ABSTRACT

Title of thesis: OPTIMIZATION OF EXPANDING TURNING VANES BY BEZIER CURVE PARAMETERIZATION
Collin Schirf, Master of Science, 2019

Thesis directed by: Dr. Jewel Barlow
A James Clark Clark School of Engineering
Department of Aerospace Engineering

The development of a new process for optimizing wind tunnel turning vanes for use in expanding corners is described. This process uses MATLAB tools to operate the infinite airfoil cascade solver MISES in order to take advantage of the powerful optimization tools already present in MATLAB. Airfoils are defined using four Bezier curves of fifth order to limit the number of design variables and take advantage of simple smoothness constraints. A parameter sweep is performed to verify the tool's operation and gain insight into the impacts of airfoil thickness, airfoil camber, cascade solidity, and expansion ratio before several optimization cases using various MATLAB optimization functions were used to show the ability of the optimizer to reduce total pressure loss and flow separation in turning vane cascades. Optimizer outputs were shown to reduce total pressure losses by up to 18% and separation magnitude by up to 53% over initial designs. Comparison with STAR-CCM+ models verified applicability of MISES cases to more accurate wind tunnel flows.

OPTIMIZATION OF EXPANDING TURNING VANES BY BEZIER CURVE PARAMETERIZATION

by

Collin J. Schirf

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2019

Advisory Committee:
Dr. Jewel Barlow, Chair/Advisor
Dr. James Baeder
Dr. Roberto Celi

© Copyright by
Collin J. Schirf
2019

Acknowledgments

I would like to acknowledge all those who helped me to complete this project. Firstly, I would like to thank my advisor Dr. Jewel Barlow for his suggestion of this topic and encouragement to pursue a graduate degree. Second, I would like to thank Dr. James Baeder and Dr. Roberto Celi for agreeing to serve on my committee and provide feedback. Third, I would like to thank Dr. Mark Drela for his permission to use and assistance with the MISES software. Fourth I would like to thank all the personnel at the Glenn L. Martin Wind Tunnel who assisted me at many points of this project.

Beyond those directly involved with the project, I would like to thank the SMART Scholarship program for providing me with the means to pursue this degree. I would also like to thank my coworkers at the Firing Tables and Ballistics Division whose advice on work-related matters ended up being crucial to my completion of this project.

Finally, I would like to thank my parents, my brother, and all my other friends who supported me during my schooling and gave me wonderful advice and encouragement. I would especially like to thank Madeleine Dwyer for proofreading, moral support, distraction, and care packages along the way. I would also like to give special thanks Shayne Suban for introducing me to Dr. Barlow many years ago–sending me down the path to reach this point.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CAD	Computer Aided Design
CFD	Computational Fluid Dynamics
ER	Expansion Ratio
GA	Genetic Algorithm
GLMWT	Glenn L. Martin Wind Tunnel
M.I.T.	Massachusets Institute of Technology
NACA	National Advisory Committee for Aeronautics
Pa	Pascals
LE	Leading Edge
STAR	STAR-CCM+
TE	Trailing Edge

Chapter 1: Introduction

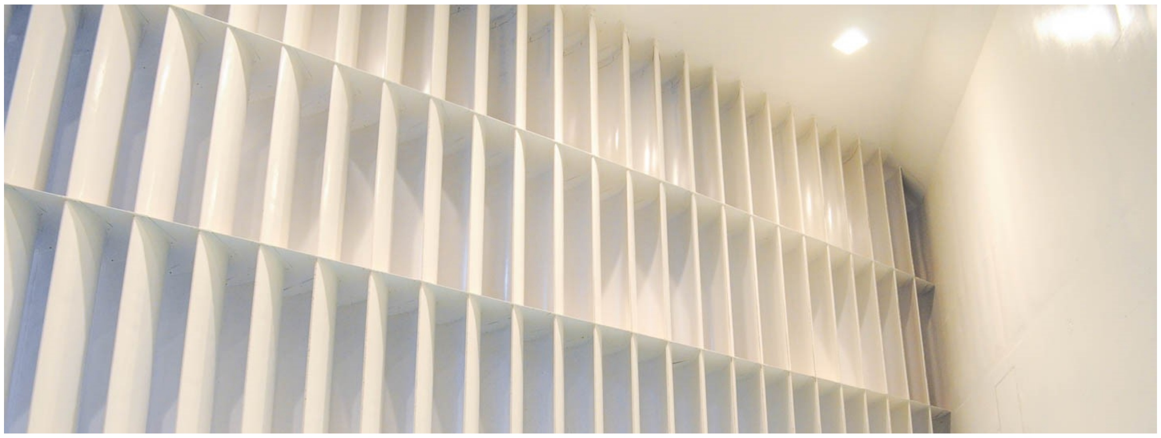


Figure 1.1: Turning vanes in the Glenn L. Martin Wind Tunnel [1]

When designing a wind tunnel for aerodynamic research, there are a near infinite number of decisions to be made. The choice of returning or non-returning tunnel is one of the most important of these decisions. Closed return wind tunnels have lower power requirements than non-returning tunnels as the flow must only be kept moving by overcoming losses rather than accelerated from stationary. These benefits, however, come at the cost of significantly increased complexity. Much of this complexity is related to the need to maintain flow uniformity around corners.

In order to preserve uniform flow and prevent losses in corners, it is necessary to create turning vanes for each corner. Turning vanes are cascades of airfoils designed to turn the flow in sections and minimize energy losses. The design of these airfoils

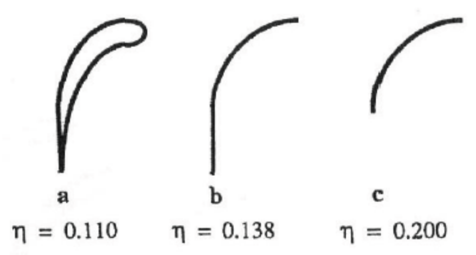


Figure 1.2: Turning vane designs and associated loss values [2]

can vary from a simple circular arc to a complex highly cambered airfoil. As with most aerospace applications, a carefully considered design can result in improved performance. The book "Low-Speed Wind Tunnel Testing" provides data for such an example where a simple bent plate generates a loss nearly double that of a specially designed high camber airfoil [2]. For a small amateur tunnel, the larger loss may be an acceptable alternative to a lengthy design process, but for a dedicated research tunnel, this loss represents significant increases to operating costs.

Thus, the development of tools for the design of turning vanes is useful to any who desire to create their own tunnel or improve an existing tunnel. Furthermore, creating tools which are simple to use could improve the ability of those with less experience and resources to create effective and efficient closed-return tunnels.

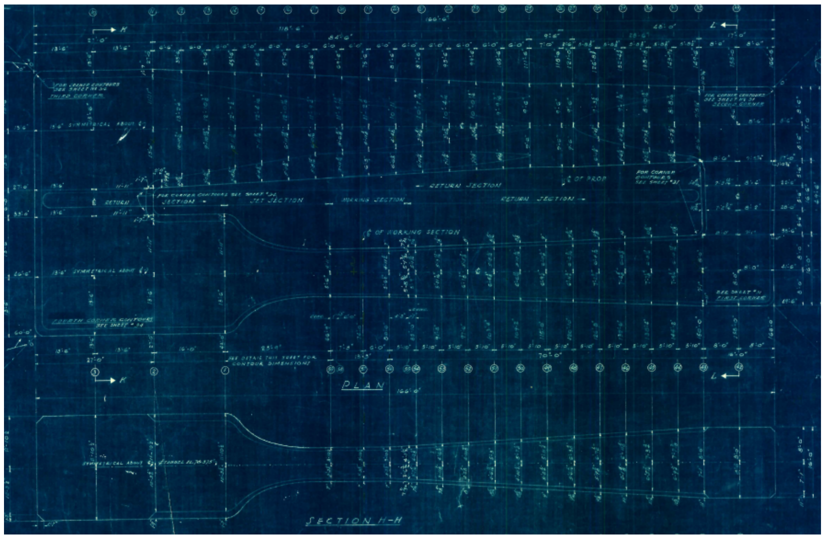


Figure 1.3: Blueprints of the GLMWT circuit

1.1 Background and Motivation

The Glenn L. Martin Wind Tunnel (GLMWT) was constructed in 1949 as a gift to the University of Maryland. The design of the GLMWT was tailored largely to facilitate aircraft testing [1]. Since its completion, though, the facility has been used for a variety of applications beyond aircraft. Notably, the GLMWT currently does extensive automotive testing. To better characterize flows in the wake of these automobiles and other bluff body type flows, the director of the wind tunnel desired to extend the test section.

Given that the GLMWT stands on the campus of the University of Maryland, College Park and is surrounded by other buildings, an extension of the building to accommodate a longer test section would not be possible. Instead, it was decided that replacing the first corner with an expanding corner could be investigated. This alteration would allow the first diffuser to be shortened and the test section to be lengthened without alterations to other portions of the tunnel. Even a modest expansion ratio would allow significant lengthening of the test section. Basic calculations using the current diffuser's area ratios show that an expansion ratio of 1.2 in the corner would allow about 17 feet to be added to the test section . Figure 1.4 demonstrates this alteration graphically.

Unfortunately, the design of turning vanes for use in expanding corners is not an extensively studied or documented field. For this reason, the research presented in this thesis was necessary to bridge the gap and assist in the investigation of a potential redesign.

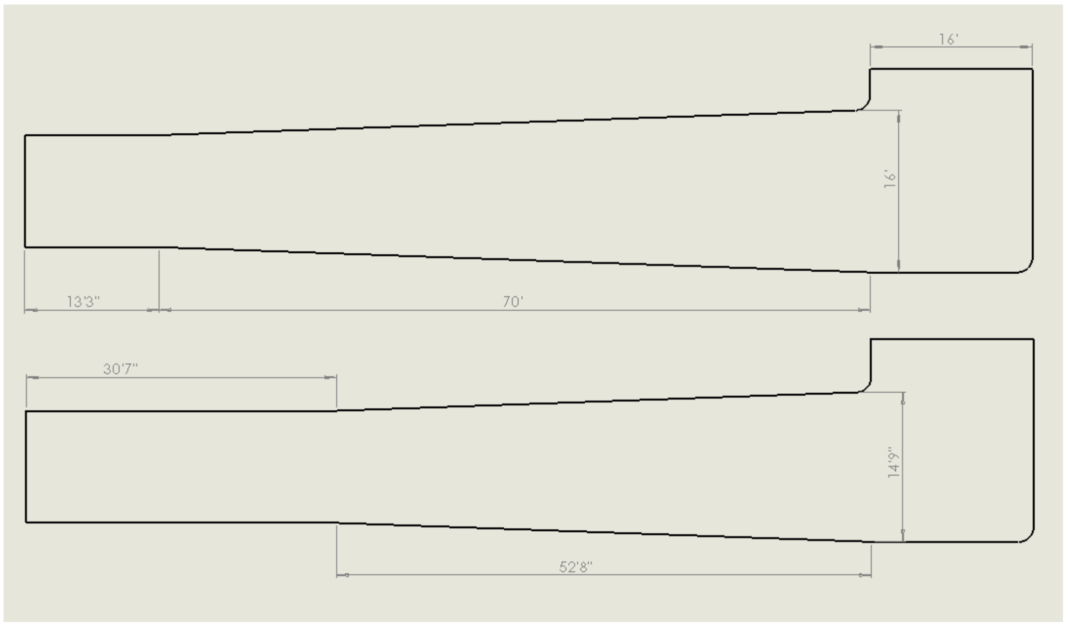


Figure 1.4: Dimensioned sketch illustrating possible test section expansion.

1.2 Prior Work

Though expanding cascades have not been extensively studied, a few papers have been written on designs for use in non-expanded corners. This research has also been supplemented by research in separate areas with results applicable to turning vanes such as the design of turbomachinery and general flow simulation though Computational Fluid Dynamics (CFD).

1.2.1 Theoretical treatment of flow about airfoil cascades

In 1944, the National Advisory Committee for Aeronautics (NACA) published research describing the use of potential flow theory to predict the flow about a lattice of airfoils with the aid of conformal mapping. This method allowed the pressure distributions about the airfoils to be calculated for incompressible, irrotational flows

[3]. Originally intended for application to any infinite cascade of airfoils, this work is cited in later works dealing with the optimization of vanes for use in turbomachinery such as the paper "Analysis of Transonic Cascade Flow Using Conformal Mapping and Relaxation Techniques" from 1977 [4]. A 1947 paper by Spurr extended the capabilities of this type of analysis with the use of thin airfoil theory. In this work, a method for calculating the cascade properties of an airfoil are related to the properties of a lone airfoil [5]. Both the 1944 and 1947 works were primarily concerned with the pressure distribution along the airfoil in these cascades rather than the flow properties behind the cascade.

In the 1970's, the increasing computational power and refinement of computational methods led to the development of CFD solvers tailored to airfoil cascades. The development of one such solver is described in the paper "A New Approach in Cascade Flow Analysis Using the Finite Element Method." These methods eschewed the explicit use of conformal mapping for the solution of the potential flow about airfoil cascades. Instead the author used a finite element method more common in modern CFD [6]. With this type of solver, a more accurate picture of the flow properties ahead of and behind turning vane cascades could be generated.

In the late 2000's, Dr. Mark Drela of M.I.T. developed the MISES flow solver for use in turbomachinery applications. This flow solver utilizes a Newton method to solve for the flow about a single airfoil and uses periodic boundary conditions to extend this flow to an infinite cascade. Included in the solver are tools for grid generation, flow property specifications, flow visualization, and optimization tools. This suite has tremendous capabilities and the specialization in infinite cascades allows the options to be tailored to suit the needs of those designing for a number of applications [7].

1.2.2 General turning vane design

As previously discussed, there is some difficulty in finding research directly related to turning vane design. Much of this design work was performed as a matter of necessity based on the aerodynamic research available at the time and was not documented publicly. Despite this, some design processes can be seen in the paper "Wind Tunnel Turning Vanes of Modern Design" published by NASA in 1986. In this paper, an inverse technique was used to generate an airfoil to create a desired pressure distribution whose performance was verified with an inviscid CFD code. Using this process, a modest improvement in corner loss was achieved over a previously-used arc-shaped vane [8].

More modern design procedures can be found in a paper from the Polytechnic University of Madrid. This paper is notable as it describes a somewhat simpler optimization process using the tools provided by MISES. To facilitate easy construction, the shape of the airfoil was assumed to be defined by a quarter circle on the lower surface and a parabolic arc on the upper surface. Based on this assumption, parameter sweeps were performed on the maximum thickness and leading edge radius to find the design with the lowest pressure loss across the corner. Following this, MISES was used to manually optimize the vanes. Also described in the article is the use of MATLAB to set up required files quickly and accurately [9].

1.2.3 Expanding turning vane design

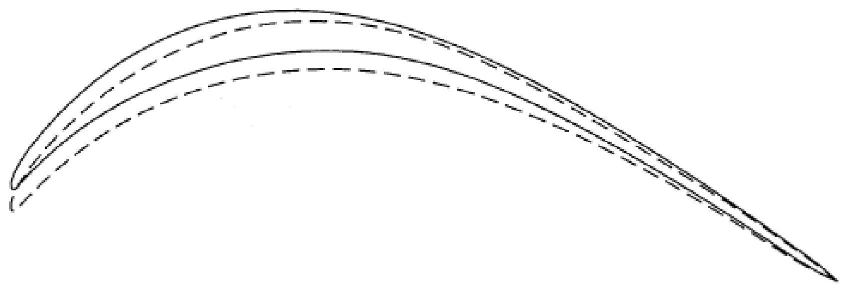


Figure 1.5: Original (solid line) and optimized (dashed line) vanes from Ref. [10]

The lack of solid information about designing turning vanes for expanding flows was breached somewhat by a paper from the Royal Institute of Technology in Sweden. This paper by Björn Lindgren and Arne Johansson describes the redesign of a small-scale subsonic tunnel to include an expanding corner. As a part of this redesign, the turning vanes in the first corner of this tunnel were optimized for use in the new corner. The inverse design features of MISES were utilized to alter the previous vane design to work with a two dimensional expansion ratio of 4/3. The resulting turning vanes had a pressure-loss coefficient of 0.041 which is only slightly worse than the original vane [10].

1.2.4 Bezier curve parameterization

While there are many ways to define an airfoil, for the sake of optimization it is beneficial to find a method which reduces the number of design variables required to fully define the shape. Bezier curves present one such method which has been described in a number of papers. The paper "A Survey of Shape Parameterization

Techniques", for instance, describes the use of Bezier control points in airfoil optimization as a way to reduce the number of design variables while still covering a large amount of design space. The paper also notes the benefits of this method over similar spline techniques due to the fairly close relation between the control points and the position of the defined curves [11].

The usefulness and versatility of this form of parameterization are expanded upon by "Optimum aerofoil parameterization for aerodynamic design". This paper demonstrated the ability of various configurations of multiple low order Bezier curves with positions calculated by airfoil parameters to replicate existing NACA airfoils. Though there was some discrepancy for all configurations, the close matching indicated a good ability to represent a wide array of geometries with few control points [12].

1.3 Scope of Present Research

The research discussed in this thesis will focus primarily on the development of a tool for use in the process of optimizing turning vanes for expanding corners. For this reason, simplifying assumptions will be made and the vane designs presented will not be claimed to be the ideal vanes for use in the GLMWT. For instance, the expansion will be assumed to occur only in the two-dimensional plane. This ignores some features of the GLMWT, but is sufficient to demonstrate the efficacy of the design process and simplifies the analysis to work with the primarily two dimensional solver used by MISES. Possible amendments to this process which may overcome this limitation will be discussed in the further work section.

In addition, due to time and resource constraints, results will be verified only by comparing to the MISES model of the current vane design to a more robust commercial CFD code. Though this will not entirely guarantee the accuracy of the results gained with MISES, agreement between the two solvers should give some proof that optimizations made using this design tool will reflect improvements in real flows.

1.4 Contributions of Present Research

This work will generate a tool with which flow conditions incident at a corner may be used to optimize a design for turning vanes to use in that corner. The aim of this work is to simplify the optimization process by eliminating the need for inverse designs and intimate knowledge of MISES so that less experienced designers can achieve reasonable designs. The research additionally aims to allow optimized designs to be less reliant on the initial design.

Chapter 2: Methodology

2.1 Airfoil Definition

The geometry for each airfoil design was defined using Bezier curves. The usage of Bezier curves to define the airfoil provides three distinct advantages. First, the relatively small number of control points used to describe the airfoil drastically reduce the number of design variables in optimization problems and thus reduces computational cost. This is especially pertinent to derivative based optimizers since finite differencing requires at least one additional function evaluation per design variable for each iteration.

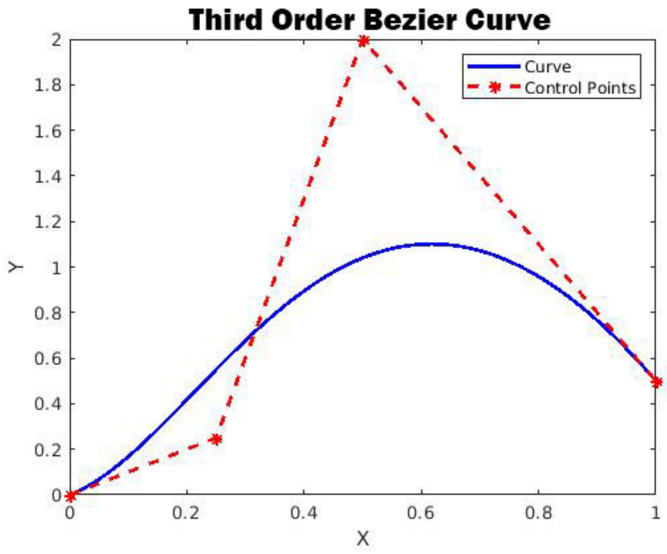


Figure 2.1: Example bezier curve displaying control points and generated curve.

Second, the nature of these curves allows large changes over the entire length of a curve to be made with the movement of a single control point. This gross alteration helps to prevent some of the strange design features that can arise in airfoil optimization.

Finally, the use of multiple Bezier curves allows fine optimization to be performed in a number of different ways by pinning or relating certain control points. Pinning the endpoints of a curve near the leading edge, for instance could allow the leading edge curve to be more finely tuned by increasing the density of control points in the area. This is of particular import in the design of turning vanes where the leading edge can define the sensitivity to incoming flow alterations [2].

2.1.1 Bezier curve definition

As discussed previously, Bezier curves are a parameterization method for defining curved lines through the use of a small number of control points. The distance from these control points is then interpolated to generate the necessary curves. Mathematically, this interpolation can be described for a curve of $(N-1)^{th}$ order using the equation:

$$X(t) = \binom{N}{1} C_{x_1}(1-t)^N + \binom{N}{2} C_{x_2}(1-t)^{N-1}t + ... \binom{N}{N} C_{x_N}; 0 \leq t \leq 1 \quad (2.1)$$

An equation of the same form may be used to describe an arbitrary number of additional coordinates in other dimensions, though for this research only two dimensions will be used.

