Multimodal optimization of NACA0012 wing with differential evolution based parallel niching algorithm and reduced-order modeling technique.

A thesis submitted

in partial fulfillment for the award of the degree of

Master of Technology

in

Aerodynamics and Flight Mechanics

by

Neelappagouda V Hiregoudar

Department of Aerospace Engineering

Indian Institute of Space Science and Technology

Thiruvananthapuram, India

July 2020

Certificate

This is to certify that the thesis titled ***Multimodal optimization of NACA0012 wing with differential evolution based parallel niching algorithm and reduced-order modeling technique.*** submitted by **Neelappagouda V Hiregoudar**, to the Indian Institute of Space Science and Technology, Thiruvananthapuram, in partial fulfillment for the award of the degree of **Master of Technology** in **Aerodynamics and Flight Mechanics** is a bona fide record of the original work carried out by him/her under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Devendra Prakash Ghate
Designation

Name of Department Head
Designation

Place: Thiruvananthapuram
Date: July 2020

Declaration

I declare that this thesis titled ***Multimodal optimization of NACA0012 wing with differential evolution based parallel niching algorithm and reduced-order modeling technique.*** submitted in partial fulfillment for the award of the degree of **Master of Technology** in **Aerodynamics and Flight Mechanics** is a record of the original work carried out by me under the supervision of **Devendra Prakash Ghate**, and has not formed the basis for the award of any degree, diploma, associateship, fellowship, or other titles in this or any other Institution or University of higher learning. In keeping with the ethical practice in reporting scientific information, due acknowledgments have been made wherever the findings of others have been cited.

Place: Thiruvananthapuram
Date: July 2020

Neelappagouda V Hiregoudar
(SC18M033)

This thesis is dedicated to . . .

Acknowledgements

I acknowledge . . .

Neelappagouda V Hiregoudar

Abstract

The thesis presented here aims at obtaining multiple optimal shapes for the NACA0012 wing. Beforehand, the Differential Evolution (DE) algorithm tested on the test function like Ackley function, Hypersphere, Egg-holder function, to name a few and found that the python optimization code implemented correctly. Using sophisticated DE-based niching algorithms like fDE, fNRAND1, fINRAND1, fCDE to name a few, it is possible to obtain local optima for test functions mentioned above.

The DE based niching algorithms (with little or no modification) are further expanded to NACA0012 wing to obtain multiple local optima. The mesh points spread over the NACA0012 airfoil using a cosine function. A Free-Form Deformation (FFD) box with 60 control points is set-up over the NACA0012 wing, which resulted in 125 Dimensional (125-D) optimization problem. With the help of Principal Component Analysis, the problem dimension gets reduced to 10 based on its percentage of random energy (λ^2). A glyph script creates the wing tip followed by the volume mesh to all perturbed wings, and the SU2 solver will evaluate for Coefficient of lift (Cl) and Coefficient of drag (Cd).

The entire optimization problem carried out in subsonic, inviscid condition (M = 0.4) subjected to several constraints with reasonably higher residual value. A full-fledged optimization code written in python and submitted to the HPC cluster via SLURM has resulted in two optima. A further modification to the existing design space results in additional optimal wing shapes.

Keywords: Optimization, Parameterization, Niching algorithms, Differential Evolution, Free - Form Deformation, Principal Component Analysis

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

GB	Gradient-Based
GF	Gradient free
DE	Differential Evolution
ASO	Aerodynamics Shape Optimization
SU2	Stanford University Unstructured
CR	Cross-over probability
AoA	Angle of Attack
FFD	Free Form Deformation
PCA	Principal Component Analysis
SVD	Singular Value Decomposition

Nomenclature

N	Population size
F	Mutation factor
v_n	Donor vector
x_n	Initial vector
u_n	Trial vector
D	Dimension of problem
CR	Cross over
$\prec$	Dominates
N_g	Number of generation
FEs	Function evaluations
FEs_{max}	Maximum number of function evaluations
C_l	Coefficient of Lift
C_d	Coefficient of Drag
x_i^0	Nominal surface geometry
$x'_{i,j}$	Error vector of the given sample
$\hat{x}_{i,j}$	Vector representing particular set of variables
$\bar{x}$	Mean of error vectors
x_r	Reduced-order model constructed using n_r leading modes from mp directions

Chapter 1

Introduction

1.1 Background

During the early day's optimization of aerodynamics, shape optimization was a tedious job. Given a problem of drag minimization on the wing surface, the designer will manufacture the wing's prototype. Then wind tunnel testing is carried out. After looking into the results, the shape gets modified, or a new model gets manufactured. Again, subject to testing. Redesigning the wing's shape involves many resources. Even after doing all these steps, the designer ends up with a single-wing shape called a single modal design, or unimodal design optimization. With computational power and sophisticated algorithms, it is now possible to have multiple optimal shapes for an objective function subjected to constraints.

In a recent decade, computational techniques have become more reliable, assisting aircraft designers in improving the process of designing new aircraft components. Additionally, it even shortened the design cycle times and to explore higher design space effectively. For example, the CFD codes allow the designers to reduce their dependence on the more expensive and time-consuming wind tunnel tests. CFD is a mature technology that is used in every industry. High-fidelity methods provide the engineers for a better understanding of the design tradeoffs and make better decisions.

Aerodynamic shape optimization (ASO) can become a promising area of research and lead to a breakthrough in aircraft components design. This approach involves using optimization algorithms in combination with the CFD solver and the parameterization technique to reduce the problem's dimension. These tools allow for robust tuning of the existing design. The goal of ASO is to improve the aerodynamic characteristics of the aircraft, resulting in increased fuel efficiency.

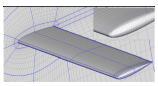
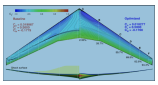
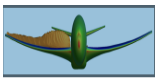
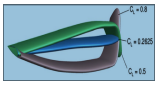
Figure	Case	Description	References
	1	Sectional optimization of a modified NACA 0012 airfoil at zero angles of attack in inviscid, transonic flow.	[6, 7, 8]
	2	Sectional optimization of an RAE 2822 airfoil in viscous, transonic flow.	[6, 7, 9, 10]
	3	Twist optimization of a rectangular wing in subsonic, inviscid flow.	[9, 10, 8]
	4	Sectional and twists optimization of the Common Research Model (CRM) wing in transonic, viscous flow.	[9, 10, 8, 6]
	5	Lift-constrained drag minimization of the CRM wing-body-tail configuration at flight Reynolds number.	[6, 11]
	6	**Multimodal subsonic inviscid lift-constrained drag minimization.**	[12, 13]

Table 1.1: ADODG benchmark cases

1.2 ADODG

Despite researching more than two decades, there is no clear idea of the problem formulations used to obtain practical aerodynamic designs, and the strategies to solve aerodynamic shape optimization problems. Further, performing the ASO based on the Reynolds-averaged Navier-Strokes (RANS) equations on a large cluster of the grid size remains challenging. Several forums have established to compare the CFD algorithms that enable researchers to validate their codes. The aerodynamic design optimization community makes a similar attempt to initiate the AIAA Aerodynamic Design Optimization Discussion Group (ADODG). The first meeting for the discussion group was in the year 2014.

To understand the aerodynamic shape optimization in the better way, the ADODG has developed a series of benchmark problems ranging from 2-D airfoil optimization on the Euler equations to the 3-D wing shape optimization based on the RANS equations. These benchmark cases mainly concentrate on drag minimization subjected to several constraints.

The benchmark cases [14] are listed in Table 1.1. The objective of the problems is to minimize the drag coefficient. Most of the mentioned cases are subjected to lift coefficient

constraints along with thickness constraints. Furthermore, in some cases, the problem is also subjected to moment coefficient constraints.

The first two cases are related to airfoil shape optimization benchmark problems: Case 1 begins with NACA 0012 airfoil as a baseline, whereas in Case 2, it begins with RAE 2822 airfoil. Apart from this, both the case problem solve different aerodynamic models; that is, Case 1 deals with Euler equations, and the counterpart with RANS equations. Furthermore, Case 1 does not involve the lift constraint, whereas the other does. Based on historical work available, it can be concluded that much work is carried out on this case problem. The reason for this could be the lower cost of solving the two-dimensional cases.

Cases 3 and 4 involve optimizing the wing shapes; Case 3 is a subsonic case-solving Euler equation, while Case 4 is a transonic case based on RANS. Case 3 optimizes only twist, but Case 4 includes both twist and airfoil shapes. All wing optimization cases are subjected to a volume constraint, where any optimized wing generated should posses the volume, which is higher than the baseline wing.

Case 5 is the extension of Case 4, which adds the fuselage and tail. The baseline shape for Case 4 is the wing from the CRM full aircraft configuration combined with the fuselage intersection. Case 5 restores the full CRM aircraft configuration and is considered as a benchmark case that suits for industrial needs. Also, it uses the flight Reynolds number for testing. The thickness constraint prevents the thickness of the optimized shape, lowering below the baseline aircraft shape. Its tail rotation angle is considered another design variable used to trim the aircraft at various flight conditions.

Finally, Case 6 is a benchmark case for multimodality. More preciously, Case 6 is the wing optimization problem that is similar to Case 3 combined with the airfoil shape variables and planform variables (Sweep, span, dihedral, and chord variation). Case 6 provides additional flexibility for design space exploration and determines the existence of multimodality.

The present work concentrates on Case 6: Multimodal optimization of the wing surface. It involves several geometric and aerodynamic constraints, which will be detailed in chapter 4. Also, the problem of interest involves parameterizing the wing, choice of CFD solver, and optimization algorithm. Each of these crucial steps is explained in detail in subsequent chapters. However, a minor introduction is provided in the following sections.

1.3 Importance of parameterization

Several approaches to the parameterization of wing profiles are available in the literature. Point clouds can describe 2-D airfoils as it is done in most airfoil libraries. The number of parameters is twice the number of points (x and y coordinates); in the case of 3-D, the parameters will be thrice as that of the number of coordinates. It is undoubtedly impossible to optimize the object using the coordinate points. Parameterization allows a larger design space to represent the given object.

Methods like Radial basis function, B-splines, CST function are generally used techniques to parameterize the given object. Radial basis function methods use basis function to represent the object. The B-splines method is considered to be easy to implement for a 2-D object. Similarly, for a 3-D surface, bezier surfaces are used to parameterize the surface. CST function involves a higher degree of computation to get the perturbed coordinates back. Among all the above methods, researchers generally prefer to opt for the B-spline methods, due to its ease of implementation. Apart from these, there are methods like NURBS (Non-Uniform Rational B-spline), PARSEC, Hicks-Henne bump function, to name a few.

However, in this work, an extended version of the B-spline method called Free-form deformation (FFD) box is used. In this method, the object is circumvented using the box, and corner (control) points coordinates are used to parameter the object. Irrespective of object shape, the FFD can be implemented. FFD method uses Bernstein polynomial to parameterize the object. More on this will be covered in chapter 4.

1.4 Solver selection

While address the aerodynamic shape optimization problem, the crucial step in the entire process is the choice of CFD solver. The advancement in computational power paved the way for researchers to build the solver, which is robust, open-source, portable, and efficient in their performance. Further, the CFD solver is expected to satisfy the practical industrial constraints. In the present condition, the solver is combined with several ASO supporting modules, namely gradient calculation, mesh deformation, grid generation, to name a few. These solvers are generally written in C++ modules for ease of readability.

The work presented here uses SU2 as the CFD solver. The choice behind the solver is its ease in the implementation, open-source, portability, and, more importantly, it is embedded with the FFD box. Python modules are used to build a SU2 solver. This further increases

its reach into the optimization problems. Also, it possesses the gradient calculation and a full-fledged optimizer, which does the entire design step. Since the present work demands the inviscid (Euler) equation to solve, the selected CFD solver is sufficient for use.

1.5 Choice of optimizer

Optimization is a computationally expensive process. A unique optimization algorithm cannot address all the optimization problems. Finding a suitable algorithm is problem-dependent. Also, it depends on the number of constraints, type of design variables, to name a few.

Based on constraint availability, an optimization problems is classified into Constraints optimization problems (COP), and Unconstrained optimization problems (UOP). Most of the real-world problems are COP. However, UOP is worth studying because COP can be converted into UOP using a penalty approach. In addition to this, an optimization problem can have discrete design variables or integer design variables. However, in a broader sense, optimization methods is classified two categories:

- Gradient-based (GB) method.
- Gradient free method.

The Gradient-based (GB) algorithm involves finding the gradient [appendix A.1] at every iteration. For example, Quasi-Newton methods, such as Broyden-Fletcher-Goldfarb-Shanno (BFGS) involves constructing a Hessian approximation [appendix A.2] at every iteration. The steepest-descent and the conjugate-gradient requires the calculation of first derivative of the objective function. While the others need the second derivative (Hessian matrix) of the objective function. Any function $f(x)$ can be expressed in terms of divergence and Hessian matrix which is as represented in appendix A.5. The GB methods is generally used for higher dimensional problems.

In real-world problems, it is quite often to encounter non-differentiable problems, non-convex design space, discrete feasible space, large dimensionality functions, multiple local optima, or multiple objectives problems. These kinds of problems are not possible to address from the gradient-based method, as they involve gradient calculation, which may not be available. Under this circumstance, Gradient free methods come in handy, which do not involve any gradient calculation and are heuristic in nature.

Many gradient-free methods are developed in the early days, like the Nelder-Mead Simplex algorithm, Simulated Annealing, Divided Rectangules Method, Genetic Algorithms,

Particle Swarm Optimization, to name a few. This work concentrate on the genetic algorithms because of implementation simplicity.

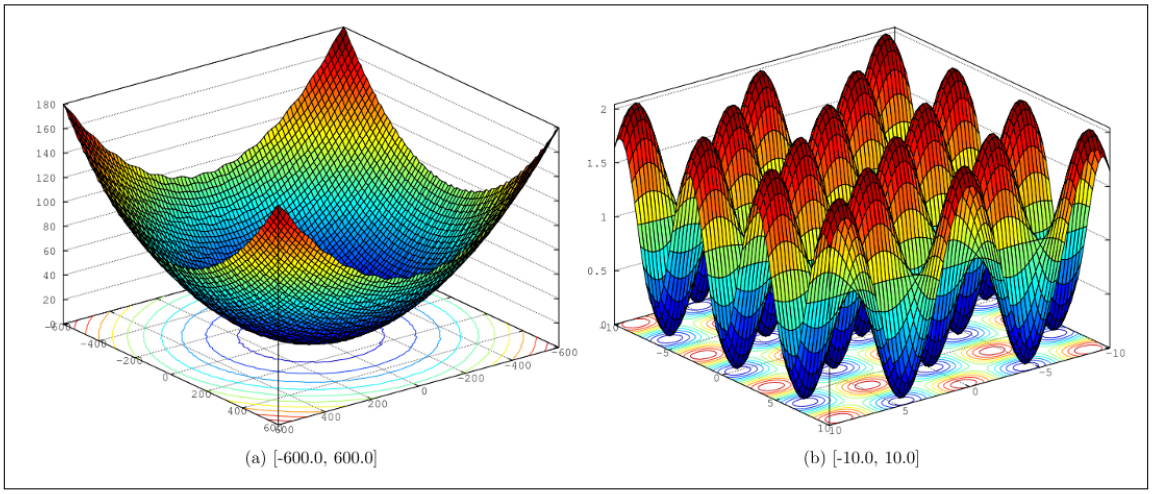
1.6 Comparision between Gradient-based and Gradient free methods.

Gradient-based methods are efficient in finding local minima for problems that generally possess nonlinear constraints, higher dimensional, and convex problems[15]. However, these methods have problems while dealing with noisy and discontinuous functions. Also, these methods are not designed to handle multi-modal problems or discrete-continuous design variables. Many gradient-free methods are heuristic in nature. Unlike the gradient-based method in a convex design space, the gradient-free methods may not converge to a global optimal. However, these may result in multiple good solutions rather than a single best solution.

Consider, for example, the Griewank function [1.1],

$$f(x) = \sum_{i=1}^{n} \left(\frac{x_i^2}{4000} \right) - \prod_{i=1}^{n} \cos \left(\frac{x_i}{\sqrt{i}} \right) + 1 \tag{1.1}$$

$$-600 \leq x_i \leq 600$$



(a) [-600.0, 600.0] (b) [-10.0, 10.0]

Figure 1.1: Griewank function for n = 2, right side image represent the domain (-600, 600), and the left side image represent the domain (-10, 10) [1]

The Griewank function appears smooth (convex) when plotted in a broader domain (left). However, the same function plotted in smaller design space result in multiple local minima (right). Every run with a gradient-based method will converge to the nearest local minima. Further, a multi-start gradient-based method is introduced. Here the optimizer is initialized from multiple points within the design space. With this, there is a possibility of finding the global optima, But not always guaranteed for convergence to global optima. The frequency of using multi-start method is rare because,

- large computational cost.
- No guarantee about the success of the search.

The problems which seem difficult to solve using gradient-based methods, the gradient-free method can be implemented. In addition to this, many of the gradient-free methods are designed to find global optimal. However, they can find multiple local optima while searching for global optima.

Differential evolution is a class under a genetic algorithm. These algorithms evolve with time, which implies the solution move to the optimal point after every generation. These algorithms follow mutation, crossover, and selection phases. However, implementing the DE will not result in obtaining the multimodality. So the modified form of the DE algorithm has to be used. These modifications are named as niches, and the modified algorithms are called as niching algorithms. More on this is explained in chapter 3.

This work constitutes finding multiple optima. On considering above mentioned methods, the gradient-free method/optimization (specifically genetic algorithm) align better with the present work. Furthermore, the implementation is straightforward and easily parallelized.

1.7 Outline

This thesis is divided into six further chapters,

- Chapter 2 provides information about the literature review related to problem mentioned in ADODG Case 6, parameterization of wing, and selection of optimizer.
- Chapter 3 describes the differential evolution based niching algorithm. Furthermore, this chapter contains several niching algorithms that are implemented over the test function. This chapter concludes by selecting one of the niching algorithms used in this work to implement the ADODG Case 6 problem.

- Chapter 4 explains the parameterization of the wing surface. It includes a detailed explanation of the FFD box, followed by the PCA implementation. Furthermore, the chapter concludes by mentioning the problem dimension after the FFD box and PCA method.

- Chapter 5 mention the choice for CFD solver and the parameter used to set up the CFD solver. Additionally, it describes the parameters involved in job submission into the HPC cluster via SLURM.

- Chapter 6 deals with the flow of data right from the baseline geometry until the end of optimized wing shapes. It contains the equation to generate the NACA 0012 airfoil, extruding the 2-D airfoil to 3-D wing, implementation of the FFD box, and PCA method, creating the winglet, followed by the SU2 solver implementation. The flow chart representing the flow of data is presented. Along with this, this chapter even mentions about the glyph script, which creates the wingtip, followed by creating volume mesh and setting boundary conditions.

- Final chapter 7 details the test function results and also the results associated after optimizing the wing. Further, it explains the conclusion and highlights the scope for future work.

Chapter 2

Literature review

2.1 Introduction

The primary constraint in Aerodynamics Shape Optimization (ASO) is objective function evaluations, which are computationally expensive. This requires optimized methods to reduce the objective function evaluations. Apart from this, the other concern is to parameterize the wing and the choice of design variables. ADODG has suggested a problem (case 6) that involves investigating the existence of multimodality in NACA0012 wing subjected to geometric and aerodynamics constraints. Subsequent sections summarizes the work involving the parameterization of an object's shape and the algorithms to find multi-modal optima within the given design space.

2.2 Historical work

Chernukhin [16] investigated the existence of multimodality in aerodynamic design space. The author proposed two novel optimization algorithms that can be used to find the existence of multimodality. They are gradient-based multi-start optimization (GB-MS) based on Sobel sampling and the genetic algorithm. These algorithms are initially tested on the 2-D ASO problem. Using the gradient-based method, the optimizer took 64000 functions evaluation to converge to single optima. However, carry out these many function evaluations would require a lot of computational time. Also, the design space chosen appears to be huge.

Further, the author extends the work to 3-D ASO (similar to that of ADODG case 6), with both transonic and subsonic conditions. The volume mesh is made of 12 blocks with 1.1m nodes. The objective function is to minimize the drag at a Mach number of 0.8 and

lift constrained 0.2625. The wing is parameterized using a 2-patch B-spline surface with an initial NACA 0012 baseline wing having a rectangular crossover and a semi span of two. With 125 design variables, the GB optimizer took 60 function evaluations. A similar approach is carried out with GB-MS method involving 128 initial geometries resulting in single optima. In subsonic conditions, that is with Mach number 0.5 and angle of attack between -3 and +6 degree, resulted in seven unique local optima whose objective values differ by more than 5%.

The author confirms that 2-D airfoil optimization is unimodal, whereas 3-D wing optimization in subsonic flow conditions is multimodal [17]. However, with additional geometric constraints may further reduce multimodality. This work uses the GB-MS algorithm, and gradient-based methods are generally sensitive to the initial population. Also, the use of a gradient-free algorithm in large design variables is computationally expensive and unacceptable. However, with reduced design variables, it is possible to use a gradient-free method efficiently.

In work published by D.J. Poole [18] $et\ al.$, the author addressed the ADODG case 6 problem. The problem is to minimize the drag subjected to aerodynamics and geometric constraints. The wing is of rectangular section with a span length of 3 units and round winglet of 0.06 units. Also, the wing is subjected to compressible, inviscid flow with a Mach number of 0.5 and a constrained lift coefficient of 0.2625. The volume mesh is made of structured multiblock with the convective term being evaluated using Jameson-Schmidt-Turkel (JST) scheme. Nearly 5.5m node points are generated in an eight-block structured C-mesh topology. Further, the surface mesh is made of 129×41 mesh points. The author recommends using radial basis functions (RBFs) for surface control and mesh deformations.

The wing is parameterized using five equally spaced layers along the span direction comprising of 24 control points each along the chord direction. In total, the problem contains 22 design variables with thickness, z-position, x-position, and a chord at five equally spaced wing sections, resulting in 20 design variables. The remaining two design variables are AoA and Twist [zero at the root and twist angle at the tip]. The AoA is varied within limits set by constraints. The author utilized a hybrid optimizer consisting of particle swarm optimization (PSO) and gravitational search algorithm (GSA).

The work contains three cases; thickness optimization only, chord optimization only, and full optimization. However, the algorithm which is adopted here result in global optimization. Achieving multimodality is not possible with these algorithms. It is found that the design space chosen for full optimization is multimodal. A strong coupling exists

between the chord and thickness of the wing, which increases the multimodality of the problem. However, the existence of multimodality is confirmed by running the optimizer several times. This allows the researcher to rethink on optimizer, which proves multimodality in a single run.

It can be concluded that for any ASO problems, the crucial steps involved are a choice of parameterization technique, selection of optimizer, and the optional step involving dimensional reduction method. Upcoming sections cover the historical work which involve the parameterization techniques, and selection of optimizer.

In the past, attempts have been made to simplify the geometry representation. Meaux $et\ al.$ [19] used (Non-Uniform Rational B-Spline) NURBS to optimize complex 3D geometries. Nadarajah[20] compares three different **parameterization schemes**: B-splines, CST method, and Mesh-point for airfoil geometry. The Mesh-point method involves representing the wing shape with x, y, and z coordinates of grid points. This method would be inefficient as it takes mesh points coordinates as design variables. Since this method involves large matrix operation, evaluating the gradient is difficult, which limits the use of descent algorithm. In the B-spline method, the degree of polynomial representing the surface act as a design variable. With a lower degree to polynomial, it is possible to reduce the dimension of the optimization problem.

