rho)*xbar - rho*v(:,end);
236         end  % if n > 1
```

```
237
238         % Basic check to if extra loops are worth computation
239         % Short circuit or makes sure all directions are explored if
240         % constraints are violated
241         if mean([fv(1:k) fv(k+2:n+1)]) < fv(k+1) || max(gv) > 0
242             x(:) = xr; [fxr,gxr] = funfcn(x,varargin{:});
243             func_evals = func_evals+1;
244         else
245             continue
246         end
247
248         if k ~= n
249             if fxr < fv(:,1) & gxr < 0 || gxr < gv(:,1) & max(gv) > 0
250                 % Calculate the expansion point
251                 xe = (1 + rho*chi)*xbar - rho*chi*v(:,end);
252                 x(:) = xe; [fxe,gxe] = funfcn(x,varargin{:});
253                 func_evals = func_evals+1;
254                 if fxe < fxr & gxe < 0 || gxe < gxr & max(gv) > 0
255                     v(:,end) = xe;
256                     fv(:,end) = fxe;
257                     gv(:,end) = gxe;
258                     how = 'expand_side';
259                 else
260                     v(:,end) = xr;
261                     fv(:,end) = fxr;
262                     gv(:,end) = gxr;
263                     how = 'reflect_side';
264                 end
265             else
266                 v  = v_old;  % Old simplex points are RE-stored
267                 fv = fv_old; % Old functions are RE-stored
268                 gv = gv_old; % Old constraints are RE-stored
269                 %disp('Alt step, NO expand/step')
270                 continue
271             end % if fxr < fv(:,1)
272         end % if k ~= n
273
274         if k == n
275             if fxr < fv(:,1) & gxr < 0 || gxr < gv(:,1) & max(gv) > 0
```

```
276                 % Calculate the expansion point
277                 xe = (1 + rho*chi)*xbar - rho*chi*v(:,end);
278                 x(:) = xe; [fxe,gxe] = funfcn(x,varargin{:});
279                 func_evals = func_evals+1;
280                 if fxe < fxr & gxe < 0 || gxe < gxr & max(gv) > 0
281                     v(:,end) = xe;
282                     fv(:,end) = fxe;
283                     gv(:,end) = gxe;
284                     how = 'expand';
285                 else
286                     v(:,end) = xr;
287                     fv(:,end) = fxr;
288                     gv(:,end) = gxr;
289                     how = 'reflect';
290                 end
291             else % fv(:,1) <= fxr
292                 if fxr < fv(:,n) & gxr < 0 || gxr < gv(:,n) & max(gv) > 0
293                     v(:,end) = xr;
294                     fv(:,end) = fxr;
295                     gv(:,end) = gxr;
296                     how = 'reflect';
297                 else % fxr >= fv(:,n)
298                     % Perform contraction
299                     if fxr < fv(:,end) & gxr < 0 || gxr < gv(:,end) ...
300                             & max(gv) > 0
301                         % Perform an outside contraction
302                         xc = (1 + psi*rho)*xbar - psi*rho*v(:,end);
303                         x(:) = xc; [fxc,gxc] = funfcn(x,varargin{:});
304                         func_evals = func_evals+1;
305 
306                         if fxc <= fxr & gxc < 0 || gxc < gxr & max(gv) > 0
307                             v(:,end) = xc;
308                             fv(:,end) = fxc;
309                             gv(:,end) = gxc;
310                             how = 'contract outside';
311                         else
312                             % perform a shrink
313                             how = 'shrink';
314                         end
```

```
315                     else
316                         % Perform an inside contraction
317                         xcc = (1-psi)*xbar + psi*v(:,end);
318                         x(:) = xcc; [fxcc,gxcc] = funfcn(x,varargin{:});
319                         func_evals = func_evals+1;
320 
321                         if fxcc < fv(:,end) & gxcc < 0 ||...
322                                 gxcc < gv(:,end) & max(gv) > 0
323                             v(:,end) = xcc;
324                             fv(:,end) = fxcc;
325                             gv(:,end) = gxcc;
326                             how = 'contract inside';
327                         else
328                             % perform a shrink
329                             how = 'shrink';
330                         end
331                     end
332                     if strcmp(how,'shrink')
333                         for j=two2np1
334                             v(:,j)=v(:,1)+sigma*(v(:,j) - v(:,1));
335                             x(:) = v(:,j); [fv(:,j),gv(:,j)] = ...
336                                 funfcn(x,varargin{:});
337                         end
338                         func_evals = func_evals + n;
339                     end
340                 end
341             end
342         end % if k == n
343 
344         if max(gv) > 0 % Sorted to constraints
345             [gv,j] = sort(gv);
346             v = v(:,j);
347             fv = fv(:,j);
348         else % Normal sort
349             [fv,j] = sort(fv);
350             v = v(:,j);
351             gv = gv(:,j);
352         end
353 
```