Helpfully, this operation may be converted to a matrix form by expanding the polynomial terms. For $N = 2$, this formula becomes:

$$X(t) = \begin{pmatrix} 1 & t & t^2 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -1 & 2 & 0 \\ 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} C_{x_1} \\ C_{x_2} \\ C_{x_3} \end{pmatrix} \tag{2.2}$$

The matrix needed to perform this operation can be defined by populating the main diagonal with the correct row of Pascal's triangle for the number of control points then populating each K^{th} diagonal with the K^{th} value of the correct row multiplied by the K^{th} row of the triangle and -1^K. For clarity, a sample execution of this procedure is shown in table 2.1.

1
1 1
1 2 1
1 3 3 1
	First four rows of Pascal's triangle, final row represents main diagonal.

$1 * [1\ 3\ 3\ 1]$ $-3 * [1\ 2\ 1]$ $3 * [1\ 1]$ $-1 * [1]$	Definition of each diagonal starting with main diagonal and heading toward the corner.
$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$	Final matrix.

Table 2.1: Demonstration of Bernstein matrix creation

More information on Bezier curves is available from Ref. [14].

2.1.2 Smoothness constraints

Another useful mathematical feature of Bezier curves is that the curve becomes tangent to the slope between the endpoint and previous control point at the endpoint of the curve. This makes enforcing smoothness between two curves simple. Noting that the endpoint for one curve will be the first control point of the next, this relationship may be written:

$$\frac{y_2 - y_1}{x_2 - x_1} = \frac{y_3 - y_2}{x_3 - y_2} \tag{2.3}$$

Assumng that the point x_3 will be altered, a value for y_3 enforcing smoothness can be found:

$$y_3 = y_2 + \frac{(x_3 - x_2)(y_2 - y_1)}{x_2 - x_1} \tag{2.4}$$

Using these types of constraints, it is possible to generate smooth airfoils with multiple curves of smaller orders. In practice, this was necessary because the matrix needed to generate a curve with more than 35 control points creates large truncation errors and does not generate the desired airfoil.

2.1.3 Additional constraints

For these optimizations, the airfoils were defined by four Bezier curves of fifth order. The endpoints of each curve were shared to ensure continuity, and the smoothness constraint discussed in the previous section was used to ensure smoothness. In theory, it would be possible to generate an infinitely sharp leading edge at a specified point by omitting this constraint, but due to the ability of a blunt leading edge to mitigate sensitivity to variance in the incident flow conditions this configuration was not investigated [2].

Additional constraints were placed to ensure that the vane was feasible. Most important of these was the constraint on self-intersection. A MATLAB program written by Antoni Canos was used to check each airfoil curve for self-intersection [15]. Whenever a vane with intersecting upper and lower surfaces is created, an error is thrown. Using try and catch functions allows this error to act as a sort of penalty function by assigning a poor fitness value to any such cases. A similar process was used to prevent situations where the upper surface of one vane would intersect the lower surface of the next vane due to high thickness, high camber, or low vane gap distance.

2.1.4 Expansion ratio representation

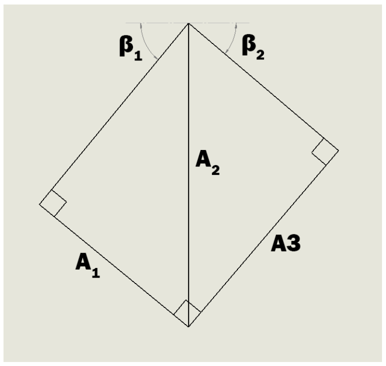


Figure 2.2: Illustration of geometric analysis for expansion ratio

Once the airfoil has been defined, it is necessary to have a parameter denoting the expansion ratio being applied to the cascade. The geometric analysis shown in figure 2.2 demonstrates that the inlet and outlet angles are sufficient to describe the expansion ratio. This geometric analysis takes the centerline of the cascade as the straight line between the inner and outer corner of the wind tunnel walls and uses the expansion ratio to set a relation between the width of the tunnel at the inlet of the turn and the width at the outlet. Using trigonometric functions to solve for the flow angle incident on the turning vanes gives the following definition:

$$\beta_1 = \frac{\pi}{2} - \beta_2 \tag{2.5}$$

$$A_2 = \frac{A_1}{sin(\frac{\pi}{2} - \beta_1)} \tag{2.6}$$

$$A_2 = \frac{A_3}{sin(\frac{\pi}{2} - \beta_2)} \tag{2.7}$$

$$\frac{A_1}{sin(\frac{\pi}{2} - \beta_1)} = \frac{A_3}{sin(\beta_1)} \tag{2.8}$$

$$\frac{A_3}{A_1} = ER \tag{2.9}$$

$$\frac{A_3}{A_1} = \frac{sin(\beta_1)}{cos(\beta_1)} \tag{2.10}$$

$$ER = tan(\beta_1) \tag{2.11}$$

This definition is expedient given that MISES defines its inlet angle in the same tangent form. Thus, the specification of the expansion ratio can be accomplished within the standard setup of the program.

2.1.5 Solidity definition

Solidity is an important parameter in any study of airfoil cascades as it defines how close the airfoils are to one another, thus affecting their impact on one another. This parameter has a number of definitions depending on the aerodynamic context, but in general, a high solidity indicates a small distance between airfoils and a low solidity indicates a large distance. For the purposes of this study, the solidity of the cascades investigated will be defined by the vane gap. This is taken as the vertical distance between one vane and the next.

2.2 MISES Operation

Analysis with the MISES software occurs in four steps, the first of which steps is file setup. In order to run, MISES requires a file defining the airfoil geometry and a file defining the flow constraints and initial values. These files are defined as blade.xxx and ises.xxx where xxx is an arbitrary case name.

The second step is grid generation, which occurs within the subprogram ISET. In a traditional case, ISET is used to define grid parameters and inspect the generated grid before writing to a file. Calling ISET loads the blade.xxx file and uses a panel code to initialize stagnation streamlines which help to define the boundaries of the grid. From here, user inputs initialize a grid between these boundaries and alter any parameters that affect point distribution along the airfoil. Fortunately, much of this process may be streamlined using a gridpar.xxx file containing parameter values. Within the context of this optimizer, the ideal value of these grid parameters should stay relatively constant, and the operation of ISET can be streamlined in this manner. Elliptical grid smoothing is also provided to increase the quality of the mesh. The generated grid with an initial condition defined by the panel code is output to a file idat.xxx.

Variable Number	**Definition**
1	Inlet flow slope
2	Exit flow slope
5	LE stagnation point
6	Grid exit static pressure
15	Inlet Mach number
Constraint Number	**Definition**
1	Drive inlet slope to SINLin
3	Set LE Kutta condition
4	Set TE Kutta condition
6	Drive inlet POa to $\frac{1}{\gamma}$
15	Drive inlet mach to MINLin

Table 2.2: ISES variables and constraints for program operation

Following the creation of the grid, the subprogram ISES must be called to complete the third step of the flow solution. ISES iterates the solution to satisfy the user-specified constraints by altering the user-specified variables. The constraints used for this paper may be seen in table 2.2. These constraints allow for the pressure ratio behind the cascade to be found for a given set of upstream flow parameters. The solver runs the first iteration as an inviscid case before including viscous effects. This efficiency measure and the use of a Newton flow solver result in convergence after relatively few iterations, usually less than 15 for the cases investigated. This step may also be performed by the subprogram POLAR. This program is generally used to sweep through flow parameters to investigate the operation of a cascade beyond its design point. Use of this program was determined to be necessary for the procurement of a value for total pressure loss.

The final step of the flow solution is performed by running the subprogram IPLOT. In standard operation, this program is used to generate and format plots of flow quantities. While this was used for debugging and testing, the primary use of IPLOT within this thesis was the creation of field.xxx files. These files contain data for a number of flow quantities along each streamline. This file simplified the transfer of data from the idat.xxx file into MATLAB.

These operations are run from the command line and primarily operate through user inputs to menu prompts. This makes the software difficult to use from within MATLAB as the command line interactions allowed by the $system()$ function do not permit the software to input information during program operation. Additionally, the complexity of the software largely precludes the creation of a MEX file which would allow operation entirely from within MATLAB. Instead, the packages TCL and Expect are used to automate the usage of MISES. These packages streamline the piping process commonly used in batch files to simulate user input. Due to the similarity of the cases presented during optimization, a single Expect file may be used to run through each step of the MISES simulation with a single command from MATLAB. For this research, a file MISExpect was written which sets up the grid, iterates using POLAR, then outputs the data to a field.xxx file.

Samples of each of these file types can be found in the Appendix to this report, and further information on MISES is available from Ref. [7].

2.3 Optimization Functions

The field of optimization is quite broad and contains a number of techniques and methods. To find the best method for optimizing this particular problem, a number of methods were explored. For the purposes of this investigation, unconstrained optimization was deemed to be the most useful. This is due to the fact that mathematically determining the existence of non-feasible designs, such as those where the turning vane surfaces intersect themselves, would be unnecessarily complex. Instead, a penalty function was added to unconstrained optimization techniques to discourage non-feasible designs.

The fitness function used in this optimization was a weighted sum of three parameters. These parameters are the total pressure loss, separation magnitude, and the difference between the turning angle and a right angle. These parameters were selected as the pressure loss and lack of separation are the primary needs of turning vanes. The turning angle parameter is included as a turning angle too high or too low would likely cause difficulties as the flow propagated through the rest of the straight tunnel section behind the corner.

2.4 MATLAB Operation

As discussed in prior sections, a number of MATLAB functions were generated to assist in the optimization presented. Though not all functions were ultimately used, they will all be presented for the purposes of future work.

2.4.1 Bezier curve functions

Two functions were used to define the curves for use in the turning vanes. The first of these functions was $BernMat()$. This function automatically generated the necessary matrix for the calculation of a Bezier curve of order $n-1$. This function was called by a separate function $Bez()$. $Bez()$ takes two vectors of control points, the number of points to be plotted in the curve, a string signifying whether multiple curves are to be used, and an optional vector containing the order of each curve. Using these inputs, the program uses for loops to generate the specified curves and outputs a vector of x values and a vector of y values.

2.4.2 File setup functions

As most MISES operations require the file they reference to be generated before the program will operate, creating functions that allow automatic creation of these files was one of the most important steps of this research.

The first of these functions is $VaneBuild()$. This function takes the control points, normalized vane gap distance, and an input structure containing flow data specified by the user. The function calls $Bez()$ with the given inputs and writes the results to a file blade.xxx. The file name and additional parameters are taken from the input structure.

The second of these functions is $InputSetup()$. This function creates the file ises.xxx for a given file name using both data specified in an input structure and values coded into the function. Though hard-coding values is a less than ideal manner of specifying them, the values handled in this way were determined to be unlikely to change for any case handled by the program. If necessary, these values could be added to the input structure and the function altered to accommodate the new format.

$DataRead()$ reads the field.xxx file for a specified file name and outputs an average pressure loss, mean separation distance, and outlet flow angle. A similar file $DataReadFit()$ performs the same process, but uses user-specified weights to calculate a fitness value from these three values. These files also call $SepVal()$ which reads the field.xxx and blade.xxx files to compare the locations of the blade surfaces and stagnation streamlines. These distances are averaged over the vane to create an approximate measure of the magnitude of separation present on the given airfoil.

2.4.3 Program operation functions

Of equal import to the functions which set up the files are the functions which operate the MISES program. The most vital of these is MISESEval which uses the $system()$ function to run the Expect file MISExpect from the command line. If the command is not carried out, an error is thrown.

To simplify other codes, the file $MISES()$ was written. This file takes the same inputs as $VaneBuild()$ and runs $VaneBuild()$, $InputSetup()$, $MISESEval()$, and $DataRead()$. As with $DataRead()$, a similar program $MISESFit()$ is used to output a fitness value directly rather than the three values output by $DataRead()$.

2.4.4 Optimization functions

With the ability to run and gather results from MISES handled by the functions described previously, functions for optimizing vanes within this environment were simple to create. Similar to Lindgren and Johansson [10], the normalized drop in total pressure across the vane was the main value being queried by the optimizer. MISES outputs this value with the definition:

$$\bar{\omega} = \frac{p_{o2}^{isen} - \bar{p_{o2}}}{p_{o1} - p_1} \tag{2.12}$$

This definition, output by the POLAR function, compares the isentropic total pressure to the mean total pressure and normalizes by the inlet conditions. This provides a convenient value to minimize to ensure good vane operation. Additionally, this value should be relatively simple to obtain from the StarCCM+ models. During early testing of some of the optimization functions, it was found that the method of calculating loss being used at the time only considered flow between the stagnation streamlines. This allowed large separations to form as the losses due to these separations were not impacting the fitness values. To counteract this, the mean separation value was added to the fitness function and highly prioritized by the weighting. Though the loss calculation was later altered to the one specified above, this parameter remains useful to ensure as little separation occurs as possible. Additionally, adding minimal deviation from a specified turning angle as an additional optimization goal could help to prevent designs which drastically overturned or under-turned the flow. With these goals and parameters set, the functions could be created and tweaked.

The first of these functions is MOptiUncon. This function performs a Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization aimed at minimizing the fitness function calculated by MISESFit. This function takes an initial design configuration and a few run parameters such as tolerance and finite differencing calculation step size and outputs a new design and the final design's fitness value. The smoothness constraint discussed in section 2.1.2 was applied within this function as it effectively reduces the number of design variables and the smooth transitions were deemed to result in a better quality of vane.

Two more functions were created to perform the same task with different optimization functions. One, MOptiSearch.m, used fminsearch() to optimize the turning vane while the other, MOptiGA.m, used ga(). The fminsearch() function uses a form of simplex algorithm to optimize a given function. Conversely, ga() operates a genetic algorithm to optimize the problem. Due to the nature of the penalty function, it is necessary to provide all three functions with a feasible initial design to achieve a feasible result. In testing, non-feasible designs were unable to provide a reasonable direction for the optimizers within the design space that pushed the altered designs to become feasible. MOptiGA was particularly susceptible to this difficulty as standard operation allows the function to generate the initial population without regard to feasibility. As such, an expression to generate the initial population by adding bounded random values to each of the parameters was added to this function.

2.4.5 Miscellaneous functions

In addition to the functions described above, a few functions were written for convenience or written and subsequently determined to be unnecessary. The function $InStructSet()$, for instance, is a function that automatically creates the necessary input structure for the file setup and MISES operation functions from a set of hard-coded values. This function is useful to ensure that element names within the structure are consistent with those used in the functions as well as to quickly generate many structures with similar values to accommodate slightly varied cases.

The functions $VaneCalc()$ and $VaneFit()$ were used early in testing to find values of the Bezier Parameters similar to those of the current turning vane. These functions used $fminsearch()$ and a distance calculation to optimize the parameters such that the curve generated by $Bez()$ passed through a given set of points along the current GLMWT turning vane design. The generation of these parameters was useful for creating an initial, feasible design for later optimizations as well as providing good practice with the MATLAB optimization toolbox.

The function $GridPar()$ was written to automatically generate a gridpar.xxx file for use in ISET operation. It was quickly determined that adding a pause function to the optimization programs and allowing a manual setup of the first parameter set was more advantageous, and this program was not utilized.

2.5 Flow Similarity Parameters

To ensure the applicability of the flows found using MISES to those generated in the real world, it was necessary to match flow parameters. For this flow, chord Reynold's matching and Mach number matching were used to achieve this. Using standard sea level conditions taken from Ref. [16] for viscosity and density, the flow speed and characteristic length were calculated by using the flow speed at the first corner corresponding to the tunnel's top speed and the chord length of the original vane. Likewise, the flow speed at the first corner was used to calculate the Mach number. Noting that the desired setup of the expansion ratio would result in a decreased area at the corner, a relation between both similarity parameters and the expansion ratio was generated. This process, largely predicated on the continuity equation, can be seen below where the subscript c denotes a value at the corner entrance, ts denotes a value at the test section, and the notation A_c' denotes the original area at the corner:

$$Re = \frac{\rho v_c l}{\mu} \tag{2.13}$$

$$v_c = v_{ts}\frac{A_{ts}}{A_c} \tag{2.14}$$

$$A_c = A_c' \cdot ER \tag{2.15}$$

$$\frac{A_{ts}}{A_c} = 0.3269 \cdot ER \tag{2.16}$$

$$Re = \frac{1.204kg/m^3(103m/s(0.3269 \cdot ER))(0.6515m)}{1.789 \cdot 10^{-5} Pa \cdot s} \tag{2.17}$$

$$Re = 1.476 \cdot 10^6 \cdot ER \tag{2.18}$$

A similar process can be used for Mach number:

$$M = \frac{v_c}{a} \tag{2.19}$$

$$a = 343m/s \tag{2.20}$$

$$M = 0.0982 \cdot ER \tag{2.21}$$

Additionally, for simplicity, all design dimensions were scaled by the chord length of the current GLMWT turning vane.

2.6 Data Acquisition

Once the functions were created, data for this research was gathered in a few steps. The first of these steps was a simple parameter sweep similar to the one performed by Lopez et al. in Ref. [9]. This process served a few purposes. First, the sweep demonstrated the ability to operate MISES from within MATLAB on valid vane geometries. Second, the results of the sweep gave some understanding of the effects of these parameters on the pressure loss associated with these blunt turning vanes. To this end, camber, thickness, and cascade solidity were all investigated for a simply modified version of the current GLMWT first corner vane. The vanes can be seen in figure 2.3. Finally, the sweep helps to bridge some gaps left by Lopez et al.[9] in terms of camber and cascade solidity.

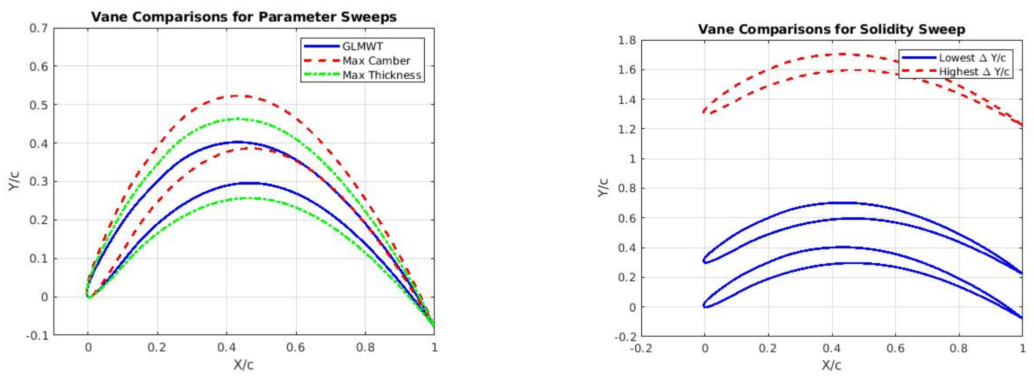


Figure 2.3: Vanes used in parameter sweeps

Following the parameter sweep, a number of optimization cases were performed. First, each optimizer was run using the GLMWT vane as an initial condition and constraining the vane gap to a constant value. This process used an expansion ratio of 1.2, but cases with other expansion ratios were also performed to test the functions. Second, the same optimization was performed using a variable solidity to determine if the added design parameter would alter the vane design significantly. During this process, alterations and fixes to each of the solvers were made to ensure acceptable performance.

Finally, the lessons from the parameter sweep and the many optimization cases were taken and used to experiment with new starting points and parameters. These cases spanned a wide range of investigations and had a large number of outcomes, so a select few with notable outputs were included in this report.

2.7 Verification

The results of this optimization were verified using the CFD software STAR-CCM+. The vane design for both the current GLMWT first corner design and the newly optimized design were modeled in the CAD package SolidWorks and imported into STAR.

2.7.1 Model generation

The models for verification were generated using the tools provided in Solidworks. First, the blade.xxx file is imported into Excel and the Cartesian coordinates of the vane are prepared for import into SolidWorks. This preparation involves scaling by the original vane chord to ensure flow similarity and placing the data in the correct format. Next, the vane is loaded into SolidWorks and used to create a 2-D curve. This curve is copied using the pattern tool to generate the correct number of vanes for the present case with the correct cascade angle to generate the necessary expansion ratio. Straight and parallel tunnel walls are then created to allow for proper wake propagation. These walls were made to be roughly 5 chord lengths ahead of the corner and 10 chord lengths behind the corner. Finally, a semi-arbitrary curve was generated at the tunnel wall using a four point style spline. This spline was constrained to be tangent to the inlet and outlet walls and dimensioned to appear similar to the vane shape. Given that MISES has no way of handling wall geometries, the primary point of interest in this verification will be away from the wall and wall geometry is thus less influential. Figure 2.4 shows the models used.