The CST method provides the mathematical description of the geometry through a combination of a shape function and class function. The class function provides for a wide variety of geometries. The shape function replaces the complex non-analytic function with a simple analytic function. The simple analytical function uses only a scalable parameter to represent a large domain of design space for the aerodynamic problem. The advantage of CST is, it is efficient in terms of a low number of design variables and allows the use of industrial-related design parameters like the radius of the leading edge or maximum thickness and location[20]. The author has shown that the B-spline method allows fewer design variables to represent the wing surface. Due to its low number of design variables, gradient-free methods can be implemented. The complexity involved in the CST method prevents the researcher from using it.

Gradient-free methods are generally adopted for unconstrained optimization problems. Additional steps need to be followed to implement constraints. One of them is constraint handling as proposed by Deb and Saha [21]. The author D.J Poole [3] refers to the constraint handling technique proposed by Deb. Additionally, the author introduces the parallel decomposition of the niching DE algorithm tested on benchmark functions and are reliable.

The author D.J Poole [5] introduced a total of 18 benchmark functions to test the DE-

based niching method. Among them, nine are from literature, and remaining are from his work. Along with benchmark functions, different niching methods are also published. The author has shown that, niching methods based on **neighbourhood search** show better results in terms of quick convergence and reliability of stable niches. However, these algorithms require an extensive amount of cost function evaluations. Hence some modification is necessary to implement them on ASO.

2.3 Conclusion

After analyzing over a few historical work, it can be said that further investigation over finding multimodality is necessary. The work presented by Chernukhin involves parameterization of the wing using B-spline surfaces. However, parameterization techniques called free-form deformation (FFD) can be adopted. The FFD is easy to implement. In case if the design variable appears more in number, then reduced-order modelling methods can be implemented, which will reduce the dimension of the problem by a significant amount.

This work focuses on using the niching algorithms to obtain all possible optima. The niching techniques balance the population diversity by modifying selection, mutation, or crossover stages of Canonical DE. Since they construct on DE, the convergence rate is slow. When combined with these algorithms, the GB method will make them become fast, resulting in Hybrid optimization. The solution is expected to converge faster. The DE based parallel niching algorithms provide a better solution to these class of problems. The scale factor F and cross-factor CR have a significant impact on the algorithm's performance, such as the quality of the optimal value and convergence rate. There is still no ideal way to determine parameters[22]. The quality of mesh that is used in D.J.Poole work is super fine. However, with the available computational power, following such a high-quality grid seems complicated.

According to Chernukhin [16], surrogate modelling is challenging to handle when the number of design variables becomes large. The 3D wing optimization with a viscous model not covered in the paper. However, this involves vast resources to optimize. With the available resources, it becomes difficult to handle. Little work has taken place in optimizing the wing with pre-defined winglet shape. The niching techniques involve a considerable quantity of cost function evaluations. So there needs to reduce cost function evaluation by modifying the code. In the end, solving the ADODG case 6 problems with FFD parameterization, combined with reduced-order modelling and the niching algorithms would lead to a new dimension to approach this problem.

Chapter 3

Niching Algorithms

3.1 Background

Aerodynamic shape optimization (ASO) is defined as finding the shape that optimizes a performance quantity subject to drag, geometrical constraints. Much of the work carried out in the past involved finding the single best optimum solution. However, during the design process, identifying other optimal designs may positively impact the cost and performance. Many aerodynamics optimization problems can be unimodal or multimodal. The parallel niching optimization algorithms direct the given multimodal optimization problem to identify the multiple optima in the given design space. This chapter contains a detailed explanation of the differential evolution (DE) based parallel niching algorithms.

The DE algorithms are defined to find global optima for a given scalar objective function $f(\mathbf{x_i})$ where, $\mathbf{x_i}$ is a design variables such that $\mathbf{i} \in [1, \ldots, N]$ and $\mathbf{x} \in \mathcal{R}^D$ with D being problem dimension and N being total population. Also, the DE algorithms are designed for unconstrained optimization problems. However, most of the real-world problems involve some constraints. In such a case, the DE algorithm needs to be modified. These modifications involve introducing a penalty method that results in a class of problems called **Multimodal optimization problem**.

Furthermore, the modifications to the DE algorithm are referred to as **Niching Algorithms**. These techniques will enhance the population's diversity by locating the optimal solution in different parts of the design space. A considerable amount of research has taken place in constraint handling and multimodal optimization using nature-inspired algorithms. Upcoming sections contain a detailed explanation of DE and the modification to algorithms.

3.2 Differential Evolution

DE is a population-based optimization algorithm, where operators for mutation, crossover, and selection act to create a new population from an existing one. Eventually, through a series of iterations, the population will converge on to a globally optimal solution [5]. The DE is suitable for problems with discontinuous objectives, also with a problem having multiple local optima (multimodality). However, DE has slow convergence. Nearly 200 times more functional evaluations are required for Gradient-free methods as compared to gradient-based methods [16].

The DE is a parallel direct search method which utilizes N D-dimensional design vectors $\mathbf{x}_{n,G}$, where n = 1, 2, ... , N and g is the generation ranging between $[1, G]$. N does not alter during the optimization process. The initial population is chosen randomly and should cover the entire design space. As a rule, it is assumed to follow a uniform probability distribution for all random decisions unless otherwise stated. In case a preliminary solution is available, the initial population might be generated by adding normal distributed random deviations to the nominal solution $\mathbf{x}_{nom,0}$. The DE generates another vectors by adding the weighted difference between two population vectors to a third vector. This operation is called *mutation*.

The mutated vector individuals are merged with the individuals of another predetermined vector, the target vector, to generate the *trial vector*. This operation is referred to as "*crossover*." If the trial vector yields a lower cost function value than the target vector, the trial vector replaces the target vector in the following generation. This last operation is called "*selection*". Each population vector has to serve once as the target vector so that N competitions occur in a given generation [2].

The mathematical representation of the DE algorithm is explained as follows.

1. Initialization: Here the initial population N is generated randomly as:

$$\mathbf{x}_{n,g}^{d} = \mathbf{L}^{d} + \mathrm{rand}(0,1)\left(\mathbf{U}^{d} - \mathbf{L}^{d}\right) \tag{3.1}$$

   where $rand(0, 1)$ is a uniformly distributed random number on the interval [0, 1]. L^d and U^d represents the lower and the upper limits of the design space respectively. And, d is dimension of problem which can be $(d \in \{1, \ldots, D\}))$.

2. Mutation: For each individual in the population, corresponding donor vector is generated as follows: initially, the weighted difference is taken between two randomly selected individuals from population,then it is added it to base vector ($\mathbf{x}_{b,G}$). There

are several strategies proposed that define the base vector. If in case DE/best/1 strategy is chosen, then the n-th donor vector is represented using equation 3.2.

$$\mathbf{v}_{n,g} = \mathbf{x}_{b,g-1} + F.\left(\mathbf{x}_{r_1,g} - \mathbf{x}_{r_2,g}\right) \tag{3.2}$$

$$b = \underset{i \in \{1,\ldots,N\}}{\arg\min} f\left(\mathbf{x}_{i,g-1}\right)$$

where, F is the scaling factor ranging between [0, 1]. A higher scaling factor may force the donor individual to fall out of design space. Whereas, the lower value will generate the donor vector close to target individual. A value of $F = 0.5$ is considered to be a good trade off. $x_{b,g-1}$ is the best individual from the previous generation. r_1 and r_2 are uniformly distributed random integers in the interval [1, N] such that $r_1 \neq r_2 \neq b$ and $r_1 \neq r_2 \neq n$. A better representation of mutation stage in 2-D design space is shown in figure 3.1.

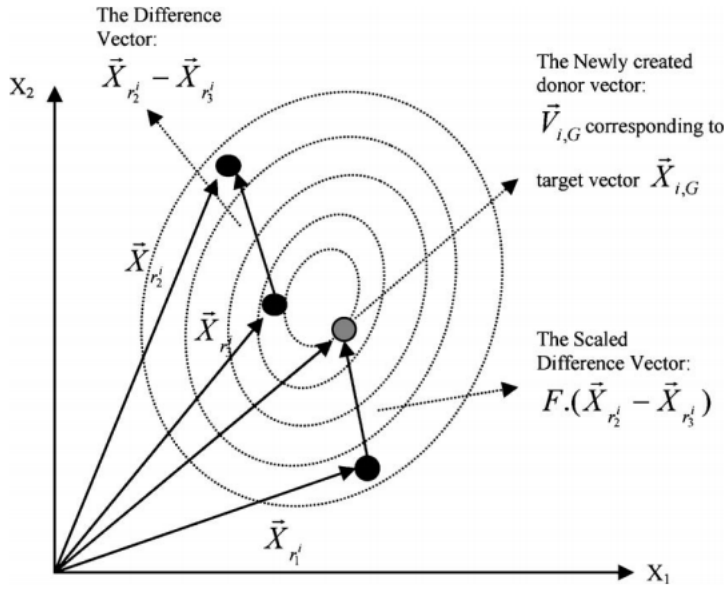


Figure 3.1: 2-D design space representing mutation stage [2].

3. Cross-over: In this stage, elements from the target vector and donor vector are combined in a specific order to get a trial vector ($\mathbf{u}_{n,g}$). This process is referred as binomial crossover. This stage play a vital role in increasing the population's diversity, and is shown in equation 3.3.

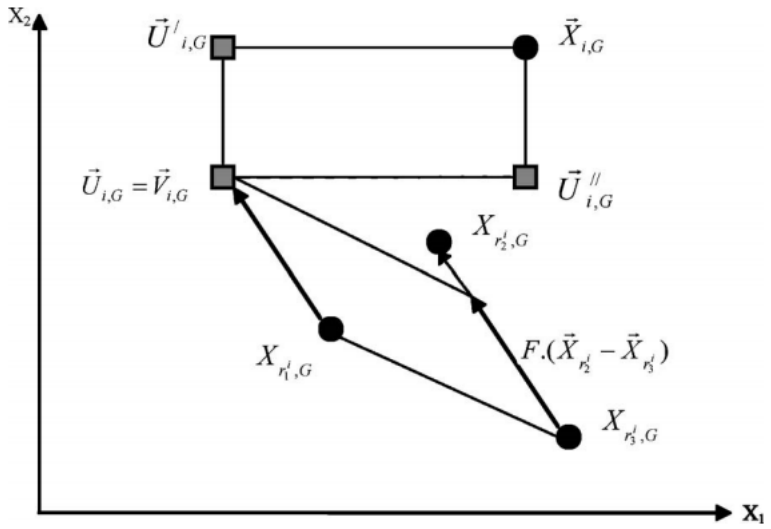


Figure 3.2: Different possible trial vector formed due to Uniform/Binomial crossover between mutant vector and target vector in 2-D design space[2].

$$\mathbf{u}_{n,g} = \begin{cases} \mathbf{v}_{n,g} & \text{if rand } (0,1) \leq CR \text{ or } r_n = d \\ \mathbf{x}_{n,g} & \text{otherwise} \end{cases} \tag{3.3}$$

where r_n is the uniformly distributed random integer in the interval [1, D], and CR is the crossover probability. $CR = 1$ implies there is no diversity, meaning the parent individuals are not carried to the next generation. On the other hand, $CR = 0$ implies that all generated elements are the same as parent elements, so there is no evolution over generations.

4. Selection: The trial individuals and target individuals are compared based on their function values. During the problem with the minimization case, an individual with minimum function value is retained. On the other hand, for maximization problems, an individual with a maximum function value is retained. Equation 3.4 represents the selection phase.

$$\mathbf{x}_{n,g} = \begin{cases} \mathbf{u}_{n,g} & \text{if } f\left(\mathbf{u}_{n,g}\right) \leq f\left(\mathbf{x}_{n,g}\right) \\ \mathbf{x}_{n,g-1} & \text{otherwise} \end{cases} \tag{3.4}$$

5. Stopping condition: The optimization can have single or multiple stopping conditions, like the maximum number of function evaluations, epsilon value, to name

a few. The author D.J.Poole used the maximum number of function evaluations (FEs_{max}) as the stopping condition for all the algorithms.

The algorithm 3.1 depicts the pseudo-code for the DE algorithm.

Algorithm 3.1: DE algorithm

```
Randomly initialise individuals and calculate objective
while FEs < FEs_max do
    for n = 1 → N do
        Perform mutation: equation 3.2
        Perform binomial crossover: equation 3.3
        Calculate objective and constraints of trial vector
    end
    for n = 1 → N do
        Update n-th target vector: equation 3.4
    end
end
```

In the next section, details about mutation strategies are explained.

3.3 DE Strategies

In the DE algorithm, there are different variants for the mutation stage. Some of these are mentioned in the upcoming subsections [23].

3.3.1 DE/rand/1

In this strategy, the mutation stage is carried out by taking single pair of the weighted difference. The individuals involved are randomly selected from the populations, Further, the weighted difference is added to base individual resulting in target vector as shown in equation 3.5.

$$\mathbf{v}_{n,g} = \mathbf{x}_{r_1,g} + F.\left(\mathbf{x}_{r_2,g} - \mathbf{x}_{r_3,g}\right) \tag{3.5}$$

3.3.2 DE/best/1

This method work the same way as DE/rand/1 strategy, except that it generates the individual base vector x_b as in equation 3.6.

$$\mathbf{v}_{n,g} = \mathbf{x}_{b,g-1} + F.\left(\mathbf{x}_{r_1,g} - \mathbf{x}_{r_2,g}\right) \tag{3.6}$$

where,

$$b = \underset{i \in \{1,\ldots,N\}}{\arg\min} f\left(\mathbf{x}_{i,g-1}\right)$$

3.3.3 DE/best/2

This strategy involve evaluating the mutant vector with two weighted (F_1, F_2) differences of vectors which are picked randomly from the population size N, and added to base vector (best) x_b. Equation 3.7 represents the same.

$$\mathbf{v}_{n,g} = \mathbf{x}_{b,g-1} + F_1.\left(\mathbf{x}_{r_1,g} - \mathbf{x}_{r_2,g}\right) + F_2.\left(\mathbf{x}_{r_3,g} - \mathbf{x}_{r_4,g}\right) \tag{3.7}$$

where,

$$b = \underset{i \in \{1,\ldots,N\}}{\arg\min} f\left(\mathbf{x}_{i,g}\right)$$

3.3.4 DE/current to best/2

In this strategy, one of the two weighted differences will take place between the best vector and the vector index for which perturbation is carried out. Equation 3.8 depicts the above strategy.

$$\mathbf{v}_{n,g} = \mathbf{x}_{b,g-1} + F_1.\left(\mathbf{x}_{r_1,g} - \mathbf{x}_{r_2,g}\right) + F_2.\left(\mathbf{x}_{b,g-1} - \mathbf{x}_{r_n,g}\right) \tag{3.8}$$

where,

$$b = \underset{i \in \{1,\ldots,N\}}{\arg\min} f\left(\mathbf{x}_{i,g-1}\right)$$

3.3.5 DE/rand/2

This strategy involve two weighted differences between individuals. These individuals are randomly selected in the given population. Equation 3.9 represents the mutant vector generation using the DE/rand/2 strategy.

$$\mathbf{v}_{n,g} = \mathbf{x}_{r_1,g} + F_1.\left(\mathbf{x}_{r_2,g} - \mathbf{x}_{r_3,g}\right) + F_2.\left(\mathbf{x}_{r_4,g} - \mathbf{x}_{r_5,g}\right) \tag{3.9}$$

where, $r_1, r_2, r_3, r_4, r_5 \in [1, N]$. F_1 and F_2 are the scaling factors ranging between [0, 1].

3.3.6 Comparision between strategies

The author H.Chi [23] mention that the strategies like *DE/best/1*, *DE/best/2* and *DE/current to best/2* can improve the convergence rate of DE, and converge in the local optimal due to usage of the best vector. On the other hand, *DE/rand/2* and *DE/rand/1* relatively enhance the convergence rate of finding the global optimal but have slow convergence speed. Several test functions like Sphere function, Rosenbock's function, Step function, Quartic function, etc. are tested using the Niching algorithm, and results obtained coincide with the actual optimal values. Chapter 7 highlights more on test function results.

3.4 Constraint handling

The feasibility rules published by Deb [24] assist in handling the constraints to the problems. The rules are as stated: when choosing between two locations, if both locations are feasible, the one with the best fitness value wins. If a single location is feasible, then select the same. Otherwise, select the location with the least constraint violations. Mathematically, using domination operator, where, given two locations x_a and x_b, x_b dominates x_a based on the conditions mentioned in equation 3.10.

$$\mathbf{x}_a \prec \mathbf{x}_b \Leftrightarrow \begin{cases} f(\mathbf{x}_b) < f(\mathbf{x}_a) & \text{and} \quad \phi(\mathbf{x}_a), \phi(\mathbf{x}_b) = 0 \\ \phi(\mathbf{x}_b) = 0 & \text{and} \quad \phi(\mathbf{x}_a) > 0 \\ \phi(\mathbf{x}_b) < \phi(\mathbf{x}_a) & \text{and} \quad \phi(\mathbf{x}_a), \phi(\mathbf{x}_b) > 0 \end{cases} \tag{3.10}$$

where ϕ is the constraint violation given by:

$$\phi(\mathbf{x}) = \sum_{i=1}^{p} \max[0, g_i(\mathbf{x})] + \sum_{j=1}^{q} |h_j(\mathbf{x})| \tag{3.11}$$

where $g_i(\mathbf{x})$ represents the i-th inequality constraint value, and $h_j(\mathbf{x})$ represents the j-th equality constraint value.

In DE, these feasibility rules are commonly used in the selection step to determine

whether the trial vector should replace the target vector. Hence at selection stage:

$$\mathbf{x}_n = \begin{cases} \mathbf{u}_n & \text{if } \mathbf{x}_n \prec \mathbf{u}_n \\ \mathbf{x}_n & \text{otherwise} \end{cases} \tag{3.12}$$

3.5 Control variable

There are a few parameters that decide the algorithm progress towards the optimal points. The N, F, and CR are the essential parameters in the DE algorithm. D.J.Poole [2] states that it is not difficult to guess the range of values for these algorithms. According to the author, N is to be between $5D$ and $10D$. A value of F = 0.5 is usually the right choice. Furthermore, for CR probability, a value of 0.1 would work fine. However, the higher the value of CR faster will be the convergence. So it is always the better choice to test the algorithm with a CR value of 0.9.

3.6 Parallel and Sequential decomposition

There are two ways to implement the algorithm, sequential flow, and parallel flow. Taking DE into consideration, the sequential form of DE constitutes the mutation, crossover, and selection phases in sequential order. For each population within a design space, all these phases are carried out in sequential order, as shown in figure [3.3]. However, this will result in an inefficient implementation. Instead, a parallel form of implementation can be followed. Here, the entire population is subjected to mutation, crossover, selection phases at once. Later, the comparison between the populations is carried out.

The most expensive part of the aerodynamic optimization problem is objective function evaluation, which is CFD evaluation. In the case of solving the above problem, the parallel decomposition of algorithm wins over the sequential. Also, each of the population can be thread into single processing resulting in parallel computation of the CFD simulations. The pictorial representation of both parallel and sequential decomposition of the algorithm is shown in figure 3.4.

3.7 Sequential niching algorithm

As mentioned before, there are two ways to implement the DE algorithm. The upcoming subsections explain different sequential niching algorithms. Among these, few of the

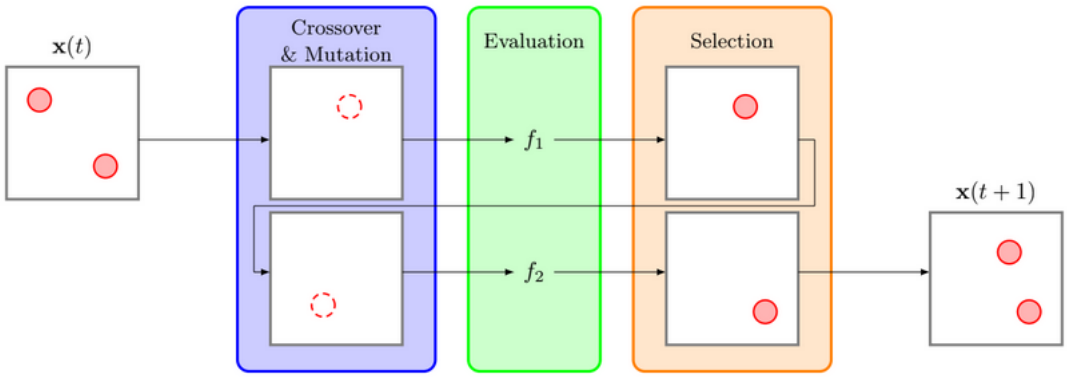


Figure 3.3: Sequential decomposition of algorithm[3].

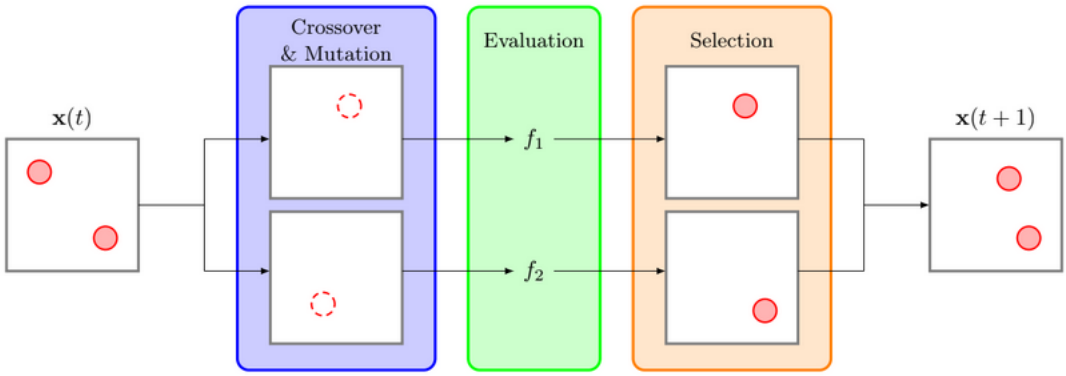


Figure 3.4: Parallel decomposition of algorithm[3].

niching algorithms are recreated in Python and implemented on test functions like Ackley function, Rastrigin function, and Egg holder function. The result coincides with the one published by the author and is discussed in chapter 7.

The algorithm involved in aerospace optimization generally follows the population crowding technique[25, 26], fitness sharing[27, 26, 28], clearing[29], speciation[30, 31], local neighborhoods[32, 33], to name a few. Li *et al.* present a full review of these properties[34]. Following are few of the sequential niching algorithms which are imple-

mented in Python and as follows [5].

- Feasible DE $(\mathrm{fDE}),$ which is based on canonical DE
- Feasible DE using nrandl mutation (fNRAND1)
- Feasible DE using inrand $1/r$ (nearest neighbour with ring network) mutation (fINRAND1)
- Feasible crowding DE (fCDE), which is based on the CDE algorithm
- Feasible neighbourhood-based CDE (fNCDE), which is based on the NCDE algorithm
- Feasible species-based DE (fSDE), which is based on the SDE algorithm
- Feasible neighbourhood-based SDE (fNSDE), which is based on the NSDE algorithm
- Feasible fitness-sharing DE (fSHDE), which is based on the SHDE algorithm

3.7.1 fDE

In this algorithm 3.2, the selection stage is modified as stated by Deb and Saha [21], which uses equation 3.12. Additionally, the mutation stage follows $rand/1/bin$ strategy. Algorithm 3.2 represents the same.

