Document of Defense

Date: 11/1/2023

I, Bharadwaj Dogga, hereby submit this original work as a part of the requirements for the degree of Master of Science in Aerospace Engineering.

It is entitled:

Airfoil parameters reverse engineering framework for plot digitized blade sections

Student Signature: Bharadwaj Dogga

This work and its defense approved by:

Chair : Mark Turner, Sc.D.

Member : Daniel Cuppoletti, Ph.D.

Member : Prashant Khare, Ph.D.

Member : Paul Orkwis, Ph.D.

Member : Benjamin Vaughan, Ph.D.

Airfoil Parameters Reverse Engineering Framework for Plot Digitized Blade Sections

A thesis submitted to the

Graduate College

of the University of Cincinnati

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Department of Aerospace Engineering and Engineering Mechanics

of the College of Engineering and Applied Science

by

Bharadwaj "Ben" Dogga

B.Engg, University of Mumbai, India, 2017

Committee Chair: Dr. Mark G. Turner

Committee Members: Dr. Paul Orkwis,

Dr. Prashant Khare,

Dr. Daniel Cuppoletti,

Dr. Benjamin Vaughan

Abstract

A framework to reverse engineer airfoil section parameters using a Turbomachinery Blade Geometry code has been developed and presented. A multivariable single objective optimization is used to reduce the sum of the square difference between the parametric blade shape and the target airfoil blade section to obtain those parameters. The method divides input airfoil into six parts to simplify blade difference calculation. A turbomachine blade section is obtained using the new input files with airfoil parameters: inlet and outlet metal angles, six curvature control points, Leading edge radius, location of maximum thickness, value of maximum thickness, and trailing edge thickness. Key issues of the process are discussed.

A demonstration of the developed method was carried out by first reverse engineering three different airfoils and then reverse engineering E3 transonic compressor blade from its sections. This blade was chosen due to its uniqueness of having a sloped hub. The Airfoil sections were plot digitized from the E3 report which were then run through the method to get Tblade3 parameters. A subsequent 3D simulation of the blade has been carried out to compare the performance of the reverse engineered blade with it's the experimental results of the actual design.

Furthermore, a grid dependence and off design study (full speedline) has been carried out to determine the most appropriate running condition for the comparison. Insights on further directions are suggested that will improve the comparison.

Acknowledgments

I would like to express my gratitude to Dr. Turner, for his support and guidance throughout the duration of this thesis. He has been instrumental in helping my transition from Mechanical to Aerospace Engineering concepts since beginning of this course. His approach of asking questions in the right direction without handing over a straight solution has been instrumental in developing my research skills. His extreme support throughout missed deadlines and the Covid-19 pandemic has ensured this work is of the highest quality as possible.

My sincere gratitude to the committee members - Dr. Orkwis, Dr. Khare, Dr. Cuppoletti, and Dr. Vaughan - who have been patient with my submission and have painstakingly offered their valuable review of this work at such short notice.

I would like to thank folks from the UC Simulation Center - Melissa Ryan, Fred Murrell, and Ken Comer for helping fund two years of my master's degree. The soft skills and CFD etiquette that I learned from their Graduate Assistantship under a Proctor and Gamble team with Dr. Hu Hua helped me develop a disciplined approach to problem-solving. Sincere thanks to the Office of Innovation - Kaethe Beck, Geoffrey Pinsky, and James Currie for offering insights into how relevant innovation is to the workspace along with mentoring me on business operations and commercialization skills.

I would also like to thank Julie Muenchen, Amanda McLaughlin, Dilip Kalagotla, Kranthi Yellugari, Gibin Raju, Saugat Ghimire, and many other fellow graduate students and staff within CEAS and UC that helped me navigate the academic environment. Their advice and mentorship has resulted in an extremely fruitful Master's education experience and you have my extreme gratitude.

Lastly, I would like to thank my parents - Karunakumari and Satyanarayana Dogga, and the extended family and friends back home for their patience in understanding my time away from them over the past few years and through a global pandemic.

Contents

List of Figures

List of Tables

Nomenclature

E^3 = Energy Efficient Engine

K = Temperature (Kelvin)

Pa = Pascals

u = X component of the normalized airfoil plot with zero stagger

v = Y component of the normalized airfoil plot with zero stagger

TE = Trailing Edge

LE = Leading Edge

tm/c = Maximum Thickness to chord ratio

pss = Part span shroud

Subscripts

r = Radial direction

θ = Tangential direction

z = Axial direction

Chapter 1

Introduction

1.1 Motivation

With the advent of modern jet engines and electric propulsion systems, reverse engineering a turbomachinery blade - in both land and flying gas turbines has gained a renewed interest. While there are many methods in place, throughout the design process a need was recognized for a design framework that could help digitize blades from plot digitized airfoil sections. The idea is to make use of Tblade3 CAD which had the capability to parametrically design airfoil sections and bring in a gradient-based optimizer to minimize the least squared difference between a target and initial airfoil until they are satisfactorily digitized. This framework would also have the advantage of dimensionalising the target airfoil in a given set of parameters instead of the input two-coordinate systems. This is essential because these parameters bring in the benefit of design optimization if the airfoil/blade is to be adopted from one use case to the other. The objective function of the framework has to be twice differentiable easing the calculation of gradient and hessian matrix and should make use of squared distance instead of linear distance for quicker convergence.

1.2 Literature review and previous work

1.2.1 Blade reverse engineering and Tblade3

Though relatively rare, blade reverse engineering methods have been previously explored for easy storage and optimization of existing designs. Chen at al [1] described a conventional approach of 3D scanning a given turbine blade and then builds on top of it using Modified Adaptive Model-based Digitizing Process (MAMDP) to refine the coordinates obtained into surfaces. Another method described by Mohaghegh et al. [2] is to use reverse engineering to counteract the irregularities that scanner errors produce in extremely delicate designs like those of turbine blades. They suggested implementing design key-points during reverse engineering and demonstrated it on a gas turbine blade.

For our work, we demonstrated the framework using blade design capabilities of an in-house code Tblade3 [3] which is a design tool for 3D blades using a parametric approach. Siddappaji [4] demonstrated its general capabilities to design three use cases - a GE 1.5 MW wind turbine fan design along with carrying out CFD and FEA analysis, a 10-stage high pressure compressor full definition design, and a low-speed centrifugal compressor design. In this thesis, a detailed optimization of Tblade3 coupled with an FEA and a CFD code using a three stage booster's blisk blades is carried out in Siddappaji et al. [5]. AF Nemnem et al.[6] expanded its capabilities to control the meanline curvature with a b-spline to create smooth airfoil surfaces. This resulted in reduced spikes in Mach numbers and pressure distributions. They demonstrated these capabilities over a range of airfoils in various aerodynamic regimes. Balasubramanian et al. [7] added in a novel curvature-based airfoil parameterization feature to tap into the direct proportionality of streamline curvature to pressure gradient using control points for meanline second derivative and a thickness distribution. Sharma et al. [8] integrated Tblade3 into Engineering Sketch Pad (ESP) [9] which is an open-source web-enabled solid modelling system. This integration also enables flends to Tbalde3's solid blade models using ESP. The resulting geometry was simulated on a FEM based centrifugal and pressure loads. Furthermore, Sharma et al. [10] added in continuous parameterization instead of discrete for better optimization control.

1.2.2 Energy Efficient Engine Fan and CFD Simulations

To demonstrate the capabilities of the framework, along with standard 2D airfoils from airfoiltools.com, a jet engine fan from the 1970s was chosen. This fan was part of the Energy Efficient Engine(E3) project from NASA and was selected because of the ease of availability of its three airfoil sections in the form of an E3 report. Initial insights of the geometry were obtained from RW Claus et al. [11]. Their paper helped get a primary understanding of how the fan geometry should look like. An AIAA meeting paper by Turner M [12] helped underscore the implications of this work in full engine simulations. The design parameters for E3 fan are obtained from NASA-GE STI design report [13], henceforth referred to as the E^3 report. Though this report has the E3 fan with a part span shroud, we will be designing it without one to focus solely on the reverse engineering capabilities of the code. Further insights into GE fan design were obtained from two papers by L.H. Smith. First discusses the evolution of compressor design [14] and second discusses how design methods have evolved to support CFD methods helping make a choice about leading and trailing edge radius values[15]. Turner et al. [16] explored the turbulence modeling of transonic fans while developing an explicit Navier-stokes solver and recommended $k - \epsilon$ solver as a go to solver. Jennions et al. [17] further demonstrated the abilities of a 3-dimensional $k - \epsilon$ solver on three major industrial configurations – NASA Rotor 67, GE/Wennerstrom Rotor 4, and the GE/NASA E^3 fan. Various parameters were considered like tip leakage flow, shock boundary layer interaction, boundary layer growth, and account of internal solid bodies such as part-span shrouds and engine splitters. Barring the E3 fan, there has been many design optimizations carried out from the reports starting with 3D all engine component simulations [18], High Pressure Turbine simulation [19] Low Pressure Turbine simulation [20] a validation study of the Low Pressure Turbine [21], Combustor design [22], Combustor to High Pressure Turbine simulation [23] and High pressure Compressor cavity simulations [24]. Furthermore, losses in the 3D simulations were analyzed using concepts from Denton et al [25].

1.2.3 Multidisciplinary Design Analysis and Optimization

While some of the above optimizations and simulations used other CAD packages to design the turbomachine blades, a major chunk of them were designed using Tblade3. This is attributed to flexibility of Tblade3 and availability of literature in various test cases. Mahmood et al. [26] presented a design process with Tblade3 as the design CAD and Dakota [27] as the optimization driver using Genetic Algorithm to explore the design space. This technique was demonstrated using rotor 6 of the E3 High pressure compressor as a testcase for a single objective optimization to optimize isentropic efficiency.

Mandal et al. [28] developed an optimization process with Dakota for a propulsor design while demonstrating a three blade-row design for NASA BLI fan. The experimental and CFD results from such a process were contrasted by Sieradzki et al. [29] who demonstrated numerical and design challenges for simulation of BLI fans while using FINE/Turbo. Chen et al. [30] studied flow diagnostics and optimization of transonic compressor/Fan rotor with a single objective function genetic algorithm using Dakota.

While Dakota was initially considered as an optimizer choice given its proven integration with Tblade3, the switch from Dakota was to OpenMDAO [31] as the optimizer was justified with the availability of python scipy's SLSQP [32] module which made both equality and inequality constraints. This was demonstrated in Tom Viars' mini thesis [33] which highlights the use of OpenMDAO to design an optimization using Tblade3. A similar approach was adopted throughout this thesis.

Chapter 2

Methodology

The framework is developed using a combination of python scripts wrapped over Tblade3 and OpenMDAO as shown in Figure 2.3. It allows the input of target airfoil coordinates and initial input files for Tblade3 which are then used to generate reverse-engineered airfoil parameters for the target airfoil. The core of the framework is the development of an airfoil difference calculator which is discussed in the section below, the entire optimization framework is discussed next, followed by an evaluation of the optimum step size for each variable.

2.1 Computing least squared difference

For nomenclature reasons - initial airfoil is generated by Tblade3 during first run before the framework is engaged. Generated airfoil is any airfoil generated by Tblade3 when the framework is being run. Target airfoil is the plot digitized airfoil from 2D sections in reports or from u, v coordinates of standard airfoils.

The least square difference calculator assigns a numerical value to the difference in two airfoil sections - a plot digitized airfoil (target) and the reverse engineered airfoil (initial) section by following these steps:

1. Import target and initial airfoil coordinates
2. Break each airfoil into six parts as shown in Figure 2.1
3. For each part, ensure that initial blade coordinates are a subset of target blade coordinates by

dropping initial blade coordinates whose abscissa is out of the target blade abscissa's range. (Section 2.1.1)

4. Next, abscissa of target blade is interpolated on the subset of initial blade to make sure that both initial and target blade parts have the same abscissa. (Section 2.1.2)

5. The new values of abscissa are used to calculate new u coordinates of each target section. This makes the calculation of least squared differences a one-dimensional calculation along v. (Section 2.1.3)

6. Finally, the difference between each airfoil parts is calculated , the difference is squared and then summed to obtain the total least squared difference value. (Section 2.1.4) The below subsections look at the mathematical equations of the above procedures.

2.1.1 Truncation of target airfoil coordinates

To ensure that the $u_{target\_airfoil}$ coordinates are a subset of generated airfoil u coordinates, all the coordinates set values with u values greater than the first coordinate in the generated airfoil are dropped for airfoil sections on the suction side. This is also repeated on the other end of the airfoil part where the cutoff criterion is:

$$\begin{aligned} u_0^{target} &\leq u_0^{generated} \\ u_n^{target} &\geq u_n^{generated} \end{aligned} \tag{2.1}$$

Conversely, for the pressure side, the cut-off criterion is flipped:

$$\begin{aligned} u_0^{target} &\geq u_0^{generated} \\ u_n^{target} &\leq u_n^{generated} \end{aligned} \tag{2.2}$$

2.1.2 Piecewise linear interpolation

u coordinate values of both airfoils need to be equated so that a squared difference can be obtained between each of the v values in a single dimension. To do this, a linear interpolation polynomial function is fit on the target airfoil and is used to obtain values of v from the target airfoil for the corresponding u value from the generated airfoil. This also has the added advantage of easy visual comparison as the number of coordinates is the same in both airfoils. Mathematically, we fit a piecewise linear interpolation function "f" on initial airfoil coordinates and then use u values from the target airfoil as input to "f" to get their corresponding v values. This will create a new set of coordinates for the generated airfoil whose u values will be equal to the target airfoil.

A piecewise linear interpolant that interpolates using data points $(u_i, v_i)_{i=0}^{n}$, where $x_0 < x_1 < \ldots < x_n$ is given by eq 2.3

$$f_{1,n}(u) = \begin{cases} v_0 \frac{u-u_1}{u_0-u_1} + v_1 \frac{u-u_0}{u_1-u_0}, & u \in [u_0, u_1] \\ v_1 \frac{u-u_2}{u_1-u_2} + v_2 \frac{u-u_1}{u_2-u_1}, & u \in [u_1, u_2] \\ \vdots \\ v_{n-1} \frac{u-u_n}{u_{n-1}-u_n} + v_n \frac{u-u_{n-1}}{u_n-u_{n-1}}, & u \in [u_{n-1}, u_n] \end{cases} \tag{2.3}$$

The i^{th} piece in each subinterval $[u_{i-1}, u_i]$ is given by Lagrange's interpolation formula for first order polynomial (eq. 2.4). These equations are represented here from Xu S[34] and were solved using Numpy interpolate's *interp1d* class.

$$v_{i-1} \frac{u - u_i}{u_{i-1} - u_i} + v_i \frac{u - u_{i-1}}{u_i - u_{i-1}}, u \in [u_{i-1}, u_i] \tag{2.4}$$

2.1.3 Quadratic least squares approximation along v

On the new interpolation generated airfoil and the target airfoil, an order two polynomial is fit on the u, v coordinates to create a pseudo camber line to separate suction and pressure side calculations. Quadratic least-squares approximation was used and the equations from Zubairi et al. [35] are reproduced here. It uses a second-degree polynomial to approximate a given set of data $(u_1, v_1), (u_2, v_2), ..., (u_n, v_n)$, where $n \geq 3$. We minimize least squares error (Π) in eq 2.5, where a, b, and c are the unknown coefficients for all u_i, v_i data points.

$$\Pi = \sum_{i=1}^{n} (v_i - (a + bu_i + cu_i^2))^2 \tag{2.5}$$

Partially differentiating Π with respect to a, b, and c and rearranging, we get the linear equations 2.6, 2.7, 2.8. Which are then solved to obtain the value of unknown coefficients.

$$\sum_{i=1}^{N} v_i = a\sum_{i=1}^{N} 1 + b\sum_{i=1}^{N} u_i + c\sum_{i=1}^{N} u_i^2 \tag{2.6}$$

$$\sum_{i=1}^{N} u_i v_i = a\sum_{i=1}^{N} u_i + b\sum_{i=1}^{N} u_i^2 + c\sum_{i=1}^{N} u_i^3 \tag{2.7}$$

$$\sum_{i=1}^{N} u_i^2 v_i = a\sum_{i=1}^{N} u_i^2 + b\sum_{i=1}^{N} u_i^3 + c\sum_{i=1}^{N} u_i^4 \tag{2.8}$$

Python Numpy's *polyfit* class solves for these coefficients. They can be called using *poly1d* class within the same module that stores the fit function (eq. 2.9) which is the mean camber line function for a given airfoil data points set.

$$v_i = \Gamma_{mcl\_curve}(u_i) \tag{2.9}$$

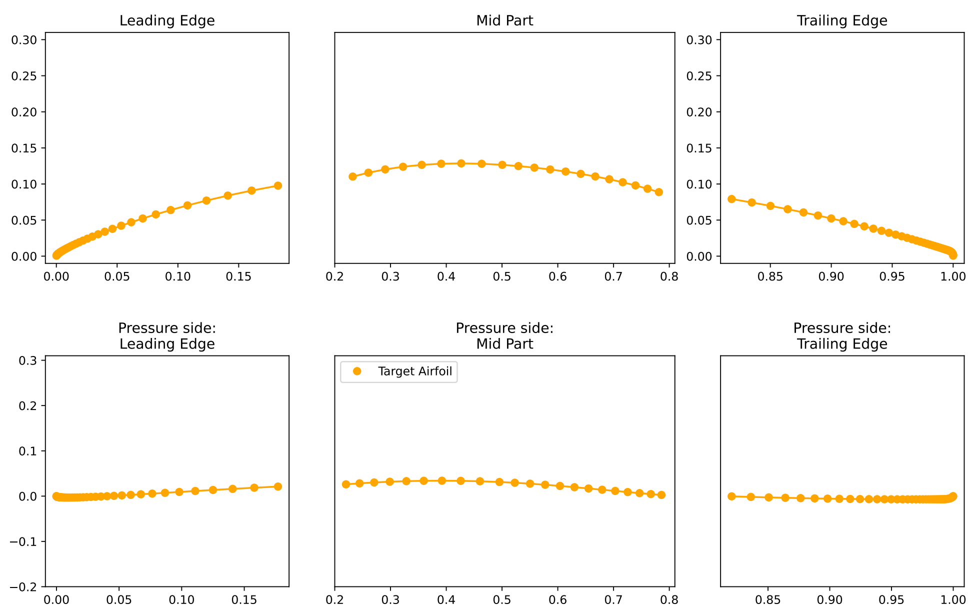


Figure 2.1: Target blade divided into six parts

2.1.4 Squared difference

Differences between each ordinate within an individual part are calculated as shown in eq. 2.10 to 2.15. Once calculated, these differences are then summed up to obtain a single value of squared difference (δ_{sd}) for the entire airfoil - eq 2.16.

$$\delta_{suc\_LE} = \sum_{j=1}^{m_{suc\_LE}} (v_j^{rev\_engg} - v_j^{target})^2 \tag{2.10}$$

$$\delta_{suc\_mid} = \sum_{j=1}^{m_{suc\_mid}} (v_j^{rev\_engg} - v_j^{target})^2 \tag{2.11}$$

$$\delta_{suc\_TE} = \sum_{j=1}^{m_{suc\_TE}} (v_j^{rev\_engg} - v_j^{target})^2 \tag{2.12}$$

$$\delta_{pre\_LE} = \sum_{j=1}^{m_{pre\_LE}} (v_j^{rev\_engg} - v_j^{target})^2 \tag{2.13}$$

$$\delta_{pre\_mid} = \sum_{j=1}^{m_{pre\_mid}} (v_j^{rev\_engg} - v_j^{target})^2 \tag{2.14}$$

$$\delta_{pre\_TE} = \sum_{j=1}^{m_{pre\_TE}} (v_j^{rev\_engg} - v_j^{target})^2 \tag{2.15}$$

$$\begin{aligned} \delta_{sd} = \delta_{suc\_LE} + \delta_{suc\_mid} + \delta_{suc\_TE} \\ + \delta_{pre\_LE} + \delta_{pre\_mid} + \delta_{pre\_TE} \end{aligned} \tag{2.16}$$

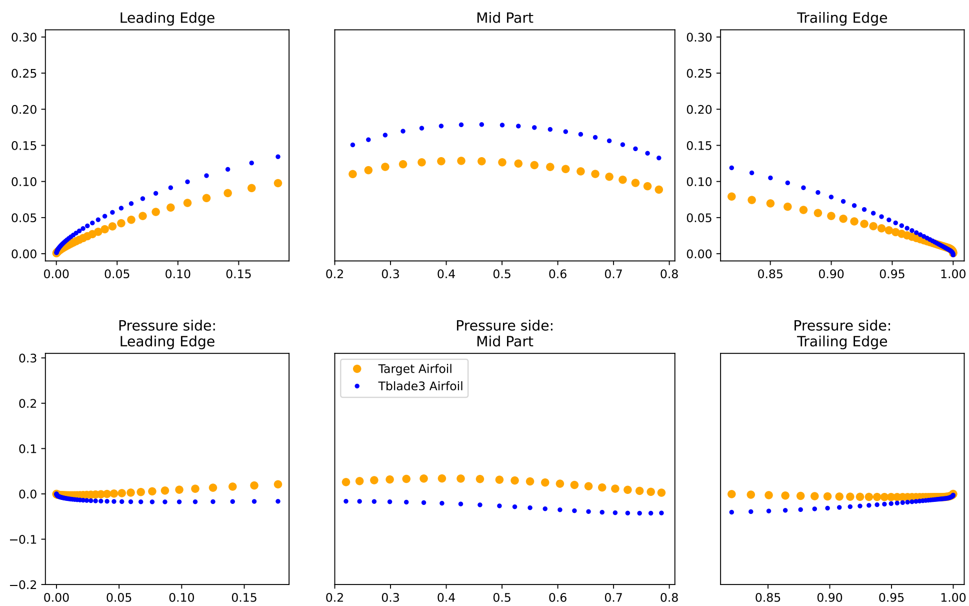


Figure 2.2: Tblade3 airfoil parts after interpolation to calculate least square difference

2.2 Optimization framework

Shown in Figure 2.3 is the overview of airfoil reverse engineering framework. The process begins with preparing the plot-digitized airfoil coordinates and sorting them into Selig airfoil format. This gives us the target coordinates. We prepare TBlade3 input files 3dbgbinput.1.dat and spancontrolinputs.1.dat and specify them with manual best guess input and output metal angles, curvature control points, LE radius, thickness of max camber, location of max. camber thickness, and TE thickness. Once we have these values, we do a run of Tblade3 to obtain the initial airfoil coordinates of our reverse engineering blade. With the target and initial blade coordinates in a standard format, the OpenMDAO script file is run. The script file then runs Tblade3 on the initial Tblade3 input files to produce the u-v coordinates of the airfoil. Once we have this airfoil, it runs an external python script to calculate squared difference between target and generated airfoil coordinates, then saves the values into a .dat file. The framework also creates a .pdf report of the airfoil sections and writes an image file showing a visual comparison of a generated airfoil with the target airfoil along with the squared difference annotated on the top right corner. Once the script is done, the OpenMDAO script takes over and prints the values of input parameters and squared difference to the screen, it also writes the parameters into the .base, .log, and .dat template files and saves them. It then checks the least squared difference value with the given tolerance, if the tolerance is lower than the specified value or if the value of the least squared difference is stagnating, then the solver exits. Else, it goes back to the beginning and alters values using a minimization algorithm - SLSQP by default, while using the generated blades as the new initial blade. This way it approaches the target blade by minimizing the least squared distance value.

Once the optimization reaches an exit value, we then run the optimization post-processor script, which takes in the log file from the final optimization run and prepares a .pdf report with iteration vs value plots of squared difference and other input variables. It also prints out a table comparing the initial vs target values of the variables.

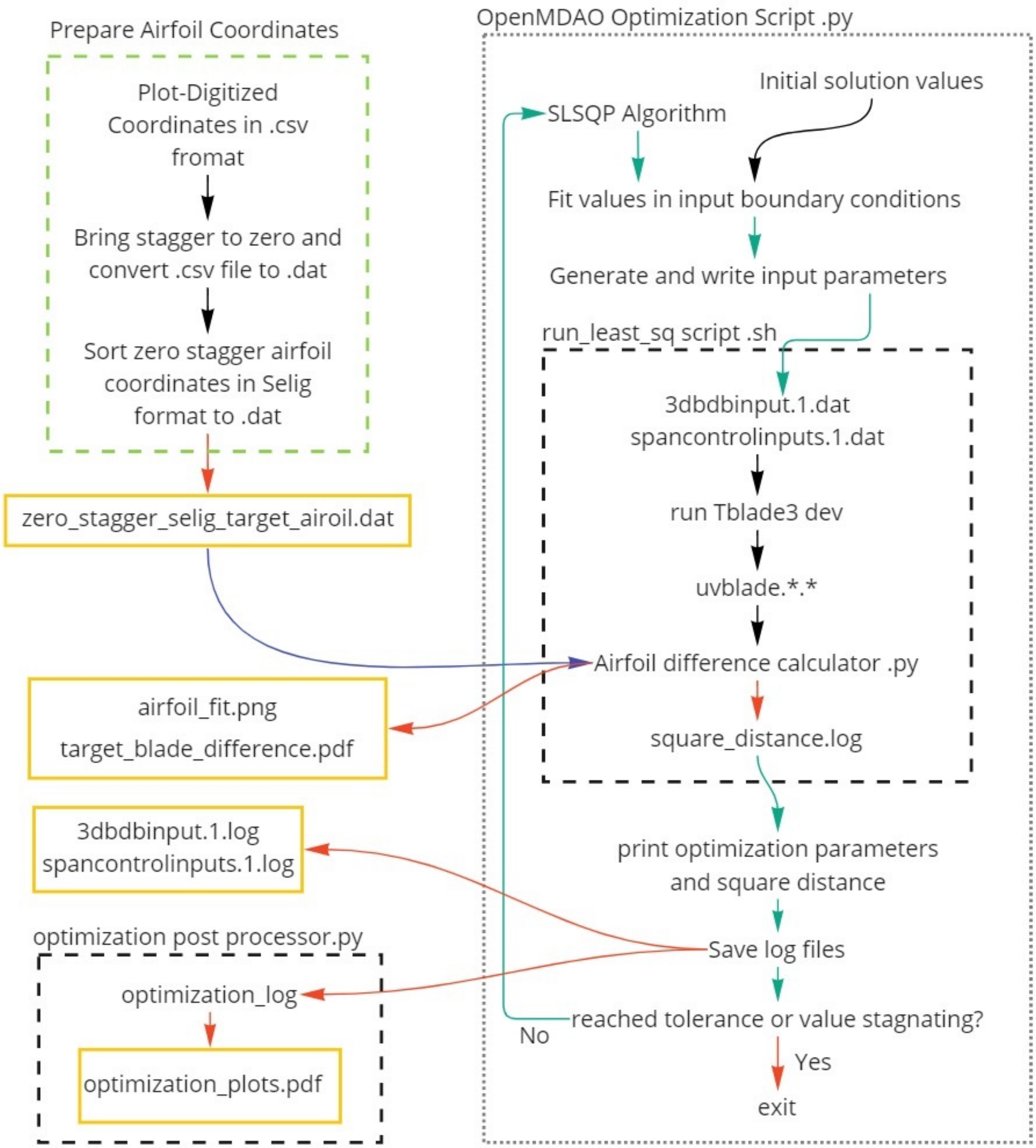


Figure 2.3: Overview of Airfoil parameters reverse engineering framework

2.3 Evaluation of optimum step size for each variable

The structure of optimization variables is described by N2 Model (Figure 2.4) which is visualized below with our squared difference objective function using twelve variables and a constant. To accurately calculate the sensitivities for the objective function using finite difference methods, a step-size study is needed. This is to obtain an efficient calculation of the gradients and hessian matrices while minimizing total error in finite difference methods, which is a combination of truncation and round-off errors.

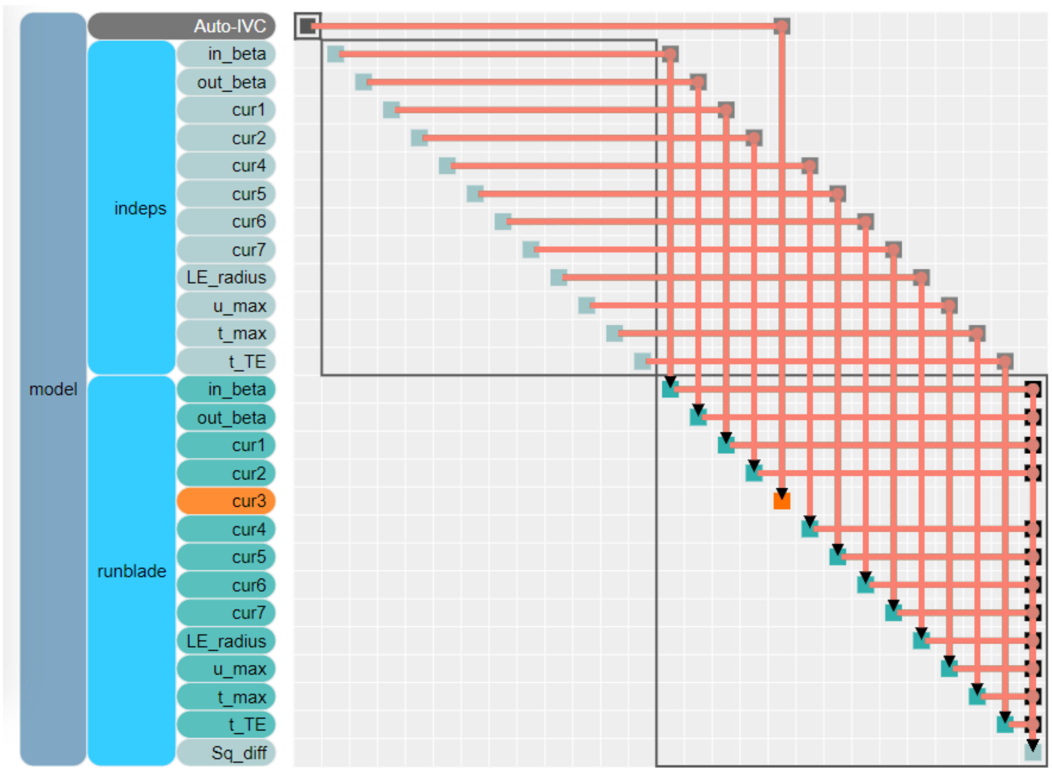
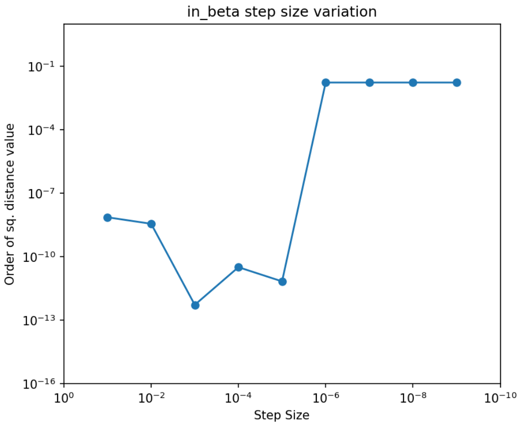
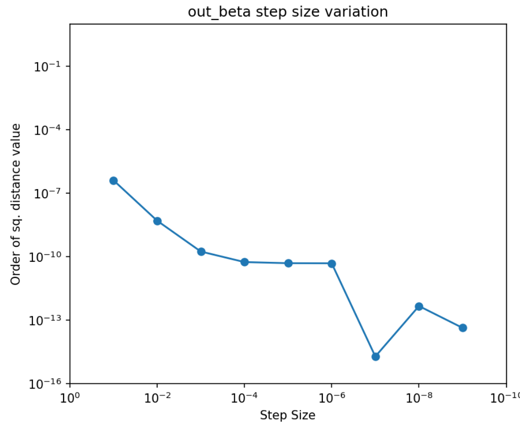


Figure 2.4: N2 model of the optimization framework

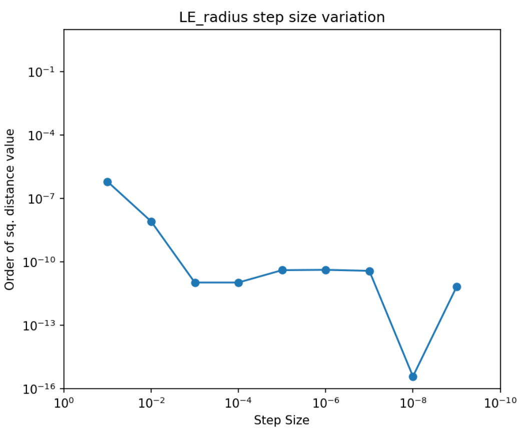
In order to perform the step-size study, a template airfoil with known parameters - in this case 0012, is taken and a single variable optimization was run with each initial parameter value increased by 10%. The error tolerance for this optimization was set to 1e-10 and multiple runs for the same variables were carried out while varying step sizes from 1 to 1e-11. Once we had the results of all the 11 step sizes for a single variable, a log-log plot of squared distance to step size was plotted for each variable as shown in Figure 2.5. By looking at these plots, we can note that the optimum step sizes for most variables is 1e-3, except for t_max and u_max for which it is 1e-4, and t_TE for which it is 1e-5. This helps us define our optimization statement in the next section.



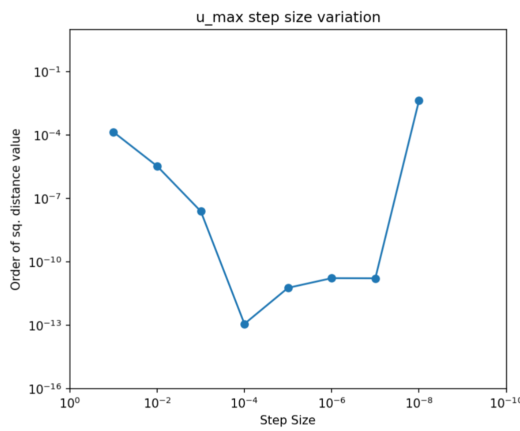
(a) *in_beta*



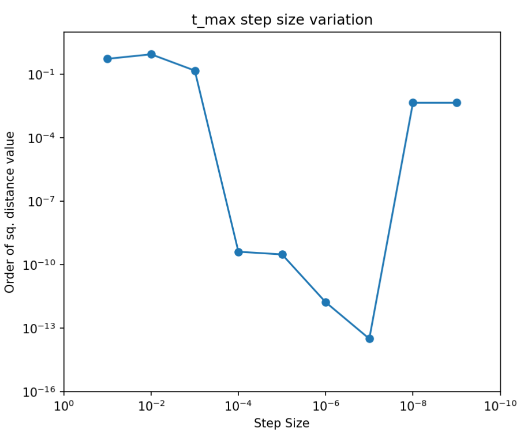
(b) *out_beta*



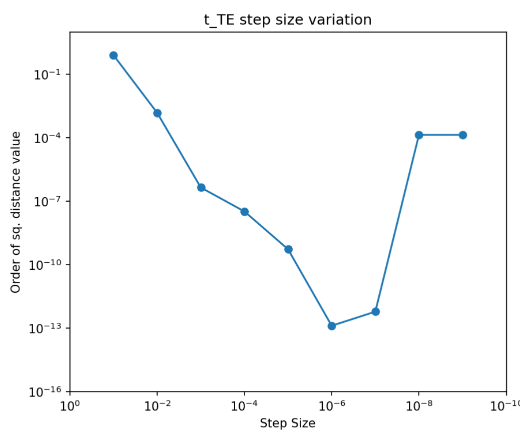
(c) *LERadius*



(d) *u_max*

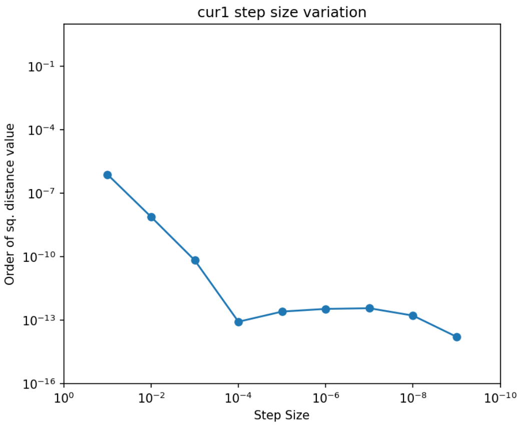


(e) *t_max*

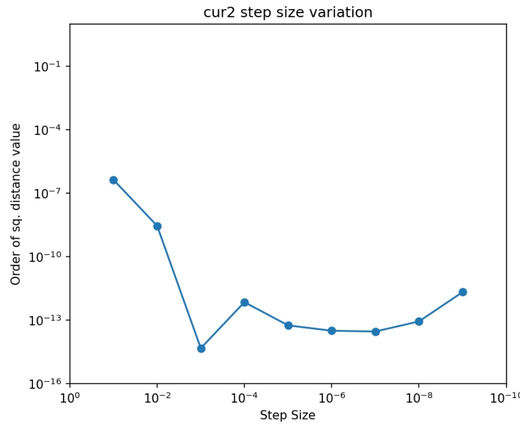


(f) *t_TE*

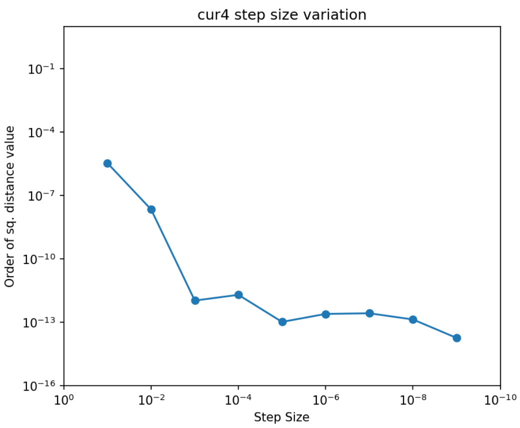
Figure 2.5: sq. distance v/s step size



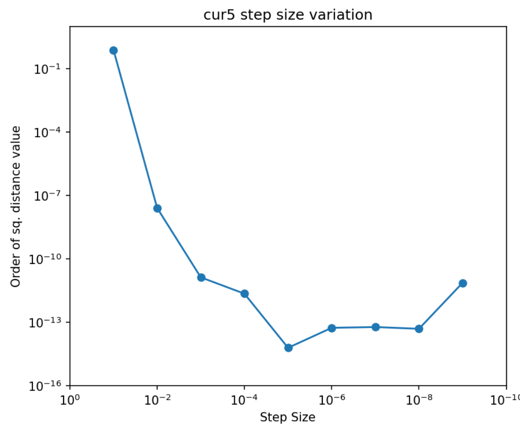
(g) cur1



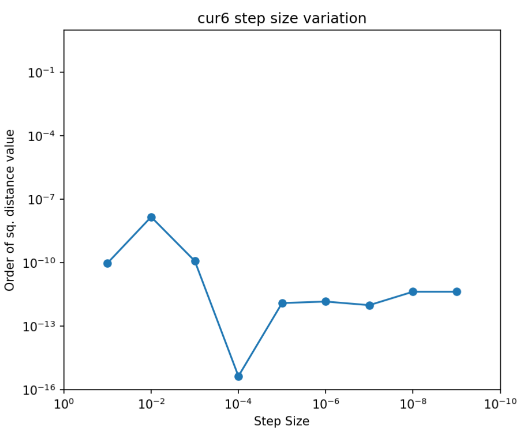
(h) cur2



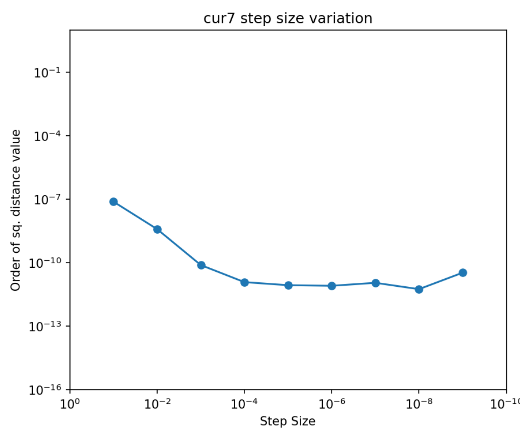
(i) cur4



(j) cur5



(k) cur6



(l) cur7

Figure 2.5: sq. distance v/s step size (contd.)

2.4 Optimization Statement

2.4.1 Minimize

Squared distance between points of a plot digitized airfoil and Tblade3 generated airfoil.

2.4.2 Design Variables

A total of 12 variables - two variables from 3dbgbinput file: *in_beta*, *out_beta*, and ten variables from spancontrolinput file: *cur*1 to 7 except *cur*3, *LE_radius*, *u_max*, *t_max*, and *t_TE* were considered.

2.4.3 SLSQP Algorithm Parameters

A tolerance of 1e-3 was set for Squared difference and maximum number of gradient evaluations was set to 400. Using inferences from step size study, the optimum step sizes were taken to be 1e-3 for all the variables except *t_max* and *u_max* for which it is 1e-4 and *t_TE* for which it is 1e-5.

2.4.4 Range of design variables

- *in_beta* ranges between *-35.10* and *55.50*
- *out_beta* ranges between *-30.00* and *20.45*
- cur points 1 to 7 values range between *-10.5* to *10.5* and are located at *0.00, 0.15, 0.25, 0.50, 0.75, 0.95,* and *1.00* span respectively.
- *LE_radius* ranges between *2.0* and *5.5*
- *u_max* ranges between *0.05* and *0.8*
- *t_max* ranges between *0.01* and *0.4*
- *t_TE* ranges between *1e-6* and *0.1*

2.4.5 Constants

Even though it can range between *-10.5* to *10.5*, to obtain a unique solution case, *cur*3 is kept constant at 2.31.

The optimization statement forms the basis for all the reverse engineering iterations, and we can next move on the demonstration of standard 2D airfoils in the next section.

Chapter 3

Airfoil demonstration cases

For 2D demonstration cases, we look at three types of airfoils: one with no camber, second with a camber, and lastly a wind turbine airfoil with a near-zero tip radius. The plot digitized target airfoil coordinates were all obtained from airfoiltools website and the stopping criterion for all these airfoil runs were set to $1e^{-03}$ with an initial condition of a generic zero camber airfoil.

3.1 NACA 0012

The NACA 0012 airfoil is a zero camber airfoil with a max thickness of 12% at 30% chord. There were 63 unit chord coordinates obtained from airfoiltools website and it took around 300 gradient evaluations of the objective function to reach a least squared difference value of $3.2659e-4$ as shown in Figure 3.1. Looking at the individual sections in Figure 3.2, the squared difference between the points is focused on the leading edge of both suction and pressure side. The optimization progression is shown in Figure 3.3, a quick stepped descent until 100th iteration with steep steps in the beginning after which the optimization slope reduces to a stagnant slope before resuming the step decline from 175th run. This continues until 250th iteration before stopping at 300th iteration. Figure 3.4 contains the initial and final values of each of the twelve parameters from the optimization, the camber - which is the difference between in_beta and out_beta angles is near zero with the maximum thickness value being 0.12336033. The plotted graph values of each variable are in Figures 3.5 and 3.6.