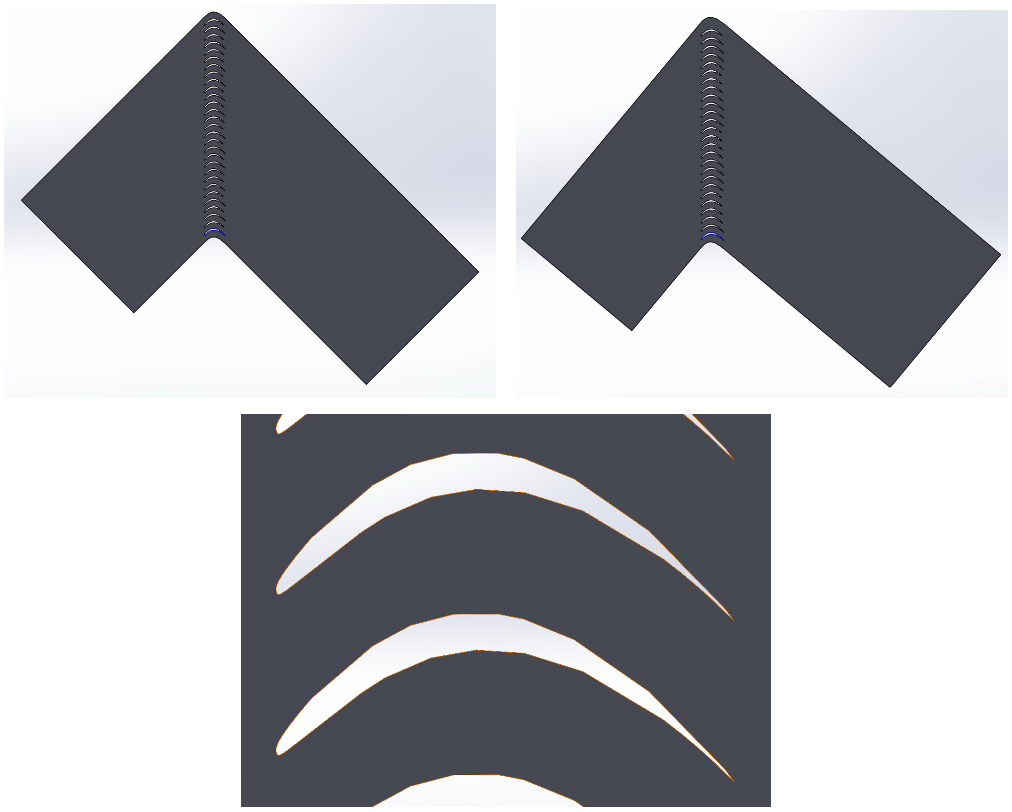


Figure 2.4: Tunnel models used in STAR-CCM+. Top left is non-expanding, top right is expanding, bottom center is detail view of the vanes.

2.7.2 Meshing

To ensure proper handling of boundary layers, the mesh generated within STAR-CCM+ was generated such that the maximum wall y+ was between 1 and 5 to fit within the bounds of the wall functions used. To achieve this, prism layers with a controlled thickness at the wall and defined total thickness were added. For simplicity, a triangular mesh was used in all other regions with wake refinement controls added to the vane geometry. The mesh for the non-expanding case using the GLMWT vane can be seen in figure 2.5 with detail views in figure 2.6. Major meshing parameters can be seen in table 2.3.

Variable Number	**Definition**
Parameter	**Value**
Minimum Surface Size	1.0e-4
Number of Prism Layers	15
Prism Layer Near Wall Thickness	1.0e-5
Prism Layer Total Thickness	0.01m
Vane Surface Controls	
Target Surface Size	0.05m
Isotropic Wake Size	0.05m
Wake Growth Rate	1.25

Table 2.3: Non-default meshing parameters

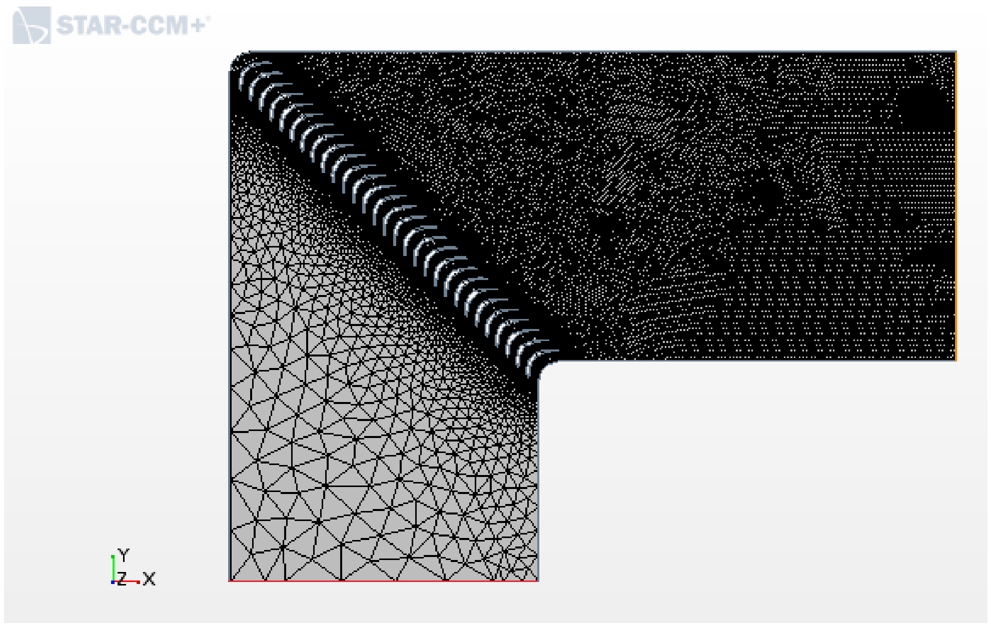


Figure 2.5: Mesh view of the non-expanding model

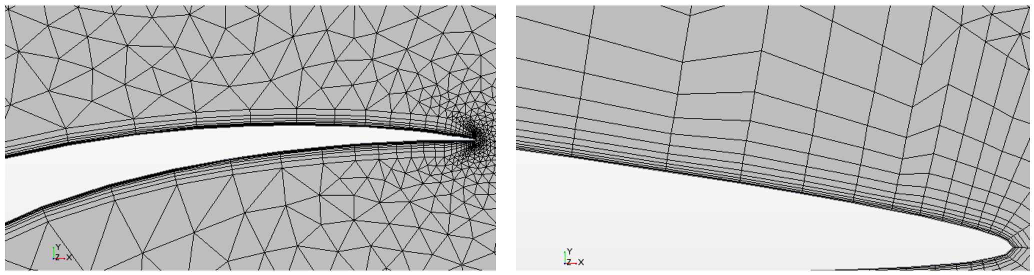


Figure 2.6: Detail views of the mesh at the trailing edge to illustrate prism layers

2.7.3 Simulation setup

To properly model the conditions in the tunnel, the similarity parameters used in the MISES file are related to the conditions of the tunnel modelled in STAR-CCM+. To do this, it was necessary to alter the properties of air in the simulation to have the same dynamic viscosity as the one used to calculate the Reynold's number in the simulation. By setting the inlet velocity to match the specified inlet Mach number and ensuring static temperature and the chord length of the model were correct, the correct Reynold's number was achieved naturally.

In selecting the models, the problem was assumed to be steady and the air was assumed to act as an ideal gas. Turbulence was modelled with a Realizable K-Epsilon solver. All other models were selected as the STAR defaults for a two dimensional steady problem.

2.7.4 Data collection

To assess the similarity in solutions between MISES and STAR, three monitors were considered. First, a visual assessment of pressure and velocity contours was used to verify that no apparent errors had occurred within the simulation. Second, the pressure contours around the vane were examined and compared to the plot generated for the identical case using MISES. Last, a line probe was generated parallel to the cascade about one chord length away and not reaching the walls. This probe was used to calculate the mean pressure loss and to compare with these values from MISES.

Chapter 3: Results

3.1 MISES Results

On operating the MATLAB wrapper for MISES, there are a number of observations to be made. First, the speed of operation should be noted. As expected from the typically quick convergence and relatively low computational requirements of MISES, operation was relatively quick. For reference, the parameter sweep routine generates and solves 60 MISES cases and takes roughly ten minutes to complete on a personal desktop with a 6-core i7 processor and 16 GB of RAM. Unfortunately, the nature of optimizers prevents the same from being true for full optimization cases. Dependent on the method used and tolerances specified, the optimizers took between thirty minutes and multiple hours to output a converged value. Compared to an optimizer using a different CFD solver, this is fairly quick. Additionally, the quick convergence of each case indicates that methods with better convergence could speed this total process in future iterations.

Second, the need for good engineering judgment in the operation of these programs must be noted. Though the optimization process requires less knowledge of the aerodynamics than an optimization within MISES, the ability of improper weighting specifications, initial vane definitions, and other issues to cause non-convergence or to generate vanes with poor performance indicates a need for careful consideration of these parameters. Further, a basic understanding of MISES operation and aerodynamics is needed to verify the results of any optimization. Certain functions generated designs with non-viable streamlines or which did not converge and had questionable pressure profiles warranting additional scrutiny.

3.1.1 Parameter sweep

The results of the parameter sweep confirmed some theories about the effects of the parameters while also contributing some new insights. It should be noted that a larger value for loss represents a worse vane as does a larger value for separation. Additionally, the turning angle is defined as the angle of the outer corner of the wind tunnel. As such, smaller values represent sharper turns and a value of 90 degrees represents a perfectly right-angled turn. Values below 90 degrees will be referred to as over-turning and values above as under-turning.

Sweeping from the current vane to a modified, high-camber version showed that increasing this camber generated increased pressure losses and increased over-turning. This is generally not surprising as the higher camber will result in an effectively larger angle on the latter portion of the vane and flow tangency will encourage a larger turning angle due to this. Despite the larger losses, the higher turning angle could prove beneficial in cascades with a larger vane gap. Depending on what flow features are generating the largest pressure losses, having fewer vanes spaced further apart with a higher turning angle could help to turn the flow with

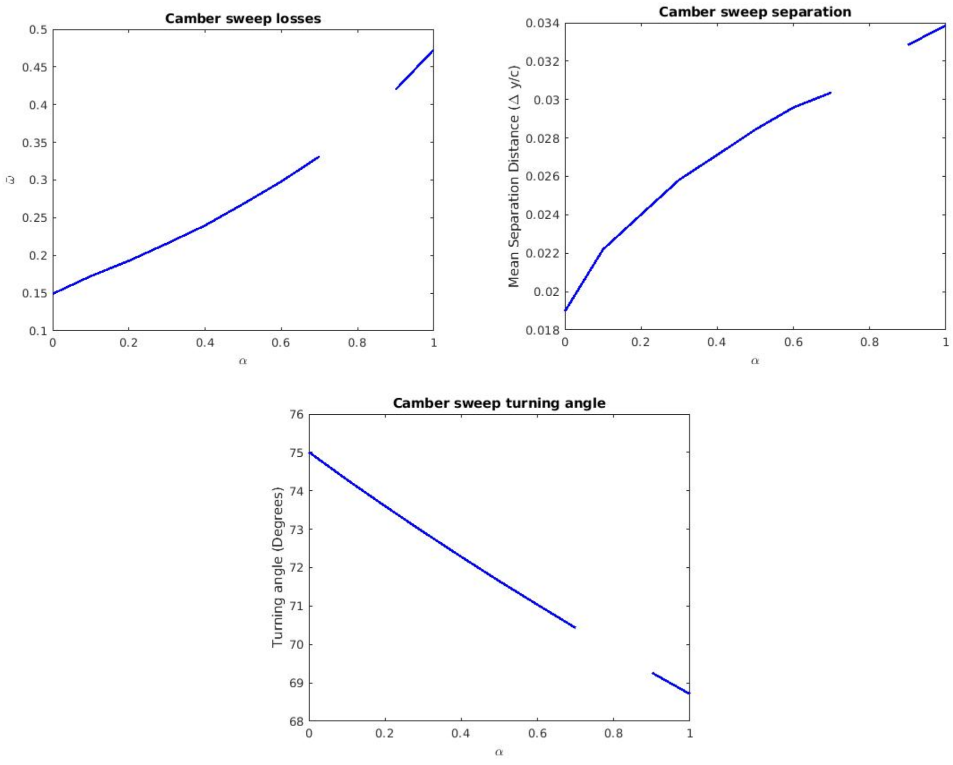


Figure 3.1: Parameter sweep plots for camber. Alpha indicates distance along interpolation from current GLMWT vane to modified version.

less overall pressure loss. Also notable is the gradual increase in mean separation distance. This seems to stem from the fact that the larger camber induced a larger adverse pressure gradient along the leading edge of the lower surface and trailing edge of the upper surface which caused a larger separation to form. This can be demonstrated in figure 3.2 which shows the stagnation streamlines generated by MISES for the maximum camber vane.

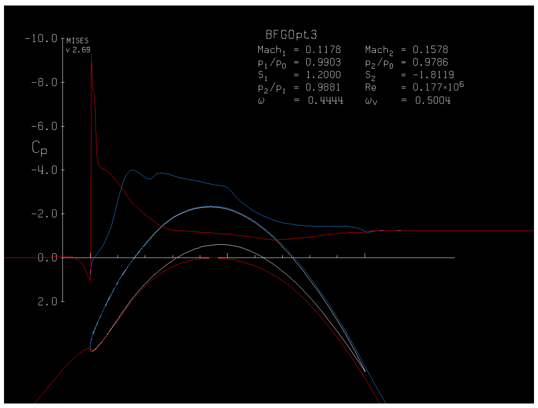


Figure 3.2: MISES plot of maximum camber vane solution

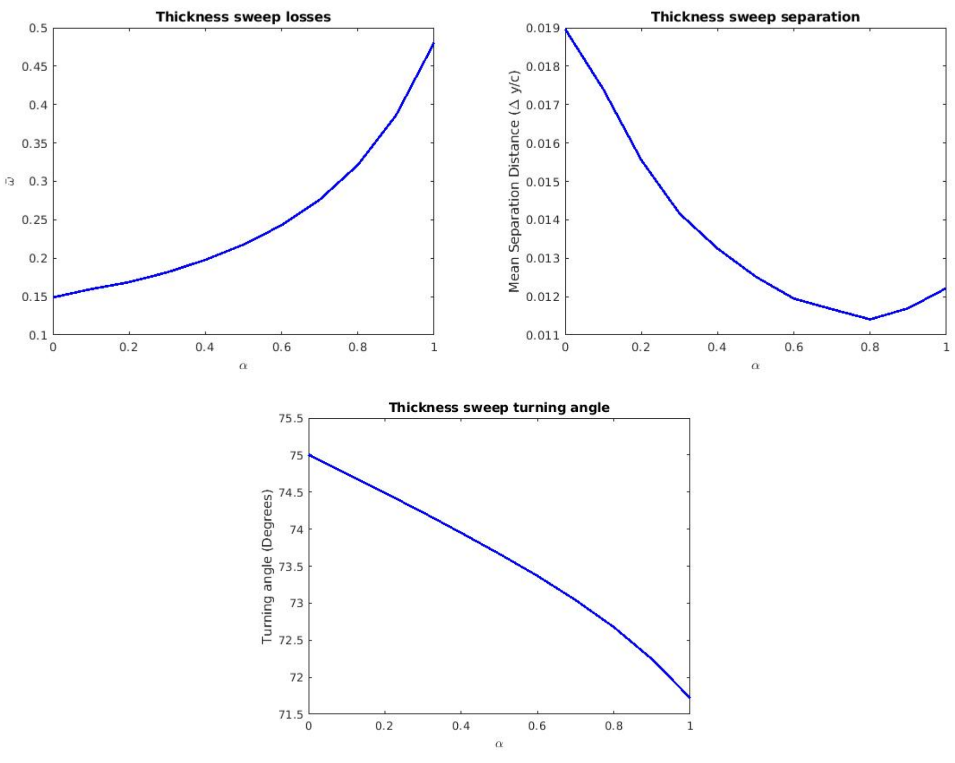


Figure 3.3: Parameter sweep plots for thickness

Performing a sweep from the current vane to a thickened version provided similar results. Adding material and constraining the channels between the vanes appears to have increased losses within the cascade but also limited separation. As with the camber sweep, these results could have implications for high vane gap cascades as the turning angle is slightly sharpened and the slightly reduced separation could be beneficial when the space between vanes is larger. To gather further data about these possibilities, both the highest camber and highest thickness vanes were subjected to a sweep of cascade solidity.

Sweeping through the solidity provided some more interesting results. Sweeping from slightly below the current vane gap to almost four times this value showed a gradual decrease in losses and over-turning with a gradual increase in separation.

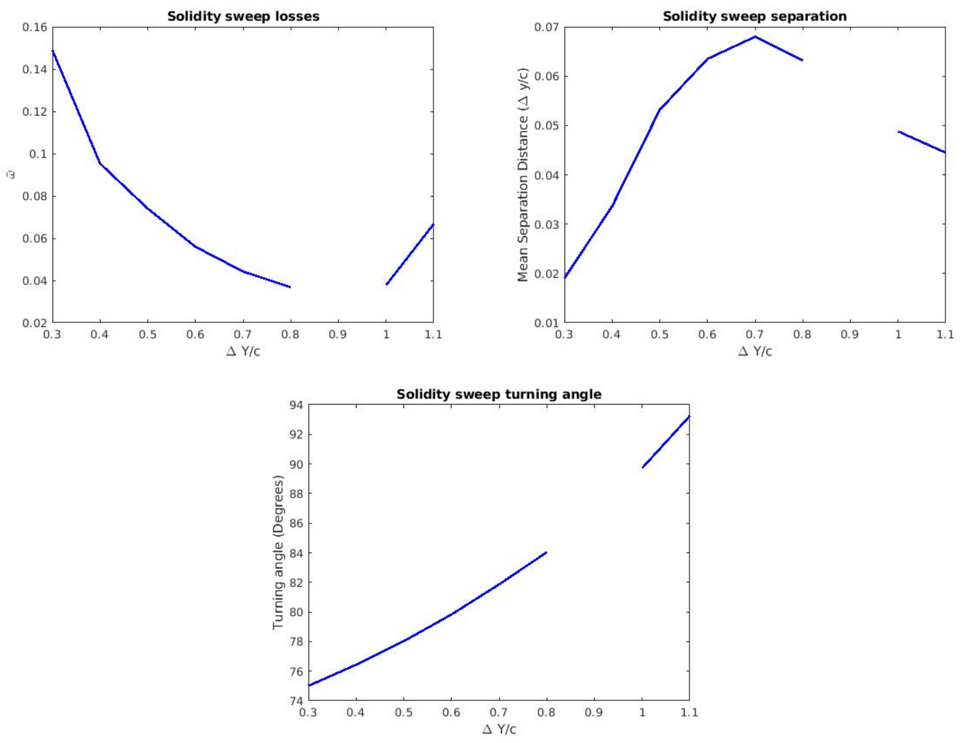


Figure 3.4: Parameter sweep plots for solidity

It seems that a vane gap that is too high induces larger separation as the larger distance between vanes limits the propagation of the flow information imparted by the vanes. Thus, a large vane gap limits the ability of the upper surface of one vane to prevent separation on the lower surface of the next vane by imposing a flow direction and vice versa. This effect can be seen in the stagnation streamline shape of figure 3.5. It is unclear what causes the convergence failure at 0.9, but the most likely culprit is

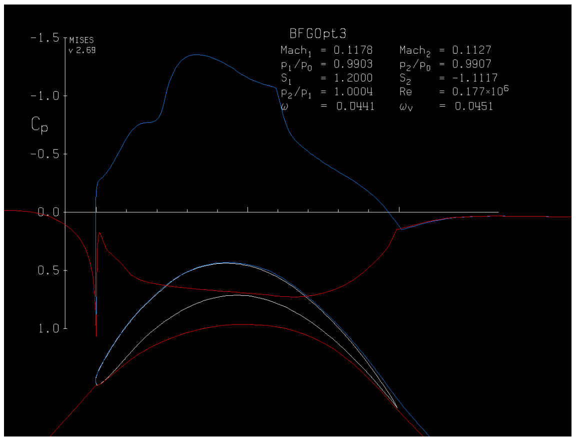


Figure 3.5: MISES plot of 0.7 vane gap solution

instability in the separation causing an unsteady problem that MISES cannot converge.

