NUMERICAL OPTIMIZATION OF AIRFOIL & DEVELOPING A NOVEL METHOD FOR AIRFOIL SELECTION

PROJECT
Submitted to:

JHARKHAND UNIVERSITY OF TECHNOLOGY, RANCHI

In Partial Fulfillment of the Requirement for the Award of Degree of

BACHELOR OF TECHNOLOGY
IN
MECHANICAL ENGINEERING

(Session: 2023-24)

Submitted by:

Name of Student	Registration No.	Roll No.
ANKIT KUMAR SINGH	20030490023	2000024
ASHUTOSH KUMAR	20030490027	2000028
PRITY CHOUDHARY	20030490064	2000066
SHREYAS KUMAR TAH	20030490087	2000087

Under the Guidance of

Prof. (Dr.) Rajan Kumar

(Associate Professor, Mechanical Engineering, BIT Sindri)

Department of Mechanical Engineering
BIT SINDRI
Department of Higher, Technical Education & Skill Development (Govt. of Jharkhand)
P.O. Sindri Institute, Dhanbad-828123 (Jharkhand)

CANDIDATE'S DECLARATION

We hereby declare that the work carried out in this report titled **"Numerical Optimization of Airfoil & Developing a Novel Method for Airfoil Selection"** is presented on behalf of partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** with specialization in **Mechanical Engineering** submitted to the department of **Mechanical Engineering, B.I.T. Sindri, Jharkhand**, under the supervision and guidance of **Prof. (Dr.) RAJAN KUMAR**, Associate Professor, Department of Mechanical Engineering, B.I.T. Sindri, Jharkhand.

We have not submitted the matter embodied in this report for the award of any other degree or diploma.

Date: - _____

Place: - BIT Sindri

_____

ANKIT KUMAR SINGH (20030490023)

_____

ASHUTOSH KUMAR (20030490027)

_____

PRITY CHOUDHARY (20030490064)

_____

SHREYAS KUMAR TAH (20030490087)

Department of Mechanical Engineering
BIT SINDRI
Department of Higher, Technical Education & Skill Development Govt. of Jharkhand.
P.O. Sindri Institute, Dhanbad-828123 (Jharkhand)

DECLARATION CERTIFICATE

This is to certify that the Project Report entitled **"Numerical Optimization of Airfoil & Developing a Novel Method for Airfoil Selection"** submitted by Ankit Kumar Singh (20030490023), Ashutosh Kumar (20030490027) , Prity Choudhary (20030490064) and Shreyas Kumar Tah (20030490087), for the partial fulfillment of the requirement for the award of ***"BACHELOR OF TECHNOLOGY"*** *in* ***MECHANICAL ENGINEERING*** **(Session 2023-24)** has been carried out under the supervision and guidance of Prof. (Dr.) Rajan Kumar, Associate Professor, Department of Mechanical Engineering, BIT Sindri, Jharkhand. It is certified that the report embodies the results of the work carried out by them within the prescribed period.

To the best of my knowledge, the content of this project does not form a basis of the award of any previous degree to anyone else.

Project Guide

Prof. (Dr.) Rajan Kumar
Associate Professor
Department of Mechanical Engineering,
BIT Sindri, Dhanbad

Department of Mechanical Engineering
BIT SINDRI
Department of Higher, Technical Education & Skill Development Govt. of Jharkhand.
P.O. Sindri Institute, Dhanbad-828123 (Jharkhand)

CERTIFICATE OF APPROVAL

The forgoing project entitled **"Numerical Optimization of Airfoil & Developing a Novel Method for Airfoil Selection"** is hereby approved as a creditable project work carried out and has been presented in satisfactory manner to warrant its acceptance as prerequisite to the degree for which it has been submitted.

It is understood that by this approval, the undersigned do not necessarily endorse any conclusion drawn or opinion expressed there in but approve the project for the purpose for which it is submitted.

(Signature of Internal Examiner) **(Signature of External Examiner)**

Prof. (Dr.) Vijay Pandey
Head of Department
Dept. of Mechanical Engineering
B.I.T. Sindri, Dhanbad

Abstract

Design for automobiles, boats, and aircraft is being completely transformed by aerodynamic shape optimization, which methodically refines objects in fluid environments to maximize efficiency, minimize drag, and increase lift. Engineers examine flow patterns using Computational Fluid Dynamics (CFD) and objective functions to inform changes. Aerodynamic shape optimization is being advanced by the use of AI optimization approaches and sophisticated algorithms. An engineer's arsenal tool includes gradient-based algorithms, gradient-free algorithms, reinforcement learning, surrogate models, and many more evolutionary techniques, which enable them to navigate complex design environments effectively. But there are obstacles in the way of this endeavor. One major challenge is the cost of computation, which makes it necessary to devise ways to increase computational efficiency without sacrificing accuracy. Moreover, the multidisciplinary nature of aerodynamic shape optimization demands careful consideration of various factors, parameters, and constraints to select the best airfoil for a particular usage. In conclusion, this abstract provides a comprehensive overview of aerodynamic shape optimization, highlighting its transformative potential and the ongoing efforts to overcome associated challenges. As the field continues to evolve, it promises to reshape the landscape of engineering design, ushering in a new era of efficiency, performance, and innovation.

Contents

Chapter 1

Introduction

In the field of engineering, optimizing foil shapes and generic wing platforms is a crucial undertaking that utilizes sophisticated AI algorithms and numerical methodologies to achieve maximum aerodynamic performance and efficiency. This project report explores the possibility of combining state-of-the-art artificial intelligence with Computational Fluid Dynamics (CFD) techniques to transform the design and optimization of aerodynamic structures. Aerodynamic shape optimization is fundamentally a combination of sophisticated computer methods and engineering know-how. Engineers analyze intricate flow patterns and pressure distributions using methods like Computational Fluid Dynamics (CFD) to guide iterative design changes. The main goal is to minimize drag coefficients and increase performance indicators like lift-to-drag ratios. Nevertheless, numerical optimization's effectiveness depends on applying advanced AI algorithms in addition to complex simulation techniques. Artificial intelligence (AI) methods enable engineers to navigate large design spaces with unmatched efficiency, revealing new solutions and pushing the bounds of aerodynamic performance. These algorithms include gradient-based optimization approaches, evolutionary strategies, and PINN-based machine-learning techniques. The goal of this research report is to investigate how advanced AI algorithms, CFD simulations, and numerical optimization approaches may work together to create the best foil shapes and wing platforms. We aim to clarify the revolutionary potential of this multidisciplinary approach in transforming the field of aerodynamic design and engineering through an extensive analysis of methodology, difficulties, and possible applications.

1.1 Problem statement

The design and optimization of airfoil shapes are critical for improving the aerodynamic performance of various systems, such as aircraft, UAVs, and wind turbines. Traditional methods, such as XFoil and RANS CFD, are often hindered by high computational costs, lack of robustness, and the necessity for extensive theoretical knowledge. These challenges complicate the efficient design of optimal airfoil shapes, limiting accessibility and effectiveness.

Additionally, choosing the right airfoil for particular use cases proved to be a considerable challenge for us during our research. The conventional selection procedure is laborious and difficult for users with little theoretical background since it necessitates a thorough comprehension of aerodynamic properties at different stall angles.

In order to overcome these problems, we started a project to improve the effectiveness of airfoil optimization by utilizing contemporary AI-based technologies like NeuralFoil, an existing machine learning model based on PINNs and tried to compare it with the traditional method. In order to streamline the selection process, we have created a novel technique for filtering airfoils according to particular aerodynamic needs and application scenarios.

1.2 Objective

The objective is to optimize the lift-to-drag ratio or minimize drag by improving the shape of the airfoil to meet certain performance requirements in order to systematically enhance aerodynamic designs. Physics-Informed Neural Networks (PINNs), gradient-based techniques, and evolutionary strategies—among other numerical optimization tools—will be used to do this. Using parametric simulations and sensitivity analysis, the effects of different design parameters, including foil form, wing geometry, and surface features, will be examined. In order to evaluate accuracy and dependability, optimized designs will be tested by comparison with experimental data or benchmark examples. By offering creative ideas and industry-best practices for improving foil shapes and generic wing platforms, this initiative seeks to further aerodynamic engineering and has ramifications for raising performance, economy, and sustainability in a variety of sectors.

1.3 Summary

This project focuses on optimizing foil shapes and wing platforms using advanced AI algorithms and Computational Fluid Dynamics (CFD) to enhance aerodynamic performance. Traditional methods face high computational costs and complexity, making airfoil design and selection challenging, especially for users with limited theoretical knowledge. By leveraging contemporary AI technologies like NeuralFoil, the project aims to improve airfoil optimization and streamline the selection process based on specific aerodynamic needs. The objective is to maximize the lift-to-drag ratio or minimize drag, demonstrating the transformative potential of AI in aerodynamic design.

Chapter 2

Literature Review

Aerodynamics, the study of the motion of air and its effects on objects in motion, forms the cornerstone of numerous engineering disciplines, from aerospace to automotive engineering. The understanding and optimization of aerodynamic shapes are critical for enhancing the performance and efficiency of various vehicles, structures, and systems. Over the years, significant advancements have been made in both the theoretical understanding and computational techniques for analyzing and optimizing aerodynamic shapes.

This literature review aims to explore and compare traditional methods and modern approaches utilized in aerodynamic shape optimization. The focus will be on two-dimensional (2-D) and three-dimensional (3-D) aerodynamics, along with the application of conventional optimization methods such as gradient-based methods, surrogate-based methods, particle swarm optimization (PSO), and gradient-free methods. Additionally, this review will delve into the emerging trends in aerodynamic shape optimization, particularly the integration of machine learning (ML) techniques and physics-informed neural networks (PINNs) into the optimization process.

The study of 2-D aerodynamics has provided fundamental insights into flow behavior and lift and drag characteristics of airfoils, wings, and other planar structures. Classical methods such as panel methods, vortex lattice methods, and boundary element methods have been extensively used for predicting aerodynamic performance in 2-D flow regimes. These methods, while effective for certain applications, have limitations in accurately capturing complex flow phenomena and three-dimensional effects.

In contrast, three-dimensional aerodynamics introduces additional complexities due to the presence of spanwise variations and three-dimensional flow structures. Computational Fluid Dynamics (CFD) simulations have become the primary tool for analyzing 3-D aerodynamic behavior. However, the computational cost associated with high-fidelity CFD simulations often limits their application in optimization studies, particularly in large-scale parametric studies.