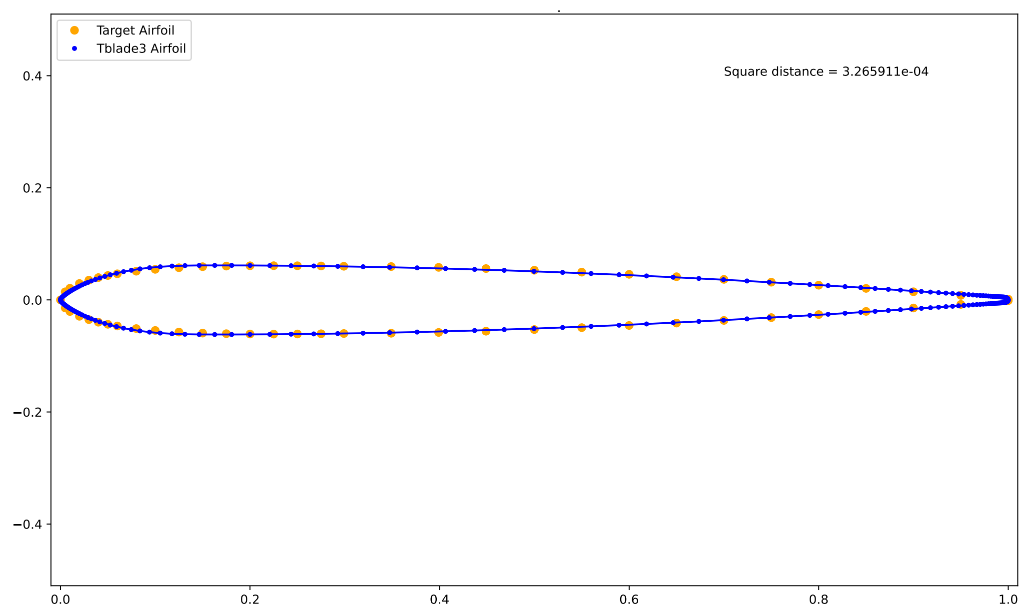


Figure 3.1: Least square difference between NACA 0012 airfoil and reverse engineered airfoil

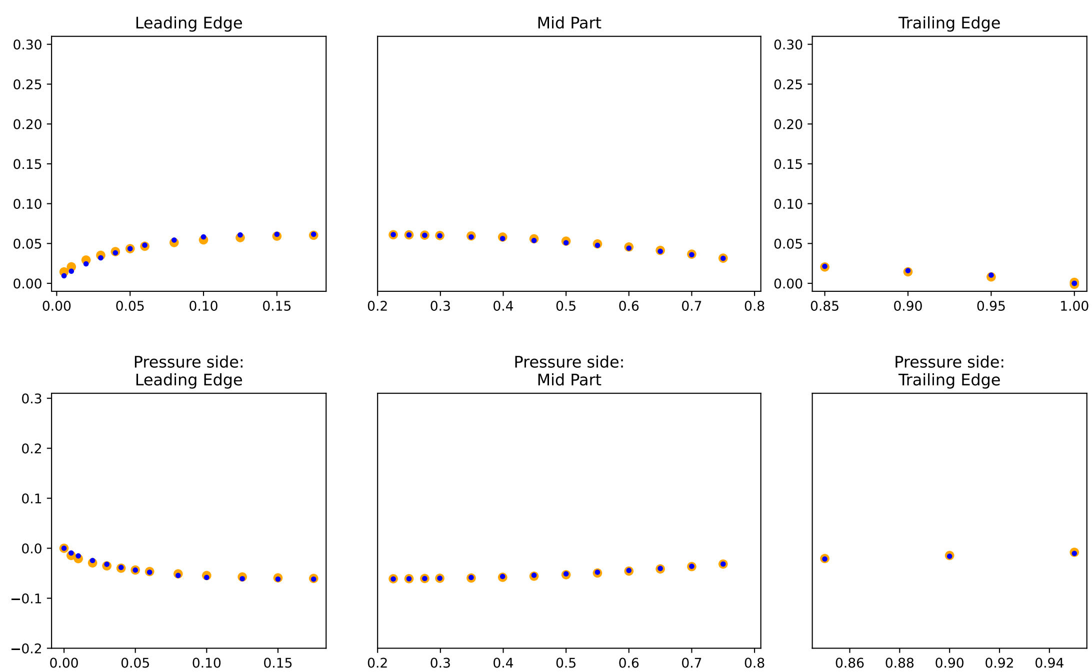


Figure 3.2: Tblade3 airfoil parts after interpolation to calculate least square difference - NACA 0012

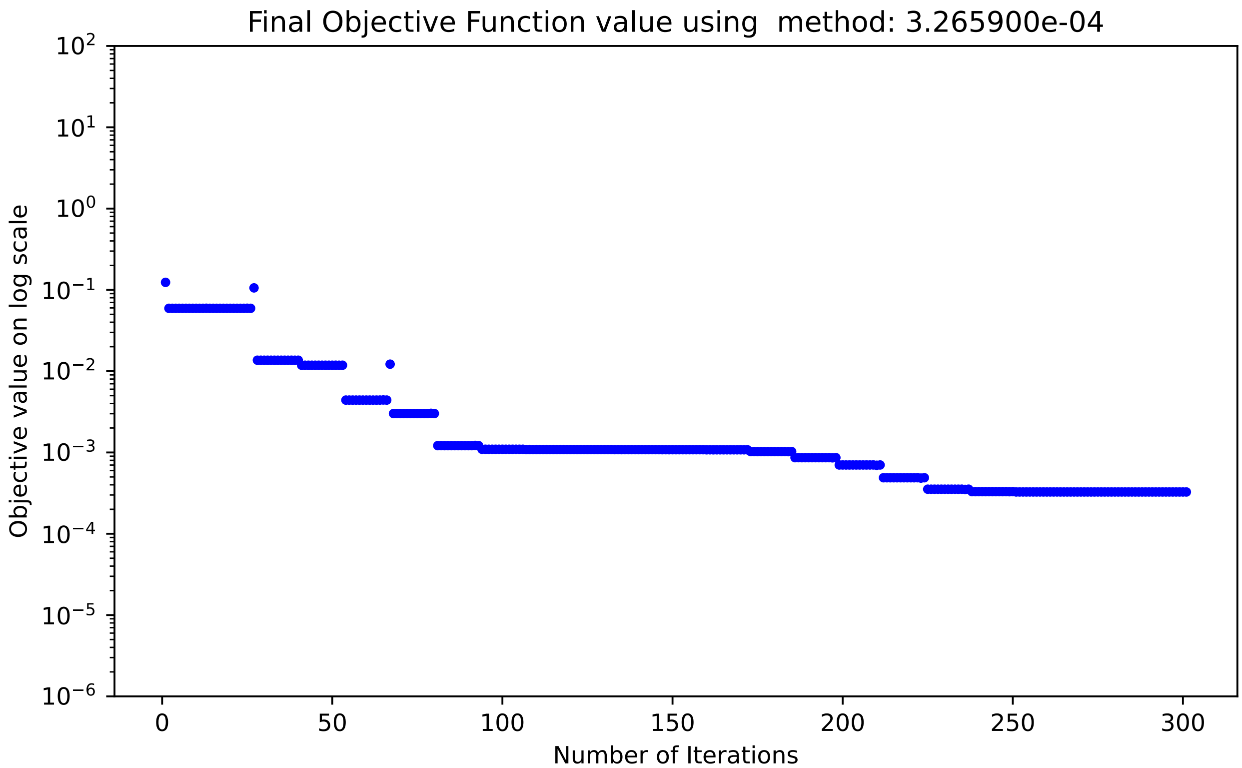


Figure 3.3: Least square difference between target airfoil and Tblade3 generated airfoil - NACA 0012

	Initial input values	Final input values
in_beta	3.0	1.46808322
out_beta	0.0	1.53708203
cur1	1.909	2.04102991
cur2	3.6695	3.71353521
cur4	1.458	1.21376261
cur5	0.8	0.79993379
cur6	0.82	0.98226726
cur7	0.81	0.90412942
LE_radius	2.5	2.78562328
u_max	0.55	0.16346221
t_max	0.246	0.12336033
t_TE	0.01548	0.0098933

Figure 3.4: Initial and Final parameter values - NACA 0012

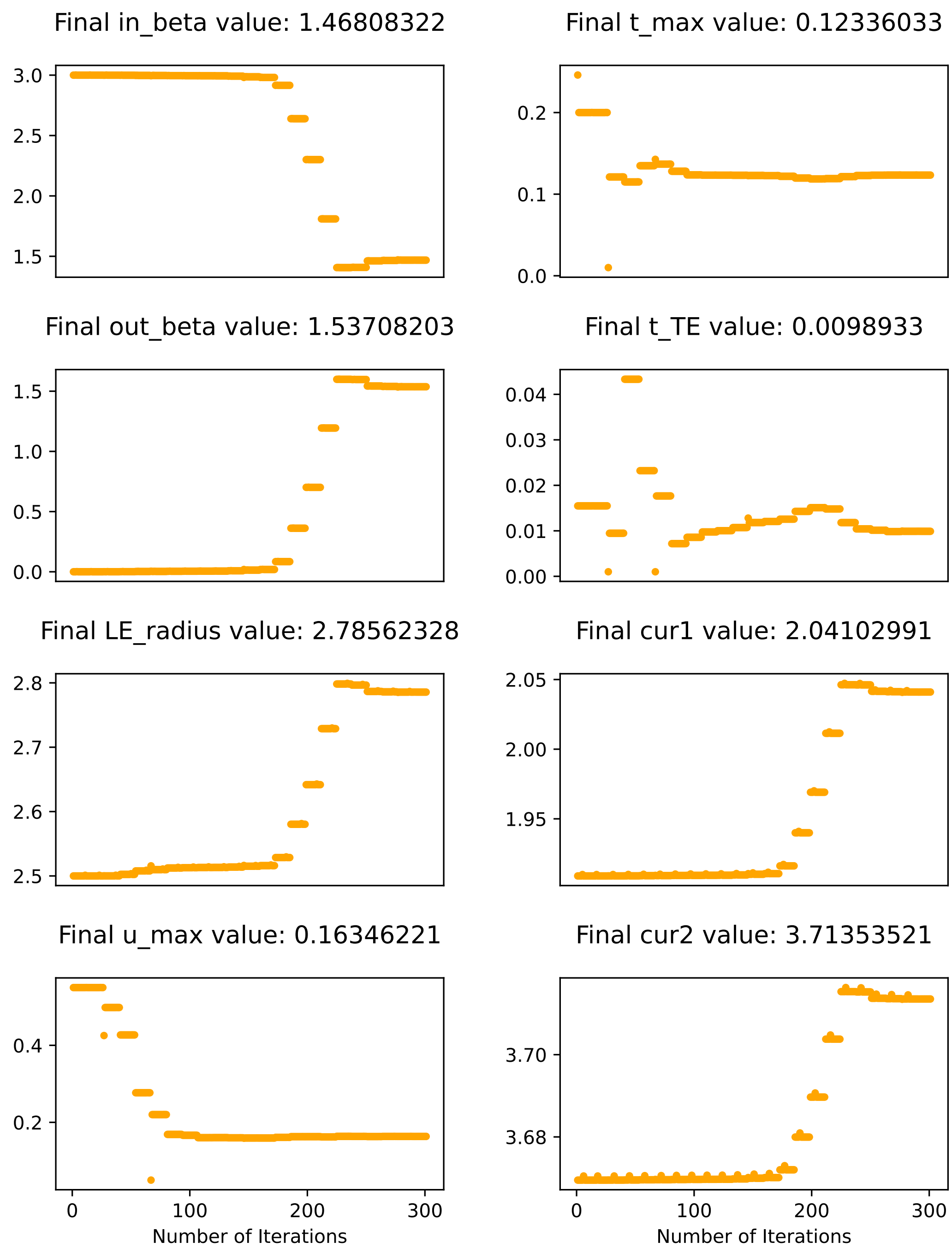


Figure 3.5: Initial and Final parameter plots - NACA 0012

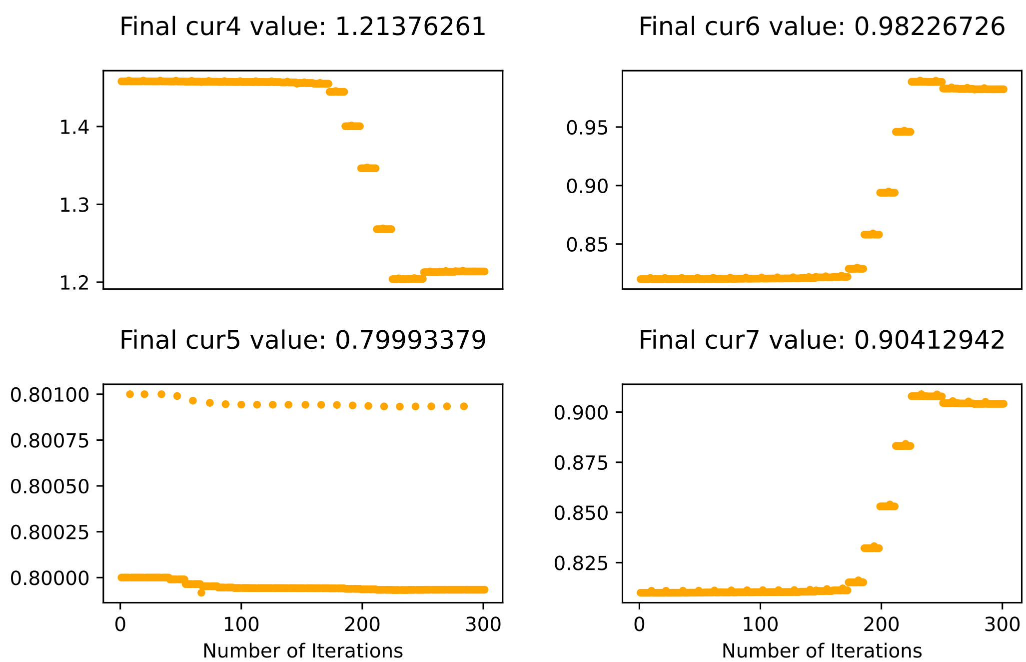


Figure 3.6: Initial and Final parameter plots - NACA 0012 (contd.)

3.2 NACA 2412

The NACA 2412 airfoil is a 2% camber airfoil at 40% chord with a max thickness of 12% at 30% chord.There were 63 unit chord coordinates obtained from airfoiltools website and it took around 450 gradient evaluations of the objective function to reach a least squared difference value of $1.5624e-04$ as shown in Figure 3.7. Looking at the individual sections in Figure 3.8, the squared difference between the points is focused on the leading edge of pressure side. The optimization progression is shown in Figure 3.9, a quick stepped descent before 50th, 200th, and 350th iterations with other areas having a dull slope. Figure 3.10 contains the initial and final values of each of the twelve parameters from the optimization, the camber is about 11.2 degrees with the maximum thickness value being 0.11943561. The plotted graph values of each variable are in Figures 3.12 and 3.11.

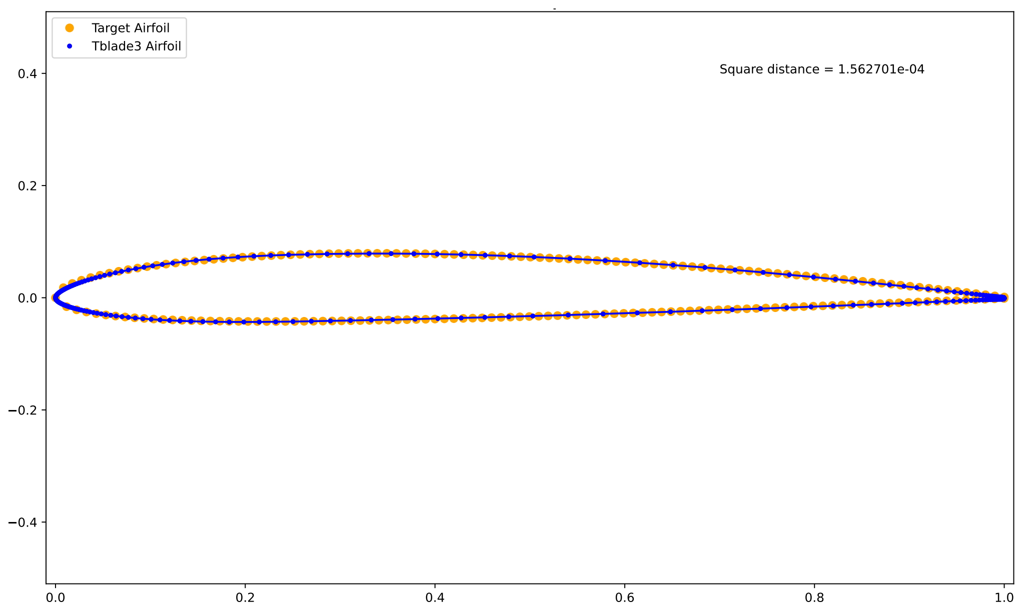


Figure 3.7: Least square difference between NACA 2412 airfoil and reverse-engineered airfoil

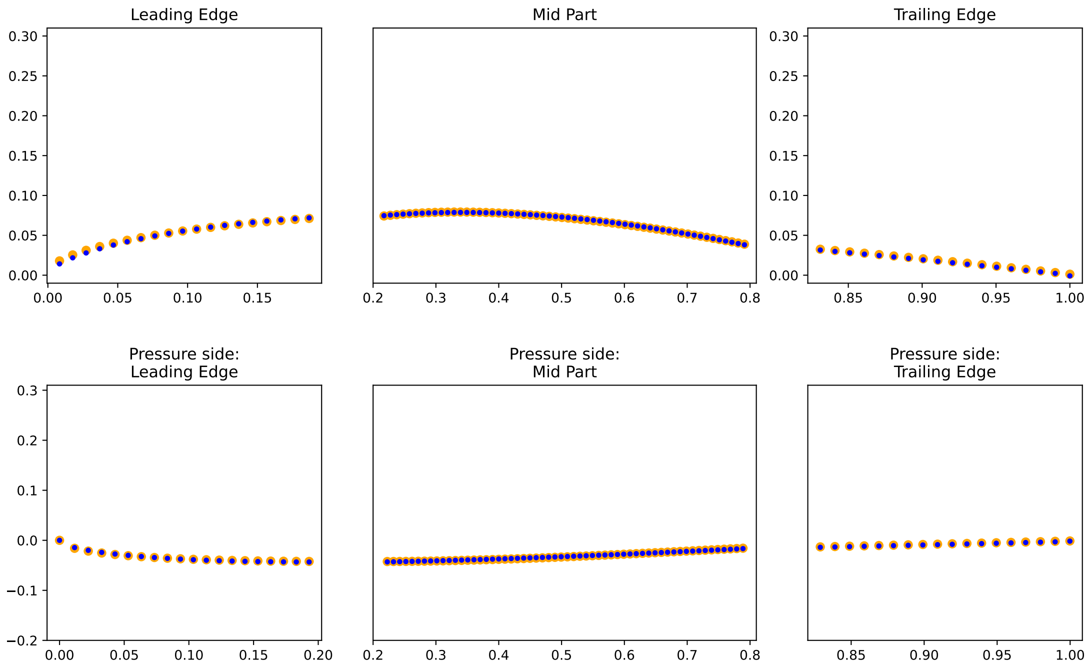


Figure 3.8: Tblade3 airfoil parts after interpolation to calculate least square difference - NACA 2412

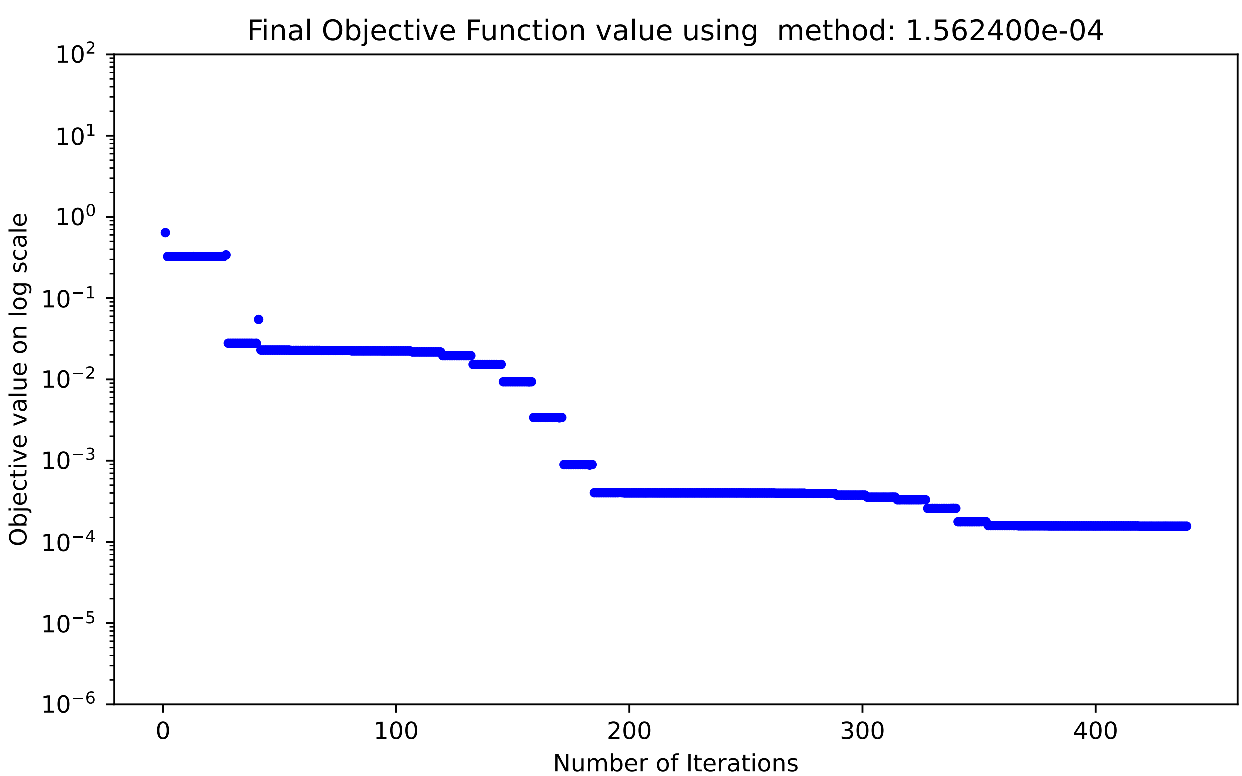


Figure 3.9: Least square difference between target airfoil and Tblade3 generated airfoil - NACA 2412

	Initial input values	Final input values
in_beta	3.0	7.10609787
out_beta	0.0	-4.14327427
cur1	1.909	1.55859773
cur2	3.6695	3.66767226
cur4	1.458	1.84772265
cur5	0.8	0.93696776
cur6	0.82	0.60758513
cur7	0.81	0.58956322
LE_radius	2.5	4.5
u_max	0.55	0.27822828
t_max	0.246	0.11943561
t_TE	0.01548	0.00302627

Figure 3.10: Initial and Final parameter values - NACA 2412

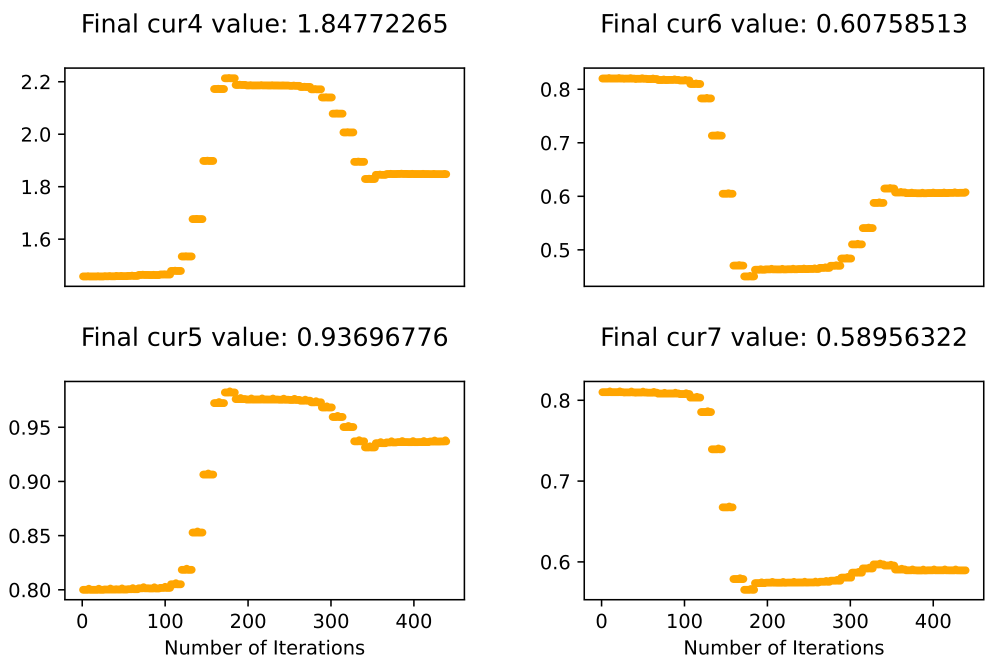


Figure 3.11: Initial and Final parameter plots - NACA 2412

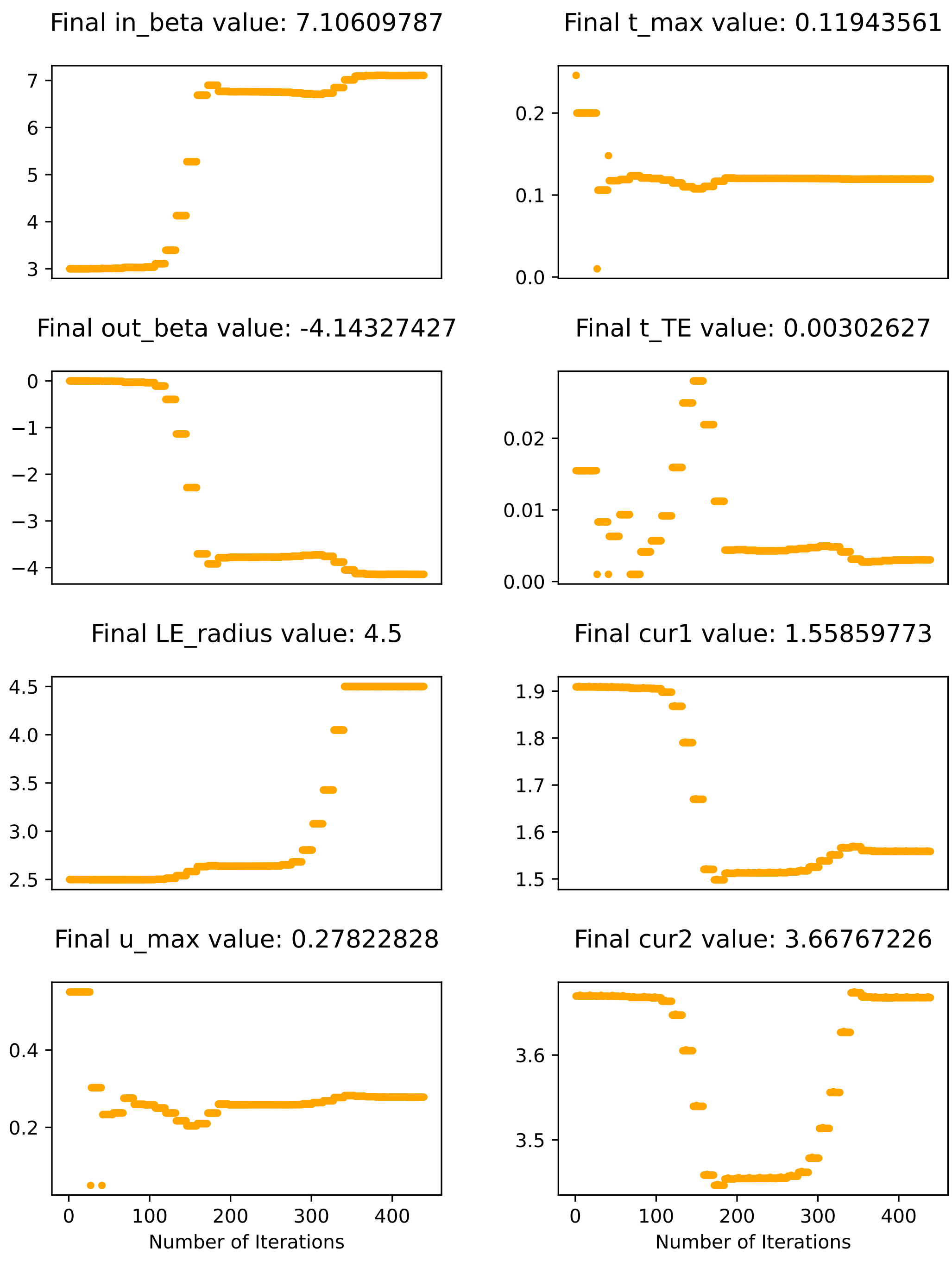


Figure 3.12: Initial and Final parameter values - NACA 2412

3.3 NREL s809

The NREL s809 airfoil is a wind turbine airfoil with Max thickness of 21% at 39.5% chord and max camber of 1% at 82.3% chord.There were 66 unit chord coordinates obtained from airfoiltools website and it took around 600 gradient evaluations of the objective function to reach a least squared difference value of $6.6977e-04$ as shown in Figure 3.13. Looking at the individual sections in Figure 3.14, the squared difference between the points is focused on the mid sections of both the suction and pressure side along with the trailing edge pressure side. The optimization progression is shown in Figure 3.15, a quick stepped descent until 100th iteration with steep steps in the beginning after which the optimization slope reduces to a stagnant slope which continues until end of the simulation, this stagnation is attributed to the near-zero tip value. Figure 3.16 contains the initial and final values of each of the twelve parameters from the optimization, the camber is near zero with the maximum thickness value being 0.21034362. The plotted graph values of each variable are in Figures 3.18 and 3.17.

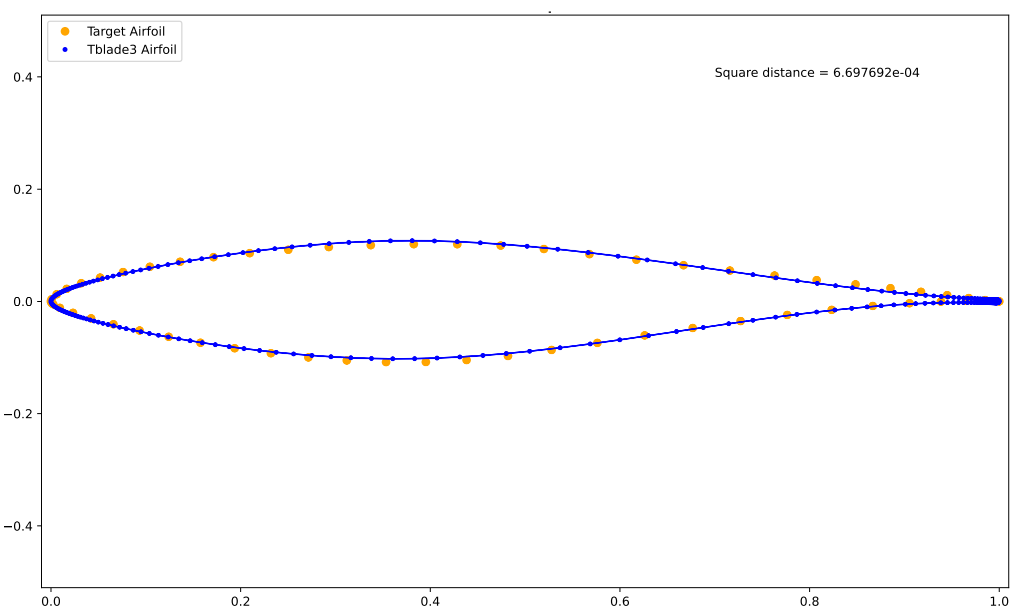


Figure 3.13: Least square difference between NREL s809 airfoil and reverse engineered airfoil

With the three types of 2D airfoil cases demonstrated we can move on to 3D airfoil demonstration cases, where we compare the traditional blade design method and look at why our reverse engineering method is needed.

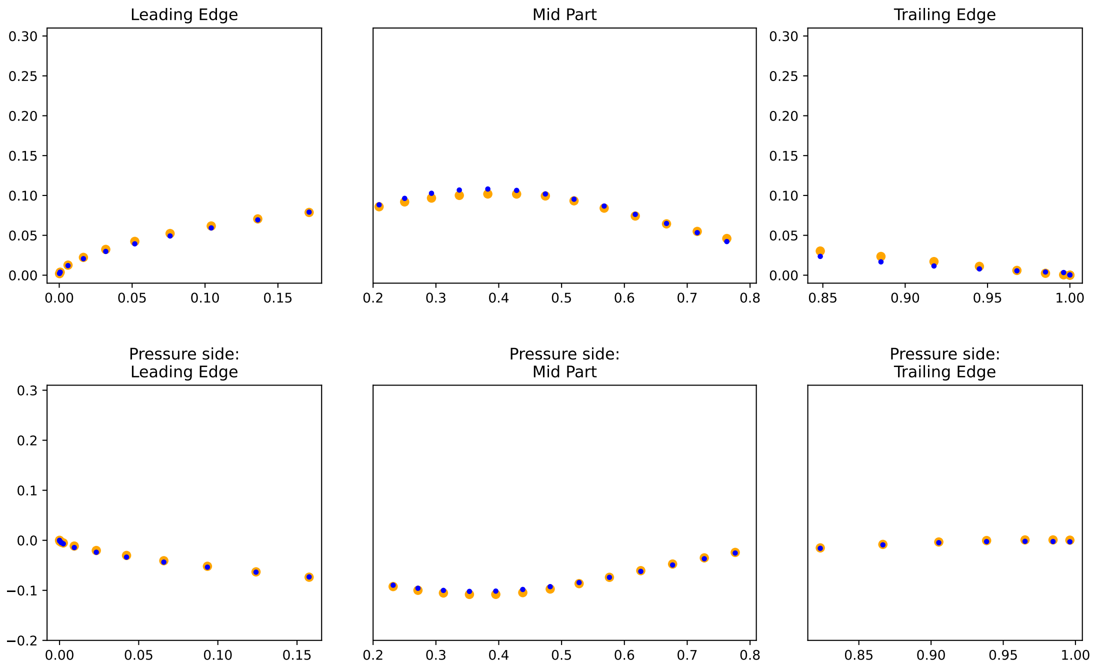


Figure 3.14: Tblade3 airfoil parts after interpolation to calculate least square difference - NREL s809

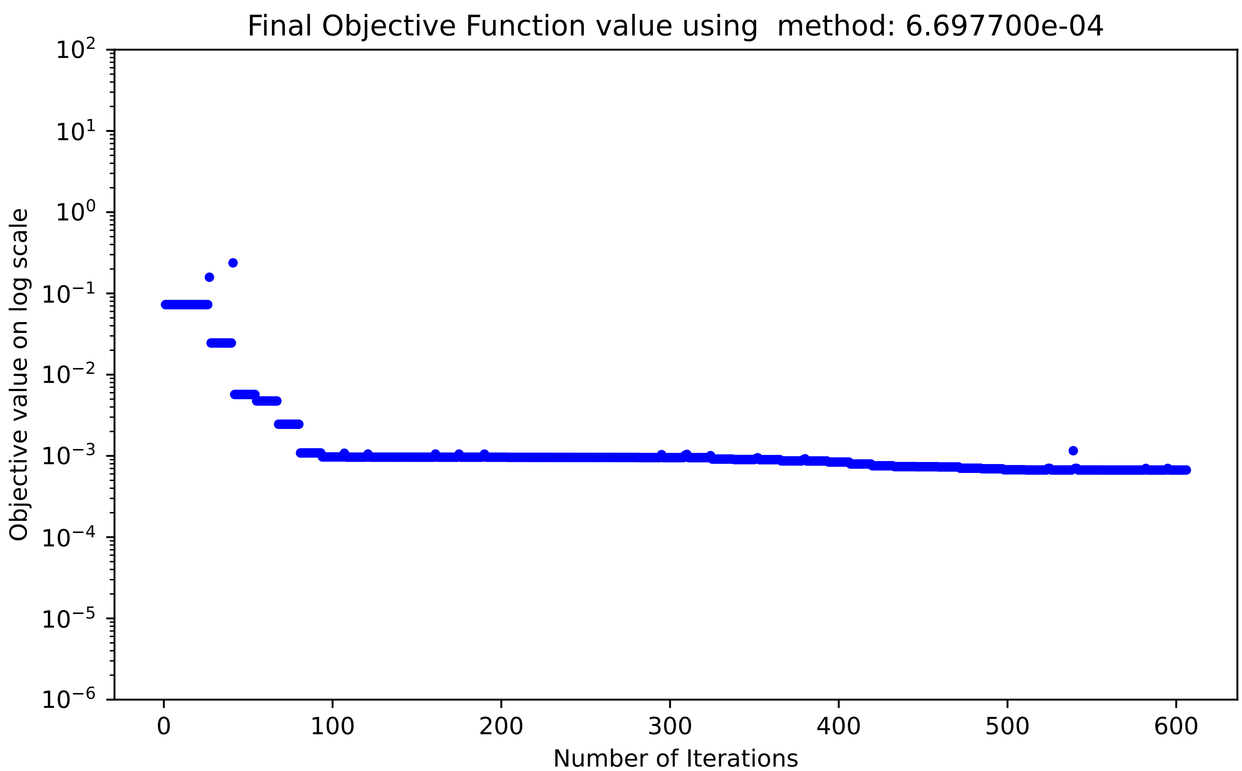


Figure 3.15: Least square difference between target airfoil and Tblade3 generated airfoil - NREL s809

	Initial input values	Final input values
in_beta	1.0	2.63609743
out_beta	-1.0	-2.62784313
cur1	-9.909	-9.80517789
cur2	3.6695	3.78786631
cur4	1.458	1.53725348
cur5	-5.8	-5.8688854
cur6	-9.82	-9.81997424
cur7	-9.81	-9.78749062
LE_radius	2.5	2.71227853
u_max	0.55	0.37184864
t_max	0.246	0.21034362
t_TE	0.0001548	0.00620374

Figure 3.16: Initial and Final parameter values - NREL s809

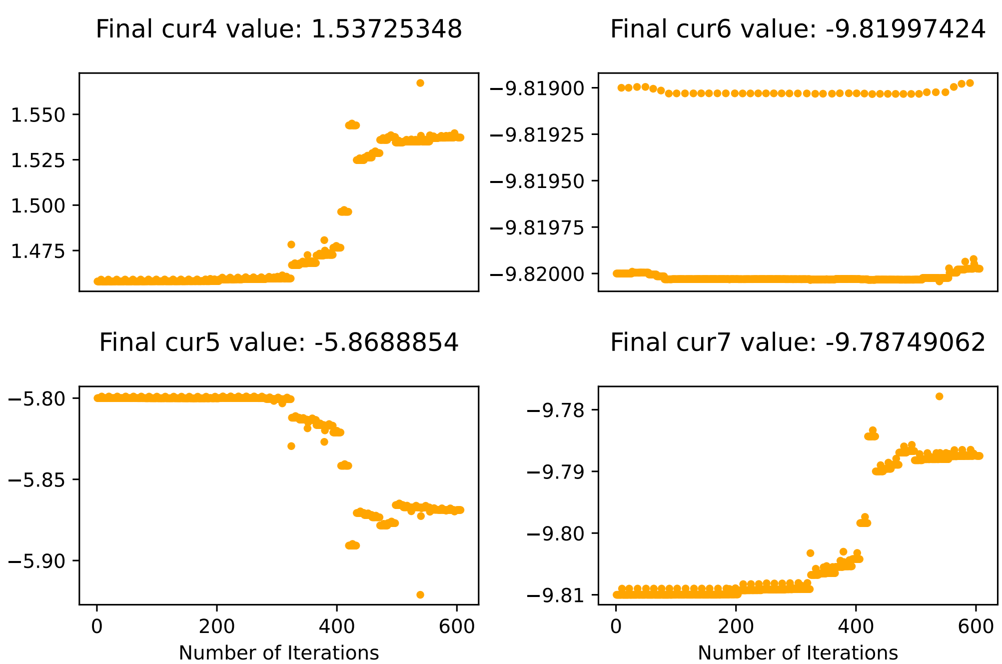


Figure 3.17: Initial and Final parameter values - NREL s809

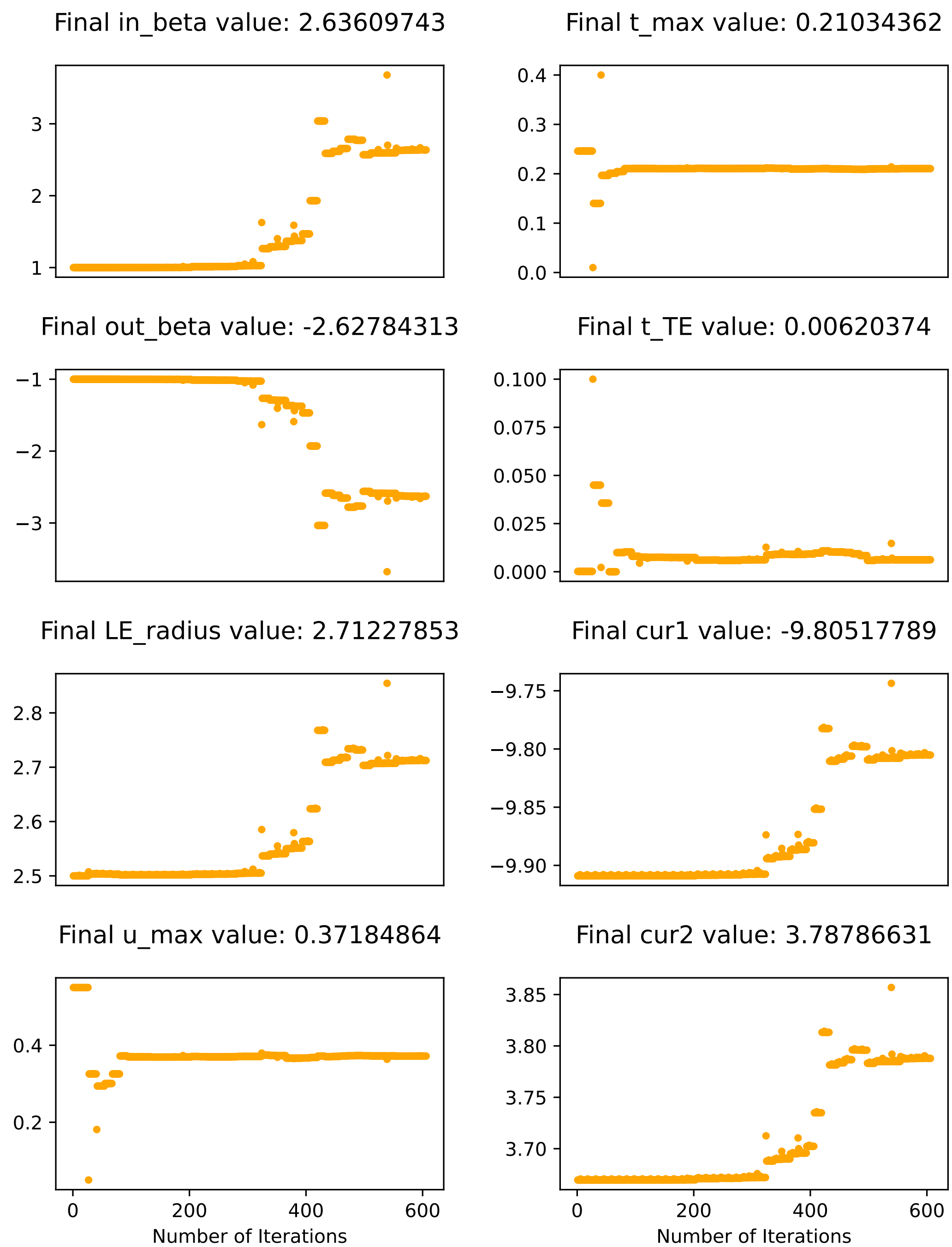


Figure 3.18: Initial and Final parameter values - NREL s809

Chapter 4

3D Blade design of E3 fan

The reverse engineering framework applied to 3D blade is demonstrated in the following sections. We look at a manual specification process that would traditionally be carried out in Tblade3 and contrast that with using reverse engineering framework for the blade design. For simplicity reasons and to focus on the blade design, the E3 Fan blade will be designed without a part-span shroud.

The E3 fan report has a snapshot of the fan configuration - Figure 4.1 and its appendix has two tables for rotor R1 describing the design properties that give further insight on the flow regime as shown in Figure 4.2 and Figure 4.3. Using information from these tables and by plot-digitizing the hub and casing we move to the first step in blade design.