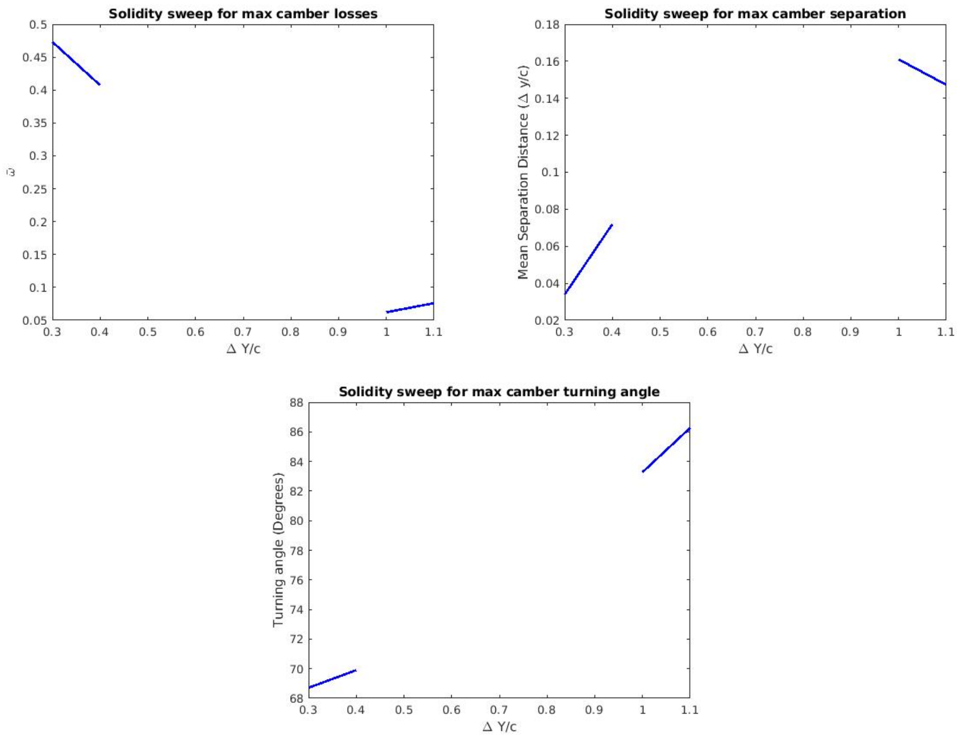


Figure 3.6: Solidity sweep for max camber vane

Repeating the solidity sweep with the high camber and high thickness vanes shows similar results, but demonstrates the effects of the altered designs. Overall, as the distance between the vanes increased, the losses incurred decreased, but the separation caused increased. Many of the solidity values for camber did not converge, likely for similar reasons to the non-convergence in the camber sweep. Due to this lack of data, it is unclear what effect the solidity had on the high camber vane, but the general trend and inability to solve the problem indicates that high camber, intermediate vane gap designs are not the best to pursue. The maximum thickness vane presents an enticing case around a normalized vane gap of 0.8 where the loss

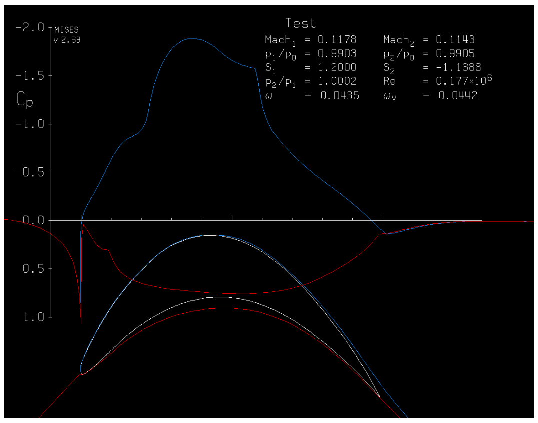


Figure 3.7: MISES plot of maximum thickness vane at 0.8 vane gap

is lower than that of the standard vane at a low vane gap and the turning is near 90, though the mean separation distance is higher. This case can be seen in figure 3.7. As speculated in the discussion of the thickness sweep, the added material limits the size of the separation. Based on the contours shown, an adjustment to further thicken the vane and alter the angle of the trailing edge could overcome this separation even at a high vane gap.

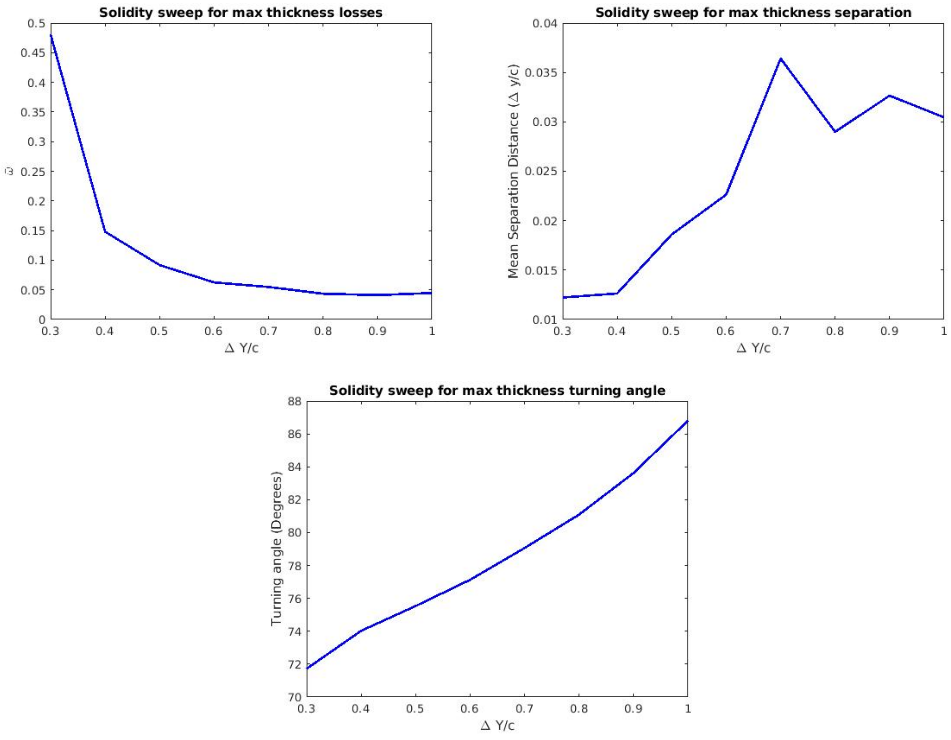


Figure 3.8: Solidity sweep for max thickness vane

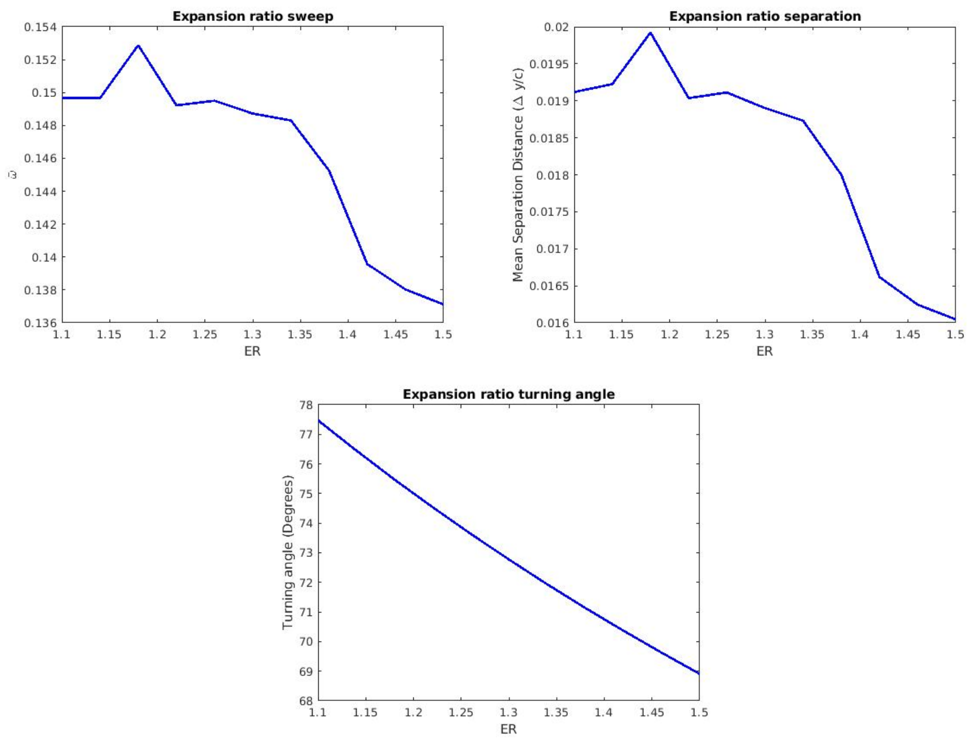
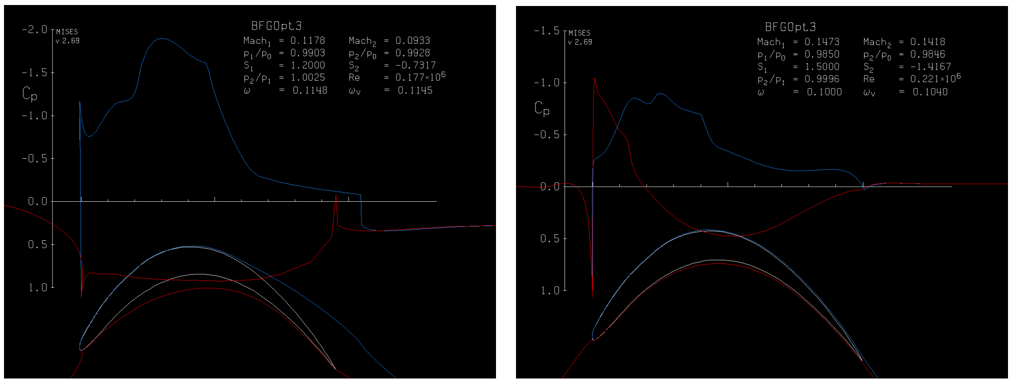


Figure 3.9: Expansion ratio sweep

Finally, sweeping through a number of expansion ratios demonstrates some unexpected phenomena. The decrease in both losses and separation seem to run counter to what one could expect from increasing the expansion ratio for a vane not optimized to that expansion. It is possible that this is the result of effectively pitching the airfoil because of the definition of expansion ratio in the parameters. Running two cases at ER=1.3 and ER=1.5 returns the plots in figure 3.10. These plots are very similar, but the differences in the parameters may help to explain some of the strange results seen in the sweep. The difference in inlet pressure ratio, for instance, could be altering the loss calculation as the formula used may be converted to the form seen in equation 3.1 by presuming that $p_{o1} = p_{o2}^{isen}$.

$$\bar{\omega} = (\frac{1}{1 - \frac{p_1}{p_{o1}}})(1 - \frac{\bar{p_{o2}}}{p_{o1}}) \tag{3.1}$$

Because the ratio in the denominator decreases as expansion ratio increases, it is possible that the actual loss in static pressure was greater for the larger expansion case but inflated by the normalization for the other cases.



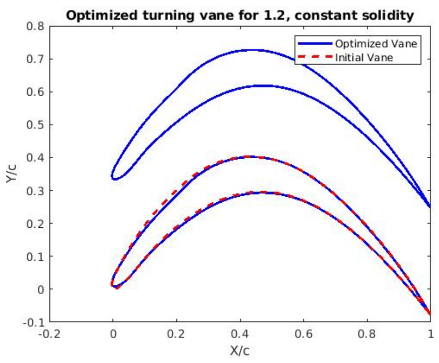
```
BFGOpt3
Mach1  = 0.1178    Mach2  = 0.0933
p1/p0  = 0.9903    p2/p0  = 0.9928
S1     = 1.2000    S2     = -0.7317
p2/p1  = 1.0025    Re     = 0.177×10^6
ω      =  0.1148   ωv     =  0.1145
```

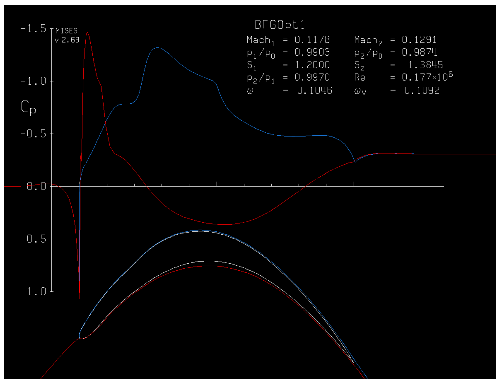
```
BFGOpt3
Mach1  = 0.1473    Mach2  = 0.1418
p1/p0  = 0.9850    p2/p0  = 0.9846
S1     = 1.5000    S2     = -1.4167
p2/p1  = 0.9996    Re     = 0.221×10^6
ω      =  0.1000   ωv     =  0.1040
```

Figure 3.10: Comparison of ER=1.2 and ER=1.5 case MISES solutions

3.1.2 Constant solidity optimizations

Setting up the optimization functions was a difficult process, but ultimately resulted in some decent optimization tools. The main difficulties in setting up these functions were determining what file formats and function syntaxes would work with the prebuilt optimization functions. Once this process was complete, finding proper option values and fitness function weights took more experimentation. Despite this, the speed of operation made experimentation less odious. The following vanes were generated using the weighting vector [10, 5, 0.1, 0.4].





```
BFGOpt1
Mach1 = 0.1178    Mach2 = 0.1291
p1/p0 = 0.9903    p2/p0 = 0.9874
S1    = 1.2000    S2    = -1.3845
p2/p1 = 0.9970    Re    = 0.177×10^6
ω     =  0.1046   ωv    =  0.1092
```

Figure 3.11: Comparison of MOptiUncon constant solidity output to current GLMWT vane and MISES solution

Understandably, *MOptiUncon*() was the fastest of the optimizers. With reasonably high tolerances, a single optimization case usually took around an hour to complete. Additionally, the optimizations were generally pushed towards cases with quick convergences indicating little to no separation or major viscous effects. One such case can be seen in figure 3.11. This optimization was run with an expansion ratio of 1.2 using the original turning vane as the initial design. The results in this case illustrate some of the features common in *MOptiUncon*() cases. The slight indent on the upper surface near the leading edge, for instance, is a feature that appeared in many cases. This feature appears to counteract some of the adverse pressure gradients that can appear and create separation. As such, very little separation appears in this design. It must be noted that in the process of testing these functions, the *MOptiUncon*() function had a tendency to fail to alter the design if not given the correct tolerance parameters. Troubleshooting this optimizer was also more difficult than the others due to the lack of visualization tools provided to observe the current state of any optimization case. Additionally, final designs were very similar to the initial designs, showing that the optimizer had difficulties moving away from local minima.

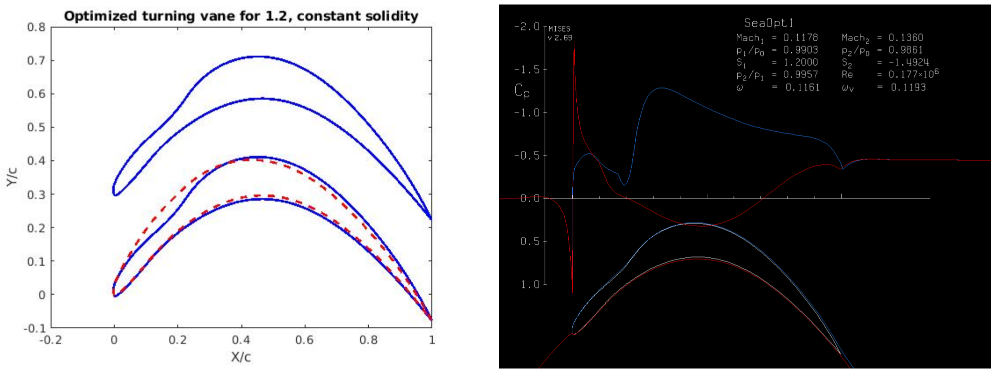


Figure 3.12: Comparison of MOptiSearch constant solidity output to current GLMWT vane and MISES solution

MOptiSearch() took longer to converge. With a similar tolerance, this optimizer took roughly one to three hours to complete. A characteristic design generated with this function can be seen in figure 3.12. This design illustrates the tendency of this optimizer to grossly alter the trailing edge as well as the mid-airfoil features. The features demonstrated in this design are also much different than those gained with *MOptiUncon*(). The strange indent seems to interact with the pressure gradients to help minimize separation on the lower surface. Plotting a contour of total pressure losses as in figure 3.13 illustrates the possible benefit of this method as the most prevalent losses were those experienced at the lower surface separation boundary. An unfortunate fea-

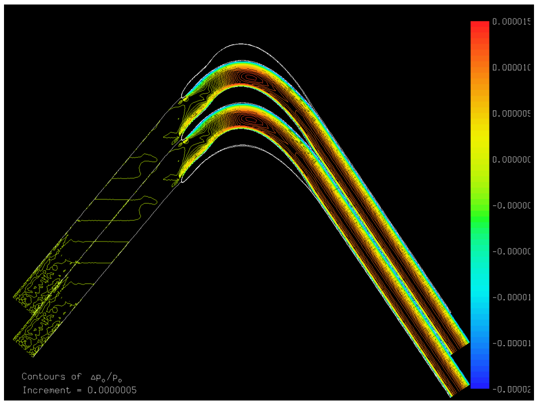


Figure 3.13: Total pressure loss contours for optimized vane. Blue areas indicate larger total pressure loss.

ture of this optimizer is its tendency to query a large number of infeasible designs. Because ISES is being run from within POLAR, it is not possible to directly constrain the allowed number of iterations. This combination is largely responsible for the lengthy optimization. This length was helped by the in-built visualization as it was more apparent when cases were not running correctly. If all initial values were the penalty function value, for example, it was immediately clear that an error was occuring and the optimization should be stopped.

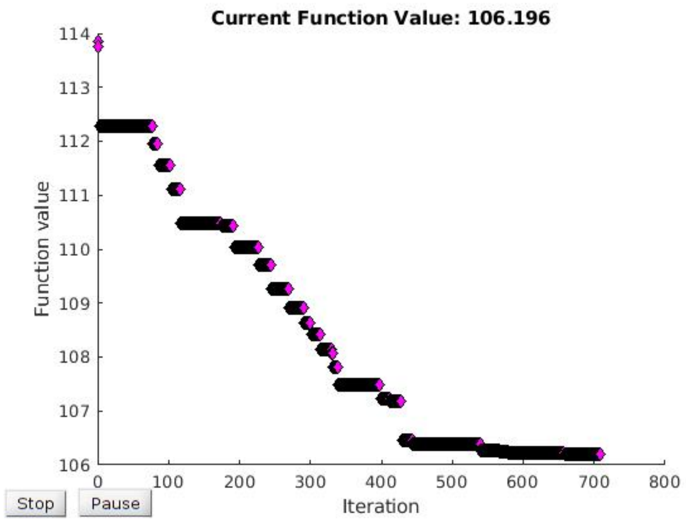
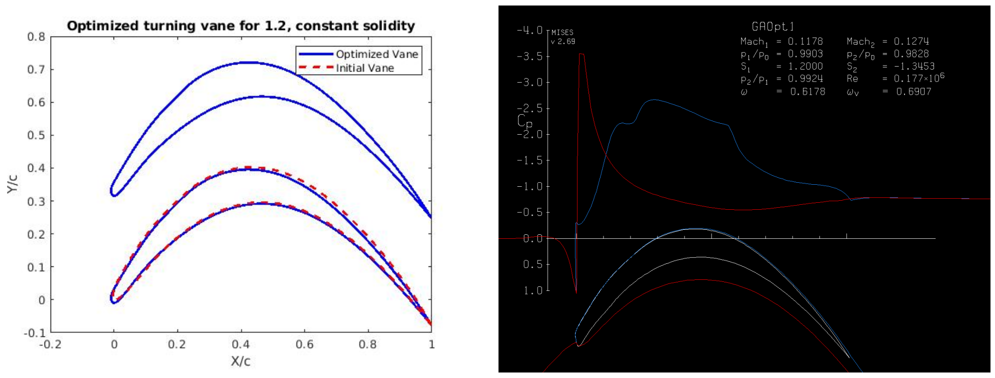


Figure 3.14: Function value monitor from MOptiSearch



```
GAOpt1
Mach1  = 0.1178    Mach2 = 0.1274
p1/p0  = 0.9903    p2/p0 = 0.9828
S1     = 1.2000    S2    = -1.3453
p2/p1  = 0.9924    Re    = 0.177×10^6
ω      =  0.6178   ωv    =  0.6907
```

Figure 3.15: Comparison of MOptiGA constant solidity output to current GLMWT vane and MISES solution

As with most genetic algorithms, *MOptiGA*() took a long time to run. This optimizer could take close to eight hours to complete a single case. Though it sometimes reaches a value much sooner, these values were often due to the initial population all being of poor fitness or luck to find a particularly good design early. Regardless, this long convergence time may be worthwhile as genetic algorithms help to avoid local minima. The relative similarity between the starting and ending profiles of the other two optimizers indicate that there are numerous local minima to be found and the overall design could suffer by using these solvers without accounting for this fact. The output of a *MOptiGA*() case can be seen in figure 3.15. Though this case was relatively similar in shape to the original design, the slightly extended chord length was a feature not seen in any of the other optimizers. The chord length was not constrained in any way, but it is likely that the many control points defining each curve diminished the effects of any one point moving to alter this parameter.

The results of a single optimization case beginning from the GLMWT vane for each solver can be seen in table 3.1. This table illustrates some of the differences in the solvers.