Algorithm 3.2: fDE algorithm

```
Randomly initialise individuals and calculate objective
while FEs < FEs_max do
    for n = 1 → N do
        Perform rand/1 mutation: equation 3.2
        Perform binomial crossover: equation 3.3
        Calculate objective and constraints of trial vector
    end
    for n = 1 → N do
        Update n-th target vector: equation 3.12
    end
end
```

3.7.2 fNRAND1

In the algorithm 3.3, mutation stage is modified by taking the target vector of the n-th individual's nearest neighbour x_{NN_n} as the base vector. The selection stage uses equation [3.12] for feasibility check. Equation 3.13 represents the mutant vector.

$$\mathbf{v}_{n,g} = \mathbf{x}_{NN_{n,g}} + F.\left(\mathbf{x}_{r_1,g} - \mathbf{x}_{r_2,g}\right) \tag{3.13}$$

where,

$$NN_{n,g} = \underset{i \in \{1,\ldots,N\}, i \neq n}{\arg\min} \left\| \mathbf{x}_{n,g} - \mathbf{x}_{i,g} \right\|_2$$

Algorithm 3.3: fNRAND1 algorithm

```
Randomly initialise individuals and calculate objective
while FEs < FEs_max do
    for n = 1 → N do
        Find the nearest neighbour to x_n
        Perform nrand/1 mutation: equation 3.13
        Perform binomial crossover: equation 3.3
        Calculate objective and constraints of trial vector
    end
    for n = 1 → N do
        Update n-th target vector: equation 3.12
    end
end
```

3.7.3 fINRAND1

The fINRAND1 algorithm 3.4 works similarly to that of the fINRAND1 algorithm. However, in the mutation stage, the algorithm uses the target vector of the n-th individual's nearest neighbor within its local neighborhood, x_{INN_n}, as the base vectors. Since it uses an index-based ring neighborhood, this algorithm reduces the computational complexity against the fNRAND1 algorithm. Equation 3.14 represents the same.

$$\mathbf{v}_{n,g} = \mathbf{x}_{INN_{n,g}} + F.\left(\mathbf{x}_{r_1,g} - \mathbf{x}_{r_2,g}\right) \tag{3.14}$$

where,

$$INN_{n,g} = \begin{cases} \underset{i \in \{N,2\}}{\arg\min} \left\| \mathbf{x}_{n,g} - \mathbf{x}_{i,g} \right\|_2 & \text{if } n = 1 \\ \underset{i \in \{N-1,1\}}{\arg\min} \left\| \mathbf{x}_{n,g} - \mathbf{x}_{i,g} \right\|_2 & \text{if } n = N \\ \underset{i \in \{n-1,n+1\}}{\arg\min} \left\| \mathbf{x}_{n,g} - \mathbf{x}_{i,g} \right\|_2 & \text{otherwise} \end{cases}$$

In the selection stage, equation 3.12 is used as feasibility criteria.

Algorithm 3.4: fINRAND1 algorithm

```
Randomly initialise individuals and calculate objective
while FEs < FEs_max do
    for n = 1 → N do
        Find the nearest neighbour to x_n in ring neighbourhood
        Perform inrand/1 mutation: equation 3.14
        Perform binomial crossover: equation 3.3
        Calculate objective and constraints of trial vector
    end
    for n = 1 → N do
        Update n-th target vector: equation 3.12
    end
end
```

3.7.4 fCDE

This algorithm 3.2 uses the standard CDE algorithm but with feasibility selection rules in the selection phase. In the CDE algorithm, the nearest individual x_{u_n} to the trial vector is found. The nearest individual replaces the target vector, which is determined using feasibility rules, as stated in equation 3.10.

$$\mathbf{x}_{u_n,g} = \begin{cases} \mathbf{u}_{n,g} & \text{if } \mathbf{x}_{u_n,g-1} \prec \mathbf{u}_{n,g} \\ \mathbf{x}_{u_n,g-1} & \text{otherwise} \end{cases} \tag{3.15}$$

Algorithm 3.5: fCDE algorithm

```
Randomly initialise individuals and calculate objective
while FEs < FEs_max do
    for n = 1 → N do
        Perform rand/1 mutation: equation 3.2
        Perform binomial crossover: equation 3.3
        Calculate objective and constraints of trial vector
    end
    for n = 1 → N do
        Find the closest individual to u_n
        Update closest individual: equation 3.15
    end
end
```

3.7.5 fNCDE

The fNCDE algorithm 3.6 follows the neighborhood search method. In this algorithm, the trial vector is generated from m nearest individuals to the n-th individual, in the design space. While performing mutation, all three vectors are derived from the m nearest neighbors. Further steps are the same as normal crowding DE.

Algorithm 3.6: fNCDE algorithm

```
Randomly initialise individuals and calculate objective
while FEs < FEs_max do
    for n = 1 → N do
        Find the nearest m individuals to x_n
        Perform rand/1 mutation using the nearest m individuals
        Perform binomial crossover: equation 3.3
        Calculate objective and constraints of trial vector
    end
    for n = 1 → N do
        Find the closest individual to u_n in the entire population
        Update closest individual: equation 3.15
    end
end
```

3.7.6 fSDE

As compared to the above algorithms, the fSDE algorithm is considered to be the most time-consuming in terms of implementation and function evaluation. In this algorithm, initially, the population needs to sort in the ascending order satisfying the feasibility rules. If the individual satisfies the feasibility rules, then they are sorted based on the fitness values. However, any individuals violating the feasibility rules; they will be sorted in ascending order based on the constraint violations. Altogether both the populations are represented under the same variable name. The list is sorted from entire populations with the most feasibility individual at first in the list, and the individual violating at high degree will be placed at the last in the list. Furthermore, species are determined based on the distance from the species seed.

Additionally, the species radius σ is set by the user. Suppose the species has less than m (user-defined) individuals. In that case, the extra individuals are randomly added to the species radius to make all equal sets of individuals around every species. During $DE/rand/1$ mutation, $r1$, $r2$, and $r3$ are uniformly distributed random integers selected

from m individuals of the species representing the n-th individual. Since the population has been increased, only the first N fittest individuals are kept for the next iteration, determined by feasibility rules. The overall algorithm is outlined in algorithm 3.7.

Algorithm 3.7: fSDE algorithm

Randomly initialise individuals and calculate objective
while $FEs < FEs_{max}$ **do**
  Generate species: algorithm 3.8
  for $n = 1 \rightarrow N$ **do**
    Perform rand/1 mutation using individuals within the species of n-th individual
    Perform binomial crossover: equation 3.3
    Calculate objective and constraints of trial vector
    If trial fitness is same as its species seed, then randomly generate new trial vector
  end
  for $n = 1 \rightarrow N$ **do**
    Update n-th target vector: equation 3.12
  end
  Compare individuals using feasibility rules and keep N fittest individuals
end

Algorithm 3.8: fSDE species generation algorithm

Sort individuals based on feasibility rules
Sorted individuals are assigned to possible candidate solutions
First species seed is best candidate solution-remove that solution from candidates
for $n = 1 \rightarrow N$ **do**
  for $s = 1 \rightarrow$ *number of species* **do**
    if *n-th candidate entry is not empty and is less than r_s away from s-th seed* **then**
      Solution is not a new seed
      Note solution is in s-th species
  end
  if *n-th candidate entry is new seed* **then**
    Increment number of species
    Store n-th candidate solution as seed and remove from list of candidates
end
for *s = 1 $\rightarrow$ number of species* **do**
  If the s-th species has less that m individuals, randomly generate new individuals within radius of species seed
end

3.8 Parallel niching algorithm

The parallel niching algorithm involves performing mutation to all individuals, then subjecting all individuals to cross-over phase, further to selection phases. All these phases are performed in separate loops. Furthermore, it results in separating the workload and performed in different threads of the CPU. The bottleneck in the wing shape optimization is objective function evaluation (CFD simulation). All populations can be subjected to the individual thread for CFD simulation to obtain better performance and time-efficient. For example, the fNRAND1 algorithm with parallel decomposition is highlighted in the algorithm 3.9.

Algorithm 3.9: Parallel decomposition of fNRAND1 algorithm [3].

```
Randomly initialise individuals and calculate objective
while FEs < FEs_max do
    for n = 1 → N do
        Find the nearest neighbour to x_n
        Perform nrand/1 mutation: equation 3.13
        Perform binomial crossover: equation 3.3
    end
    for n = 1 → N do
        if n = procid then
            Objective of n-th trial vector
    end
    for n = 1 → N do
        Update n-th target vector: equation 3.12
    end
end
```

3.9 Parameter Tuning

From the literature review, a higher number of generations will result in tremendous computational resources. The number of runs on each function set to be fifty for all the niching algorithms, while the population N is suggested as $N = 40\sqrt{D.G}$, where D is the dimension of problem, G is maximum generation opted. Each niching algorithms perform better at a specific range of mutation factor and a crossover probability. The author D.J.Poole [5] suggested the values after considering all these factors and is mentioned in table **??**.

	Algorithm							
Param	fCDE	fDE	fINRAND1	fNCDE	fNRAND1	fNSDE	fSDE	fSHDE
C R	0.1	0.1	0.9	0.9	0.9	0.1	0.9	0.1
F	0.1	0.9	0.9	0.9	0.9	0.9	0.9	0.1
σ							10 %	0.1 %
m				10		10	20	

Table 3.1: Parameter tuned for different niching algorithms[5].

3.10 Conclusion

The two neighborhood-based algorithms (fNSDE and fNCDE) have the fastest overall convergence rates. However, fNSDE has little difficulty finding all of the optima quickly, and the niches formed appear to be **unstable**. For example, fNSDE locates all of the optima rapidly and then cannot maintain these niches. However, for function F14 (*modified Rastrigin function*), the majority of optima are located. However, the niches are unstable, resulting in global optima. On the other hand, fNCDE located optima at a slightly slower rate than fNSDE but is able to maintain stable niches. The average convergence speed of fDE is lower than fCDE, fINRAND1, fSDE and fSHDE, which demands for greater complexity as compared to fDE.

In terms of algorithmic development, fINRAND1 and fNRAND1 require the least effort to develop, with only a few extra lines of code added and no change in code logic. On comparing **fDE** and **fNRAND1**, the **fNRAND1 is superior** in terms of convergence speeds, peak ratios, and success rates. The fNRAND1 involves finding the nearest neighbor individual, increasing the algorithm complexity by up to $O(N^2)$ per iteration, compared to $O(N)$ for fDE.

The fDE inherently has difficulty maintaining stable niches and appears to converge to global optima or single optima. For function F14 (Himmelblau function), it appears that fDE has four optimal. However, with additional generations, all optimal points merge to a single point.

During higher functional evaluation, the comparison of PR (Peak Ratio) [5] points to three unique best-performing algorithms: **fCDE, fNCDE, and fNRAND1**. If the function evaluation is of lower quantity, the explicit winner is **fNSDE**. As discussed before, fNSDE is outstanding at creating niches. Higher the algorithm evolves, the greater are the chance of niches break down. The fNRAND1 is simple to implement, whereas its neighbor, fINRAND1, has an excellent overall performance. The more complex algorithm wins against,

the fewer one. However, the trade-off in complexity is a call for an hour.

The results of the test function evaluation are mentioned in chapter 7. Furthermore, the respective test function equations will be appended in the appendix A.3.

Chapter 4

Parameterization

4.1 Background

Aerodynamics shape optimization (ASO) is the process that involves optimizing an aero-shape (wing, airfoil, fuselage, so on) under given constraints. The objective function can be drag, lift, aerodynamic efficiency, and so on. Representing the wing requires a tedious job of plotting the actual surface. The wing surface is to be parameterized using parameterization schemes. There are multiple ways to parametrize the wing, which include mesh-points, B-splines, cubic splines, free-form deformation. Representing the wing using the parameterization reduces the dimension of the design space.

A shape parameterization system typically involves the coupling of a CAD tool and a grid generator. Every time the shape design variables are changed, a new grid must get generated without human intervention. For complex problems and especially involving RANS models, fully automatic grid generation can be difficult or impossible. An alternative approach is to use a reference shape, usually the starting design, and deform this shape by various techniques. In the present work, the FFD parameterization technique is selected because it is easy to implement and gives more flexibility while perturbating control points. The subsequent sections cover the detailed approached on FFD.

4.2 Problem statement

Before discussing the parameterization and construction of the wing, the problem statement and constraints will be discussed initially.

The objective is to minimize C_d, subject to constraints as mentioned.

C_l = 0.2625

C_{M_x} - 0.1069 $\leq$ 0

-3.0 $\leq$ AoA $\leq$ 6.0

0.1 $\leq$ thickness $\leq$ 0.15

0.7 $\leq$ chord $\leq$ 1.4

2.7 $\leq$ semi span $\leq$ 3.3

The AoA and C_l constraint are satisfied by the CFD solver (SU2). The root bending moment (C_{M_x}) constraint is implemented into the Python code using feasibility rules proposed by Deb and Saha [21]. The shape constraints like thickness, chord, semi span of a wing are satisfied using the Python code coupled with FFD control points displacement, which will be discussed in further sections.

4.3 Free Form Deformation

FFD is a process by which shape change can be made to geometry by manipulating the location of points which are related to the geometry. Soderberg and Parry first introduced this technique in 1986. It can be correlated to the Bezier curves of parameterization of given curves. Before going directly with wing parameterization, a short introduction on the FFD applied to 2D geometries will be presented.

As mentioned before, FFD manipulates the geometry by attaching it to bezier curves. And these curves are defined by the grid of control points defined as $u \in [0, 1]$, $t \in [0, 1]$.The Bernstein polynomials govern the control points' influence on the space within the grid. The effect of perturbation reduces with the distance between mesh points from the control point. However, it is observed that moving the control points within the grid will have less effects. Figure 4.1 represents the same.

Equation 4.1 represents the mathematically form of 2-D FFD box.

$$\mathbf{X}(u,t) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbf{a}^{(i,j)} f_i(u) g_j(t) \tag{4.1}$$

where **X**(u, t) is the Cartesian coordinates of the new, deformed location of a point at (u, t). And, $f_i(u)g_j(t)$ represents the Bernstein polynomial, $a^{(j,i)}$ represent the control point being displaced. To make this work, define a shape in parametric form and apply equation 4.1. However, care should be taken to inscribe the entire object, subjected to deformation, into the FFD box; else, only the inscribed part will be deformed.

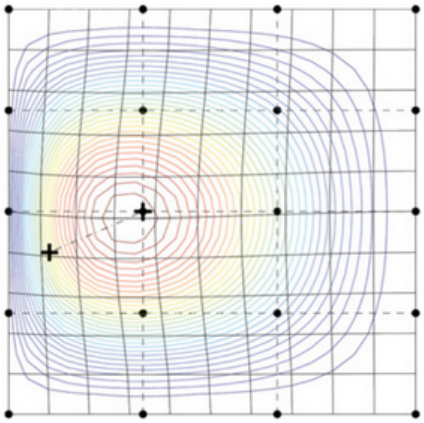


Figure 4.1: FFD control point grid with the a(2,3) control point displaced. The solid lined grid show how mesh points are shifted according to the influence of the control points[4].

Extending further, same concept can be implemented to 3D problems. Every object point **X** has (s,t,u) coordinates in the parallelpiped coordinate region[35].

$$\mathbf{X} = \mathbf{X}_0 + s\mathbf{S} + t\mathbf{T} + u\mathbf{U} \tag{4.2}$$

Let $\mathbf{P}_{ijk}$, i = 0, . . . , l, j = 0,. . . , m, k = 0,. . . , n are control points on the lattice. Equation 4.3 represents the mathematical form of 3D FFD box equation.

$$\mathbf{X}(u,t,s) = \sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{k=0}^{p} \mathbf{a}^{(i,j,k)} f_i^m(u) g_j^n(t) h_k^p(s) \tag{4.3}$$

where Bernstein's polynomial $f_i^m(u)$ is defined as,

$$f_i^m(u) = \frac{(m)!}{(i)!(m-i)!} u^i (1-u)^{m-i} \tag{4.4}$$

The new position of a point $\mathbf{X}_{def}$ is computed by:

$$\mathbf{X}_{def} = \sum_{i=0}^{l} \begin{pmatrix} l \\ i \end{pmatrix} (1-s)^{l-i} s^i \left[\sum_{j=0}^{m} \begin{pmatrix} m \\ j \end{pmatrix} (1-t)^{m-j} t^j \left(\sum_{k=0}^{n} \begin{pmatrix} n \\ k \end{pmatrix} (1-u)^{n-k} u^k \mathbf{P}_{ijk} \right) \right]$$

As compared to 2D FFD equation, 3D FFD requires an additional set of Bernstein's polynomial and a 3D grid points. Figure 4.2 illustrates the influence of the control point movement on the sphere, when one of the control points is displaced away. Similarly,

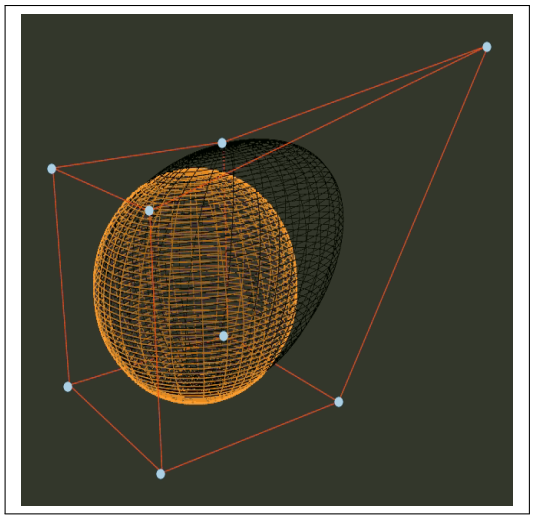


Figure 4.2: Influence of the control point over the sphere body with $a^{(m,n,p)}$ as $2\times2\times2$ control points.

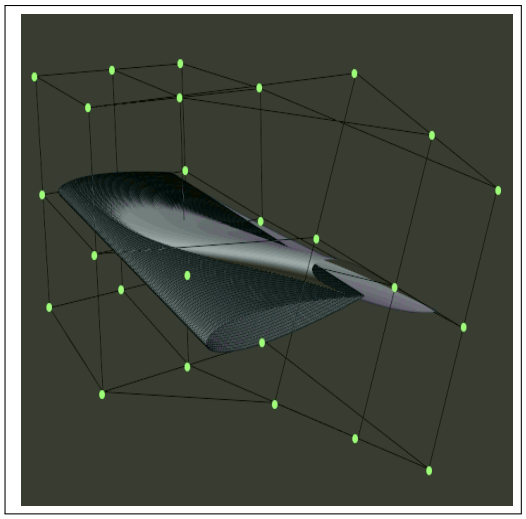


Figure 4.3: FFD of a wing shape, with the control points on the face of the wing-tip end are rotated and translated.

another example mentioned in figure 4.3 represents the wing shape deformation (preciously surface mesh deformation) due to wingtip control points being rotated and translated.

In figure 4.3, the wing surface mesh is parameterized using the FFD with 27 control points. However, this work contains 60 control points to deform a wing surface mesh. Full details about the distribution of control points will be explained in the upcoming chapter. Figure 4.4 to 4.9 represents the perturbed wing which are obtained after implementing FFD box and perturbing the control points of FFD box.

Figure 4.4: Span view of wing 1.

Figure 4.5: Iso view of wing 1.

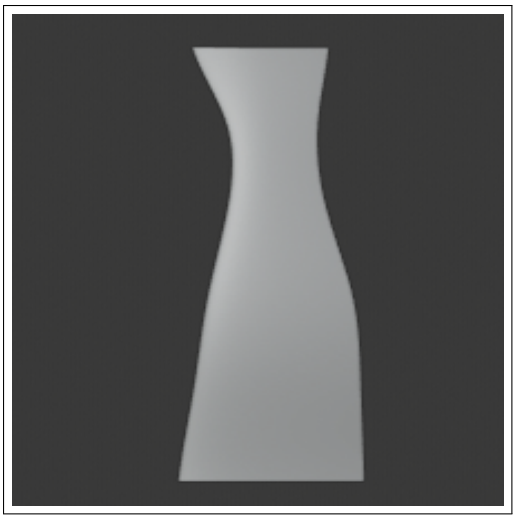


Figure 4.6: Span view of wing 2.

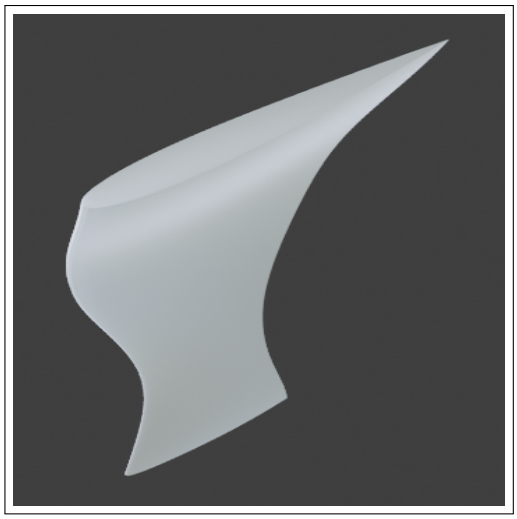


Figure 4.7: Iso view of wing 2.

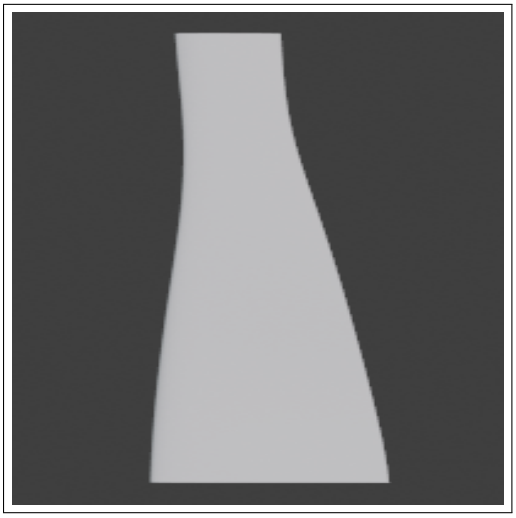


Figure 4.8: Span view of wing 3.

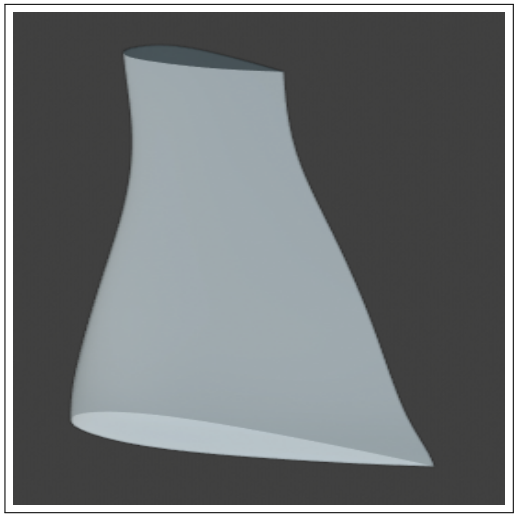


Figure 4.9: Iso view of wing 3.