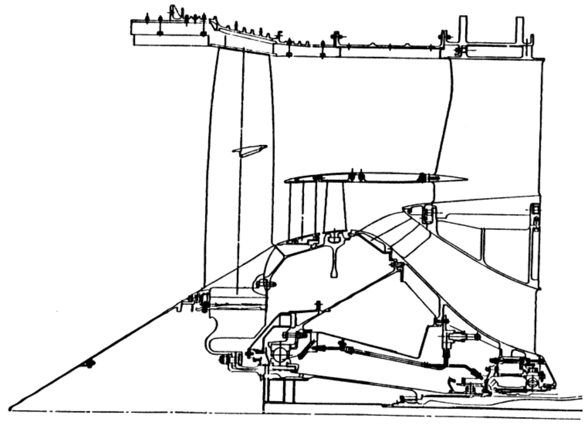


Figure 4.1: E3 report - fan configuration

BLADE ROW PRINTOUT FOR R1

(METRIC) UNITS *** EEE FAN STATION DATA

PCT EDGE IMM IN	OUT	AXIAL LOC-Z IN	OUT	RADIUS IN	OUT	STREAM FUNCT IN	OUT	MERID ANGLE IN	OUT	ABS ANGLE IN	OUT	REL ANGLE IN	OUT	TOTAL PRESS IN	OUT	TOTAL TEMP IN	OUT
0.	0.	0.467	15.961	105.167	102.995	0.	0.	-7.97	-7.97	0.	39.60	63.27	66.32	101.33	169.21	288.2	346.9
7.32	8.29	0.050	16.305	100.110	98.424	0.100	0.100	-4.20	-3.89	0.	38.32	61.00	62.01	101.33	172.26	288.2	344.3
14.34	15.60	-0.347	16.674	95.260	94.355	0.200	0.200	-1.80	-0.82	0.	38.20	60.56	49.69	101.33	174.22	288.2	342.3
21.62	23.70	-0.707	17.062	90.230	90.211	0.300	0.300	0.46	1.66	0.	37.96	56.63	46.71	101.33	174.79	288.2	340.5
29.30	30.99	-0.900	17.350	84.922	85.915	0.400	[illegible]	2.85	3.62	0.	36.42	54.86	43.56	101.33	173.95	288.2	339.1
37.50	39.29	-1.173	17.646	79.254	81.340	0.500	0.500	5.13	6.72	0.	39.37	53.09	40.20	101.33	171.64	288.2	337.7
42.73	44.61	-1.213	17.657	76.639	78.412	0.560	0.560	6.61	7.26	0.	39.76	51.89	38.13	101.33	169.47	288.2	336.2
44.77	46.67	-1.237	17.699	74.234	77.273	0.583	0.583	7.04	8.07	0.	39.84	51.41	37.30	101.33	168.62	288.2	335.6
44.77	47.06	-1.237	17.706	74.234	77.057	0.583	0.583	7.04	9.09	0.	40.04	51.41	37.16	101.33	168.62	288.2	336.6
48.25	50.64	-1.306	17.767	71.824	75.000	0.620	0.620	8.03	11.30	0.	40.21	50.55	36.72	101.33	166.76	288.2	334.4
51.14	53.60	-1.364	17.793	69.829	73.453	0.650	0.650	8.94	12.23	0.	40.23	49.83	34.45	101.33	165.16	288.2	333.4
56.16	58.00	-1.420	17.783	66.362	70.588	0.700	0.700	10.60	13.77	0.	40.52	48.56	32.40	101.33	162.32	288.2	331.6
61.40	64.42	-1.449	17.782	62.683	67.491	0.750	0.750	12.93	15.67	0.	41.60	47.22	30.91	101.33	158.88	288.2	329.5
64.40	67.63	-1.441	17.800	60.613	65.725	0.777	0.777	14.40	14.56	0.	41.91	46.53	29.62	101.33	157.01	288.2	328.3
64.40	67.63	-1.441	17.800	60.613	65.726	0.777	0.777	14.40	14.66	0.	41.91	46.53	29.62	101.33	157.01	288.2	328.3
69.64	73.02	-1.422	17.850	67.049	62.754	0.820	0.820	17.09	13.19	0.	41.03	45.50	26.61	101.33	154.52	288.2	326.7
76.49	79.76	-1.431	17.892	62.313	59.044	0.872	0.872	20.83	15.36	0.	39.63	44.30	20.62	101.33	151.17	288.2	324.6
76.49	79.76	-1.431	17.892	52.313	59.044	0.872	0.872	20.83	16.36	0.	39.63	44.30	20.62	101.33	151.17	288.2	324.6
80.58	83.54	-1.429	17.901	49.400	56.950	0.900	0.900	23.16	16.90	0.	39.30	43.79	17.92	101.33	149.35	288.2	323.5
89.95	90.81	-1.366	17.884	43.705	52.947	0.950	0.950	28.36	20.71	0.	40.03	43.23	11.90	101.33	145.91	288.2	322.2
100.00	100.00	-1.270	17.709	36.060	47.884	1.000	1.000	32.50	29.04	0.	39.95	42.12	1.46	101.33	142.07	288.2	322.0

MERID VEL IN	OUT	ABS VEL IN	OUT	REL VEL IN	OUT	TANG VEL IN	OUT	BLADE SPEED IN	OUT	ABS MACH NO IN	OUT	REL MACH NO IN	OUT	BLKG IN	OUT	RCU IN	OUT
200.7	170.5	200.7	231.0	460.6	311.6	0.	146.6	410.5	402.1	0.630	0.644	1.407	0.968	0.990	0.990	0.	15101.
217.2	186.0	217.2	236.0	447.1	301.7	0.	146.7	390.0	384.2	0.666	0.664	1.370	0.846	0.990	0.990	0.	14434.
227.5	187.4	227.5	238.5	435.9	289.6	0.	147.5	371.9	368.3	0.700	0.671	1.342	0.815	0.990	0.990	0.	13915.
232.0	191.3	232.0	242.6	421.0	270.9	0.	149.2	352.2	352.1	0.716	0.686	1.301	0.780	0.990	0.990	0.	13455.
233.6	192.7	233.6	245.7	405.6	265.6	0.	152.5	331.5	335.4	0.721	0.697	1.252	0.753	0.990	0.990	0.	13105.
233.3	191.6	233.3	247.4	387.5	250.3	0.	156.4	309.4	317.5	0.720	0.704	1.196	0.712	0.990	0.990	0.	12724.
233.1	190.9	233.1	247.4	376.2	241.9	0.	157.5	295.3	306.1	0.719	0.706	1.161	0.690	0.990	0.990	0.	12347.
233.0	190.9	233.0	247.6	371.0	239.1	0.	157.7	289.0	301.6	0.719	0.707	1.147	0.683	0.990	0.990	0.	12185.
233.0	191.1	233.0	248.0	371.0	238.4	0.	160.1	289.0	300.0	0.719	0.708	1.147	0.681	0.990	0.990	0.	12185.
233.0	191.1	233.0	248.2	364.6	233.0	0.	158.4	280.4	293.1	0.719	0.710	1.125	0.669	0.990	0.990	0.	11892.
232.9	191.5	232.9	248.5	358.5	230.6	0.	158.3	272.6	286.7	0.719	0.712	1.106	0.661	0.990	0.990	0.	11629.
232.7	190.5	232.7	247.6	348.2	223.0	0.	158.1	259.0	275.6	0.718	0.712	1.074	0.643	0.990	0.990	0.	11163.
232.3	183.9	232.3	242.0	337.4	212.3	0.	157.3	244.7	263.5	0.717	0.696	1.041	0.611	0.990	0.990	0.	10620.
231.6	180.0	231.6	239.5	331.1	206.3	0.	157.1	236.6	256.6	0.714	0.690	1.021	0.594	0.990	0.990	0.	10324.
231.6	180.0	231.6	239.5	331.1	206.3	0.	157.1	236.6	256.6	0.714	0.690	1.021	0.594	0.990	0.990	0.	10324.
229.0	186.4	229.0	244.4	319.4	205.7	0.	158.0	222.7	245.0	0.705	0.707	0.984	0.595	0.990	0.990	0.	9913.
223.2	190.5	223.2	254.0	302.5	211.1	0.	158.5	204.2	230.5	0.686	0.741	0.930	0.616	0.990	0.990	0.	9358.
223.2	190.5	223.2	254.0	302.5	211.1	0.	158.5	204.2	230.6	0.686	0.741	0.930	0.616	0.990	0.990	0.	9358.
219.2	203.6	219.2	250.6	292.2	213.1	0.	159.4	193.2	222.3	0.672	0.757	0.896	0.624	0.990	0.990	0.	9077.
206.2	210.3	206.2	267.4	267.7	214.3	0.	155.2	170.6	206.7	0.629	0.780	0.817	0.631	0.990	0.990	0.	8749.
184.6	247.7	184.6	307.0	232.2	247.7	0.	181.4	140.0	186.9	0.559	0.923	0.703	0.745	0.990	0.990	0.	8606.

Figure 4.2: E3 report - fan data table 1

BLADE ROW PRINTOUT FOR R1

(METRIC) UNITS *** EEE FAN STATION DATA

PCT IMM	RBAR	INC (INPUT)	X-FACT	DEV (C-R)	TURN	LOSS COEFF	CAMBER	STGR	SOL	TMC	D-FACT	CHORD	AXIAL VEL-R	EFFICIENCY ADIA	POLY	ACC PT RATIO	ACC TT RATIO
0.	104.001	6.00	0.	2.48	7.96	0.178	5.43	55.55	1.4062	0.0252	0.435	11.314	0.855	0.775	0.791	1.670	1.2038
7.76	99.267	4.66	0.	2.56	8.99	0.127	7.29	52.69	1.4384	0.0275	0.430	11.030	0.857	0.841	0.863	1.700	1.1948
14.93	94.800	4.95	0.	3.22	8.87	0.087	7.14	50.04	1.4781	0.0295	0.449	10.833	0.824	0.893	0.901	1.719	1.1878
22.32	90.220	6.50	0.	3.43	9.91	0.059	7.84	47.20	1.5132	0.0320	0.456	10.554	0.824	0.929	0.935	1.725	1.1817
30.05	85.419	5.66	0.	3.93	11.31	0.047	9.50	44.42	1.5427	0.0360	0.468	10.187	0.824	0.945	0.949	1.717	1.1769
38.30	80.297	5.70	0.	4.56	12.09	0.047	11.75	41.51	1.5890	0.0403	0.483	9.863	0.820	0.947	0.951	1.694	1.1716
43.56	77.025	5.73	0.	4.90	13.76	0.044	12.93	39.70	1.6221	0.0424	0.480	9.650	0.818	0.951	0.956	1.673	1.1667
45.61	75.754	5.60	0.	5.05	14.11	0.043	13.47	38.99	1.6369	0.0429	0.489	9.586	0.817	0.952	0.956	1.663	1.1645
45.79	75.646	5.65	0.	5.05	14.25	0.043	13.64	38.94	1.6382	0.0430	0.491	9.580	0.814	0.952	0.956	1.663	1.1645
49.31	73.456	5.65	0.	5.26	14.82	0.042	14.43	37.60	1.6646	0.0439	0.492	9.452	0.812	0.954	0.957	1.646	1.1606
52.23	71.641	6.55	0.	5.45	15.30	0.041	15.78	36.64	1.6872	0.0448	0.491	9.344	0.813	0.956	0.958	1.630	1.1570
67.33	68.476	6.00	0.	5.90	16.16	0.039	17.06	35.03	1.7282	0.0463	0.493	9.146	0.809	0.958	0.960	1.602	1.1508
62.79	65.007	4.76	0.	6.27	16.31	0.039	17.82	33.56	1.7731	0.0484	0.507	8.921	0.783	0.958	0.960	1.568	1.1434
66.00	63.169	4.60	0.	6.46	16.91	0.039	18.69	32.50	1.7982	0.0500	0.514	8.701	0.700	0.958	0.958	1.550	1.1394
66.00	63.169	4.60	0.	6.46	16.91	0.039	18.69	32.60	1.7982	0.0500	0.514	8.781	0.780	0.958	0.960	1.550	1.1394
71.14	59.902	4.65	0.	6.86	19.89	0.039	22.10	29.79	1.8424	0.0531	0.497	8.631	0.829	0.959	0.961	1.525	1.1339
77.94	55.670	4.72	0.	7.40	23.77	0.039	26.53	26.40	1.9140	0.0583	0.447	8.230	0.917	0.960	0.962	1.492	1.1264
77.94	55.670	4.72	0.	7.40	23.77	0.039	26.53	26.40	1.9140	0.0583	0.447	8.230	0.917	0.960	0.962	1.492	1.1264
81.89	53.223	4.80	0.	7.99	25.07	0.042	29.06	24.46	1.9628	0.0618	0.419	8.076	0.966	0.960	0.960	1.474	1.1226
89.70	48.326	5.25	0.	9.50	31.33	0.077	35.66	20.15	2.1005	0.0714	0.360	7.847	1.004	0.931	0.934	1.440	1.1182
100.00	41.976	3.50	0.	12.63	40.66	0.167	49.79	13.72	2.3976	0.0965	0.119	7.700	1.391	0.881	0.886	1.410	1.1173

INLET CORR WTFLOW	PRESS RATIO	TEMP RATIO	ADIA EFF	INLET CORR RPM	NB	AR
643.72	1.6342	1.1646	0.917	3727.7	32	2.66

Figure 4.3: E3 report - fan data table 2

4.1 Preparation of Input files

4.1.1 3dbgbinput file setup

The primary input file for Tblade3 is "3dbgbbuilder.1.dat". It takes in the metal angles, axial and radial coordinates, and surface of rotation coordinates all of which were obtained from the report except for metal angles which were along the planar sections of the blade but the input file needed them in streamline sections. This meant that the metal angles had to be estimated along the streamlines.

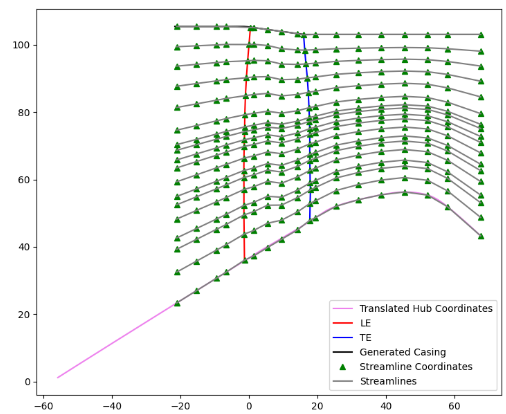


Figure 4.4: Generated E^3 surfaces of revolution for Tblade3

Along with allowing control over the airfoil section's metal angles, Tblade3 also allows these sections to be placed along surfaces of revolutions. Since page 12 of the report had the hub and casing positions, and the Appendix A had LE_z, TE_r, TE_z, and TE_r values. A python script (Appendix C.1), was written to generate these streamlines or surfaces of revolutions for Tblade3 as shown in Figure 4.4.

Now that we have the primary input file ready, we next move on to gather input values to control the curvature of the blade.

4.1.2 Spancontrolinputs file setup

The secondary input file for Tblade3 is the spancontrolinputs file. This input file has two tables, the first one helps to control the curvature of the camber and the second one helps control the LE and TE tip thickness, and location and value of maximum thickness. Three spans of the blade were chosen to control these parameters: 0, 55, and 100. Both table parameters were defined along these spans. All the curvature points in the first table were obtained through traditional manual parameter specification of $(u - v)$ sections. The second table within the spancontrolinputs file, however, was a bit different. While the LE and TE tip thickness points were estimated using trial and error, the location and value of maximum thickness was obtained from the E^3 report. The plots in the report - Figure 4.5, Figure 4.6, Figure 4.7 were digitized and the values of total thickness at the required spans were linearly interpolated. Once we had both the input files ready, a final plotting script was written to visually compare (Figure 4.9) the output files from Tblade3 and plot digitized airfoil sections from $E3$ report. While the reports do offer relevant information, it is not a complete list of airfoil design data. To obtain that CAD info, we will be using two methods – first is a traditional manual parameter specification based, and another using the framework in the next sections.

4.2 Generation of input parameters

We start by looking at the report to analyze the information available about the three airfoils - hub, part-span shroud, and tip. The hub section's max thickness location is at 42% chord and has a tm/c value of 0.125 - Figure 4.5. Flow in the hub is subsonic with a Mach number of 0.70 at inlet rising to 0.75 at the outlet. Part span shroud section's maximum thickness location is at 55% chord with a tm/c value of 0.044 - Figure 4.6. Flow at inlet has a Mach number of 1.15 with an exit mach number of 0.68. Moving on, tip section's maximum thickness location is at 59% chord with a tm/c value of 0.024 - Figure 4.7. Flow at the inlet has the highest Mach number of 1.41 near the tip section falling to a transonic range of 0.87 at the outlet of the tip. The report also mentions the presence of a favorable oblique shock near the part-span shroud and tip section airfoil before the normal shock at the design case. With these paramaters in mind, we then move to the first approach.

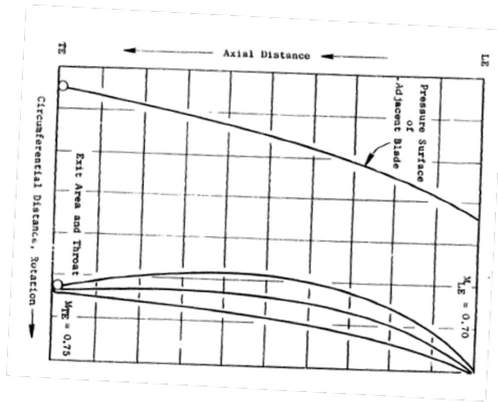


Figure 4.5: E3 engine fan - Hub airfoil section

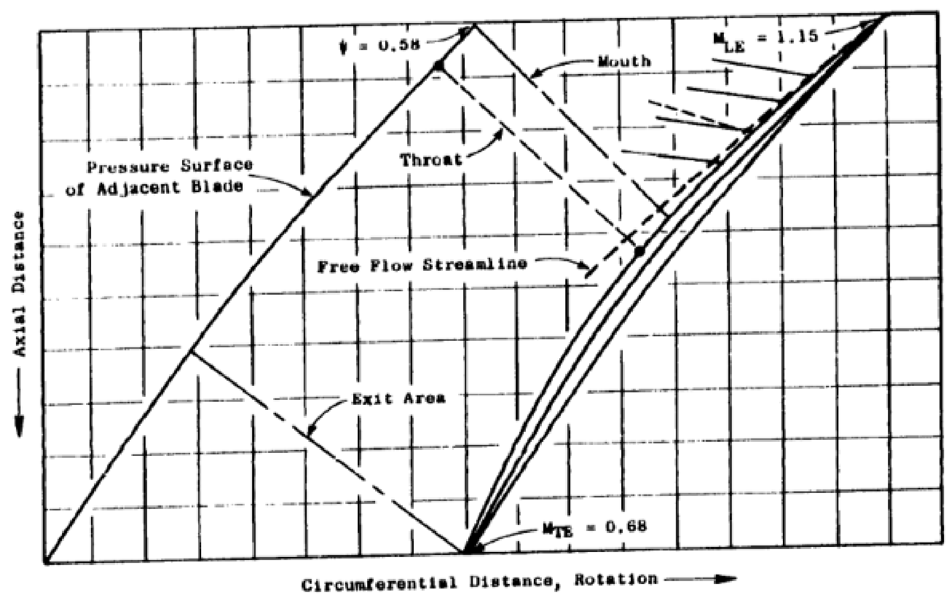


Figure 4.6: E3 engine fan - part-span shroud airfoil section

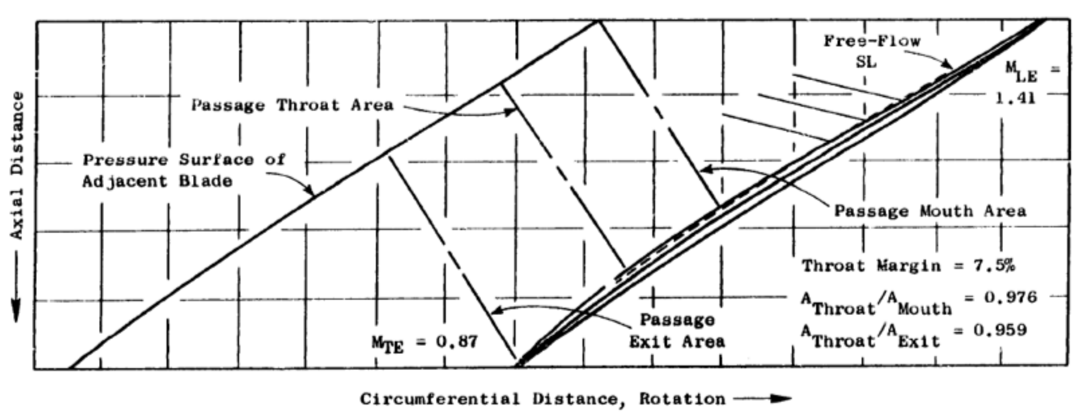
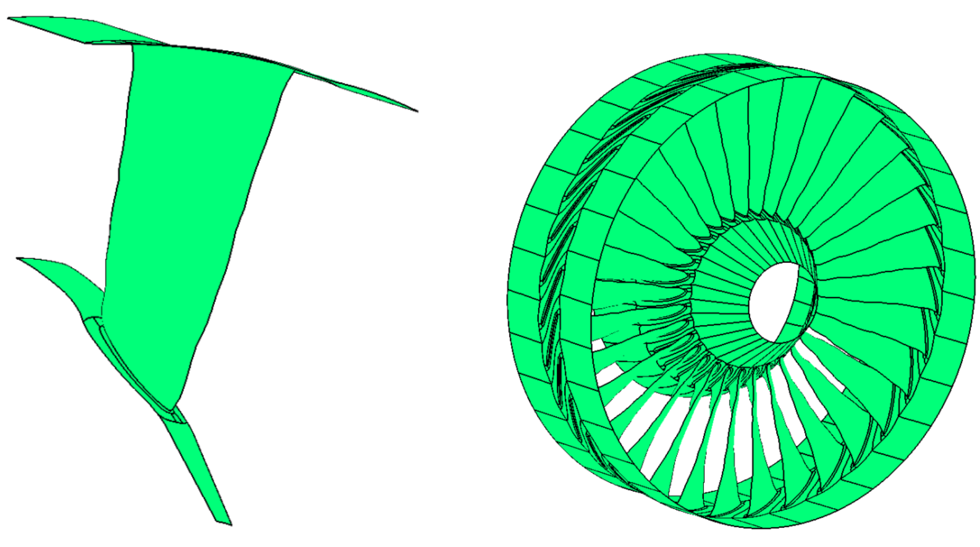


Figure 4.7: E3 engine fan - tip airfoil section

4.2.1 Traditional manual parameter specification method

Figure 4.9 shows the comparison of generated airfoil section coordinates - shown in blue and plot digitized streamlines sections shown in orange. The coordinates were normalized and rotated such that the stagger is zero and can be expressed in $(u-v)$ coordinate system with the u value between 0 and 1.

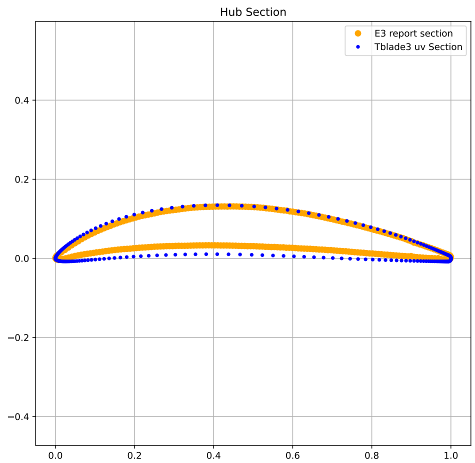
The sections in Figure 4.9 were the closest that we could obtain from traditional manual parameter specification. Figure 4.8a shows the 3D blade view of a single blade passage as visualized along with its hub and casing. On its right is its full-annulus view with all 32 blade passages. Here, we can clearly see the cone like hub. Once the blade sections were close enough to the streamline sections from the report, the Tblade3 output files were converted into a .geomturbo file format to carry out grid generation. The input values parameter values for these files were only precise until the second decimal and can be found in Appendix E.1.



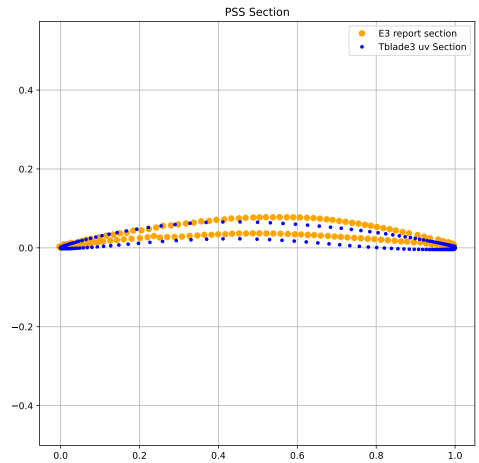
(a) Single blade with hub and casing (b) Full-annulus view

Figure 4.8: 3D blade shape from traditional manual parameter specification in Tblade3

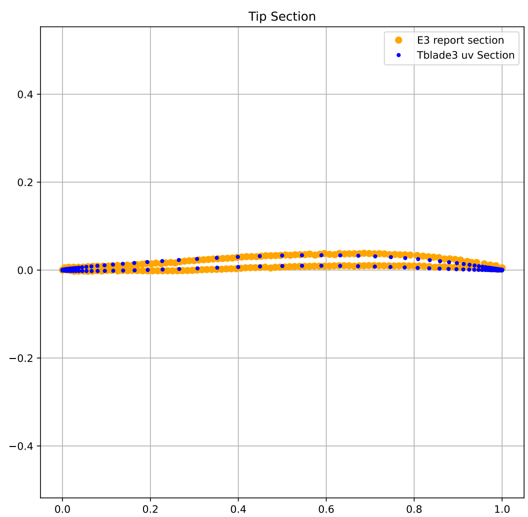
This traditional manual parameter specification approach takes a lot of back-and-forth and is not ideal when precision is needed. Reverse engineering framework is bought in next to demonstrate how quick and more precise the process can be.



(a) At 0.00% span



(b) At 55.00% span



(c) At 100.00% span

Figure 4.9: E3 Sections Comparison - sections from traditional manual parameter specification of input parameters

4.2.2 Reverse engineering framework method

E3 Hub Section

The stopping criterion was set to $1e^{-03}$ and for the plot digitized hub it took around 350 gradient evaluations for the objective function to reach a least squared difference value of $9.0006e-04$ as shown in Figure 4.10. We can clearly see the difference in closeness when compared to Figure 4.9a. Looking at the individual sections in Figure 4.11, the squared difference between the points is the minimum near the mid-section points and increases as we move towards the extremes with the highest difference at the suction side trailing edge. The optimization progression is shown in Figure 4.12, a steep descent until 150th iteration is observed with a steady descent until 200th iteration followed by stagnation and some noise before the optimization stops near 350th iteration. Figure 4.13 contains the initial and final values of each of the twelve parameters from the optimization, the location of maximum thickness is at 0.4565529 with the maximum thickness value being 0.09702214. The plotted graph values of each of the 12 variables is in Figure 4.15 and Figure 4.14.

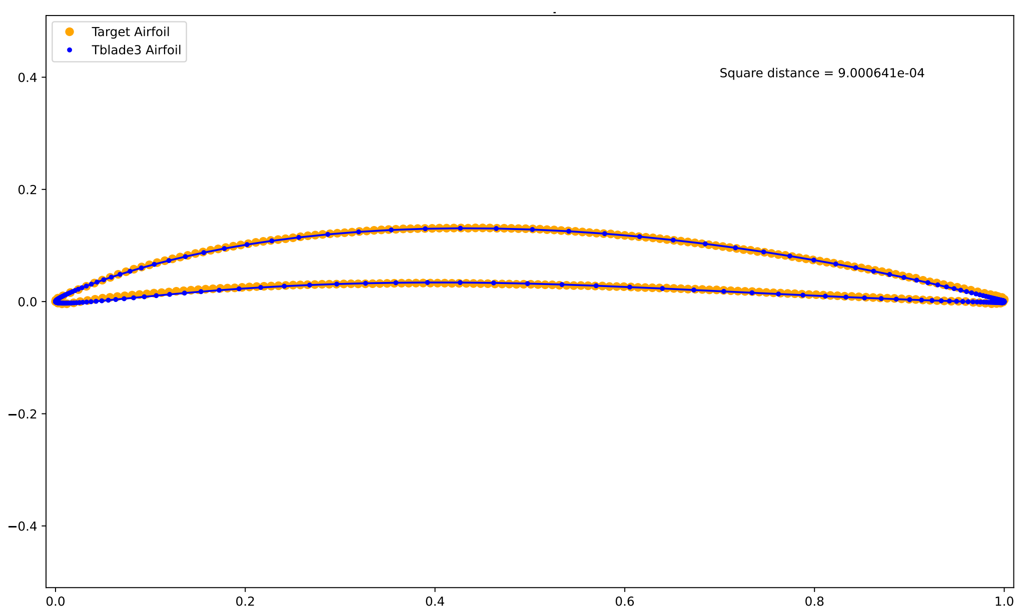


Figure 4.10: Least square difference between plot digitized hub and reverse engineered airfoil

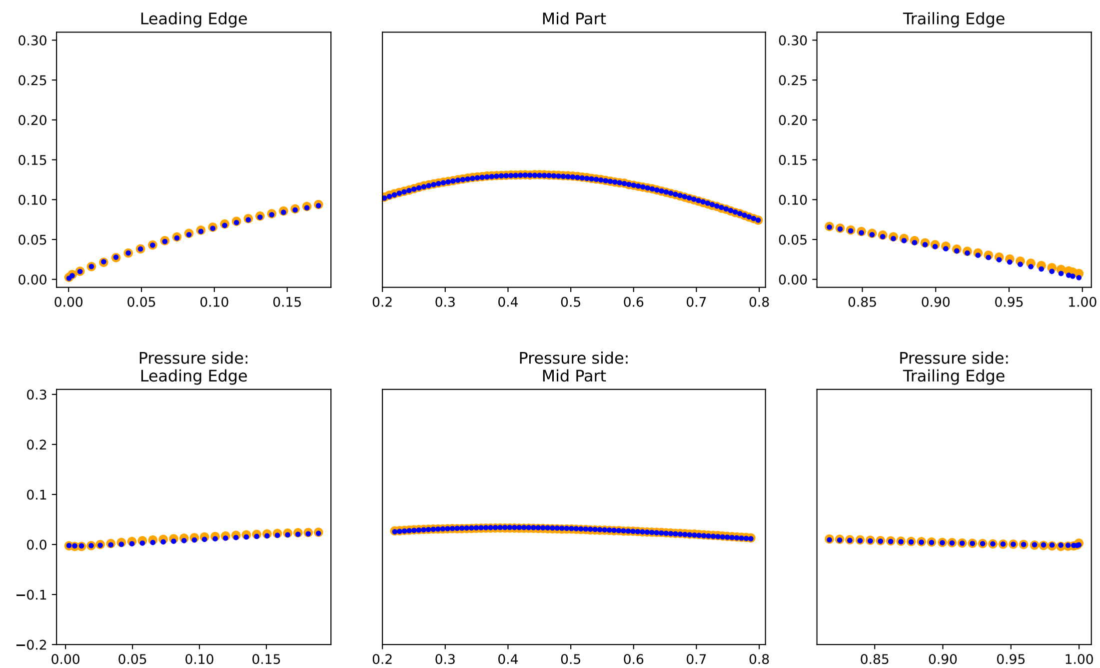


Figure 4.11: Reverse engineered section comparison for E3 hub airfoil

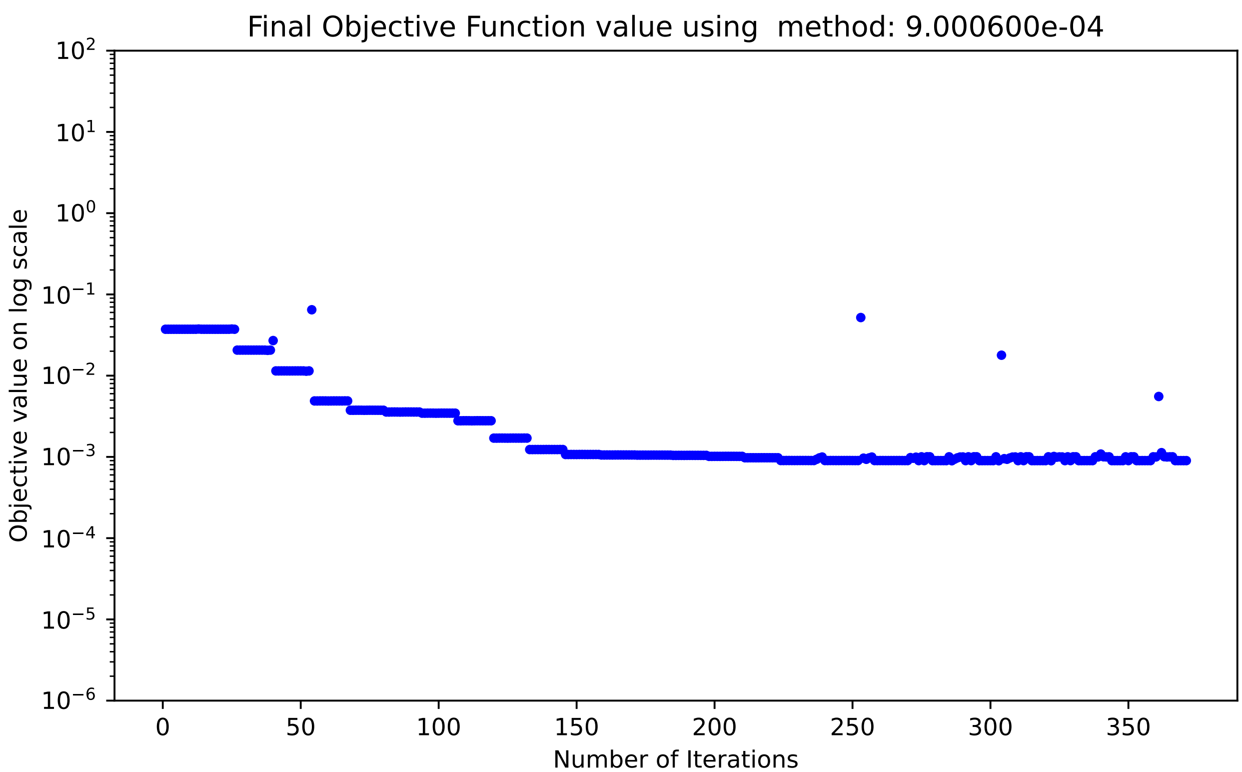


Figure 4.12: Reverse engineered section - Objective function vs iterations for E3 hub

	Initial input values	Final input values
in_beta	32.79	33.01142082
out_beta	-7.87	-8.07361942
cur1	1.909	1.69591536
cur2	3.6695	3.631585
cur4	1.458	1.75181355
cur5	0.8	0.70739264
cur6	0.82	0.5110641
cur7	0.81	0.63949916
LE_radius	2.5	2.35916208
u_max	0.42	0.4565529
t_max	0.124	0.09702214
t_TE	0.01548	0.0036372

Figure 4.13: Reverse engineered Hub section - Initial and Final parameter values

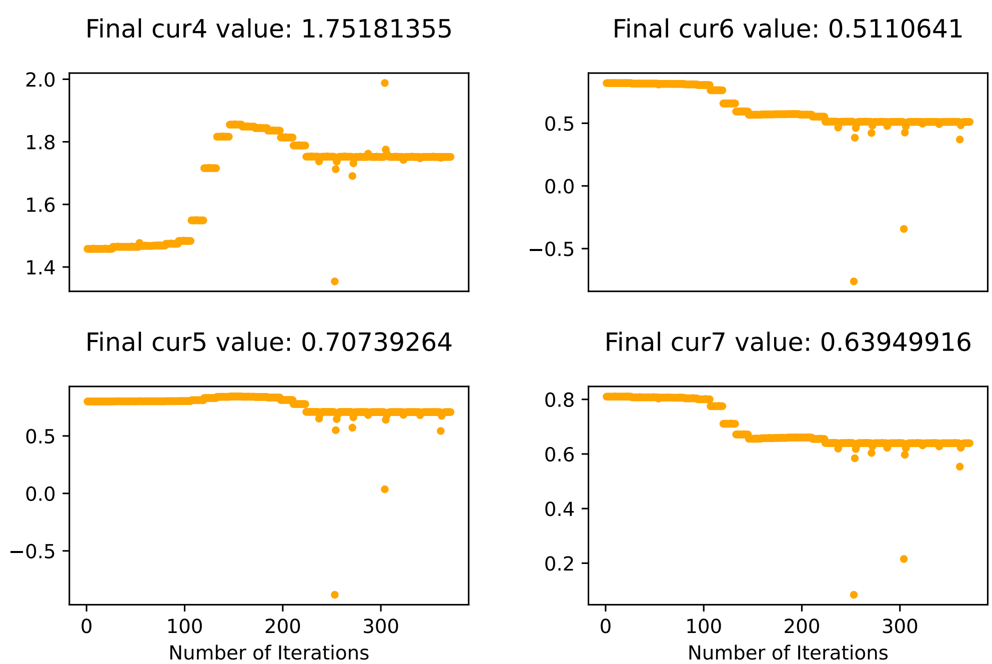


Figure 4.14: Reverse engineered Hub section - plots of variables (contd.)

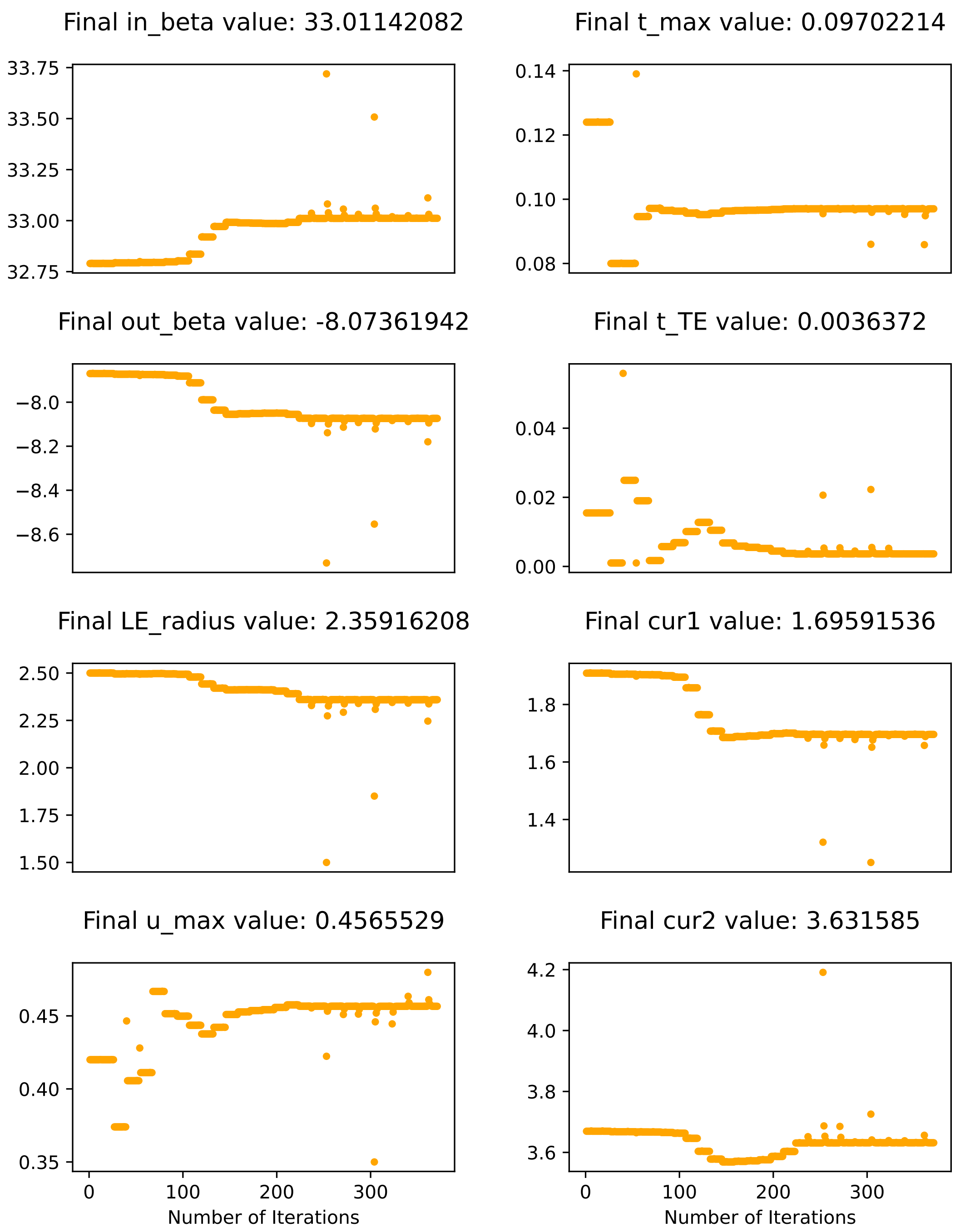


Figure 4.15: Reverse engineered Hub section(contd.) - plots of variables

E3 part-span shroud Section

The stopping criterion was set to $1e^{-03}$ and for the plot digitized part-span shroud it took around 450 gradient evaluations of the objective function to reach a least squared difference value of $9.7875e-04$ as shown in Figure 4.16. We can clearly see the difference in closeness when compared to Figure 4.9b. Looking at the individual sections in Figure 4.17, the squared difference between the points is spread out over the span of the airfoil and is not concentrated in any particular section. The optimization progression is shown in Figure 4.18, a stepped descent until 200th iteration is observed with a steady descent until 350th iteration followed by a final step descent before the optimization stops near 450th iteration. Figure 4.19 contains the initial and final values of each of the twelve parameters from the optimization, the location of maximum thickness is at 0.61471969 with the maximum thickness value being 0.03565671. The plotted graph values of each variable are in Figure 4.21 and Figure 4.20.

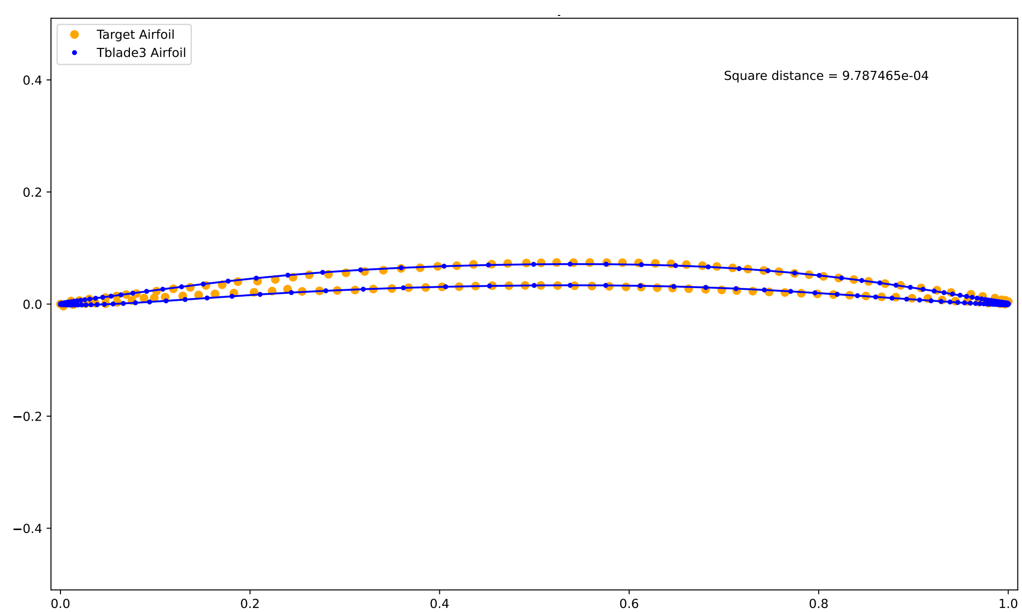


Figure 4.16: Least square difference between plot digitized part-span shroud and reverse engineered airfoil

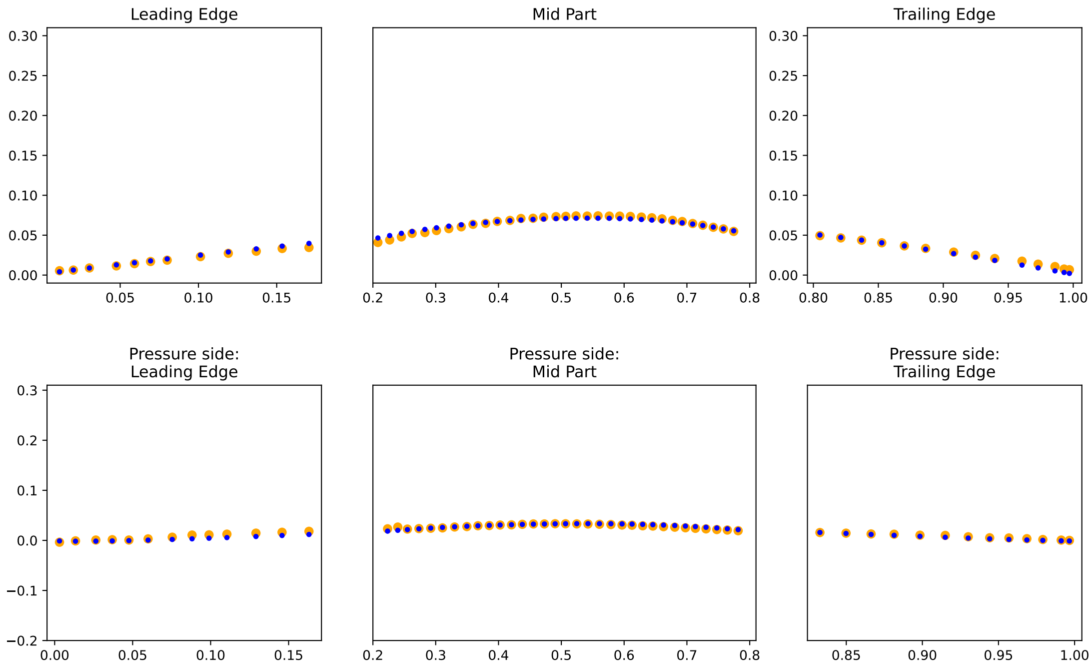


Figure 4.17: Airfoil parts comparison for E3 part-span shroud airfoil

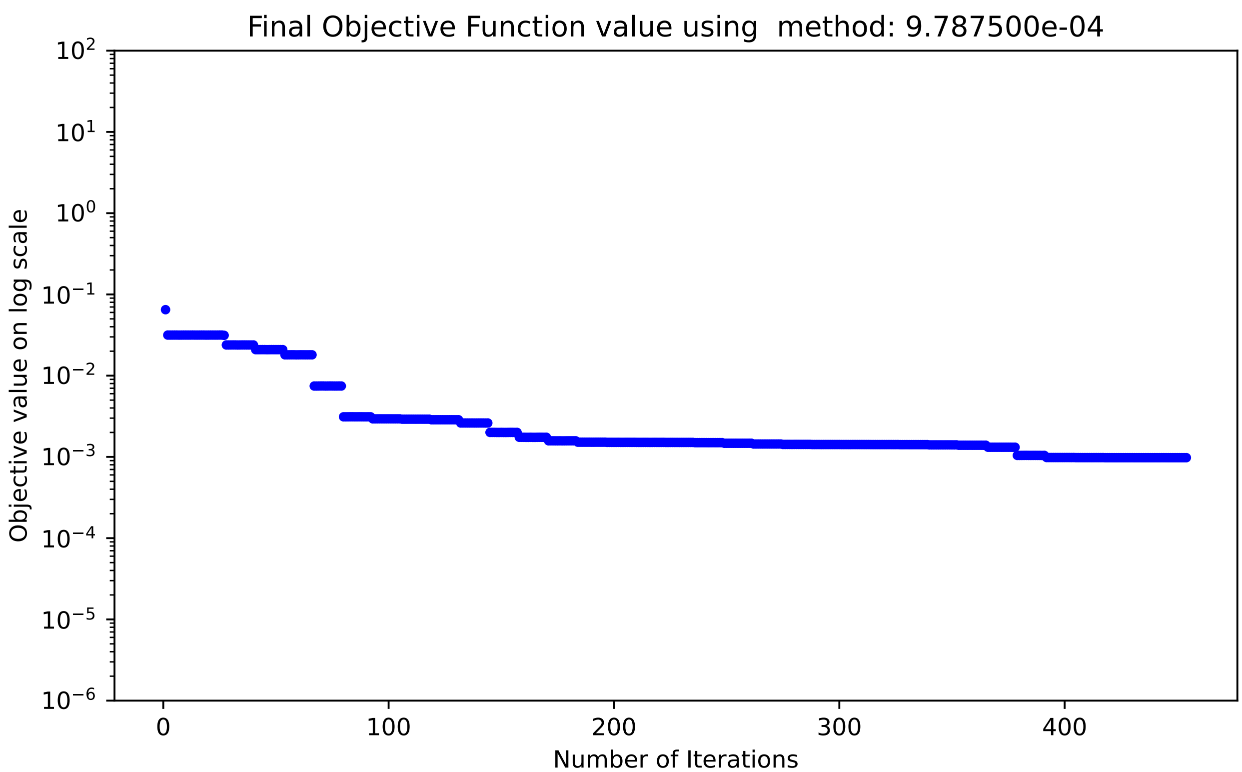


Figure 4.18: Objective function vs iterations for E3 part-span shroud

	Initial input values	Final input values
in_beta	12.01	12.6239569
out_beta	-2.81	-3.41018501
cur1	0.1	-0.95219209
cur2	0.1	0.30333961
cur4	0.1	0.02079878
cur5	0.3	0.84501973
cur6	0.4	-0.01305618
cur7	0.1	-0.66589321
LE_radius	2.5	2.22871712
u_max	0.58	0.61471969
t_max	0.044	0.03565671
t_TE	0.1	0.0001

Figure 4.19: E3 part-span shroud - Initial and Final parameter values

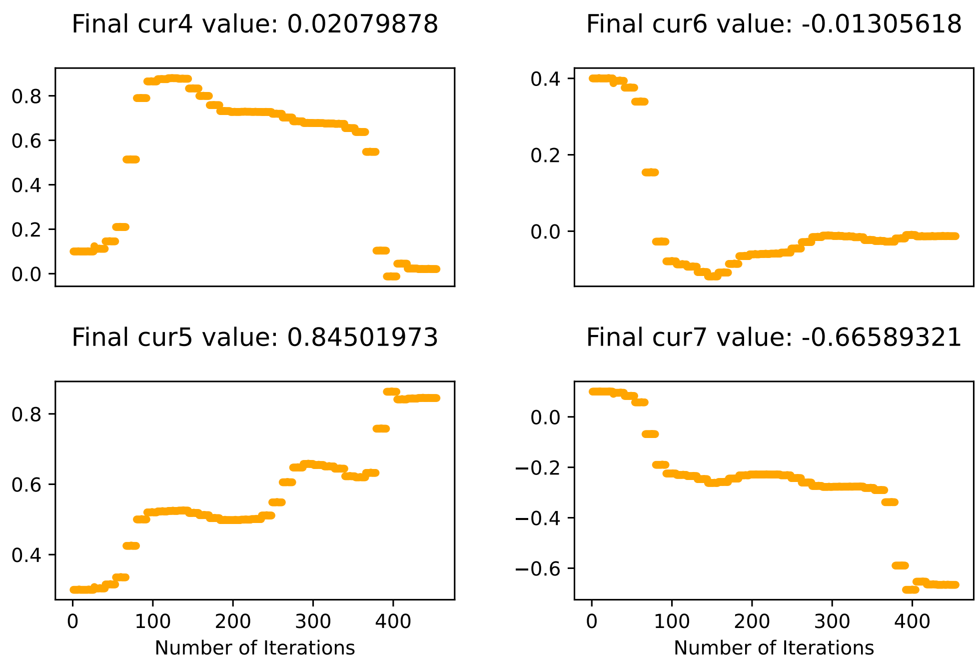


Figure 4.20: E3 part-span shroud - plots of variables (contd.)