```
354         itercount = itercount + 1;
355         if prnt == 3
356     disp(sprintf(' %5.0f        %5.0f     %12.6g     %12.6g          %s',...
357                 itercount, func_evals, fv(1), gv(1), how))
358         elseif prnt == 4
359             disp(' ')
360             disp(how)
361             v
362             fv
363             gv
364             func_evals
365         end
366         % OutputFcn call
367         if haveoutputfcn
368             [xOutputfcn, optimValues, stop] = callOutputFcn(outputfcn,...
369                 x,xOutputfcn,'iter',itercount, ...
370                 func_evals, how, fv(:,1), gv(:,1),varargin{:});
371             if stop  % Stop per user request.
372                 [x,fval,exitflag,output] = cleanUpInterrupt(xOutputfcn,...
373                     optimValues);
374             if  prnt > 0
375                 disp(output.message)
376             end
377             return;
378             end
379         end
380
381         if k ~= n
382             %disp('Alt step, expand/step')
383             break
384         end
385     end % for k = 1:n
386 end   % while
387
388 x(:) = v(:,1);
389 if prnt == 4,
390     % reset format
391     set(0,{'format','formatspacing'},formatsave);
392 end
```

```
393 output.iterations = itercount;
394 output.funcCount = func_evals;
395 output.algorithm = 'Nelder-Mead simplex direct search';
396
397 fval = min(fv);
398
399 % OutputFcn call
400 if haveoutputfcn
401     callOutputFcn(outputfcn,x,xOutputfcn,'done',itercount,...
402         func_evals, how, f, varargin{:});
403 end
404
405 if func_evals >= maxfun
406     msg = sprintf(['Exiting: Maximum number of function evaluations'...
407                    'has been exceeded\n' ...
408                    '         - increase MaxFunEvals option.\n' ...
409                    '         Current function value: %f \n'], fval);
410     if prnt > 0
411         disp(' ')
412         disp(msg)
413     end
414     exitflag = 0;
415 elseif itercount >= maxiter
416     msg = sprintf(['Exiting: Maximum number of iterations has been '...
417                    'exceeded\n' ...
418                    '         - increase MaxIter option.\n' ...
419                    '         Current function value: %f \n'], fval);
420     if prnt > 0
421         disp(' ')
422         disp(msg)
423     end
424     exitflag = 0;
425 else
426     msg = ...
427       sprintf(['Optimization terminated:\n', ...
428                ' the current x satisfies the termination criteria'...
429                'using OPTIONS.TolX of %e \n' ...
430                ' and F(X) satisfies the convergence criteria'...
431                'using OPTIONS.TolFun of %e \n'], ...
```

```
                tolx, tolf);
    if prnt > 1
        disp(' ')
        disp(msg)
    end
    exitflag = 1;
end

output.message = msg;

%--------------------------------------------------------------------------
function [xOutputfcn, optimValues, stop] = callOutputFcn(outputfcn,x,...
    xOutputfcn,state,iter,numf,how,f,varargin)
% CALLOUTPUTFCN assigns values to the struct OptimValues and then calls the
% outputfcn.
%
% state - can have the values 'init','iter', or 'done'.
% We do not handle the case 'interrupt' because we do not want to update
% xOutputfcn or optimValues (since the values could be inconsistent) before
% calling the outputfcn; in that case the outputfcn is called directly
% rather than calling it inside callOutputFcn.

% For the 'done' state we do not check the value of 'stop' because the
% optimization is already done.
optimValues.iteration = iter;
optimValues.funccount = numf;
optimValues.fval = f;
optimValues.procedure = how;

xOutputfcn(:) = x;  % Set x to have user expected size
switch state
    case {'iter','init'}
        stop = outputfcn(xOutputfcn,optimValues,state,varargin{:});
    case 'done'
        stop = false;
        outputfcn(xOutputfcn,optimValues,state,varargin{:});
    otherwise
        error('MATLAB:fminsearch:InvalidState', ...
            'Unknown state in CALLOUTPUTFCN.')
```