Case	**Fitness**	**Pressure Loss**	**Mean Separation Distance**	**Turning Angle**
GLMWT Vane	3.3566	0.1284	0.0218	75.3624
MOptiUncon	3.0599	0.1046	0.0156	75.64
MOptiUncon	3.0599	0.1046	0.0156	75.64
MOptiUncon	3.0599	0.1046	0.0156	75.64

Table 3.1: Fitness comparisons for constant vane gap cases

For the cases presented, each of the solvers arrived at similar values despite the drastically different designs. Each of the new designs demonstrates a decreased total pressure loss and separation distance, though *MOptiSearch*() increased the amount of overturning. This likely has to do with the weighting parameters used which heavily favored pressure loss and separation as optimization targets. Different weighting parameters or initial designs would likely result in even better vane designs.

3.1.3 Variable solidity optimizations

Repeating the optimization process with no constraints on the solidity provides similar results. Though it was theorized that allowing variation in vane gap distance would create major changes in the designs, actual alterations were usually minor. This lends more credence to the idea that many designs were being pushed to local minima. Additionally, computation times for these optimizations were similar to those with constant solidity as only one more design variable was being considered.

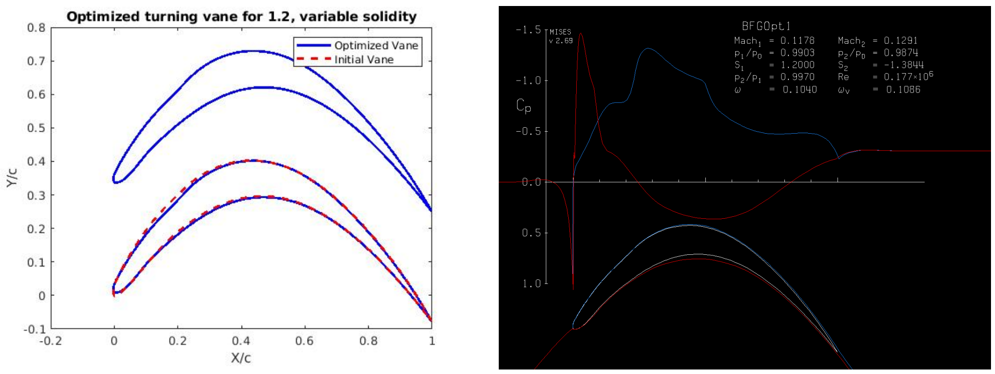
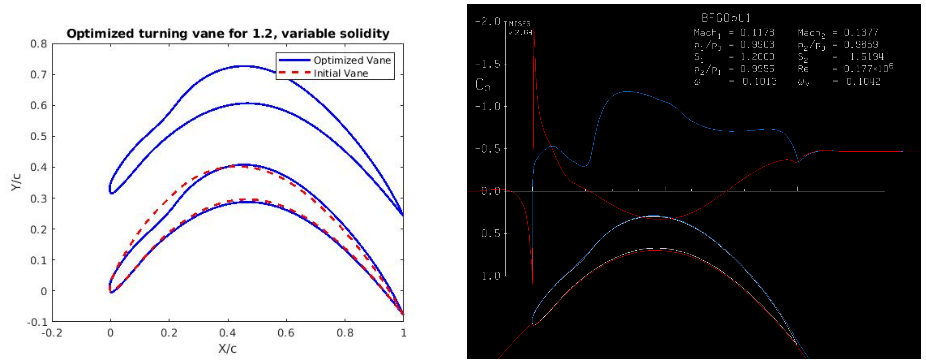


Figure 3.16: Comparison of MOptiUncon variable solidity output to current GLMWT vane and MISES solution

Figure 3.16 shows the output of one such optimization using *MOptiUncon*(). The vane created looks similar to the vane shown in figure 3.11. The solidity for this case was altered slightly which helped to decrease the pressure loss by 2.5%

compared to the constant solidity case. This comes at the cost of a minor increase in mean separation and in real design cases, altering the solidity may impact the way the vanes are placed in the corner. For this reason, if major improvements are not made, it may be beneficial to hold solidity constant.

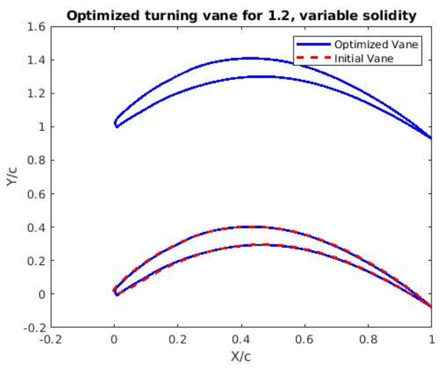


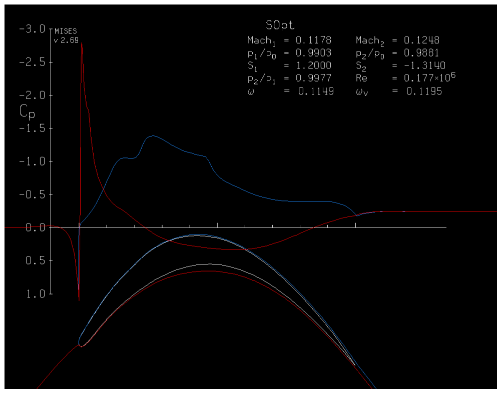
```
BFGOpt1
Mach1   = 0.1178    Mach2   = 0.1377
p1/p0   = 0.9903    p2/p0   = 0.9859
S1      = 1.2000    S2      = -1.5194
p2/p1   = 0.9955    Re      = 0.177×10^6
ω       =  0.1013   ωv      =  0.1042
```

Figure 3.17: Comparison of MOptiSearch variable solidity output to current GLMWT vane and MISES solution

Running a variable solidity case using $MOptiSearch()$ gives another vane similar to the constrained solidity case. Shown in figure 3.17, this case has an almost identical shape to the one shown in figure 3.12. As seen in the MOptiUncon optimization, the resultant case has a similar pressure loss at the cost of very minor additional separation. Again, this result encourages consideration as to whether the design benefits of allowing the vane gap to vary outweigh the added complexity later in the design process.

The most intriguing result of this round of optimizations was given by the genetic algorithm in $MoptiGA()$. The vane, seen in figure 3.18, had a dramatically altered vane gap distance close to 1.0. This extreme vane gap was complimented by a dramatically altered leading edge shape. The shape seen appears to be almost sharp. It is unclear whether this is the result of a failure in the smoothness constraint





```
SOpt
Mach₁  = 0.1178    Mach₂  = 0.1248
p₁/p₀  = 0.9903    p₂/p₀  = 0.9881
S₁     = 1.2000    S₂     = -1.3140
p₂/p₁  = 0.9977    Re     = 0.177×10⁶
ω      =  0.1149   ωv     =  0.1195
```

Figure 3.18: Comparison of MOptiGA variable solidity output to current GLMWT vane and MISES solution

or merely a very small radius curve that still mathematically satisfies the constraint while being sampled out by the number of points used to plot. Regardless, the fitness values and flow parameters seen in this case were fantastic. A 70% decrease in pressure loss and a turning angle of almost exactly 90 degrees was associated with only a small increase in separation. These results warranted further study, so a STAR case was performed which will be discussed later.

Comparing the values gained from these cases in table 3.2 shows the possible power of the genetic algorithm. If the correct configuration is generated, drastic improvements can be made. Likewise, the ability of the optimizers to alter solidity seems to have created a slight benefit in terms of fitness values. As mentioned previously, this benefit must be weighed against possible complications that could arise from slight alterations.

Case	Fitness	Pressure Loss	Mean Separation Distance	Turning Angle
GLMWT Vane	3.3566	0.1284	0.0218	75.3624
MOptiUncon	3.0292	0.1019	0.0157	75.6838
MOptiSearch	3.2481	0.1013	0.0101	73.1568
MoptiGA	0.6714	0.0382	0.0341	90.2386

Table 3.2: Fitness comparisons for variable vane gap cases

3.1.4 Additional designs of note

Using the information gained in this process, more optimizations were performed with alternate initial designs and weighting parameters. The results of these optimizations can be seen in table 3.2. Parameters not specified are identical to those used in the constant solidity cases.

Case	Fitness	Pressure Loss	Mean Separation Distance	Turning Angle
GLMWT Vane	3.3566	0.1284	0.0218	75.3624
MOptiUncon, max thickness, 0.8 vane gap	1.8805	0.0408	0.0220	81.3803
MOptiSearch, max thickness, W=[8,1,2,5]	37.4946 (3.1158)	0.2542	0.0072	67.2733
MOptiUncon, 0.4 vane gap, W=[10,1,2,5]	29.8180 (2.4543)	0.0854	0.0290	80.4471
MoptiUncon, ER=1.5	2.4955	-8.2e-4	0.0018	70.1867

Table 3.3: Fitness comparisons for cases of note. Fitness values in parantheses are using original weighting.

The results shown in this table demonstrate that alterations made to the initial conditions and weighting functions made significant impacts to the operation of the program. Two cases optimized using the maximum thickness vane, for instance were able to take advantage of the lower separation values seen in the parameter sweep. The first case used $MOptiUncon()$ to alter the vane at a large vane gap. Despite the vane gap, the losses experienced are one third those of the current vane, though this value should be taken with skepticism for reasons discussed in **??**. This design is shown in figure 3.19.

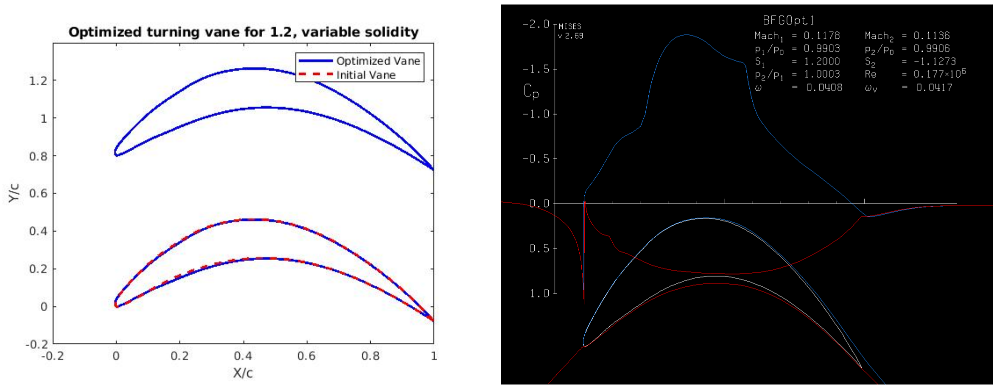
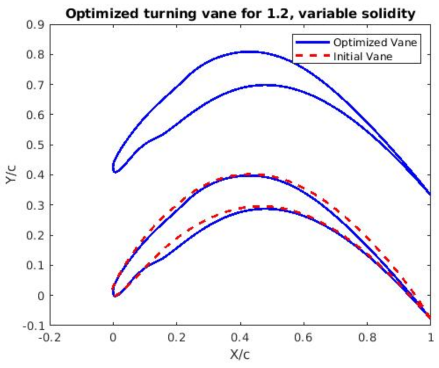
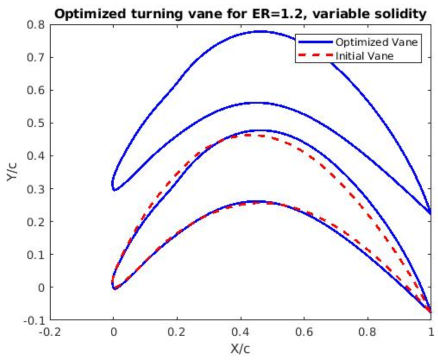


Figure 3.19: Comparison of MOptiUncon maximum thickness, 0.8 vane gap output to initial design and solution of MISES case

In one of the functions where the parameter weights was altered, the separation distance was significantly lowered despite having less importance ascribed to it. It is possible that this happened by chance, but could also indicate that the new weighting altered the gradients in a manner that led designs to be pushed towards lower separation as a means of obtaining the other quantities. Other cases support this thought as ignored parameters were seen to improve despite not being queried directly. More exhaustive experimentation would be required to determine whether this is true or more coincidence. Both designs generated with alternate weighting functions are shown in figure 3.20.



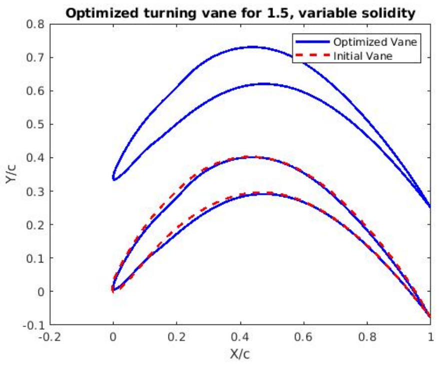
$MOptiUncon()$, W=[8,1,2,5]

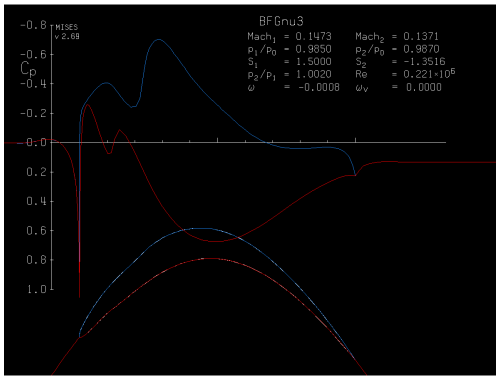


$MOptiSearch()$, W=[10,1,2,5]

Figure 3.20: Comparison of alternate weighting solutions to initial designs

A final design worthy of note was a case using $MOptiUncon()$ to optimize a case with an expansion ratio of 1.5. The design resulting from this case appears to have a negative pressure loss. It is not clear how this occurred, but it may be related to the trend seen in the expansion ratio sweep performed in section 3.1.1. Regardless of the cause, this negative value is almost certainly not physical, though it likely still corresponds to reduced losses.





```
BFGnu3
Mach1   =  0.1473     Mach2  =  0.1371
p1/p0   =  0.9850     p2/p0  =  0.9870
S1      =  1.5000     S2     = -1.3516
p2/p1   =  1.0020     Re     =  0.221×10^6
ω       = -0.0008     ωv     =  0.0000
```

Figure 3.21: Comparison of $MOptiUncon()$ ER=1.5 output to current GLMWT vane and solution of MISES case

3.2 STAR-CCM+ Results

Setting up the STAR cases proved to be somewhat difficult given the size of the cascades. Though the simulations were run on the computer cluster at the GLMWT with 48 cores and 256GB RAM, meshing procedures took roughly twenty minutes and simulations took at least 2 hours to converge when using a residual drop of four orders of magnitude. Despite this, the low number of simulations required for verification made the increased fidelity of the large mesh desirable.

The results from these simulations provide some insight into the feasibility of using this design technique.

3.2.1 Original vane, no expansion

Creating a simulation for the original vane in a configuration similar to how it is used in the wind tunnel currently resulted in the solution shown in figure 3.22.

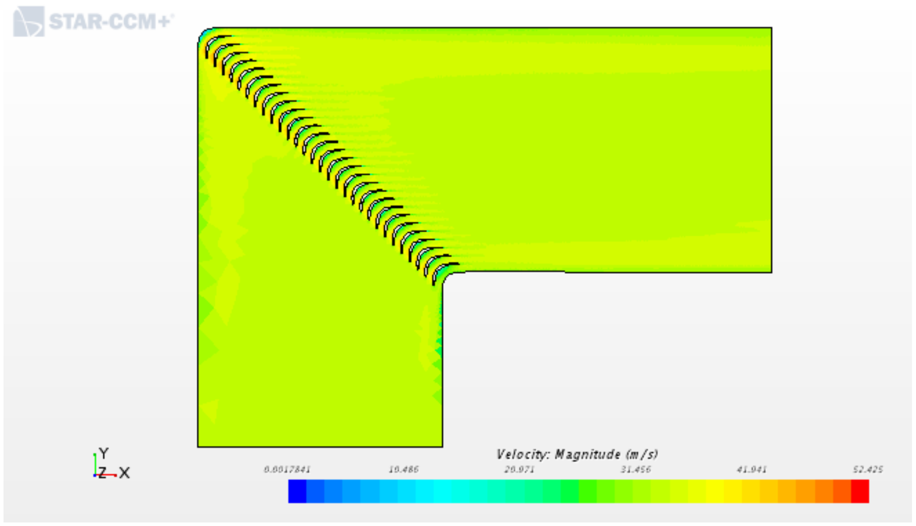


Figure 3.22: STAR-CCM+ solution of non-expanding case with current GLMWT turning vane

This solution acts as one might expect from the vanes being used in roughly their intended configuration. There are no apparent detached flows, but a slight velocity defect does form in the wake. These results run somewhat contrary to the results found using MISES. Namely, MISES appears to predict a moderate separation along the lower surface of the vane and higher losses than are predicted by the STAR model. In fact, the losses predicted by STAR simulation were 0.066 which is a little less than half of those predicted by MISES. Many possible explanations exist for this discrepancy including possible difficulties with the mesh refinement of the simulation, differences in the calculation of the flow properties used in the loss calculation, or non-intuitive interactions between the tunnel walls and the flow. To investigate, flow contours were compared in figure 3.23.

Though it is difficult to compare values due to the difference in formats, the contours shown in the MISES plot seem to reflect those in the STAR solution. The location of the stagnation point and various flow features such as the low pressure area stretching along the upper surface of the vane are very similar. It is also somewhat likely that the bubble of high pressure in the center of the lower surface corresponds to the separation seen in the MISES case. Noting these similarities, the suggestion that flow interactions with the wall may impact losses by imposing a new flow direction is also supported. Further research is necessary to determine the exact cause.

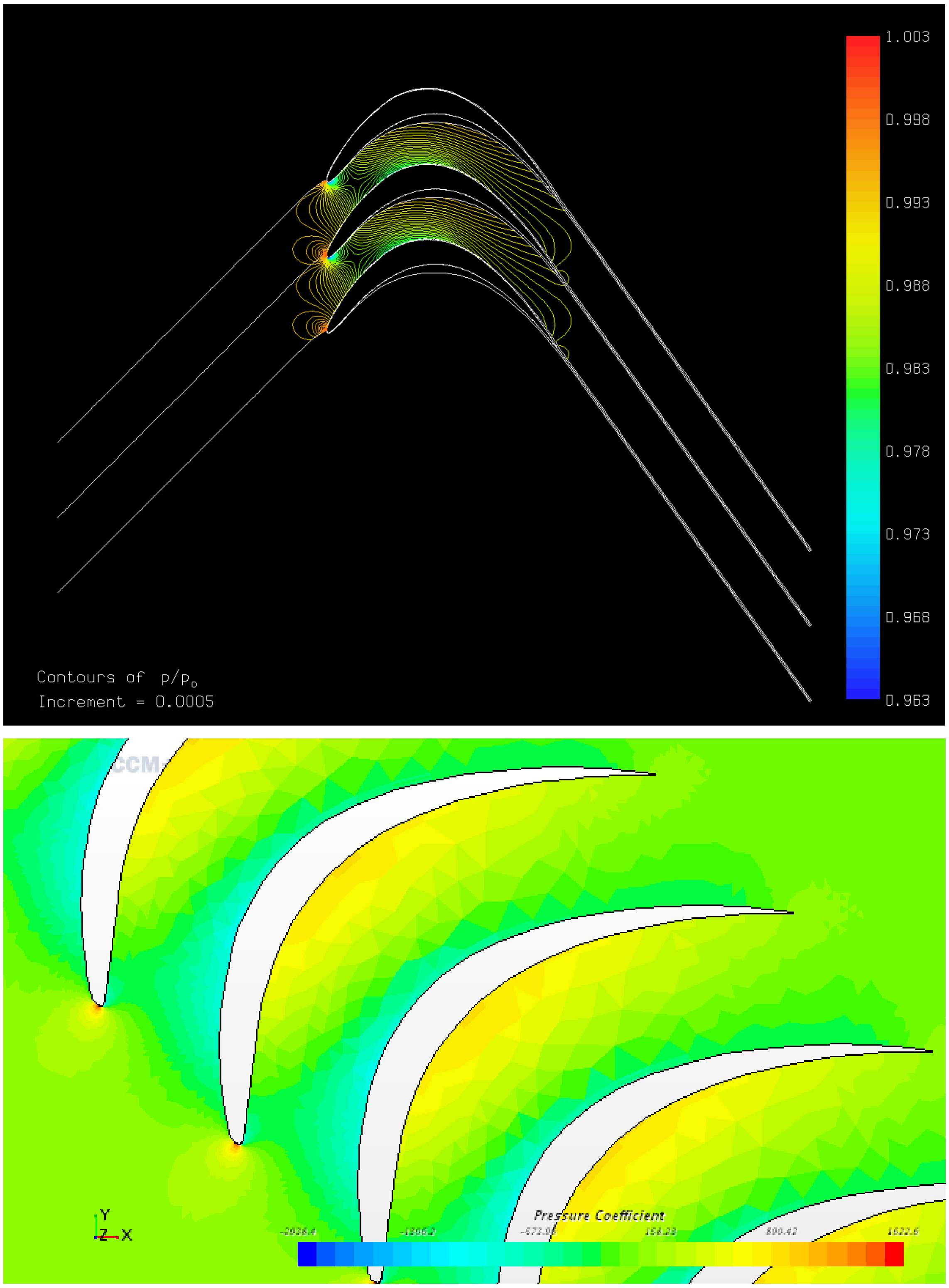


Figure 3.23: Detail of pressure coefficient from STAR-CCM+ solution of non-expanding case compared to MISES contours

3.2.2 Original vane, ER=1.2

Creating another simulation to represent the same vanes in an expanding corner helps to clarify some of the issues seen in the first simulation. Figure 3.24 shows the velocity profile of the expanded tunnel with the location of the line probe and inlet probe highlighted. Results of this simulation also follow the expected patterns of a vane in this configuration. The flow behind the corner has been decelerated by the expansion and contains some velocity defects in the wake. As with the losses predicted in the previous case, MISES predicts losses more than twice that of those predicted with STAR-CCM+. Comparison of these values gave a promising sign, however, as the ratio between the Star and MISES values were nearly the same for both cases and the percentage difference between the MISES values was extremely similar. This similarity in change gives a good indication that though the losses are not the same, the trends in losses likely are. Additionally, this change supports the veracity of the trend seen during the parameter sweep where fewer losses were incurred by larger expansions. Thus, the conclusions that this result is either physical or a quirk of the pressure loss measurement have additional credence.