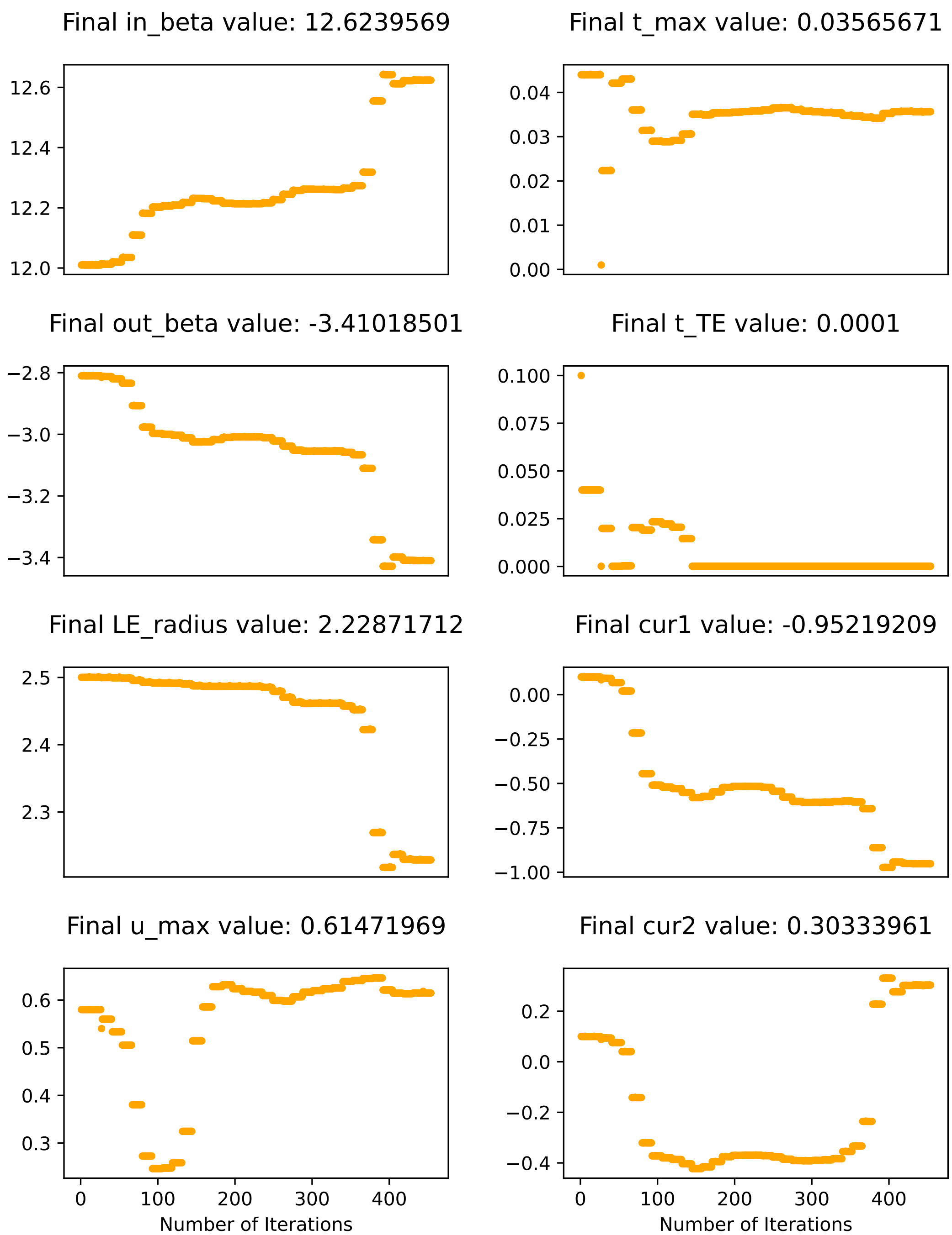


Figure 4.21: E3 part-span shroud - plots of variables

E3 tip Section

The stopping criterion was set to $1e-03$ and for the plot digitized tip it took around 350 gradient evaluations of the objective function to reach a least squared difference value of $5.125e-04$ as shown in Figure 4.22. We can clearly see the difference in closeness when compared to Figure 4.9c. Looking at the individual sections in Figure 4.23, the squared difference between the points is focused on the suction side mid part and the suction side trailing edge section. The optimization progression is shown in Figure 4.24, a quick stepped descent until 300th iteration with some big steps in the middle along with some areas of stagnant slopes before it stabilizes after 350th iteration. Figure 4.25 contains the initial and final values of each of the twelve parameters from the optimization, the location of maximum thickness is at 0.63168476 with the maximum thickness value being 0.02622002. The plotted graph values of each variable are in Figure 4.27 and Figure 4.26.

Shown in Figure 4.28 is the E3 blade obtained from the reverse engineering framework, ready for CFD simulation. The input files for Tblade3 for this particular case can be found in Appendix E.2.

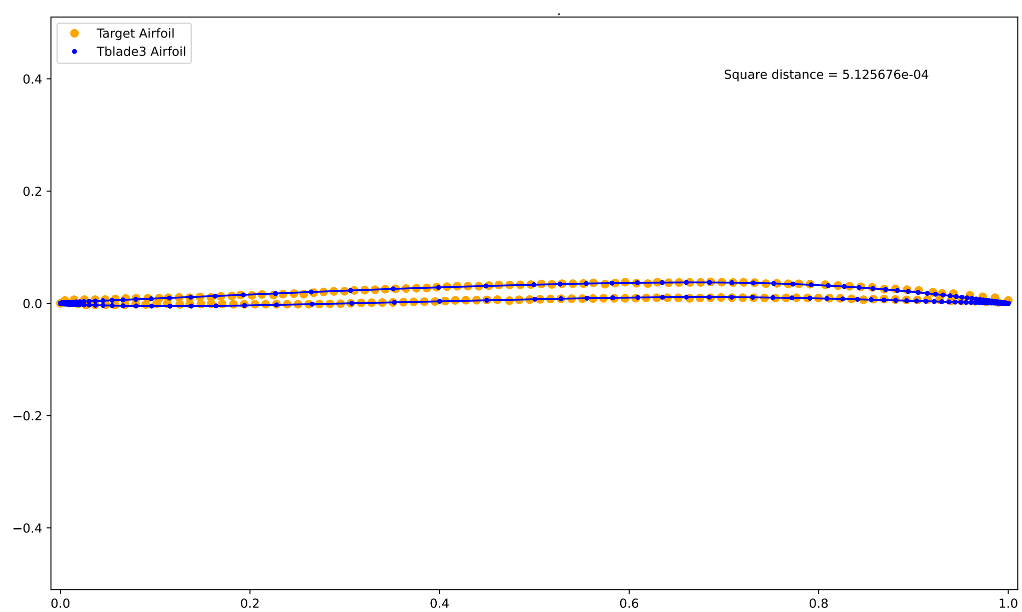


Figure 4.22: Least square difference between plot digitized tip and reverse engineered airfoil

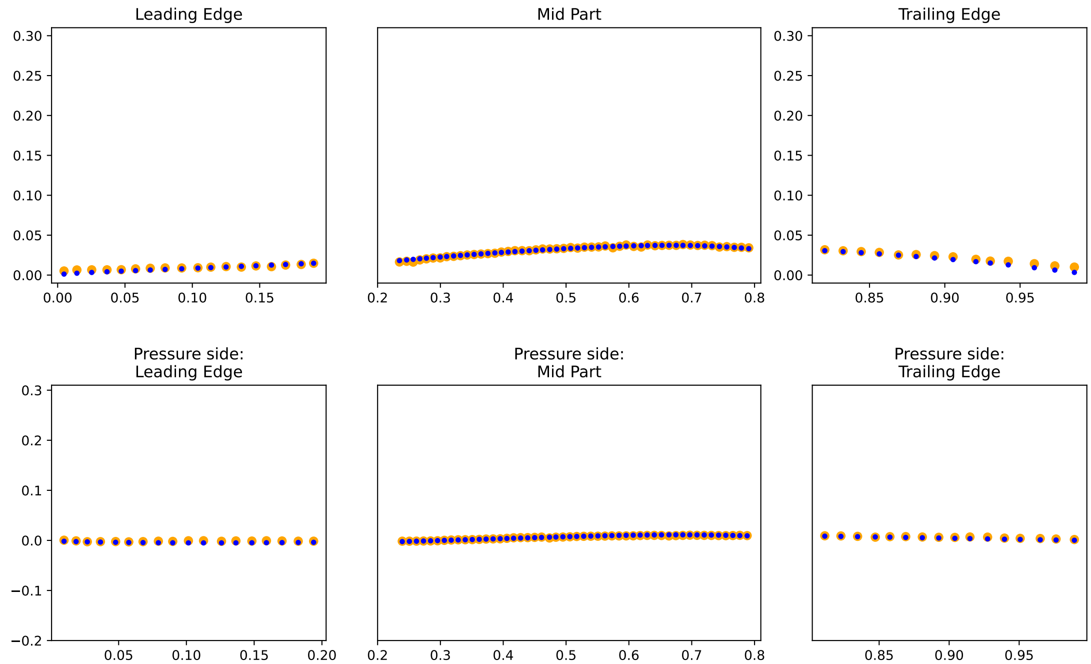


Figure 4.23: Airfoil parts comparison for E3 tip airfoil

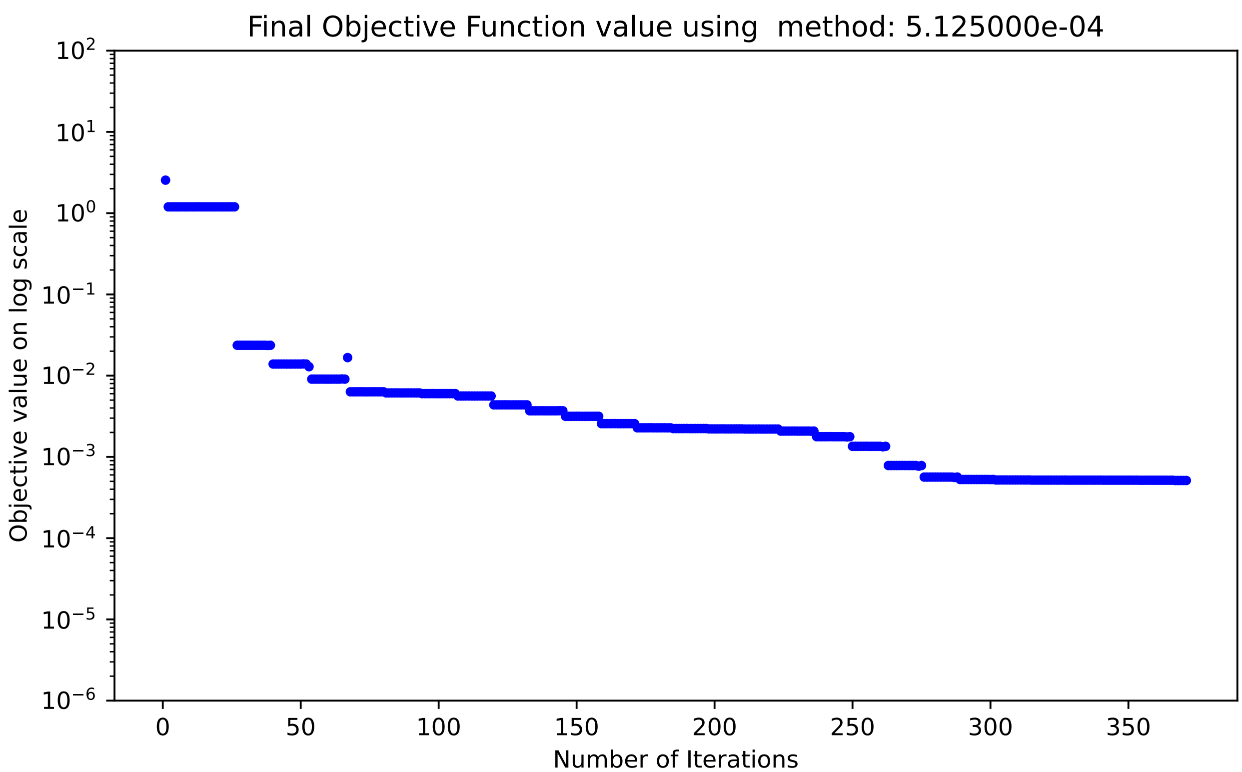


Figure 4.24: Objective function vs iterations for E3 tip

	Initial input values	Final input values
in_beta	6.52	6.60407821
out_beta	-1.42	-1.50501033
cur1	0.1	-0.43388997
cur2	0.1	-1.10842219
cur4	0.1	-0.14973654
cur5	0.3	1.81450868
cur6	0.4	1.14302249
cur7	0.1	0.10694238
LE_radius	2.5	2.4577201
u_max	0.65	0.63168476
t_max	0.24	0.02622002
t_TE	0.1	0.0001

Figure 4.25: E3 tip - Initial and Final parameter values

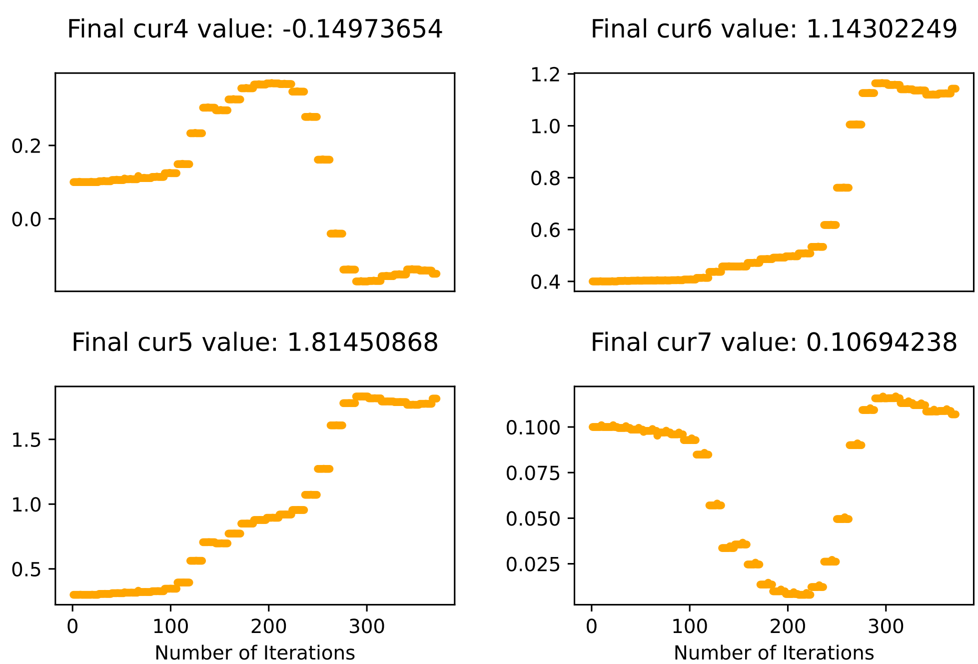


Figure 4.26: E3 tip - plots of variables (contd.)

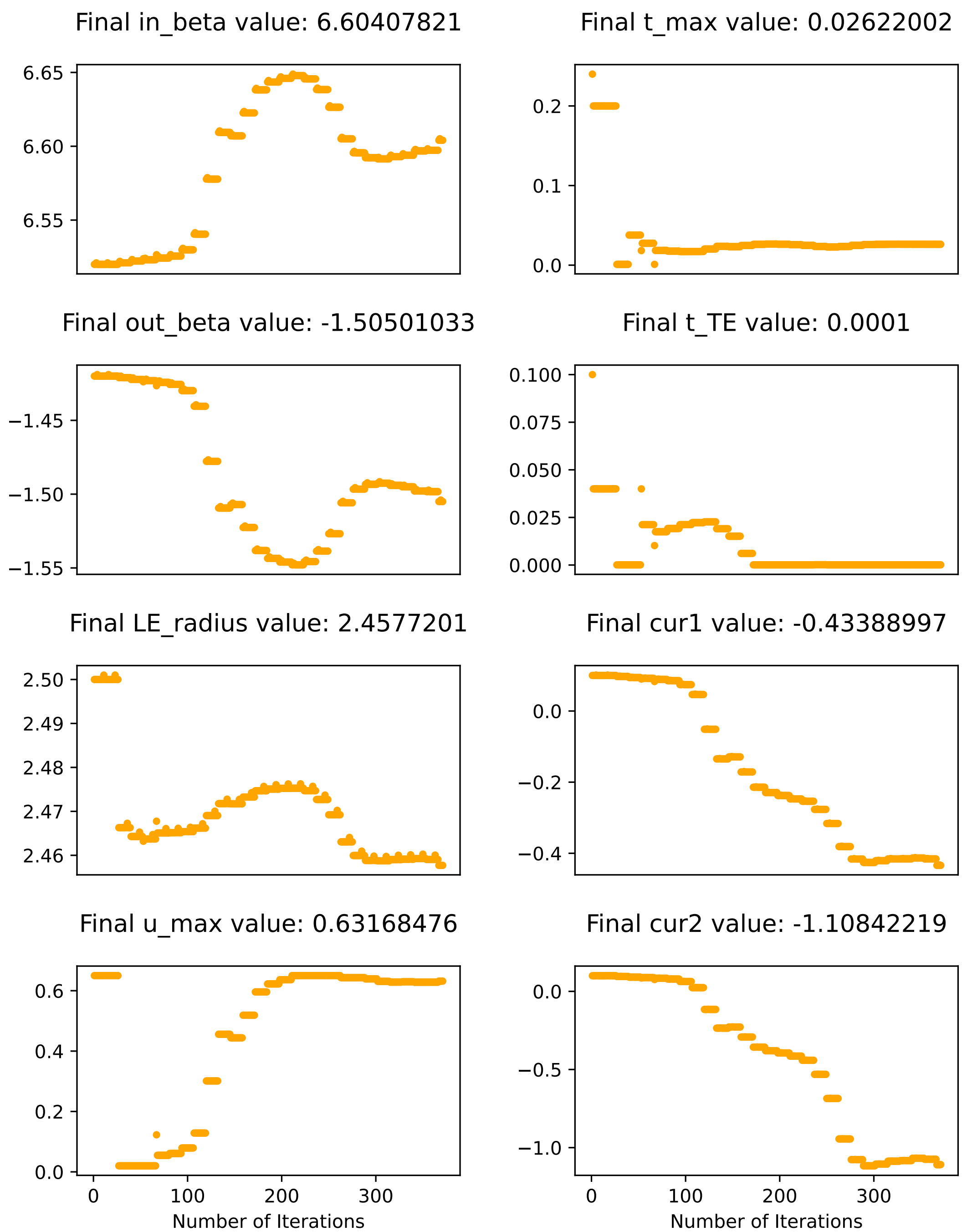


Figure 4.27: E3 tip - plots of variables

With the blade parameters found, both using traditional manual parameter specification or with the reverse engineering framework, the Tblade3 *xyz* coordinates were then converted into .geomturbo format using the code from TBlade3's website. We can now move to CFD simulation.

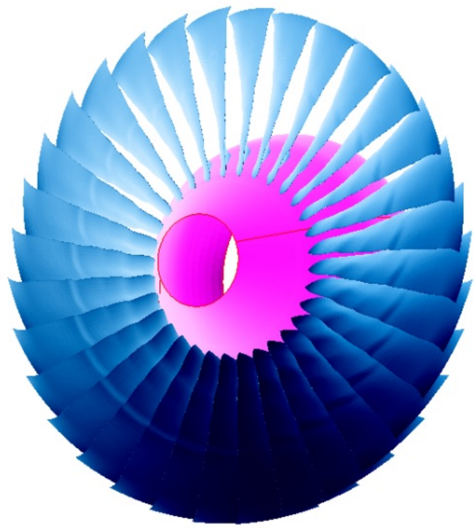


Figure 4.28: Full Annulus view of reverse engineered blade

Chapter 5

CFD simulation of E3 fan Blade

5.1 Simulation Setup

The .geomturbo file generated in the previous chapter is the input file for the FINE/Turbo environment which helps initialize the grid file to carry out further simulations. The speed case for the simulations was chosen to be one for the maximum climb with a tip speed of $411.5m/s$ and a target bypass pressure ratio of 1.65. The corrected airflow for this case with part span shroud was given to be $643.6kg/s$, ideally the obtained airflow for our simulations should be a little higher than this number for the pressure ratio.

5.1.1 Grid Generation

The .geomturbo file was imported into Autogrid5, to setup the gridding inputs for grid generation. The outlet was shrunk to a $z = 0.45$ to reduce the effects of nozzle expansion. The number of blades and calculated blade speed was input by using the LE radius as a parameter. On the next menu, tip gap was input to be 0.015% of LE and TE radius. Layer control was added with 161 spanwise grid point numbers and a wall cell width of $5.6249e-06$, this number was obtained from Autogrid5's wall cell width formula for a y+ of 50. The number of points in the cell was set to around 20 million which gave around 60 Million cells for the entire grid. This was deemed sufficient to carry out the simulations in three grid levels: 222, 111, and 000 - the resolution of the cells increases from coarse to fine as grid number decreases. With the number of cells falling rapidly at each grid level to carry out speedline calculations and study grid dependence. The maximum expansion ratio was 2.3467.

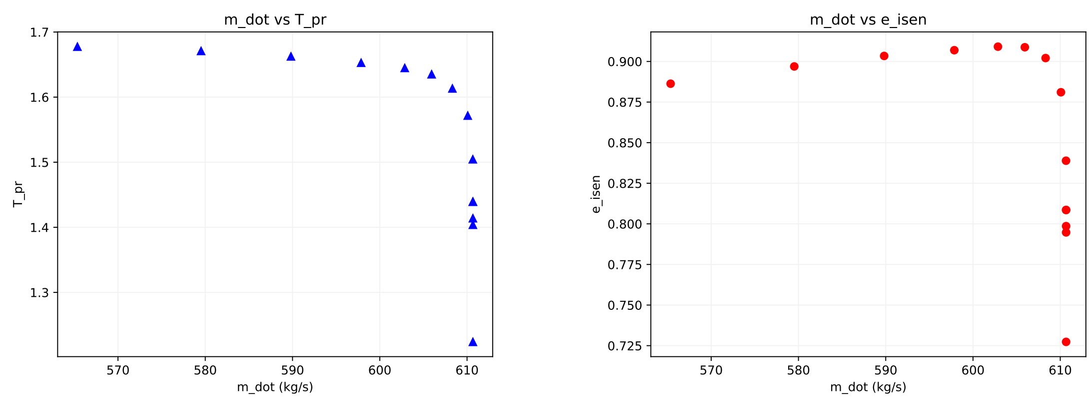
5.1.2 Solver Inputs

Once we had the grid read, it was imported into FINE/Turbo's solver to setup the inputs for the CFD simulation. The fluid model selected was perfect air with a Spalart-Allmaras Turbulent Navier stokes flow model. To calculate the Reynolds number, the solver takes in a few more parameters: Characteristic length = $0.15m$, this was the approximate chord length of the blade. Characteristic velocity = $411.5m/s$, tip velocity for the maximum climb case. Characteristic density = $1.225kg/m^3$. These input numbers were used to input Reynolds number for the simulation. Reference values of Temperature and pressure were $288.15K$ *and* $101300Pa$ respectively. Moving on to boundary conditions, the inlet conditions were "Total Quantities Imposed" with a tabular data of $tan(V_r/V_z)$ values varying with radius of the inlet given in to compensate for the hub curvature and minimize entropy generation with the inlet fluid crashing into the conical hub surface. The Absolute Total Pressure and Temperature values at inlet were $101300Pa$ and $288.15K$ respectively. The outlet boundary condition was a pressure imposed boundary condition with Radial equilibrium enabled. This allows the solver to calculate the out boundary conditions using the input static pressure value at the hub to the tip. We input this pressure at various back pressure cases at the hub with a tip radius of $0.103m$. The sides of the grid were both assigned periodic boundary conditions and the hub and blade were given a rotation speed of 3728.21 rpm while the shroud was kept stationary.

Now that we had the basic parameters set, we move to tweak the numerical model parameters. The CFL number for the simulations was left at a default 3 and the grid level was altered based on the required level of 222, 111, or 000. Then based on the back pressure value, we initialized the file using "for turbomachinary" or previously run case file. Once we had the initial solution and numerical model setup, we move on to define the required outputs: entropy, Total and static Pressure, Temperature; Absolute and relative Mach number and various angles that were relevant to the simulation. Some quantities like $beta_z$ were not readily available and calculated accordingly in the post processor. Finally, we specify the convergence criteria, number of iterations and select the solver precision as required. Then started the simulations.

5.2 Traditional manual parameter specification simulation

Based on the above boundary conditions we did a speedline for the E3 geometry using FINE/Turbo. Once we had a mesh, we ran a case with coarse grid of level 222 and an assumed design back pressure of 95kPa. From the results of this case, we created a speedline to find a more suitable design case using the assumed design case as an initial solution. This was found at around 110250 Pa of back pressure with an efficiency of 0.9091 as shown in Figure 5.1. From E^3 report the required mass flow with a part-span shroud for this particular tip speed of 411.5m/s is 643.6kg/s and a pressure ratio of 1.65. This design case however, the obtained mass flow was 602.86kg/s with a pressure ratio of 1.6448. We need to tweak the airfoil sections further to obtain the required mass flow value of a few percentage points above 643.6kg/s so as to compensate the lack of part-span shroud.



(a) Mass flow v/s efficiency (b) Mass flow v/s pressure ratio

Figure 5.1: Speedline for traditional manual parameter specification of E3 geometry - required design mass flow is $643.6kg/s$

The next step was to study grid dependence. Table 5.1 shows the values of mass flow, pressure ratio, and efficiency at various grid levels: 222, 111, and 000. These grid levels represent the coarseness of the grid with higher numbers being coarser. From looking at the tables we can clearly see that the values are grid dependent. So, coarse grid of 222 is only used to obtain the characteristic curve and then the obtained back pressure value is used with lower-level grids to obtain the accurate qualities at those back pressures.

Table 5.1: Grid Dependence at 110250 Pa back pressure for traditional manual parameter specification geometry simulation

Grid level	**Mass flow** Kg/s	**Pressure ratio** T_pr	**efficiency** η
222	602.86	1.6448	0.9091
111	585.76	1.6356	0.8898
000	591.73	1.6413	0.8976

5.2.1 Line plots of Design case

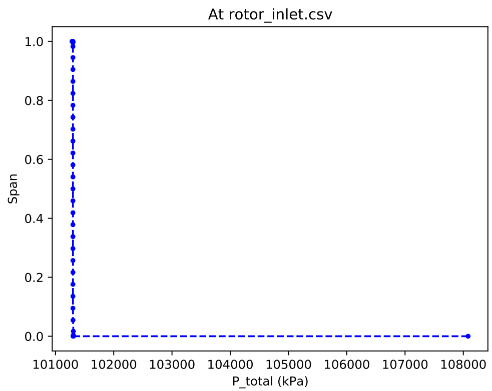
A more clear idea of the inflow and outflow parameters can be obtained from the profile plots at the inlet and outlet of the fan blade in the traditional manual parameter specification case simulation. Figure 5.2 shows the profile plots at inlet and Figure 5.4 shows the profile plots at outlet.

Profile plots at Inlet

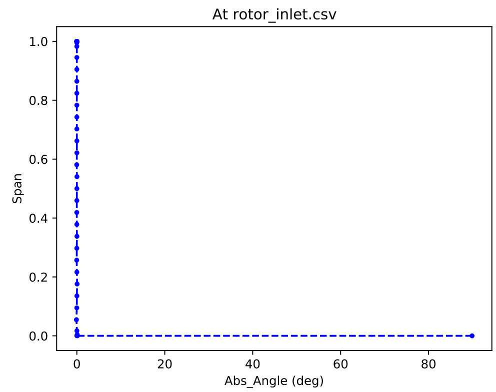
The total pressure at the rotor inlet is at a steady 101.3 kPa from 0% span at the hub to 100% span at the tip. In accordance with the input profile in FINE/Turbo, the Phi angle linearly varies from 30 to zero degrees when going from the hub to tip. This makes sure that the flow remains relatively parallel to the slope of the hub and tip.

Another important observation is in the form of the three independent directions of velocities - tangential, axial, and radial. Starting with the tangential direction, the flow from the rotating hub shows a straight impact on the flow in this direction. It is maximum on the attached flow near the hub, dropping instantaneously to zero from the next streamline and staying there for the remainder of the flow. For velocity in the radial direction, we observe that even though the flow at the hub starts at zero, it increases to a maximum near the hub showing that the flow is attached to a rotating hub. Unlike tangential velocity, radial velocity linearly falls to zero near the tip. Axial velocity vs span plot shows the characteristic "D" shape that is formed in piped flows. The velocity in this direction is zero near the hub and tip while increasing to a maximum in the majority of the flow.

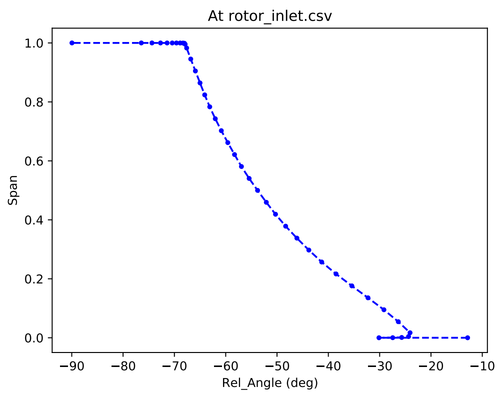
There are more plots from relative angles to Mach numbers for further diagnosis and provide an in-depth insight into where losses are being generated for the designer to create changes in the geometry.



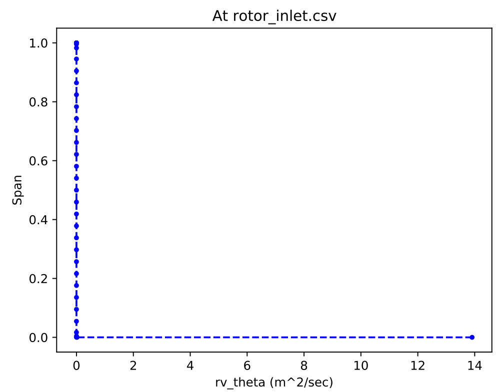
(a) Span v/s Total Pressure



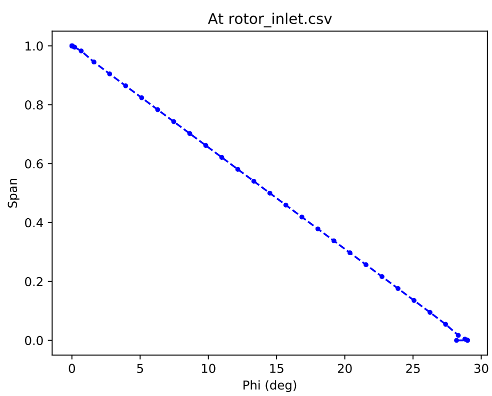
(b) Span v/s Absolute Angle



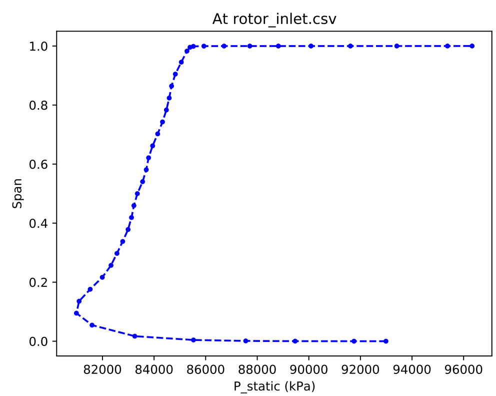
(c) Span v/s Relative Angle



(d) Span v/s $r * V_\theta$

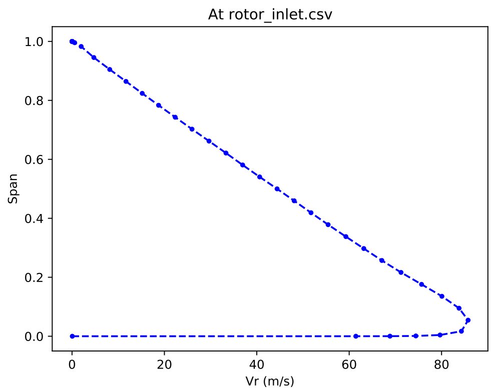


(e) Span v/s Phi

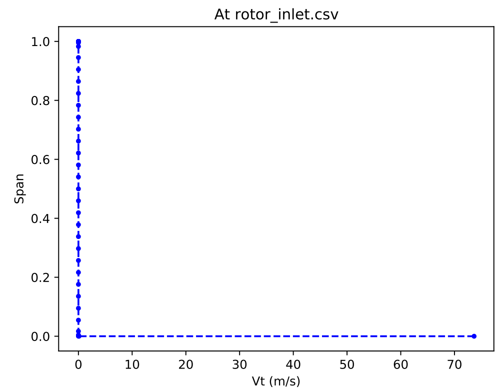


(f) Span v/s Static Pressure

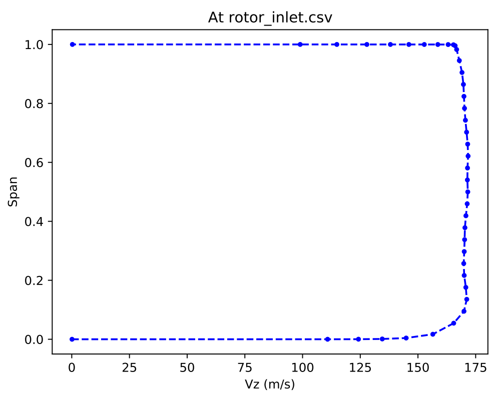
Figure 5.2: Profile plots at E^3 fan inlet for traditional manual parameter specified design



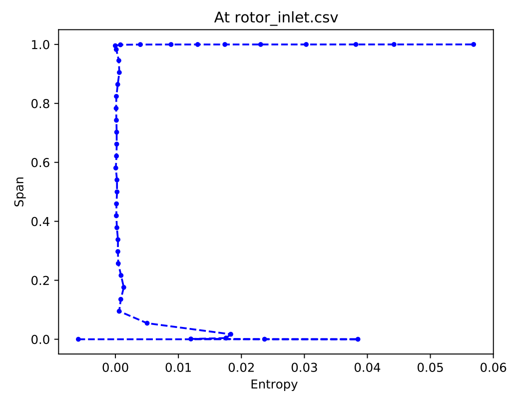
(a) Span v/s Radial Velocity



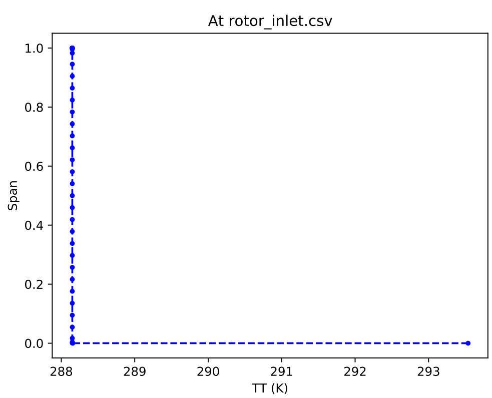
(b) Span v/s Tangential Velocity



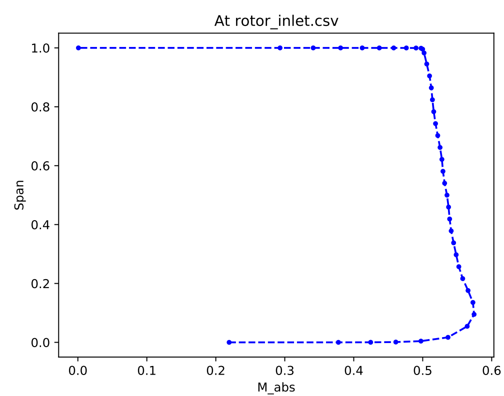
(c) Span v/s Axial Velocity



(d) Span v/s Entropy

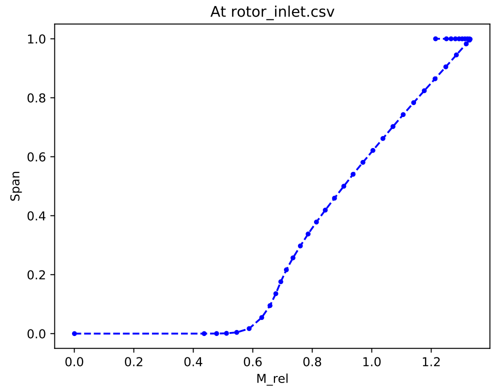


(e) Span v/s Total Temperature

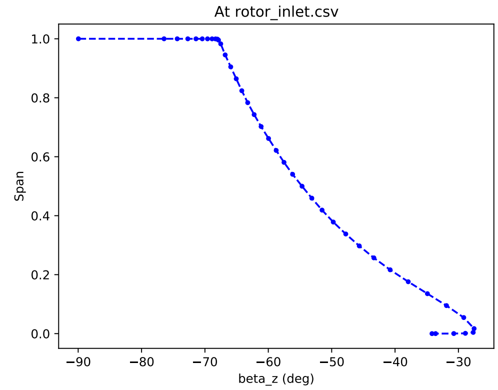


(f) Span v/s Absolute Mach Number

Figure 5.3: Profile plots at E^3 fan inlet for traditional manual parameter specified design



(g) Span v/s Relative Mach Number



(h) Span v/s β_z

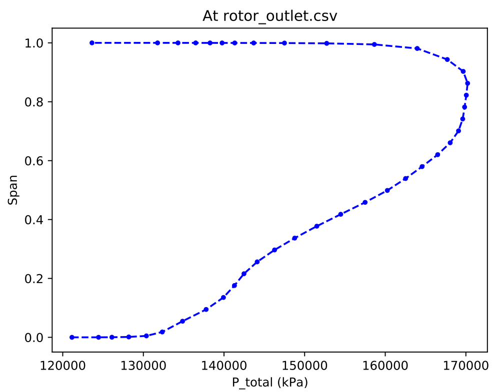
Figure 5.3: Profile plots at E^3 fan inlet for traditional manual parameter specified design

Profile plots at outlet

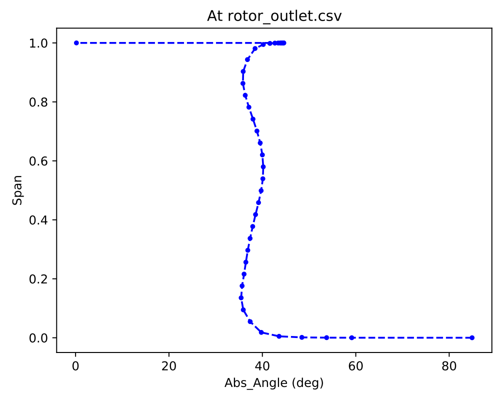
In contrast to the inlet total pressure v/s span plots, there is an addition of total pressure at the outlet. Maximum increase is seen near the upper span regions before steeply falling off near the tip. This is expected in a rotor simulation as total pressure is added to the flow. The slope of the phi angle plot is now non-linear with negative angles near the tip indicating that the flow near the tip is interfering with the other streamlines in the upper-middle spans. This will be more evident when we look at the contours of entropy in the next section.

For the velocity components, the axial component retains its characteristic shape with some added velocity near the hub and maximum velocity location. The radial velocity component too shows a similar change with a flow in the downward direction near the tip and a non-zero upward flow near the hub. In the tangential direction, however, there is a sharp contrast between the inlet flow where the air in the tangential direction is relatively stagnant to the outlet flow where there is a tangential velocity component throughout the span except for the tip.

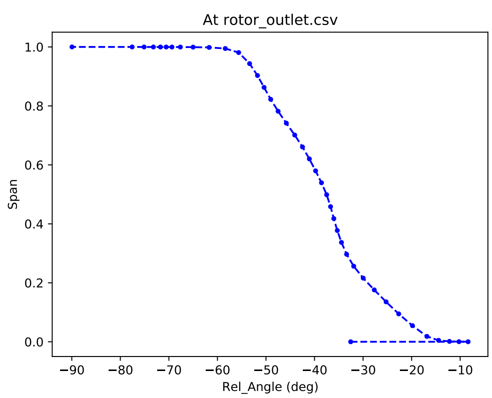
After looking at the line plots data from the traditional manual parameter specification case, we had a confirmation of the simulation input conditions to be adequately good enough for the simulations. Though more time can be spent comparing these line plots to the ones from the E3 report and those from reverse engineering framework blade simulation, emphasis is given on demonstration of these tools and comparison of the contour plots instead. To do that, we will now move to set up the reverse-engineered geometry's simulation next.