Conventional optimization methods have been widely employed in aerodynamic shape optimization to improve the performance of aerodynamic surfaces. Gradient-based methods, including steepest descent, quasi-Newton methods, and adjoint methods, offer efficient convergence but may struggle with highly nonlinear and discontinuous objective functions. Surrogate-based methods alleviate the computational burden by approximating the expensive objective functions with inexpensive surrogate models, such as response surface models and kriging models. Particle swarm optimization and other evolutionary algorithms provide robustness in handling complex, multimodal optimization problems.

Recent advancements in machine learning have sparked new avenues for aerodynamic shape optimization. Machine learning techniques, including artificial neural networks (ANNs), support vector

machines (SVMs), and genetic programming, have been integrated into optimization frameworks to accelerate the design process and enhance accuracy. Physics-informed neural networks, a subset of ML techniques, leverage physical principles to guide the learning process and improve generalization to unseen data.

2.1 2-Dimensional Aerodynamics

The science of Aerodynamics deals with the study of air flow over bodies and the effect of forces induced on the body, the evaluation of whose helps in estimating if the force requirements of force characteristics for the design specimen are met or not. On a more complex level, aerodynamic coefficients obtained are coupled with Rigid body Dynamics model of the aircraft and the essential study of stability is carried out. A full scale aerodynamic study considers full 3 dimensional effects of airflow over the aircraft body and naturally, for high fidelity 3-D aerodynamic simulation is required.

In our study, we are considering 2-Dimensional aerodynamics since our workflow involves the initial study of airfoils. Airfoil can be thought of as a 2-Dimensional cut cross-section of a wing or in other words a 3-D wing with infinite wing span such that the 2 dimensional aerodynamic properties are extended in the third dimension in both directions.

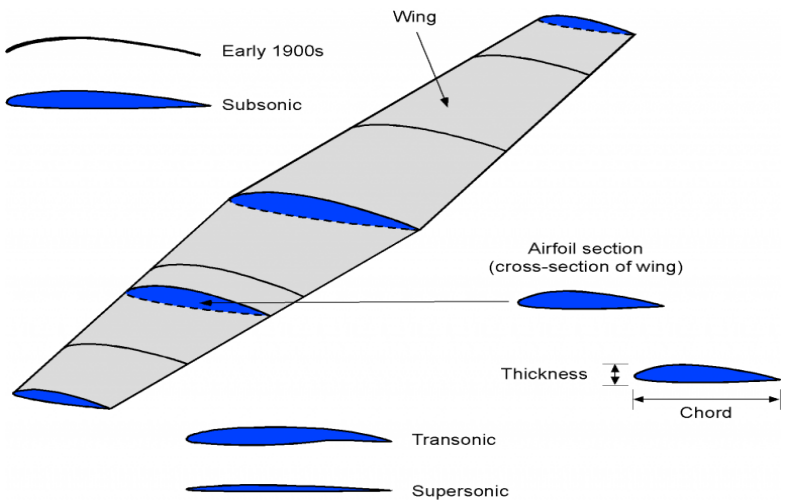

Figure 2.1: An airfoil depicted as cut section of a wing

A 2-D airfoil, a fundamental concept in aerodynamics, represents the cross-sectional shape of an aircraft wing, propeller blade, or any aerodynamic surface. In two dimensions, it is defined by key parameters that determine its geometry and aerodynamic performance. At its core lies the chord line, extending from the leading edge to the trailing edge, defining the airfoil's length. The leading edge marks the point where airflow first interacts with the surface, while the trailing edge is where the airflow separates from the airfoil.

The key components of a 2-D airfoil include:

1. **Chord Length (c):** The chord length is the distance from the leading edge to the trailing edge along the chord line. It serves as the reference length for aerodynamic calculations.

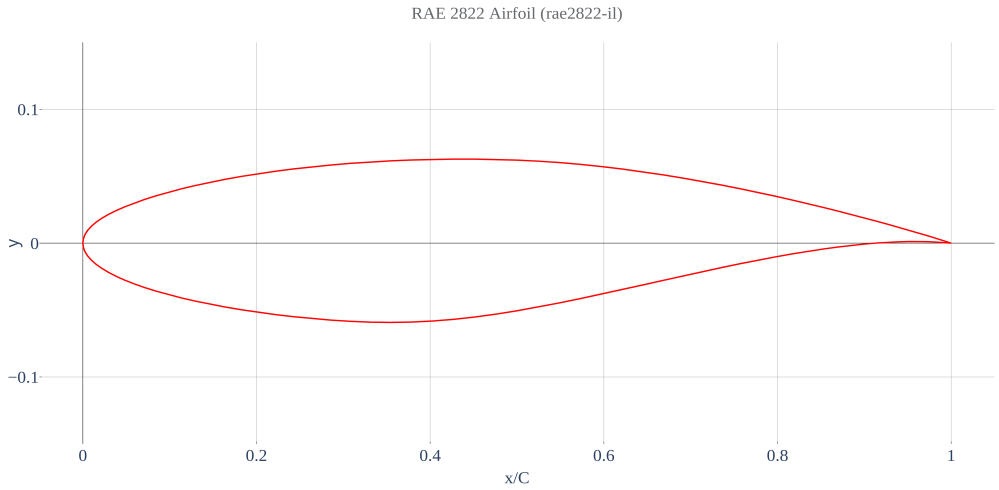

Figure 2.2: A 2-D Airfoil (rae2822-il)

2. **Leading Edge:** This is the foremost point of the airfoil where the airflow first encounters the surface. It's where the relative wind starts.

3. **Trailing Edge:** The trailing edge is the rear edge of the airfoil where the airflow exits. It's where the relative wind leaves the airfoil.

4. **Camber:** Camber refers to the curvature of the airfoil's upper and lower surfaces. It's the maximum distance between the chord line and the mean camber line.

5. **Thickness:** Thickness is the maximum distance between the upper and lower surfaces of the airfoil, measured perpendicular to the chord line.

2.2 Numerical Optimization Techniques

Techniques for numerical optimization are essential for methodically improving aerodynamic designs to meet performance goals. Gradient-based techniques, like gradient descent, follow the direction of the steepest descent in the objective function landscape and repeatedly use derivatives to improve designs towards optimal solutions. By iteratively growing a population of candidate solutions, evolutionary algorithms, such as genetic algorithms, and particle swarm optimization, efficiently explore and exploit design spaces by emulating the principles of natural selection. With the use of neural networks to solve partial differential equations (PDEs) governing fluid flow, physics-informed neural networks (PINNs) offer a novel optimization technique that directly incorporates physical rules into the optimization process. By combining neural network architectures with physics-based constraints, PINNs enable efficient exploration of design spaces while ensuring solutions adhere to fundamental principles of aerodynamics.

The CFD based optimization framework that was used for this project is based on OpenFoam and uses discrete adjoint method for computation of gradients. DAFoam (Discrete Adjoint Foam) is a computational fluid dynamics (CFD) framework designed for aerodynamic shape optimization using the discrete adjoint method. This approach involves the numerical differentiation of the discretized flow equations to compute gradients of a cost function with respect to design variables, which are essential for gradient-based optimization.

DAFoam uses the adjoint method to efficiently compute the total derivatives $\partial f / \partial \mathbf{x}$, where f is the objective or constraint function (e.g., drag, lift, or torque) and $\boldsymbol{x}$ is the vector of design variables

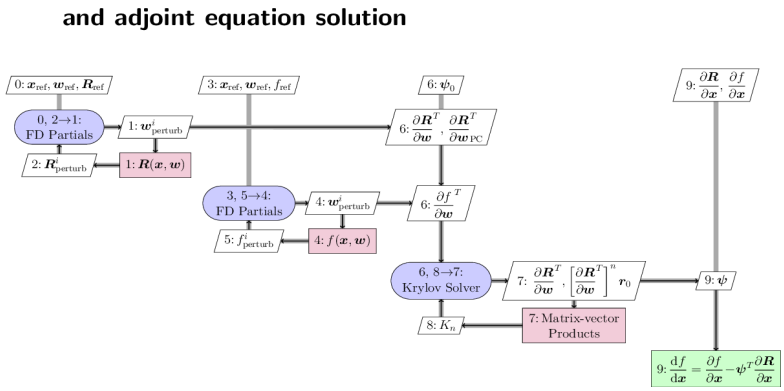

Figure 2.3: Solver-agnostic FD Jacobian adjoint approach in DAFoam

Algorithm 1 Discrete Adjoint Optimization for Airfoil Shape

Input: Initial airfoil shape S_0, target performance metrics P_{target}, convergence threshold ϵ
Output: Optimized airfoil shape S^*
Initialize airfoil shape $S \leftarrow S_0$
Initialize iteration counter $k \leftarrow 0$
Initialize convergence criterion *converged* $\leftarrow$ False
while not *converged* **do**
    $k \leftarrow k + 1$
    Solve the primal flow equations to obtain the flow field variables **U**
    Compute the objective function $J(\mathbf{U}, S)$ and its gradient w.r.t shape S
    Solve the adjoint equations to obtain the adjoint variables $\mathbf{\Psi}$
    Calculate the gradient of the objective function using the adjoint variables
    Update the airfoil shape S using a suitable optimization method (e.g., gradient descent, quasi-Newton)
    Evaluate the new performance metrics P_{new}
    if $|P_{new} - P_{target}| < \epsilon$ **then**
        converged $\leftarrow$ True
    end if
end while
$S^* \leftarrow S$
Return S^*

(e.g., positions of control points that morph the design surface).

In the discrete approach, it is assumed that a discretized form of governing equations is available through the primal solver, and that the design variable vector

$$\boldsymbol{x} \in \mathbb{R}^{n_x} \quad \text{and} \quad \boldsymbol{w} \in \mathbb{R}^{n_w} \quad \text{satisfy the discrete residual equations} \quad \boldsymbol{R(x,w)} = 0,$$

where $\boldsymbol{R} \in \mathbb{R}^{n_w}$ is the residual vector.

2.3 The use of Advanced AI

Theory-trained neural networks (TTNs), also known as physics-informed neural networks (PINNs), are a class of universal function approximators that may be represented by partial differential equations (PDEs) and can incorporate into the learning process the knowledge of any physical rules that govern a given data set. PINNs are a cutting-edge technology that has enabled the development of new classes of numerical solvers for PDEs and enabled the solution of a wide variety of computer science issues. Neural

networks with a physics background (PINNs) are a powerful tool for solving problems with sparse or noisy experimental data. These networks function as flexible tools in the field of supervised learning because of their exceptional capacity to combine known input while strictly following predetermined physical principles represented by intricate nonlinear partial differential equations.