4.4 FFD implementation algorithm

Evaluation and implementation of the FFD equation are explained in the algorithm 4.1. Input to this algorithm will be parametric coordinates of object surface mesh. Output of the algorithm will be the cartesian coordinates of deformed surface mesh.

Note that function $binom$ is the computation of binomial number and function $pow(a, b)$ is power a^b.

Algorithm 4.1: FFD algorithm

Input: parameters (s,t,u)
Output: new coordinates of the point **X** for parameters (s,t,u)
for *i=0, . . . , l* **do**
  for *j=0, . . . , m* **do**
    for *k=0, . . . , n* **do**

$$X = X + \text{binom}(l,i) * \text{pow}(1-s, l-i) * \text{pow}(s,i) * \text{binom}(m-1,j) * \text{pow}(1-t, m-j) * \text{pow}(t,j) * \text{binom}(n,k) * \text{pow}(1-u, n-k) * \text{pow}(u,k) * \boldsymbol{P}_{ijk}$$

    end
  end
end

4.5 Reduced-order model

After considering the effort involved in building the FFD box, the next issue to address is the algorithm's credibility over solving the higher dimension problem. At present, the problem dimension is 125. The DE based niching algorithm is not tested for the 125 dimension problem, resulting in reconsider to reduce the problem dimension. The preferred way to carry out the dimension reduction is reduced-order modeling.

Complex geometries encountered in aerodynamics typically require a large number of grid points to define the surface boundaries. For example, the representation of the standard wing involves a million+ grid points. Apart from this, if the number of random input variables (geometry variables) is not restricted, then the computational cost would increase linearly. Figure 4.10 represents the time taken by the algorithm versus the dimension of the problem. The time taken by the algorithm increases linearly with the dimension of the problem.

To reduce the additional computational cost, reduced-order modeling of the geometry is necessary. Once the dimension of problem is reduced to a manageable level, optimization of the NACA0012 wing can be implemented.

Reduced-order modeling is a highly complex topic and has found many applications in various fields of engineering. Typically reduced-order modeling methods can be categorized as parametric and non-parametric.

Parametric methods require a priori knowledge about the system to be reduced, whereas the non-parametric does not require any prior knowledge of the geometric uncertainty. One of the most important methods is Principal Component Analysis (PCA). PCA is also called

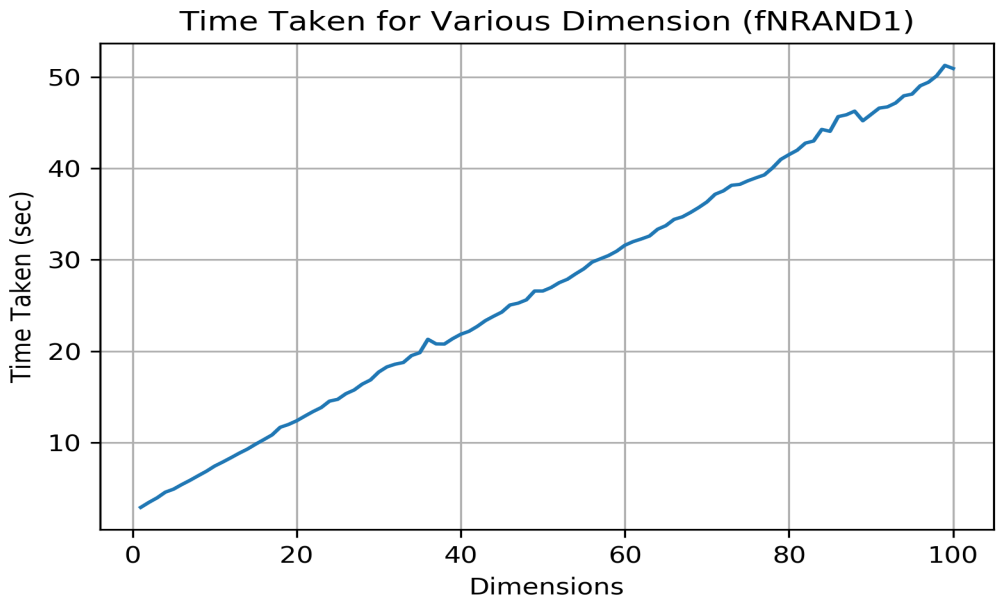


Figure 4.10: Computational power varies linearly with dimension.

by different names like Proper Orthogonal Diagnolisatopn (POD), the Hotelling transform, and the discrete Karhunen-Loeve transform (KLT).

PCA identifies the mutually uncorrelated basis vectors in the diminishing order of their importance. Also, it uses only the second-order statistical information (variance and covariance). All the higher-order information is discarded, resulting in a computationally efficient analysis.

In work published by Garzon [36], the successful reduced-order model can be derived using PCA for real fan blade measurements without any prior knowledge of the sources causing variations in the data. Furthermore, author reported using only five leading eigenmodes to capture 99% of the scatter energy-measure of geometric variability.

Chapter 6 address the implementation of PCA to a given problem.

4.6 Conclusion

Understanding the concepts of FFD and PCA appears to be complicated. However, when it comes to implementation, both the methods are straight forward. For PCA, there are standard modules available in Python. Implementing the FFD concept to a specific problem is a bit intricate. However, many articles are available to assist the user. The decision to select the number of control points in the FFD box plays a crucial role in obtaining

various shapes of wings. The fewer number of control points will result in less significant wing shapes. At the same time, higher control points will result in a higher-dimensional problem, and the algorithm is not tested for higher-dimensional problems. A clever decision over the number of control points needs to be selected, following which PCA is implemented. Based on the total scatter energy E, a decision to select design variables is chosen. Generally, for aerospace applications, E of more than 85% is considered a good choice. Based on this (leading principal modes), the corresponding decision variables are selected. A detailed explanation about the FFD and PCA implementation to the problem of interest will be explained in chapter 6.

Chapter 5

Mesh generation and CFD solver

In a recent decade, the capability of computational analysis and optimization has made considerable improvement, which made many researchers rethink solutions to real-world problems. One such problem addressed here is the optimization of wing surface geometry. There requires several software packages and link them using some tools to address these problems. In the entire process of wing shape optimization, the bottleneck is found to be a CFD solution. More importance needs to be given to minimize the time taken by the solver. The SU2 solver is chosen based on its ease of availability and other constraint involved. The solver is initially tested on sample wing shape (NACA 0012 wing) for its reliability, and results seem positive. Also, the SU2 solver is open-source. Hence, more flexibility will be available to tweak the code. In this chapter, the emphasis is given to implement the SU2 solver and Pointwise (meshing software).

5.1 Mesh generation

The work presented in this thesis involves the use of proprietary software called Pointwise for mesh generation. This software provides the flexibility of using well-known scripting languages like python and tcl (tool command line) for mesh generation, which is the crucial feature that makes it possible for 3-D shape optimization. Also, it exports the mesh file in su2 format. More information about the mesh size will be mentioned in chapter 7.

In this work, after obtaining the perturbed wing surface coordinates in Plot3D format, it is subjected to volume mesh generation. A glyph script is constructed for volume mesh generation, and it can be accessed using the link mentioned here, *pointwise glyph script*.

The script initially imports the perturbed wing coordinates (*.x3d), which was generated using the principal components of the sampled wings. It is known that winglets will reduce the induced drag without actually increasing the aspect ratio of the wing. Thereby,

the cumulative lift is increased. An attempt is made to club the winglet formation before the wing is imported into meshing software. However, this method faced difficulty obtaining the winglet, as winglets need to be adaptive with the wingtip section. There cannot be a universal winglet that fits all generated perturbed wings inside the generation. Altogether, the winglet creation needs to be automated based on the shape of the wingtip section of a given wing.

After importing the perturbed wing to pointwise using glyph script, the next step is to make the script identify the wingtip section. There are several ways, namely, identifying the extreme points of a connector which connects the wing root section and tip section or identify by connector names.

5.1.1 Winglet creation

This work follows identifying the wingtip section curve by connector name (in this work, it was con-2). The connector (con-2) is split at its leading edge point, which is identified by finding the minimum coordinate value in the curve (con-2) in the chord direction. With this, the wingtip curve is split into two halves (con-2-split-1,con-2-split-2) at the identified minimal point. Figure 5.1 represents the same.

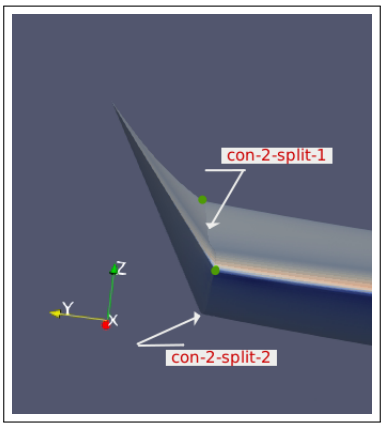


Figure 5.1: Wingtip connector split into two halves.

Then, calculate the dimension (number of mesh points) of either of the split curves. To construct the winglet, initially, the winglet curve is to be constructed. A minimum of three points is required to define a curve. Among them, two points are the end of the wingtip curve (con-2), whereas the third point (also called a shoulder point) coordinate is evaluated as follows,

- Along chord direction (x-coordinate): The weighted average of the x-coordinates of

extreme points in the wingtip section curve. However, weight is user-defined. A weight of 25% and 75% are provided to extreme x-coordinates of the curve (con-2-split-1).

- Along span direction (y-coordinate): The y-coordinate of the shoulder point will be 2% of the given wingspan, that is, $1.02s$, where s is span length.

- Along thickness direction (z-coordinate): The z-coordinate of the shoulder point is crucial. A wrong calculation of this coordinate will lead to an unrealistic winglet. In this work, the slope of the curve (con-3) connecting the wingtip and wing root section is evaluated. Further, the z-coordinate is obtained by the method of extrapolation, using the slope obtained and the 2% of span length(s).

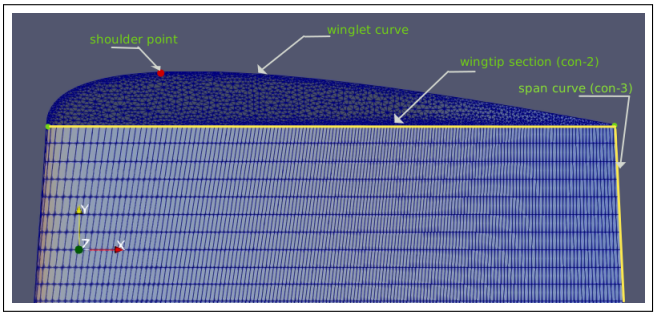


Figure 5.2: Winglet frontview.

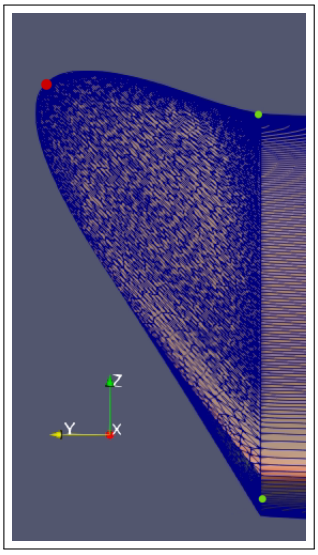


Figure 5.3: Winglet sideview.

Figure 5.2 depict the same. The red color dot represents the shoulder point. A curve is generated using three points is named as a winglet curve. The dimension of the winglet curve is set equal to either of con-2-split-1 or con-2-split-2. An unstructured mesh is generated between the winglet curve and the wingtip section curve (con-2). The outcome is shown in figure 5.3. Green colored dots represent the extreme mesh points along the wingtip section.

5.1.2 Volume mesh generation

Once the winglet generation is completed, the surface mesh is subject to volume mesh generation. In this work, the T-Rex method is implemented to generate the volume mesh. This method was introduced in Gridgen in the year 2007 and has seen several improvements continuously. T-Rex generates hybrid meshes that resolve boundary layers, wakes, and other phenomena in viscous flows by extruding layers of high-quality, high aspect ratio tetrahedra that can be post-processed into stacks of prisms. The algorithm includes tools for optimizing cell quality and avoiding collisions of adjacent layers of cells. More on this can be found in the link mentioned here, *T-Rex*.

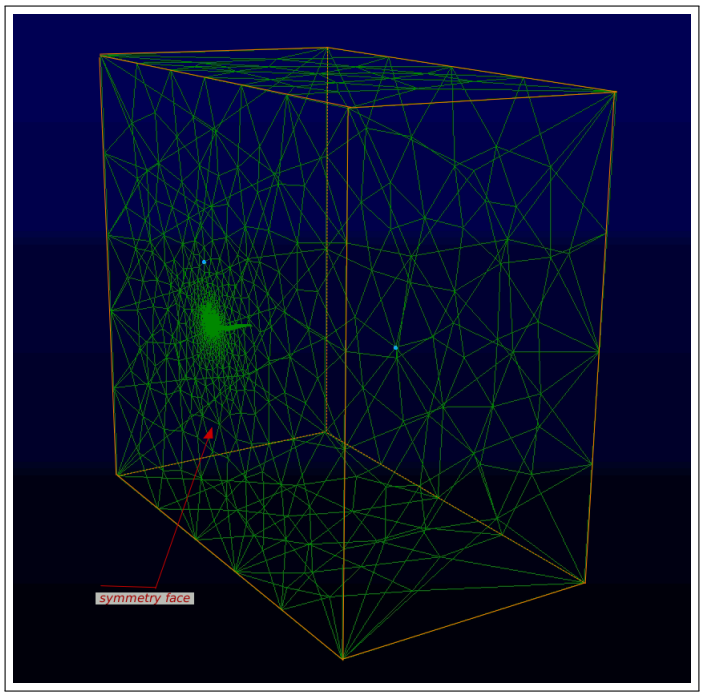


Figure 5.4: T-Rex mesh

The T-Rex method involves setting the boundaries around the wing surface. The literature shows that a scale factor of 15 units (1 unit = 1 chord length) would be sufficient to cover all the wake region, and the far-field boundary surface would look similar to the one shown in figure 5.6. A distance of 15 units is available on both the leading edge side, trailing edge side, and the span direction. A growth factor of 1.2 is set, and the T-Rex mesh generation is initialized. The outcome of the T-Rex mesh is shown in figure 5.4.

The number of mesh points in the T-Rex depends on the wing surface mesh points, the growth rate of the mesh, and the full layers selected. This work contains no full layers

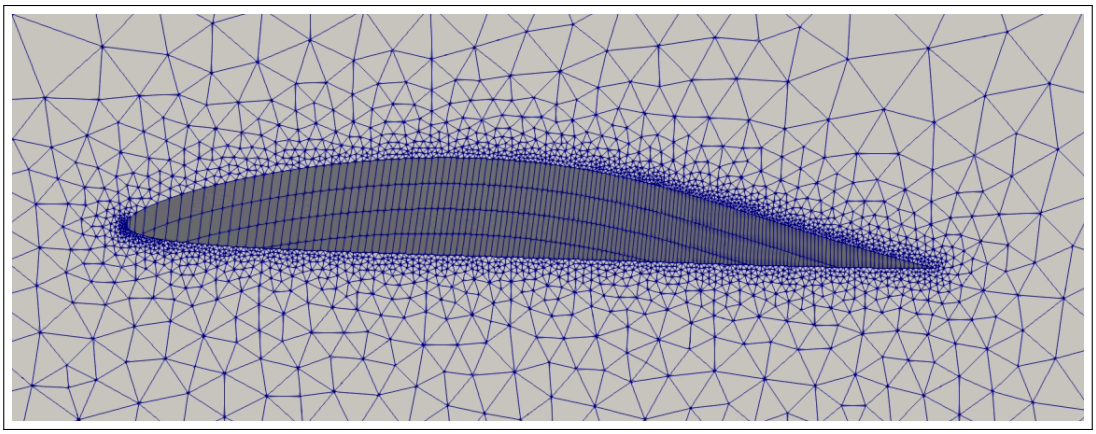


Figure 5.5: Mesh around one of perturbed wing sectioned at symmetry plane.

and has a single block mesh. In the previous work, researchers attempted to implemented multi-block mesh to these problems. However, while addressing these problems, setting up the multi-block mesh seems intricate. Moreover, building the script which creates the multi-block mesh to all perturbed wings is a difficult task. A mesh generated over the airfoil section at symmetry is shown in figure 5.5.

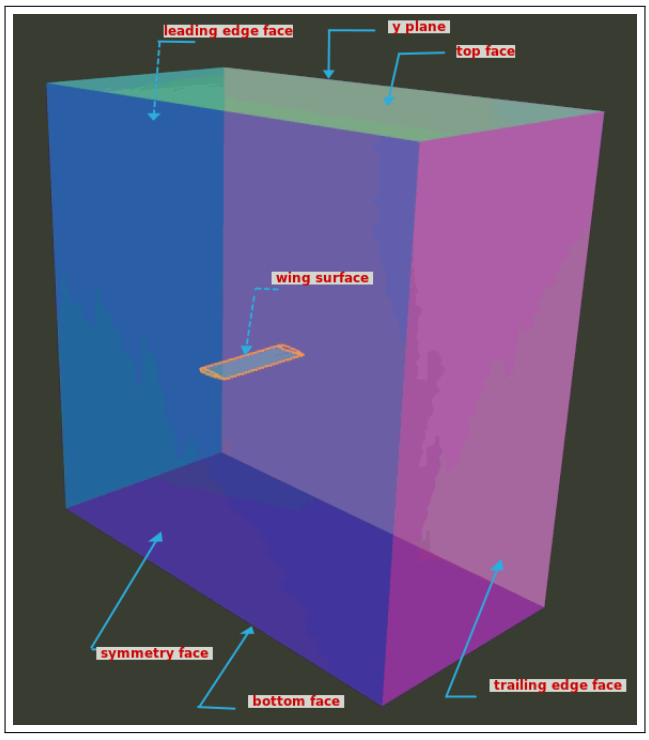


Figure 5.6: Far field boundaries.

Now, each of these far-field faces is assigned a specific name that will assist the solver

in interpreting and assigning the boundary condition values. Table 5.1 refer the same.

General values	Boundary surface value
Leading edge face	XNORMAL_FACES
Trailing edge face	XNORMAL_FACES
Top face	ZNORMAL_FACES
Bottom face	ZNORMAL_FACES
Y-plane	YNORMAL_FACE
Symmetry face	SYMMETRY
Wing surface (including winglet)	WALL

Table 5.1: Boundary surface values

After setting the boundary surface name, the volume mesh is exported in the su2 format (*.su2). However, the numerical values for boundary surfaces will be assigned in the solver configuration file, which is explained in the upcoming section.

5.2 CFD solver

After the check over a few CFD packages like CFD++, ANSYS, and SU2, it is decided to use the SU2 solver for the CFD simulation. The reason for the choice is that the SU2 solver is an open-source project and can be installed in clusters without admin permissions. Other CFD packages require licenses, and limited licenses may cause license error when other users are using it.

The SU2 suite is an open-source collection of C++ based software tools for performing Partial Differential Equation (PDE) analysis and solving PDE-constrained optimization problems. The toolset is designed with Computational Fluid Dynamics (CFD) and aerodynamic shape optimization in mind. More information about solver can be found in the link mentioned here, *SU2 solver*.

The sample script of solver configuration is mentioned here, *solver configuration file*. The CFD solver setup is straight forward. The variable called MACH_NUMBER is used to set up the free stream Mach number. A similar approach is carried to other variables too. Since the SU2 solver provides more features for ASO, one such feature optimizes the lift coefficient based on the change in the angle of attack. In this work, the problem demands a lift constraint of 0.2625, and the value can be optimized with this help of solver alone.

The template contains the variable suffix called MARKERS, which refers to boundary condition value, as mentioned in table 5.1. The solver output contains a vast amount of

aerodynamic data. Extracting the required data from the output file makes it possible to optimize the wing surface and assist in moving to the next generation. The solver also offers the output in both data format (*.csv) and the visualization format (Paraview, Tecplot). It even has the feature of restarting the solution. However, this work does not require the use of restarting the solution.

5.3 Boundary conditions

Equation 5.1 represents the 3-D inviscid fluid flow governed by the Euler equation.

$$\frac{\partial \mathrm{Q}}{\partial t} + \frac{\partial \mathrm{E}}{\partial x} + \frac{\partial \mathrm{F}}{\partial y} + \frac{\partial \mathrm{G}}{\partial z} = 0 \tag{5.1}$$

where,
Q is a vector of conservative flow variables, **E**, **F**, and **G** are convective flux vectors.

Table 5.1 represents the boundary surface values shown in figure 5.6. Free stream flow direction across the XNORMAL_FACES, and there exist no crossflow across ZNORMAL_FACES and YNORMAL_FACE. Both SYMMETRY and WALL surfaces are assigned to Euler wall conditions (no-slip condition). The XNORMAL_FACES, YNORMAL_FACE, and ZNORMAL_FACES are farfield boundaries. The CFD solver is set at a Mach number of 0.4, free stream pressure of 1 atmosphere, and at free stream temperature of 288K. The initial angle of attack is set to be 3^0. However, as mentioned, the lift coefficient constraint is achieved by perturbing the angle of attack. From trial and error, it is noted that the CFD iteration of 3000 would be sufficient to converge the solution to the residual order of -8. More on this is mentioned in chapter 7.

5.4 Job submission

Optimization is a computationally expensive process. A proper computation facility is required to carry out such a process. In this work, during the testing phase of optimization codes, the work is carried out in the workstation, with ten cores (20 CPU). However, this is noted that this will not be sufficient to submit the entire optimization code. Then the work is shifted to the HPC cluster, where sufficient resource was available.

Every CFD simulation is a time consuming expensive computational process. As a result, the handling of each simulation (as called a job) is necessary. A scheduling package called SLURM (Simple Linux Utility Resource Management) is used to address this. More

on this can be found in the link, *SLURM*.

A sample template used for job submission is shown here, *slurm script*. In the initial days of the project, it was planned to submit each CFD simulation as an individual job to the compute node using the SLURM script. However, the restrictions did not allow this to take place. With further modifications to the python optimization code, each generation is submitted as an individual job. Every job demands 20 CPU (population size is 20), and each CPU will carry out the CFD simulation on each perturbed wing in a given generation. This process repeats for the entire set of generation cycles.

5.5 Conclusion

In this chapter, detailed information about the mesh generation (particularly volume mesh) is explained. The creation of a winglet was a tedious job and involved a considerable amount of time. Setting up the boundary values are straight forward. The mesh quality, inlet boundary conditions and free stream conditions are not explained here. They will be addressed in chapter 7. The selection of CFD solver and the reasons for its choice is mentioned. Finally, the way each simulation is fired into compute nodes is mentioned. For all the work mentioned above, proper scripting is involved. The template scripts referring to each of them are linked in the respective sections.

Chapter 6

Methodology

In the previous chapters, extensive details over the parameterization of the wing surface, CFD solver, and niching algorithms are covered. This chapter deals with the flow of data from the initial step of creating NACA 0012 airfoil as baseline geometry to the end that is obtaining the optimized wing surfaces. More emphasis is given on the implementation of the FFD box, SVD method, use of glyph script for meshing in pointwise, use of SU2 solver, and way to submit jobs into the cluster. Figure 6.1 represents the flowchart mentioning the steps involved in obtaining perturbed control points. Every segment in the flowchart will be explained in detail in the subsequent sections.