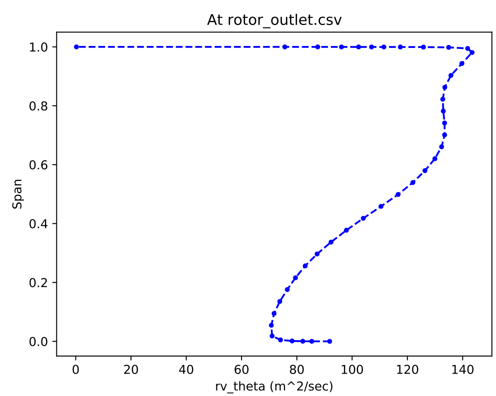
(a) Span v/s Total Pressure



(b) Span v/s Absolute Angle

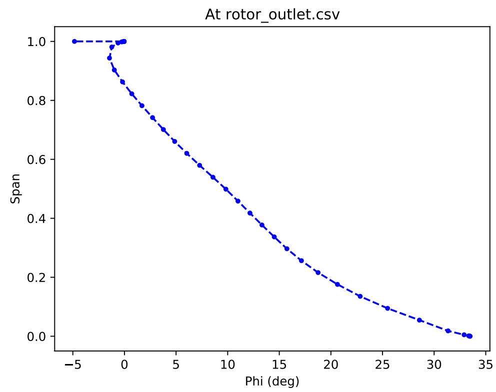


(c) Span v/s Relative Angle

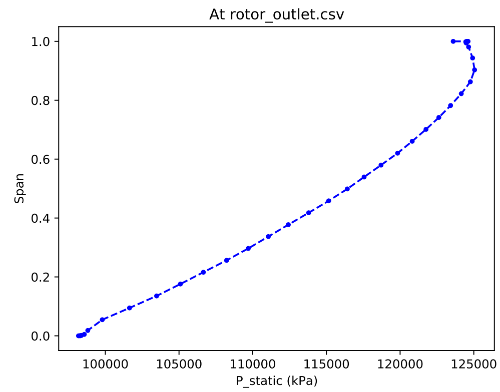


(d) Span v/s $r * V_\theta$

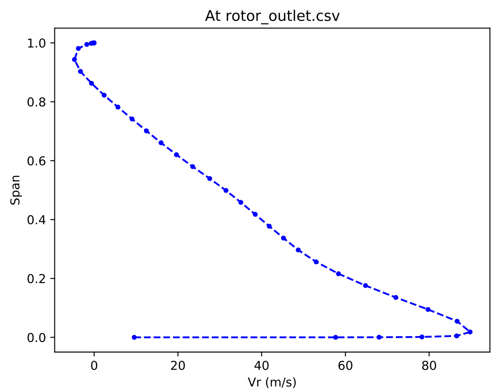
Figure 5.4: Profile plots at E^3 fan outlet for traditional manual parameter specified design



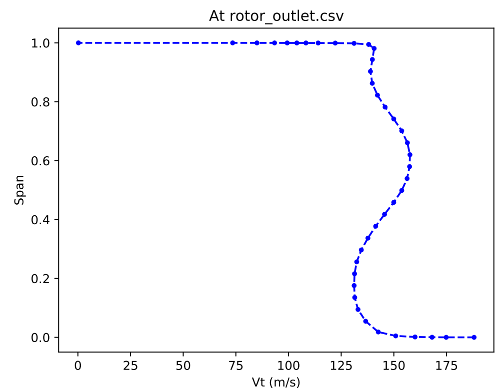
(e) Span v/s Phi



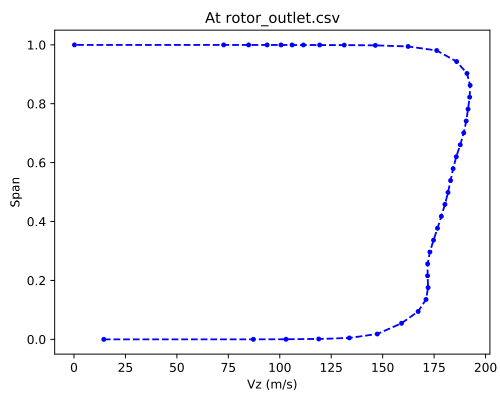
(f) Span v/s Static Pressure



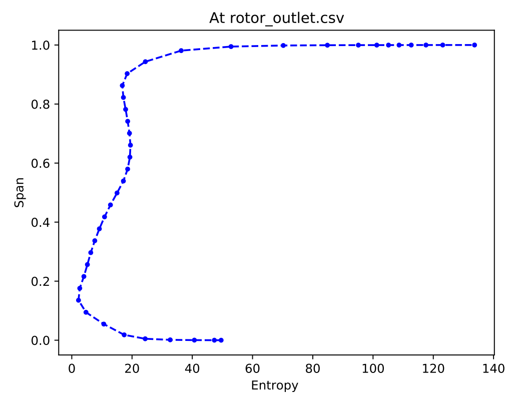
(g) Span v/s Radial Velocity



(h) Span v/s Tangential Velocity

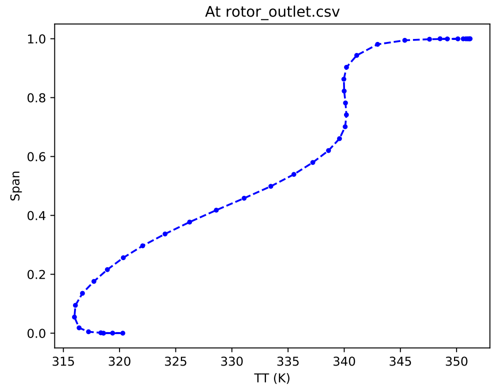


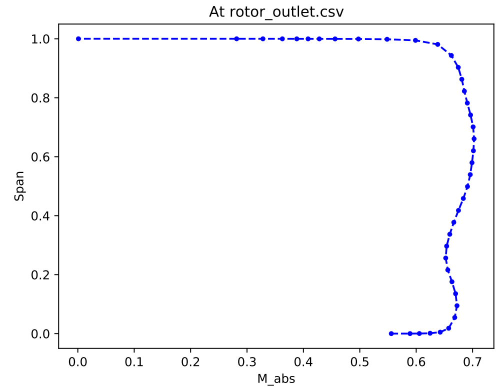
(i) Span v/s Axial Velocity



(j) Span v/s Entropy

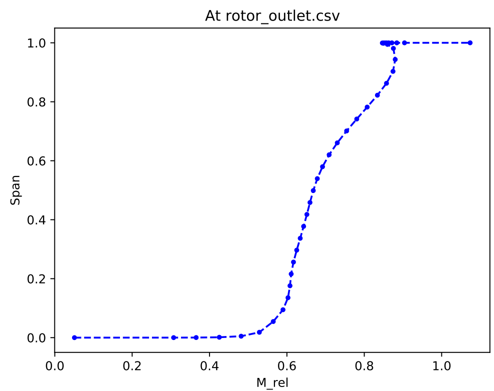
Figure 5.4: Profile plots at E^3 fan outlet for traditional manual parameter specified design

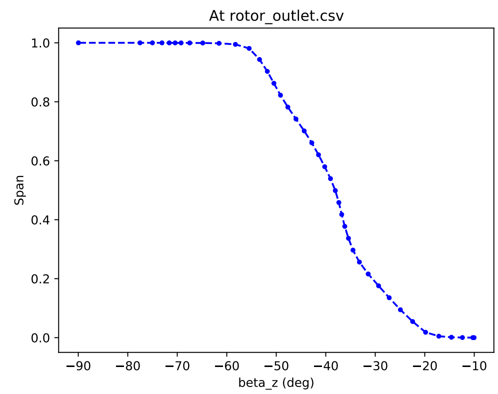




(k) Span v/s Total Temperature

(l) Span v/s Absolute Mach Number





(m) Span v/s Relative Mach Number

(n) Span v/s β_z

Figure 5.4: Profile plots at E^3 fan outlet for traditional manual parameter specified design

5.3 Reverse engineered geometry's simulation

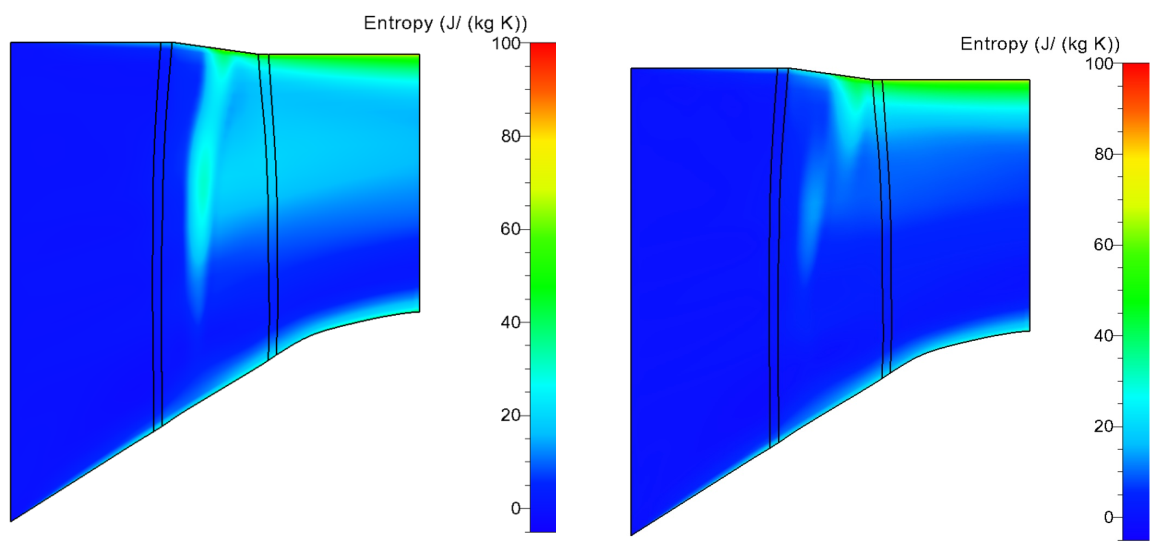
The .geomturbo input file generated from the reverse-engineered Tblade3 output file was put through the same simulation procedures with the design case back pressure. While a speedline study could have been conducted, I have left that for further scope and will focus on the demonstration of the framework. Mass flow of this simulation with a 222 (coarse) grid level was $593.38kg/s$ at a pressure ratio of 1.62. The isentropic efficiency was 0.91. There is a 1.5% reduction of mass flow from the same coarse grid level simulation with traditional manual parameter specification digitized airfoils. Both the simulations, however, are at least 7% away from the experimental mass flow of the compressor fan with a part span shroud from the report which is at $643.6kg/s$. This loss of Mass flow is attributed to Radial equilibrium boundary condition at outlet of Fan. With the quarter-stage booster aft of the fan, the back pressure will be different than radial equilibrium.

The pressure ratio could be improved from the current 1.62 to 1.65 in order to properly compare the simulations with a speedline study but is left for further work due to time constraints.

5.4 Comparison of results at design case

5.4.1 Contour plots of Entropy

An axisymmetric mass average contour plot takes a certain quantity, in this case entropy, and provides an average value for the magnitude in the radial and tangential directions. It is shown in Figure 5.5. Since the hub is conical, we needed to check if the inlet angles for the boundary conditions were causing any unnecessary separation near the hub. Contour plots are a great tool to achieve this because they map a third dimension in the form of a color-map onto a 2D Figure. In both the simulations, there is a similar amount of entropy being generated along the hub surface but as we move away from the span, we can see a drastic drop in entropy generation from the mid-span area. This sheds more light on why reverse engineering method is beneficial. To understand these differences better we need to look at blade-to-blade section plots along the spans. For simplicity, we took at these sections along three spans: 0% - hub, 55% part-span shroud and finally 99.5% - tip.



(a) Traditional manual parameter specified Geometry

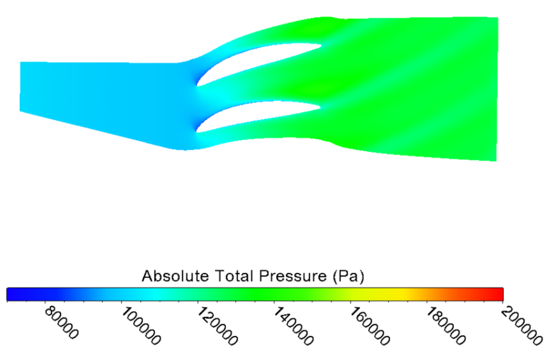
(b) Reverse engineered Geometry

Figure 5.5: Axisymmetric averages of Entropy

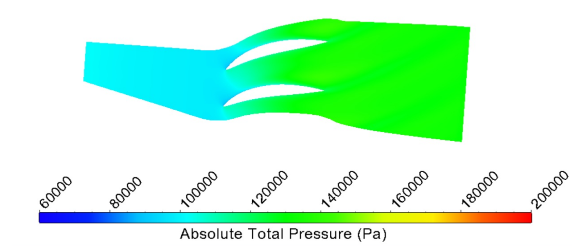
Unlike the axisymmetric mass average contour plot, the blade-to-blade contour plots do not take an average quantity but plot the exact mesh values of the quantity at a constant radius from the origin. There is a provision in FINE/Turbo to specify how many blade channels are visible at a given time we have chosen two to show the throat area.

5.4.2 Contour plots at Hub

Near the hub, we start by looking at total pressure values across the blade span. There is no abrupt pressure change in the contour plots across this span Figure 5.6, it is also the case in total temperature plots - Figure 5.6. For relative Mach number plots, we can see that the reverse engineering case is not reaching a greenish color near below the suction side indicating that a Speedline simulation run might help increase mass flow unlike the case with traditional manual parameter specification designed blade. In both cases, there is a significant reduction in massflow near the Leading Edge and Trailing Edge, and the design could be further tweaked to minimize these losses.

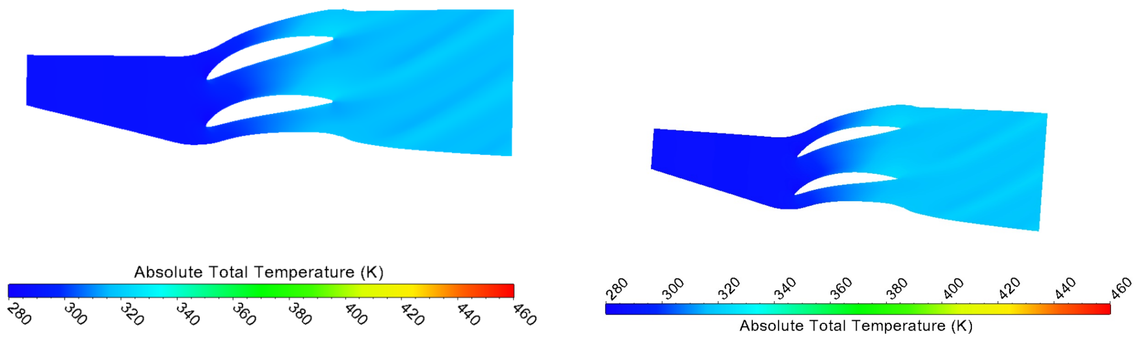


(a) Traditional manual parameter specified Geometry



(b) Reverse engineered Geometry

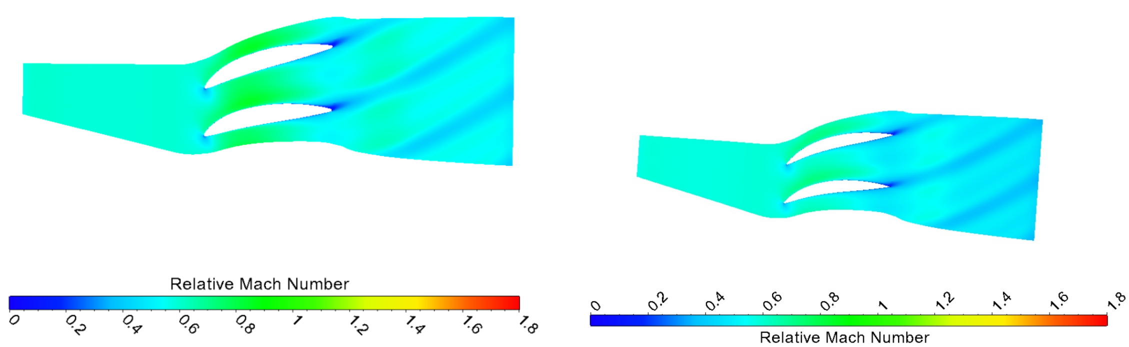
Figure 5.6: Contour plots of Total Pressure near hub



(a) Traditional manual parameter specified Geometry

(b) Reverse engineered Geometry

Figure 5.7: Contour plots of Total Temperature near hub



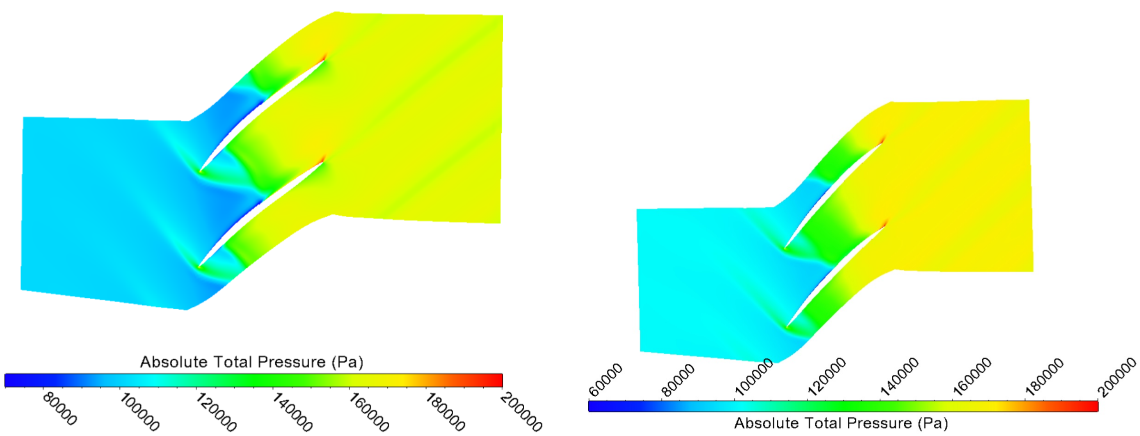
(a) Traditional manual parameter specified Geometry

(b) Reverse engineered Geometry

Figure 5.8: Contour plots of Relative Mach Number near hub

5.4.3 Contour plots at Part-span shroud

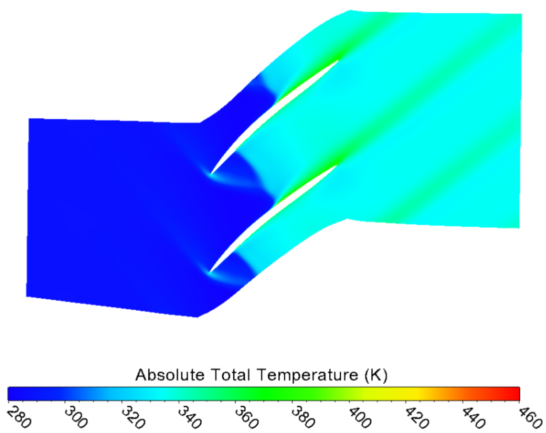
Figure 5.11 shows the relative Mach number contour plots for part-span-shroud section located at 55% blade span. Quickly looking at part-span shroud from the E3 report - Figure 4.6, we note that the location of the throat is after the 50% chord. While there is a normal shock for both the airfoils that originate near the 55% blade chord, as seen from the sudden rise in total temperature plots. The normal shock in the case of reverse engineered blade is not completely developed on the throat and is impinged by the oblique shock near the suction side of the airfoil. From the relative Mach number plots, it is also evident that the flow stalls after the shock in the traditional manual parameter specified design and is not as much stall when it comes to the reverse engineered blade.



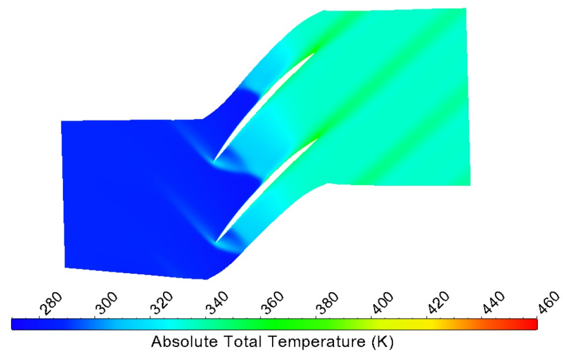
(a) Traditional manual parameter specified Geometry

(b) Reverse engineered Geometry

Figure 5.9: Contour plots of Total Pressure near part-span shroud

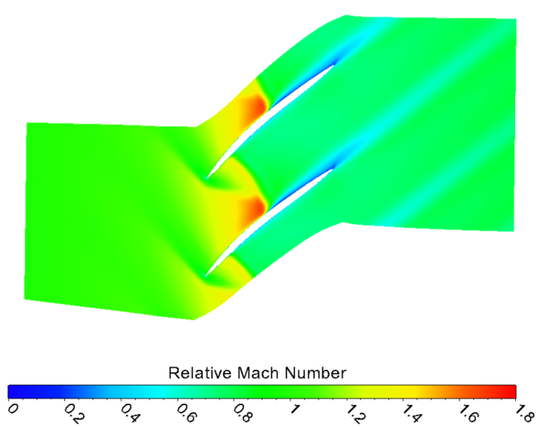


(a) Traditional manual parameter specified Geometry

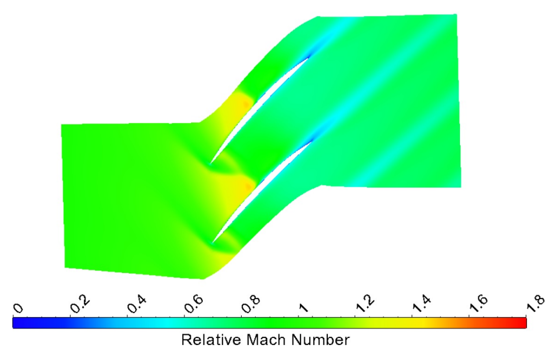


(b) Reverse engineered Geometry

Figure 5.10: Contour plots of Total Temperature near part-span shroud



(a) Traditional manual parameter specified Geometry

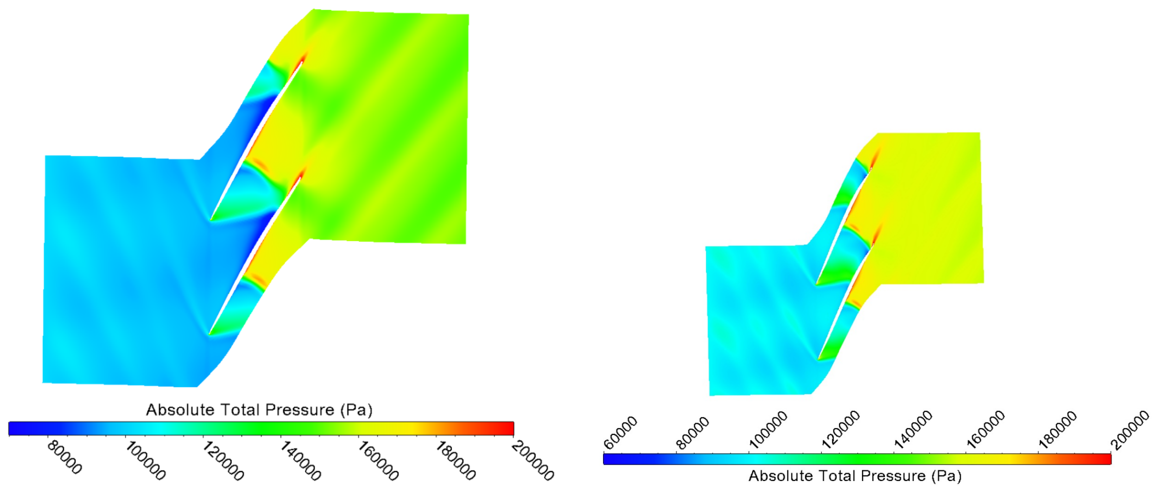


(b) Reverse engineered Geometry

Figure 5.11: Contour plots of Relative Mach Number near part-span shroud

5.4.4 Contour plots at Casing

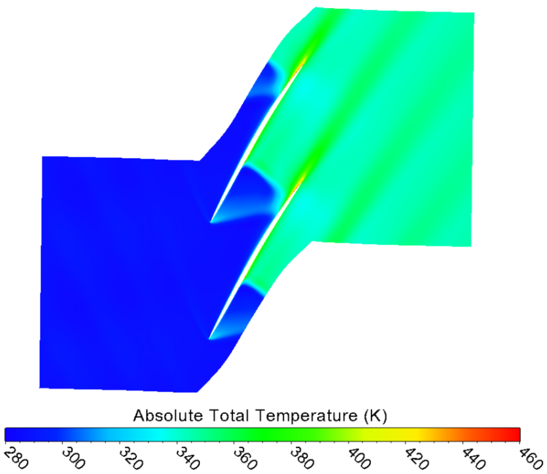
As we move higher up the span, a much stronger oblique shock is formed near the LE of the tip section blade as seen in Absolute Total Pressure plots. Figure 5.12. The throat area is shown in Figure 5.14 by 95% chord in the reverse-engineered airfoil section. This is in contrast to the normal shock location in E3 report Figure 4.7 where the throat area is near 80% chord. The traditional manual parameter specified geometry better matches this shock location but suffers from more stall after the shock.



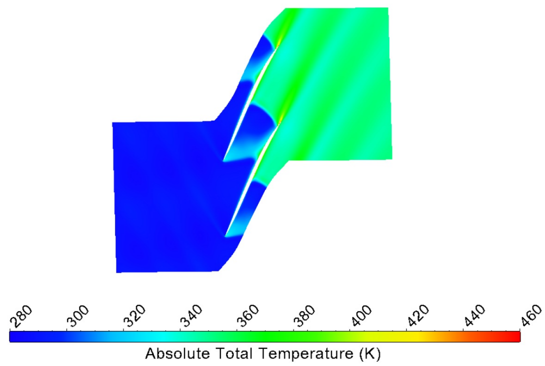
(a) Traditional manual parameter specified Geometry

(b) Reverse engineered Geometry

Figure 5.12: Contour plots of Total Pressure near tip

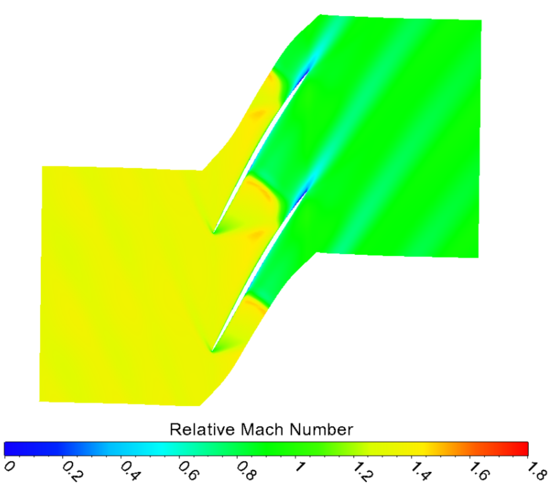


(a) Traditional manual parameter specified Geometry

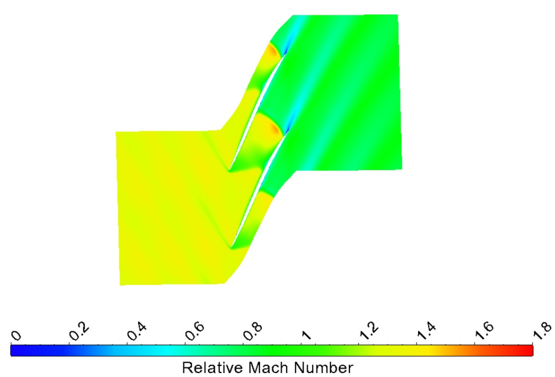


(b) Reverse engineered Geometry

Figure 5.13: Contour plots of Total Temperature near tip



(a) Traditional manual parameter specified Geometry



(b) Reverse engineered Geometry

Figure 5.14: Contour plots of Relative Mach Number near tip

5.5 Suggestions to improve mass flow and total pressure ratio comparison

A comparison of total pressure ratio and mass flow values from three design sources- GE E3 report, Traditional manual parameter specified Geometry simulation, and Reverse engineered Geometry simulation for maximum climb are summarized in Table 5.2. The target mass flow from the E3 report is for a case with the part-span shroud.

5.5.1 Improving total pressure ratio and mass flow

By comparing the shock locations from relative Mach number contours in the design case of the traditional manual parameter specification geometry (Figure 5.14a) and reverse engineered blade geometry (Figure 5.14b), one can infer that the back pressure for the reverse engineering case is near choke. The required total pressure ratio of 1.65 can be obtained by changing back pressure to get a better total pressure ratio although that would further reduce the mass flow rate. Discrepancies in mass flow are speculated to be due to:

- Not having the true blades,
- Blade stagger angles are not correct,
- Number of blade span definitions is three and can be as increased to many more,
- Not identical hub and casing geometry, and
- Lack of inclusion of quarter stage booster in the simulations and the island splitter which lead to an incorrect static pressure profile.

The above suggestions can help increase mass flow to a given extent. However, the target mass flow from the E3 report is for a case with the part-span shroud. When simulating without a part-span shroud, the mass flow value should be higher.. With the focus of this thesis being on the demonstration of reverse engineering framework, this aspect is left for future work.

Table 5.2: Comparison of mass flow and pressure ratio across all three sources for maximum climb

Design source	**Mass flow** Kg/s	**Pressure Ratio** T_pr
GE E3 report - with part span shroud	643.6	1.65
Traditional manual parameter specified Geometry - without part span shroud	602.86	1.6448
Reverse Engineered Geometry - without part span shroud	593.38	1.6176

Chapter 6

Outcomes

1. This work was presented at AIAA SciTech 2023 conference held in Washington D.C.[36].
2. The author also went through UC's Venture Lab program to study the commercialization potential of the code and to further understand the relevance of this work in the industry. customer discovery interviews were carried out with blade manufacturers in ASME TurboExpo 2023 held in June.
3. The resulting venture "DigiE3Turbo" has been accepted into Founder Institute's Chicago cohort for Spring 2024 where further commercialization exploration is being carried out by the author.

Chapter 7

Conclusion

A framework to reverse engineer airfoil section parameters has been presented. A multivariable single objective optimization is used to reduce the difference between various parts of an airfoil blade section to obtain Tblade3 parameters. The method divides input airfoil into six parts to simplify blade difference calculation. A turbomachine blade section is obtained using the new input files which contain the optimized parameters in beta, out beta, six curvature control points, LE radius, u max, t max, t TE.

A demonstration of the developed method was carried out by reverse engineering the E3 transonic fan blade from its sections. This fan blade was chosen due to its uniqueness of having a sloped hub. The airfoil sections were plot digitized from E3 report which were run through the framework to get reverse engineered parameters. A subsequent 3D simulation of the blade has been carried out to compare the reverse engineered blade with its design report. A grid dependence of coarse level with fine level was established using the rotor's off design (full speedline) simulation. Insights on further directions to obtain design corrected mass flow rate were suggested after inspecting shock locations of blade at different spans using contour plots and profile plots to improve the comparison.

This capability can be utilized as general capability to reverse engineer other blade sections.

Chapter 8

Future Work

While care was to taken think and work on all the suggested directions throughout the thesis, there are a lot of directions for future work that can be used to improve the work. The following list is an attempt to capture those directions:

1. Modify the design further to match the mass flow and pressure ratios from the report
2. Create a new design with a span shroud and understand how this will impact mass flow in contrast to a part-span shroud less design.
3. Carry out the simulations at multiple design speed conditions and a grid dependence study.
4. Increase the number of curvature points in spancontrolinputs file to obtain better curvature control.
5. Carry out design of $E3$ engine fan with a few parameters from the report, instead of a 13 variable design.
6. Rewrite the blade difference calculator to consider squared difference in u and v dimensions instead of the current v.
7. Study the effects of using piecewise cubic and higher order interpolations and compare the errors with piecewise linear interpolation.

Bibliography

[1] Liang-Chia Chen and Grier CI Lin. Reverse engineering in the design of turbine blades–a case study in applying the mamdp. *Robotics and Computer-Integrated Manufacturing*, 16(2-3):161–167, 2000.

[2] Kamran Mohaghegh, MH Sadeghi, and A Abdullah. Reverse engineering of turbine blades based on design intent. *The International Journal of Advanced Manufacturing Technology*, 32(9):1009–1020, 2007.

[3] Kiran Siddappaji, Mark G Turner, and Ali Merchant. General capability of parametric 3d blade design tool for turbomachinery. In *Turbo Expo: Power for Land, Sea, and Air*, volume 44748, pages 2331–2344. American Society of Mechanical Engineers, 2012.

[4] Kiran Siddappaji. *Parametric 3d blade geometry modeling tool for turbomachinery systems.* PhD thesis, University of Cincinnati, 2012.

[5] Kiran Siddappaji, Mark G Turner, Soumitr Dey, Kevin Park, and Ali Merchant. Optimization of a 3-stage booster: Part 2—the parametric 3d blade geometry modeling tool. In *Turbo Expo: Power for Land, Sea, and Air*, volume 54679, pages 1431–1443, 2011.

[6] Ahmed F Nemnem, Mark G Turner, Kiran Siddappaji, and Marshall Galbraith. A smooth curvature-defined meanline section option for a general turbomachinery geometry generator. In *Turbo Expo: Power for Land, Sea, and Air*, volume 45615, page V02BT39A026. American Society of Mechanical Engineers, 2014.

[7] Karthik Balasubramanian, Mark G Turner, and Kiran Siddappaji. Novel curvature-based airfoil parameterization for wind turbine application and optimization. In *Turbo Expo: Power*

for Land, Sea, and Air, volume 50961, page V009T49A020. American Society of Mechanical Engineers, 2017.

[8] Mayank Sharma, John F Dannenhoffer III, Justin Holder, and Mark G Turner. Integration of an open source blade geometry generator using a physics based parameterization with the engineering sketch pad. *Turbo Expo: Power for Land, Sea, and Air*, 84089:V02CT35A048, 2020.

[9] Robert Haimes and John Dannenhoffer. The engineering sketch pad: A solid-modeling, feature-based, web-enabled system for building parametric geometry. In *21st AIAA Computational Fluid Dynamics Conference*, page 3073, 2013.

[10] Mayank Sharma and Mark G Turner. Continuous modified naca four-digit thickness distribution for turbomachinery geometries. *AIAA Journal*, 59(4):1501–1505, 2021.

[11] Russell W Claus, Tim Beach, Mark Turner, Kiran Siddappaji, and Eric S Hendricks. *Geometry and Simulation Results for a Gas Turbine Representative of the Energy Efficient Engine (EEE)*. National Aeronautics and Space Administration, Glenn Research Center, 2015.

[12] Mark Turner. Lessons learned from the ge90 3-d full engine simulations. In *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, page 1606, 2010.

[13] T J Sullivan. Energy efficient engine: fan test hardware detailed design report. *NASA Technical Reports Server*, 10 1980.

[14] Leroy H Smith. Nasa/ge fan and compressor research accomplishments. In *ASME 1993 International Gas Turbine and Aeroengine Congress and Exposition*, pages 555–569. American Society of Mechanical Engineers Digital Collection, 1993.

[15] Leroy H Smith Jr. Axial compressor aerodesign evolution at general electric. *J. Turbomach.*, 124(3):321–330, 2002.

[16] MARK G Turner and IK Jennions. An investigation of turbulence modeling in transonic fans including a novel implementation of an implicit k–ε turbulence model. 1993.

[17] IK Jennions and MG Turner. Three-dimensional navier–stokes computations of transonic fan flow using an explicit flow solver and an implicit κ–ε solver. 1993.

[18] Edward J Hall, Sean R Lynn, Nathan J Heidegger, and Robert A Delaney. Energy efficient engine low pressure subsystem flow analysis. Technical report, NASA TRS, 1998.

[19] Numeca. Multistage axial turbine. *FINE/Turbo 14.2 Tutorials*, pages 641–697, 2020.

[20] MG Turner. Multistage turbine simulations with vortex–blade interaction. *Journal of Turbomachinary*, 1996.

[21] Murari Sridhar, Sathish Sunnam, Shraman Goswami, and Jong S Liu. Cfd aerodynamic performance validation of a two-stage high pressure turbine. In *Turbo Expo: Power for Land, Sea, and Air*, volume 54679, pages 1175–1184, 2011.

[22] Kenji Miki, Jeffrey Moder, and Meng-Sing Liou. Enhancement of the open national combustion code (openncc) and initial simulation of energy efficient engine combustor. In *AIAA/SAE/ASEE Joint Propulsion Conference*, GRC-E-DAA-TN34437, pages 01–15, 2016.

[23] Kenji Miki, Changju T Wey, and Jeffrey Moder. Computational study of modeling fully-coupled combustor-turbine interactions by the open national combustion code (openncc). In *AIAA Propulsion and Energy 2020 Forum*, page 3689, 2020.

[24] Syed Moez Hussain Mahmood and Mark G Turner. Modeling capability for cavity flows in an axial compressor. In *Turbo Expo: Power for Land, Sea, and Air*, volume 50794, page V02BT41A028. American Society of Mechanical Engineers, 2017.

[25] J Denton. *Loss mechanisms in turbomachines*, volume 78897. American Society of Mechanical Engineers, 1993.

[26] Syed Moez Hussain Mahmood, Mark G Turner, and Kiran Siddappaji. Flow characteristics of an optimized axial compressor rotor using smooth design parameters. In *Turbo Expo: Power for Land, Sea, and Air*, volume 49712, page V02CT45A018. American Society of Mechanical Engineers, 2016.

[27] Brian Adams, William Bohnhoff, Keith Dalbey, Mohamed Ebeida, John Eddy, Michael Eldred, Russell Hooper, Patricia Hough, Kenneth Hu, John Jakeman, et al. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 6.13 user's manual. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2020.

[28] Pritesh Mandal, Justin Holder, Mark G Turner, and Mark L Celestina. Design and optimization of a boundary layer ingesting propulsor. In *Turbo Expo: Power for Land, Sea, and Air*, volume 84065, page V02AT32A057. American Society of Mechanical Engineers, 2020.

[29] Adam Sieradzki, Tomasz Kwiatkowski, Mark Turner, and Borys Łukasik. Numerical modeling and design challenges of boundary layer ingesting fans. In *Turbo Expo: Power for Land, Sea, and Air*, volume 84065, page V02AT32A050. American Society of Mechanical Engineers, 2020.

[30] Huanlong Chen, Mark G Turner, Kiran Siddappaji, Syed Moez Hussain Mahmood, et al. Flow diagnosis and optimization based on vorticity dynamics for transonic compressor/fan rotor. *Open Journal of Fluid Dynamics*, 7(01):40, 2017.

[31] Justin S. Gray, John T. Hwang, Joaquim R. R. A. Martins, Kenneth T. Moore, and Bret A. Naylor. OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization. *Structural and Multidisciplinary Optimization*, 59(4):1075–1104, April 2019.

[32] Dieter Kraft. A software package for sequential quadratic programming. *Forschungsbericht-Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[33] Tom Viars. Optimization of the design of a turbine blade using openmdao. *Unpublished - Mini-Thesis for a Master's of Science in Aerospace Engineering*, 2019.

[34] Sheng Xu. *An Introduction to Scientific Computing with MATLAB® and Python Tutorials*. CRC Press, 2022.

[35] Omair Zubairi and Fridolin Weber. *Introduction to Computational Physics for Undergraduates*. Morgan & Claypool Publishers, 2018.

[36] Bharadwaj Dogga and Mark G Turner. Reverse engineering eee transonic compressor fan blade. In *AIAA SCITECH 2023 Forum*, page 2345, 2023.

Appendix A

Procedure to run airfoil reverse engineering code

The framework started from Tom Viars mini-thesis and Chris Feldmen who worked on an initial version of blade digitizer which was rewritten along with added preprocessing of coordinates to account for any general selig format airfoil. My version preprocesses data, has vectorized operations, has a more robust error handling, uses total least squared difference objective function and provides pdf reports to understand the optimization progression.

A.1 Test Run

It is assumed that the code is being run on a Linux system with enough privileges to run a shell script on a given directory. Python, Tblade3, and OpenMDAO are also assumed to be installed. Tblade3 can be obtained from GTSL's Github website. The test case steps below are for reverse engineering the NREL s809 airfoil.

1. Open a terminal in /$ ./AREF v1.0.0/001 test case files/ This directory contains the files required to test if our Framework is properly set up.

2. Run $ tblade3 3dbgbinput.1.dat dev > tblade.log to generate Tblade3 output files and initial files for the optimizer. This also tests a successful setup of Tblade3.

3. $python3 blades_difference_calculator *.py to test squared-difference calculator and obtain initial square difference log files.

4. $python3 blade_reverse_engineering*.py > optimization*.txt This runs the optimization code and writes input variables and square distance values from each iteration.

5. $ python3 optimization_post_processor_*.py optimization*.txt This produces a .pdf file with plots of the iteration data to look at the optimization status.

A.2 Input and output files Description

Before we begin reverse engineering we need to make sure that we have all the prerequisite files shown in Table A.1.

A.3 Optimization Framework

Shown in figure A.1 is the overview of ARE framework. The process begins with preparing the plot-digitized airfoil coordinates and sorting them into Selig airfoil coordinates format. This gives us the target coordinates for the reverse engineering framework. Once we have them, we prepare TBlade3 input files 3dbgbinput.1.dat and spancontrolinputs.1.dat and populate them with input and output metal angles, curvature control points, LE radius, thickness of max camber, location of max. camber thickness, and TE thickness. These values could be a best guess value for each variable. Though more modifications can be carried out to obtain a blade for a given set of coordinates, we assume that the user is currently interested only in reverse engineering airfoil coordinate values and not in obtaining a 3D blade. Once we have these values, we do a run of tblade3 to obtain the initial airfoil coordinates of our reverse engineering blade.

Now that we have the target and initial blade coordinates in a standard format, we will then run the OpenMDAO script file. The script file runs Tblade3 on the initial Tblade3 input files to produce the u-v coordinates of the airfoil. Once we have this airfoil, it runs an external python script to calculate squared difference between target and generated airfoil coordinates, then saves the values into a .dat file. This script also creates a .pdf report of the airfoil sections and writes an image file showing a visual comparison of generated airfoil with the target airfoil along with the

Table A.1: Input files for Reverse Engineering framework

File name	**Description**
$3dbgbinputfile.1.base$	Stores the template file for OpenMDAO to write Tblade3 input $3dbgbinputfile.1.dat$ file.
$3dbgbinputfile.1.dat$	Tblade3 input file: controls input, and output angles in this optimization. These values are overwritten after each iteration.
$spancontrolinputs.1.base$	Stores the template file for OpenMDAO to write Tblade3 input $3dbgbinputfile.1.dat$ file.
$spancontrolinputs.1.dat$	Tblade3 input file: stores spanwise curvature control points, leading-edge radius value, chord-wise location of the maximum thickness, maximum thickness, and thickness of the trailing edge.
$plot\_digitized\_coordinates.dat$	Plot digitized blade coordinates in $*.dat$ format. These coordinates should start at TE going up in v towards LE and return to TE - Selig format, Section B.1 has a code to rewrite a given set of coordinates into this specific order.
$blades\_difference\_calculator*.py$	Calculates squared difference between plot digitized coordiantes and reverse engineered blade iterations.
$uvblade.1.1.*$	u-v coordinates of reverse engineered blade after initial run. Blade difference calculator will use this as a starting point to calculate the squared difference.
$run\_least\_squares\_*.sh$	This shell script governs the optimization process and clears off the input files to help with the next run. It also automates Tblade3 run and calculation of squared differences.
$blade\_reverse\_engineering*.py$	Contains the OpenMDAO script with parameters for running the optimizer. The output from this code is saved into a *.txt file.
$optimization\_post\_processor\_*.py$	Post processor to visualize optimization run .dat file.

squared difference annotated on the top right corner. Once the script is done, the OpenMDAO script takes over and prints the values of input parameters and squared difference to the screen, it also writes the parameters into the .base, .log, and .dat template files and saves them. It then checks the least squared difference value with the given tolerance, if the tolerance is lower than the specified value or if the value of the least squared difference is stagnating, then the solver exits. Else, it goes back to the beginning and alters values using the SLSQP algorithm while using the

Table A.2: Output files for Reverse Engineering Framework

File name	**Description**
tblade.log	Stores run information from Tblade3 run. This file is to be used to troubleshoot a failed Tblade3 run.
Sq_diff_out.dat	Stores value of squared distance between plot digitized and reverse engineered airfoil.
./reports/problem1/n2.html	N^2 diagram for the optimization model.
uvblade.1.1.∗	Reverse engineered blade u-v coordinates
blades_difference_calculator∗.pdf	Stores .pdf plot reports of blade difference from final run of optimization run.
optimization∗ .txt	Output file from OpenMDAO run. This is used as the input file for *optimization_post_processor∗.py* script to obtain *optimization∗ .txt.pdf*
optimization∗ .txt.pdf	Post-processing plots from *optimization_post_processor_ ∗ .py* and *optimization ∗ .txt* files. This .pdf file plots squared distance, and other control variables with respect to iterations.
Airfoil_fit.png	A picture comparing plot digitized target airfoil with the reverse engineered plot with squared distance annotated on the top right.

generated blades as the new initial blade. This way it approached the target blade by minimizing the least squared distance value.

Once the optimization reaches an exit value, we then run the optimization post-processor script, which takes in the log file from the final optimization run and prepares a .pdf report with iteration vs value plots of squared difference and other input variables. It also prints out a table comparing the initial vs target values of the variables.

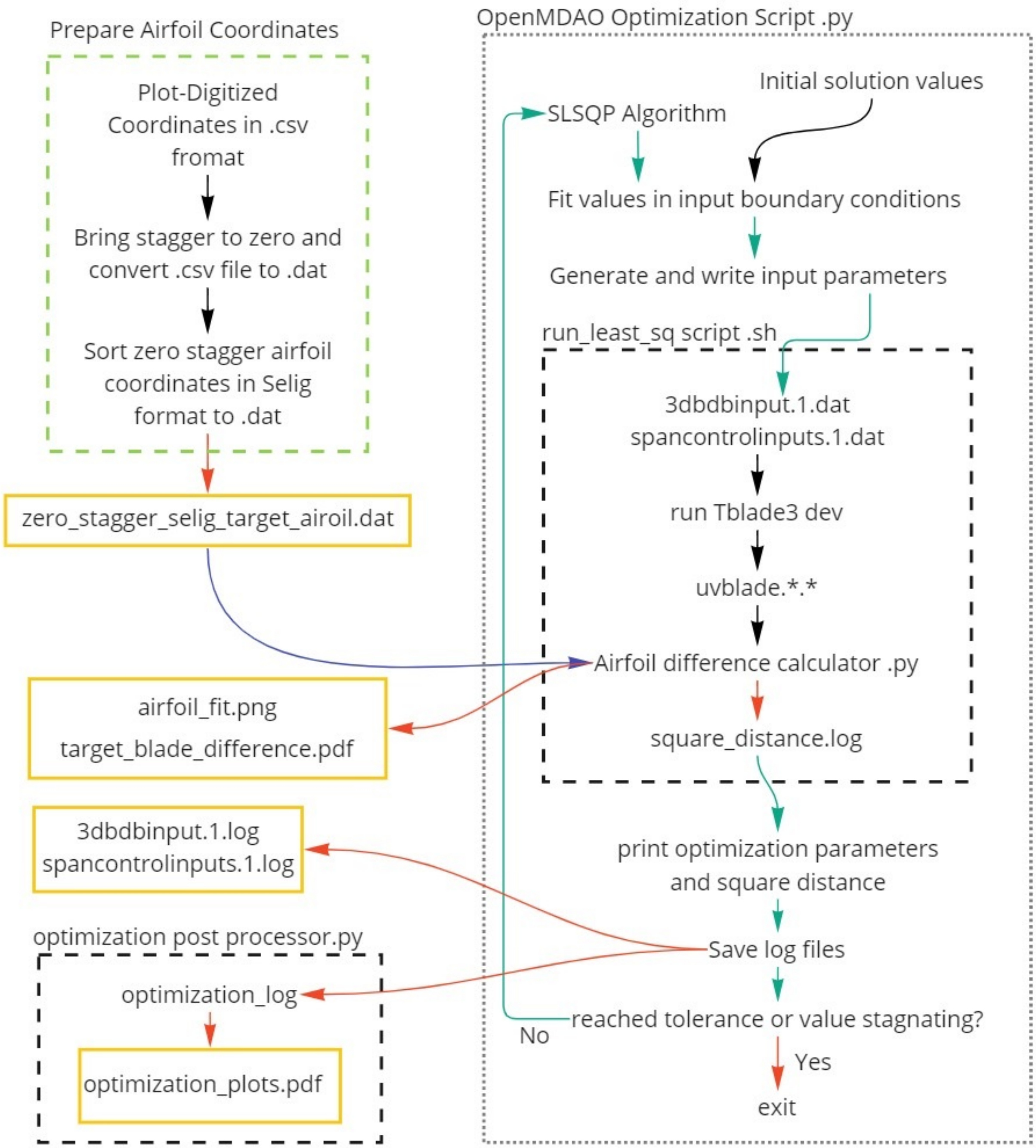


Figure A.1: Overview of Airfoil parameters reverse engineering framework

Appendix B

Codes for Airfoil Manipulation

Several airfoil manipulation codes are included in this section to help prepare any airfoil coordinates as input to our framework.