PINNs can be viewed as a mesh-free substitute for innovative data-driven methods for system identification and model inversion and more conventional techniques (such as CFD for fluid dynamics). They leverage deep learning to provide accurate and efficient solutions to complex problems by embedding the governing physical laws directly into the neural network training process. This results in models that not only fit the data but also respect the underlying physics, making them particularly useful for applications in aerodynamics, structural analysis, and beyond.

Even if PINN capabilities have been enhanced by published research, there are still a lot of unanswered questions. These are wide-ranging and include anything from theoretical issues like convergence and stability to practical difficulties like neural network design, general PINN architecture, boundary condition management, and optimization elements. Addressing these challenges will further unlock the potential of PINNs, making them indispensable tools in scientific computing and engineering design.

2.4 Aero-Sandbox

AeroSandbox is a Python program and flexible computational framework that makes it easier to build, optimize, and analyze aircraft and other engineered systems. With its easy-to-use Python interface, individuals with different degrees of programming experience can utilize it. Critical aerodynamic parameters including lift, drag, and pressure distributions may be calculated with this tool, which is excellent at performing extensive aerodynamic evaluations of airfoil shapes, wings, and entire aircraft. Its capacity to incorporate sophisticated optimization algorithms is one of its most notable characteristics. This allows users to effectively optimize aerodynamic forms to satisfy particular performance requirements, such optimizing lift-to-drag ratios or lowering drag. AeroSandbox enables multidisciplinary optimization by integrating control systems, propulsion, and structural analysis. This enables comprehensive design optimization across a range of performance factors. The framework is excellent for rapid prototyping and iterative design processes, and it facilitates integration with other tools and libraries. It is also quite flexible. The accuracy and speed of aerodynamic predictions can also be improved by integrating AeroSandbox with machine learning models such as NeuralFoil. AeroSandbox is an invaluable tool for professional aerospace engineering applications as well as educational reasons. Its substantial documentation and community assistance further add to its value.

2.4.1 Numerical Backends and Frameworks

In AeroSandbox, the CasADi package provides an end-to-end differentiable backend, though it has some limitations. These include the inability to create arrays with more than two dimensions, unconventional sparse matrix formats, and a lack of parallelization capabilities due to the inability to serialize graph structures as-is. Future work could explore offering multiple numerical backends. While maintaining multiple backends might initially seem challenging, the relative isolation of the aerosandbox.numpy module within the AeroSandbox library could allow for their implementation with minimal effort. Other Python automatic differentiation backends, such as JAX and PyTorch, address many issues CasADi presents and are strong candidates for inclusion. However, they each come with their limitations. JAX, for instance, lacks Windows binaries and has a more limited library of differentiable primitive operators compared to CasADi. PyTorch, on the other hand, tends to delete the computational graph from memory after each backward pass—a feature useful in batch machine learning but unsuitable for engineering design optimization where graphs are generally static. Despite their broader use, both JAX and PyTorch do not include interfaces to second-order gradient-based solvers like IPOPT. In the long term, Julia-based automatic differentiation frameworks, particularly Zygote,

could become more attractive. This potential depends on the increased maturity and stability of the core language, greater user adoption, and the development of a robust scientific package ecosystem.

2.5 NeuralFoil

NeuralFoil is a sophisticated tool for analyzing aerofoil aerodynamics that forecasts the performance of various airfoil geometry using physics-based machine learning. It is an innovative tool that provides quick and precise predictions of aerodynamic parameters including lift, drag, and pressure distributions. While classic tools like XFoil rely on solving intricate fluid dynamics equations, NeuralFoil employs neural networks trained on large datasets of airfoil simulations. Compared to classic airfoil analysis tools such as XFoil and RANS CFD, NeuralFoil offers a significant improvement and is especially well-suited for contemporary aerodynamic applications due to its combination of speed, efficiency, and ease of integration. Due to its speed advantage over XFoil, it can evaluate large batches of airfoil cases more quickly than XFoil, which is crucial for effective design and optimization procedures. NeuralFoil is also made to be flexible; it may be used as an integrated module within Aerosandbox or as a stand-alone module. This allows for flexibility in a variety of applications, such as real-time control on embedded systems and flight simulation, and streamlines dependencies. NeuralFoil's stability and dependability are two of its main advantages. NeuralFoil consistently produces dependable results, in contrast to XFoil, which can crash under demanding computations. This is an important characteristic for real-time applications and design optimization when continuous performance is necessary. In addition, NeuralFoil's Python and NumPy implementation makes it simple to install and use on several systems, in contrast to the Fortran-based XFoil, which has more complicated compilation requirements. NeuralFoil's architecture is made up of feed-forward neural networks that have been extensively trialed and errored to optimize their layers and layer widths. This guarantees great accuracy while making effective use of computing resources. The architecture of NeuralFoil is designed to ensure C1-continuity, which means that the first derivative of its output with respect to its input is continuous. The architecture is as follows:

- xxsmall: 2 layers, 32 neurons wide
- xsmall: 3 layers, 32 neurons wide
- small: 3 layers, 48 neurons wide
- medium: 4 layers, 64 neurons wide
- large: 4 layers, 128 neurons wide
- xlarge: 4 layers, 256 neurons wide
- xxlarge: 5 layers, 256 neurons wide
- xxxlarge: 5 layers, 512 neurons wide

All layers are fully connected (dense) with weights and biases along with ReLU used as the activation functions between layers to ensure C1-continuity ensuring smooth gradient calculations.

With the help of intensive training on tens of millions of XFoil runs, NeuralFoil can accurately forecast a wide range of aerodynamic conditions and generalize well to new, unexplored airfoils. Due to the high computational expense of obtaining enough training data, neural networks trained on RANS CFD have difficulty achieving accurate out-of-sample predictions. This broad dataset exposure is essential for this. To sum up, NeuralFoil blends resilience, speed, precision, and ease of integration. Its sophisticated neural network architecture guarantees accurate and dependable performance, meeting the many requirements of contemporary aerodynamic applications. This sets it apart from more conventional tools such as XFoil and other computational techniques.

2.6 Computational Fluid Dynamics using Finite Volume Method

The framework which was used for Aerodynamic Optimization using gradient-based method, DAFoam, is based on OpenFoam. OpenFoam in turn uses the Finite Volume Formulation for the computation of fluid flow.

OpenFOAM (Open Source Field Operation and Manipulation) is a versatile and widely-used open-source software package primarily designed for computational fluid dynamics (CFD) but also applicable to a range of multi-physics simulations. Developed by OpenCFD Ltd and distributed by the OpenFOAM Foundation, it provides an extensive suite of tools for solving complex fluid flows involving turbulence, heat transfer, and chemical reactions. OpenFOAM's flexibility stems from its modular C++ library, which allows users to develop custom solvers and utilities tailored to specific applications. It supports a variety of mesh types, including structured, unstructured, and polyhedral grids, and offers advanced features like dynamic mesh handling, parallel processing, and pre- and post-processing capabilities. The software's open-source nature encourages collaboration and innovation within a global community of researchers, engineers, and scientists, who contribute to its continuous development and refinement. OpenFOAM's applications span numerous industries, including automotive, aerospace, energy, and environmental engineering, making it a critical tool for both academic research and industrial practice.

The governing equations on which the solvers are based on for fluid mechanics are described next.

2.6.1 Governing Equations

In Computational Fluid Dynamics (CFD), the governing equations are mathematical models that describe the behavior of fluid flow. These equations are derived from fundamental principles of physics and are used to predict how fluids (liquids and gases) move and interact with their surroundings. The primary governing equations in CFD are the Navier-Stokes equations, which are based on the principles of conservation of mass, momentum, and energy.

The Continuity Equation

The continuity equation ensures that mass is conserved within a fluid flow. It states that the rate of change of mass within a control volume is equal to the net mass flux across the control volume boundaries. For an incompressible fluid, the continuity equation is:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_i}{\partial x_i} = 0 \tag{2.1}$$

For incompressible flow, the density is constant. Therefore eq 2.1 reads

$$\frac{\partial \rho u_i}{\partial x_i} = 0 \tag{2.2}$$

The Momentum Equation

The Navier-Stokes equations describe the conservation of momentum, essentially Newton's second law of motion applied to fluid elements. They account for forces due to pressure, viscous stresses, and external forces. The general form of the Navier-Stokes equations is:

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_j u_i}{\partial x_j} = -\frac{\partial P}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} + \rho f_i \tag{2.3}$$

Where, for a Newtonian fluid,

$$\tau_{ij} = \mu(\partial u_i/\partial x_j + \partial u_j/\partial x_i) - \frac{2}{3}\mu \partial u_k/\partial x_k \tag{2.4}$$

For constant μ incompressible flow and neglecting body forces, equation 2.3 can be written in conservative form as:

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial P}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_i \partial x_j} \tag{2.5}$$

Here, ν is the kinematic viscosity defined as the ratio of dynamic viscosity and fluid density.

The Energy Equation

The energy equation accounts for the conservation of thermal energy within the fluid. It combines the first law of thermodynamics with the principles of fluid dynamics. The general form of the energy equation is:

$$\frac{\partial \rho E}{\partial t} + \frac{\partial \rho u_j E}{\partial x_j} = -\frac{\partial p u_j}{\partial x_j} + \frac{\partial u_i \tau_{ji}}{\partial x_j} + \frac{\partial}{\partial x_j}\left(k \frac{\partial T}{\partial x_j}\right) + S_E \tag{2.6}$$

In the set of partial differential equations, the unknown thermodynamic variables are : $\rho, p, i, and T$ Equations of state relate the other variables to the two state variables. Using ρ and T as state variables, we have state equations for p and i :

$$p = p(\rho, T) \quad and \quad i = i(\rho, T) \tag{2.7}$$

2.7 Finite Volume Method

The set of partial differential equations is converted into a set of linear algebraic equations by the finite volume approach. However, the discretization process utilized in the finite volume technique is unique and consists of just two simple phases. The partial differential equations are integrated and converted into balance equations over an element in the first stage. This entails employing an integrated quadrature of a specific order of precision to convert the surface and volume integrals into discrete algebraic relations over elements and their surfaces. A collection of semi-discretized equations is the end product. The second phase involves selecting interpolation profiles to approximate the variation of the variables inside the element, connecting the surface values of the variables to their cell values, and converting the algebraic relations into algebraic equations.