6.1 Generation of NACA0012 wing surface

This work starts with creating the baseline NACA0012 airfoil using standard equations and extruding it in the third dimension to get the wing surface. The detailed steps involved in this process are explained here.

Constructing the NACA four digit airfoils starts with finding the camber line distance (y_c), and the thickness (y_t) of airfoil upper and lower surfaces from the chord line. After that appending both the values to get the coordinates of the upper and lower surfaces of the airfoil. Since the airfoil of interest is symmetric, both the camber line and chord line coincide, resulting in y_c equal to zero.

Equation 6.1 represents the formula to obtain the half-thickness of symmetric NACA four-digits airfoil.

$$y_t = 5t \left[0.2969\sqrt{x} - 0.1260x - 0.3516x^2 + 0.2843x^3 - 0.1015x^4\right] \tag{6.1}$$

where,

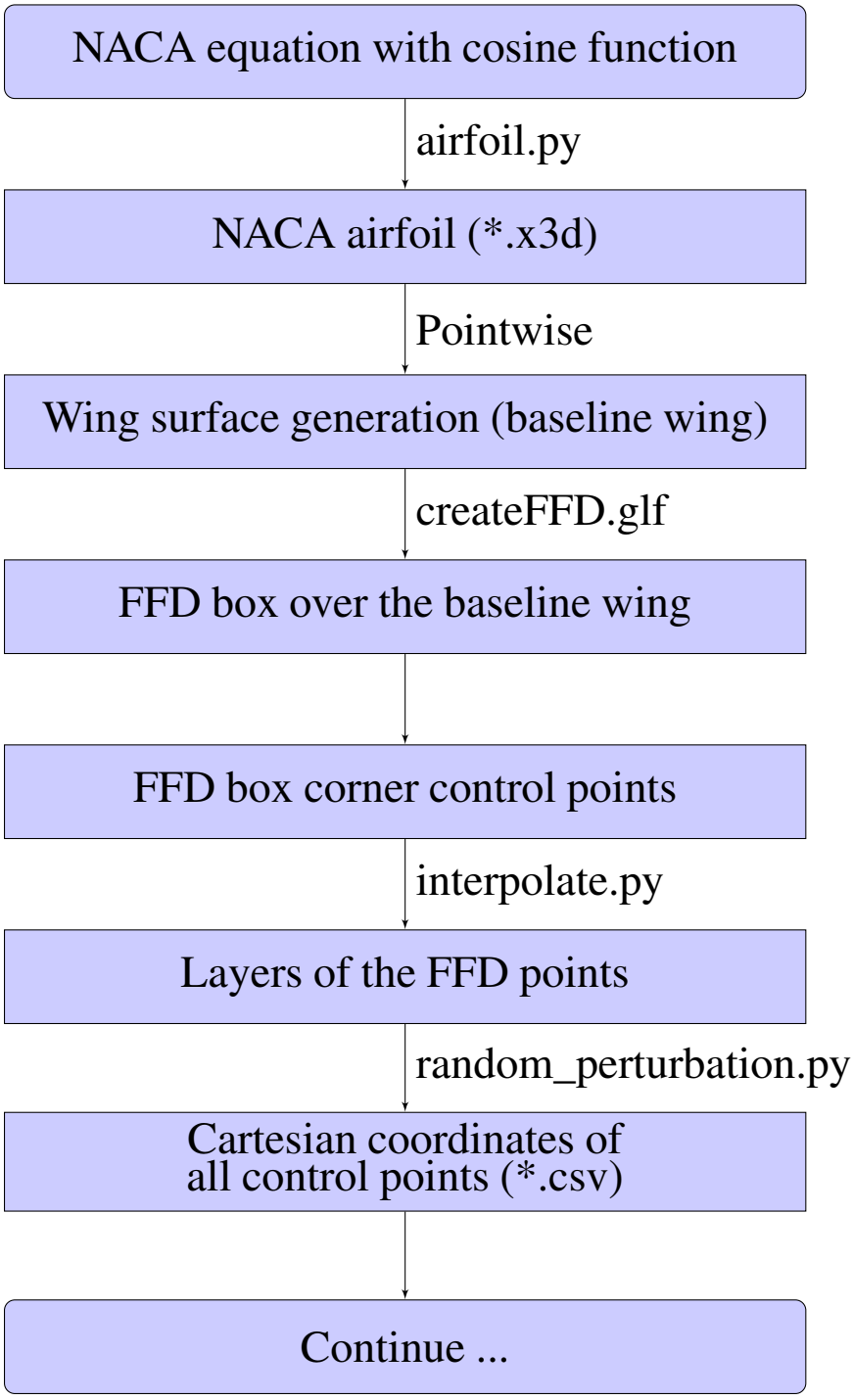


Figure 6.1: Flowchart representing the steps involved in generating the perturbed control points

x is position along the chord from 0 to 1.
y_t is the half thickness for a given x.
t is the maximum thickness represented in terms of chord.

Equation 6.1 will result in the blunt airfoil at the trailing edge. If a sharp trailing edge is required, one of the coefficients should be modified to zero their sum. Modifying the last coefficient to -0.1036 will result in a small change in the overall shape of the airfoil, and a sharp trailing edge is obtained.

Let (x_U, y_U), and (x_L, y_L) presents the upper and the lower airfoil surface coordinates respectively. Since the airfoil is symmetric, y_c is equal zero. This results in the airfoil coordinates as shown in equation 6.2.

$$\begin{aligned} x_U &= x \quad y_U = +y_t \\ x_L &= x \quad y_L = -y_t \end{aligned} \tag{6.2}$$

Considering all the above equation; a Python script is constructed to obtain airfoil coordinates, and the outcome is represented in figure 6.2.

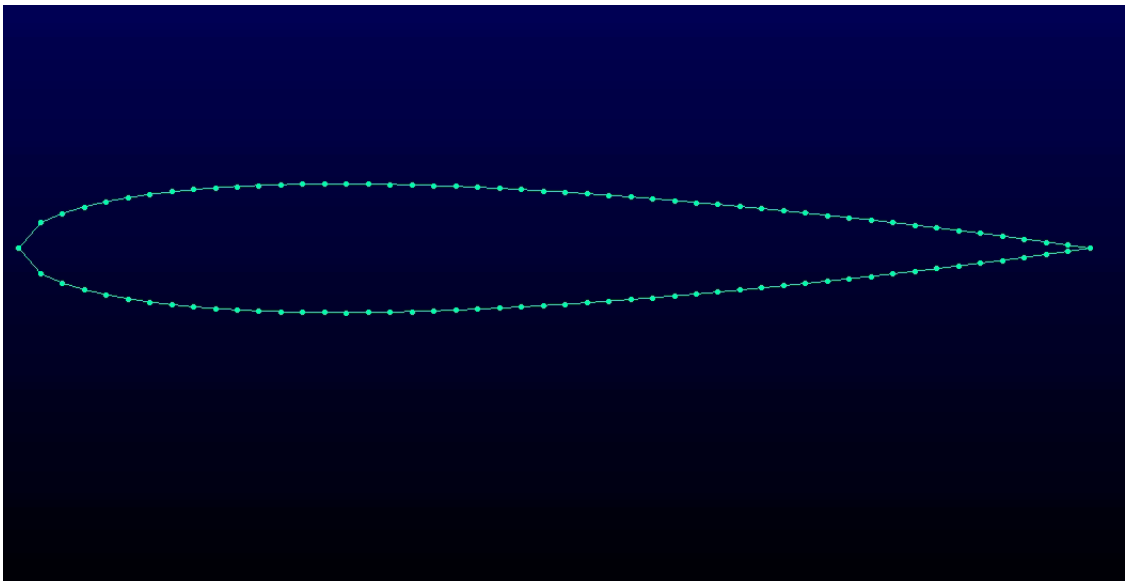


Figure 6.2: Uniform distribution of points along horizontal direction [total 100 grid points].

From figure 6.2, it can be noted that the distribution of grid points is uniform from leading edge to trailing edge. However, this will result in incorrect results of pressure distribution while performing CFD simulation. To obtain a good mesh where more points are clustered at the leading edge, cosine function (equation 6.3) is used, instead of the linear

distribution of grid points. Figure 6.3 represent the cosine functional distribution of grid points in horizontal directions.

$$\begin{aligned} x &= 1 - \cos\theta \\ \forall\theta &\in \left(0, \tfrac{\pi}{2}\right) \end{aligned} \tag{6.3}$$

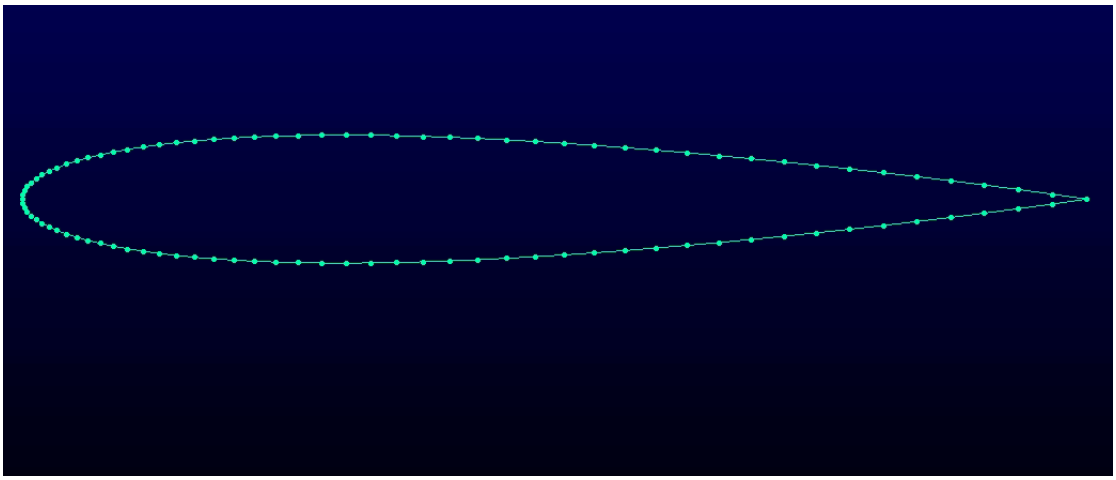


Figure 6.3: Cosine function distribution of points along horizontal direction [total 100 grid points].

Further, the cosine functional airfoil data is imported into pointwise (meshing software) in the Plot3D (*.x3d) format. Plot3D format is chosen because of the ease of readability. Other formats can also be opted based on convenience. In this work, grid points of 300 along the chord direction are used. Further, the obtained airfoil is extruded (normal extrusion) for three units (1 unit = 1 chord length) in the third dimension (span direction), resulting in the NACA0012 wing mesh. Based on the grid sensitivity analysis, a mesh point of 200 is used along the span direction. Figure (6.4) represents the outcome of spanwise extrusion. This wing surface is a reference (baseline surface), and this work focuses on optimizing the obtained wing shape.

6.2 FFD box over baseline mesh

After obtaining the baseline wing mesh, the FFD box or lattice needs to be covered over the surface where deformation is required. In this work, the entire wing surface is subject of interest. Hence, the whole body is covered with the FFD box. The FFD can either be constructed using the available GitHub code or by manually noting down the extreme ends of mesh and adding the scale factor to the coordinates points. In this work, code in glyph

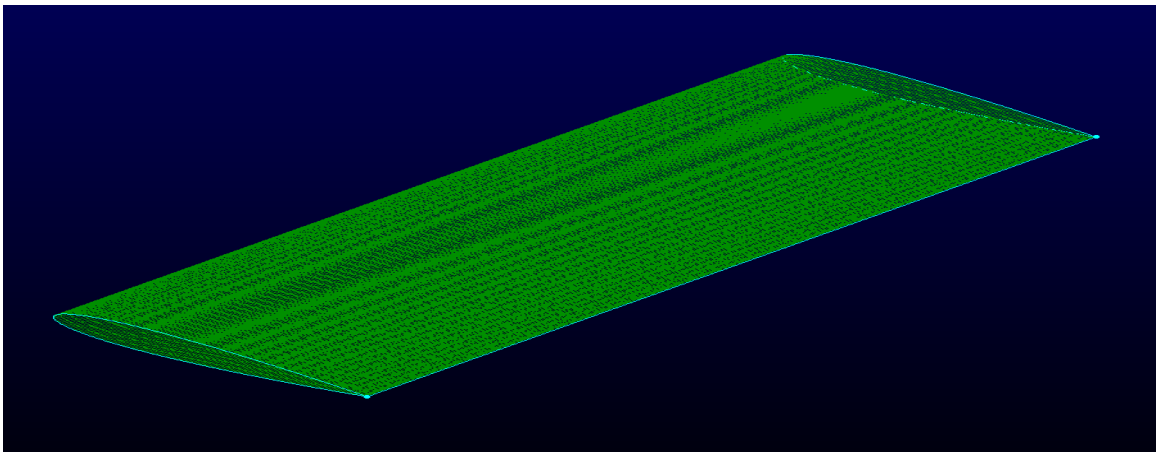


Figure 6.4: NACA0012 surface mesh (300 $\times$ 200) grid points.

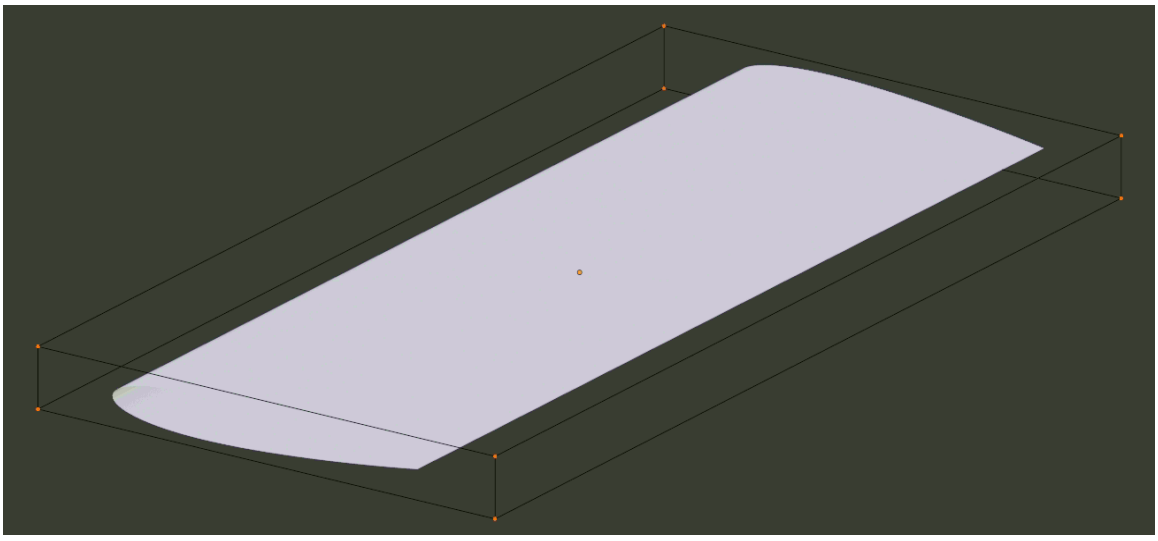


Figure 6.5: FFD box coupled with NACA0012 wing surface. Corner points highlighted act as control points.

script is used to get the FFD box coordinates points. Here, the scale factor of 1.5 times the chord's length is used along the chord direction. The 50% higher value of scale factor is used because the nearer the FFD control points, higher will be their influence over the object in interest. To reduce the influence and have higher tolerance (design space), an additional scale factor is used. In the same line, a scale factor of 1.1 is used along the thickness direction, and 1 unit along span direction is used. Figure 6.5 represents the wing surface being circumvented by the FFD box. The highlighted points (corner points) represents the control points of the FFD box. From figure 6.5, it is observed that the control points in the chord direction are 25% away from the extreme ends of the wing shape. Finally, the FFD box of size $1.5 \times 0.14 \times 3$ $(chord \times thickness \times span)$ units is obtained.

6.3 Interpolating FFD box control points

In figure 6.5, there are a total of eight control points (active control points), and each control point can be displaced in XYZ space independently. This results in a total of 24 (8×3) independent perturbation in a given space. In other words, the dimension of the problem becomes 24. However, in figure 4.2, eight control points were circumventing the sphere. The effect of perturbing right top corner point resulting in sphere shape perturbation. The effect of perturbation follows Bernstein polynomial (equation 4.4), and the effect is always surrounded near to the perturbed control point (figure 4.1).

Furthermore, to get a required shape, a higher displacement of control points will be necessary. To solve this issue and to distribute the effect of control points perturbation over an object, more control points have to spread over the subject of interest. There are several ways to add more control points, like log scale distribution, radial distribution, to name a few. However, the simplest one is considered a linear distribution of additional control points between the corner control points.

A python script is constructed, which takes corner points coordinates, the number of additional points required as input, and result in intermediate control points coordinates as output. Figure 6.6 represents the corner control points along with intermediate control points. In this work, there are a total of five control points along chord direction, two along the thickness direction, and six control points along span direction. In total, there are 60 control points, or 180 effective points available for perturbation. Now, the dimension of the problem is 180 (60×3).

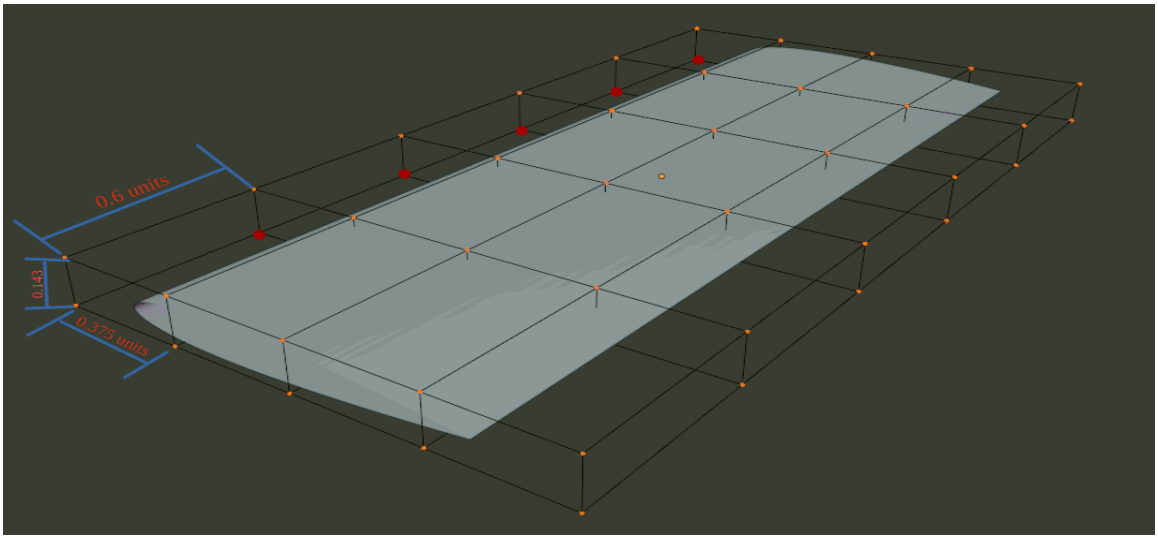


Figure 6.6: FFD box along with interpolated points and corner points.

6.4 Control points perturbation

After placing the desired number of control points over the wing surface, the wing surface mesh points need to be parameterized for the FFD box. This implies the cartesian coordinates of wing surface mesh points will be mapped into parametric form (X, Y, Z) to (s, t, u). This is because once the control points are perturbed, the cartesian coordinates x, y, z of a given point converts to x_1, y_1, z_1. However, the given point's parametric form remains the same as s, t, u. This is the crucial step in implementing the FFD box method for parameterizing the wing surface. For any i^{th} surface mesh point, the parametric form can be represented using the equation 6.4.

$$s_i = \frac{x_i - x_{origin}}{\Delta x}, \quad t_i = \frac{y_i - y_{origin}}{\Delta y}, \quad u_i = \frac{z_i - z_{origin}}{\Delta z} \tag{6.4}$$

where,

s_i, t_i, u_i are the parametric form of i^{th} mesh point.

x_i, y_i, z_i are the cartesian form of i^{th} mesh point.

x_{origin}, y_{origin}, z_{origin} are the cartesian coordinates of the selected FFD box corner point.

Δx , Δy, Δz are the differences between extreme points coordinates in the FFD box.

Now, the entire wing mesh points are mapped to parametric space. After this, perturbating any point of the FFD will lead to an object's shape deformation. For example, consider the entire FFD control points to be divided into layers along the span direction. Layer 0

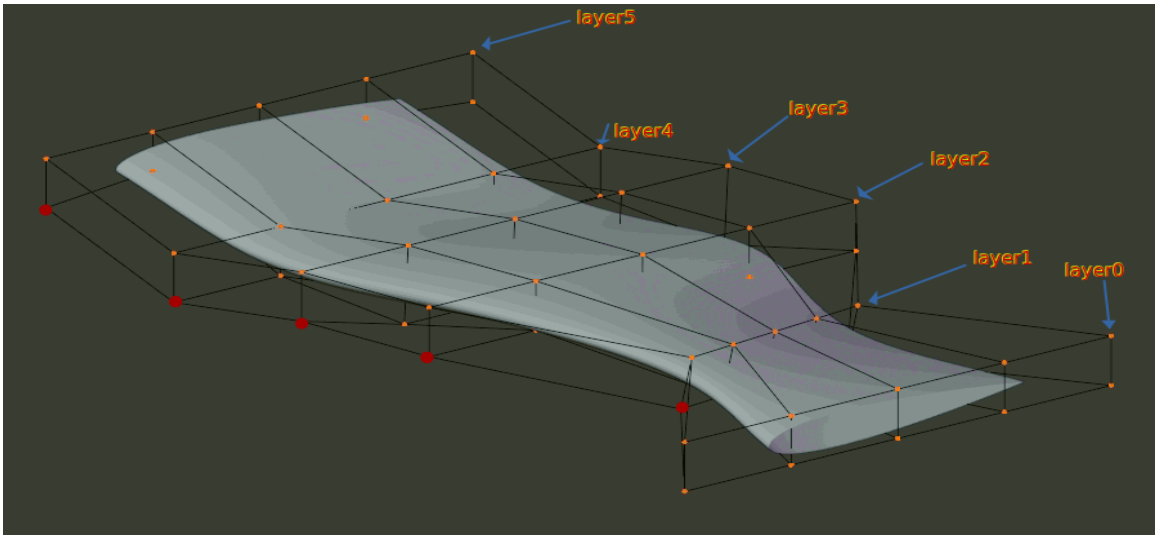


Figure 6.7: Effect of control points perturbation on the wing surface.

is located at the wing root section, layer 1 at 0.6 units away from the root section,. . . , and layer 5 at the wing tip section. From figure 6.7, layer 1 is contracted from all directions resulting in thickness and chord reduction. Similarly, layer 2 is slightly uplifted, which shows the sign of an anhedral effect. Other layers can also be perturbed similarly to obtain different behaviors towards airflow.

It should be noted that each control point can be perturbed independently, resulting in camber airfoil properties in some sections along the span direction. The choice in the number of layers is user-dependent. However, a choice of lower layers of the FFD along span direction would not be sufficient to obtain multimodality. Similarly, it is applicable in chord direction too. However, a two FFD point in thickness direction would be sufficient because the thickness length is insignificant over the chord and span direction length of the wing. In this work, a polynomial function of order five, four, linear will fit in the span, chord, and thickness direction respectively.