B.1 Code to sort Airfoil coordinates from TE to TE

This code sorts airfoil coodrinates in any format into a selig format: *LE to TE and then back to LE.*

```
"""
UC GTSL Lab: Sort airfoil coordiantes from LE to TE and back to LE.

Iteration 001: Calculate a mean camberline from the input airfoil coordiantes using linear curve fitting to obtain (u,v) coordiantes of mean camber line. Then for a given u in the input coordiantes if the value of v is higher than the orresponding value in the mean camber line the side will be pressure side. If not , then the side will be classified as suction side.
Iteration 005: We just realized that the order to sort airfoils was TE to LE to TE and not the originally thought version of LE to TE to LE.


# Ecxpect issue with the u extremes where u = 0 and 1. Just assign them to Suction side.
# Issue with 001 mcl line not being as expected. We intened to curve the mid section of  mcl line to produce optimum line.
```

```
# Added in plotteer to help write changes to file. Also added in a "v_sorting_flag" to compensate for optional requirement of v sorting near TE of Tip airfoil


### Always use np.argsort to sort an array by column.


Written by: Bharadwaj "Ben" Dogga

"""



import os
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages


#%% Functions to perform operations

#import both normalized airfoils using numpy and ensure that coordinates fit into a 0<u<1 and -1<v<1 range

def import_airfoil_coordinates(name_argument1, name_argument2):

    # Determine the name of the main T-Blade3 input files
    current_dir                     = os.getcwd()
    files                           = os.listdir(current_dir)
    for file in files:
        if name_argument1 in file and name_argument2 in file:
            input_file              = file.strip()
            break

    # Read main T-Blade3 input file and store the input in a list
    f                               = open(input_file, 'r')
```

```
    lines                                   = f.readlines()
    f.close()

    # lines1 = lines[106:197] # Select the range of lines that has 3 coordinates of the airfoil
    lines1 = lines.copy() # Select the range of lines that has 3 coordinates of the airfoil

    # numpy arrays for storing blade section coordinates read from the file
    m_prime                 = np.zeros(len(lines1))
    theta                   = np.zeros(len(lines1))

    # Store the blade coordinates read from the file in the arrays defined above
    i                       = 0
    for line in lines1:
        m_prime[i]          = float(line.split()[0])
        theta[i]            = float(line.split()[1])
        i                   = i + 1

    m_prime = m_prime - np.amin(m_prime)
    theta = theta - theta[np.argmin(m_prime)]

    airfoil = np.column_stack((m_prime, theta))

    return airfoil


#%% Import the airfoil coordinates:


foil1_numpy = import_airfoil_coordinates(name_argument1="E3_hub_coordinates", name_argument2=".dat")
foil1_numpy = foil1_numpy/np.amax(foil1_numpy[:,0])


```

```python
foil2_numpy = import_airfoil_coordinates(name_argument1="E3_pss_coordinates",
    name_argument2=".dat")
foil2_numpy = foil2_numpy/np.amax(foil2_numpy[:,0])


foil3_numpy = import_airfoil_coordinates(name_argument1="E3_tip_coordinates",
    name_argument2=".dat")
foil3_numpy = foil3_numpy/np.amax(foil3_numpy[:,0])

#%% A function to calculate mean camberline from input coordiantes

def fit_camber_using_polyfit(foil_numpy_input, polyfit_order=2, plot_mcl='no'):


    z = np.polyfit(foil_numpy_input[:, 0], foil_numpy_input[:, 1], polyfit_order)
    p = np.poly1d(z)
    foil_numpy_side_u = foil_numpy_input[:, 0]

    mcl_curve = np.stack((foil_numpy_side_u, p(foil_numpy_side_u)), axis=1)
    mcl_curve = mcl_curve[np.argsort(mcl_curve[:, 0])]


    if (plot_mcl=='yes'):
        plt.plot(mcl_curve[:, 0], mcl_curve[:, 1], 'o')

    return mcl_curve


#%% Function to divide sections of airfoil and calculate coordinates of mcl line to
    fix hub airfoil TE section


def calc_mean_camber_line(foil_numpy, view_airfoil=False):


    foil1_numpy_LE_side = foil_numpy[foil_numpy[:, 0] <= 0.2]
```

```python
    foil1_numpy_mcl_LE = fit_camber_using_polyfit(foil1_numpy_LE_side, polyfit_order
=2, plot_mcl='no')


    foil1_numpy_mid = foil_numpy[(foil_numpy[:, 0] > 0.2) & (foil_numpy[:, 0] <=
0.8)]
    foil1_numpy_mcl_mid = fit_camber_using_polyfit(foil1_numpy_mid, polyfit_order=2,
 plot_mcl='no')


    foil1_numpy_TE_side = foil_numpy[foil_numpy[:, 0] > 0.8]
    foil1_numpy_mcl_TE1 = fit_camber_using_polyfit(foil1_numpy_TE_side,
polyfit_order=2, plot_mcl='no')


    mcl_curve = np.concatenate((foil1_numpy_mcl_LE, foil1_numpy_mcl_mid,
foil1_numpy_mcl_TE1))


    if view_airfoil==True:

        plt.plot(foil1_numpy_LE_side[:, 0], foil1_numpy_LE_side[:, 1], 'b.', label='
input airfoil')
        plt.plot(foil1_numpy_mcl_LE[:, 0], foil1_numpy_mcl_LE[:, 1], 'g.', label='
Mean Camber line')

        plt.plot(foil1_numpy_mid[:, 0], foil1_numpy_mid[:, 1], 'b.')
        plt.plot(foil1_numpy_mcl_mid[:, 0], foil1_numpy_mcl_mid[:, 1], 'g.')

        plt.plot(foil1_numpy_TE_side[:, 0], foil1_numpy_TE_side[:, 1], 'b.')
        plt.plot(foil1_numpy_mcl_TE1[:, 0], foil1_numpy_mcl_TE1[:, 1], 'g.')

        plt.gca().set_aspect('equal')
        plt.ylim([-0.01, 1.1])
        # plt.ylim([-0.01, 0.05])
        plt.xlim([-0.01, 1.1])
```

```
        # plt.xlim([-0.08, 0.1])
        plt.legend()
        plt.figure()


    return mcl_curve

mcl_curve_001 = calc_mean_camber_line(foil1_numpy, view_airfoil=False)
mcl_curve_002 = calc_mean_camber_line(foil2_numpy, view_airfoil=False)
mcl_curve_003 = calc_mean_camber_line(foil3_numpy, view_airfoil=False)


#%% Use the mcl curve values for u coordinates and compare their v values to separate the airfoil coordinates into suction and pressure sides.

# Add in a third row to store flag of pressure side and suction side: 1 => suction side and 0 => pressure side
foil1_numpy = np.append(foil1_numpy,np.zeros([len(foil1_numpy),1]),1)
foil2_numpy = np.append(foil2_numpy,np.zeros([len(foil2_numpy),1]),1)
foil3_numpy = np.append(foil3_numpy,np.zeros([len(foil3_numpy),1]),1)

# Now, change the flag of suction side by comparing 'v' values of mcl and airfoil coordinates for a given 'u'.

def order_problems_in_coordinate_position(foil1_numpy, mcl_curve_001, v_sorting_flag=False):

    indices_suction = []
    for ii in range(len(foil1_numpy)):
        indices_suction_side = np.where((foil1_numpy[ii,0] == mcl_curve_001[:,0]))[0][0]
        if (foil1_numpy[ii, 1] >= mcl_curve_001[indices_suction_side,1]):
            foil1_numpy[ii, 2] = 1
        indices_suction.append(indices_suction_side)

```

```
    # Sort the values in third column such that all the coordinates are grouped by their suction or pressure side value.
    foil1_numpy = foil1_numpy[foil1_numpy[:, 2].argsort()]

    ## Now break the airfoil into suction and pressure side coordinates
    foil1_numpy_001_suction_side = foil1_numpy[foil1_numpy[:, 2] == 1]
    foil1_numpy_001_pressure_side = foil1_numpy[foil1_numpy[:, 2] == 0]

    ## Sort the above arrays by u values:
    foil1_numpy_001_suction_side = foil1_numpy_001_suction_side[foil1_numpy_001_suction_side[:, 0].argsort()]
    foil1_numpy_001_pressure_side = foil1_numpy_001_pressure_side[foil1_numpy_001_pressure_side[:, 0].argsort()[::-1]]


    ## Break the sides into LE and TE quadrants
    index_max_suction = np.argmax(foil1_numpy_001_suction_side[:, 1])
    foil1_numpy_001_suction_side_LE = foil1_numpy_001_suction_side[:index_max_suction, :]
    foil1_numpy_001_suction_side_TE = foil1_numpy_001_suction_side[index_max_suction:, :]


    ## Sort by v only for hub airfoil as the order of points makes sense there.
    if v_sorting_flag==True:
        ### Sort these by v values

        foil1_numpy_001_suction_side_LE = foil1_numpy_001_suction_side_LE[foil1_numpy_001_suction_side_LE[:, 0].argsort()]
        foil1_numpy_001_suction_side_TE = foil1_numpy_001_suction_side_TE[foil1_numpy_001_suction_side_TE[:, 1].argsort()[::-1]]


    # plt.plot(foil1_numpy_001_suction_side_LE[:, 0], foil1_numpy_001_suction_side_LE[:, 1], '-')
```

```
    # plt.plot(foil1_numpy_001_suction_side_TE[:, 0], foil1_numpy_001_suction_side_TE[:, 1], '-')


    foil1_numpy_001_suction_side = np.concatenate((foil1_numpy_001_suction_side_LE, foil1_numpy_001_suction_side_TE))

    # finally combine both suction and pressure side into one airoil coordiantes. This is the line you change to order TE to LE and back to TE:
    new_foil1 = np.concatenate((foil1_numpy_001_pressure_side, foil1_numpy_001_suction_side))
    new_foil1 = new_foil1[:, :2]

    return new_foil1

new_foil1 = order_problems_in_coordinate_position(foil1_numpy, mcl_curve_001, v_sorting_flag=True)
new_foil2 = order_problems_in_coordinate_position(foil2_numpy, mcl_curve_002, v_sorting_flag=True)
new_foil3 = order_problems_in_coordinate_position(foil3_numpy, mcl_curve_003, v_sorting_flag=False)

#%% plot to compare at resuts - debugging


def compare_results(old_coordinates, new_coordinates, mcl_line):
    plt.figure(figsize=(5.0, 2.5), dpi=250)
    plt.plot(old_coordinates[:, 0], old_coordinates[:, 1], '.', label='imported coordinates')
    plt.plot(new_coordinates[:, 0], new_coordinates[:, 1], '-', label='sorted coordinates')
    plt.plot(mcl_line[:, 0], mcl_line[:, 1], '-', label='mcl coordinates')
    plt.legend()
    plt.ylim([-0.02, 0.505])
    plt.xlim([-0.02, 1.01])

```

```
# compare_results(foil1_numpy, new_foil1, mcl_curve_001)
# compare_results(foil2_numpy, new_foil2, mcl_curve_002)
# compare_results(foil3_numpy, new_foil3, mcl_curve_003)

#%%

def plot_airfoil(E3_report, tblade3_file, mcl_line, page_label, ylimits=[-0.4, 0.4])
    :

    plt.figure(figsize=(8.5, 8.5), dpi=250)
    # plt.figure(dpi=250)
    plt.title(page_label)
    # plt.xlim([0.6, 1.01])
    # plt.ylim([-0.02, 0.0505])
    # plt.grid(True, which='both')
    plt.plot(E3_report[:, 0], E3_report[:, 1], '-', color='orange', label='Plot
    digitized Coordiantes')
    plt.plot(tblade3_file[:, 0], tblade3_file[:, 1], '-', color='blue', label='
    Ordered Coordiantes')
    plt.plot(mcl_line[:, 0], mcl_line[:, 1], '-', color='green', label='Interpolated
     Mean camber line')
    plt.axis('equal')
    plt.grid()
    # plt.gca().set_aspect("equal")
    plt.legend()



# Add in titles, grid and write to pdf

with PdfPages('order_airfoil_plots_005.pdf') as pdf:


### Code snippet to add in title page to a matplotlib pdf
    # plt.figure()
    # plt.axis('off')
```

```
    # plt.text(0.5,0.5,"my title",ha='center',va='center')
    # pdf.savefig()
    # plt.close()

    # Begin of actual plot code
    plot_airfoil(foil1_numpy, new_foil1, mcl_curve_001, page_label='Hub Section',
    ylimits=[-0.1, 1.1])
    pdf.savefig()  # saves the current figure into a pdf page
    plt.close()

    plot_airfoil(foil2_numpy, new_foil2, mcl_curve_002, page_label='PSS Section',
    ylimits=[-0.1, 1.1])
    pdf.savefig()
    plt.close()

    plot_airfoil(foil3_numpy, new_foil3, mcl_curve_003, page_label='Tip Section',
    ylimits=[-0.1, 1.1])
    pdf.savefig()  # or you can pass a Figure object to pdf.savefig
    plt.close()



#%% Save the coordinates to a file.

np.savetxt('E3_hub_coordinates_sorted_TE_to_LE.dat', new_foil1)
np.savetxt('E3_pss_coordinates_sorted_TE_to_LE.dat', new_foil2)
np.savetxt('E3_tip_coordinates_sorted_TE_to_LE.dat', new_foil3)
```

Appendix C

Codes for OpenMDAO and Tblade3

The following chapter of codes help prepare data to be input for OpenMDAO and Tblade3.

C.1 Streamlines Generator for 3dbgbinputfile

This code helps generate streamlines or surface-of-revolution data for Tblade3. It takes in hub and casing's 2D coordinates and generates intermediate streamlines using 1D interpolation.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import csv
import itertools
from scipy.interpolate import CubicSpline



#Import the Coordinates for Hub
hub = pd.read_csv("hub_export.dat",na_values = None,skipinitialspace=True,skiprows
    =[0],sep="\s+",keep_default_na=False,names = ["X","Y","Z"])
hub.dropna(axis=0,how='any',thresh=None)
hub.astype(float)
#print(hub.head())

```

```
Xmax_hub = hub['X'].max()
Xmin_hub = hub['X'].min()
Ymax_hub = hub['Y'].max()
Ymin_hub = hub['Y'].min()

hub['X'] = hub['X']-Xmin_hub
Xdiff_hub = Xmax_hub-Xmin_hub
Ydiff_hub = Ymax_hub-Ymin_hub
#print(Xdiff_hub)



#Import the Coordinates for LE and TE
LE_TE = pd.read_csv("unique_LE_TE_coordinates.dat",na_values = None,skipinitialspace=True,skiprows=[0],sep="\s+",keep_default_na=False,names = ["xLE","rLE","xTE","rTE"])
#Be careful wilth the order of coordinates. These start at hub and go down to casing. (Imagine measuring while standing on the ground)
LE_TE.dropna(axis=0,how='any',thresh=None)
LE_TE.astype(float)
LE_TE = LE_TE.sort_values(by=['rLE'], ascending=False)
#print(LE_TE.tail())

LE = LE_TE.iloc[:,0:2].rename(columns={"xLE": "X", "rLE": "Y"})
LE_x_coord = LE.iloc[:,0].round(3).tolist()
LE_y_coord = LE.iloc[:,1].round(3).tolist()
LE_z_coord = [0.0] * len(LE['X'])
LE_df = pd.DataFrame(list(zip(LE_x_coord, LE_y_coord, LE_z_coord)), columns =['X', 'Y', 'Z'], dtype=float).sort_values(by=['Y'], ascending=False)
LE_df = LE_df.reset_index(drop=True)
#print(LE_df)
#LE_df.to_csv('sorted_LE_df.dat', sep='\t', index=False)

TE = LE_TE.iloc[:,2:4].rename(columns={"xTE": "X", "rTE": "Y"})
TE_x_coord = TE.iloc[:,0].round(3).tolist()
TE_y_coord = TE.iloc[:,1].round(3).tolist()
```

```
TE_z_coord = [0.0] * len(TE['X'])
TE_df = pd.DataFrame(list(zip(TE_x_coord, TE_y_coord, TE_z_coord)), columns =['X', 'Y', 'Z'], dtype=float).sort_values(by=['Y'], ascending=False)
TE_df = TE_df.reset_index(drop=True)
#print(TE_df.tail())
#TE_df.to_csv('sorted_TE_df.dat', sep='\t', index=False)


# There's beeen a mismatch between the coordinates of LE, TE and Hub. So we will normlaize the hub coordinates and then use the distance been the LE and TE coordinates to estimate a scaling factor for the hub coordnates.
# Then translate the hub coordinates so that the lowest coordinate of TE coinsides with the 128th coordinate of the hub, this number is an estimate for now and could be easily altered when the right constraint is found.
#     We are translating the hub instead of the blade because the LE and TE coordinates are from the NASA-GE E3 document where as the hub is just an export from FINE/TURBO IGG, so the primary source is given more preference.
# The estimation is done using Figure 40: Full-Scale Fan Test Vehicle on Page 45 of ""NASA-GE E3 fan design.pdf"" file
# Remember to update the indices with the maximum and minimum value
dist_LE0_TE0 = np.sqrt( ((LE_TE.iloc[-1,0]-LE_TE.iloc[-1,2])**2)+((LE_TE.iloc[-1,1]-LE_TE.iloc[-1,3])**2) )
#print(dist_LE0_TE0)
norm_hub = hub.copy()
norm_hub = norm_hub/Xdiff_hub
#print(norm_hub.head())
scaled_hub = norm_hub.copy()
scaled_hub = scaled_hub*dist_LE0_TE0*5.5
translated_hub = scaled_hub.copy()
xhub_TE_diff = LE_TE.iloc[-1,2] - translated_hub.iloc[127,0]
yhub_TE_diff = LE_TE.iloc[-1,3] - translated_hub.iloc[127,1]
#print(xhub_TE_diff)
#print(yhub_TE_diff)
translated_hub['X'] = translated_hub['X'] + xhub_TE_diff
translated_hub['Y'] = translated_hub['Y'] + yhub_TE_diff - 30e-2
translated_hub = translated_hub.round(17)
```

```
translated_hub.to_csv('translated_hub.dat', sep='\t', index=False)


# Using LE, TE and the scaled Hub Coordinates, construct the casing.
#  We will construct the casing using three sections, one before the LE, second from LE to TE and finally

casing = translated_hub.copy()
#print(casing.head())
#   Part 1 of casing
casing.iloc[:95,1] = LE_TE.iloc[0,1]+30e-2
casing1 = casing.iloc[0:95,:]
#print(casing1.tail())

#   Part 2 of casing
def intermediates(p1, p2, nb_points=8):
    """"Return a list of nb_points equally spaced points
    between p1 and p2"""
    # If we have 8 intermediate points, we have 8+1=9 spaces
    # between p1 and p2
    x_spacing = (p2[0] - p1[0]) / (nb_points + 1)
    y_spacing = (p2[1] - p1[1]) / (nb_points + 1)

    return [[p1[0] + i * x_spacing, p1[1] +  i * y_spacing, 0.0]
            for i in range(1, nb_points+1)]
case3_points = intermediates([casing.iloc[93,0]+10e-1, LE_TE.iloc[0,1]+30e-2], [casing.iloc[125,0]-10e-2, LE_TE.iloc[0,3]+1e-2], nb_points=29)
case3_df = pd.DataFrame(case3_points, columns=['X','Y', 'Z'])
caseing1_df = casing1.append(case3_df, ignore_index=True)

#  Part 3 of casing
casing.iloc[124:,1] = LE_TE.iloc[0,3]+10e-2
caseing1_dg = caseing1_df.append(casing.iloc[124:,], ignore_index=True)
caseing1_dg = caseing1_dg.round(6)
caseing1_dg.to_csv('caseing1_dg.dat', sep='\t', index=False)

```

```python
#This equates the number of points in the casing to the number of points in the hub,
# this can also be achieved by simply making sure that the ending abscissa of both
# the casing and hub are the same

    #final comparison of coordinates:
#print(translated_hub.tail())
#print(caseing1_dg.head())



# Streamline construction: We need to construct 21 streamlines between the hub and casing. Also, we need to make sure that the streamlines include the LE and TE coodinates.
    # This is done in three parts. We start with making a dataframe of all the points on the left of LE and divide them into 21 sections

def streamline_coordinates_construct(x_iloc):

    if x_iloc <110:
        #Use the LE coodinates to space the streamlines
        temp_coords=[]
        for i in range(len(LE_df['X'])):
            y20 = translated_hub.iloc[x_iloc,1]+((LE_df.iloc[i,1])*((caseing1_dg.iloc[x_iloc,1]-translated_hub.iloc[x_iloc,1])/(LE_df.iloc[0,1]-LE_df.iloc[-1,1])))
            temp_coords.append(y20)
            temp_coords = [round(num, 6) for num in temp_coords]
            sorted(temp_coords, reverse=True)
        #print(temp_coords)
        #print(translated_hub.iloc[x_iloc, 1])
        stream_deav = max(temp_coords) - caseing1_dg.iloc[x_iloc, 1]
        temp_coords = [x-stream_deav for x in temp_coords]
        streamlines_sect_1_y = temp_coords
        # to evenly space the streamlines
        #streamlines_sect_1_y = np.linspace(caseing1_dg.iloc[x_iloc,1],translated_hub.iloc[x_iloc,1],21)
        #print(streamlines_sect_1_y)
```

```
        streamlines_sect_1_x = [caseing1_dg.iloc[x_iloc,0]] * len(streamlines_sect_1_y)
        #print(streamlines_sect_1_x)
        streamlines_sect_1_z = [0.0] * len(streamlines_sect_1_x)
        #print(streamlines_sect_1_z)
        streamlines_sect_next_df = pd.DataFrame(list(zip(streamlines_sect_1_x, streamlines_sect_1_y, streamlines_sect_1_z)), columns =['X', 'Y', 'Z'], dtype=float)
        #print(streamlines_sect_next_df.tail())
    else:
        temp_coords=[]
        for i in range(len(TE_df['X'])):
            y20 = translated_hub.iloc[x_iloc,1]+((TE_df.iloc[i,1])*((caseing1_dg.iloc[x_iloc,1]-translated_hub.iloc[x_iloc,1])/(TE_df.iloc[0,1]-TE_df.iloc[-1,1])))
            temp_coords.append(y20)
            temp_coords = [round(num, 6) for num in temp_coords]
            sorted(temp_coords, reverse=True)
        stream_deav = max(temp_coords) - caseing1_dg.iloc[x_iloc, 1]
        temp_coords = [x-stream_deav for x in temp_coords]
        streamlines_sect_1_y = temp_coords
        streamlines_sect_1_x = [caseing1_dg.iloc[x_iloc,0]] * len(streamlines_sect_1_y)
        streamlines_sect_1_z = [0.0] * len(streamlines_sect_1_x)
        streamlines_sect_next_df = pd.DataFrame(list(zip(streamlines_sect_1_x, streamlines_sect_1_y, streamlines_sect_1_z)), columns =['X', 'Y', 'Z'], dtype=float)
        #streamlines_sect_next_df = streamlines_sect_next_df.sort_values(by='Y', ascending=False)
    return streamlines_sect_next_df

list_x_vals = [np.arange(60, 209, 10)]
#print(list_x_vals)

#Part1: Generate the points from LE and LE

streamlines_sect01_next_df = streamline_coordinates_construct(60)
```

```
#print(streamlines_sect01_next_df.tail())
streamlines_sect02_next_df = streamline_coordinates_construct(70)
streamlines_sect03_next_df = streamline_coordinates_construct(80)
streamlines_sect04_next_df = streamline_coordinates_construct(85)
streamlines_sect05_next_df = streamline_coordinates_construct(98)
streamlines_sect06_next_df = streamline_coordinates_construct(105)
streamlines_sect07_next_df = streamline_coordinates_construct(112)
streamlines_sect08_next_df = streamline_coordinates_construct(120)
streamlines_sect09_next_df = streamline_coordinates_construct(130)
streamlines_sect10_next_df = streamline_coordinates_construct(140)
streamlines_sect11_next_df = streamline_coordinates_construct(150)
streamlines_sect12_next_df = streamline_coordinates_construct(160)
streamlines_sect13_next_df = streamline_coordinates_construct(170)
streamlines_sect14_next_df = streamline_coordinates_construct(180)
streamlines_sect15_next_df = streamline_coordinates_construct(190)
streamlines_sect16_next_df = streamline_coordinates_construct(209)

#print(list_of_coordinates)

""" There is an issue with the current LE and TE coordinates where in they are no uniformly spced and there are duplicated values as well. We replace them with an evenly spaced linear interpolation version using "interpolation_LE_TE.py" file"""

#interp_LE_df = pd.read_csv("/home/bharadwaj/work/Meeting_10/Meeting docs/Chris Feldman Docs/E3_trial1/E3_streamlines/sorted_LE_df.dat",na_values = None, skipinitialspace=True,sep="\s+")
#print(interp_LE_df)
#interp_TE_df = pd.read_csv("/home/bharadwaj/work/Meeting_10/Meeting docs/Chris Feldman Docs/E3_trial1/E3_streamlines/sorted_TE_df.dat",na_values = None, skipinitialspace=True,sep="\s+")
#print(interp_TE_df)

#Part2: Apppend the generated points into a single dataframe

streamlines_final_df = streamlines_sect01_next_df.append(streamlines_sect02_next_df, ignore_index = True)
```

```
#print(streamlines_final_df.tail())
streamlines_final_df = streamlines_final_df.append(streamlines_sect03_next_df,
    ignore_index = True)
streamlines_final_df = streamlines_final_df.append(streamlines_sect04_next_df,
    ignore_index = True)
streamlines_final_df = streamlines_final_df.append(LE_df, ignore_index = True)
streamlines_final_df = streamlines_final_df.append(streamlines_sect05_next_df,
    ignore_index = True)
streamlines_final_df = streamlines_final_df.append(streamlines_sect06_next_df,
    ignore_index = True)
streamlines_final_df = streamlines_final_df.append(streamlines_sect07_next_df,
    ignore_index = True)
streamlines_final_df = streamlines_final_df.append(streamlines_sect08_next_df,
    ignore_index = True)
streamlines_final_df = streamlines_final_df.append(TE_df, ignore_index = True)
streamlines_final_df = streamlines_final_df.append(streamlines_sect09_next_df,
    ignore_index = True)
streamlines_final_df = streamlines_final_df.append(streamlines_sect10_next_df,
    ignore_index = True)
streamlines_final_df = streamlines_final_df.append(streamlines_sect11_next_df,
    ignore_index = True)
streamlines_final_df = streamlines_final_df.append(streamlines_sect12_next_df,
    ignore_index = True)
streamlines_final_df = streamlines_final_df.append(streamlines_sect13_next_df,
    ignore_index = True)
streamlines_final_df = streamlines_final_df.append(streamlines_sect14_next_df,
    ignore_index = True)
streamlines_final_df = streamlines_final_df.append(streamlines_sect15_next_df,
    ignore_index = True)
streamlines_final_df = streamlines_final_df.append(streamlines_sect16_next_df,
    ignore_index = True)
#print(streamlines_final_df.head())
#print(streamlines_final_df.tail())

#df.iloc[::5, :]

```

```python
#Part3: Regroup the coodinates into separate dataframes containing individual
    streamline data.
strm_cutoff = int(len(TE_df))

streamline_01_coordiantes = streamlines_final_df.iloc[::strm_cutoff, :].reset_index(
    drop=True)
#print(streamline_01_coordiantes.tail())
streamline_02_coordiantes = streamlines_final_df.iloc[1::strm_cutoff, :].reset_index
    (drop=True)
streamline_03_coordiantes = streamlines_final_df.iloc[2::strm_cutoff, :].reset_index
    (drop=True)
streamline_04_coordiantes = streamlines_final_df.iloc[3::strm_cutoff, :].reset_index
    (drop=True)
streamline_05_coordiantes = streamlines_final_df.iloc[4::strm_cutoff, :].reset_index
    (drop=True)
streamline_06_coordiantes = streamlines_final_df.iloc[5::strm_cutoff, :].reset_index
    (drop=True)
streamline_07_coordiantes = streamlines_final_df.iloc[6::strm_cutoff, :].reset_index
    (drop=True)
streamline_08_coordiantes = streamlines_final_df.iloc[7::strm_cutoff, :].reset_index
    (drop=True)
streamline_09_coordiantes = streamlines_final_df.iloc[8::strm_cutoff, :].reset_index
    (drop=True)
streamline_10_coordiantes = streamlines_final_df.iloc[9::strm_cutoff, :].reset_index
    (drop=True)
streamline_11_coordiantes = streamlines_final_df.iloc[10::strm_cutoff, :].
    reset_index(drop=True)
streamline_12_coordiantes = streamlines_final_df.iloc[11::strm_cutoff, :].
    reset_index(drop=True)
streamline_13_coordiantes = streamlines_final_df.iloc[12::strm_cutoff, :].
    reset_index(drop=True)
streamline_14_coordiantes = streamlines_final_df.iloc[13::strm_cutoff, :].
    reset_index(drop=True)
streamline_15_coordiantes = streamlines_final_df.iloc[14::strm_cutoff, :].
    reset_index(drop=True)
```

```
streamline_16_coordiantes = streamlines_final_df.iloc[15::strm_cutoff, :].
    reset_index(drop=True)
streamline_17_coordiantes = streamlines_final_df.iloc[16::strm_cutoff, :].
    reset_index(drop=True)
streamline_18_coordiantes = streamlines_final_df.iloc[17::strm_cutoff, :].
    reset_index(drop=True)


#print(streamline_18_coordiantes.tail())

# Though this was a crude way to get the streamlines, a more smoother approach
    instead of calculating all the streamline coordninates using LE or TE coordintes
    would be to just calculate the first and last column of coodrdinates using the
    above method (Part1) and then linerarly interpolate any number of points between
    them such that you would have a smoother transition between inlet, LE, TE and
    outlet.


caseing1_dg = caseing1_dg.iloc[59:,:] #This chops off the casing extesnion and
    provides a more realistic view of the geometry.


# Part4: Create an output file such that you can paste the hub, streamlines and
    casing data into 3dbgb input file. The order is: Hub, Streamlines, casing.
list_0 = [0]
zero_df = pd.DataFrame(list(zip(list_0, list_0)), columns =['x_s', 'r_s'], dtype=int
    )


streamlines_hub_tblade3 = pd.DataFrame(list(zip(translated_hub['X'], translated_hub[
    'Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_df = streamlines_hub_tblade3.copy()
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_01_tblade3 = pd.DataFrame(list(zip(streamline_18_coordiantes['X'],
    streamline_18_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_01_tblade3 = streamlines_01_tblade3.round(7)
```

```python
streamlines_df = streamlines_df.append(streamlines_01_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_02_tblade3 = pd.DataFrame(list(zip(streamline_17_coordiantes['X'],
    streamline_17_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_02_tblade3 = streamlines_02_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_02_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_03_tblade3 = pd.DataFrame(list(zip(streamline_16_coordiantes['X'],
    streamline_16_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_03_tblade3 = streamlines_03_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_03_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_04_tblade3 = pd.DataFrame(list(zip(streamline_15_coordiantes['X'],
    streamline_15_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_04_tblade3 = streamlines_04_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_04_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_05_tblade3 = pd.DataFrame(list(zip(streamline_14_coordiantes['X'],
    streamline_14_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_05_tblade3 = streamlines_05_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_05_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_06_tblade3 = pd.DataFrame(list(zip(streamline_13_coordiantes['X'],
    streamline_13_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_06_tblade3 = streamlines_06_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_06_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_07_tblade3 = pd.DataFrame(list(zip(streamline_12_coordiantes['X'],
    streamline_12_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_07_tblade3 = streamlines_07_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_07_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_08_tblade3 = pd.DataFrame(list(zip(streamline_11_coordiantes['X'],
    streamline_11_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_08_tblade3 = streamlines_08_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_08_tblade3, ignore_index = True)
```

```
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_09_tblade3 = pd.DataFrame(list(zip(streamline_10_coordiantes['X'],
    streamline_10_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_09_tblade3 = streamlines_09_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_09_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_10_tblade3 = pd.DataFrame(list(zip(streamline_09_coordiantes['X'],
    streamline_09_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_10_tblade3 = streamlines_10_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_10_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_11_tblade3 = pd.DataFrame(list(zip(streamline_08_coordiantes['X'],
    streamline_08_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_11_tblade3 = streamlines_11_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_11_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_12_tblade3 = pd.DataFrame(list(zip(streamline_07_coordiantes['X'],
    streamline_07_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_12_tblade3 = streamlines_12_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_12_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_13_tblade3 = pd.DataFrame(list(zip(streamline_06_coordiantes['X'],
    streamline_06_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_13_tblade3 = streamlines_13_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_13_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_14_tblade3 = pd.DataFrame(list(zip(streamline_05_coordiantes['X'],
    streamline_05_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_14_tblade3 = streamlines_14_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_14_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_15_tblade3 = pd.DataFrame(list(zip(streamline_04_coordiantes['X'],
    streamline_04_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_15_tblade3 = streamlines_15_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_15_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
```

```
streamlines_16_tblade3 = pd.DataFrame(list(zip(streamline_03_coordiantes['X'],
    streamline_03_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_16_tblade3 = streamlines_16_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_16_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_17_tblade3 = pd.DataFrame(list(zip(streamline_02_coordiantes['X'],
    streamline_02_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_17_tblade3 = streamlines_17_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_17_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_18_tblade3 = pd.DataFrame(list(zip(streamline_01_coordiantes['X'],
    streamline_01_coordiantes['Y'])), columns =['x_s', 'r_s'], dtype=float)
streamlines_18_tblade3 = streamlines_18_tblade3.round(7)
streamlines_df = streamlines_df.append(streamlines_18_tblade3, ignore_index = True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
streamlines_casing_tblade3 = pd.DataFrame(list(zip(caseing1_dg['X'], caseing1_dg['Y'
    ])), columns =['x_s', 'r_s'], dtype=float)
streamlines_df = streamlines_df.append(streamlines_casing_tblade3, ignore_index =
    True)
streamlines_df = streamlines_df.append(zero_df, ignore_index = True)
print(streamlines_df)
streamlines_df.to_csv('tblade3_streamlines_w_format.dat', sep='\t', index=False)



# Plot the files
plt.plot(translated_hub['X'],translated_hub['Y'], label='Translated Hub Coordinates'
    , color='Violet')
plt.plot(LE_TE['xLE'],LE_TE['rLE'], label='LE', color='red')
plt.plot(LE_TE['xTE'],LE_TE['rTE'], label='TE', color='blue')
plt.plot(caseing1_dg['X'],caseing1_dg['Y'], label='Generated Casing', color='black')
plt.plot(streamlines_final_df['X'], streamlines_final_df['Y'], 'g^', label='
    Streamline Coordinates')

# Plot Streamlines
```

```
plt.plot(streamline_01_coordiantes['X'],streamline_01_coordiantes['Y'], label='Streamlines', color='grey')
plt.plot(streamline_02_coordiantes['X'],streamline_02_coordiantes['Y'], color='grey')
plt.plot(streamline_03_coordiantes['X'],streamline_03_coordiantes['Y'], color='grey')
plt.plot(streamline_04_coordiantes['X'],streamline_04_coordiantes['Y'], color='grey')
plt.plot(streamline_05_coordiantes['X'],streamline_05_coordiantes['Y'], color='grey')
plt.plot(streamline_06_coordiantes['X'],streamline_06_coordiantes['Y'], color='grey')
plt.plot(streamline_07_coordiantes['X'],streamline_07_coordiantes['Y'], color='grey')
plt.plot(streamline_08_coordiantes['X'],streamline_08_coordiantes['Y'], color='grey')
plt.plot(streamline_09_coordiantes['X'],streamline_09_coordiantes['Y'], color='grey')
plt.plot(streamline_10_coordiantes['X'],streamline_10_coordiantes['Y'], color='grey')
plt.plot(streamline_11_coordiantes['X'],streamline_11_coordiantes['Y'], color='grey')
plt.plot(streamline_12_coordiantes['X'],streamline_12_coordiantes['Y'], color='grey')
plt.plot(streamline_13_coordiantes['X'],streamline_13_coordiantes['Y'], color='grey')
plt.plot(streamline_14_coordiantes['X'],streamline_14_coordiantes['Y'], color='grey')
plt.plot(streamline_15_coordiantes['X'],streamline_15_coordiantes['Y'], color='grey')
plt.plot(streamline_16_coordiantes['X'],streamline_16_coordiantes['Y'], color='grey')
plt.plot(streamline_17_coordiantes['X'],streamline_17_coordiantes['Y'], color='grey')
plt.plot(streamline_18_coordiantes['X'],streamline_18_coordiantes['Y'], color='grey')
```

```
#plt.plot(streamline_21_coordiantes['X'],streamline_21_coordiantes['Y'], color='grey')




#plt.plot(case3_df['X'],case3_df['Y'], label='slant Casing', color='red')
plt.legend()
plt.show()

#Write to file
#caseing1_df.to_csv('caseing1_df.dat', sep='\t', index=False)

#Debugging
#plt.plot(foil_level05['X'],foil_level05['Y'], label='Blade Level 05', color='blue')
#plt.plot(r_div, casing['Y'], label='r_div', marker='o')
#plt.plot(hub['X'],hub['Y'], label='Hub Coordinates', color='red')
#plt.plot(scaled_hub['X'],scaled_hub['Y'], label='Scaled Hub Coordinates', color='red')
#plt.plot(norm_hub['X'],norm_hub['Y'], label='Norm Hub Coordinates', color='pink')
#print(intermediates([casing.iLoc[127,1]+10e-4, 2+10e-4], [10+10e-4, 6.5+10e-4], nb_points=34))
#print(casing.iloc[127,1]+10e-4)
#print(casing.iloc[91,1]+10e-4)
#plt.plot(casing['X'],casing['Y'], label='Generated Casing', color='blue')
```

C.2 Blade difference calculator

This code implements least squared difference calculation as explained in Section 2.1 along with diagnosis plots on optimization.

```
"""

Why iter 007:
    The initial idea was to use squared difference instead of root of squared difference and we missted that part.
```

```
        This issue was fixed by removing the root in all difference calulations.

Why iter 006:
    There is an issue with the new airfoils where the mid suction side only has one point. Figure out why?
        The issue was caused by the definitiaon with which we defined suction and pressure sides using mean of the v values on both sides.
        Once thickness began to reduce, the code began eliminating suction side points.
        We fixed it by adding in a function to calculate mcl line and then use the mcl line to define which point goes to the suciton side and which one goes to the pressure side. That fixed the issue for now.


Written by: Bharadwaj "Ben" Dogga

"""



import os
import sys
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate # Had to use this instead of np.interp because of some bugs in that module
from matplotlib.backends.backend_pdf import PdfPages



#%% Functions to perform operations

#import both normalized airfoils using numpy and ensure that coordinates fit into a 0<u<1 and -1<v<1 range

def import_airfoil_coordinates(name_argument1, name_argument2):
```

```
    # Determine the name of the main T-Blade3 input files
    current_dir                     = os.getcwd()
    files                           = os.listdir(current_dir)
    for file in files:
        if name_argument1 in file and name_argument2 in file:
            input_file              = file.strip()
            break

    # Read main T-Blade3 input file and store the input in a list
    f                               = open(input_file, 'r')
    lines                           = f.readlines()
    f.close()

    # lines1 = lines[106:197] # Select the range of lines that has 3 coordinates of the airfoil
    lines1 = lines.copy() # Select the range of lines that has 3 coordinates of the airfoil

    # numpy arrays for storing blade section coordinates read from the file
    m_prime             = np.zeros(len(lines1))
    theta               = np.zeros(len(lines1))

    # Store the blade coordinates read from the file in the arrays defined above
    i                   = 0
    for line in lines1:
        m_prime[i]      = float(line.split()[0])
        theta[i]        = float(line.split()[1])
        i               = i + 1

    m_prime = m_prime - np.amin(m_prime)
    theta = theta - theta[np.argmin(m_prime)]

    airfoil = np.column_stack((m_prime, theta))

    return airfoil
```

```python



# Interpolate values between two sets

def interpolate_section(xvals, xx, yy):
    yvals = np.interp(xvals, xx, yy)
    f = interpolate.interp1d(xx, yy)
    yvals = f(xvals)
    return np.array((xvals, yvals)).T
    # return {'yvalues': yvals, 'xvalues': xvals}



# Make sure that x is a subset of xx
def make_sure_x_subset_of_xx_2D_array(x, xx, ascending=False):

    if ascending==False:
        for ii in range(len(x)):
            if (x[0,0] > xx[0,0]):
                # print(x[0,0] < xx[0,0])
                x = np.delete(x, [0, 0], axis=0)
                # print(x)
                if (x[0,0] == xx[0,0]):
                    break
            else:
                # print("within xx - lower")
                break
        # print(x)

        for jj in reversed(range(len(x))):
            if (x[-1,0] < xx[-1,0]):
                # print(x[-1,0] < xx[-1,0])
                x = np.delete(x, [-1,-1], axis=0)
                if (x[-1,0] == xx[-1,0]):
                    break
```

```
            else:
                # print("within xx - upper")
                break

    else:
        for ii in range(len(x)):
            if (x[0,0] < xx[0,0]):
                # print(x[0,0] < xx[0,0])
                x = np.delete(x, [0, 0], axis=0)
                # print(x)
                if (x[0,0] == xx[0,0]):
                    break
            else:
                # print("within xx - lower")
                break
        # print(x)

        for jj in reversed(range(len(x))):
            if (x[-1,0] > xx[-1,0]):
                x = np.delete(x, [-1,-1], axis=0)
                if (x[-1,0] == xx[-1,0]):
                    break
            else:
                # print("within xx - upper")
                break
    # print(x)
    return x



def test_plot(numpy_array):
    plt.figure(dpi=150)
    plt.plot(numpy_array[:, 0], numpy_array[:, 1], '-', numpy_array[:, 0], numpy_array[:, 1], '.', color='blue')


```

```
def assign_coordinates_to_suction_and_pressure_side(suction_side, pressure_side):
    #### Ensure that pressure and suction are are appropriately allocated
    if (sum(pressure_side[:, 1]) > sum(suction_side[:, 1])):
        suction_side_store = pressure_side
        pressure_side_store = suction_side
    else:
        pressure_side_store = pressure_side
        suction_side_store = suction_side

    return suction_side_store, pressure_side_store


def order_coordinates(airfoil_part_coordinates, suction_side_flag=False):
    if (suction_side_flag==True):
        if (airfoil_part_coordinates[0, 0] < airfoil_part_coordinates[-1, 0]):
            airfoil_part_coordinates = airfoil_part_coordinates[::-1]

    else:
        if (airfoil_part_coordinates[0, 0] > airfoil_part_coordinates[-1, 0]):
            airfoil_part_coordinates = airfoil_part_coordinates[::-1]

    return airfoil_part_coordinates



def center_v_1_in_Trailing_Edge_array(Trailing_Edge_array):
    """
    This function takes in the trailing edge section and returns the array rolled
     with the maximum v difference points on the extremes of array.
    It ensures that the cut section array begins at u = uTE and goes up to u = 1 and
     comes back to u = uTE to ensure proper calculation of difference.
    """

    difference_array = np.zeros(len(Trailing_Edge_array))
    for ii in range(len(Trailing_Edge_array)):
        if ii == len(Trailing_Edge_array):
```

```
            difference_value = Trailing_Edge_array[-1, 1] - Trailing_Edge_array[0, 1]
            difference_array[-1] = abs(difference_value)
        else:
            difference_value = Trailing_Edge_array[ii, 1] - Trailing_Edge_array[ii-1, 1]
            difference_array[ii] = abs(difference_value)

    #### The next line is the code that fixed issues with rolling.
    difference_index = len(Trailing_Edge_array) -  np.argmax(difference_array)
    Trailing_Edge_array = np.roll(Trailing_Edge_array, difference_index, axis=0)

    return Trailing_Edge_array


#%% A function to calculate mean camberline from input coordiantes

def fit_camber_using_polyfit(foil_numpy_input, polyfit_order=2, plot_mcl='no'):


    z = np.polyfit(foil_numpy_input[:, 0], foil_numpy_input[:, 1], polyfit_order)
    p = np.poly1d(z)
    foil_numpy_side_u = foil_numpy_input[:, 0]

    mcl_curve = np.stack((foil_numpy_side_u, p(foil_numpy_side_u)), axis=1)
    mcl_curve = mcl_curve[np.argsort(mcl_curve[:, 0])]


    if (plot_mcl=='yes'):
        plt.figure(dpi=150)
        plt.plot(mcl_curve[:, 0], mcl_curve[:, 1], '.', label='mcl_curve')
        plt.plot(foil_numpy_input[:, 0], foil_numpy_input[:, 1], '.', label='airfoil_coordiantes')
        plt.xlim(-0.01, 1.01)
        plt.ylim(-0.07, 0.2)
        plt.legend()
```

```python


        return mcl_curve


#%% Use mcl function to separate suction side coordinates from pressre side ones.