```
471 end
472
473 %-------------------------------------------------------------------------
474 function [x,FVAL,EXITFLAG,OUTPUT] =cleanUpInterrupt(xOutputfcn,optimValues)
475 % CLEANUPINTERRUPT updates or sets all the output arguments of FMINBND
476 % when the optimization is interrupted.
477
478 x = xOutputfcn;
479 FVAL = optimValues.fval;
480 EXITFLAG = -1;
481 OUTPUT.iterations = optimValues.iteration;
482 OUTPUT.funcCount = optimValues.funccount;
483 OUTPUT.algorithm = 'golden section search, parabolic interpolation';
484 OUTPUT.message = 'Optimization terminated prematurely by user.';
485
486 %-------------------------------------------------------------------------
487 function f = checkfun(x,userfcn,varargin)
488 % CHECKFUN checks for complex or NaN results from userfcn.
489
490 f = userfcn(x,varargin{:});
491 % Note: we do not check for Inf as FMINSEARCH handles it naturally.
492 if isnan(f)
493     error('MATLAB:fminsearch:checkfun:NaNFval', ...
494         'User function ''%s'' returned NaN when evaluated at %g;\n'...
495         'FMINSEARCH cannot continue.', ...
496         localChar(userfcn), x);
497 elseif ~isreal(f)
498     error('MATLAB:fminsearch:checkfun:ComplexFval', ...
499         'User function ''%s'' returned a complex value when evaluated'...
500         'at %g;\n FMINSEARCH cannot continue.', ...
501         localChar(userfcn),x);
502 end
503
504 %-------------------------------------------------------------------------
505 function strfcn = localChar(fcn)
506 % Convert the fcn to a string for printing
507
508 if ischar(fcn)
509     strfcn = fcn;
```

```
elseif isa(fcn,'inline')
    strfcn = char(fcn);
elseif isa(fcn,'function_handle')
    strfcn = func2str(fcn);
else
    try
        strfcn = char(fcn);
    catch
        strfcn = '(name not printable)';
    end
end
```

A.8 NACA_4_mid_slope

NACA_4_mid_slope.m is a MATLAB function that determines the polynomial coefficients for a modified NACA 4-series airfoil.

```
% m-function file to generate the leading and trailing edge constants for
% NACA 4 and 5 digit modified series.
%
% Where leadthick  = leading edge polynomial equation
%       trailthick = trailing edge polynomial equation
%
%       thickness  = maximum thickness as fraction of chord
%       m_thick    = position of maximum thickness
%       I          = nose radius constant -6 indicates same as normal
%                    NACA profile

function [leadthick,trailthick] = NACA_4_mod(thickness,m_thick,I_nose,...
    edge_fract,trail_slope,mid_slope)

% The trailing edge is calculated first
LHS      = zeros(4);
LHS(1,1) = 1;
LHS(2,2) = -1;
LHS(3,:) = [0          -1 -2*(1-m_thick) -3*((1-m_thick)^2)];
LHS(4,:) = [1 (1-m_thick)  (1-m_thick)^2      (1-m_thick)^3];

```

```
%Calculates the slope of the trailing edge
if isempty(trail_slope)
    DY_DX         = [0.858838 1 1.17 1.575 2.325 3.5]';
    m_thick_table = [.1 .2 .3 .4 .5 .6]';
else
    DY_DX         = [trail_slope trail_slope trail_slope ...
                     trail_slope trail_slope trail_slope]';
    m_thick_table = [0 .1 .3 .5 .7 .9]';
end


%Calculates the trailing edge thickness polynomials
RHS    = zeros(4,1);
RHS(1) = edge_fract*thickness;
RHS(2) = -thickness*interp1(m_thick_table,DY_DX,m_thick,'pchip');
RHS(3) = mid_slope;
RHS(4) = thickness/2;
trailthick = (LHS\RHS)';


% Leading edge is dependant on trailing edge
LHS      = zeros(4);
LHS(1,1) = 1;
LHS(2,:) = [sqrt(m_thick)     m_thick m_thick^2   m_thick^3];
LHS(3,:) = [0.5/sqrt(m_thick)       1 2*m_thick 3*m_thick^2];
LHS(4,:) = [-0.25/(m_thick^(3/2))   0         2   6*m_thick];

% Determines the leading edge trickness polynomial
RHS    = zeros(4,1);
RHS(1) = sqrt(2*1.1019*((thickness*I_nose/6)^2));
RHS(2) = thickness/2;
RHS(3) = mid_slope;
RHS(4) = 2*trailthick(3) + 6*trailthick(4)*(1-m_thick);

leadthick = (LHS\RHS)';

% Trailing edge equation contains only polynomial parts.
% The constant required by trail thickness is added in aerogrid
trailthick = ([0 trailthick(2:4)]);
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B:
CFX Report

B.1 Example CFX Report

This is an example report from ANSYS CFX for the fluid simulations performed in this thesis. The report includes information on the mesh settings, the fluid domain, boundary settings, and more.