Case	**MISES Losses**	**STAR-CCM+ Losses**
Expanded	0.1776	0.066
Non-Expanded	0.1148	0.042
Percent Change	35.4%	36.4%

Table 3.4: Comparison of MISES and STAR-CCM+ loss values

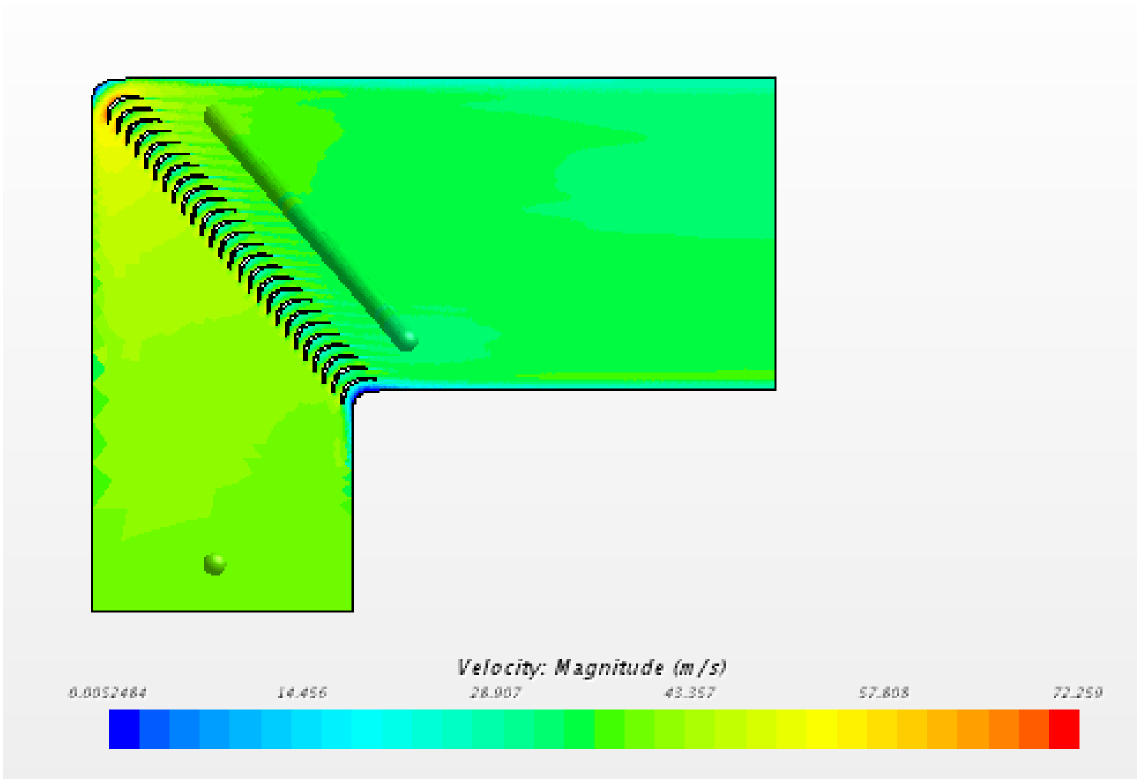


Figure 3.24: STAR-CCM+ solution of expanding case with current GLMWT turning vane

Also similar to the previous case is the resemblance of the pressure contours. Figure 3.25 demonstrates these similarities. The approximate locations of the stagnation point are close, however it should be noted that MISES predicts a stagnation point further towards the upper surface of the turning vane. This distinction may be exacerbated by more extreme expansions, but further research is again needed to determine an exact cause.

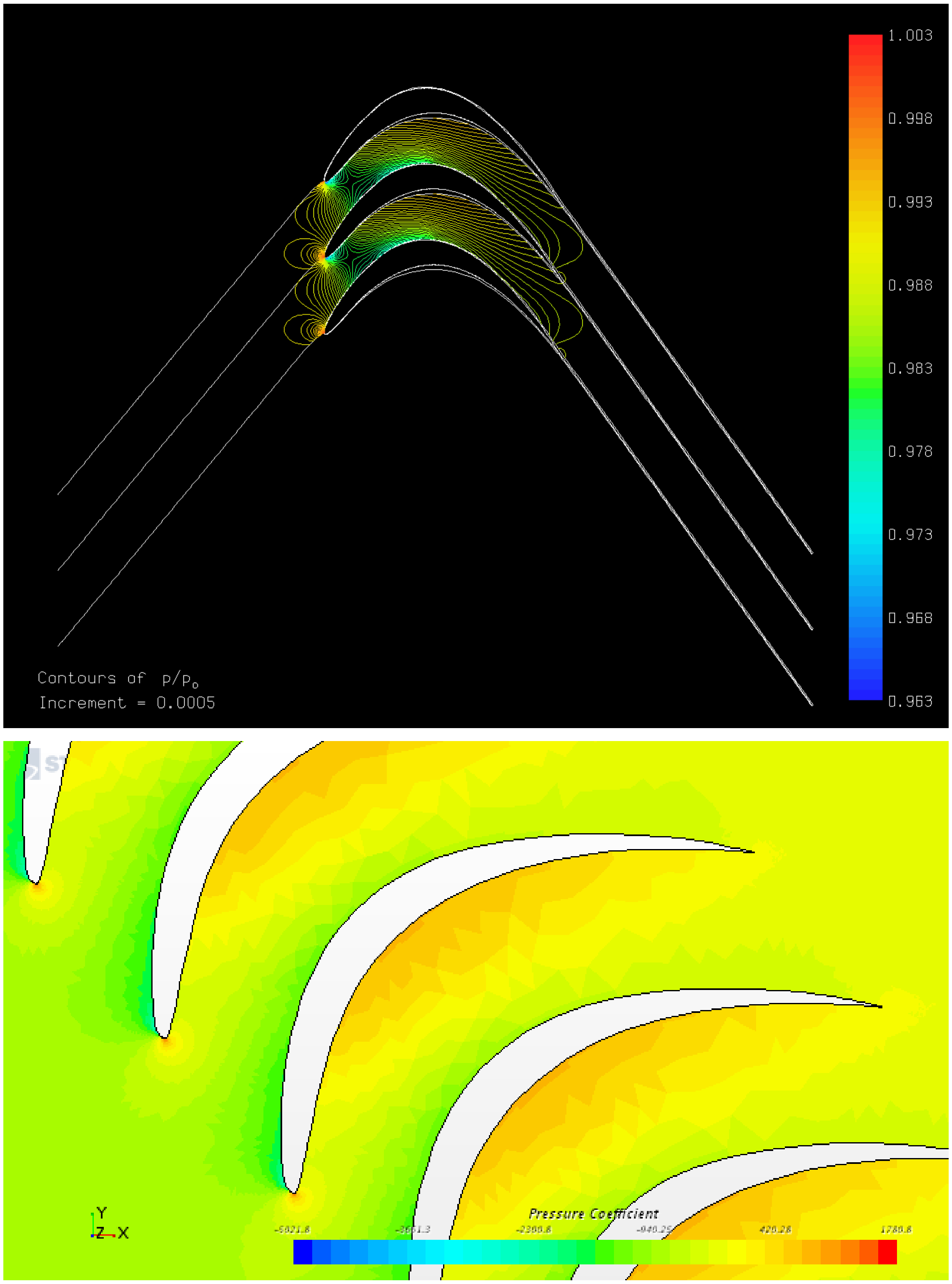


Figure 3.25: Detail of pressure coefficient from STAR-CCM+ solution of expanding case compared to MISES contours

3.2.3 Optimized vane, ER=1.2

Simulating the vane with the best fit provides an excellent warning for future users of this tool. The vane obtained with the $MOptiGA()$ function during the constant solidity optimization at an expansion ratio of 1.2 had excellent flow parameters that pushed the bounds of credibility. Simulating the vane with StarCCM+ proves this skepticism to be warranted. Figure 3.26 illuminates the difficulty with the optimized vane.

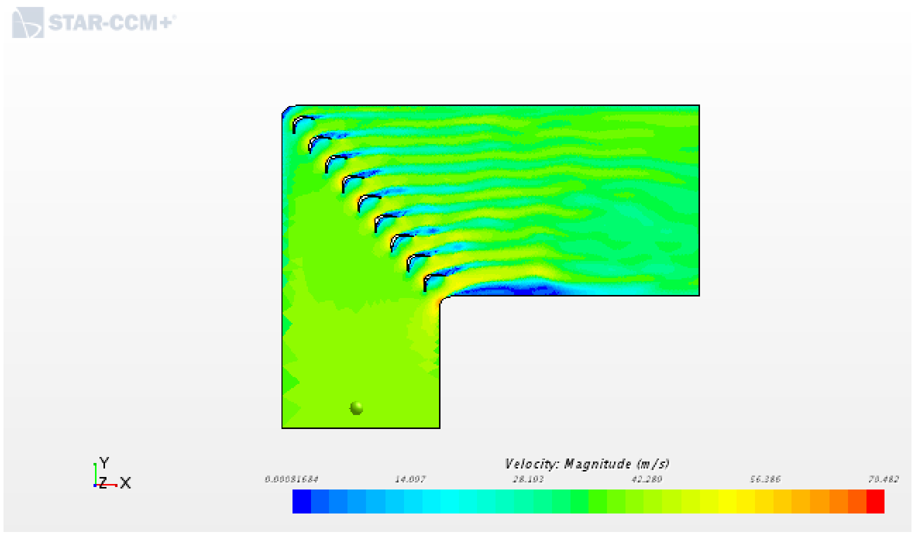


Figure 3.26: STAR-CCM+ solution of expanding case using optimized turning vane

Though the MISES prediction of this vane indicates little to no separation, the separation was so great that the STAR solution could not converge properly. Where the first two cases were iterated until their residuals dropped by four to five orders of magnitude, the residuals for this case only dropped three orders of magnitude before beginning to oscillate.

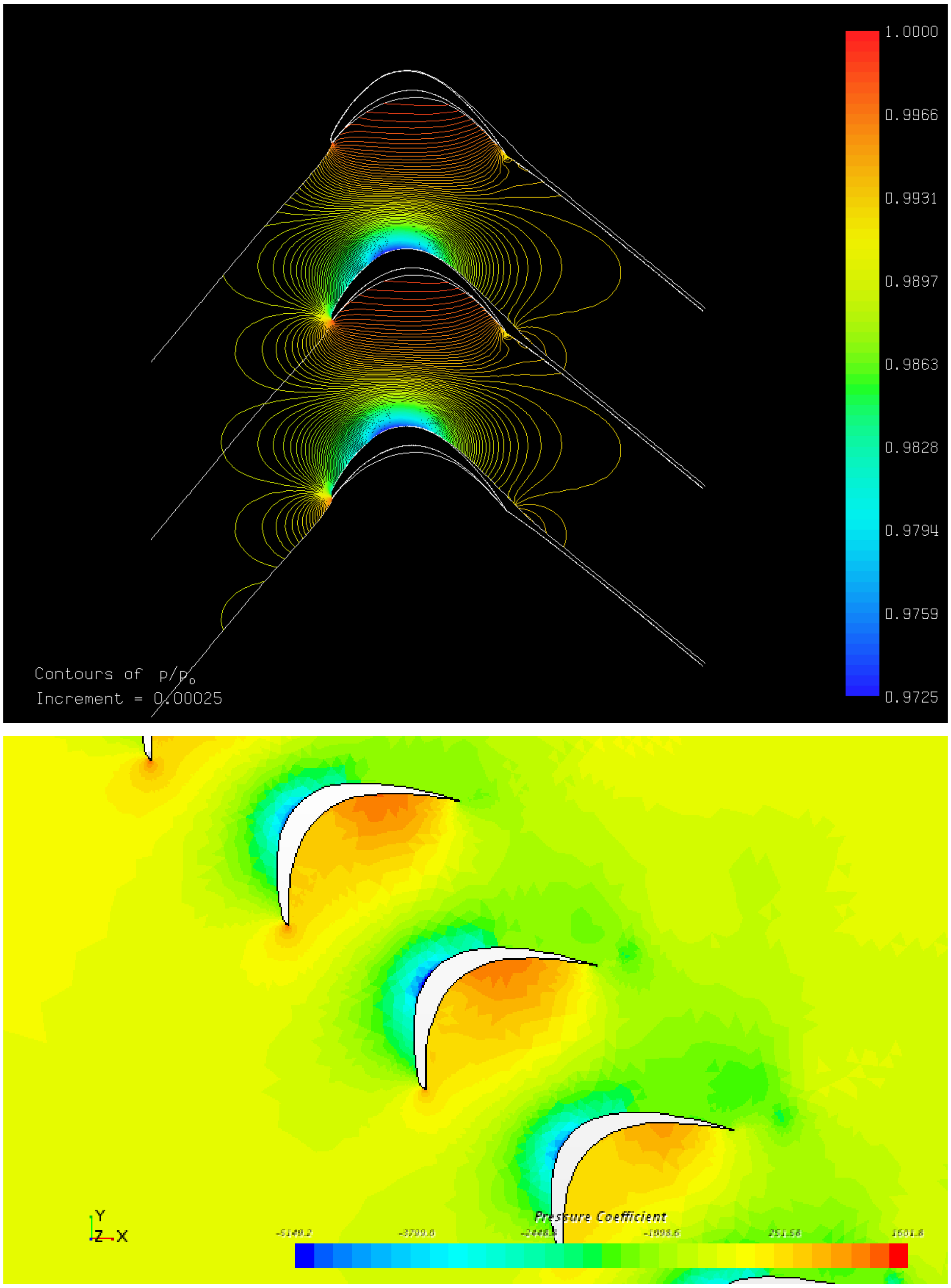


Figure 3.27: Detail of pressure coefficient from STAR-CCM+ solution of optimized expanding case compared to MISES contours

Though the contours shown in figure 3.27 seem to roughly match, the parameters queried by the optimizer give no indication of the large shedding phenomena that occur. This shedding results in a predicted loss of 0.214. Assuming that the trends from the other two cases remain true, this corresponds to a loss of 0.579 in MISES, much higher than what MISES actually predicted. For this reason, any users attempting to optimize cascades with large vane gaps should be wary of the results they obtain. It is possible that this phenomena could be predicted with another parameter in MISES, but no such parameter is known at present.

Chapter 4: Conclusions and Future Work

Overall the results of this research were promising. The short simulation time and relatively simple operation of the program achieved most of the goals set out at the beginning of this paper. The vanes generated, though certainly not the finalized configuration for use in the tunnel, show the ability of this optimization process to contribute to a design process. Furthermore, most of the problems encountered in the application of this tool could be overcome with further testing and more careful usage.

4.1 Optimization Efficacy

As discussed in the results section, the vane designs output by each of the optimizers improved on the initial designs they were given for each particular case. With proper weighting, this improvement created vanes that largely avoided separation and lowered total pressure losses. In some cases, the optimizers reduced pressure losses by as much as 18%. In addition, mean separation distance was reduced by as much as 53%. While most cases could not alter the turning angle of the flow significantly, STAR-CCM+ modelling suggests that the interactions with the tunnel walls may reduce the impact of this parameter.

Despite this demonstration of ability, the optimizers were still fairly slow. This lack of speed could be partially offset by running multiple cases at once, but this increases the likelihood of errors caused by both instances altering the same files. To avoid the need for this, the efficiency of the optimizers stands to be improved. Additionally, the manner in which variables are assigned in MATLAB caused numerous errors at the end of runs which lost significant computational time. Though further operation with the programs created for this thesis should avoid most of these errors, any further alterations or attempts to improve the program may encounter these frustrating setbacks.

4.2 Possible Improvements

Even with the success of many portions of this work, numerous improvements could be made. Primary among these is an alteration of the penalty function. As the penalty function was effectively of zeroth order, optimizers relying on derivatives to generate directions will be unable to escape from non-feasible design spaces. This drawback can also affect directed optimizers operating in the feasible design space as the finite difference used in the definition of direction can hit one of these boundaries and push the design in an undesirable direction. Additionally, genetic algorithms and other non-directed optimizers can be fooled into thinking an optimum has been reached when they are given a non-feasible starting point and do not happen to find a viable design.

Other improvements include ease of use features such as a graphical vane creation tool. This would allow more initial designs to be considered, especially designs not based on a current design. Though the current GLMWT turning vane provided a good starting point, having the ability to quickly generate radically different vanes could improve the ability of the optimizer to be agnostic of the initial design. Further alterations of airfoil definition could come from methods similar to the Bezier-PARSEC method described in **??**. Tying control points to airfoil geometry parameters like leading edge thickness or trailing edge angle could avoid some of the strange design features seen in the optimizations presented here. In a simpler alteration, converting the airfoil definitions to a single high order curve could allow more gross alteration of characteristics like chord length, pitch, and camber.

Finally, new optimization codes could be tested for their viability. The three functions used in this research provided good results, but the MATLAB optimization suite is broad and contains a plethora of additional options. Further experimentation could produce a significantly faster or more effective optimizer than the one detailed here. In fact, simply altering the existing codes to take advantage of parallelism could drastically reduce optimization times.

4.3 Future Work

Beyond improvements to the program, many possibilities exist for future works to build on this research. Most apparent is an extension to three dimensions. Though a two dimensional expansion will be sufficient for many wind tunnels, the

GLMWT requires a more three dimensional expansion to limit the amount of alterations needed to implement an expanding corner. It is likely that a modification of the ISES parameters to include non-axial flow could replicate the action of 3D wind tunnel turning vanes. As such, optimizing the vanes with various amounts of this non-planar flow could create an optimal vane profile for all stations along the height of the corner or a number of profiles to be used as cross sections at different stations.

Even more ambitious related works could replace MISES altogether. If a new code were found or created, the geometry of the entire corner could be investigated. This would allow the corner geometry to be better incorporated and investigated with the vane geometry, but would also require a much larger simulation as periodic boundaries could no longer be used to limit cell count.

Finally, a full expanding corner redesign using the tool in its current state could further solidify the usefulness of such a design tool. This work would require significantly more study of the flow features within the expanding corner their representation in MISES, but would ultimately lead to an improvement that could make a tangible change to any aerodynamic research institution.