#%% function to divide airfoil into six parts:
def break_airfoil_into_six_parts(airfoil_coordinates_array, uLE, uTE, visual_debug=False):

    """
    This function takes in an airfoil section coordinates as numpy array and breaks it down into six sections based on input uLE and uTE.
    All u coordinates on suction side need to be sorted in descending order and pressure side u needs to be in ascending order.
    We need to ensure that all suction parts stay on the suction side and vice versa for pressure side.
    """

    # Start with cutting the blade along u: LE | mid | TE
    ## Target Airoil section: LE
    airfoil_coordinates_array_Leading_Edge = airfoil_coordinates_array[np.where(airfoil_coordinates_array[:,0] < uLE)]

    ### Divide the LE side into pressure and suction side:
    airfoil_coordinates_array_u_0_index = np.argwhere(airfoil_coordinates_array_Leading_Edge[:,0] == 0)
    airfoil_coordinates_array_Leading_Edge_pressure = airfoil_coordinates_array_Leading_Edge[:airfoil_coordinates_array_u_0_index[0][0],:]
    airfoil_coordinates_array_Leading_Edge_suction = airfoil_coordinates_array_Leading_Edge[airfoil_coordinates_array_u_0_index[0][0]:,:]
```

```python

airfoil_coordinates_array_Leading_Edge_suction ,
 airfoil_coordinates_array_Leading_Edge_pressure =
 assign_coordinates_to_suction_and_pressure_side(
 airfoil_coordinates_array_Leading_Edge_suction ,
 airfoil_coordinates_array_Leading_Edge_pressure)
airfoil_coordinates_array_Leading_Edge_suction = order_coordinates(
 airfoil_coordinates_array_Leading_Edge_suction , suction_side_flag=True)
airfoil_coordinates_array_Leading_Edge_pressure = order_coordinates(
 airfoil_coordinates_array_Leading_Edge_pressure , suction_side_flag=False)


if (visual_debug==True):
    test_plot(airfoil_coordinates_array_Leading_Edge)




## Target Airoil section: mid
### The airfoil mid sections had to be sorted to align with other parts.
airfoil_coordinates_array_mid = airfoil_coordinates_array[np.where((
 airfoil_coordinates_array[:,0] > uLE) & (airfoil_coordinates_array[:,0] < uTE))]
foil_numpy_mcl_mid = fit_camber_using_polyfit(airfoil_coordinates_array_mid ,
 polyfit_order=2, plot_mcl='no')
foil_numpy = np.append(airfoil_coordinates_array_mid ,np.zeros([len(
 airfoil_coordinates_array_mid),1]),1)

for ii in range(len(foil_numpy)):
    indices_suction_side = np.where((foil_numpy[ii,0] == foil_numpy_mcl_mid[:,0])
 )[0][0]
    if (foil_numpy[ii, 1] >= foil_numpy_mcl_mid[indices_suction_side ,1]):
        foil_numpy[ii, 2] = 1
# print(np.shape(foil_numpy))

airfoil_coordinates_array_mid_pressure = foil_numpy[foil_numpy[:, 2] == 1]
airfoil_coordinates_array_mid_suction = foil_numpy[foil_numpy[:, 2] == 0]
```

```
airfoil_coordinates_array_mid_suction, airfoil_coordinates_array_mid_pressure = assign_coordinates_to_suction_and_pressure_side(airfoil_coordinates_array_mid_suction, airfoil_coordinates_array_mid_pressure)
airfoil_coordinates_array_mid_suction = order_coordinates(airfoil_coordinates_array_mid_suction, suction_side_flag=True)
airfoil_coordinates_array_mid_pressure = order_coordinates(airfoil_coordinates_array_mid_pressure, suction_side_flag=False)
airfoil_coordinates_array_mid_suction = airfoil_coordinates_array_mid_suction[:, :2]
airfoil_coordinates_array_mid_pressure = airfoil_coordinates_array_mid_pressure[:, :2]
# print(np.shape(airfoil_coordinates_array_mid_suction))


plot_mcl='no'
if (plot_mcl=='yes'):
    plt.figure(dpi=150)
    plt.plot(foil_numpy_mcl_mid[:, 0], foil_numpy_mcl_mid[:, 1], '.', label='mcl_curve')
    plt.plot(airfoil_coordinates_array_mid_suction[:, 0], airfoil_coordinates_array_mid_suction[:, 1], '--', label='suction_side', color='blue')
    plt.plot(airfoil_coordinates_array_mid_pressure[:, 0], airfoil_coordinates_array_mid_pressure[:, 1], '--', label='pressure_side', color='orange')
    plt.xlim(-0.01, 1.01)
    plt.ylim(-0.07, 0.2)
    plt.legend()



## Target Airoil section: TE
airfoil_coordinates_array_Trailing_Edge = airfoil_coordinates_array[np.where(airfoil_coordinates_array[:,0] > uTE)]
airfoil_coordinates_array_Trailing_Edge = center_v_1_in_Trailing_Edge_array(airfoil_coordinates_array_Trailing_Edge)
```

```python

    if (visual_debug==True):
        test_plot(airfoil_coordinates_array_Trailing_Edge)

    ### Divide the TE side into pressure and suction side:
    airfoil_coordinates_array_u_1_index = np.argwhere(
     airfoil_coordinates_array_Trailing_Edge[:,0] == 1)
    airfoil_coordinates_array_Trailing_Edge_pressure =
     airfoil_coordinates_array_Trailing_Edge[:airfoil_coordinates_array_u_1_index
     [0][0],:]
    airfoil_coordinates_array_Trailing_Edge_suction =
     airfoil_coordinates_array_Trailing_Edge[airfoil_coordinates_array_u_1_index
     [0][0]:,:]
    airfoil_coordinates_array_Trailing_Edge_suction,
     airfoil_coordinates_array_Trailing_Edge_pressure =
     assign_coordinates_to_suction_and_pressure_side(
     airfoil_coordinates_array_Trailing_Edge_suction,
     airfoil_coordinates_array_Trailing_Edge_pressure)
    airfoil_coordinates_array_Trailing_Edge_suction = order_coordinates(
     airfoil_coordinates_array_Trailing_Edge_suction, suction_side_flag=True)
    airfoil_coordinates_array_Trailing_Edge_pressure = order_coordinates(
     airfoil_coordinates_array_Trailing_Edge_pressure, suction_side_flag=False)



    ### Combine all sections into a dictionary

    return {'suction_TE': airfoil_coordinates_array_Trailing_Edge_suction, '
     suction_mid': airfoil_coordinates_array_mid_suction, 'suction_LE':
     airfoil_coordinates_array_Leading_Edge_suction, 'pressure_LE':
     airfoil_coordinates_array_Leading_Edge_pressure, 'pressure_mid':
     airfoil_coordinates_array_mid_pressure, 'pressure_TE':
     airfoil_coordinates_array_Trailing_Edge_pressure}



```

```
#%% store coordinates in a numpy array called foil#_numpy


foil1_numpy = import_airfoil_coordinates(name_argument1="E3_hub_coordinates_sorted_TE_to_LE", name_argument2=".dat")
# foil1_numpy = import_airfoil_coordinates(name_argument1="tblade3_test_blade", name_argument2=".dat")
# foil1_numpy = import_airfoil_coordinates(name_argument1="uvblade.1.1", name_argument2="GE_E3_Fan_rotor")
foil1_numpy = foil1_numpy/np.amax(foil1_numpy[:,0])


test_foil1 = foil1_numpy

foil2_numpy = import_airfoil_coordinates(name_argument1="uvblade.1.1", name_argument2="GE_E3_Fan_rotor")
# foil2_numpy = import_airfoil_coordinates(name_argument1="uvblade.1.1_issue1_001", name_argument2="GE_E3_Fan_rotor")
# foil2_numpy = import_airfoil_coordinates(name_argument1="uvblade.1.1_issue1_002", name_argument2="GE_E3_Fan_rotor")
foil2_numpy = foil2_numpy/np.amax(foil2_numpy[:,0])


test_foil2 = foil2_numpy

#Set u values at ends of LE and TE
uLE = 0.20
uTE = 0.80

visual_debug = True
#%% Break target airfoil coordinates into six parts:

foil1_numpy = break_airfoil_into_six_parts(foil1_numpy, uLE, uTE, visual_debug=False)
foil2_numpy = break_airfoil_into_six_parts(foil2_numpy, uLE, uTE, visual_debug=False)
```

```python

#%% Make sure each airfoil part is a subsection of the other and calculate the differences between two foils:


"""
We start with removing points from plot digitized u coordinates part to make their u coordinates a subset of tblade3 part u coordinates.
Next, we will interpolate between plot digitized values using u coordinates from tblade3 files.
Ensuring that the plot digitized airfoil's u coordinates are a subset of tblade3 u coordiantes simplifies the difference calulation while retaining enough information from the plot digitized file.
"""


# Suction side
# Calculate difference for Suction side Trailing Edge part:
part='suction_TE'
foil1_numpy[part] = make_sure_x_subset_of_xx_2D_array(foil1_numpy[part], foil2_numpy[part], ascending=False)
foil2_numpy[part] = interpolate_section(foil1_numpy[part][:,0], foil2_numpy[part][:,0], foil2_numpy[part][:,1])
diff1 = (foil2_numpy[part][:,1]-foil1_numpy[part][:,1])**2

"""
We obtain a new set of foil1 u coordinates by making sure that they are a subset of foil2 u coordiantes.
Then we take the new foil1 u coodrinates and interpolate them around the foil2 coordinates such that they have the same u values but only differ by v values.
These steps are essential so that taking difference would make a lot of sense.
The "ascending" flag is detemined by the order of u coordinates from index 0.
"""

# # Calculate difference for Suction side mid part:
part='suction_mid'
```

```
foil1_numpy[part] = make_sure_x_subset_of_xx_2D_array(foil1_numpy[part], foil2_numpy[part], ascending=False)
foil2_numpy[part] = interpolate_section(foil1_numpy[part][:,0], foil2_numpy[part][:,0], foil2_numpy[part][:,1])
diff2 = (foil2_numpy[part][:,1]-foil1_numpy[part][:,1])**2

## Calculate difference for Suction side Leading Edge part:
part='suction_LE'
foil1_numpy[part] = make_sure_x_subset_of_xx_2D_array(foil1_numpy[part], foil2_numpy[part], ascending=False)
foil2_numpy[part] = interpolate_section(foil1_numpy[part][:,0], foil2_numpy[part][:,0], foil2_numpy[part][:,1])
diff3 = (foil2_numpy[part][:,1]-foil1_numpy[part][:,1])**2

## Calculate difference for Pressure side Leading Edge part:
part='pressure_LE'
foil1_numpy[part] = make_sure_x_subset_of_xx_2D_array(foil1_numpy[part], foil2_numpy[part], ascending=True)
foil2_numpy[part] = interpolate_section(foil1_numpy[part][:,0], foil2_numpy[part][:,0], foil2_numpy[part][:,1])
diff4 = (foil2_numpy[part][:,1]-foil1_numpy[part][:,1])**2

## Calculate difference for Pressure side mid part:
part='pressure_mid'
foil1_numpy[part] = make_sure_x_subset_of_xx_2D_array(foil1_numpy[part], foil2_numpy[part], ascending=True)
foil2_numpy[part] = interpolate_section(foil1_numpy[part][:,0], foil2_numpy[part][:,0], foil2_numpy[part][:,1])
diff5 = (foil2_numpy[part][:,1]-foil1_numpy[part][:,1])**2

## Calculate difference for Pressure side TE part:
part='pressure_TE'
foil1_numpy[part] = make_sure_x_subset_of_xx_2D_array(foil1_numpy[part], foil2_numpy[part], ascending=True)
foil2_numpy[part] = interpolate_section(foil1_numpy[part][:,0], foil2_numpy[part][:,0], foil2_numpy[part][:,1])
```

```python
diff6 = (foil2_numpy[part][:,1]-foil1_numpy[part][:,1])**2


## Now that we have all the differences, sum the arrays and then sum the sums to obtain total difference.
diff_total = (diff1.sum() + diff2.sum() + diff3.sum() + diff4.sum() + diff5.sum() + diff6.sum())
s = "Square distance = {}".format(diff_total)
print(s)
f = open("Sq_diff_out.dat", "w")
wr = np.array2string(diff_total)
f.write(wr)
f.close()

#%% Finally, stitch the new coordinates together to prepare for post processing


foil1_post_process = np.vstack((foil1_numpy['suction_TE'], foil1_numpy['suction_mid'], foil1_numpy['suction_LE'], foil1_numpy['pressure_LE'], foil1_numpy['pressure_mid'], foil1_numpy['pressure_TE']))
foil2_post_process = np.vstack((foil2_numpy['suction_TE'], foil2_numpy['suction_mid'], foil2_numpy['suction_LE'], foil2_numpy['pressure_LE'], foil2_numpy['pressure_mid'], foil2_numpy['pressure_TE']))

#%% Function to Plot and write to pdf

def plots_and_write_to_pdf():

    """
    This function creates the pdf to view and share results. The three page report contains:
      Page 1: A plot of the plot digitized airfoil.
      Page 2: A plot comparing the interpolated sections, highlighting the visual difference of the six parts
      Page 3: A plot with plot digitized airfoil superimposed with Tblade3 airfoil to diagnose and improve upon further iterations. This plot also shows the square
```

```
distance.
"""

with PdfPages('blades_difference_calculator_iter006.pdf') as pdf:

    ### Page 1: Plot digitized coordinates
    ## Define page attributes
    fig, ax1 = plt.subplots(2,10, figsize=(14,8.5), dpi=150)
    grid1 = plt.GridSpec(2, 10, wspace=0.9, hspace=0.4)
    fig.suptitle('Plot Digitized airfoil', fontsize=20)

    ## Define plots - suction side
    ## Suction side Trailing Edge:
    plt.subplot(grid1[0, 7:])
    part='suction_TE'
    plt.plot(foil1_numpy[part][:,0], foil1_numpy[part][:,1],'o',foil1_numpy[part][:,0],foil1_numpy[part][:,1],'-', color='orange')
    plt.ylim([-0.01, 0.31])
    # plt.gca().axes.get_yaxis().set_visible(False)
    plt.title('Suction side:\nTrailing Edge')

    ## Suction side Mid section:
    plt.subplot(grid1[0, 3:7])
    part='suction_mid'
    plt.plot(foil1_numpy[part][:,0], foil1_numpy[part][:,1],'o',foil1_numpy[part][:,0],foil1_numpy[part][:,1],'-', color='orange')
    plt.gca().axes.get_yaxis().set_visible(False)
    plt.ylim([-0.01, 0.31])
    plt.xlim([0.2,0.81])
    plt.title('Suction side:\nMid Part')

    ## Suction side Leading Edge:
    plt.subplot(grid1[0, :3])
    part='suction_LE'
    plt.plot(foil1_numpy[part][:,0], foil1_numpy[part][:,1],'o',foil1_numpy[part][:,0],foil1_numpy[part][:,1],'-', color='orange')
```

```
    plt.ylim([-0.01, 0.31])
    plt.title('Suction side:\nLeading Edge')

    ## Define plots - Pressure side
    ## Suction side Leading Edge:
    plt.subplot(grid1[1, :3])
    part='pressure_LE'
    plt.ylim([-0.1, 0.31])
    plt.plot(foil1_numpy[part][:,0], foil1_numpy[part][:,1],'o',foil1_numpy[part][:,0],foil1_numpy[part][:,1],'-', color='orange')
    plt.title('Pressure side:\nLeading Edge')

    ## Suction side Mid section:
    plt.subplot(grid1[1, 3:7])
    part='pressure_mid'
    plt.plot(foil1_numpy[part][:,0], foil1_numpy[part][:,1],'o',foil1_numpy[part][:,0],foil1_numpy[part][:,1],'-', color='orange')
    plt.gca().axes.get_yaxis().set_visible(False)
    plt.ylim([-0.1, 0.31])
    plt.xlim([0.2,0.81])
    plt.title('Pressure side:\nMid Part')

    ## Pressure side Trailing Edge:
    plt.subplot(grid1[1, 7:])
    part='pressure_TE'
    plt.plot(foil1_numpy[part][:,0], foil1_numpy[part][:,1],'o',foil1_numpy[part][:,0],foil1_numpy[part][:,1],'-', color='orange')
    plt.gca().axes.get_yaxis().set_visible(False)
    plt.ylim([-0.1, 0.31])
    plt.title('Pressure side:\nTrailing Edge')
    pdf.savefig()


    ### Page 2: Compare interpolated sections
    ## Define page attributes
    fig, ax2 = plt.subplots(2,10, figsize=(14,8.5), dpi=150)
```

```
    grid2 = plt.GridSpec(2, 10, wspace=0.9, hspace=0.4)
    fig.suptitle('Compare interpolated parts', fontsize=20)

    ## Define plots - suction side
    ## Suction side Trailing Edge:
    plt.subplot(grid2[0, 7:])
    part='suction_TE'
    plt.plot(foil1_numpy[part][:,0], foil1_numpy[part][:,1],'o', color='orange')
    plt.plot(foil2_numpy[part][:,0], foil2_numpy[part][:,1],'.', color='blue')
    plt.ylim([-0.01, 0.31])
    # plt.gca().axes.get_yaxis().set_visible(False)
    plt.title('Suction side:\nTrailing Edge')

    ## Suction side Mid section:
    plt.subplot(grid2[0, 3:7])
    part='suction_mid'
    plt.plot(foil1_numpy[part][:,0], foil1_numpy[part][:,1],'o', color='orange')
    plt.plot(foil2_numpy[part][:,0], foil2_numpy[part][:,1],'.', color='blue')
    plt.gca().axes.get_yaxis().set_visible(False)
    plt.ylim([-0.01, 0.31])
    plt.xlim([0.2,0.81])
    plt.title('Suction side:\nMid Part')

    ## Suction side Leading Edge:
    plt.subplot(grid2[0, :3])
    part='suction_LE'
    plt.plot(foil1_numpy[part][:,0], foil1_numpy[part][:,1],'o', color='orange')
    plt.plot(foil2_numpy[part][:,0], foil2_numpy[part][:,1],'.', color='blue')
    plt.ylim([-0.01, 0.31])
    plt.title('Suction side:\nLeading Edge')

    ## Define plots - Pressure side
    ## Suction side Leading Edge:
    plt.subplot(grid2[1, :3])
    part='pressure_LE'
    plt.ylim([-0.1, 0.31])
```

```python
    plt.plot(foil1_numpy[part][:,0], foil1_numpy[part][:,1],'o', color='orange')
    plt.plot(foil2_numpy[part][:,0], foil2_numpy[part][:,1],'.', color='blue')
    plt.title('Pressure side:\nLeading Edge')

    ## Suction side Mid section:
    plt.subplot(grid2[1, 3:7])
    part='pressure_mid'
    plt.plot(foil1_numpy[part][:,0], foil1_numpy[part][:,1],'o', color='orange')
    plt.plot(foil2_numpy[part][:,0], foil2_numpy[part][:,1],'.', color='blue')
    plt.gca().axes.get_yaxis().set_visible(False)
    plt.ylim([-0.1, 0.31])
    plt.xlim([0.2,0.81])
    plt.title('Pressure side:\nMid Part')

    ## Pressure side Trailing Edge:
    plt.subplot(grid2[1, 7:])
    part='pressure_TE'
    plt.plot(foil1_numpy[part][:,0], foil1_numpy[part][:,1],'o', color='orange')
    plt.plot(foil2_numpy[part][:,0], foil2_numpy[part][:,1],'.', color='blue')
    plt.gca().axes.get_yaxis().set_visible(False)
    plt.ylim([-0.1, 0.31])
    plt.title('Pressure side:\nTrailing Edge')
    pdf.savefig()

    ### Page 3: Compare two airfoils
    plt.figure(figsize=[14, 8.5], dpi=150)
    plt.plot(test_foil1[:,0], test_foil1[:,1], 'o', color='orange')
    plt.plot(test_foil2[:,0], test_foil2[:,1], '.', test_foil2[:,0], test_foil2[:,1], '-', color='blue')

    plt.text(0.7,0.4,s)
    plt.ylim([-0.51,0.51])
    plt.xlim([-0.01,1.01])
    plt.legend(['Target Airfoil','Tblade3 Airfoil'],loc='upper left')
    plt.title('Blade Comparison', fontsize=20)
    plt.savefig('Airfoil_fit.png', dpi=150, transparent=False)
```

```
        pdf.savefig()




#%% Write to pdf only if argument "plot" is given


plots_and_write_to_pdf()

# if len(sys.argv) > 1:
#     if (sys.argv[1] == 'plot'):
#         plots_and_write_to_pdf()
#         print("Saved figures to pdf.")
```

C.3 Blade Reverse Engineering - OpenMDAO script

This code carries out the optimization process using OpenMDAO as mentioned in section 2.2 and writes optimized values to 3dbgbinput and spancontrolinputs file.

```
# -*- coding: utf-8 -*-
"""

Use the solution from the first run to do second round of optimizaitons. SO that the cur values and LE edge values move.
Step size of 1e-12 did not work. Moving to step size of 1e-8. Was too fine for the optimizer. Moving on to Step size 1e-6. This bought down the difference of 20 percent t_max down to 1e-5 instead of the usual 1e-1. MOving this to version 003.
Also experimenting with 1e-4 step size: Nope. It did not go well and stopped until 1e-3.
1e-6 Yeyy!
1e-5:

Figure out the right step size by running single variables optimiztion for each variables. Start with t_max. FINished all the variables.
```

```
Figure out the optimial step sizes for each variable by running single variable optimization.



"""


import openmdao.api as om
import numpy as np


from openmdao.utils.file_wrap import InputFileGenerator
from openmdao.utils.file_wrap import FileParser

writeParser = InputFileGenerator()
readParser = FileParser()

# Next line makes sure that the numpy arrays all carry values of 16 decimal values instead of turncating at the usual 8.
## https://openmdao.org/newdocs/versions/latest/other_useful_docs/file_wrap.html
## https://stackoverflow.com/questions/12956333/printing-numpy-float64-with-full-precision
## https://numpy.org/doc/stable/reference/generated/numpy.set_printoptions.html

# np.set_printoptions(precision=16)

class BladeDesignOptimizerMDAO(om.ExternalCodeComp):

    def setup(self):

        #link inputs
        self.add_input('in_beta')
        self.add_input('out_beta')
        self.add_input('cur1')
        self.add_input('cur2')
```

```
        self.add_input('cur3')
        self.add_input('cur4')
        self.add_input('cur5')
        self.add_input('cur6')
        self.add_input('cur7')
        self.add_input('LE_radius')
        self.add_input('u_max')
        self.add_input('t_max')
        self.add_input('t_TE')


        #link outputs
        self.add_output('Sq_diff')

        #setup filenames
        self.tbladeinputfilebase = '3dbgbinput.1.base'
        self.spancontrolfilebase = 'spancontrolinputs.1.base'

        self.tbladeinputfile = '3dbgbinput.1.dat'
        self.spancontrolfile = 'spancontrolinputs.1.dat'


        self.Sq_diffoutfile = 'Sq_diff_out.dat'

        self.options['external_input_files'] = [self.tbladeinputfile, self.
spancontrolfile]
        self.options['external_output_files'] = [self.Sq_diffoutfile]

        #setup run command
        self.options['command'] = ["bash", "run_least_squares_002.sh"]

        # self.timeout = 10
        step_size_var = 1e-03 #-8
        #have openmdao calculate partialerivs
        self.declare_partials(of='*', wrt='in_beta', method='fd', step=step_size_var
)
```

```
        self.declare_partials(of='*', wrt='out_beta', method='fd', step=step_size_var)
        self.declare_partials(of='*', wrt='cur1', method='fd', step=step_size_var)
        self.declare_partials(of='*', wrt='cur2', method='fd', step=step_size_var)
        # self.declare_partials(of='*', wrt='cur3', method='fd', step=1e-08)
        self.declare_partials(of='*', wrt='cur4', method='fd', step=step_size_var)
        self.declare_partials(of='*', wrt='cur5', method='fd', step=step_size_var)
        self.declare_partials(of='*', wrt='cur6', method='fd', step=step_size_var)
        self.declare_partials(of='*', wrt='cur7', method='fd', step=step_size_var)
        self.declare_partials(of='*', wrt='LE_radius', method='fd', step=step_size_var)
        self.declare_partials(of='*', wrt='u_max', method='fd', step=1e-04)
        self.declare_partials(of='*', wrt='t_max', method='fd', step=1e-04)
        self.declare_partials(of='*', wrt='t_TE', method='fd', step=1e-05)


    def compute(self,inputs,outputs):


        in_beta = [float(inputs['in_beta'][0].astype(str))] # This code converts numpy output to 16 digit precision instead of the usual 9.
        out_beta = [float(inputs['out_beta'][0].astype(str))]
        cur1 = [float(inputs['cur1'][0].astype(str))]
        cur2 = [float(inputs['cur2'][0].astype(str))]
        # cur3 = inputs['cur3']
        cur4 = [float(inputs['cur4'][0].astype(str))]
        cur5 = [float(inputs['cur5'][0].astype(str))]
        cur6 = [float(inputs['cur6'][0].astype(str))]
        cur7 = [float(inputs['cur7'][0].astype(str))]
        LE_radius = [float(inputs['LE_radius'][0].astype(str))]
        u_max = [float(inputs['u_max'][0].astype(str))]
        t_max = [float(inputs['t_max'][0].astype(str))]
        t_TE = [float(inputs['t_TE'][0].astype(str))]
```

```
    #write output files

    #Tblade file
    writeParser.set_template_file(self.tbladeinputfilebase)
    writeParser.set_generated_file(self.tbladeinputfile)
    writeParser.reset_anchor()
    writeParser.mark_anchor("Sectionwise")
    writeParser.transfer_var(in_beta[0], 2, 2)
    writeParser.transfer_var(out_beta[0], 2, 3)
    writeParser.generate()



    #spancontrol file
    writeParser.set_template_file(self.spancontrolfilebase)
    writeParser.set_generated_file(self.spancontrolfile)
    writeParser.reset_anchor()
    writeParser.mark_anchor("Span control points, Chord & curv")
    writeParser.transfer_var("0.00", 3, 1)
    writeParser.transfer_var("0.15", 3, 2)
    writeParser.transfer_var("0.25", 3, 3)
    writeParser.transfer_var("0.50", 3, 4)
    writeParser.transfer_var("0.75", 3, 5)
    writeParser.transfer_var("0.95", 3, 6)
    writeParser.transfer_var(cur1[0], 3, 7)
    writeParser.transfer_var(cur2[0], 3, 8)
    writeParser.transfer_var("2.31", 3, 9)
    writeParser.transfer_var(cur4[0], 3, 10)
    writeParser.transfer_var("{}\t{}\t{}".format(cur5[0], cur6[0], cur7[0]), 3,
 11)
    # Move to second table
    writeParser.reset_anchor()
    writeParser.mark_anchor("Span control points, Chord & thick")
    writeParser.transfer_var("0.00\t{}".format(LE_radius[0]), 3, 1)
    writeParser.transfer_var(u_max[0], 3, 2)
    writeParser.transfer_var(t_max[0], 3, 3)
```

```
        writeParser.transfer_var(t_TE[0], 3, 4)
        writeParser.generate()


        #execute
        super(BladeDesignOptimizerMDAO, self).compute(inputs,outputs)

        #Parse:
        with open(self.Sq_diffoutfile, 'r') as output_file:
            Sq_diff = float(output_file.read())

        print("in_beta=", in_beta)
        print("out_beta=", out_beta)
        print("cur1=", cur1)
        print("cur2=", cur2)
        # print("cur3=",cur3)
        print("cur4=", cur4)
        print("cur5=", cur5)
        print("cur6=", cur6)
        print("cur7=", cur7)
        print("LE_radius=", LE_radius)
        print("u_max=", u_max)
        print("t_max=", t_max)
        print("t_TE=", t_TE)
        print("Sq_diff=",Sq_diff)
        print("End iteration.\n")

        outputs['Sq_diff']=Sq_diff



if __name__ == "__main__":
    prob = om.Problem()

    model = prob.model

```

```
    indeps = prob.model.add_subsystem('indeps', om.IndepVarComp())

    indeps.add_output('in_beta', 42.12)
    indeps.add_output('out_beta', 0.0)
    indeps.add_output('cur1', 1.909)
    indeps.add_output('cur2', 3.6695)
  # indeps.add_output('cur3', 2.31)
    indeps.add_output('cur4', 1.458)
    indeps.add_output('cur5', 0.80)
    indeps.add_output('cur6', 0.82)
    indeps.add_output('cur7', 0.81)
    indeps.add_output('LE_radius', 2.5)
    indeps.add_output('u_max', 0.55)
    indeps.add_output('t_max', 0.1146)
    indeps.add_output('t_TE', 0.01548)

    model.add_subsystem('runblade', BladeDesignOptimizerMDAO())

    prob.driver = om.ScipyOptimizeDriver()
    prob.driver.options['optimizer'] = 'SLSQP'
    prob.driver.options['maxiter'] = 200

    prob.model.connect('indeps.in_beta','runblade.in_beta')
    prob.model.connect('indeps.out_beta','runblade.out_beta')
    prob.model.connect('indeps.cur1', 'runblade.cur1')
    prob.model.connect('indeps.cur2', 'runblade.cur2')
      # prob.model.connect('indeps.cur3', 'runblade.cur3')
    prob.model.connect('indeps.cur4', 'runblade.cur4')
    prob.model.connect('indeps.cur5', 'runblade.cur5')
    prob.model.connect('indeps.cur6', 'runblade.cur6')
    prob.model.connect('indeps.cur7', 'runblade.cur7')
```

```
    prob.model.connect('indeps.LE_radius', 'runblade.LE_radius')
    prob.model.connect('indeps.u_max', 'runblade.u_max')
    prob.model.connect('indeps.t_max', 'runblade.t_max')
    prob.model.connect('indeps.t_TE', 'runblade.t_TE')


    prob.model.add_design_var('indeps.in_beta', lower=35.10, upper=55.50)
    prob.model.add_design_var('indeps.out_beta', lower=-30.00, upper=7.45)
    prob.model.add_design_var('indeps.cur1', lower=1.7100, upper=2.10)
    prob.model.add_design_var('indeps.cur2', lower=3.0695, upper=4.0695)
    # prob.model.add_design_var('indeps.cur3', lower=2.01, upper=2.91)
    prob.model.add_design_var('indeps.cur4', lower=0.615, upper=1.615)
    prob.model.add_design_var('indeps.cur5', lower=0.50, upper=1.50)
    prob.model.add_design_var('indeps.cur6', lower=0.50, upper=1.50)
    prob.model.add_design_var('indeps.cur7', lower=0.50, upper=1.50)
    prob.model.add_design_var('indeps.LE_radius',lower=2.00, upper=4.0)
    prob.model.add_design_var('indeps.u_max', lower=0.4, upper=0.7)
    prob.model.add_design_var('indeps.t_max', lower=0.08, upper=0.2)
    prob.model.add_design_var('indeps.t_TE', lower=0.001, upper=0.1)


    prob.model.add_objective('runblade.Sq_diff')

    prob.driver.options['tol'] = 1e-10

    prob.setup()

    # prob.run_model()
    prob.run_driver()

    print("Sq_diff=",prob['runblade.Sq_diff'])
```

C.4 Controller script

This bash shell script controls the operations to ensure logs are saved from each run to later diagnose optimization runs and plot progression of parameters.

```
#!/bin/bash


#clean:
rm tblade.log spancontrolinputs.1.log T-Blade3_run.log square_distance.log uvblade
    .1.1.GE_E3_Fan_rotor # Airfoil_fit.png

tblade3 3dbgbinput.1.dat dev > tblade.log
python3 'blades_difference_calculator_iter007.py' > square_distance.log

#gio open *iter012.pdf
```

C.5 Results post-processing file

Takes log files from controller script and creates tables and line plots with respect to iterations.

```
# -*- coding: utf-8 -*-
"""
This post processor code takes in output from our optimization runs and produces two
    plots:
    1. Line plot of objective function vs  iterations
    2. Two pages with line plots of 12 input variables v/s iterations
    3. Added the capability to plot by passing filename through terminal with syntax:
       python3 optimizatin_post_processor_002.py
    optimization_12_variables_Nelder_Mead_001_001.txt
    4. Add in capability to plot numbers in the scientific form because they appear
    below 1e-5.
    5. Add in a table in page 1 to show a table of the parameters.
    5. 001. Fixed issue with variable writing where iniitla number is omitted.

"""

import os
import sys
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages

```

```
#%% Functions to perform operations

#import optimization output

# print(len(sys.argv)) # Prints the number of arguments available
# print(sys.argv[1]) # prints the name of file that we will plot

name_argument=sys.argv[1]
# name_argument="optimization_12_variables_Nelder_Mead_001_001.txt"
# name_argument="optimization_12_variables_SimpleGADriver.txt"
# name_argument="optimization_12_variables_SLSQP.txt"

# Determine the name of the main T-Blade3 input files
current_dir                         = os.getcwd()
files                               = os.listdir(current_dir)
for file in files:
    if name_argument in file:
        input_file                  = file.strip()
        break

# Read main T-Blade3 input file and store the input in a list
f                                   = open(input_file, 'r')
lines                               = f.readlines()
f.close()

# Find the index number of the final "End Iteration" String:
line_index = [ii for ii, xx in enumerate(lines) if xx == "End iteration.\n"]

# Delete the excess data and calculate number of iterations
lines = lines[line_index[0]-13:line_index[-1]+1]
iterations = [ii+1 for ii, xx in enumerate(line_index)]


#%% Once we read the file, we next collect variable values in a dictionary.

# Collect values of variables:
```

```python

def collect_values_of_variable(lines, variable_limit=7, variable_extent=3, variable_name="in_beta"):

    values_of_variable = [xx for ii, xx in enumerate(lines) if xx[:variable_limit] == variable_name]
    if (variable_name=="Sq_diff"):
        values_of_variable = [float(xx[variable_limit+variable_extent:]) for ii, xx in enumerate(values_of_variable)]
    else:
        values_of_variable = [float(xx[variable_limit+variable_extent:-2]) for ii, xx in enumerate(values_of_variable)]

    return values_of_variable

def collect_vaiables_in_the_form_of_lists(line_index, lines):


    in_beta = collect_values_of_variable(lines, variable_name="in_beta", variable_limit=7)
    out_beta = collect_values_of_variable(lines, variable_name="out_beta", variable_limit=8)
    cur1 = collect_values_of_variable(lines, variable_name="cur1", variable_limit=4)
    cur2 = collect_values_of_variable(lines, variable_name="cur2", variable_limit=4)
    cur4 = collect_values_of_variable(lines, variable_name="cur4", variable_limit=4)
    cur5 = collect_values_of_variable(lines, variable_name="cur5", variable_limit=4)
    cur6 = collect_values_of_variable(lines, variable_name="cur6", variable_limit=4)
    cur7 = collect_values_of_variable(lines, variable_name="cur7", variable_limit=4)
    LE_radius = collect_values_of_variable(lines, variable_name="LE_radius", variable_limit=9)
    u_max = collect_values_of_variable(lines, variable_name="u_max", variable_limit=5)
    t_max = collect_values_of_variable(lines, variable_name="t_max", variable_limit=5)
    t_TE = collect_values_of_variable(lines, variable_name="t_TE", variable_limit=4)
```

```python
    sq_diff = collect_values_of_variable(lines, variable_name="Sq_diff", variable_limit=7, variable_extent=2)

    return { 'sq_diff': sq_diff, 'in_beta': in_beta, 'out_beta': out_beta, 'cur1': cur1, 'cur2': cur2, 'cur4': cur4, 'cur5': cur5, 'cur6': cur6, 'cur7': cur7, 'LE_radius': LE_radius, 'u_max': u_max, 't_max': t_max, 't_TE': t_TE}

parameters_to_plot = collect_vaiables_in_the_form_of_lists(line_index, lines)


#%% Next, we plot the variables.

def plot_variable(iterations, variable_name, iterable):
    plt.figure(figsize=(7,5), dpi=150)
    plt.plot(iterations, variable_name[iterable], '-', iterations, variable_name[iterable], '.', color='blue')
    if iterable=="sq_diff":
        plt.yscale("log")
        plt.title("Final Objective Function value: {}".format(variable_name['sq_diff'][-1]))
        plt.ylim(1e-6, 1e2)
        plt.ylabel("Objective value on log scale")
        plt.xlabel("Number of Iterations")
    else:
        plt.ylabel("value")
        plt.xlabel("Number of Iterations")
        plt.title("{}".format(iterable))


# for ii in parameters_to_plot:
#     # print(ii)
#     plot_variable(iterations, parameters_to_plot, ii)

def plot_to_pdf(iterations, parameters_to_plot, variable_name):

    if variable_name=="sq_diff":
```

```
        # plt.plot(iterations, parameters_to_plot[variable_name], '-', color='blue')
        plt.plot(iterations, parameters_to_plot[variable_name], '.', color='blue')
        plt.yscale("log")
        plt.title("Final Objective Function value using {} method: {}".format(name_argument[26:-4], parameters_to_plot['sq_diff'][-1]))
        plt.ylim(1e-6, 1e2)
        plt.ylabel("Objective value on log scale")
        plt.xlabel("Number of Iterations")
    else:
        # plt.plot(iterations, parameters_to_plot[variable_name], '-', color='orange')
        plt.plot(iterations, parameters_to_plot[variable_name], '.', color='orange')
        plt.title("Final {} value: {}".format(variable_name, parameters_to_plot[variable_name][-1]), y=1.11, fontsize=13)

rows_to_write = [['', 'in_beta', 'out_beta', 'cur1', 'cur2', 'cur4', 'cur5', 'cur6', 'cur7', 'LE_radius', 'u_max', 't_max', 't_TE'], [parameters_to_plot[xx][0] for xx in parameters_to_plot], [parameters_to_plot[xx][-1] for xx in parameters_to_plot]]#, ['Known target Blade values', '42.12', '0.0', '1.909', '3.6695', '1.215', '0.8', '0.82', '0.81', '2.5', '0.55', '0.0955', '0.01548']]

rows_to_write[1][0] = 'Initial input values'
rows_to_write[2][0] = 'Final input values'

rows_to_write1 = list(zip(*rows_to_write))


def plots_and_write_to_pdf():

    """
    This function creates the pdf to view and share results. The three page report contains:
        Page 1: A plot of the plot digitized airfoil.
        Page 2: A plot comparing the interpolated sections, highlighting the visual difference of the six parts
        Page 3: A plot with plot digitized airfoil superimposed with Tblade3 airfoil to diagnose and improve upon further iterations. This plot also shows the square
```

```
    distance.
    """


with PdfPages('{}.pdf'.format(name_argument)) as pdf:

    ### Page 0: Add in parameter info from initial and final properties:
    fig, ax0 = plt.subplots(figsize=(8.5, 11), dpi=150)
    # hide axes
    ax0.axis('off')
    ax0.axis('tight')
    # grid0 = plt.GridSpec(12, 2, wspace=0.3, hspace=10.2)
    fig.suptitle('Variables for Optimization', fontsize=20)
    # fig.text(0.5,0.5,"Initial t_max={}".format(parameters_to_plot['t_max'][0]), transform=fig.transFigure, size=24, ha="center")
    # fig.text(0.5,0.3,"Final t_max={}".format(parameters_to_plot['t_max'][-1]), transform=fig.transFigure, size=24, ha="center")
    # fig.text(0.5,0.2 ,"Target t_max=0.0955", transform=fig.transFigure, size=24, ha="center")
    the_table = ax0.table(cellText=rows_to_write1, colWidths=[0.28]*4, loc='upper center')
    the_table.set_fontsize(54)
    pdf.savefig()


    ### Page 1: Plot digitized coordinates
    ## Define page attributes
    fig, ax1 = plt.subplots(12, 2, figsize=(8.5, 11), dpi=150)
    grid1 = plt.GridSpec(12, 2, wspace=0.3, hspace=10.2)
    fig.suptitle('Variables for Optimization', fontsize=20)

    using_SLSQP=False

    ## Define plots - First 6 variables
    ## in_beta:
    plt.subplot(grid1[:3, 0])
```

```python
    plt.gca().axes.get_xaxis().set_visible(False)
    plot_to_pdf(iterations, parameters_to_plot, variable_name='in_beta')

    ## out_beta:
    plt.subplot(grid1[3:6, 0])
    plt.gca().axes.get_xaxis().set_visible(False)
    plot_to_pdf(iterations, parameters_to_plot, variable_name='out_beta')

    ## LE_radius:
    plt.subplot(grid1[6:9, 0])
    plt.gca().axes.get_xaxis().set_visible(False)
    plot_to_pdf(iterations, parameters_to_plot, variable_name='LE_radius')

    plt.subplot(grid1[9:, 0])
    plt.xlabel("Number of Iterations")
    plot_to_pdf(iterations, parameters_to_plot, variable_name='u_max')

    ## u_max:
    plt.subplot(grid1[:3, 1])
    plt.gca().axes.get_xaxis().set_visible(False)
    plot_to_pdf(iterations, parameters_to_plot, variable_name='t_max')

    ## t_max:
    plt.subplot(grid1[3:6, 1])
    plt.gca().axes.get_xaxis().set_visible(False)
    plot_to_pdf(iterations, parameters_to_plot, variable_name='t_TE')

    ## t_max:
    plt.subplot(grid1[6:9, 1])
    plt.gca().axes.get_xaxis().set_visible(False)
    if (using_SLSQP==True):
        plt.ylim([parameters_to_plot['cur1'][0]-1e-2, parameters_to_plot['cur1'][0]+1e
    -2])
    plot_to_pdf(iterations, parameters_to_plot, variable_name='cur1')

    ## t_TE:
```

```
plt.subplot(grid1[9:, 1])
plt.xlabel("Number of Iterations")
if (using_SLSQP==True):
    plt.ylim([parameters_to_plot['cur2'][0]-1e-2, parameters_to_plot['cur2'][0]+1e-2])
plot_to_pdf(iterations, parameters_to_plot, variable_name='cur2')
pdf.savefig()
plt.close()

### Page 2: Plot digitized coordinates
## Define page attributes
fig, ax1 = plt.subplots(12, 2, figsize=(8.5, 11), dpi=150)
grid1 = plt.GridSpec(12, 2, wspace=0.3, hspace=10.2)
fig.suptitle('Variables and Objective Function plots', fontsize=20)

## Define plots - First 6 variables
## in_beta:
plt.subplot(grid1[:3, 0])
if (using_SLSQP==True):
    plt.ylim([parameters_to_plot['cur4'][0]-1e-3, parameters_to_plot['cur4'][0]+1e-3])
plt.gca().axes.get_xaxis().set_visible(False)

plot_to_pdf(iterations, parameters_to_plot, variable_name='cur4')

## out_beta:
plt.subplot(grid1[3:6, 0])
if (using_SLSQP==True):
    plt.ylim([parameters_to_plot['cur5'][0]-1e-3, parameters_to_plot['cur5'][0]+1e-3])
plt.xlabel("Number of Iterations")
plot_to_pdf(iterations, parameters_to_plot, variable_name='cur5')

## LE_radius:
plt.subplot(grid1[:3, 1])
if (using_SLSQP==True):
```

```
        plt.ylim([parameters_to_plot['cur6'][0]-1e-3, parameters_to_plot['cur6'][0]+1e-3])
    plt.gca().axes.get_xaxis().set_visible(False)
    plot_to_pdf(iterations, parameters_to_plot, variable_name='cur6')

    plt.subplot(grid1[3:6, 1])
    if (using_SLSQP==True):
        plt.ylim([parameters_to_plot['cur7'][0]-1e-3, parameters_to_plot['cur7'][0]+1e-3])
    plt.xlabel("Number of Iterations")
    plot_to_pdf(iterations, parameters_to_plot, variable_name='cur7')

    ## sq_diff:
    plt.subplot(grid1[6:, :])
    plot_to_pdf(iterations, parameters_to_plot, variable_name='sq_diff')
    pdf.savefig()
    plt.close()
```

Appendix D

Codes for FINE/Turbo input files

D.1 Input profile generator

This code generates input file for FINE/Turbo to ensure that the inflow is parallel to the hub.

```
import numpy as np
import math


col_31 = np.asarray([0.0]*18)
col = list(map(str, col_31))
col = [ii[:-2] for ii in col]

def profile_plot(physical_parameter, radius):

    physical_parameter = physical_parameter.copy()
    col_3 = np.asarray([0]*len(physical_parameter))
    col_3 = list(map(int, col_3))
    #map(int, col_3)


    physical_parameter_plot = np.stack((radius, physical_parameter, col_3), axis=1)
    physical_parameter_plot = np.round(physical_parameter_plot, 6)
    physical_parameter_plot1 = [" ".join(item) for item in physical_parameter_plot.astype(str)]

```

```python
    physical_parameter_export = line1.copy()
    physical_parameter_export.append(line2[0])
    [physical_parameter_export.append(ii) for ii in physical_parameter_plot1]
    physical_parameter_export = [ii[:-2] for ii in physical_parameter_export]
    physical_parameter_export[0] = line1[0]
    physical_parameter_export[1] = line2[0]
    return physical_parameter_export

def generate_FINE_Turbo_input_file(file_path, physical_parameter_export):



    with open(str(file_path), "w") as outfile:
        for ii in physical_parameter_export:
            outfile.write(ii)
            outfile.write("\n")

    # Essential to replace Windows file ending with Linux ones:

    # replacement strings
    WINDOWS_LINE_ENDING = b'\r\n'
    UNIX_LINE_ENDING = b'\n'

    with open(file_path, 'rb') as open_file:
        content = open_file.read()

    content = content.replace(WINDOWS_LINE_ENDING, UNIX_LINE_ENDING)

    with open(file_path, 'wb') as open_file:
        open_file.write(content)



line1 = ['FINE profile file']
line2 = ['4 18 ']

```

```
map(str, line1)
map(str, line2)


#Hub and shroud coordinates: import these from .mf file
hub_radius = 0.18884 # default = 0.23640
shroud_radius = 1.0517 # default = 1.0517

delta_x = (-0.152382135 - -0.284952849)
delta_y = (0.272952139 - 0.188841641)
phi_hub = math.degrees(math.atan(delta_y / delta_x))
slope = delta_y/delta_x
phi_hub = 29.0000

phi = np.linspace(0, phi_hub, 18)
radius = np.linspace(shroud_radius, hub_radius, 18)

# Vz_by_Vm = np.cos(np.radians(phi))
# Vr_by_Vm = np.sin(np.radians(phi))
arctan_Vr_by_Vz = np.radians(phi)

# relative or absolute file path, e.g.:
# file_path = "Vr_by_Vm_generated.p"
# Vr_by_Vm_export = profile_plot(Vr_by_Vm, radius)
# generate_FINE_Turbo_input_file(file_path, Vr_by_Vm_export)


# relative or absolute file path, e.g.:
# file_path = "Vz_by_Vm_generated.p"
# Vz_by_Vm_export = profile_plot(Vz_by_Vm, radius)
# generate_FINE_Turbo_input_file(file_path, Vz_by_Vm_export)



# tan inverse of Vr_by_Vz in radians:
file_path = "arctan_Vr_by_Vz_generated_04232021_0018.p"
```

```
arctan_Vr_by_Vz_export = profile_plot(arctan_Vr_by_Vz, radius)
generate_FINE_Turbo_input_file(file_path, arctan_Vr_by_Vz_export)
```

D.2 Profile plots generator

This code plots profile plots of span vs dimension from FINE/Turbo data.