Invoking Gauss' Theorem for volume integrals is the essential first step before discretization of surface integrals.

Gauss's theorem, expressed using the divergence theorem in tensor notation, relates the volume integral of a vector field ϕ over a region CV to the surface integral of the vector field's flux ϕ through the closed surface S enclosing the region:

$$\int_{CV} \frac{\partial \phi_i}{\partial x_i}\, dV = \oint_S \phi_i n_i\, dS \tag{2.8}$$

where:

- $\int_C V$ represents the volume integral over Control Volume region CV,
- $\oint_S$ represents the surface integral over the closed surface ∂V,
- $\frac{\partial \phi_i}{\partial x_i}$ is the divergence of vector field ϕ,
- dV represents an infinitesimal volume element,
- n_i is the outward unit normal vector to the surface S, and

- dS represents an infinitesimal outward-pointing surface area element on the closed surface S.

The flow is laminar and steady and thus the discretization of temporal terms is not considered. We discuss the discretization of advective terms and diffusion terms in next sub-sections.

2.7.1 Discretization of Convective Term

The convective term in the Navier-Stokes equations accounts for the transport of momentum due to the bulk motion of the fluid. Before a suitable method of the discretization of the advection term can be deployed, it is converted to a simpler form using the Gauss Theorem :

$$\int_{CV} \frac{\partial \rho u_j \phi_i}{\partial x_j} \, dV = \sum_k \oint_{S_k} n_j \rho u_j \phi_i \, dS_k \tag{2.9}$$

where the integer k denotes the number of faces of the control volume.

We have used the bounded Upwind Scheme for the discretization of the Convective term. The linear upwind scheme is a discretization method commonly used in computational fluid dynamics to solve advection-dominated problems.

Consider a scalar quantity ϕ that is being advected by the velocity field in which the advection term is modeled as $\frac{\partial \rho u_i \phi}{\partial x_i}$.

The linear upwind scheme approximates the gradient of ϕ using a one-sided difference based on the direction of the velocity field:

$$\frac{\partial \phi}{\partial x} \approx \begin{cases} \frac{\phi_{i+1,j,k} - \phi_{i,j,k}}{\Delta x} & \text{if } U_{i,j,k} > 0 \\ \frac{\phi_{i,j,k} - \phi_{i-1,j,k}}{\Delta x} & \text{if } U_{i,j,k} \leq 0 \end{cases} \tag{2.10}$$

where:

- $\phi_{i,j,k}$ represents the value of ϕ at grid point (i, j, k),
- $U_{i,j,k}$ is the x-component of the velocity at grid point (i, j, k),
- Δx is the grid spacing in the x-direction.

This scheme ensures that the numerical solution uses information from the upstream direction of the velocity field, reducing numerical diffusion for advection-dominated problems.

Bounding the linear upwind scheme involves incorporating the velocity gradient to prevent excessive numerical oscillations. This can be achieved by limiting the gradient used in the linear upwind discretization. In index notation (i, j, k), the mathematical formulation of the bounded linear upwind scheme can be expressed as follows:

To bound this scheme using the velocity gradient, we can modify the gradients as follows:

$$\frac{\partial (\mathbf{U}\phi)}{\partial x} \approx \begin{cases} \min\left(\frac{\phi_{i+1,j,k} - \phi_{i,j,k}}{\Delta x}, \max(0, \nabla U_{i,j,k})\right) & \text{if } U_{i,j,k} > 0 \\ \min\left(\frac{\phi_{i,j,k} - \phi_{i-1,j,k}}{\Delta x}, \max(0, -\nabla U_{i,j,k})\right) & \text{if } U_{i,j,k} \leq 0 \end{cases} \tag{2.11}$$

where $\nabla U_{i,j,k}$ represents the gradient of the x-component of the velocity at the grid point (i, j, k). The min function ensures that the bounded gradient is always smaller than or equal to the original linear upwind gradient, and the max function prevents the gradient from becoming negative in the opposite direction of the velocity.

By incorporating the velocity gradient in the bounded linear upwind scheme, numerical oscillations can be mitigated while still capturing the advection characteristics of the flow.

By incorporating the velocity gradient in the bounded linear upwind scheme, numerical oscillations can be mitigated while still capturing the advection characteristics of the flow.

2.7.2 Discretization of Diffusion Term

The diffusion term in the Navier-Stokes equations (which is of the form : $\frac{\partial}{\partial x_i}\left(\Gamma \frac{\partial u_i}{\partial x_i}\right)$) represents the effects of molecular viscosity in fluid flow. It describes how momentum is transferred between adjacent fluid particles due to their relative motion and interaction on a microscopic scale. The integration over control volume and conversion into surface integrals take the form :

$$\int_{CV} \frac{\partial}{\partial x_i}\left(\Gamma \frac{\partial}{\partial x_k}\phi\right) dV = \sum_k \oint_{S_k} n_j \Gamma u_j \frac{\partial \phi}{\partial x_j} dS_k \tag{2.12}$$

Calculating gradients at the surface of the control volumes will be needed for this. The gradients of the cell centers can be calculated, and the value on the cell faces can then be determined using interpolation. The average value across the cell is essentially used for estimating the gradient's value at the cell center P:

$$\left(\frac{\partial \phi}{\partial x_i}\right)_P \approx \frac{\int_\Omega \frac{\partial \phi}{\partial x_i} d\Omega}{\Delta \Omega} \tag{2.13}$$

This is further simplified using the Finite Volume Method approximation and can be written as:

$$\left(\frac{\partial \phi}{\partial x_i}\right)_P \approx \frac{\sum_k \phi_k \Delta S_k n_i^{(k)}}{\Delta \Omega} \tag{2.14}$$

where $n_i^{(k)}$ is the outward normal of the surface k in the control volume. After being calculated, the gradients can then be interpolated to the faces. After being calculated, the gradients can then be interpolated to the faces.

2.7.3 Steady-State DDT Scheme

The "ddtSchemes" dictionary in OpenFOAM allows users to specify how the time derivative terms are handled. For steady-state simulations, two common approaches are:

1. Steady-State DDT Scheme: This explicitly sets the time derivative to zero. In this case, the governing equations are solved without considering any transient effects, focusing entirely on the spatial terms. For a steady-state simulation, the time derivative of any scalar field ϕ is set to zero. Mathematically, this can be represented as:

$$\frac{\partial \phi}{\partial t} = 0 \tag{2.15}$$

This simplification implies that the partial differential equations (PDEs) governing the fluid flow reduce to a form where only the spatial terms are considered, eliminating any transient terms.

2. Implicit Time-Stepping Schemes: Even in steady-state simulations, implicit schemes like the Euler implicit scheme can be used for iterative convergence purposes. These schemes treat the time derivative in a manner that aids the solver in reaching a steady-state solution iteratively, although the time step is not physically meaningful in this context. While the Euler implicit scheme is typically a first-order time discretization method used for transient simulations, it can also be applied iteratively in

steady-state simulations to help the solver converge. The implicit discretization of the time derivative for a scalar field ϕ in this context can be represented as:

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} = 0 \tag{2.16}$$

In a steady-state simulation, Δt (the time step) is not physically relevant, but it serves as a relaxation parameter to help the solver stabilize and converge.

2.7.4 Turbulence Modeling in CFD

All turbulent motion, also referred to as eddies, is included in turbulent flow, which has unstable pressure and velocity. These eddies have different spatial sizes; the largest eddies correspond to the length of the largest physical scale, while the tiniest eddies are those where dissipation occurs. Kinetic energy is converted to internal energy during dissipation.

There are two methods for solving turbulent flow issues numerically: Turbulence Models and Direct Numerical Simulation (DNS). Given that it provides a perfect answer for the flow field in both space and time, DNS is the most accurate approach currently in use. Nevertheless, this approach is only practical at low Reynolds numbers. The grid must be finer to resolve the Kolmogorov scales the higher the Reynolds number since the smallest length and time scales decrease with $Re^{-3/4}$ and $Re^{-3/4}$ respectively.

Turbulence models are utilized to solve the Navier-Stokes equation at a reasonable computational cost. They are essentially a Navier-Stokes equation approximation. These models are hybrid RANS/LES models, Reynolds Averaged Navier-Stokes (RANS) models, and Large Eddy Simulation (LES) models.

Reynolds-Averaged Navier-Stokes (RANS)

RANS models decompose the instantaneous velocity into a mean and fluctuating component, averaging the Navier-Stokes equations over time. This approach simplifies the equations by focusing on the mean flow, at the expense of modeling the effects of turbulence.

Mathematical Formulation

The instantaneous velocity $\mathbf{u}$ is decomposed into mean $\overline{\mathbf{u}}$ and fluctuating $\mathbf{u}'$ components:

$$\mathbf{u} = \overline{\mathbf{u}} + \mathbf{u}' \tag{2.17}$$

The Reynolds-averaged Navier-Stokes equations for incompressible flow are:

$$\frac{\partial \overline{u}_i}{\partial t} + \overline{u}_j \frac{\partial \overline{u}_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial \overline{p}}{\partial x_i} + \nu \frac{\partial^2 \overline{u}_i}{\partial x_j^2} - \frac{\partial \overline{u_i' u_j'}}{\partial x_j} \tag{2.18}$$

The term $\overline{u_i u_j}$ represents the Reynolds stresses, which require modeling.

Common RANS Models:

- $k - \epsilon$ **Model:** Uses two transport equations for turbulent kinetic energy (k) and its dissipation rate (ϵ).
- $k - \omega$ **Model:** Uses two transport equations for turbulent kinetic energy (k) and specific dissipation rate (ω).
- **Reynolds Stress Model (RSM):** Directly models the Reynolds stresses using transport equations for each component.

Large Eddy Simulation

LES resolves the large-scale turbulent structures directly and models the smaller scales using a subgrid-scale model. This approach is more accurate than RANS for capturing transient and spatially varying turbulence but is more computationally expensive.

Mathematical Formulation
The filtered Navier-Stokes equations for LES are:

$$\frac{\partial \overline{u}_i}{\partial t} + \overline{u}_j \frac{\partial \overline{u}_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial \overline{p}}{\partial x_i} + \nu \frac{\partial^2 \overline{u}_i}{\partial x_j^2} - \frac{\partial \tau_{ij}}{\partial x_j} \tag{2.19}$$

The subgrid-scale (SGS) stress tensor τ_{ij} is modeled to account for the effects of unresolved scales:

$$\tau_{ij} = \overline{u_i u_j} - \overline{u}_i \overline{u}_j \tag{2.20}$$

Common RANS Models:

- **Smagorinsky Model:** Relates the SGS stress to the strain rate with a constant coefficient.
- **Dynamic Smagorinsky Model:** Adjusts the Smagorinsky constant dynamically based on local flow conditions.