Appendix A: MATLAB code

```
function M=BernMat(n)
%This function generates a Bernstein Matrix for a polynomial of n values
    in
%order to use with a Bezier curve formulation. The matrix is created by
%applying coefficients from ascending levels of pascal’s triangle to the
%values from the nth level along the diagonals.

M=zeros(n); %Initialize the matrix

%Write Pascal’s triangle to a matrix
tri=zeros(n,n); tri(1,1)=1;
for i=2:n
    tri(i,1)=1;
    for j=2:i
        tri(i,j)=tri(i-1,j-1)+tri(i-1,j);
    end
end

%Sweep through matrix to apply proper elements
for k=0:n-2 %Sweep diagonals
    for i=1:n %Sweep rows
        for j=1:i %Sweep columns
            if j==1 %Load left side
                M(i,j)=(-1)^(i+1)*tri(n,i);
            elseif j==i-k %Load diagonals
                M(i,j)=M(k+1,1)*tri(n-k,j);
            end
        end
    end
end
end
```

```
function [X,Y]=Bez(xp,yp,n,Type,Split)
%This function converts bezier curve control points to the corresponding
%cartesian curves.

%Check Inputs
switch nargin
    case 5
        %Accept all inputs
    case 4
        %Generate generic splits
        Split=[4,4,4];
    case 3
        Type='Smooth'; %Default to smooth
        %Generate generic splits
        Split=[4,4,4];
    case 2
        %Set default values if not specified
        n=100;
        Type='Smooth'; %Default to smooth
        %Generate generic splits
        Split=[4,4,4];
    otherwise
        error('Not enough arguments')
end

%Convert to column vectors if not already
if iscolumn(xp)~=1
    xp=xp';
end
if iscolumn(yp)~=1
    yp=yp';
end

if strcmp(Type,'Split')
    %Split curve into a number of curves of order Split(i)-1
    v=1; %Initialize index variable to prevent point doubling
    %Populate segment matrix based on split order matrix
    S2(1)=1;
    for k=2:length(Split)+1
        S2(k)=S2(k-1)+Split(k-1)-1;
    end
    %Find point scaling
    Order=Split./length(xp);
    for k=1:length(Split) %Calculate line segments
        clear T t %Reset size of T and t
        t=linspace(0,1,Order(k)*n); %Define t
        B=BernMat(Split(k)); %Define Bernstein Matrix
        for i=1:length(t)
            for j=0:Split(k)-1
                T(1,j+1)=t(i)^j; %Populate interpolation matrix
            end
            if k~=1 && i==1 %Check for segment start to avoid repeat point
                v=v-1;
            end
            X(v)=T*B*xp(S2(k):S2(k+1)); %Find vth value of X
            Y(v)=T*B*yp(S2(k):S2(k+1)); %Find vth value of Y
            v=v+1;
        end
    end
else %Default to single smooth curve (not ideal, breaks above ~35 points)
    t=linspace(0,1,n); %Define t
    B=BernMat(length(xp)); %Define Bezier Matrix
    for i=1:n
        for j=0:length(xp)-1
            T(1,j+1)=t(i)^j; %Populate interpolation matrix
        end
        X(i)=T*B*xp; %Find ith value of X
        Y(i)=T*B*yp; %Find ith value of Y
    end
end

end
```

```
function Input=InStructSet(Name)
%This is a convenience function to correctly set up the input structure
%required by other functions. Hard-coded values can be easily changeed
%within the function to ensure proper run parameters.

%Expansion ratio
er=1.2; %Equivalent to tan(alpha)
%Curve parameters
Input.n=250;
Input.Type='Split';
Input.Split=[6 6 6 6];
Input.FileName=Name;
%ISES parameters (Named as in MISES user guide)
Input.MINLin=0.0982*er;
Input.MOUTin=0.0;
Input.P1PTin=((1+0.2*Input.MINLin^2)^(-1.4/0.4));
Input.SINLin=er; %Inlet angle equivalent to ER
Input.SOUTin=tan(atan(er)+pi/2); %Calculate desired tangent of outlet
Input.CHINL=1.0;
Input.CHOUT=1.0;
Input.PITCH=0.3;
Input.XINLin=-0.5;
Input.XOUTin=1.5;
Input.REYNin=1.476e6*er;
Input.NCrit=4;

end
```

```
function VaneBuild(xp,yp,c,Input)
%This function takes a filename and a structure. The structure contains
%two vectors, a number of points, and a specification of the type of
%curve desired (Split or not), inlet tangent, outlet tangent, inlet and
%outlet locations. The fuction then calculates the proper curves and
%writes to a blade.xxx file with the proper structure.

%Calculate the shape
[X,Y]=Bez(xp,yp,Input.n,Input.Type,Input.Split);
%Check for self intersection
[xint,~,~]=selfintersect(X,Y);
if length(xint)>1
    %Throw error if intersecting
    error('Vane is self intersecting')
end
%Check for vane intersection by traversing backwards across first vane to
%see if upper surface intersects lower surface of next vane.
if max(fliplr(Y)>Y+c)==1
    %Throw error if solidity too low
    error('Solidity too low for vane')
end
%Open file if valid vane
F=fopen(['blade.',Input.FileName],'w');
%Print name and flow parameters
fprintf(F,[Input.FileName,'\n',num2str(Input.SINLin,'%1.4f'),' ',...
    num2str(Input.SOUTin,'%1.4f'),' ',num2str(Input.CHINL,'%1.4f'),' ',...
    num2str(Input.CHOUT,'%1.4f'),' ',num2str(c,'%1.4f'),'\n']);
%Print coordinates to file
for i=1:length(X)
    fprintf(F,[num2str(X(i),'%1.5f'),' ',num2str(Y(i),'%1.5f'),'\n']);
end

%Close file
fclose(F);

end
```

```
function InputSetup(Input)
%This function sets up the parameter file for Mises runs. The function
%requires a filename and an input structure. The Input must have fields
%for inlet mach, inlet static pressure, inlet angle (tan(theta)), inlet
%distance, outlet mach, outlet angle, outlet distance, reynolds number,
%and Ncrit.

%Open Input File
f=fopen(['ises.',Input.FileName],'w');

%Select between sharp and non sharp leading edge
%Print variable constants line
%Line currently represents inlet angle, exit slope, exit pressure, and
%inlet mach
fprintf(f,' 1 2 15 6 5\n');
%Print constraint constants line
%Line currently represents inlet angle, outlet slope, TE Kutta, inlet
%Mach, inlet pressure ratio, inlet Reynold's number
fprintf(f,' 1 4 15 6 3\n');

%Print inputs
%Inlet conditions (Mach, pressure ratio, angle, distance)
fprintf(f,[' ',num2str(Input.MINLin,'%1.4f'),' ',...
    num2str(Input.P1PTin,'%1.4f'),' ',num2str(Input.SINLin,'%1.4f'),...
    ' ',num2str(Input.XINLin,'%1.4f'),'\n']);
%Outlet conditions (Mach, pressure ratio, angle, distance)
fprintf(f,[' ',num2str(Input.MOUTin,'%1.4f'),' 0.0 ',...
    num2str(Input.SOUTin,'%1.4f'),'
        ',num2str(Input.XOUTin,'%1.4f'),'\n']);
%Splitter parameters
fprintf(f,[' 0.0 1.0\n']);
%Viscous parameters (Reynold's, ncrit for transition model)
Rey=num2str(Input.REYNin,'%1.3E');%Eliminate illegal character
fprintf(f,[' 0.',Rey(1),Rey(3:6),Rey(8:end),' ',...
    num2str(Input.NCrit,'%1.4f'),'\n']); %ncrit is freestream turbulence
        if negative
%Forced transition locations (ignored if >=1)
fprintf(f,[' 1.0 1.0\n']);
%Isentropy and dissipation (model selection 1-4, critical mach, artificial
%dissipation)
fprintf(f,[' 3 0.970 1.00\n']);

%Close file
fclose(f);

%Write spec File for use by POLAR
f=fopen(['spec.',Input.FileName],'w');
```

```
fprintf(f,[' 1\n ',num2str(Input.SINLin)]);
fclose(f);

end
```

```matlab
function [Ploss,Sep,Turn]=DataRead(FileName)
%This function reads the output files of a Mises run to output the needed
%data.

%Open Blade File
fb=fopen(['blade.',FileName],'r');
%Open Polar File
fp=fopen(['polar.',FileName],'r');

%Load inlet angle
for i=1:2
    line=fgetl(fb);
    if i==2
        Inlet=atand(str2double(line(1:7)));
    end
end
%Read loss and outlet angle from polar file
for i=1:8
    line=fgetl(fp);
    if i==8
        Sout=atand(str2double(line(12:20)));
        Ploss=str2double(line(75:84));
    end
end

%Calculate total turning angle (degrees)
Turn=180-(Inlet-Sout);

%Call SepVal to output mean separation distance
Sep=SepVal(FileName);

%Close files
fclose(fp); fclose(fb);

end
```

```
function Fit=DataReadFit(FileName,w)
%This function reads the output file of a Mises run to output the needed
%data then converts data to fitness value using given weighting vector.

%Open blade file
fb=fopen(['blade.',FileName],'r');
%Open Polar File
fp=fopen(['polar.',FileName],'r');

%Load inlet angle
for i=1:2
    line=fgetl(fb);
    if i==2
        Inlet=atand(str2double(line(1:7)));
    end
end
%Read loss and outlet angle from polar file
for i=1:8
    line=fgetl(fp);
    if i==8
        Sout=atand(str2double(line(12:20)));
        Ploss=str2double(line(75:84));
    end
end

%Calculate total turning angle (degrees)
Turn=180-(Inlet-Sout);

%Call SepVal to output mean separation distance
Sep=SepVal(FileName);


%Calculate fitness function
if isnan(Ploss)==1
        Fit=500;
elseif Turn<=90 %prioritize overturning to 95 degrees
    Fit=w(1)*Ploss+w(2)*Sep+w(3)*abs(95-Turn);
else %Punish underturning (disabled if w(4)=0)
    Fit=w(1)*Ploss+w(2)*Sep+(w(3)+w(4))*abs(90-Turn);
end

%Close files
fclose(fb); fclose(fp);

end
```

```
function [sepdis]=SepVal(FileName)
%This function reads the blade and field files for a given case and uses
%interpolation to determine the average distance from the stagnation
%streamlines to the blade surface. This provides a quantitative measure of
%separation.

%Read blade coordinates from blade file
bfi=fopen(['blade.',FileName],'r');
i=1;
while ~feof(bfi)
    line=fgetl(bfi);
    if i==2
        c=str2double(line(30:end));
    elseif i>=3
        Bx(i-2,1)=str2double(line(1:7));
            By(i-2,1)=str2double(line(9:end));
    end
    i=i+1;
end
%Split surfaces and invert upper surface to correct orientation
dx=Bx(2:end)-Bx(1:end-1);
le=find(dx>0,1);
Ux=flipud(Bx(1:le-2)); Uy=flipud(By(1:le-2));
Lx=Bx(le+2:end); Ly=By(le+2:end);

%Close file
fclose(bfi);

%Read field file to place streamline coordinates into matrix
k=1; i=1; j=0; check=0;
f=fopen(['field.',FileName],'r');
while ~feof(f)
    line=fgetl(f);
    if k==1 %Skip header
        k=k+1;
    elseif isempty(line) %Find start of new streamline
        check=1;
        j=j+1; i=1;
    else %Read values into matrix
        Sx(i,j)=str2double(line(1:14)); Sy(i,j)=str2double(line(15:27));
        i=i+1; check=0;
    end
end

%Close file
fclose(f);
```

```
%Interpolate y distance between stream and blade for first and last
%streamlines (Upper and lower stagnation streamlines)
%Find approximate blade coordinates within streamline
les=find(Sx(:,1)>0.1);
tes=find(Sx(:,1)>0.9);
%Find vertical distance for upper surface
dyu=abs(Sy(les:tes,1)-interp1(Ux,Uy-c,Sx(les:tes,1)));
%Find approximate blade coordinates within streamline
les=find(Sx(:,end)>0.1);
tes=find(Sx(:,end)>0.9);
%Find vertical distance for lower surface
dyl=abs(Sy(les:tes,end)-interp1(Lx,Ly,Sx(les:tes,end)));

%Average values and sum
sepdis=mean(dyu)+mean(dyl);

end
```

```
function [ploss,sep,turn]=MISES(X,Y,c,Input)
%This function is a simple wrapper for the other functions which allows
%them to be called within Matlab's optimization functions. The function
%checks if an input file has been prepared to prevent function
%repetition.

try
    % Clear any previous field and polar files to prevent unwanted
    % data carryover on failed runs
    delete(['field.',Input.FileName]);
    delete(['polar.',Input.FileName]);
    delete(['idat.',Input.FileName]);

    % Build vane
    VaneBuild(X,Y,c,Input);

    % Set up ises if it doesn't exist
    if ~isfile(['ises.',Input.FileName])
        InputSetup(Input);
    end

    % Run Mises Solver
    MISESEval(Input);
    % Pull values from outputs
    [ploss,sep,turn]=DataRead(Input.FileName);
catch %Check for errors and allow larger operation to continue
    % Set values to NaN to indicate failure
    ploss=NaN;
    sep=NaN;
    turn=NaN;
end
end
```

```matlab
function fit=MISESFit(X,Y,c,Input,w)
%This function is a simple wrapper for the other functions which allows
%them to be called within Matlab's optimization functions. The function
%assumes that an input file has been prepared to prevent function
%repetition.

try
    % Clear any previous data files to prevent unwanted data
    % carryover in failed runs
    delete(['field.',Input.FileName]);
    delete(['polar.',Input.FileName]);
    delete(['idat.',Input.FileName]);

    %Build the Vane
    VaneBuild(X,Y,c,Input);

    % Set up ises if it doesn't exist
    if ~isfile(['ises.',Input.FileName])
        InputSetup(Input);
    end

    % Run Mises Solver
    MISESEval(Input);
    % Pull fitness value from output
    fit=DataReadFit(Input.FileName,w);
catch %Check for errors to allow larger functions to continue
    % Set loss to large value as a form of penalty function
    fit=500;
end

end
```

```
function fit=MISESFit2(In,Input,fixc,w,Xin,Yin,Cin,Xend,Ybeg,Yend)
%This function is a simple wrapper for the other functions which allows
%them to be called within Matlab's optimization functions. The function
%assumes that an input file has been prepared to prevent function
%repetition. This function is used when optimization functions allow
%additional inputs to be passed directly to the output function

%Parse inputs
X=[Xin(1),In(1:Xend),Xin(end)]';
Y=[Yin(1),In(Ybeg:Yend),Yin(end)]';
if fixc==1
    c=Cin;
else
    c=In(end);
end
%Apply Smoothness Constraint
for i=1:length(Input.Split)-1
    if i==1
        S=Input.Split(i);
    else
        S=sum(Input.Split(1:i))-(i-1);
    end
    Y(S+1)=Y(S)+(X(S+1)-X(S))*(Y(S)-Y(S-1))/(X(S)-X(S-1));
end


try
    % Clear any previous data files to prevent unwanted data
    % carryover in failed runs
    delete(['field.',Input.FileName]);
    delete(['polar.',Input.FileName]);
    delete(['idat.',Input.FileName]);

    %Build the Vane
    VaneBuild(X,Y,c,Input);

    % Set up ises if it doesn't exist
    if ~isfile(['ises.',Input.FileName])
        InputSetup2(Input);
    end

    % Set up Grip Parameters if it doesn't exist
    if ~isfile(['gridpar.',Input.FileName])
        GridPar(Input);
    end

    % Run Mises Solver
    MISESEval(Input);
    % Pull value from output
    fit=DataReadFit(Input.FileName,w);
catch %Check for errors to allow larger functions to continue
    % Set loss to large value as a form of penalty function
    fit=500;
end
end
```

```
function [Xout,Yout,cout,fit]=MOptiUncon(Xin,Yin,Cin,Input,tol,h,fixc,w)
%This function performs an optimization using the fminuncon() function to
%generate a vane design which best satisfies the fitness conditions given
%by DataReadFit(). Solidity may be fixed using fixc=1 and a weighting
%vector w of length 4 is required. tol and h set convergence tolerance and
%finite difference step size respectively.

%Build First vane
VaneBuild(Xin,Yin,Cin,Input);
%Set up ises file
InputSetup(Input);
%Pause to allow Gridpar generation with ISET (if necessary)
pause

%Set initial guess
In=[Xin(2:end-1);Yin(2:end-1);Cin];
%Determine beginning and end points for variables in vector
Xend=length(Xin)-2; Ybeg=Xend+1; Yend=Xend+length(Yin)-2;
%Set optimization options
Opt=optimoptions('fminunc','DiffMaxChange',5,'FiniteDifferenceStepSize',h,...
    'MaxFunctionEvaluations',1e3*Xend*2,'OptimalityTolerance',tol);

%Perform optimization
[Out,fit]=fminunc(@Call,In,Opt);

%Output data and plot new vane
Xout=[Xin(1);Out(1:Xend);Xin(end)];
Yout=[Yin(1);Out(Ybeg:Yend);Yin(end)];
if fixc==1
    cout=Cin;
else
    cout=Out(end);
end
%Apply smoothness constraint
for i=1:length(Input.Split)-1
    if i==1
        S=Input.Split(i);
    else
        S=sum(Input.Split(1:i))-(i-1);
    end
Yout(S+1)=Yout(S)+(Xout(S+1)-Xout(S))*(Yout(S)-Yout(S-1))/(Xout(S)-Xout(S-1));
end

%Display optimized vane
[X,Y]=Bez(Xout,Yout,Input.n,Input.Type,Input.Split);
[Xor,Yor]=Bez(Xin,Yin,Input.n,Input.Type,Input.Split);
figure
```

```
plot(X,Y,'b-',Xor,Yor,'r--',X,Y+cout,'b-','LineWidth',2.0)
if fixc==1
    title(['Optimized turning vane for ER=',num2str(Input.SINLin),...
        ', constant solidity'])
else
    title(['Optimized turning vane for ER=',num2str(Input.SINLin),...
        ', variable solidity'])
end
xlabel('X/c')
ylabel('Y/c')
legend('Optimized Vane','Initial Vane')

    %Embed function to allow easier handling of variables
    function fit=Call(Var)
        %Parse inputs
        Xp=[Xin(1);Var(1:Xend);Xin(end)];
        Yp=[Yin(1);Var(Ybeg:Yend);Yin(end)];
        %Enforce smoothness
        for i=1:length(Input.Split)-1
            if i==1
                S=Input.Split(i);
            else
                S=sum(Input.Split(1:i))-(i-1);
            end
        Yp(S+1)=Yp(S)+(Xp(S+1)-Xp(S))*(Yp(S)-Yp(S-1))/(Xp(S)-Xp(S-1));
        end
        %Perform run and calculate fitness
        if fixc==1
            fit=MISESFit(Xp,Yp,Cin,Input,w);
        else
            fit=MISESFit(Xp,Yp,Var(end),Input,w);
        end
    end
end
```

```
function
    [Xnu,Ynu,c,fval,exitflag]=MOptiSearch(Xin,Yin,Cin,Input,tol,fixc,w)
%This function performs an optimization using the fminsearch() function to
%generate a vane design which best satisfies the fitness conditions given
%by DataReadFit(). Solidity may be fixed using fixc=1 and a weighting
%vector w of length 4 is required. tol sets a convergence tolerance.

%Build First vane
VaneBuild(Xin,Yin,Cin,Input);
%Set up ises file
InputSetup(Input);
%Pause to allow Gridpar generation with iset (if necessary)
pause

%Define input vector
if isrow(Xin)==1
    In=[Xin(2:end-1),Yin(2:end-1),Cin];
else
    In=[Xin(2:end-1);Yin(2:end-1);Cin]';
end
%Determine beginning and end points for variables in vector
Xend=length(Xin)-2; Ybeg=Xend+1; Yend=Xend+length(Yin)-2;

%Set options
Opt=optimset('MaxFunEvals',5e4,'TolX',tol,'PlotFcns',@optimplotfval);
%Call optimization function
[Out,fval,exitflag]=fminsearch(@MISESFit2,In,Opt,Input,fixc,w,Xin,Yin,Cin,Xend,Ybeg,Yenc
%Find Output
Xnu=[Xin(1),Out(1:length(Xin)-2),Xin(end)]';
Ynu=[Yin(1),Out(length(Xin)-1:end-1),Yin(end)]';
if fixc==1
    c=Cin;
else
    c=Out(end);
end
%Enforce smoothness
for i=1:length(Input.Split)-1
    if i==1
        S=Input.Split(i);
    else
        S=sum(Input.Split(1:i))-(i-1);
    end
    Ynu(S+1)=Ynu(S)+(Xnu(S+1)-Xnu(S))*(Ynu(S)-Ynu(S-1))/(Xnu(S)-Xnu(S-1));
end

%Display optimized vane
[X,Y]=Bez(Xnu,Ynu,Input.n,Input.Type,Input.Split);
```

```
[Xor,Yor]=Bez(Xin,Yin,Input.n,Input.Type,Input.Split);
figure
plot(X,Y,'b-',Xor,Yor,'r--',X,Y+c,'b-','LineWidth',2.0)
if fixc==1
    title(['Optimized turning vane for ER=',num2str(Input.SINLin),...
        ', constant solidity'])
else
    title(['Optimized turning vane for ER=',num2str(Input.SINLin),...
        ', variable solidity'])
end
xlabel('X/c')
ylabel('Y/c')
legend('Optimized Vane','Initial Vane')

end
```

```
function [Xnu,Ynu,c,fval,exitflag]=MOptiGA(N,Xin,Yin,Cin,Input,fixc,w)
%This function performs an optimization using the ga() function to
%generate a vane design which best satisfies the fitness conditions given
%by DataReadFit(). Solidity may be fixed using fixc=1 and a weighting
%vector w of length 4 is required.

%Set correct number of variables
if fixc==1
    Nvar=N*2;
    IMat(1,:)=[Xin(2:end-1)',Yin(2:end-1)'];
else
    Nvar=N*2+1;
    IMat(1,:)=[Xin(2:end-1)',Yin(2:end-1)',Cin];
end
%Generate initial population with bounded random alterations
for i=2:(N*5)
    if fixc==1
        IMat(i,:)=[Xin(2:end-1)'-0.01+0.02.*rand(1,N),...
            Yin(2:end-1)'-0.01+0.02.*rand(1,N)];
    else
        IMat(i,:)=[Xin(2:end-1)'-0.01+0.02.*rand(1,N),...
            Yin(2:end-1)'-0.01+0.02*rand(1,N),Cin+rand(1)];
    end
end

%Set up ises file
InputSetup(Input);

%Set options
Opt=optimoptions('ga','InitialPopulationMatrix',IMat,'PlotFcn',...
    'gaplotbestf');
%Call fitting function
[Out,fval,exitflag]=ga(@Call,Nvar,Opt);

%Find Output
Xnu=[Xin(1),Out(1:N),Xin(end)]';
if fixc==1
    c=Cin;
    Ynu=[Yin(1),Out(N+1:end),Yin(end)]';
else
    c=Out(end);
    Ynu=[Yin(1),Out(N+1:end-1),Yin(end)]';
end
%Enforce Smoothness
for i=1:length(Input.Split)-1
    if i==1
        S=Input.Split(i);
    else
        S=sum(Input.Split(1:i))-(i-1);
    end
    Ynu(S+1)=Ynu(S)+(Xnu(S+1)-Xnu(S))*(Ynu(S)-Ynu(S-1))/(Xnu(S)-Xnu(S-1));
end

%Display optimized vane
[X,Y]=Bez(Xnu,Ynu,Input.n,Input.Type,Input.Split);
[Xor,Yor]=Bez(Xin,Yin,Input.n,Input.Type,Input.Split);
figure
plot(X,Y,'b-',Xor,Yor,'r--',X,Y+c,'b-','LineWidth',2.0)
if fixc==1
    title(['Optimized turning vane for ER=',num2str(Input.SINLin),...
        ', constant solidity'])
else
    title(['Optimized turning vane for ER=',num2str(Input.SINLin),...
        ', variable solidity'])
end
xlabel('X/c')
ylabel('Y/c')
legend('Optimized Vane','Initial Vane')

    %Embed function to allow easier handling of variables
    function fit=Call(In)
        %Parse inputs
        X=[Xin(1),In(1:N),Xin(end)]';
        Y=[Yin(1),In(N+1:N*2),Yin(end)]';
        if fixc==1
            c=Cin;
        else
            c=In(end);
        end
        %Enforce Smoothness
        for i=1:length(Input.Split)-1
            if i==1
                S=Input.Split(i);
            else
                S=sum(Input.Split(1:i))-(i-1);
            end
            Y(S+1)=Y(S)+(X(S+1)-X(S))*(Y(S)-Y(S-1))/...
                (Y(S)-Y(S-1));
        end
        %Perform run and calculate fitness
        fit=MISESFit(X,Y,c,Input,w);
    end
end
```

```
function [Xnu,Ynu,fval,exitflag]=VaneCalc(Xdes,Ydes,Xin,Yin,Input,tol)
%This function performs an optimization using the VaneFit function to
%generate a vane design which best approximates the control points
    required
%to make a curve which passes through the given coordinates.

%Define input vector
if isrow(Xin)==1
    In=[Xin(2:end-1),Yin(2:end-1)];
else
    In=[Xin(2:end-1);Yin(2:end-1)]';
end

%Set options
Opt=optimset('MaxFunEvals',1e5,'TolX',tol,'PlotFcns',@optimplotfval);
%Call fitting function
[Out,fval,exitflag]=fminsearch(@VaneFit2,In,Opt,Xdes,Ydes,Input,1);
%Find Output
Xnu=[Xdes(1),Out(1:length(Xin)-2),Xdes(end)]';
Ynu=[Ydes(1),Out(length(Xin)-1:end),Ydes(end)]';
%Enforce smoothness
for i=1:length(Input.Split)-1
    if i==1
        S=Input.Split(i);
    else
        S=sum(Input.Split(1:i))-(i-1);
    end
    Ynu(S+1)=Ynu(S)+(Xnu(S+1)-Xnu(S))*(Ynu(S)-Ynu(S-1))/(Xnu(S)-Xnu(S-1));
end

end
```

```
function VaneBuild(xp,yp,c,Input)
%This function takes a filename and a structure. The structure contains
%two vectors, a number of points, and a specification of the type of
%curve desired (Split or not), inlet tangent, outlet tangent, inlet and
%outlet locations. The fuction then calculates the proper curves and
%writes to a blade.xxx file with the proper structure.

%Calculate the shape
[X,Y]=Bez(xp,yp,Input.n,Input.Type,Input.Split);
%Check for self intersection
[xint,~,~]=selfintersect(X,Y);
if length(xint)>1
    %Throw error if intersecting
    error('Vane is self intersecting')
end
%Check for vane intersection by traversing backwards across first vane to
%see if upper surface intersects lower surface of next vane.
if max(fliplr(Y)>Y+c)==1
    %Throw error if solidity too low
    error('Solidity too low for vane')
end
%Open file if valid vane
F=fopen(['blade.',Input.FileName],'w');
%Print name and flow parameters
fprintf(F,[Input.FileName,'\n',num2str(Input.SINLin,'%1.4f'),' ',...
    num2str(Input.SOUTin,'%1.4f'),' ',num2str(Input.CHINL,'%1.4f'),' ',...
    num2str(Input.CHOUT,'%1.4f'),' ',num2str(c,'%1.4f'),'\n']);
%Print coordinates to file
for i=1:length(X)
    fprintf(F,[num2str(X(i),'%1.5f'),' ',num2str(Y(i),'%1.5f'),'\n']);
end

%Close file
fclose(F);

end
```

```matlab
function GridPar(Input)
%This function writes a grid parameter file for a given input. Parameters
%are currently hard coded, but may be changed with additions to the Input
%parameter and variable definitions within this program.

%Open grid parameter file
f=fopen(['gridpar.',Input.FileName],'w');

%Print parameters:

%Grid Type: first letter is inlet, second is outlet. F for low speed
%(periodic H grid), T for supersonic or possible shocks (offset I grid)
fprintf(f,'t t\n');
%Grid points: first number for inlet, second for outlet
fprintf(f,'30 30\n');
%Stream Lines:
fprintf(f,'28\n');
%X-spacing Parameter
fprintf(f,'1.5\n');
%Cell aspect ratio at stagnation point
fprintf(f,'0.8\n');
%Additional data
fprintf(f,'0.8\n');

end
```

```
% This script runs a parameter sweep of solidity, camber, and thickness
clear
close all

%Set up input parameters
Input=InStructSet('Test');
%Set initial ISES file
InputSetup(Input);

%Set original vane
xo=[0.998260468936824;0.848220983111959;0.607545266882477;...
    0.565550982506851;0.382451992576048;0.220742379895922;...
    0.124082372790477;0.0884215889499774;0.0142653913612633;...
    -0.0139077061338826;-0.000155079169895315;0.00309238216483985;...
    0.00889717165950690;0.0538700718446630;0.0840117802842175;...
    0.126921595862059;0.197413108131486;0.305726160100718;...
    0.468481478520226;0.663875348116633;0.998260468936824];
yo=[-0.0767000000000000;0.187330000000000;0.400920000000000;...
    0.393380000000000;0.467090000000000;0.318370000000000;...
    0.232050000000000;0.202670000000000;0.0531700000000000;...
    0.0331500000000000;-0.00117000000000000;-0.00351000000000000;...
    -0.00767000000000000;0.0347100000000000;0.0750100000000000;...
    0.121160000000000;0.200850000000000;0.274300000000000;...
    0.384410000000000;0.338130000000000;-0.0767000000000000];
c=0.3;
%Set vane for camber sweep
ycam=yo.*1.3; ycam(1)=yo(1); ycam(end)=yo(end);
%Set vane for thickness sweep
ythi=yo; ythi(2:10)=yo(2:10).*1.15; ythi(11:end-1)=yo(11:end-1)./1.15;
%Plot vanes
[Xor,Yor]=Bez(xo,yo,Input.n,Input.Type,Input.Split);
[Xcam,Ycam]=Bez(xo,ycam,Input.n,Input.Type,Input.Split);
[Xthi,Ythi]=Bez(xo,ythi,Input.n,Input.Type,Input.Split);
figure %vane shape comparisons
plot(Xor,Yor,'b-',Xcam,Ycam,'r--',Xthi,Ythi,'g-.','LineWidth',2.0)
title('Vane Comparisons for Parameter Sweeps')
xlabel('X/c')
ylabel('Y/c')
axis([-0.1 1 -0.1 0.7])
legend('GLMWT','Max Camber','Max Thickness')
grid on
figure %Solidity comparison
plot(Xor,Yor,'b-',Xor,Yor+1.3,'r--',Xor,Yor+0.3,'b-','LineWidth',2.0)
title('Vane Comparisons for Solidity Sweep')
xlabel('X/c')
ylabel('Y/c')
legend('Lowest \Delta Y/c','Highest \Delta Y/c')
```

```
grid on

%Sweep for camber
S=ycam-yo;
for i=0:10
    alf(i+1)=i.*0.1;
    Ynu=yo+alf(i+1).*S;
    [CPcam(i+1),Sep(i+1),Turn(i+1)]=MISES(xo,Ynu,c,Input);
end
figure
plot(alf,CPcam,'b-','LineWidth',2.0)
title('Camber sweep losses')
xlabel('\alpha')
ylabel('$\bar{\omega}$','Interpreter','LaTeX')
figure
plot(alf,Sep,'b-','LineWidth',2.0)
title('Camber sweep separation')
xlabel('\alpha')
ylabel('Mean Separation Distance (\Delta y/c)')
figure
plot(alf,Turn,'b-','LineWidth',2.0)
title('Camber sweep turning angle')
xlabel('\alpha')
ylabel('Turning angle (Degrees)')

%Sweep for thickness
S=ythi-yo;
for i=0:10
    alf(i+1)=i.*0.1;
    Ynu=yo+alf(i+1).*S;
    [CPthi(i+1),Sep(i+1),Turn(i+1)]=MISES(xo,Ynu,c,Input);
end
figure
plot(alf,CPthi,'b-','LineWidth',2.0)
title('Thickness sweep losses')
xlabel('\alpha')
ylabel('$\bar{\omega}$','Interpreter','LaTeX')
figure
plot(alf,Sep,'b-','LineWidth',2.0)
title('Thickness sweep separation')
xlabel('\alpha')
ylabel('Mean Separation Distance (\Delta y/c)')
figure
plot(alf,Turn,'b-','LineWidth',2.0)
title('Thickness sweep turning angle')
xlabel('\alpha')
ylabel('Turning angle (Degrees)')
```

```
%Sweep for solidity
c=0.3;
S=1.3-c;
for i=0:10
    C(i+1)=c+(i.*0.1)*S;
    [CPc(i+1),Sep(i+1),Turn(i+1)]=MISES(xo,yo,C(i+1),Input);
end
figure
plot(C,CPc,'b-','LineWidth',2.0)
title('Solidity sweep losses')
xlabel('\Delta Y/c')
ylabel('$\bar{\omega}$','Interpreter','LaTeX')
figure
plot(C,Sep,'b-','LineWidth',2.0)
title('Solidity sweep separation')
xlabel('\Delta Y/c')
ylabel('Mean Separation Distance (\Delta y/c)')
figure
plot(C,Turn,'b-','LineWidth',2.0)
title('Solidity sweep turning angle')
xlabel('\Delta Y/c')
ylabel('Turning angle (Degrees)')

%Sweep for solidity (Max Camber)
c=0.3;
S=1.3-c;
for i=0:10
    C(i+1)=c+(i.*0.1)*S;
    [CPc(i+1),Sep(i+1),Turn(i+1)]=MISES(xo,ycam,C(i+1),Input);
end
figure
plot(C,CPc,'b-','LineWidth',2.0)
title('Solidity sweep for max camber losses')
xlabel('\Delta Y/c')
ylabel('$\bar{\omega}$','Interpreter','LaTeX')
figure
plot(C,Sep,'b-','LineWidth',2.0)
title('Solidity sweep for max camber separation')
xlabel('\Delta Y/c')
ylabel('Mean Separation Distance (\Delta y/c)')
figure
plot(C,Turn,'b-','LineWidth',2.0)
title('Solidity sweep for max camber turning angle')
xlabel('\Delta Y/c')
ylabel('Turning angle (Degrees)')
```

```
%Sweep for solidity (Max thickness)
c=0.3;
S=1.3-c;
for i=0:10
    C(i+1)=c+(i.*0.1)*S;
    [CPc(i+1),Sep(i+1),Turn(i+1)]=MISES(xo,ythi,C(i+1),Input);
end
figure
plot(C,CPc,'b-','LineWidth',2.0)
title('Solidity sweep for max thickness losses')
xlabel('\Delta Y/c')
ylabel('$\bar{\omega}$','Interpreter','LaTeX')
figure
plot(C,Sep,'b-','LineWidth',2.0)
title('Solidity sweep for max thickness separation')
xlabel('\Delta Y/c')
ylabel('Mean Separation Distance (\Delta y/c)')
figure
plot(C,Turn,'b-','LineWidth',2.0)
title('Solidity sweep for max thickness turning angle')
xlabel('\Delta Y/c')
ylabel('Turning angle (Degrees)')

%Sweep for expansion ratio
c=0.3;
S=1.5-1.1;
for i=0:10
    er(i+1)=1.1+(i.*0.1)*S;
    Input.SINLin=er(i+1);
    Input.SOUTin=tand(atand(er(i+1))+90);
    [CPc(i+1),Sep(i+1),Turn(i+1)]=MISES(xo,yo,c,Input);
end
figure
plot(er,CPc,'b-','LineWidth',2.0)
title('Expansion ratio sweep')
xlabel('ER')
ylabel('$\bar{\omega}$','Interpreter','LaTeX')
figure
plot(er,Sep,'b-','LineWidth',2.0)
title('Expansion ratio separation')
xlabel('ER')
ylabel('Mean Separation Distance (\Delta y/c)')
figure
plot(er,Turn,'b-','LineWidth',2.0)
title('Expansion ratio turning angle')
xlabel('ER')
ylabel('Turning angle (Degrees)')
```