```
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.backends.backend_pdf import PdfPages


stations = ['rotor_inlet.csv']
values = ('P_total (kPa)','Abs_Angle (deg)','Rel_Angle (deg)','rv_theta (m^2/sec)','Phi (deg)','P_static (kPa)','Vr (m/s)','Vt (m/s)','Vz (m/s)', 'Entropy ', 'TT (K)', 'M_abs', 'M_rel', 'beta_z (deg)')


with PdfPages('profileplots_rotor_inlet_E3_GE_wo_pss.pdf') as pdf:

    for stname in stations:
        read_begin = pd.read_csv(stname)
        for ii in range(14):
            plt.figure()
            plt.plot(read_begin[values[ii]], read_begin['Span'])
            plt.title('At {}'.format(stname))
            plt.xlabel('{}'.format(values[ii]))
            plt.ylabel('Span')
            pdf.savefig() # saves the current figure into a pdf page
            plt.close()
```

Appendix E

Input files for cases

E.1 Traditional manual parameter specified Geometry's Tblade3 input files

E.1.1 3dbgbinput.1.dat file

```
Input parameters (version 1.31_TEST)
GE_E3_Fan_rotor
Blade row #:
         1
Number of blades in this row:
        32
Blade Scaling factor (mm):
  10.000000000000000
Number of streamlines:
        18
Angles in the input file (0=Beta_z (default),1=Beta_r):
          0   inci_dev_spline
Airfoil camber defined by curvature control (0=no,1=yes ;spanwise_spline=
   spancontrolinputs file):
          1   spanwise_spline
Airfoil Thickness distribution (0=Wennerstrom,1=Spline,2=Spline + sharp TE):
         5
Airfoil Thickness multiplier (0=no,1=yes):
```

```
            0
Airfoil LE defined by spline (0=no,1=yes):
            0
Non-dimensional Actual chord (0=no,1=yes,2=spline):
            2
True lean and sweep (0=no,1=yes):
            0
Clustering control (0=uniform,1=sine,2=exponential,3=hyperbolic,4=elliptical):
                 4   15.0
Sectionwise properties:
J          in_Beta          out_Beta            mrel_in              chord               t/c_max
              Incidence          Deviation         Sec. Flow Angle
1            42.12            0.00            0.00000000            0.00000000            0.000000
              3.50             12.63            0.00000000
2            43.20            11.90             0.00000000            0.00000000
  0.000000            5.25             9.58             0.00000000
3            43.79            17.92             0.00000000            0.00000000
  0.000000            4.80             7.99             0.00000000
4            44.38            20.62             0.00000000            0.00000000
  0.000000            4.72             7.48             0.00000000
5            45.50            25.61             0.00000000            0.00000000
  0.000000            4.65             6.86             0.00000000
6            46.53            29.62             0.00000000            0.00000000
  0.000000            4.68             6.46             0.00000000
7            47.22            30.91             0.00000000            0.00000000
  0.000000            4.76             6.27             0.00000000
8            48.66            32.40             0.00000000            0.00000000
  0.000000            5.00             5.90             0.00000000
9            49.83            34.45             0.00000000            0.00000000
  0.000000            5.55             5.45             0.00000000
10           50.55            35.72             0.00000000            0.00000000
  0.000000            5.65             5.26             0.00000000
11           51.41            37.30             0.00000000            0.00000000
  0.000000            5.68             5.05             0.00000000
12           51.89            38.13             0.00000000            0.00000000
  0.000000            5.73             4.90             0.00000000
```

```
13          53.09         40.20         0.00000000         0.00000000
    0.000000          5.70          4.56          0.00000000
14          54.86         43.56         0.00000000         0.00000000
    0.000000          5.66          3.93          0.00000000
15          56.63         46.71         0.00000000         0.00000000
    0.000000          5.50          3.43          0.00000000
16          58.56         49.69         0.00000000         0.00000000
    0.000000          4.95          3.22          0.00000000
17          61.00         52.81         0.00000000         0.00000000
    0.000000          4.66          2.96          0.00000000
18          63.27         55.32         0.00000000         0.00000000
    0.000000          5.00          2.48          0.00000000

 LE / TE curve (x,r) definition :
 Number of Curve points :
           18
   xLE  rLE     xTE     rTE
   -1.270       36.068  17.789  47.884
   -1.366       43.785  17.884  52.947
   -1.429       49.488  17.981  56.958
   -1.431       52.313  17.892  59.044
   -1.422       57.049  17.85   62.754
   -1.441       60.613  17.8    65.725
   -1.449       62.683  17.782  67.491
   -1.42        66.362  17.783  70.588
   -1.364       69.829  17.793  73.453
   -1.306       71.824  17.767  75.088
   -1.237       74.234  17.699  77.273
   -1.213       75.639  17.657  78.412
   -1.173       79.254  17.546  81.34
   -0.988       84.922  17.35   85.915
   -0.707       90.238  17.062  90.211
   -0.347       95.26   16.674  94.355
   0.050        100.11  16.305  98.424
   0.467        105.167 15.981  102.995
```

```
# Airfoil type and Variable Radial Stacking information.          #
# stack_u: % chord stack (0.00 to 100.00).                        #
# stack_v: % below or above meanline stack (-100.00 to +100.00). #
# Use +200 for stacking on airfoil area centroid.                 #
Variable Radial stacking (0=no,1=yes):
           0
J   type |stk_u |stk_v |umxthk |lethk |tethk  |Jcells(Grid:4n+1) |eta_ofst(<=10)[thkc/Jmax]  |BGgrid(0=no,1=yes) |
1    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
2    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
3    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
4    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
5    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
6    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
7    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
8    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
9    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
10   sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
11   sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
12   sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
13   sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
14   sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
15   sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
16   sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
17   sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
18   sect1  25.00   0.00   0.30   0.02   0.02  15  10   0

Control table for blending section variable:
            4            0            0
       span                        bf1          bf2
   0.0000000000000000               1            0
  0.33333333333333331               1            0
  0.66666666666666663               1            0
   1.0000000000000000               1            0

Stacking axis location (200=centroid):
```

```
      55000

Control points for sweep:
           4
       span                    delta_m
  0.0000000000000000       0.0000000000000000
 0.25000000000000000       0.0000000000000000
 0.50000000000000000       0.0000000000000000
  1.0000000000000000       0.0000000000000000

Control points for lean:
           4
       span                     delta_theta
  0.0000000000000000       0.0000000000000000
 0.25000000000000000       0.0000000000000000
 0.50000000000000000       0.0000000000000000
  1.0000000000000000       0.0000000000000000

Control points for in_beta*:
           5
       span                      in_beta*
  0.0000000000000000       0.0000000000000000
  0.2500000000000000       0.0000000000000000
 0.50000000000000000       0.0000000000000000
 0.75000000000000000       0.0000000000000000
 1.00000000000000000       0.0000000000000000

Control points for out_beta*:
           5
       span                      out_beta*
  0.0000000000000000       0.0000000000000000
  0.2500000000000000       0.0000000000000000
 0.50000000000000000       0.0000000000000000
 0.75000000000000000       0.0000000000000000
 1.00000000000000000       0.0000000000000000
```

```
Control points for chord_multiplier:
          4
       span                  chord_multiplier
   0.0000000000000000         0.0000000000000000
  0.25000000000000000         0.0000000000000000
  0.50000000000000000         0.0000000000000000
   1.0000000000000000         0.0000000000000000

Control points for tm/c:
          4
       span                            tm/c
   0.0000000000000000         0.0000000000000000
  0.25000000000000000         0.0000000000000000
  0.50000000000000000         0.0000000000000000
   1.0000000000000000         0.0000000000000000

Hub offset
   0.0000000000000000
Tip offset
   0.0000000000000000

 Streamline Data
x_s       r_s
-28.495285        18.884164
-15.238214        27.295215
-9.472585         30.949586
-6.590009         32.77715
-1.27    36.068
1.35768  37.855728
5.446081          40.393893
9.534482          42.852843
14.20694          45.649056
17.789   47.884
19.492925         49.019945
25.425061         52.353179
31.961715         54.306013
```

```
38.62798	55.772693
45.39551	56.603429
52.137207	55.696729
57.971122	52.232159
67.62676	43.577681
0.0	0.0
-28.495285	21.208355
-15.238214	29.619406
-9.472585	33.273777
-6.590009	35.101341
-1.27	38.068
1.35768	39.855728
5.446081	42.393893
9.534482	44.852843
14.20694	47.649056
17.789	49.884
19.492925	51.319945
25.425061	54.653179
31.961715	56.606013
38.62798	58.072693
45.39551	58.903429
52.137207	57.996729
57.971122	54.532159
67.62676	45.877681
0.0	0.0
-28.495285	28.808422
-15.238214	36.280124
-9.472585	39.526374
-6.590009	41.149834
-1.366	43.785
1.35768	45.339242
5.446081	47.534739
9.534482	48.452355
14.20694	50.936022
17.884	52.947
19.492925	54.248797
```

```
25.425061	57.275809
31.961715	59.049238
38.62798	60.381176
45.39551	61.135593
52.137207	60.31219
57.971122	57.165907
67.62676	49.306509
0.0	0.0
−28.495285	35.903042
−15.238214	42.680548
−9.472585	45.625189
−6.590009	47.097814
−1.429	49.488
1.35768	50.869692
5.446081	52.81195
9.534482	52.888389
14.20694	55.124451
17.981	56.958
19.492925	58.153522
25.425061	60.937941
31.961715	62.569241
38.62798	63.794433
45.39551	64.488389
52.137207	63.730976
57.971122	60.836846
67.62676	53.607324
0.0	0.0
−28.495285	39.417386
−15.238214	45.851019
−9.472585	48.646257
−6.590009	50.044165
−1.431	52.313
1.35768	53.609219
5.446081	55.426035
9.534482	55.195437
14.20694	57.302727
```

```
17.892	59.044
19.492925	60.184252
25.425061	62.842504
31.961715	64.399888
38.62798	65.569566
45.39551	66.232077
52.137207	65.508984
57.971122	62.745991
67.62676	55.844048
0.0	0.0
−28.495285	45.309042
−15.238214	51.166188
−9.472585	53.710958
−6.590009	54.983606
−1.422	57.049
1.35768	58.201926
5.446081	59.808443
9.534482	59.298575
14.20694	61.176842
17.85	62.754
19.492925	63.795952
25.425061	66.229816
31.961715	67.655738
38.62798	68.72668
45.39551	69.333267
52.137207	68.671212
57.971122	66.141449
67.62676	59.822115
0.0	0.0
−28.495285	49.742713
−15.238214	55.166033
−9.472585	57.522316
−6.590009	58.700702
−1.441	60.613
1.35768	61.658094
5.446081	63.106353
```

```
9.534482		62.584404
14.20694		64.279266
17.8	65.725
19.492925		66.688233
25.425061		68.942404
31.961715		70.26305
38.62798		71.254924
45.39551		71.816727
52.137207		71.203551
57.971122		68.860562
67.62676		63.007785
0.0	0.0
-28.495285		52.317825
-15.238214		57.489174
-9.472585		59.735984
-6.590009		60.859622
-1.449	62.683
1.35768	63.665464
5.446081		65.021806
9.534482		64.537542
14.20694		66.123387
17.782	67.491
19.492925		68.407441
25.425061		70.554801
31.961715		71.812869
38.62798		72.757745
45.39551		73.292927
52.137207		72.708806
57.971122		70.476837
67.62676		64.901387
0.0	0.0
-28.495285		61.207559
-15.238214		65.509064
-9.472585		67.37795
-6.590009		68.312587
-1.364	69.829
```

```
1.35768 70.595254
5.446081        71.634283
9.534482        71.131318
14.20694        72.34912
17.793  73.453
19.492925       74.211473
25.425061       75.998238
31.961715       77.045046
38.62798        77.831253
45.39551        78.276565
52.137207       77.790532
57.971122       75.933366
67.62676        71.294172
0.0     0.0
-28.495285      66.687447
-15.238214      70.452755
-9.472585       72.088678
-6.590009       72.906809
-1.237  74.234
1.35768 74.866976
5.446081        75.710404
9.534482        75.356112
14.20694        76.338101
17.699  77.273
19.492925       77.930259
25.425061       79.485982
31.961715       80.39743
38.62798        81.081974
45.39551        81.469704
52.137207       81.046519
57.971122       79.429498
67.62676        75.390187
0.0     0.0
-28.495285      68.435289
-15.238214      72.029574
-9.472585       73.591192
```

```
−6.590009    74.372163
−1.213  75.639
1.35768 76.229467
5.446081     77.010506
9.534482     76.615809
14.20694     77.527485
17.657  78.412
19.492925    79.03908
25.425061    80.525914
31.961715    81.397002
38.62798     82.051234
45.39551     82.421795
52.137207    82.017349
57.971122    80.471932
67.62676     76.611486
0.0     0.0
−28.495285   72.932404
−15.238214   76.086655
−9.472585    77.45709
−6.590009    78.14245
−1.173  79.254
1.35768 79.735091
5.446081     80.355609
9.534482     79.854081
14.20694     80.585008
17.546  81.34
19.492925    81.8895
25.425061    83.199242
31.961715    83.966578
38.62798     84.542887
45.39551     84.869311
52.137207    84.513037
57.971122    83.15169
67.62676     79.751049
0.0     0.0
−28.495285   79.983484
```

```
−15.238214    82.447799
−9.472585     83.518476
−6.590009     84.053926
−0.988   84.922
1.35768  85.2316
5.446081      85.600433
9.534482      84.913881
14.20694      85.362386
17.35    85.915
19.492925     86.343282
25.425061     87.376318
31.961715     87.98154
38.62798      88.436094
45.39551      88.693555
52.137207     88.41255
57.971122     87.338811
67.62676      84.656616
0.0      0.0
−28.495285    86.596669
−15.238214    88.413896
−9.472585     89.203432
−6.590009     89.598281
−0.707   90.238
1.35768  90.386759
5.446081      90.519539
9.534482      89.665116
14.20694      89.848423
17.062   90.211
19.492925     90.525456
25.425061     91.29866
31.961715     91.751656
38.62798      92.091879
45.39551      92.284583
52.137207     92.074257
57.971122     91.270588
67.62676      89.263024
```

```
0.0 0.0
−28.495285 92.844115
−15.238214 94.050041
−9.472585 94.573982
−6.590009 94.836008
−0.347 95.26
1.35768 95.256813
5.446081 95.166594
9.534482 94.248244
14.20694 94.175736
16.674 94.355
19.492925 94.559657
25.425061 95.082223
31.961715 95.388378
38.62798 95.618316
45.39551 95.748554
52.137207 95.606406
57.971122 95.063251
67.62676 93.706449
0.0 0.0
−28.495285 98.877589
−15.238214 99.493151
−9.472585 99.760595
−6.590009 99.894346
0.05 100.11
1.35768 99.960072
5.446081 99.654491
9.534482 98.748424
14.20694 98.424732
16.305 98.424
19.492925 98.520845
25.425061 98.79731
31.961715 98.959281
38.62798 99.08093
45.39551 99.149833
52.137207 99.074629
```

```
57.971122     98.787272
67.62676      98.069455
0.0     0.0
−28.495285    103.167
−15.238214    103.167
−9.472585     103.167
−6.590009     103.167
0.467   103.167
3.5698  102.7326
6.6726  102.2982
9.7754  101.8638
12.8782 101.4294
15.981  100.995
19.492925     100.995
25.425061     100.995
31.961715     100.995
38.62798      100.995
45.39551      100.995
52.137207     100.995
57.971122     100.995
67.62676      100.995
0.0     0.0
−28.495285    105.167
−28.1286738   105.167
−27.7620626   105.167
−27.3954514   105.167
−27.0288402   105.167
−26.662229    105.167
−26.2956178   105.167
−25.9290066   105.167
−25.5623954   105.167
−25.1957842   105.167
−24.829173    105.167
−24.4625618   105.167
−24.0959506   105.167
−23.7293394   105.167
```

```
−23.3627282     105.167
−22.996117      105.167
−22.6295058     105.167
−22.2628946     105.167
−21.8962834     105.167
−21.5296722     105.167
−21.1630609     105.167
−20.7964497     105.167
−20.4298385     105.167
−20.0632273     105.167
−19.6966161     105.167
−19.3300049     105.167
−18.9633937     105.167
−18.5967825     105.167
−18.2301713     105.167
−17.8635601     105.167
−17.4969489     105.167
−17.1303377     105.167
−16.7637265     105.167
−16.3971153     105.167
−16.0305041     105.167
−15.6638929     105.167
−15.2972817     105.167
−14.9306705     105.167
−14.5640593     105.167
−14.1974481     105.167
−13.8308369     105.167
−13.4642257     105.167
−13.0976145     105.167
−12.7310033     105.167
−12.3643921     105.167
−11.9977809     105.167
−11.6311697     105.167
−11.2645585     105.167
−10.8979473     105.167
−10.5313361     105.167
```

```
-10.1647249      105.167
-9.7981137       105.167
-9.4315025       105.167
-9.0648913       105.167
-8.6982801       105.167
-8.3316689       105.167
-7.9650577       105.167
-7.5984465       105.167
-7.2318353       105.167
-6.8652241       105.167
-6.4986128       105.167
-6.1320016       105.167
-5.7653904       105.167
-5.3987792       105.167
-5.032168        105.167
-4.6655568       105.167
-4.2989456       105.167
-3.9323344       105.167
-3.5657232       105.167
-3.199112        105.167
-2.8325008       105.167
-2.4658896       105.167
-2.0992784       105.167
-1.7326672       105.167
-1.366056        105.167
-0.9994448       105.167
-0.6328336       105.167
-0.2662224       105.167
0.1003888        105.167
0.467    105.167
0.6561951        105.1405122
0.8453902        105.1140244
1.0345854        105.0875366
1.2237805        105.0610488
1.4129756        105.034561
1.6021707        105.0080732
```

```
1.7913659	104.9815854
1.980561	104.9550976
2.1697561	104.9286098
2.3589512	104.902122
2.5481463	104.8756341
2.7373415	104.8491463
2.9265366	104.8226585
3.1157317	104.7961707
3.3049268	104.7696829
3.494122	104.7431951
3.6833171	104.7167073
3.8725122	104.6902195
4.0617073	104.6637317
4.2509024	104.6372439
4.4400976	104.6107561
4.6292927	104.5842683
4.8184878	104.5577805
5.0076829	104.5312927
5.196878	104.5048049
5.3860732	104.4783171
5.5752683	104.4518293
5.7644634	104.4253415
5.9536585	104.3988537
6.1428537	104.3723659
6.3320488	104.345878
6.5212439	104.3193902
6.710439	104.2929024
6.8996341	104.2664146
7.0888293	104.2399268
7.2780244	104.213439
7.4672195	104.1869512
7.6564146	104.1604634
7.8456098	104.1339756
8.0348049	104.1074878
8.224	104.081
8.4131951	104.0545122
```

8.6023902	104.0280244
8.7915854	104.0015366
8.9807805	103.9750488
9.1699756	103.948561
9.3591707	103.9220732
9.5483659	103.8955854
9.737561	103.8690976
9.9267561	103.8426098
10.1159512	103.816122
10.3051463	103.7896341
10.4943415	103.7631463
10.6835366	103.7366585
10.8727317	103.7101707
11.0619268	103.6836829
11.251122	103.6571951
11.4403171	103.6307073
11.6295122	103.6042195
11.8187073	103.5777317
12.0079024	103.5512439
12.1970976	103.5247561
12.3862927	103.4982683
12.5754878	103.4717805
12.7646829	103.4452927
12.953878	103.4188049
13.1430732	103.3923171
13.3322683	103.3658293
13.5214634	103.3393415
13.7106585	103.3128537
13.8998537	103.2863659
14.0890488	103.259878
14.2782439	103.2333902
14.467439	103.2069024
14.6566341	103.1804146
14.8458293	103.1539268
15.0350244	103.127439
15.2242195	103.1009512

```
15.4134146      103.0744634
15.6026098      103.0479756
15.7918049      103.0214878
15.981  102.995
16.6347438      102.995
17.2884876      102.995
17.9422314      102.995
18.5959752      102.995
19.249719       102.995
19.9034628      102.995
20.5572066      102.995
21.2109504      102.995
21.8646942      102.995
22.518438       102.995
23.1721818      102.995
23.8259256      102.995
24.4796694      102.995
25.1334132      102.995
25.787157       102.995
26.4409008      102.995
27.0946446      102.995
27.7483884      102.995
28.4021322      102.995
29.0558759      102.995
29.7096197      102.995
30.3633635      102.995
31.0171073      102.995
31.6708511      102.995
32.3245949      102.995
32.9783387      102.995
33.6320825      102.995
34.2858263      102.995
34.9395701      102.995
35.5933139      102.995
36.2470577      102.995
36.9008015      102.995
```

37.5545453	102.995
38.2082891	102.995
38.8620329	102.995
39.5157767	102.995
40.1695205	102.995
40.8232643	102.995
41.4770081	102.995
42.1307519	102.995
42.7844957	102.995
43.4382395	102.995
44.0919833	102.995
44.7457271	102.995
45.3994709	102.995
46.0532147	102.995
46.7069585	102.995
47.3607023	102.995
48.0144461	102.995
48.6681899	102.995
49.3219337	102.995
49.9756775	102.995
50.6294213	102.995
51.2831651	102.995
51.9369089	102.995
52.5906527	102.995
53.2443965	102.995
53.8981403	102.995
54.5518841	102.995
55.2056278	102.995
55.8593716	102.995
56.5131154	102.995
57.1668592	102.995
57.820603	102.995
58.4743468	102.995
59.1280906	102.995
59.7818344	102.995
60.4355782	102.995

```
61.089322        102.995
61.7430658       102.995
62.3968096       102.995
63.0505534       102.995
63.7042972       102.995
64.358041        102.995
65.0117848       102.995
65.6655286       102.995
66.3192724       102.995
66.9730162       102.995
67.62676         102.995
0.0      0.0
```

E.1.2 spancontrolinputs.1.dat file

```
Casename:
 GE_E3_Fan_rotor
Blade row#:
1
Span control points, Chord & curvature control points
                5       7
Span    u2        u3      u4      u5      u6      cur1        cur2        cur3        cur4        cur5        cur6  cur7
0.00    0.15      0.25    0.50    0.75    0.95    1.909   3.6695     2.31   1.215   0.80    0.82   0.81
0.25    0.15      0.25    0.50    0.75    0.95    0.00    0.20       0.19   0.105   0.00    0.00   0.00
0.50    0.15      0.25    0.50    0.75    0.95    0.1   0.20  0.30   0.393   -0.04   0.00    0.00
0.75    0.15      0.25    0.50    0.75    0.95    0.00    0.10       0.10   0.105   0.00    0.00   0.00
1.00    0.15      0.25    0.50    0.75    0.95    -0.001   -0.002    0.00   0.33   0.4   0.2    0.1

Span control points, Chord & thickness control points
                5   1
Span    LE_radius    u_max    t_max    t_TE
```

```
0.00      2.5    0.55   0.0955  0.01548
0.25      3.0    0.41   0.0583  0.0153
0.50      3.0    0.43   0.0439  0.008415
0.75      3.0    0.48   0.0320     0.00364
1.00      3.0    0.60   0.0251  0.001
```

E.2 Reverse Engineered Geometry's Tblade3 input files

E.2.1 3dbgbinput.1.dat file

```
 Input parameters (version 1.31_TEST)
 GE_E3_Fan_rotor
 Blade row #:
           1
 Number of blades in this row:
          32
 Blade Scaling factor (mm):
   10.000000000000
 Number of streamlines:
          18
 Angles in the input file (0=Beta_z (default),1=Beta_r):
            0   inci_dev_spline
 Airfoil camber defined by curvature control (0=no,1=yes ;spanwise_spline=spancontrolinputs file):
            1   spanwise_spline
 Airfoil Thickness distribution (0=Wennerstrom,1=Spline,2=Spline + sharp TE):
           5
 Airfoil Thickness multiplier (0=no,1=yes):
           0
 Airfoil LE defined by spline (0=no,1=yes):
           0
 Non-dimensional Actual chord (0=no,1=yes,2=spline):
           2
 True lean and sweep (0=no,1=yes):
           0
 Clustering control (0=uniform,1=sine,2=exponential,3=hyperbolic,4=elliptical):
               4   15.0
```

```
Sectionwise properties:
J          in_Beta          out_Beta            mrel_in             chord              t/c_max
             Incidence          Deviation        Sec. Flow  Angle
1          42.340505069815           1.25546483167338           0.00000000
   0.00000000            0.000000             3.50            12.63            0.00000000
2          43.20           11.90           0.00000000           0.00000000
   0.000000           5.25            9.58            0.00000000
3          43.79           17.92           0.00000000           0.00000000
   0.000000           4.80            7.99            0.00000000
4          44.38           20.62           0.00000000           0.00000000
   0.000000           4.72            7.48            0.00000000
5          45.50           25.61           0.00000000           0.00000000
   0.000000           4.65            6.86            0.00000000
6          46.53           29.62           0.00000000           0.00000000
   0.000000           4.68            6.46            0.00000000
7          47.22           30.91           0.00000000           0.00000000
   0.000000           4.76            6.27            0.00000000
8          48.66           32.40           0.00000000           0.00000000
   0.000000           5.00            5.90            0.00000000
9          49.83           34.45           0.00000000           0.00000000
   0.000000           5.55            5.45            0.00000000
10         51.1586409283285           35.124499025686           0.00000000
   0.00000000            0.000000             5.65            5.26            0.00000000
11         51.41           37.30           0.00000000           0.00000000
   0.000000           5.68            5.05            0.00000000
12         51.89           38.13           0.00000000           0.00000000
   0.000000           5.73            4.90            0.00000000
13         53.09           40.20           0.00000000           0.00000000
   0.000000           5.70            4.56            0.00000000
14         54.86           43.56           0.00000000           0.00000000
   0.000000           5.66            3.93            0.00000000
15         56.63           46.71           0.00000000           0.00000000
   0.000000           5.50            3.43            0.00000000
16         58.56           49.69           0.00000000           0.00000000
   0.000000           4.95            3.22            0.00000000
17         61.00           52.81           0.00000000           0.00000000
```

```
0.000000          4.66          2.96          0.00000000
18          63.3524040738981          55.2433155340594          0.00000000          0.00000000          0.000000          5.00          2.48          0.00000000

LE / TE curve (x,r) definition :
Number of Curve points :
          18
  xLE   rLE      xTE      rTE
  -1.270        36.068   17.789   47.884
  -1.366        43.785   17.884   52.947
  -1.429        49.488   17.981   56.958
  -1.431        52.313   17.892   59.044
  -1.422        57.049   17.85    62.754
  -1.441        60.613   17.8     65.725
  -1.449        62.683   17.782   67.491
  -1.42         66.362   17.783   70.588
  -1.364        69.829   17.793   73.453
  -1.306        71.824   17.767   75.088
  -1.237        74.234   17.699   77.273
  -1.213        75.639   17.657   78.412
  -1.173        79.254   17.546   81.34
  -0.988        84.922   17.35    85.915
  -0.707        90.238   17.062   90.211
  -0.347        95.26    16.674   94.355
  0.050         100.11   16.305   98.424
  0.467         105.167  15.981   102.995

# Airfoil type and Variable Radial Stacking information.          #
# stack_u: % chord stack (0.00 to 100.00).                          #
# stack_v: % below or above meanline stack (-100.00 to +100.00). #
# Use +200 for stacking on airfoil area centroid.                 #
Variable Radial stacking (0=no,1=yes):
           0
J    type |stk_u |stk_v |umxthk |lethk |tethk  |Jcells(Grid:4n+1) |eta_ofst(<=10)[thkc/Jmax]  |BGgrid(0=no,1=yes) |
1    sect1 25.00  0.00  0.30  0.02  0.02 15 10  0
```

```
 2    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
 3    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
 4    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
 5    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
 6    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
 7    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
 8    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
 9    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
10    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
11    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
12    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
13    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
14    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
15    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
16    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
17    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0
18    sect1  25.00   0.00   0.30   0.02   0.02  15  10   0

Control table for blending section variable:
          4            0            0
      span                        bf1          bf2
    0.0000000000000000              1            0
   0.33333333333333331              1            0
   0.66666666666666663              1            0
    1.0000000000000000              1            0

Stacking axis location (200=centroid):
      055000

Control points for sweep:
          4
       span                    delta_m
    0.0000000000000000       0.0000000000000000
   0.25000000000000000       0.0000000000000000
   0.50000000000000000       0.0000000000000000
    1.0000000000000000       0.0000000000000000
```

```

Control points for lean:
           4
        span                     delta_theta
  0.0000000000000000        0.0000000000000000
 0.25000000000000000        0.0000000000000000
 0.50000000000000000        0.0000000000000000
  1.0000000000000000        0.0000000000000000

Control points for in_beta*:
           5
        span                      in_beta*
  0.0000000000000000        0.0000000000000000
  0.2500000000000000        0.0000000000000000
 0.50000000000000000        0.0000000000000000
 0.75000000000000000        0.0000000000000000
 1.00000000000000000        0.0000000000000000

Control points for out_beta*:
           5
        span                      out_beta*
  0.0000000000000000        0.0000000000000000
  0.2500000000000000        0.0000000000000000
 0.50000000000000000        0.0000000000000000
 0.75000000000000000        0.0000000000000000
 1.00000000000000000        0.0000000000000000

Control points for chord_multiplier:
           4
        span                 chord_multiplier
  0.0000000000000000        0.0000000000000000
 0.25000000000000000        0.0000000000000000
 0.50000000000000000        0.0000000000000000
  1.0000000000000000        0.0000000000000000

Control points for tm/c:
```

```
         4
      span                          tm/c
   0.0000000000000000          0.0000000000000000
  0.25000000000000000          0.0000000000000000
  0.50000000000000000          0.0000000000000000
   1.0000000000000000          0.0000000000000000

 Hub offset
   0.0000000000000000
 Tip offset
   0.0000000000000000

  Streamline Data
x_s      r_s
-28.495285       18.884164
-15.238214       27.295215
-9.472585        30.949586
-6.590009        32.77715
-1.27    36.068
1.35768  37.855728
5.446081         40.393893
9.534482         42.852843
14.20694         45.649056
17.789   47.884
19.492925        49.019945
25.425061        52.353179
31.961715        54.306013
38.62798         55.772693
45.39551         56.603429
52.137207        55.696729
57.971122        52.232159
67.62676         43.577681
0.0      0.0
-28.495285       21.208355
-15.238214       29.619406
-9.472585        33.273777
```

```
-6.590009	35.101341
-1.27	38.068
1.35768	39.855728
5.446081	42.393893
9.534482	44.852843
14.20694	47.649056
17.789	49.884
19.492925	51.319945
25.425061	54.653179
31.961715	56.606013
38.62798	58.072693
45.39551	58.903429
52.137207	57.996729
57.971122	54.532159
67.62676	45.877681
0.0	0.0
-28.495285	28.808422
-15.238214	36.280124
-9.472585	39.526374
-6.590009	41.149834
-1.366	43.785
1.35768	45.339242
5.446081	47.534739
9.534482	48.452355
14.20694	50.936022
17.884	52.947
19.492925	54.248797
25.425061	57.275809
31.961715	59.049238
38.62798	60.381176
45.39551	61.135593
52.137207	60.31219
57.971122	57.165907
67.62676	49.306509
0.0	0.0
-28.495285	35.903042
```

```
−15.238214      42.680548
−9.472585       45.625189
−6.590009       47.097814
−1.429  49.488
1.35768 50.869692
5.446081        52.81195
9.534482        52.888389
14.20694        55.124451
17.981  56.958
19.492925       58.153522
25.425061       60.937941
31.961715       62.569241
38.62798        63.794433
45.39551        64.488389
52.137207       63.730976
57.971122       60.836846
67.62676        53.607324
0.0     0.0
−28.495285      39.417386
−15.238214      45.851019
−9.472585       48.646257
−6.590009       50.044165
−1.431  52.313
1.35768 53.609219
5.446081        55.426035
9.534482        55.195437
14.20694        57.302727
17.892  59.044
19.492925       60.184252
25.425061       62.842504
31.961715       64.399888
38.62798        65.569566
45.39551        66.232077
52.137207       65.508984
57.971122       62.745991
67.62676        55.844048
```

```
0.0 0.0
−28.495285 45.309042
−15.238214 51.166188
−9.472585 53.710958
−6.590009 54.983606
−1.422 57.049
1.35768 58.201926
5.446081 59.808443
9.534482 59.298575
14.20694 61.176842
17.85 62.754
19.492925 63.795952
25.425061 66.229816
31.961715 67.655738
38.62798 68.72668
45.39551 69.333267
52.137207 68.671212
57.971122 66.141449
67.62676 59.822115
0.0 0.0
−28.495285 49.742713
−15.238214 55.166033
−9.472585 57.522316
−6.590009 58.700702
−1.441 60.613
1.35768 61.658094
5.446081 63.106353
9.534482 62.584404
14.20694 64.279266
17.8 65.725
19.492925 66.688233
25.425061 68.942404
31.961715 70.26305
38.62798 71.254924
45.39551 71.816727
52.137207 71.203551
```

```
57.971122         68.860562
67.62676          63.007785
0.0        0.0
-28.495285        52.317825
-15.238214        57.489174
-9.472585         59.735984
-6.590009         60.859622
-1.449   62.683
1.35768  63.665464
5.446081          65.021806
9.534482          64.537542
14.20694          66.123387
17.782   67.491
19.492925         68.407441
25.425061         70.554801
31.961715         71.812869
38.62798          72.757745
45.39551          73.292927
52.137207         72.708806
57.971122         70.476837
67.62676          64.901387
0.0        0.0
-28.495285        61.207559
-15.238214        65.509064
-9.472585         67.37795
-6.590009         68.312587
-1.364   69.829
1.35768  70.595254
5.446081          71.634283
9.534482          71.131318
14.20694          72.34912
17.793   73.453
19.492925         74.211473
25.425061         75.998238
31.961715         77.045046
38.62798          77.831253
```

```
45.39551	78.276565
52.137207	77.790532
57.971122	75.933366
67.62676	71.294172
0.0	0.0
-28.495285	66.687447
-15.238214	70.452755
-9.472585	72.088678
-6.590009	72.906809
-1.237	74.234
1.35768	74.866976
5.446081	75.710404
9.534482	75.356112
14.20694	76.338101
17.699	77.273
19.492925	77.930259
25.425061	79.485982
31.961715	80.39743
38.62798	81.081974
45.39551	81.469704
52.137207	81.046519
57.971122	79.429498
67.62676	75.390187
0.0	0.0
-28.495285	68.435289
-15.238214	72.029574
-9.472585	73.591192
-6.590009	74.372163
-1.213	75.639
1.35768	76.229467
5.446081	77.010506
9.534482	76.615809
14.20694	77.527485
17.657	78.412
19.492925	79.03908
25.425061	80.525914
```

```
31.961715     81.397002
38.62798      82.051234
45.39551      82.421795
52.137207     82.017349
57.971122     80.471932
67.62676      76.611486
0.0     0.0
−28.495285    72.932404
−15.238214    76.086655
−9.472585     77.45709
−6.590009     78.14245
−1.173  79.254
1.35768 79.735091
5.446081      80.355609
9.534482      79.854081
14.20694      80.585008
17.546  81.34
19.492925     81.8895
25.425061     83.199242
31.961715     83.966578
38.62798      84.542887
45.39551      84.869311
52.137207     84.513037
57.971122     83.15169
67.62676      79.751049
0.0     0.0
−28.495285    79.983484
−15.238214    82.447799
−9.472585     83.518476
−6.590009     84.053926
−0.988  84.922
1.35768 85.2316
5.446081      85.600433
9.534482      84.913881
14.20694      85.362386
17.35   85.915
```

```
19.492925       86.343282
25.425061       87.376318
31.961715       87.98154
38.62798        88.436094
45.39551        88.693555
52.137207       88.41255
57.971122       87.338811
67.62676        84.656616
0.0     0.0
−28.495285      86.596669
−15.238214      88.413896
−9.472585       89.203432
−6.590009       89.598281
−0.707  90.238
1.35768 90.386759
5.446081        90.519539
9.534482        89.665116
14.20694        89.848423
17.062  90.211
19.492925       90.525456
25.425061       91.29866
31.961715       91.751656
38.62798        92.091879
45.39551        92.284583
52.137207       92.074257
57.971122       91.270588
67.62676        89.263024
0.0     0.0
−28.495285      92.844115
−15.238214      94.050041
−9.472585       94.573982
−6.590009       94.836008
−0.347  95.26
1.35768 95.256813
5.446081        95.166594
9.534482        94.248244
```

```
14.20694        94.175736
16.674  94.355
19.492925       94.559657
25.425061       95.082223
31.961715       95.388378
38.62798        95.618316
45.39551        95.748554
52.137207       95.606406
57.971122       95.063251
67.62676        93.706449
0.0     0.0
−28.495285      98.877589
−15.238214      99.493151
−9.472585       99.760595
−6.590009       99.894346
0.05    100.11
1.35768 99.960072
5.446081        99.654491
9.534482        98.748424
14.20694        98.424732
16.305  98.424
19.492925       98.520845
25.425061       98.79731
31.961715       98.959281
38.62798        99.08093
45.39551        99.149833
52.137207       99.074629
57.971122       98.787272
67.62676        98.069455
0.0     0.0
−28.495285      103.167
−15.238214      103.167
−9.472585       103.167
−6.590009       103.167
0.467   103.167
3.5698  102.7326
```

```
6.6726   102.2982
9.7754   101.8638
12.8782  101.4294
15.981   100.995
19.492925       100.995
25.425061       100.995
31.961715       100.995
38.62798        100.995
45.39551        100.995
52.137207       100.995
57.971122       100.995
67.62676        100.995
0.0      0.0
-28.495285      105.167
-28.1286738     105.167
-27.7620626     105.167
-27.3954514     105.167
-27.0288402     105.167
-26.662229      105.167
-26.2956178     105.167
-25.9290066     105.167
-25.5623954     105.167
-25.1957842     105.167
-24.829173      105.167
-24.4625618     105.167
-24.0959506     105.167
-23.7293394     105.167
-23.3627282     105.167
-22.996117      105.167
-22.6295058     105.167
-22.2628946     105.167
-21.8962834     105.167
-21.5296722     105.167
-21.1630609     105.167
-20.7964497     105.167
-20.4298385     105.167
```

−20.0632273	105.167
−19.6966161	105.167
−19.3300049	105.167
−18.9633937	105.167
−18.5967825	105.167
−18.2301713	105.167
−17.8635601	105.167
−17.4969489	105.167
−17.1303377	105.167
−16.7637265	105.167
−16.3971153	105.167
−16.0305041	105.167
−15.6638929	105.167
−15.2972817	105.167
−14.9306705	105.167
−14.5640593	105.167
−14.1974481	105.167
−13.8308369	105.167
−13.4642257	105.167
−13.0976145	105.167
−12.7310033	105.167
−12.3643921	105.167
−11.9977809	105.167
−11.6311697	105.167
−11.2645585	105.167
−10.8979473	105.167
−10.5313361	105.167
−10.1647249	105.167
−9.7981137	105.167
−9.4315025	105.167
−9.0648913	105.167
−8.6982801	105.167
−8.3316689	105.167
−7.9650577	105.167
−7.5984465	105.167
−7.2318353	105.167

```
−6.8652241 105.167
−6.4986128 105.167
−6.1320016 105.167
−5.7653904 105.167
−5.3987792 105.167
−5.032168 105.167
−4.6655568 105.167
−4.2989456 105.167
−3.9323344 105.167
−3.5657232 105.167
−3.199112 105.167
−2.8325008 105.167
−2.4658896 105.167
−2.0992784 105.167
−1.7326672 105.167
−1.366056 105.167
−0.9994448 105.167
−0.6328336 105.167
−0.2662224 105.167
0.1003888 105.167
0.467 105.167
0.6561951 105.1405122
0.8453902 105.1140244
1.0345854 105.0875366
1.2237805 105.0610488
1.4129756 105.034561
1.6021707 105.0080732
1.7913659 104.9815854
1.980561 104.9550976
2.1697561 104.9286098
2.3589512 104.902122
2.5481463 104.8756341
2.7373415 104.8491463
2.9265366 104.8226585
3.1157317 104.7961707
3.3049268 104.7696829
```

3.494122	104.7431951
3.6833171	104.7167073
3.8725122	104.6902195
4.0617073	104.6637317
4.2509024	104.6372439
4.4400976	104.6107561
4.6292927	104.5842683
4.8184878	104.5577805
5.0076829	104.5312927
5.196878	104.5048049
5.3860732	104.4783171
5.5752683	104.4518293
5.7644634	104.4253415
5.9536585	104.3988537
6.1428537	104.3723659
6.3320488	104.345878
6.5212439	104.3193902
6.710439	104.2929024
6.8996341	104.2664146
7.0888293	104.2399268
7.2780244	104.213439
7.4672195	104.1869512
7.6564146	104.1604634
7.8456098	104.1339756
8.0348049	104.1074878
8.224	104.081
8.4131951	104.0545122
8.6023902	104.0280244
8.7915854	104.0015366
8.9807805	103.9750488
9.1699756	103.948561
9.3591707	103.9220732
9.5483659	103.8955854
9.737561	103.8690976
9.9267561	103.8426098
10.1159512	103.816122

```
10.3051463      103.7896341
10.4943415      103.7631463
10.6835366      103.7366585
10.8727317      103.7101707
11.0619268      103.6836829
11.251122       103.6571951
11.4403171      103.6307073
11.6295122      103.6042195
11.8187073      103.5777317
12.0079024      103.5512439
12.1970976      103.5247561
12.3862927      103.4982683
12.5754878      103.4717805
12.7646829      103.4452927
12.953878       103.4188049
13.1430732      103.3923171
13.3322683      103.3658293
13.5214634      103.3393415
13.7106585      103.3128537
13.8998537      103.2863659
14.0890488      103.259878
14.2782439      103.2333902
14.467439       103.2069024
14.6566341      103.1804146
14.8458293      103.1539268
15.0350244      103.127439
15.2242195      103.1009512
15.4134146      103.0744634
15.6026098      103.0479756
15.7918049      103.0214878
15.981  102.995
16.6347438      102.995
17.2884876      102.995
17.9422314      102.995
18.5959752      102.995
19.249719       102.995
```

19.9034628	102.995
20.5572066	102.995
21.2109504	102.995
21.8646942	102.995
22.518438	102.995
23.1721818	102.995
23.8259256	102.995
24.4796694	102.995
25.1334132	102.995
25.787157	102.995
26.4409008	102.995
27.0946446	102.995
27.7483884	102.995
28.4021322	102.995
29.0558759	102.995
29.7096197	102.995
30.3633635	102.995
31.0171073	102.995
31.6708511	102.995
32.3245949	102.995
32.9783387	102.995
33.6320825	102.995
34.2858263	102.995
34.9395701	102.995
35.5933139	102.995
36.2470577	102.995
36.9008015	102.995
37.5545453	102.995
38.2082891	102.995
38.8620329	102.995
39.5157767	102.995
40.1695205	102.995
40.8232643	102.995
41.4770081	102.995
42.1307519	102.995
42.7844957	102.995

43.4382395	102.995
44.0919833	102.995
44.7457271	102.995
45.3994709	102.995
46.0532147	102.995
46.7069585	102.995
47.3607023	102.995
48.0144461	102.995
48.6681899	102.995
49.3219337	102.995
49.9756775	102.995
50.6294213	102.995
51.2831651	102.995
51.9369089	102.995
52.5906527	102.995
53.2443965	102.995
53.8981403	102.995
54.5518841	102.995
55.2056278	102.995
55.8593716	102.995
56.5131154	102.995
57.1668592	102.995
57.820603	102.995
58.4743468	102.995
59.1280906	102.995
59.7818344	102.995
60.4355782	102.995
61.089322	102.995
61.7430658	102.995
62.3968096	102.995
63.0505534	102.995
63.7042972	102.995
64.358041	102.995
65.0117848	102.995
65.6655286	102.995
66.3192724	102.995

```
66.9730162        102.995
67.62676          102.995
0.0       0.0
```

E.2.2 spancontrolinputs.1.dat file

```
 Casename:
 GE_E3_Fan_rotor
Blade row#:
1
Span control points, Chord & curvature control points
                  3        7
Span    u2          u3      u4      u5      u6      cur1            cur2    cur3                    cur4                    cur5                    cur6  cur7
0.00    0.15    0.25    0.50   0.75    0.95   1.695915356433488   3.631585002764526   2.31    1.751813546611551   0.7073926409875017        0.5110641023004737    0.6394991587684383
0.55    0.15    0.25    0.50   0.75    0.95   -0.9521920902369565   0.3033396118222525    0.50   0.02079877565624548   0.8450197313160744   -0.01305617829214443    -0.6658932068859058
1.00    0.15    0.25    0.50   0.75    0.95   -0.4338899693604618   -1.109422191592448    0.50   -0.1487365420570926   1.81450867904997     1.1430224859446774    0.10694237796936842

Span control points, Chord & thickness control points
                  3    1
Span     LE_radius     u_max    t_max    t_TE
0.00     2.3591620846850745    0.4565529040410505   0.09702213967820521    0.003637203014408098
0.55     2.228717117381842    0.6147196915472138   0.03565671006561507    0.0001000000000000715
1.00     2.4577201043140238    0.6316847625321914   0.02622001980009441  0.0001
```