Large Eddy Simulation

DNS resolves all scales of turbulence directly by solving the Navier-Stokes equations without any modeling. This approach provides the most accurate results but is extremely computationally intensive and impractical for most engineering applications.

In our workflow, we have used the Spalart-Allmaras model for Finite Volume Method based shape optimization. The Spalart-Allmaras model is a one-equation turbulence model specifically designed for aerospace applications. It solves a single transport equation for a modified turbulent viscosity, making it computationally efficient and suitable for external aerodynamics. The Spalart–Allmaras model allows for cost-effective boundary layer calculations in external aerodynamics and only requires one transport equation for kinematic eddy viscosity parameter $\overline{\nu}$. It also specifies a length scale using an algebraic method.

Mathematical Formulation The transport equation for the modified turbulent viscosity $\overline{\nu}$ is :

$$\frac{\partial \tilde{\nu}}{\partial t} + u_j \frac{\partial \tilde{\nu}}{\partial x_j} = C_{b1}(1 - f_{t2})\tilde{S}\tilde{\nu} + \frac{1}{\sigma}\left[\frac{\partial}{\partial x_j}\left((\nu + \tilde{\nu})\frac{\partial \tilde{\nu}}{\partial x_j}\right) + C_{b2}\frac{\partial \tilde{\nu}}{\partial x_j}\frac{\partial \tilde{\nu}}{\partial x_j}\right] - \left(C_{w1} f_w - \frac{C_{b1}}{\kappa^2} f_{t2}\right)\left(\frac{\tilde{\nu}}{d}\right)^2 \tag{2.21}$$

where,

The modified vorticity $\tilde{\nu}$ is defined as:

$$\tilde{S} = S + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v1} \tag{2.22}$$

The damping function f_w is given by:

$$f_w = g\left[\frac{1+C_{w3}^6}{g^6+C_{w3}^6}\right]^{\frac{1}{6}} \tag{2.23}$$

g is defined as :

$$g = r + C_{w2}(r^6 - r) \tag{2.24}$$

r is defined as :

$$r = \frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2} \tag{2.25}$$

In the $k-\epsilon$ model the length scale is found by combining the two trans- ported quantities k and ϵ. In a one-equation turbulence model the length scale cannot be computed, but must be specified to determine the rate of dissipation of the transported turbulence quantity.

These model constants and three further ones hidden in the wall functions were tuned for external aerodynamic flows, and the model has been shown to give good performance in boundary layers with adverse pressure gradients, which are important for predicting stalled flows.

2.8 Previous studies and research

1. Adjoint-Based Aerodynamic Shape Optimization Including Transition to Turbulence Effects

Gustavo L. O. Halilaa , Joaquim R. R. A. Martinsa , Krzysztof J. Fidkowskia

This study introduces an aerodynamic shape optimization (ASO) framework that incorporates transition to turbulence effects, addressing a limitation of traditional 12 Reynolds-Averaged Navier-Stokes (RANS) models. By utilizing the AFT-S model in computational fluid dynamics (CFD) simulations, the framework extends optimization capabilities to include laminar and transitional flow regimes, crucial for achieving drag reduction in both subsonic and transonic conditions. Optimization in the subsonic regime focuses on extending laminar flow regions to minimize skin friction drag, while in the transonic regime, it balances increasing laminar flow extent with weakening shock waves. The proposed ASO framework yields more efficient, low-drag airfoil designs by leveraging extended laminar flow regions. Moreover, it offers high-fidelity optimization without the need for additional external modules, ensuring compatibility with industry standards.

2. A reinforcement learning approach to airfoil shape optimization

Thomas P. Dussauge, Woong Je Sung, Olivia J. Pinon Fischer & Dimitri N. Mavris

This research presents a Deep Reinforcement Learning (DRL) approach to airfoil shape optimization. Motivated by recent successes in data-driven approaches and the exploratory nature of RL, airfoil design is formulated as a Markov decision process. A custom RL environment integrates a low-fidelity aerodynamic solver, enabling the DRL agent to learn optimal modifications to increase aerodynamic performance. Results demonstrate the DRL agent's ability to learn optimal policies within limited iterations and produce high-performing airfoil shapes. Contributions include demonstrating RL's applicability to physics-based problems and providing enhanced action freedom for modifying airfoil parameters. Future efforts aim to expand the agent's action space, improve aerodynamic solver robustness, and explore a broader design space.

3. Airfoil optimization using a machine learning-based optimization algorithm

Xueyi Song, Lin Wang, Xianwu Luo

This study introduces a machine learning-based optimization algorithm and genetic algorithm for the airfoil optimization problem, focusing on maximizing the lift-to-drag ratio while maintaining the

lift coefficient. The comparison of aerodynamic performance between numerical results and experimental data demonstrates good agreement. Results indicate that the proposed method achieves optimized airfoil shapes with high aerodynamic performance, outperforming traditional genetic algorithm methods. Additionally, the machine learning-based approach significantly reduces simulation time. These findings suggest promising potential for machine learning-based optimization in fluid machinery design applications in the future.

4. Shape Optimization of Airfoils by Machine Learning-Based Surrogate Models

Marco Zanichelli

Airfoil shape optimization often requires numerous calls to computationally expensive numerical Computational Fluid Dynamics (CFD) solvers. To mitigate these computational costs, surrogate models are proposed as an attractive alternative. This study demonstrates that surrogate models if accurately approximating the underlying 13 Partial Differential Equation (PDE) and are computationally efficient, can be effectively integrated into standard optimization algorithms. Among the tested procedures, the Standard Surrogate Model Optimization offers cost-effective optimization with various goals, yielding significant improvements in aerodynamic efficiency and drag reduction. The Iterative Surrogate Model Optimization further reduces the computational time for generating training data while achieving notable reductions in drag coefficient. Contrastingly, Direct Optimization using the CFD solver proves to be computationally intensive and less effective within the same number of iterations. These findings underscore the potential of surrogate models in streamlining airfoil shape optimization processes, offering computational advantages without compromising optimization outcomes.

Chapter 3

Methodology

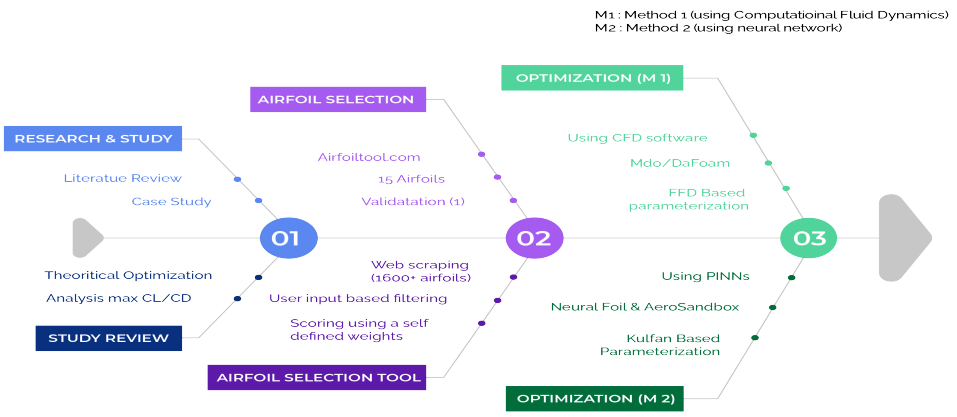

Figure 3.1: Overview of our working methodology

Our case study compares the efficiency of two methods for airfoil shape optimization: conventional Computational Fluid Dynamics (CFD) and contemporary machine learning techniques. There are multiple crucial steps in the methodology:

3.1 Overview :

In this case study, we concentrate on improving an airfoil's shape using two different methods: a contemporary machine learning technique and the conventional Computational Fluid Dynamics (CFD) method. The conventional approach makes use of computational fluid dynamics (CFD), a proven technique for examining pressure distributions and flow patterns surrounding aerodynamic components. CFD may efficiently direct the optimization process by using goal functions, such as optimizing lift-to-drag ratios or lowering drag coefficients. The discrete adjoint approach is a potent methodology in this field that may precisely optimize airfoil forms to suit specific performance objectives, such as maximizing lift or minimizing drag, when combined with gradient-based optimization algorithms. The modern approach, on the other hand, makes use of machine learning techniques, which are gaining traction in the field of design and analysis. Machine learning can forecast ideal forms and performance characteristics by training models on large datasets; this could lead to speedier and more flexible solutions than traditional methods. We shall compare the efficiency of these two methods—the contemporary machine learning methodology and the conventional CFD method—in terms of optimizing the forms of airfoils. Our objective is to ascertain whether the approach performs better in terms of computational effectiveness, optimization accuracy, and usefulness for aerodynamic design.

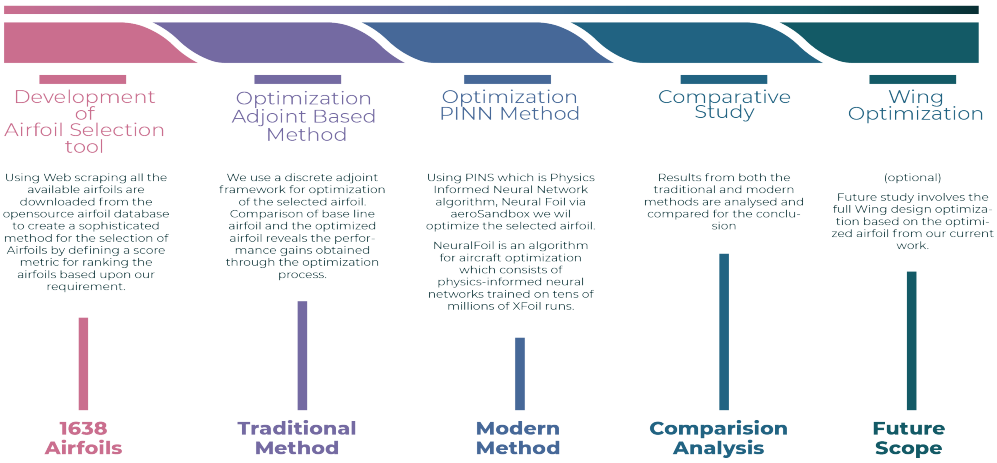

Figure 3.2: Overall approach of our project

3.1.1 Constraints :

Considering we are using the data points for the design of wing planform of a generic Unmanned Aerial Vehicle, the range of cruise speed generally doesn't exceed 16 m/s taking in account that the overall air traffic management will have to be more robust in order to handle higher speeds at a flying altitude of 5000m. In our case we had a target for cruise speed of 12 m/s for initial sizing and constraint analysis for evaluation of target power and torque requirements. With a rectangular wing span in consideration for initial optimization, the chord length for each cross-section remains the same.