Appendix B: Sample Files

B.1 Sample MatLAB Commands

```
In=InStructSet('BFGnu3');
[loss,sep,turn]=MISES(xp,yp,0.325,In)
```

Alternately:

```
In=InStructSet('BFGnu3');
VaneBuild(xp,yp,0.325,In);
InputSetup(In);
MISESEval(In);
[loss,sep,turn]=DataRead(In.FileName)
```

B.2 Input Files

B.2.1 blade.BFGnu3 (Truncated)

```
BFGnu3

1.5000 -0.6667 1.0000 1.0000 0.3260

0.99826 -0.07670

0.98737 -0.05795

0.97615 -0.03943

0.96466 -0.02118

0.95294 -0.00320

...

0.86453 0.07534

0.88556 0.05342

0.90707 0.03018

0.92908 0.00559

0.95160 -0.02040

0.97466 -0.04782

0.99826 -0.07670
```

B.2.2 ises.BFGnu3

```
1 2 15 6 5

1 4 15 6 3

0.1473 0.9850 1.5000 -0.5000

0.0000 0.0 -0.6667 1.5000

0.0 1.0

0.2214E06 4.0000

1.0 1.0

3 0.970 1.00
```

B.2.3 spec.BFGnu3

```
1

1.2
```

B.3 Output

B.3.1 MatLAB command line output

```
loss =

   0.1163

sep =

   0.0097

turn =

   73.1048
```

B.3.2 polar.BFGnu3

MISES polar driver Version 2.69

Calculated polar for: BFGnu3 1 elements

Sinl Sout Minl Mout Pinl/Po1 Pout/Po1 Re/1e6 Tu % omega omega_V Xtr_top Xtr_bot rVt1 d(rVt) DF cl Phi Psi
—— —— —— —— —— —— —— —— —— —— —— —— —— ——
—— —— —— ——

1.50000 -1.39777 0.14730 0.14044 0.98496 0.98506 0.221 0.57166 0.08446 0.08769 0.4206 0.1152 0.8321 1.6076 0.3086 1.0738 0.6667 107.4875

B.3.3 field.BFGnu3 (Truncated)

```
# x y rho/rho0 p/p0 u/a0 v/a0 q/a0 M

-1.0730 -1.9089 0.98923 0.98496 0.81531E-01 0.12230 0.14698 0.14730
-1.0699 -1.9042 0.98923 0.98496 0.81531E-01 0.12230 0.14698 0.14730
-1.0674 -1.9005 0.98923 0.98496 0.81531E-01 0.12230 0.14698 0.14730
...
2.0850 -1.8976 0.98931 0.98506 0.85263E-01 -0.11914 0.14651 0.14682
2.0935 -1.9095 0.98931 0.98506 0.85277E-01 -0.11915 0.14652 0.14683

-1.0780 -1.9056 0.98923 0.98496 0.81531E-01 0.12230 0.14698 0.14730
-1.0750 -1.9011 0.98923 0.98496 0.81531E-01 0.12230 0.14698 0.14730
-1.0724 -1.8972 0.98923 0.98496 0.81531E-01 0.12230 0.14698 0.14730
...
2.2305 -1.7952 0.98931 0.98507 0.85193E-01 -0.11905 0.14639 0.14671
2.2386 -1.8065 0.98931 0.98507 0.85165E-01 -0.11907 0.14639 0.14671
```

Appendix C: Installation of Necessary Software

1. Install desired distribution of Linux. Ubuntu was used for this thesis.
2. Install MATLAB from MathWorks Site.
3. Use *apt−get* command to install TCL and Expect or download from respective websites.
4. Ensure that a FORTRAN compiler and *make* command are installed on system.
5. Obtain MISES from MIT Technology Licensing Office or Dr. Mark Drela.
6. Create new folder and extract MISES to it.
7. Alter plotlib.make in the plotlib folder and Makefile in the bin folder to reflect installed FORTRAN compiler.
8. Use *make all* command in plotlib folder.
9. Make new folder within the bin folder of MISES directory and alter Makefile to output to this folder.
10. Use *make all* command in bin folder.
11. Copy and save all MATLAB functions to folder created in step 9
12. Test by generating a vane and using command *./iset vane.xxx* from the new folder terminal with xxx replaced by your vane's name.

Bibliography

[1] “Glenn L. Martin Wind Tunnel” Available: https://windtunnel.umd.edu [retrieved 22 Feb. 2019].

[2] Barlow, J. B., Rae, W. H., Jr., and Pope, A., “Wind Tunnel Design,” Low Speed Wind Tunnel Testing, 3rd ed., Wiley, New York, 1999, pp. 61-135.

[3] Garrick, I. E., “On the Plane Potential Flow Past a Lattice of Arbitrary Airfoils,” NACA TR-788, January 1944.

[4] Ives, D. C., and Liutermoza, J. F., “Analysis of Transonic Cascade Flow Using Conformal Mapping and Relaxation Techniques,” AIAA Journal, Vol. 15, No. 5, 1977, pp. 647-652.

[5] Spurr, R.A., and Allen, H. J., “A Theory of Unstaggered Airfoil Cascades in Compressible Flow,” NACA RM-A7E29, January 1947.

[6] Baskharone, E., and Hamed, A., “A New Approach in Cascade Flow Analysis Using the Finite Element Method,” AIAA Journal, Vol.19, No. 1, 1981, pp. 65-71.

[7] Drela, M., and Youngren, H., “A User’s Guide to MISES 2.63,” MIT Aerospace Computational Design Laboratory, Cambridge, MA, February 2008.

[8] Gelder, T. F., Moore, R. D., Sanz, J. M., and Mcfarland, E. R., “Wind Tunnel Turning Vanes of Modern Design,” NASA TM-87146, January 1985.

[9] López de Vega, L., Maldonado Fernández, F. J., and Muñoz Botas, P., “Optimisation of a Low-Speed Wind Tunnel. Analysis and Redesign of Corner Vanes.” Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio, Universidad Politécnica de Madrid, Madrid, Spain, May 2014.

[10] Lindgren, B., Österlund, J., and Johansson, A.V., "Measurement and Calculation of Guide Vane Performance in Expanding Bends for Wind-Tunnels," Springer-Verlag, Vol. 24, Issue 3, 1998, pp. 265-272.

[11] Samareh, J. A., "A Survey of Shape Parameterization Techniques," NASA CP-1999-209136/PT1, June 1999.

[12] Derksen, R. W., Rogalsky, T., "Optimum Aerofoil Parameterization for Aerodynamic Design," Wessex Institute of Technology Transactions on The Built Environment, Vol. 106, 2009, pp. 197-206.

[13] Jaiswal, A. S., "Shape Parameterization of Airfoil Shapes Using Bezier Curves," Innovative Design and Development Practices in Aerospace and Automotive Engineering, Springer Science, Singapore, 2016, pp. 79-85.

[14] Kamermans, M. "A Primer on Bezier Curves," [online], [2018]. Available: `https://pomax.github.io/bezierinfo/`

[15] Canos, A. "Fast and Robust Self-Intersections," [online], [13 December 2006]. Available: `https://www.mathworks.com/matlabcentral/fileexchange/13351-fast-and-robust-self-intersections`

[16] Engineering ToolBox, "Air - Density, Specific Weight and Thermal Expansion Coefficient at Varying Temperature and Constant Pressures," [online], [2003]. Available: `https://www.engineeringtoolbox.com/air-density-specific-weight-d_600.html?vA=293&units=K#`