Date

2022/05/18 12:34:17

Contents

1. File Report
   Table 1 File Information for P Beta 20 Turbulent 45 40 55
2. Mesh Report
   Table 2 Mesh Information for P Beta 20 Turbulent 45 40 55
3. Physics Report
   Table 3 Domain Physics for P Beta 20 Turbulent 45 40 55
   Table 4 Boundary Physics for P Beta 20 Turbulent 45 40 55
4. Solution Report
   Table 5 Boundary Flows for P Beta 20 Turbulent 45 40 55
   Table 6 Forces and Torques for P Beta 20 Turbulent 45 40 55
5. User Data
   Chart 1

1. File Report

Table 1. File Information for P Beta 20 Turbulent 45 40 55

Case	P Beta 20 Turbulent 45 40 55
File Path	C:\Users\mpinn\Documents\Thesis\Cam_opt\CFX_Cascade\CFX_Cascade_files\dp0\CFX-17\CFX\Fluid Flow CFX_002.res
File Date	17 May 2022
File Time	10:46:49 PM
File Type	CFX5
File Versio	20.2

n	

2. Mesh Report

Table 2. Mesh Information for P Beta 20 Turbulent 45 40 55

Domain	Nodes	Elements
Default Domain	653728	325275

3. Physics Report

Table 3. Domain Physics for P Beta 20 Turbulent 45 40 55

Domain - Default Domain	
Type	Fluid
Location	B277
Materials	
Air at 25 C	
Fluid Definition	Material Library
Morphology	Continuous Fluid
Settings	
Buoyancy Model	Non Buoyant
Domain Motion	Stationary
Reference Pressure	1.0000e+0 [atm]
Heat Transfer Model	Total Energy
Include Viscous Work Term	True
Turbulence Model	SST
Transitional Turbulence	Gamma Theta Model
Transition Onset Correlation	Langtry Menter
Turbulent Wall Functions	Automatic
High Speed Model	Off
Domain Interface - Domain Interface 1	
Boundary List1	Domain Interface 1 Side 1
Boundary List2	Domain Interface 1 Side 2

Interface Type	Fluid Fluid
Settings	
Interface Models	Translational Periodicity
Mass And Momentum	Conservative Interface Flux
Mesh Connection	Direct

Table 4. Boundary Physics for P Beta 20 Turbulent 45 40 55

Domain	**Boundaries**	
Default Domain	**Boundary - Inlet**	
	Type	INLET
	Location	Inlet
	Settings	
	Flow Regime	Subsonic
	Heat Transfer	Static Temperature
	Static Temperature	2.5000e+1 [C]
	Mass And Momentum	Cartesian Velocity Components
	U	UInletVelocity
	V	VInletVelocity
	W	0.0000e+0 [m s^-1]
	Turbulence	Medium Intensity and Eddy Viscosity Ratio
	Boundary - Domain Interface 1 Side 1	
	Type	INTERFACE
	Location	TopWall
	Settings	
	Heat Transfer	Conservative Interface Flux
	Mass And Momentum	Conservative Interface Flux
	Turbulence	Conservative Interface Flux
	Boundary - Domain Interface 1 Side 2	
	Type	INTERFACE
	Location	BottomWall
	Settings	
	Heat Transfer	Conservative Interface Flux
	Mass And Momentum	Conservative Interface Flux

	Turbulence	Conservative Interface Flux
	Boundary - Outlet	
	Type	OUTLET
	Location	Outlet
	Settings	
	Flow Regime	Subsonic
	Mass And Momentum	Average Static Pressure
	Pressure Profile Blend	5.0000e-2
	Relative Pressure	0.0000e+0 [Pa]
	Pressure Averaging	Average Over Whole Outlet
	Boundary - SideWalls	
	Type	SYMMETRY
	Location	SideWalls
	Settings	
	Boundary - Blade	
	Type	WALL
	Location	Blade
	Settings	
	Heat Transfer	Adiabatic
	Mass And Momentum	No Slip Wall
	Wall Roughness	Smooth Wall