3.1.2 Data Collection and Preparation :

For this project, we gathered airfoil data from open-source websites like Airfoil Tools (airfoiltools.com), which makes airfoil datasets for research accessible to the public. We used web scraping techniques to extract all 1,638 airfoil datasets that were available on the website. The geometric and aerodynamic properties of each airfoil are described in depth in each dataset, which is saved in.dat files.

We then utilized these.dat files to create our unique airfoil choosing tool. Using particular aerodynamic factors and constraints suited to our optimization objectives, we may use this tool to filter and rank the airfoils. Ten airfoils that most closely met our requirements were shortlisted during this process. Finally, we used our selection tool to choose the single airfoil that had the greatest score in order to proceed further optimization and analysis. This method ensured a rigorous and objective approach to choosing the most suitable airfoil for our specific needs.

Web Scraping Pseudocode Logic

```
Imports:
- Import the necessary libraries: requests, BeautifulSoup, pandas, os.

Pseudocode:
1. Import necessary libraries: requests, BeautifulSoup, pandas, os.
2. Set the URL of the webpage to scrape.
3. Set the header to mimic a web browser.
4. Send a GET request to the URL with headers and parse the content
with BeautifulSoup.
5. Find all <td> tags with class="link" to extract airfoil links.
6. Iterate through each <td> tag and extract the href attribute to
get airfoil links.
7. Print the extracted airfoil links.

8. Define the base URL and create complete airfoil links.
9. Replace a part of the URL to get the links for .dat files.
10. Iterate through modified links to download .dat files:
    10.1. Set the folder path where .dat files will be saved.
    10.2. Iterate through modified links.
        10.3. Extract the filename from the modified link.
        10.4. Create the file path by joining the folder path and
        filename.
        10.5. Check if the file already exists, if yes, skip.
        10.6. If the file doesn't exist:
            10.7. Download the content from the modified link.
            10.8. Write the content to a .dat file.
            10.9. Print a message confirming the file download.
```

3.1.3 Traditional Method- CFD Optimization :

The provided script (see Appendix) in DAFoam sets up and runs an aerodynamic optimization for the FX63137SM-IL airfoil at low speeds using OpenMDAO and DAFoam. The main goal is to optimize the airfoil shape to minimize drag (CD) while meeting a target lift coefficient (CL). The process involves setting up the computational environment and specifying the optimization parameters.

The algorithm starts by parsing command-line arguments to choose the optimizer and task type. It defines initial conditions like freestream velocity (U0), pressure (p0), turbulence (nuTilda0), and the target lift coefficient ($C_{L,target}$). DAFoam options are configured for the solver, boundary conditions, objective functions, and adjoint equations.

The 'Top' class, inheriting from 'Multipoint', is defined to set up the optimization problem. This involves initializing the DAFoam solver, mesh, and geometry components. An aerodynamic scenario for the optimization is added, and connections between mesh coordinates and geometry are established. The angle of attack (AOA) is defined as a design variable and linked to the solver.

Constraints are imposed on shape deformation using Free-Form Deformation (FFD) points, ensuring symmetry, thickness, and volume constraints are maintained. The optimization problem is solved using a selected optimizer (IPOPT, SLSQP, or SNOPT), with settings tailored to each optimizer.

Finally, depending on the task, the script either runs the optimization, executes a primal run, computes adjoint solutions, or checks total derivatives against finite differences. Results, including

Algorithm 2 DAFoam Optimization Algorithm

Initialize Input Parameters:
    $U0 \leftarrow 12.0$
    $p0 \leftarrow 0.0$
    $\nu Tilda0 \leftarrow 4.5 \times 10^{-5}$
    $CL_{target} \leftarrow 1$
    $aoa0 \leftarrow 1.5$
    $A0 \leftarrow 0.1$
    $\rho 0 \leftarrow 1.0$
Initialize DAFoam Options and Mesh Deformation Setup
Parse Command Line Arguments:
    $optimizer \leftarrow GetArgument(" - optimizer", "IPOPT")$
    $task \leftarrow GetArgument(" - task", "opt")$
Create `Top` class to set up the optimization problem
    Class Top(Multipoint):
        Method setup(self):
        Initialize DAFoam Builder with Options and Mesh
        Add Subsystems: dvs, mesh, geometry
        Add Aerodynamic Scenario
        Connect Mesh and Geometry Components
        Method configure(self):
        Add Objective Function to Scenario
        Get Surface Coordinates from Mesh
        Add Pointset to Geometry Component
        Set Triangular Points for Geometric Constraints
        Define Function to Change Angle of Attack
        Pass AOA Function to Scenario
        Select FFD Points to Move
        Setup Symmetry Constraints
        Setup Volume and Thickness Constraints
Initialize OpenMDAO Problem with Top Model
Generate N2 Diagram
Initialize Optimization Function with Options and Problem
Setup Optimizer:
if $optimizer ==$ "$SNOPT$" **then**
    Configure SNOPT Settings
else if $optimizer ==$ "$IPOPT$" **then**
    Configure IPOPT Settings
else if $optimizer ==$ "$SLSQP$" **then**
    Configure SLSQP Settings
else
    Print "Optimizer argument not valid!" and exit
end if
Set Debug Print Options for Optimizer
Set Optimization History File

optimal design variables, are saved to a JSON file.

3.1.4 Modern Method - Machine Learning Optimization

- **Airfoil Selection:** Choosing the right airfoil is essential for maximizing the stability, effectiveness, and performance of different aerodynamic systems. A variety of application cases, including racing automobiles, hydrofoils, wind turbines, aircraft wings, and gliders, all have unique performance requirements that influence the choice of airfoil. For example, wind turbines require airfoils that function well over a variety of wind speeds, and commercial airplanes need airfoils with high lift-to-drag ratios and good stall characteristics. Airfoils designed for high-speed airplanes are advantageous as they reduce drag and regulate shock waves, while airfoils for racing vehicles must produce significant downforce while minimizing drag. A customized approach to airfoil selection is required for each application to optimize the system's overall performance and guarantee that the desired aerodynamic and structural requirements are met. We created a novel tool that allows manual modifications of aerodynamic characteristics to filter airfoils based on specific needs, hence simplifying the airfoil selection process. This tool makes it easier to determine which airfoil is optimal for our optimization needs. The need to develop a tool for airfoil selection emerged from the awareness that existing approaches are laborious and require a thorough grasp of parameter analysis at different stall angles in addition to a great deal of theoretical knowledge. Our application simplifies the choosing process by offering an interactive and straightforward method, making it effective and accessible for users with different levels of experience.

- **Model Selection:** Choosing the right algorithms to forecast airfoil performance based on shape characteristics is known as model selection, and it is an essential stage in machine learning. We employ NeuralFoil, a neural network framework, for this investigation. Because neural networks can simulate intricate, non-linear interactions between input data (airfoil forms) and output targets (aerodynamic performance measurements), they are favored in this domain. With the help of the fundamental physics of fluid dynamics, NeuralFoil, a physics-based machine learning tool, improves prediction accuracy.

- **Training:** A collection of airfoil forms and the performance characteristics that accompany them, derived from CFD simulations, are fed into the machine learning model during training. The model iteratively modifies its internal parameters to reduce errors and determine the correlations between shape parameters and aerodynamic performance. The training procedure identifies the key patterns and trends in the training dataset, ensuring that the model can generalize effectively to new data. We create 450 rows of datasets for training our model within the alpha range of -20 to 25.

- **Optimization:** To determine the ideal airfoil shape, the optimization framework combines optimization methods with a trained machine-learning model. Optimization's objective function is defined according to certain performance requirements, including lift-to-drag ratio maximization or drag minimization. Various optimization algorithms, including Bayesian optimization and evolutionary algorithms, are utilized to effectively explore the design space. Until the ideal design is found, these algorithms repeatedly modify the airfoil shape parameters by predicting performance using the machine learning model.

Airfoil Optimization Algorithm in NeuralFoil

The algorithmic procedure used in the NeuraFoil Code can be divided into two parts. The algorithm optimizes the shape of an airfoil using Kulfan parameterization and aerodynamic constraints to achieve specific lift coefficients (CL) while minimizing drag. Initially, the airfoil coordinates are loaded from a file and converted into Kulfan parameters, which define the airfoil shape using a set of control points. An optimization problem is then formulated, with design variables representing the Kulfan parameters and angle of attack (alpha).

Aerodynamic targets for multiple points, including lift coefficients and corresponding Reynolds numbers, are defined. The optimization constraints include ensuring a high confidence level in aerodynamic analysis, achieving target CL values at specified alphas, maintaining positive increments in alphas, and adhering to constraints on moment coefficient, local thickness at specific chord locations, and trailing edge angle. Bounds are set for Kulfan parameters to ensure realistic shapes, and a wiggliness penalty is applied to avoid overly complex geometries.

The objective function minimizes the weighted drag coefficient (CD) across the specified conditions. A callback function captures the optimization progress, storing the evolving airfoil shapes and aerodynamic performance metrics. The optimization problem is solved using a nonlinear solver, and the resulting optimized airfoil shape and aerodynamic properties are saved for further analysis. This process ensures the airfoil meets the desired aerodynamic performance while maintaining practical geometric constraints.