It is observed that allowing all control points will result in complete shape deformation. In some cases, the wing shapes could be infeasible. For example, the perturbed wing could possess a higher slender ratio, or the upper surface and the lower surface of the wing may intersect each other. Avoiding infeasible wing shape is possible by providing proper tolerance (design space) to all control points. Along with this, even some of the control points movement need to be restricted in a particular direction.

In the present work, the control points at the wing root section are restricted in the chord and thickness direction only. In other words, the control points in layer 0 can perturb in

chord and thickness direction and no perturbation in span direction resulting in a reduction of the dimension by 10. Also, the control points from layer 1 to layer 5 can perturb by the same values in the span direction. For example, all control points in layer1 can perturb by the same value (say 0.2) in span direction. A similar approach is carried to layer 2, 3, and 5. With this, there will be a further reduction in the dimension by 45 (5 layers $\times$ 9 values). In total, dimension of problem is reduced by 55. By the end of this process, the dimension will be 125 (180 - 55). In other words, 125 values are required to define a wing.

After fixing the dimension (125), design space (tolerance for each dimension) must be sensibly selected. If higher design space is selected, the intersection of the upper and lower wing surface will occur. On the other hand, selecting a lower design space may not result in multimodality.

As mentioned before, there are five, two, and six control points in this work, along with chord, thickness, and span directions, respectively. A scale factor of 1.5 is selected along the chord direction. Therefore, the distance between subsequent control points along the chord direction is 0.375 units (1 unit = 1 chord length). At maximum, the control points along the chord direction can perturb by the value, which is half the distance between them, that is $\leq$ 0.1875 units. Similarly, along the thickness direction, a control point can perturb by value $\leq$ 0.07 units. Along the span direction, it will be $\leq$ 0.3 units.

A .csv file $(designspace.csv)$ containing design space vectors in 60 rows and 3 columns is created. The point in which perturbation is limited to specific direction contains positive values, whereas those points whose movement is not allowed in a particular direction are set to zero. It can be observed that along the thickness direction, not much design space is available for optimization.

Consider figure 6.6, there are total 6 layers along the span directions (layer 0 to layer 5). The control point represented with red dot acts as a leading control point in that particular layer. During generating a wing, a leading control point is initially perturbed along thickness direction by the value $\pm$0.25 about the initial coordinate point (leading point coordinate corresponding to initial baseline wing). Further, the remaining lower control points (four in number) in that layer are perturbed by value $\pm$0.1 about the perturbed leading control point coordinate. The upper control points in that layer are perturbed between the corresponding lower control point coordinate and 0.2 above it (lower control point + 0.2). Following this tedious process, it is possible to obtain the curvilinear (of order 5) wing shape along the span direction. Performing this step will guarantee additional design space, which also results in enhancing the possibility of obtaining the multimodality. The following lines represent the upper and lower limits (box constraint) of the design space in

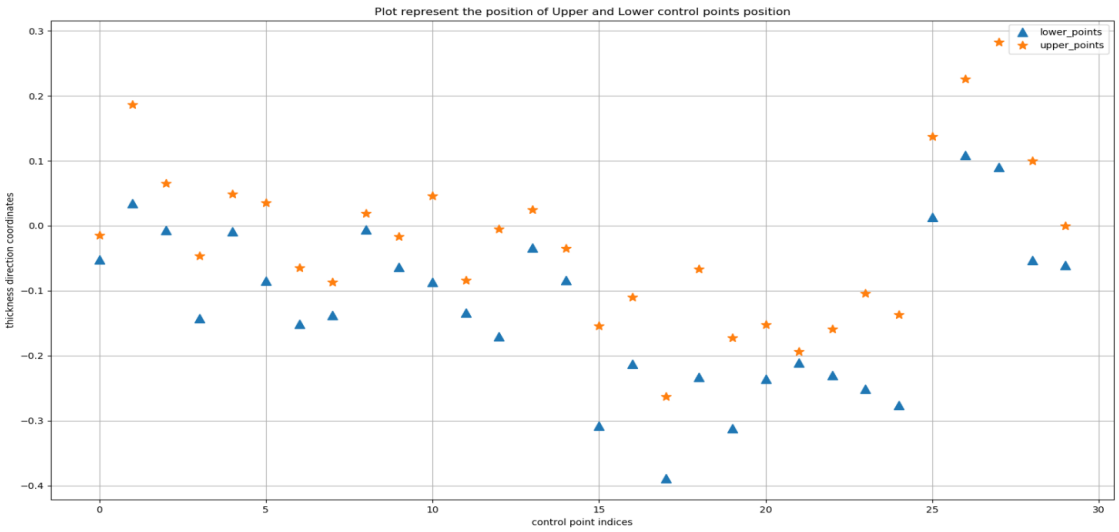


Figure 6.8: Plot representing distribution of control points in thickness direction.

all possible dimensions (individuals).

1. Along chord direction: All control points (60 in number) are allowed to perturb by 0.1875 about their initial coordinate value (control point coordinate representing baseline geometry).

2. Along span direction: Except root section control points, all other control points are allowed to perturb by value 0.3.

3. Along thickness direction:

   - Leading control point of a layer is perturbed by value $\pm$ 0.25 about the initial FFD control point coordinate.
   - Lower control points in that layer are perturbed by value $\pm$ 0.1 about leading control point coordinate.
   - Upper control points in that layer are perturbed between the corresponding lower control point and 0.2 higher than it.
   - Same process is repeated to all remaining layers along span direction.

A plot representing the control point position in the thickness direction is shown in figure 6.8. Y-axis represents the thickness direction position of control points. The X-axis represents the indices of control points. Index 0 to 4 corresponds to layer 0 control points; index 5-9 represent the layer one control points, and so on. The plot shows that for a given

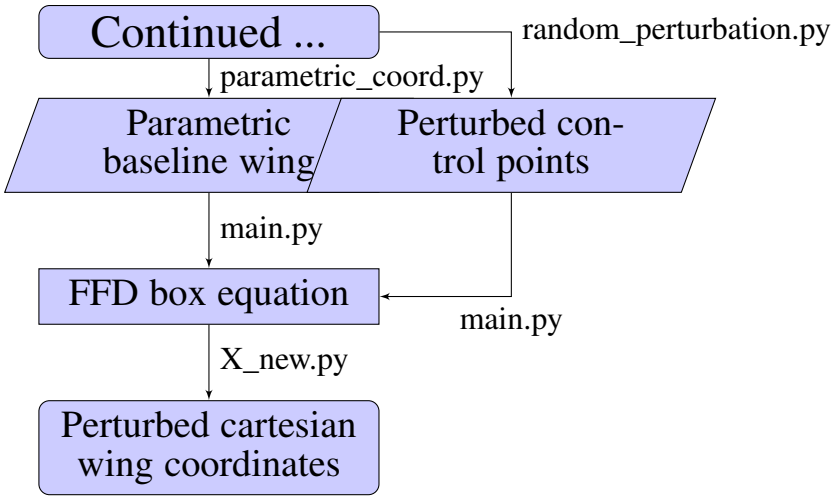


Figure 6.9: Block diagram representing the input and output to FFD box equation.

wing, the upper control points are always positioned above the corresponding lower control points. This indicates that the wing upper surface will not intersect with the lower wing surface. Also, as mentioned before, the upper control point is at a distance of ≤ 0.2 above the corresponding lower control point.

After completing the design space choice, perturbed control points coordinates are chosen between the respective allowable design space. Further, the parametric form of baseline wing and the new perturbed control points are passed as input to the FFD box equation 4.3. Here, by considering new control points coordinates, a new set of cartesian coordinates of the perturbed wing is obtained as output. A block diagram representing the same is as shown (figure 6.9). Further details about the FFD box equation are explained in chapter 4.

6.5 Implementing PCA

With 125 design variables defining the feasible design space will satisfy the geometric constraints. Now, it is planned to reduce the dimension representing the design space. However, by using a lower dimension (reduced-order modelling) to represent the design space will not cover the entire design space. Furthermore, by perturbing each design variables with a narrow range, it is possible to obtain the feasible design space. Also, for optimization, the DE algorithm is implemented given geometric constraints. The DE algorithms are not tested for the problem with a dimension greater than 30. There is a need to reduce the problem dimension, which can be achieved by implementing the PCA.

Initially, the sample space containing perturbed wings needs to be constructed. For ex-

ample, let N be the number of perturbed wings being generated within the design space. It should be noted that, the range for each of these design variable (125) is the same as mentioned in the previous section. In this work, based on the literature survey and computational power available, a generation cycle of 50 and a population size of 20 is chosen, resulting in the total CFD calculation (functional evaluation) of 1000 (50×20). Based on experience, $N = 1000$ perturbed wings are generated. It is possible to generate a large number of the perturbed wings. But, this will create issues while calculating SVD. Each of these wing coordinates $(x_1, y_1, z_1, x_2, y_2, ..., x_n, y_n, z_n)$ are arranged in a row. Also, each wing is defined by $n = 30000$ grid points resulting in a matrix (S) of size 1000×90000. In other words, this is defined as N samples having n features.

Mathematically, let the coordinates of baseline geometry be defined by p coordinated points $x_i^0 \in \mathbb{R}^m$, $i = 1, \ldots, p$, where m represents the data's dimension. Also, let n sets of measurements $\{\hat{x}_{i,j} \in \mathbb{R}^m \mid i = 1, \ldots, p\}; j = 1, \ldots, n$ be available. Here index j refers to a specific instance of measurement, and index i refers to measurements representing a unique nominal coordinate point.

Later, the error matrix is calculated by taking the difference between the S matrix and baseline vector (NACA0012 wing) B. The resultant matrix is subtracted from the average error vector, resulting in a covariance matrix C_x. Subject the covariance matrix C_x for Singular Value Decomposition (SVD) calculation. There are builtin package available in python to calculate SVD. The outcome of SVD is three matrices U, Σ, V^T. The matrix U is of size 1000×1000, Σ is a diagonal matrix of size 1000×1000, and the matrix V^T is of size 1000×90000. Mathematically,

The error from the nominal geometry is given as,

$$x'_{i,j} = \hat{x}_{i,j} - x_i^0$$

Further, subtracting the error vectors from their mean gives a centered set of m-dimensional vectors as follows.

$$\tilde{x}_{i,j} = x'_{i,j} - \bar{x}_i \mid i = 1, \ldots, p; j = 1, \ldots, n \tag{6.5}$$

where, the mean of these error vectors is calculated as follows.

$$\bar{x}_i = \frac{1}{n} \sum_{j=1}^{n} x'_{i,j}, i = 1, \ldots, p \tag{6.6}$$

Representing the centered error vectors in a matrix form $\tilde{\mathbf{X}}$ with j^{th} column as, $\tilde{X}_j = \left[\tilde{x}_{1,j}^T, \ldots, \tilde{x}_{p,j}^T\right]^T$. This implies $\tilde{X}_j$ contains the centered set of error vector representing the j^{th} set of measurement. $\tilde{\mathbf{X}}$ will be generally of size $mp \times n$ matrix. The covariance matrix of $\tilde{\mathbf{X}}$ is as shown in equation 6.7.

$$C_x = \frac{1}{n-1}\tilde{X}\tilde{X}^T \tag{6.7}$$

As mentioned earlier, PCA typically identifies the directions that are mutually uncorrelated to each other. Along with this, it identifies the directions of the greatest variance of the data. This can be achieved by finding a transformation matrix which diagonalizes the covariance matrix C_x. One such method is obtaining the eigenvalue of the covariance matrix C_x. The more prominent way of approaching this problem is to perform a Singular Value Decomposition (SVD) of a matrix $\tilde{\mathbf{X}}$[37].

$$\tilde{X} = M\Sigma N^T \tag{6.8}$$

Here, M, N are $mp \times mp$, $n \times n$ orthonormal matrix respectively. And, Σ is $mp \times n$ is a diagonal entries ordered in decreasing value.

Each row of matrix V^T is principal components (directions) in a given design space. The first principal component (first row) is the most prominent direction for the highest variance. The variance keeps reducing down the rows. The matrix Σ contains the values arranged to decrease λ_i^2, where λ_i is the eigenvalue of the i^{th} row (also called singular values).

6.6 Problem dimension selection

Reduced-order modeling is based on selecting the leading modes of the principal components obtained by finding the SVD of the covariance matrix $\tilde{\mathbf{X}}$. However, to select a specific number of principal components, the scatter energy method is used. The total scatter energy E is as shown as in equation 6.9.

$$\begin{aligned} E &= \mathrm{tr}\left(\tilde{X}\tilde{X}^T\right) \\ &= \mathrm{tr}\left(M\Sigma N^T N\Sigma M^T\right) \\ &= \mathrm{tr}\left(M\Sigma\Sigma M^T\right) \\ &= \|M\Sigma\|_F^2 \end{aligned} \tag{6.9}$$

where $\|.\|_F^2$ is the Frobenius norm. Since the Frobenius norm is invariant under unitary multiplication.

$$E = \|M\Sigma\|_F^2 = \|\Sigma\|_F^2 = \mathrm{tr}\left(\Sigma\Sigma^T\right) = \sum_{i=1}^{mp} \lambda_i^2$$

where, λ_i is the i^{th} diagonal entry of Σ. Selecting the first few modes from the matrix M and taking note of their corresponding eigenvalues makes it possible to predict the amount of scatter energy being captured. This is a principal of using PCA based reduced-order modeling.

Another reason for using PCA based reduced-order modeling is the ease of implementation and computational efficiency of the PCA algorithm. SVD is the most popular way of implementing PCA. Predefined modules are available in both MATLAB® and Python, which further strengthens the use of SVD.

Discussing the derivation of PCA based reduced-order modeling is out of scope here. Therefore, a reduced-order model for the geometric uncertainty can be written as,

$$x_r = x^0 + \bar{x} + \sum_{k=1}^{n_r} a_k \mathcal{N}(0,1) \tag{6.10}$$

where, $n_r \leq mp$ is the number of leading modes selected and a^k is the k^{th} column of $\frac{1}{\sqrt{n}} M\Sigma$. x^0 is the row representing the baseline wing coordinate, and $\bar{x}$ is the centered average wing. $\mathcal{N}(0,1)$ is the random variable with zero mean and unit variance. As n_r increases the total scatter of x_r tends to $\tilde{x}$.

In this work, the singular values of leading principal components were of order 10^2 compared to the latter once, which were of order 10^{-5}. This indicates that the first few leading principal components were sufficient to represent the entire design space available approximately. The scatter random energy concept is used to quantify the selection of singular values. Equation 6.9 represents the calculation related to the scatter random energy.

Figure 6.10 represents the scatter energy distribution against the number of singular values selected. By selecting the first ten principal components, it is possible to cover over 85% of total scatter energy. Similarly, with 30 principal components, over 95% of total energy is covered. Generally, the value of over 85% is considered to be the good choice. Table 6.1 represents the exact number about the scatter energy distribution over the singular value selected.

It is possible to capture over 97% of total scatter energy by selecting the first 20 singular values, whereas selecting the first 30 principal components will capture over 99% of total

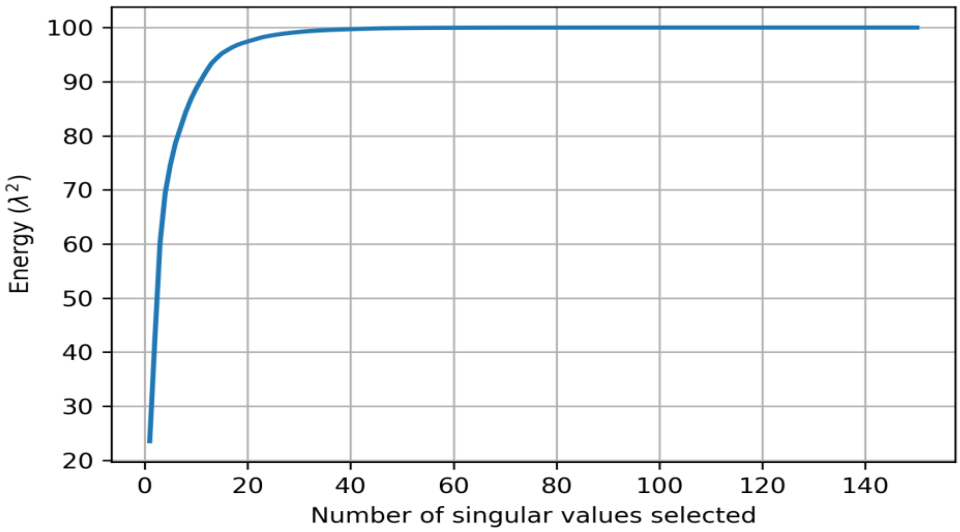


Figure 6.10: Scatter energy verses number of singular values selected.

Singular values selected	Scatter energy (%)
1	23.52
2	42.48
5	74.63
10	88.70
20	**97.42**
30	99.17
50	99.98

Table 6.1: Singular value selected against scatter energy

scatter energy. However, there is no significant improvement in the total scatter energy captured. Also, with the higher singular value selected, more will be the computational cost, which affects the convergence rate. So a singular value of 20 is considered to be a good choice. Also, the number of the singular value selected is analogous to the dimension of problem. By selecting the first 20 leading principal components, the entire dimension of problem is reduced from 125 (previous) to 20 (present). This is a significant improvement in terms of the convergence rate of the optimizer.

Figure 6.11 represents the norm value calculated for various singular values selected. Y-axis represents the norm value which is calculated using equation 6.11.

$$\text{Norm value} = \frac{norm(X - X^*)}{norm(X)} \tag{6.11}$$

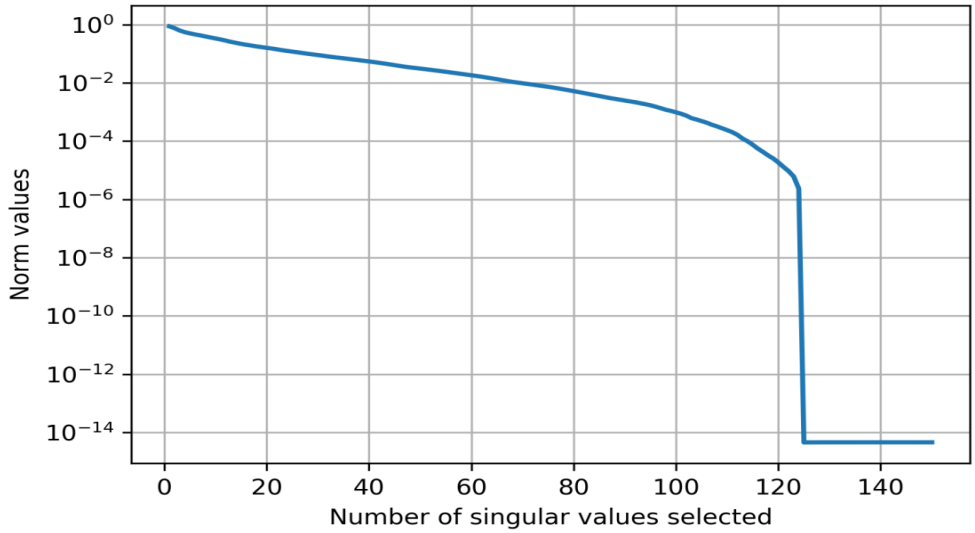


Figure 6.11: Plot representing the norm value against the singular value selected.

where,

X is the covariance matrix of a full size (1000×90000).

X^* is the covariance matrix of a singular values selected.

For example, let t be the number of leading principal components being selected. The matrix X^* will be a matrix multiplication between U_s, Σ_s, and V_s^T. that is,

$$X^* = U_s \Sigma_s V_s^T$$

where,

U_s is the matrix consisting of first t columns $(1000 \times t)$.

Σ_s is the diagonal vector with t elements $(t \times t)$.

V_s^T is the matrix with first t rows $(t \times 90000)$.

From figure 6.11, it is observed that at the singular value of 20, the norm value obtained is of order 10^{-1}, which is a good choice to proceed with the optimization. Further, at the singular value of 125, the graph drops vertically to 10^{-14}, which is $\approx$ zero, indicates that if 125 singular values are selected, then $X = X^*$. This step confirms that the previous dimension was 125.

6.7 Selecting design space

After selecting the number of singular values to 20, evaluation of the matrix U_s and Σ_s is carried. let A be a matrix such that $A = U_s \Sigma_s$. The matrix A is of size (1000×20).

Later, the maxima and minima of each column of A are identified. If the maxima and minima values are the same in magnitude, then the design space is symmetric with the origin. Otherwise, select the smallest (by magnitude) among them and replace it with the previous one.

For example, if a and b are the maximum and minimum values of a given column, $|a| < |b|$, then chose a as the extreme point and replace the previous design space to $(a, -a)$ and vice verse. The design space obtained in this work is appended in C. In all 20 dimensions (direction), the design space is made symmetric about the origin.

6.8 Generating initial population

After deciding the design space, the next step is to evaluate the wing surface coordinates. Based on the availability of computational power, a population size (N) of 20 is chosen. Further, between individual design space, a set of random numbers of size equals to population size (in this work it will be 20) are generated. Each vector is named as target vector in optimization algorithm. This results in a matrix A_g of size 20×20. The matrix A_g is multiplied with Σ_g and V_g^T resulting in the matrix X_g in equation 6.12.

$$X_g = A_g \Sigma_g V_g^T \tag{6.12}$$

where,

X_g is the covariance matrix representing the randomly generated wings.

σ_g is the diagonal matrix containing the first 20 singular values.

V_g^T is the matrix containing the first 20 rows of V^T.

Further, the obtained matrix (X_g) is added to mean of error vectors $(\bar{x})$, and the baseline geometry coordinates (x^0) resulting in matrix M. In the present work, the matrix M is of order 20×90000. Each row in M represents the cartesian coordinates of the randomly generated wing and is arranged in the order $(x_1, y_1, z_1, ..., x_n, y_n, z_n)$. Each wing's coordinates are rearranged to the Plot3D format (*.x3d) and script to a file.

$$M = x^0 + \bar{x} + X_g$$

M is as called a reduced-order model.

The generated wing is imported to pointwise in the Plot3D format. Similarly, the principal modes of wings are generated and imported to pointwise. However, they did not depict any positive sense. For example, it was expected that the first principal mode would

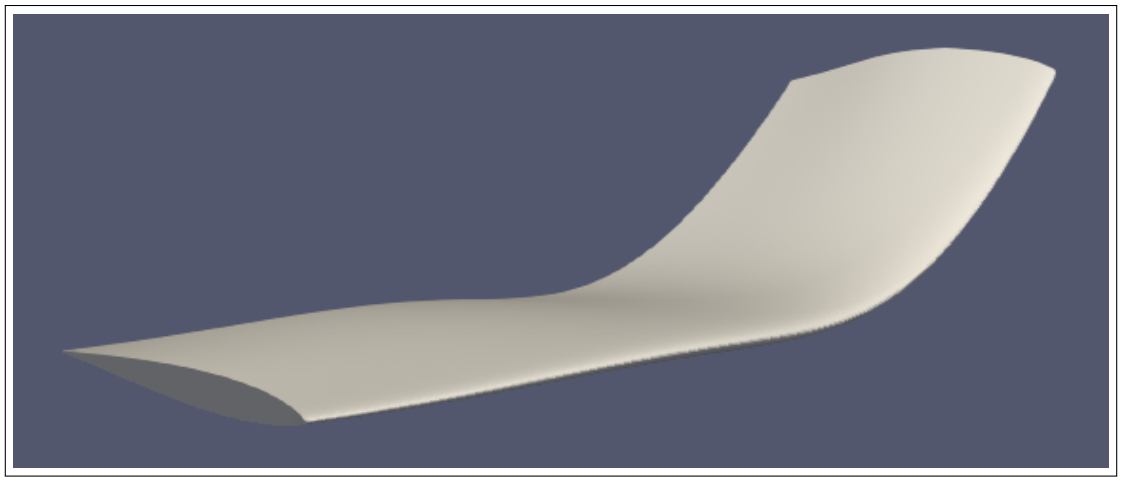


Figure 6.12: General SVD wing 1.