4. Solution Report

Table 5. Boundary Flows for P Beta 20 Turbulent 45 40 55

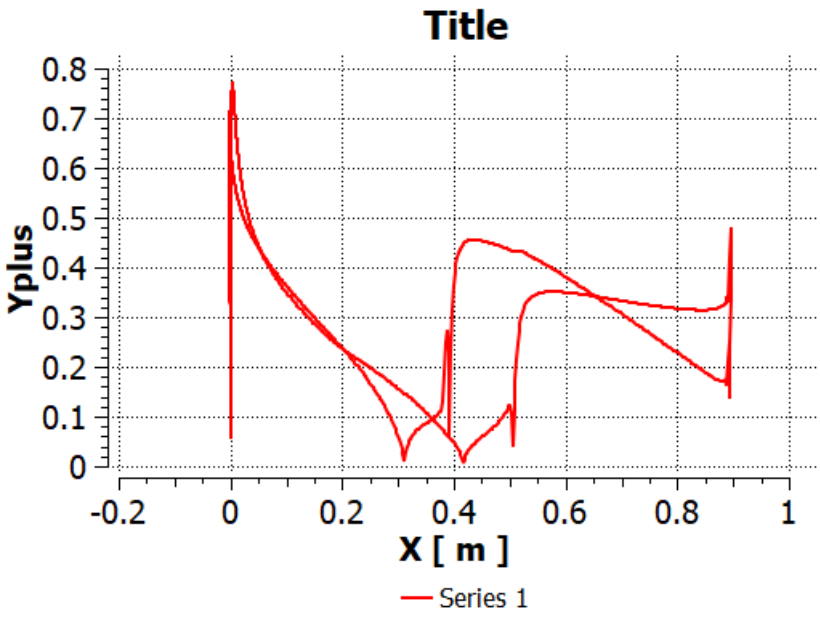
Location	Type	Mass Flow	Momentum		
			X	Y	Z
Blade	Boundary	0.0000e+0	6.4458e-1	-9.6476e-1	-9.8318e-14
Inlet	Boundary	1.4945e-1	9.4366e-1	1.5851e+0	-6.7688e-17
Outlet	Boundary	-1.4945e-1	-1.5881e+0	-6.2034e-1	2.3405e-13
SideWalls	Boundary	0.0000e+0	5.0113e-17	-1.6354e-16	-9.1987e-11

Table 6. Forces and Torques for P Beta 20 Turbulent 45 40 55

Location	Type	X	Y	Z
Blade	Pressure Force	-6.5436e-1	9.5897e-1	9.8317e-14
	Viscous Force	9.7756e-3	5.7937e-3	3.6702e-19
	Total Force	-6.4458e-1	9.6476e-1	9.8318e-14
	Pressure Torque	1.2233e-11	-2.1124e-11	4.5163e-1
	Viscous Torque	2.1529e-12	-2.4783e-12	-9.3549e-4
	Total Torque	1.4386e-11	-2.3603e-11	4.5069e-1

5. User Data

Chart 1.



THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] R. I. Lewis, *Vortex Element Methods for Fluid Dynamic Analysis of Engineering Systems*. Cambridge, England: Cambridge University Press, 1991.

[2] J. Grimson, *Mechanics and Thermodynamics of Fluids*. London, England: McGraw-Hill, 1970.

[3] W. Du and S. Kinnas, "Coupling a numerical optimization technique with a panel method or a vortex lattice method to design cavitating propellers in non-uniform inflows," presented at the 10th International Symposium on Cavitation, Baltimore, MD, USA, 2018.

[4] M. Parenteau, E. Laurendeau, and G. Carrier, "Combined high-speed and high-lift wing aerodynamic optimization using a coupled VLM-2.5D RANS approach," *Aerospace Science and Technology*, vol. 76, pp. 484–496, may 2018.

[5] S. Drayton, "Design, test, and evaluation of a transonic axial compressor rotor with splitter blades," Ph.D. dissertation, Dept. of Mech. and Aero. Eng., Naval Postgraduate School, Monterey, CA, USA, 2013.

[6] N. L. Sanger, "Design methodology for the NPS transonic compressor," Turbopropulsion Laboratory, Naval Postgraduate School, Monterey, CA, USA, Rep. 99-01, 1999.

[7] M. Brantner, "Optimization routine for compressor design using commercial software," M.S. thesis, Dept. of Mech. and Aero. Eng., Naval Postgraduate School, Monterey, CA, USA, 2019 [Online]. Available: http://hdl.handle.net/10945/62852

[8] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[9] A. J. Gannon, private communication, Jan. 2022.

[10] I. H. Abbott and A. E. von Doenhoff, *Theory of Wing Sections: Including a Summary of Airfoil Data*. New York, USA: Dover Publications, 1959.

[11] S. R. Johnson, "Computational aerodynamic analysis of airfoils for energy-producing sailing ships," M.S. thesis, Dept. of Mech. and Aero. Eng., Naval Postgraduate School, Monterey, CA, USA, 2020 [Online]. Available: http://hdl.handle.net/10945/65559

[12] F. M. White, *Fluid Mechanics*. Boston, MA, USA: McGraw-Hill, 2003.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California