Algorithm 3 Airfoil Shape Optimization using Kulfan Parameters and Aerodynamic Constraints in NeuralFoil

Input: Initial airfoil coordinates from file
Output: Optimized airfoil shape and aerodynamic performance

function OptimizeAirfoil
    Load initial airfoil coordinates from file
    Convert initial airfoil to Kulfan parameter representation
    Initialize optimization problem
    Define aerodynamic targets and weights for multiple points
    Calculate Reynolds number for each target lift coefficient
    Define design variables for Kulfan parameters and angle of attack
    Define aerodynamic analysis function using neural network model
    Set optimization constraints:
        - Confidence level of aerodynamic analysis > 0.90
        - Target lift coefficients CL at specified angles of attack
        - Positive increments in angles of attack
        - Moment coefficient $CM \geq -0.133$
        - Minimum local thickness at specified chord locations
        - Minimum trailing edge angle
        - Lower and upper bounds for Kulfan parameters
        - Minimize wiggliness of airfoil shape
    Define objective function to minimize weighted drag coefficient
    Define callback function to store optimization progress
    Solve optimization problem
    Store optimized airfoil shape and aerodynamic performance
end function

Algorithm 4 Optimization Algorithm in Neural Foil

```
procedure MAIN
    Load initial airfoil coordinates from file
    Convert initial airfoil to Kulfan parameter representation
    Initialize optimization problem
    Define aerodynamic targets and weights for multiple points
    Calculate Reynolds number for each target lift coefficient
    Define design variables for Kulfan parameters and angle of attack
    Define aerodynamic analysis function using neural network model
    Set optimization constraints
    Define objective function to minimize weighted drag coefficient
    Define callback function to store optimization progress
    Solve optimization problem
    Store optimized airfoil shape and aerodynamic performance
end procedure
```

3.2 Computational Framework Setup

The Computational Framework Setup for aerodynamic shape optimization of airfoils involved selecting a high-fidelity Computational Fluid Dynamics (CFD) solver capable of handling complex geometries and turbulence modeling. Airfoil geometries from the dataset were discretized into a computational mesh with careful attention to mesh quality and resolution. Realistic boundary conditions were defined, including freestream velocity and turbulence intensity. Parallel computing resources were utilized to expedite simulations, and post-processing tools were employed for analysis. Overall, the framework provided a scalable platform for conducting aerodynamic simulations and shape optimization, facilitating efficient exploration and refinement of airfoil designs to achieve desired performance objectives.

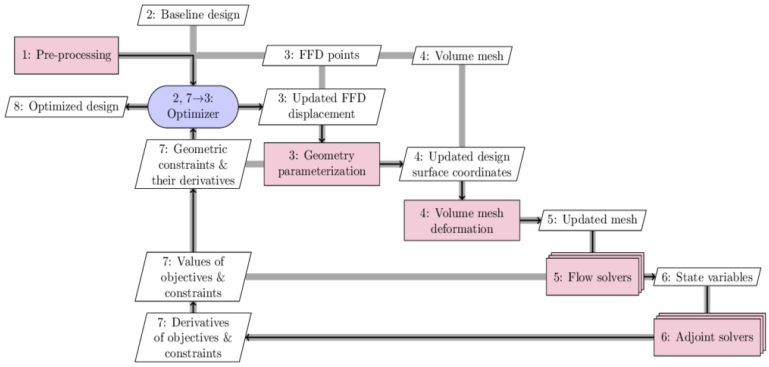

Figure 3.3: Optimization Framework in DAFoam

3.3 Airfoil Selection Tool

The Airfoil Selection Tool developed by us is a comprehensive and user-friendly application designed to streamline the process of selecting optimal airfoils for various aerodynamic applications. It begins by collecting and preparing airfoil data from open-source websites like Airfoil Tools (airfoiltools.com), utilizing web scraping techniques to extract detailed geometries from .dat files. With access to 1,640 airfoil datasets, the tool allows users to upload their own .dat files or use pre-existing data from a built-in database. Users can specify aerodynamic conditions such as alpha range, Reynolds number, and Mach number. The tool calculates critical aerodynamic parameters (C_L, C_D, C_M) over a specified

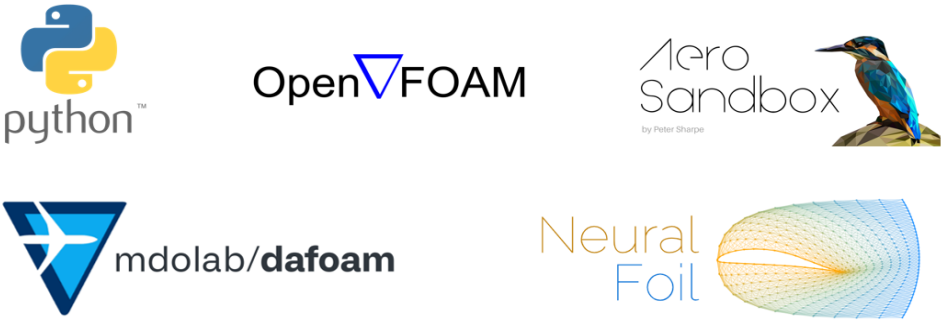

Figure 3.4: Tools and software used in our project

range of angles of attack (α) and provides options to include additional data columns such as C_{pmin}, Top_Xtr, Bot_Xtr, and $mach\_crit$. Advanced filtering options enable users to apply multiple rules, such as maximizing C_L, minimizing C_D, setting α to 0, maximizing the C_L/C_D ratio, and selecting high α values. These filtering capabilities efficiently generate a combined dataset of top-performing airfoils.

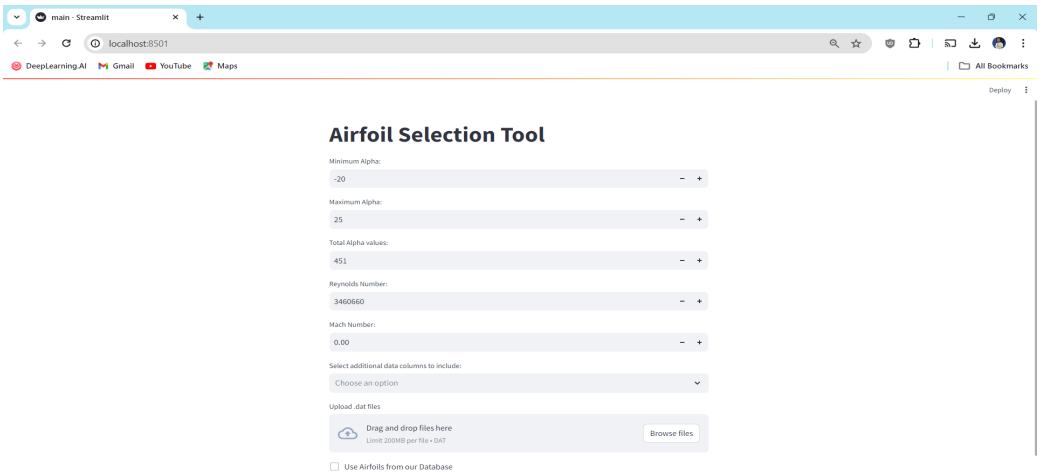

Figure 3.5: User-Interface of the Selection Tool

Furthermore, the tool features an intuitive scoring system that allows users to assign weights to different aerodynamic parameters (C_L, C_D, C_M, α, C_L/C_D), calculate comprehensive scores for each airfoil, and rank them accordingly. Detailed visualizations of filtered and scored airfoils facilitate easy comparison, and the data can be exported for further analysis or direct application in aerodynamic designs. This tool significantly reduces the time required for airfoil selection, making it accessible even to those with limited theoretical knowledge, and supports informed decision-making for researchers, engineers, and designers across a wide range of aerodynamic applications.

3.3.1 Benefits of Using the Airfoil Selection Tool

- **Efficiency:** Automates data collection, processing, filtering, and scoring, reducing time and effort needed for airfoil selection.
- **User-Friendly Interface:** Easy-to-use interface suitable for users with varying levels of theoretical knowledge in aerodynamics.

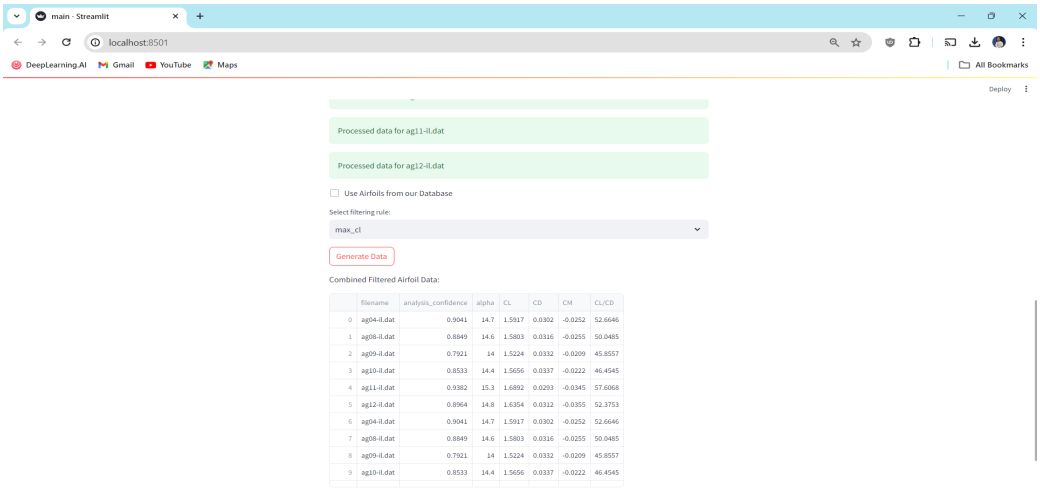

Figure 3.6: Results obtained using our Selection Tool

- **Comprehensive Data Analysis:** Provides detailed analysis of aerodynamic parameters, facilitating informed decision-making.
- **Advanced Filtering Options:** Offers multiple filtering rules to identify top-performing airfoils based on specific criteria.
- **Intuitive Scoring System:** Allows for customizable weighting of aerodynamic parameters to rank airfoils according to user-defined priorities.
- **Versatility:** Suitable for a wide range of aerodynamic applications, from research to practical design optimization.
- **Access to Extensive Data:** Utilizes a database of 1,640 airfoil datasets, providing a broad selection for various needs and allowing users to upload their own datasets.
- **Data Export:** Facilitates further analysis or direct application in designs by allowing the export of filtered and scored results.
- **Integration with Existing Workflows:** Seamlessly integrates with existing aerodynamic design workflows, enhancing productivity and accuracy.