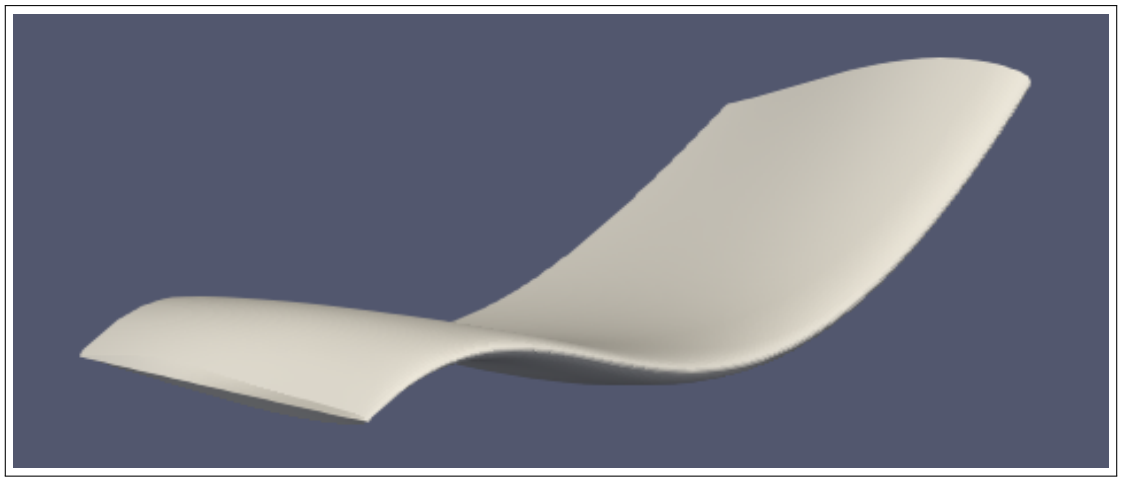


Figure 6.13: General SVD wing 2.

indicate the span property, thickness property, independently. Instead, the principal modes showed the mixed behavior of thickness, span, chord variation property. This leads to an insignificant conclusion towards wing behaviors. Figure 6.12 to 6.14 illustrates the wings generated after implementing the PCA. It is observed that the wing shape generated almost represents the wings those being generated after the FFD box implementation(fig 4.5). More on this is explained in chapter 7.

6.9 Implementing niching algorithm

Several niching algorithms were implemented and tested successfully over the test functions. However, with the available time, it became possible to implement the fNRAND1

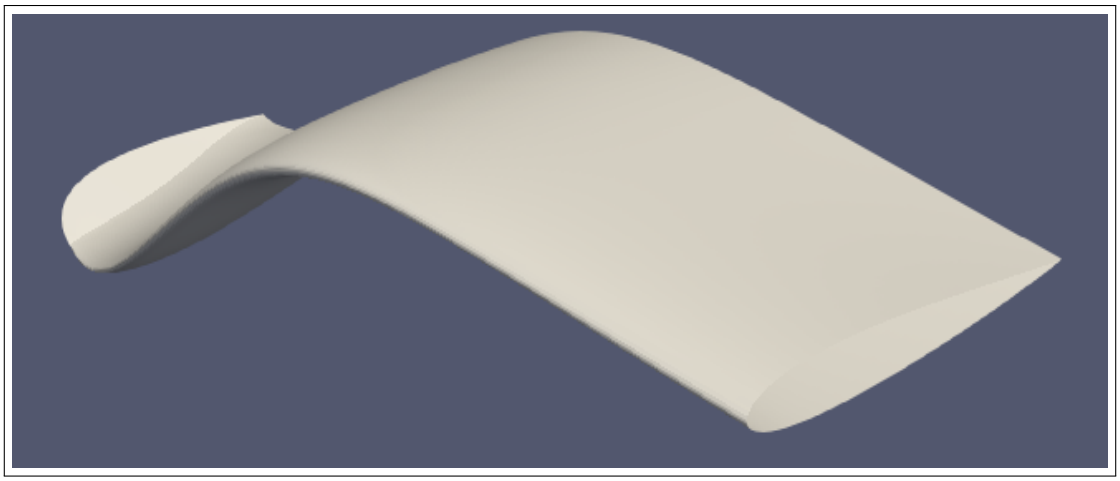


Figure 6.14: General SVD wing 3.

algorithm alone for multimodal optimization of the NACA0012 wing. A detailed explanation of the algorithm is presented in section 3.9. A fewer modification to the algorithm is made to make it fit the problem. These issues are highlighted, and the solution to mitigate them are discussed.

The first step in implementing the algorithm is to construct the initial population of the wings. Initially, the population is randomly constructed, as mentioned above and the target vector is saved to python variable. The entire population is subjected individually for winglet creation. A glyph script is built, which imports the geometry (*.x3d) into pointwise and identifies the extreme coordinates wing tip section.

To construct the winglet, a shoulder point of the curve is the first requirement. First, evaluate the average value of the extreme coordinates along chord and thickness direction, and along the span direction, the coordinates will be 2% of the span length. With the help of shoulder point coordinates, the curve is constructed. Further, an unstructured mesh is generated from the wingtip section to the curve. Figure 6.15 represents the winglet created at the wingtip section.

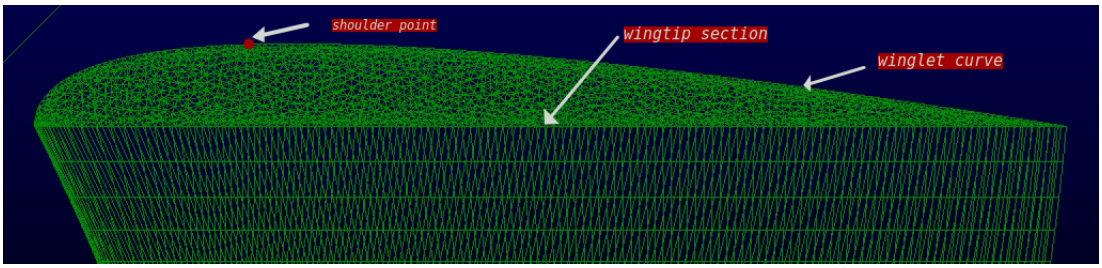


Figure 6.15: Winglet at wingtip section

After completing the winglet section, the next approach is to generate a volume mesh over the entire wing surface. A box volume mesh of size 15 units (1 unit = 1 chord length of baseline wing) in all three directions is generated. Further details about the volume mesh and boundary conditions will be explained in chapter 5. Once the volume mesh is generated and boundary conditions are applied, the mesh is exported in the SU2 format (*.su2). Each of these su2 files is subjected to function evaluations, which is SU2 simulation. Each simulation is considered as an individual job and subjected to different threads of the cluster.

After the end of the simulation, the lift coefficient, drag coefficient, and moment coefficient about the chord direction are filtered from the file and subject to further evaluations. Proper coordination between the wing generation and mesh generation using pointwise and CFD simulations using SU2 solver is primarily managed by building a python code.

If any wing fails to create the volume mesh, the possible reason could be the presence of wing surface wrinkles. The better approach to resolve this issue is by regenerating the wing and subject it to generate the volume mesh.

After generating the initial random wings, they are subjected to optimization cycles. A generation cycle of 50 and a population size of 20 is chosen. Mutation step contains a scaled vector addition of target vectors, resulting in a mutant vector. Later, the mutant vector is subjected to the crossover phase. For each element in the mutant vector, corresponding random value between 0 and 1 is generated and compared with the predefined value (crossover probability factor). After the end, a trial vector is obtained, and using this trail vector; wing surface coordinates are generated and script to a file. Further, this file is imported to pointwise using glyph script, and after creating winglet and volume mesh, it is export to SU2 solver. After completion of the CFD simulation, the required coefficients are extracted from the output file.

The moment coefficient constraint is satisfied using feasibility rules given by [21]. Based on the fitness value, the target vector will get replaced by a trial vector. With the modified target vector, the generation cycles are continued for the remaining number of generation cycles. After each generation's end, the target vector is scripted to a file and is subjected to check for the existence of multimodality in a given design space. Specific details about how CFD simulation is carried and the ways to submit jobs are explained in chapter 5.

All python codes and other related information can be found in GitHub.

Chapter 7

Result and Discussions

Once all software packages are aligned under a single python script, the python script is fired into the HPC cluster. After covering a considerable amount of time, the results are extracted, and the post-processing is carried using Paraview. This chapter contains detailed information about the number of computational hours used, the CFD solver setup, optimized wings, and more importantly, the comments over the obtained results.

The objective function that is aimed is the drag coefficient minimization. Furthermore, the drag is made up of two parts. They are pressure drag and the induced drag. Since this work involves the inviscid solution, the drag represented here will be the induced drag alone. Equation 7.1 represents the equation for induced drag evaluation [18].

$$C_{D_i} = \frac{C_L^2}{\pi AR}(1+\delta) \tag{7.1}$$

where,

$$\delta = 8\left(\frac{3\pi}{2}\frac{C_{M_x}}{C_L} - 1\right)^2$$

For the given problem (ADODG case 6) with aspect ratio 6, the theoretical minimum induced drag is found to be 25.5 counts (1 count = 10^{-3} drag units).

7.1 Algorithm Assessment: Finding local minima for the test functions.

The niching algorithm is initially tested on an analytical test function to assess their performance. Appendix A.3 contains the test functions that are analyzed. For example,

consider the sphere function, as shown in equation B.4.

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2 \tag{7.2}$$

Further, the test function is subjected to constraints as mentioned in [5]. There exists four optima located at (1, 1), (-1, 1), (-1, -1), and (1, -1). Niching algorithm as mentioned in chapter 3 are implemented over this test function and the result are promising. Figure 7.1 illustrates the outcome of fINRAND1 niching algorithm over the given 2-D test function.

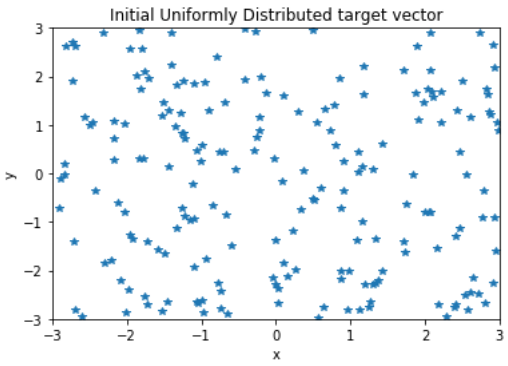


Figure 7.1: Initial 200 population.

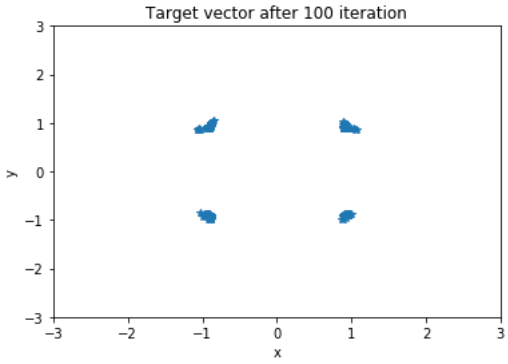


Figure 7.2: Optimal points after 100 generations.

Overall, a population of 200 are used within the design space $x_1 : (3, -3)$ and $x_2 : (3, -3)$. A total of 100 generations are carried out to get an optimal solution. However, in the case of CFD simulation running for such higher generations seems impractical. A trade-off is required between the generations and the solutions obtained. This confirms that the algorithm is capable of identifying the local optima in a given design space.

7.2 Computational power

In the initial stage, the niching algorithms were tested in the workstation with Intel Xeon W-2155 @ 3.3 GHz, 20 CPU. However, the assigned workstation would not be sufficient to run the full-fledged optimizer. As a result, the entire work is executed in the HPC cluster containing 56 CPU with Intel Xeon E5-2690 @ 2.6 GHz. This work demands for 20 CPU of a given compute node and takes less than 180 minutes to complete a generation.

7.3 Grid sensitivity analysis

In the entire optimization process, the objective function evaluation (CFD simulation) is a bottleneck. With higher mesh points, more will be the computational cost. So it is very crucial to choose the mesh size, which is feasible for given computational power. A baseline wing with two mesh quality is generated. Table 7.1 provide additional details about the mesh quality.

Mesh quality	Surface mesh	Volume mesh
Coarse	150×100	0.11m
Fine	300×200	0.25m

Table 7.1: Grid sensitivity analysis

The baseline wing with coarse mesh (150 along the chord, 100 along the span) would take less than 50 minutes (workstation) to complete the CFD run, whereas, the fine mesh takes about 120 minutes (workstation). Further, with additional lift coefficient constraint the fine mesh completed the CFD run in 180 minutes (workstation). This implies if the problem is set to run in parallel with each CPU assigned to one of the perturbed wings, after every 180 minutes, one optimization generation get completed.

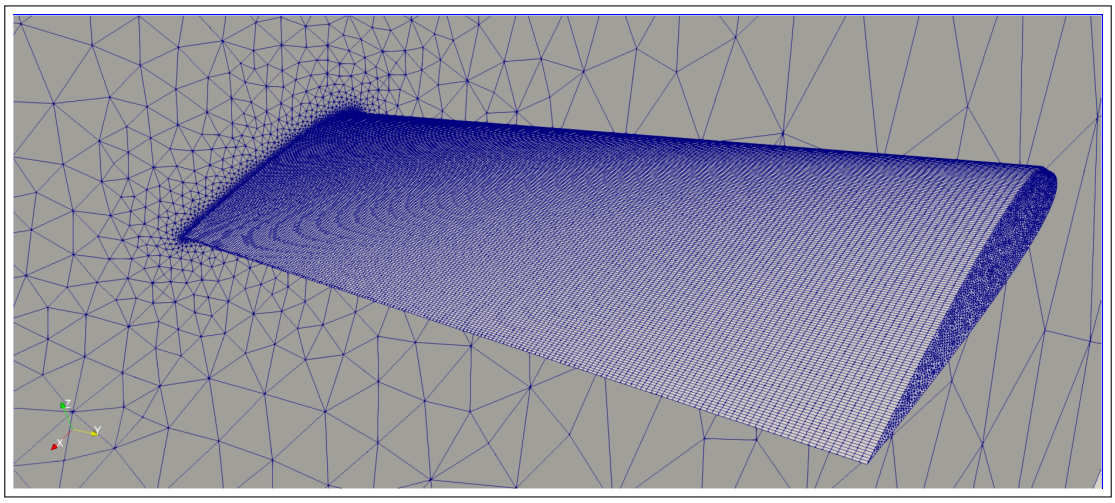


Figure 7.3: Fine mesh with 300×200 surface mesh points.

Figure 7.3 represents the fine mesh containing 0.25m mesh points with 300×200 mesh points over the wing surface. The distribution of mesh points at the symmetry surface is appropriate. Cell size is increasing from the wing surface towards the wall surfaces at a rate of 20%.

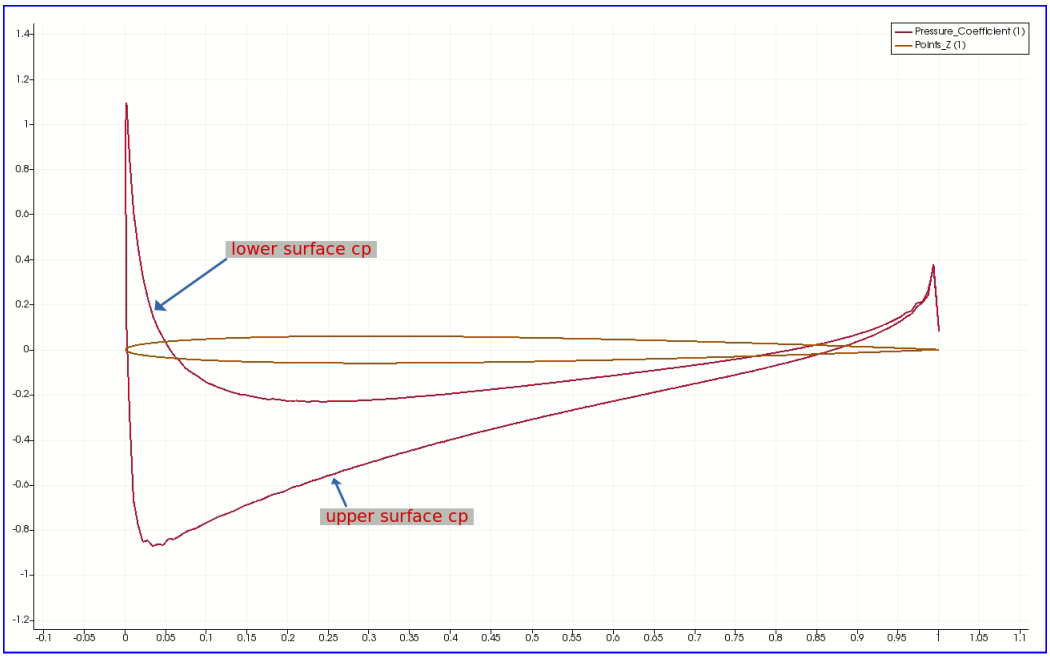


Figure 7.4: Cp curve at root section of fine mesh (300 $\times$ 200).

Figure 7.4 represents the coefficient of pressure at the root section of the baseline wing for a fine mesh. The results of the cp curve meet the expected values. There is some disturbance in the Cp curve at the trailing edge. The reason for this is because the airfoil has a sharp trailing edge. The fine mesh appears appropriate to proceed with full optimization.

7.4 Multimodality

In this problem, the flow is subsonic and inviscid. The objective is to minimize the drag coefficient of C_d. The Mach number is 0.4. The lift coefficient of C_l is constrained to 0.2625. The angle of attack is varied between -3^0 and +6^0. The angle of attack is a design variable which is optimized using CFD solver. The root bending moment is constrained to be $\leq$ 0.1039. Geometry is parameterized using the FFD box with 5 points along chord direction, 2 points along the thickness direction, and 6 points along the span direction, resulting in 60 control points. Each perturbed wing can vary in span direction between 2.7 units and 3.3 units. Due to FFD box control points perturbation, the perturbed wing will possess the twist and sweep angles. Each of the control points is perturbed such that it always results in feasible wings. More details on FFD control point perturbation is mentioned in chapter 6.

The first 20 initial geometries (population) generated by the random selection of point

within the PCA selected design space is shown in figure 7.5. These population cover all the design space like dihedral, taper wings, sweep backward/forward, winglet up/down, and span long/short.

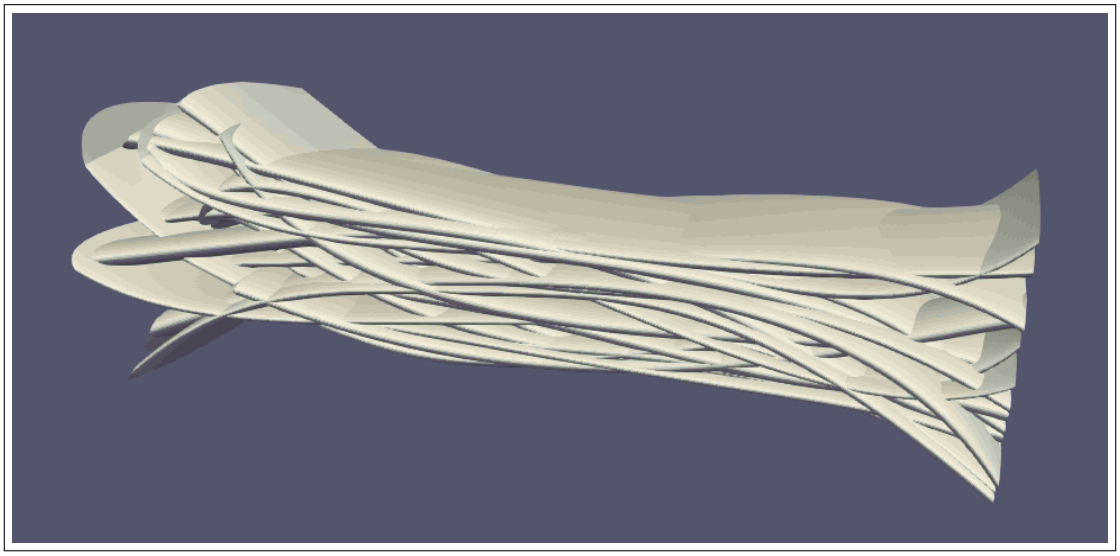


Figure 7.5: Initial 20 populations.

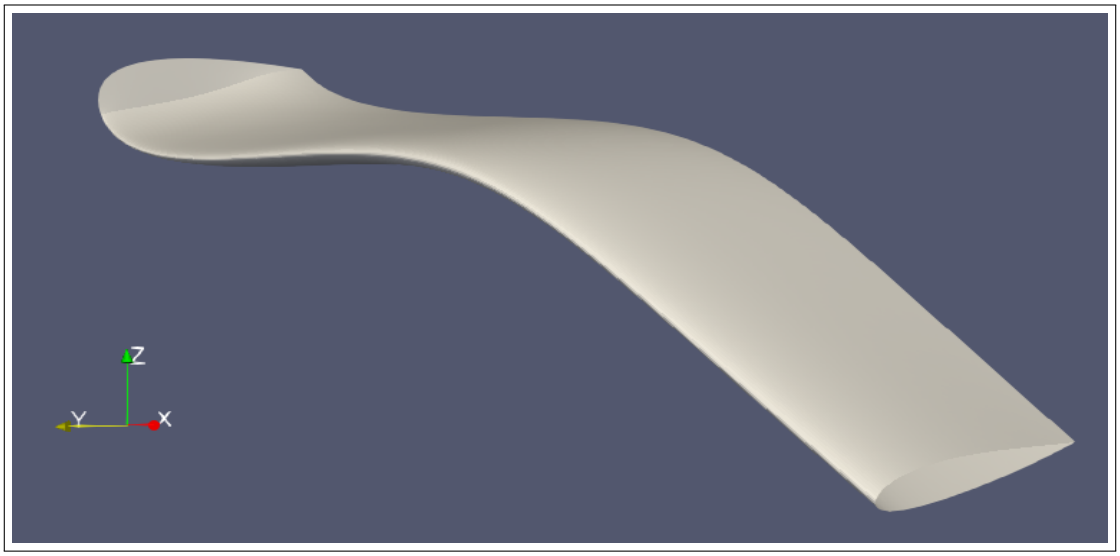


Figure 7.6: Isometric view of local optima 1.

Since the flow is subsonic and inviscid, only the induced drag is present. The baseline (NACA 0012) geometry has C_d of 5.08 $\times$ 10^{-2}, with C_l constraint satisfied by adjusting the angle of attack. After 1000 objective function evaluation, the first local optima was

found to possess the C_d value of 1.12 $\times 10^{-2}$. Figure 7.6 and 7.7 illustrates the isometric and leading-edge view of local optima 1 respectively.

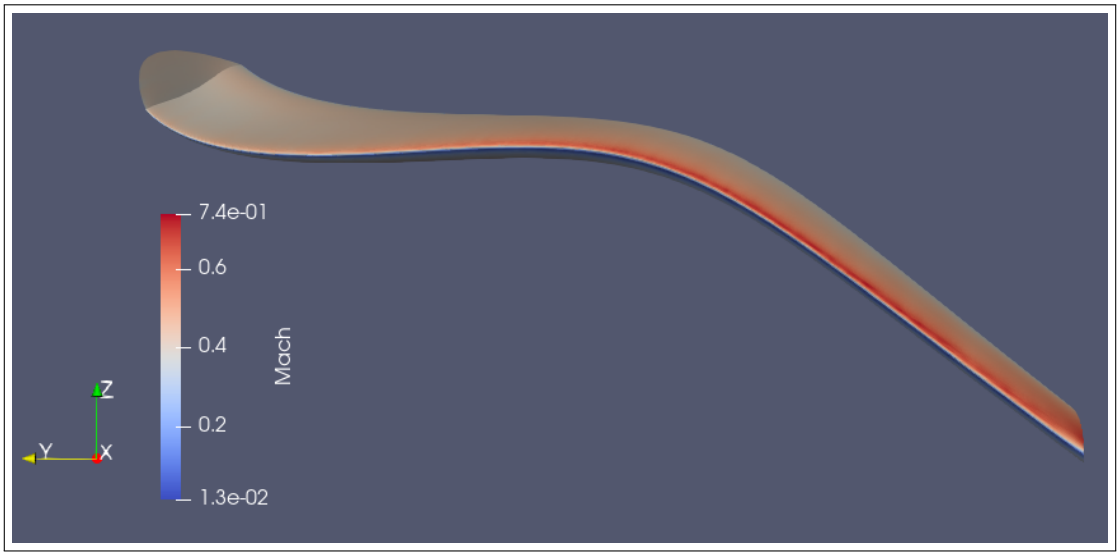


Figure 7.7: Leading edge view of local optima 1.

Figure 7.8 represents the Cp distribution for local optima 1 at three sections, that is, root section, midsection, and tip section. At root section, near the leading edge, there is a sharp decline in Cp value in the upper surface. The reason for this is incorrect volume mesh generation around the leading edge. A similar explanation can be provided for trailing edge Cp disturbance. The airfoil is twisted by small value and lift by a unit in the midsection of the span. At the tip section, the results got disrupted entirely. The airfoil is highly perturbed, resulting in negative lift.

After further analysis, it was found that the glyph script, that was designed to automate the volume mesh generation is incorrect. The script could not generate enough mesh points to get correct CFD results. No conclusion can be drawn from incorrect CFD solution. Due to insufficient time, the glyph script could not be corrected. However, assuming that the CFD results are correct, table 7.2 represents local optima with additional details. More on these optima is covered in the upcoming section.

7.5 Discussion

After a detailed examination of all steps, it is noticed that the wing parameterization, the FFD box implementation, the PCA implementation, and niching algorithm implementation is working correctly. The sole issue for incorrect results was inadequate volume mesh

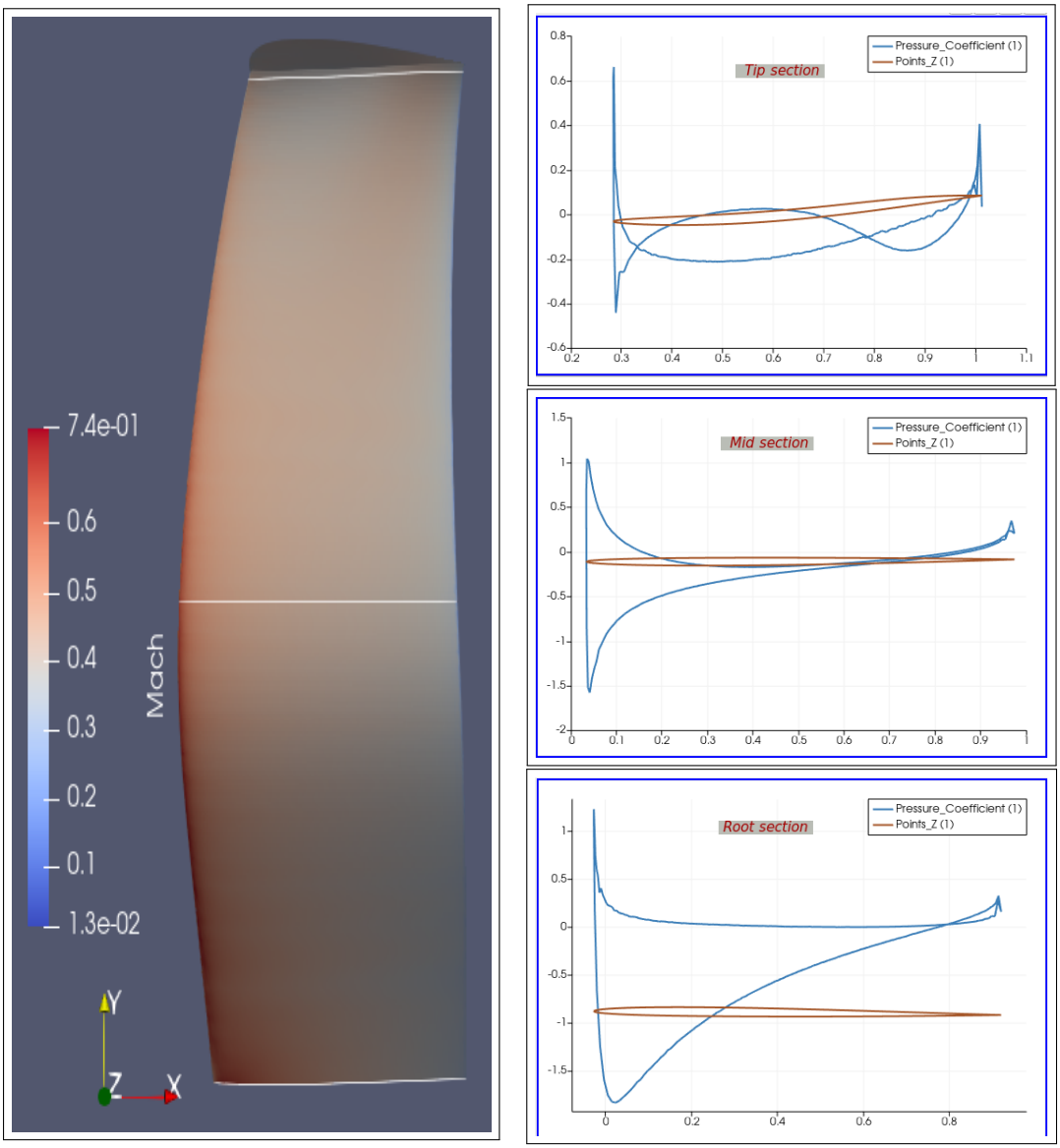


Figure 7.8: Cp distribution at various sections in local optima 1 along span directions.

	C_l	C_d	C_{M_x}	AoA	Span	Obj.diff
local optima 1	0.2625	0.01120	0.1038	5.535^0	2.779	0.0%
local optima 2	0.2625	0.01203	0.1005	3.213^0	2.918	7.4%
local optima 3	0.2625	0.01204	0.1028	1.972^0	2.726	7.5%
local optima 4	0.2625	0.01215	0.998	2.774^0	3.291	8.4%

Table 7.2: C_d values for various local optima.

points. To reduce the computational cost, the number of mesh points is kept on the lower side (0.25m). With the same number of mesh points, the grid sensitivity analysis is carried out for baseline geometry, and the results (cp curve) seems acceptable.

Following this, the same glyph script, which is meant for volume mesh generation, is used to generate the volume mesh for perturbed wings. However, at the end of full optimization, it was found that the grid resolution and grid clustering algorithm was not adequate, which resulted in erroneous flow-field. Also, at some sections in the optimal wings, inverted airfoils are generated. The cp curve shows inappropriate behaviour at those sections and is propagated through all sections of the given wing. Figure 7.8 depicts the same.

The present work aimed to identify the existence of multimodality in a given design space. However, due to the present situation, COVID-19 pandemic, the work is partially completed. Also, glyph script can not be corrected remotely.

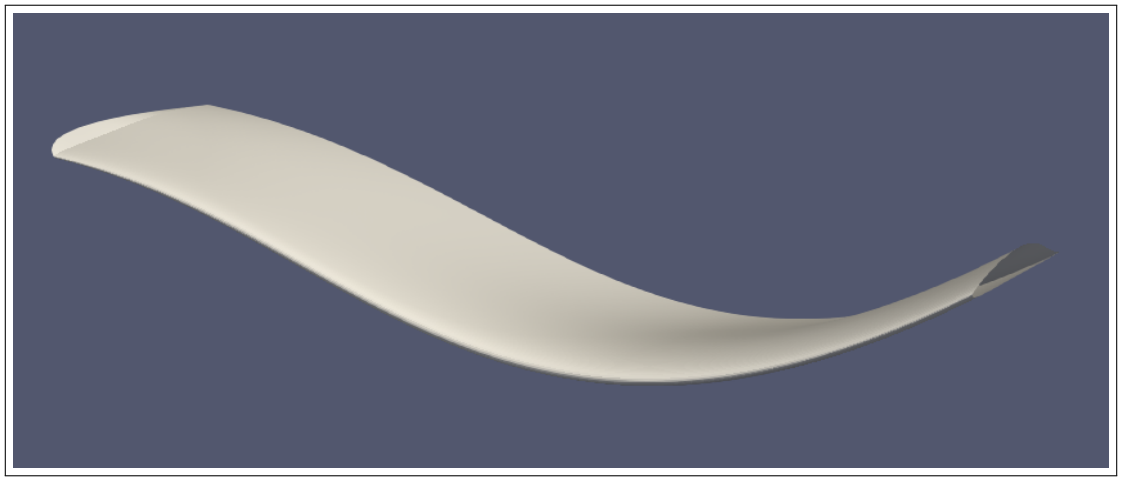


Figure 7.9: Local optima 2

Assuming that the CFD solution is correct, figure 7.9 to 7.11 represents the local optima in the increasing order of their objective values. Local optima 2 (figure 7.9) show an anhedral behaviour at the root section and dihedral at the wing tip section. There exists a smooth transition between an anhedral and the dihedral wing sections. Local optima 3 (figure 7.10) contain the waveform at the wing-tip section. Furthermore, a small leading-edge sweepback angle is observed in the root section. Local optima 4 (figure 7.11) reach the upper limit of span (3.3 units).

The objective function of these optima varies by less than 9%. As compared to results in Chernukhin [17], the relative difference of objective function is on the higher side.

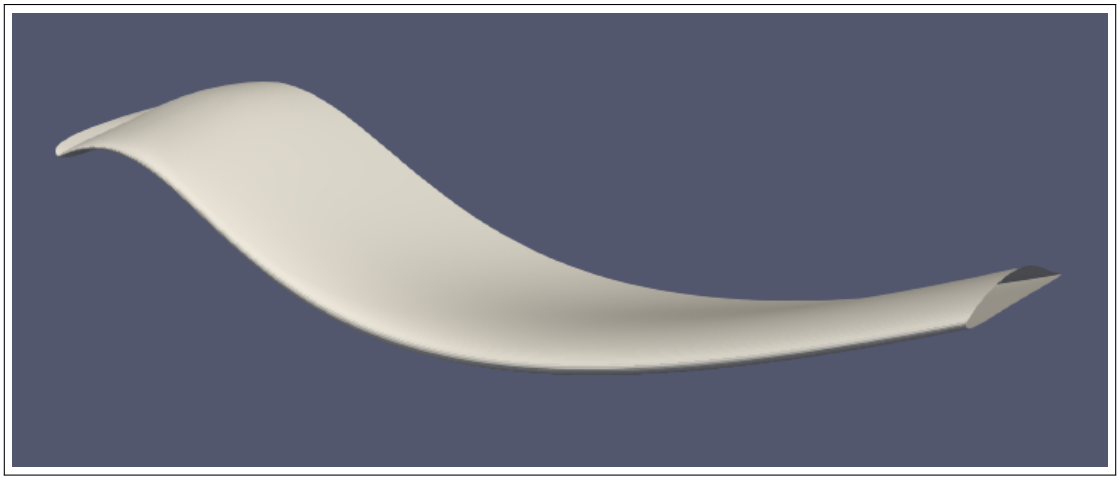


Figure 7.10: Local optima 3

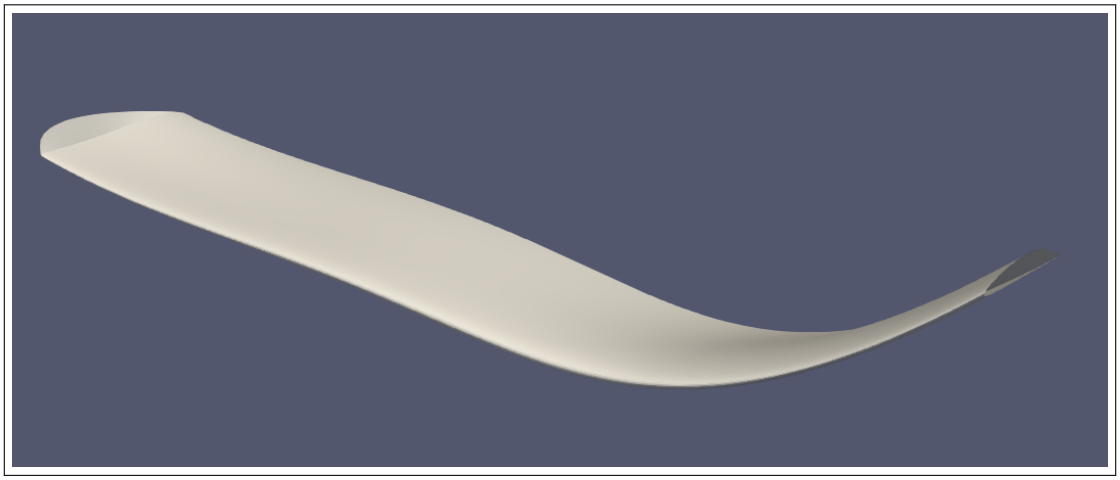


Figure 7.11: Local optima 4

The mesh size that is generated by the glyph script is five times lower. Furthermore, the objective function is found to be ten times higher.

7.6 Future work

This work addresses some of the issues with regards to the implementation of the FFD box, choice of design space, the dimension reduction techniques, winglet creations, and others. There exists some more critical question that needs to be addressed. As mentioned, the glyph script failed to reach the expectation. Further work is necessary to improve the performance of glyph script. Also, the choice of design space needs to be examined.

Further, niching algorithms requires computationally expensive CFD simulation. Reducing the number of CFD runs will significantly reduce the time taken for full optimization. As mentioned by Chernukhin [17], performing the optimization under viscous flowfield will decrease the number of optima. Once all the above objectives are addressed, the work can further extended to viscous flowfield.

Bibliography

[1] "Griewank function definition." [Online]. Available: https://dev.heuristiclab.com/trac.fcgi/wiki/Documentation/Reference/TestFunctions

[2] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, no. 11, pp. 341–359, 1997.

[3] D. J.Poole *et al.*, "Identifying multiple optima in aerodynamic design spaces," *AIAA*, 2018.

[4] A. Sóbester and A. I. J. Forrester, *Aircraft aerodynamic design: geometry and optimization*. United Kingdom: John Wiley & Sons, 2015.

[5] C. B. Allen *et al.*, "Constraint niching using differential evolution," *AIAA*, 2019.

[6] A. Dumont *et al.*, "Gradient-based optimization of crm wing-alone and wing-body-tail configurations by rans adjoint technique," *AIAA*, 2016.

[7] G. Carrier *et al.*, "Gradient-based aerodynamic optimization with the elsa software," *AIAA*, 2014.

[8] C. Lee *et al.*, "Aerodynamic shape optimization of benchmark problems using jetstream," *AIAA*, 2015.

[9] David.Zingg *et al.*, "Application of jetstream to a suite of aerodynamic shape optimization problems," *AIAA*, 2014.

[10] Nadarajah *et al.*, "Adjoint-based aerodynamic optimization of benchmark problems," *AIAA*, 2015.

[11] A.Martins *et al.*, "Aerodynamic shape optimization of the common research model wing-body-tail configuration," *Journal of Aircraft*, vol. 53, no. 1, pp. 276–293, 2016.

[12] N. Bons *et al.*, "Multimodality in aerodynamic wing design optimization," *AIAA*, vol. 57, no. 3, pp. 1004–1018, 2019.

[13] G. Streuber *et al.*, "Investigation of multimodality in aerodynamic shape optimization based on the reynolds-averaged navier–stokes equations," in *Proceedings of the 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Denver, CO, 2017.

[14] J. Martins, "Perspectives on aerodynamic design optimization," 01 2020.

[15] "Gradient free optimization - stanford university." [Online]. Available: http://adl.stanford.edu/aa222/Lecture_Notes_files/chapter6_gradfree.pdf

[16] O. Chernukhin and D. W.Zingg, "Multimodality and global optimization in aerodynamic design," *AIAA*, vol. 51, no. 6, 2013.

[17] O. Chernukhin, "Global optimization algorithms for aerodynamic design," Ph.D. dissertation, University of Toronto, 2011.

[18] C.B.Allen *et al.*, "Global optimization of wing aerodynamic optimization case exhibiting multimodality," *AIAA*, 2018.

[19] M. Meaux *et al.*, "Viscous aerodynamic shape optimization based on the discrete adjoint state for 3d industrial configurations," *European Congress on Compurational Methods in Applied Sciences and Engineering*, 2004.

[20] S. K. Nadarajah *et al.*, "Survey of shape parameterization techniques and its effect on three-dimensional aerodynamic shape optimization," *AIAA*, 2007.

[21] K. Deb, "An efficient constraint handling method for genetic algorithms," *ELSEVIER*, vol. 186, pp. 311–338, 2000.

[22] H. Zhehuang and C. Yidong, "An improved differential evolution algorithm based on adaptive parameter," *Journal of Control Science and Engineering*, vol. 2013, p. 5, 2013.

[23] Y. Ao and H. Chi, "Experimental study on differential evolution strategies," *IEEE*, 2009.

[24] D. J.Poole and C.B.Allen, "Constraint niching using differential evolution." [Online]. Available: https://www.researchgate.net/publication/328886741_Constrained_Niching_Using_Differential_Evolution

[25] K. A. De Jong, "Analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, University of Michigan, 1975.

[26] R. Thomsen, "Multimodal optimization using crowding-based differential evolution," *Congress on Evo- lutionary Computation*, 2004.

[27] H. Holland, J, *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[28] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, 1987.

[29] A. Petrowski, ""a clearing procedure as a niching method for genetic algorithms"," in *Proceedings of IEEE International Conference on Evolutionary Computation*, Nagoya, Japan, 1996.

[30] P. J. Clarkson *et al.*, "A species conserving genetic algorithm for multimodal function optimization," in *A species conserving genetic algorithm for multimodal function optimization*, vol. 10. Evolutionary Computation, 2002, pp. 207–234.

[31] X. Li, "Efficient differential evolution using speciation for multimodal function optimization," *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, 2005.

[32] M. G. Epitropakis *et al.*, "Finding multiple global optima exploiting differential evolution's niching capability," *IEEE Symposium on Differential Evolution (SDE), Paris, France*, 2011.

[33] Epitropakis *et al.*, "Multimodal optimization using niching differential evolution with index-based neighborhoods," *IEEE Congress on Evolutionary Computation, Brisbane, Australia*, 2012.

[34] X. Li *et al.*, "Seeking multiple solutions: An updated survey on niching methods and their applications," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 4, pp. 518–538, 2017.

[35] T. W. Soderberg *et al.*, "Free-form deformation of solid geometric models," *SIGGRAPH*, vol. 20, no. 4, 1986.

[36] V. Garzon, "Probabilistic aerothermal design of compressor airfoils," Ph.D. dissertation, Dept. of Aeronautics and Astronautics, MIT, 2002.

[37] D. Ghate, "Inexpensive uncertainty analysis for cfd applications," Ph.D. dissertation, University of OXFORD, 2014.

Appendix A

Formula related to GB

A.1 Gradient vector

Following equation represents the mathematical form to calculate gradient of n-dimensions.

$$c = df_{x^*} = \left[\frac{\partial f}{\partial x_1}(x*), \cdots, \frac{\partial f}{\partial x_n}(x*)\right]^T \tag{A.1}$$

where, c is gradient vector, $f(x)$ is function of x, and $\frac{\partial f}{\partial x_n}$ is the partial derivative of function with respect x^k.

A.2 Hessian matrix

If the given function $f(x)$ is twice differentiable, then the Hessian matrix will be,

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{A.2}$$

or in the index form it can be represented as,

$$\mathbf{H}_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j} \tag{A.3}$$

The Hessian matrix is symmetric, this implies,

$$\frac{\partial^2 f}{\partial x_j \partial x_i} = \frac{\partial^2 f}{\partial x_i \partial x_j} \tag{A.4}$$

A.3 Taylor series

Below equation is the n^{th} - order representation of taylor series in matrix notation,

$$f(x) = f(x^*) + \nabla f^T(x - x^*) + \frac{1}{2}(x - x^*)^T H(x - x^*) + R \tag{A.5}$$

where, ∇ is divergence of f, H is Hessian matrix, and R is residual of the equation.

Appendix B

Test function equations

B.1 Ackley function

$$f(\mathbf{x}) = -a \exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos\left(cx_i\right)\right) + a + \exp(1) \tag{B.1}$$

B.2 Egg holder function

$$f(\mathbf{x}) = -\left(x_2 + 47\right)\sin\left(\sqrt{\left|x_2 + \frac{x_1}{2} + 47\right|}\right) - x_1\sin(\sqrt{|x_1 - (x_2 + 47)|}) \tag{B.2}$$

B.3 Rastrigin function

$$f(\mathbf{x}) = 10d + \sum_{i=1}^{d}\left[x_i^2 - 10\cos\left(2\pi x_i\right)\right] \tag{B.3}$$

B.4 Sphere function

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2 \tag{B.4}$$

Appendix C

Design space of a problem

-46.09843825380174	46.09843825380174
-44.314212305940025	44.314212305940025
-35.431203844866744	35.431203844866744
-34.538072456635255	34.538072456635255
-24.650027391004958	24.650027391004958
-18.54560942319531	18.54560942319531
-14.94031887649444	14.94031887649444
-11.060172594761685	11.060172594761685
-8.556500608111564	8.556500608111564
-10.065740658558237	10.065740658558237
-6.771986021692888	6.771986021692888
-7.187241953459317	7.187241953459317
-6.7617795906794695	6.7617795906794695
-6.038968263125511	6.038968263125511
-6.6222576181178265	6.6222576181178265
-4.908118491814942	4.908118491814942
-4.942860889588044	4.942860889588044
-3.735500965329549	3.735500965329549
-3.903805735223391	3.903805735223391
-4.223239993663491	4.223239993663491