Airfoil Selection Tool Pseudocode Logic

```
Imports:
- Import necessary libraries: streamlit, os, pandas, numpy,
aerosandbox, tempfile, tqdm, MinMaxScaler.

Function Definitions:
- process_dat_file(): Process individual DAT files to extract
aerodynamic data.
- load_pre_existing_data(): Load pre-existing data from a directory.
- process_airfoil_data(): Filter and process airfoil data based on
specified criteria.
```

```
- airfoil_score(): Calculate scores for airfoils based on given weights.
- normalize_aerodynamic_data(): Normalize aerodynamic data.

Main Function (main()):
1. Set up the Streamlit sidebar menu with options: "Airfoil Tool" and
"Project Details".
2. If "Airfoil Tool" mode is selected:
    2.1. Display input fields for aerodynamic parameters: minimum alpha,
    maximum alpha, Reynolds number, and Mach number.
    2.2. Allow the user to select additional data columns to include.
    2.3. Allow users to upload .dat files and process them.
    2.4. Optionally, allow users to use pre-existing airfoil data.
    2.5. Filter and process airfoil data based on selected rules and
    thresholds.
    2.6. Calculate scores for airfoils based on user-defined weights.
3. If "Project Details" mode is selected:
    3.1. Display project overview and features.
    3.2. Show project team details.

Function Calls:
- main(): Run the main function to start the Streamlit app.
```

Chapter 4

Results and Discussions

In the process of deploying 2-Dimensional analysis for the workflow, it's necessary to validate the XFoil results against the experimental results so as to conform with the standard wind tunnel data. For this purpose, we used the experimental data from NASA Langley Research center for NACA 0012 Turbulence Modeling Resource.

The flow conditions as per the data provided are Ma = 0.15 and Re = 6 Million. The results obtained using inviscid and viscous methods in XFLR5 (using XFoil framework) exhibited satisfactory results with the viscous results being closer to the experimental results. For zero degree angle of attack, we can see that near the trailing edge, the numerical results are somewhat not as close to as what appear from 20 % chord length. However, it's within the acceptable range and the discrepancy is reduced moving with progressing dimensionless chord length.

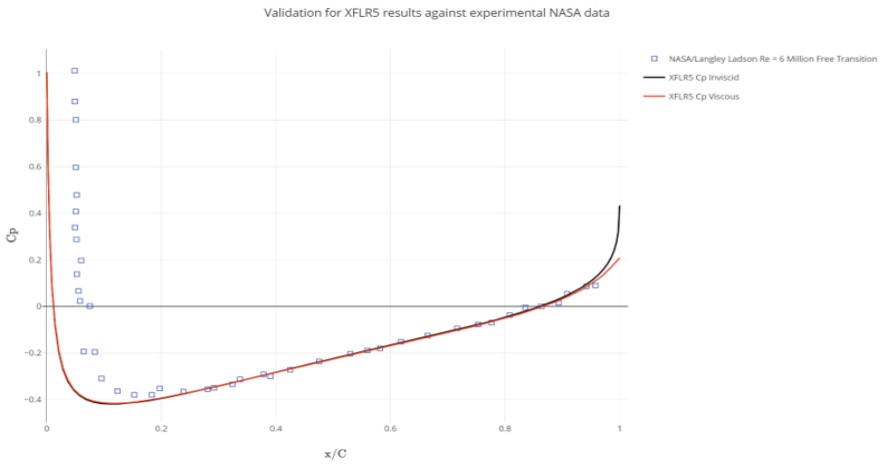

Figure 4.1: Validation against Experimental Results at 0^o A.O.A.

In a different angle of attack (20^o) that we compared with the experimental results, it can seen that the results obtained from XFoil completely overlaps the experimental results and since it being a near stall region, we may come to the conclusion that the 2-Dimensional analysis tool is capable enough to handle the aerodynamic evaluating results in the stall regions too. Hence for initial 2-D bunch analysis, we can effectively use it as a tool.

We then collected a set of 15 airfoils on the basis of design parameters like nimble stability characteristics, popular usage and ease of manufacturing. Evaluation of aerodynamics properties at the stated Reynolds Number of Re =800,000 and Ma = 0, we compare the results obtained across several design parameters, two of the most important being benign stall characteristics and maximum lift

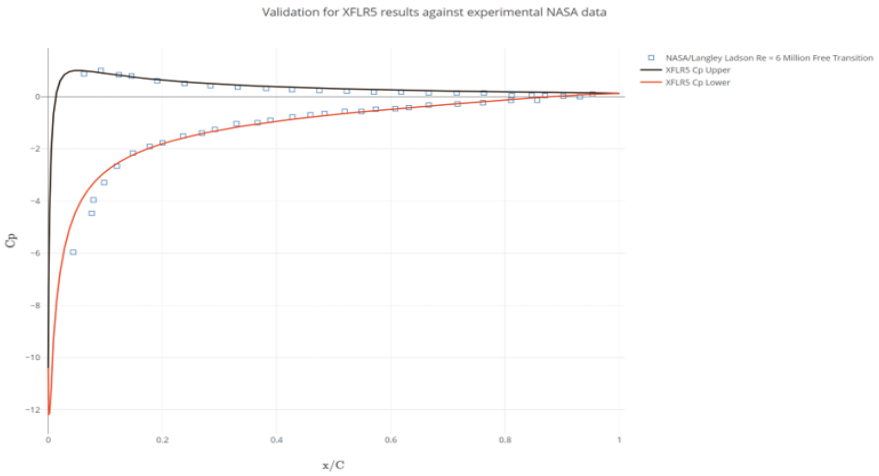

Figure 4.2: Validation against Experimental Results at 20^o A.O.A.

coefficient C_l at both 0^o and over the operating range of angle of attacks.

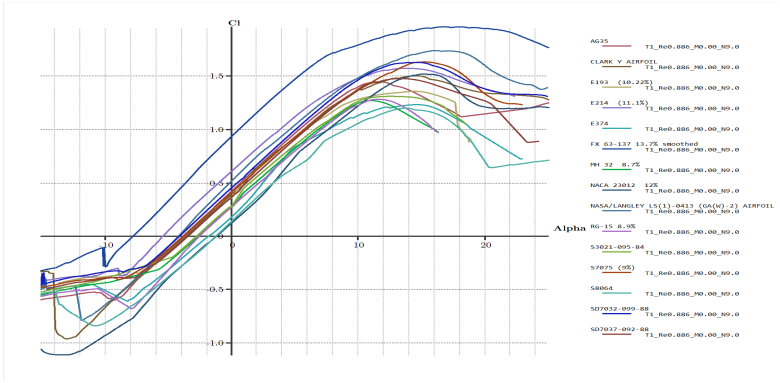

Figure 4.3: C_l vs α graph for the 15 selected airfoils

A closer look at the drag bucket graph helps understand the probability of section lift coefficients ($C_{l,climb}$ and $C_{l,cruise}$) generated by the lifting surfaces will lie inside the bucket itself during lift and climb, regardless of the combination of weight and CG location.

The left graph, Figure 4.4, is essentially drag analysis gives the insight to drag coefficient values with increasing value of angle of attack. It is evident that due to stalling at both high negative and positive values of positive and negative angle of attack, the drag coefficient rises. In the vicinity of zero angle of attack, however, we can observe a flat horizontal curve in almost all airfoils.

The graph on the right compares how the value of coefficient of lift, C_d against C_l behaves and the drag bucket is the section in which the values of C_d remains almost constant for a specific range of values of C_l. Both graphs when inferred to gives an insight as to comparison of drag performance of the airfoils.

Referring to the next two figures, we can see two important parameters which directly effect the range and endurance of the aircraft. The C_l/C_d ratio or also commonly known as L/D ratio influences the range of the aircraft that is the maximum distance it can travel while being in the air. For majority

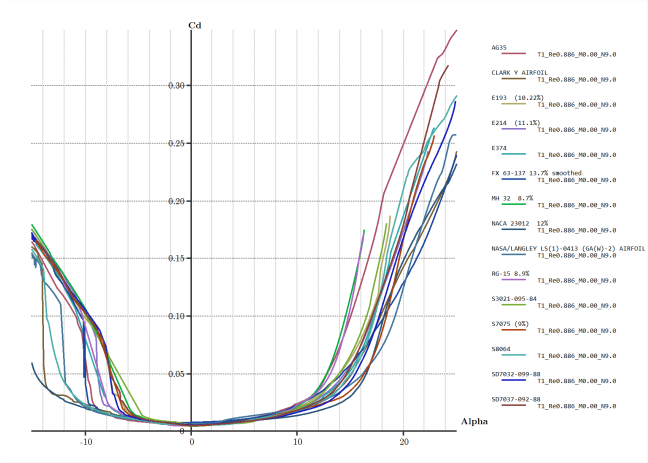

Figure 4.4: C_d vs α

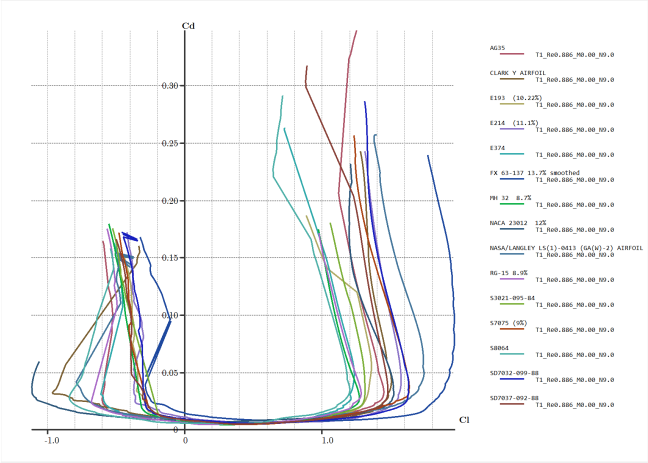

Figure 4.5: C_d vs C_l

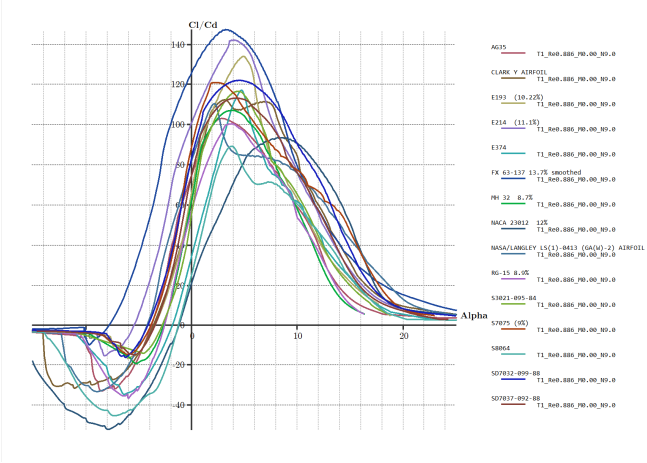

Figure 4.6: C_l/C_d vs α

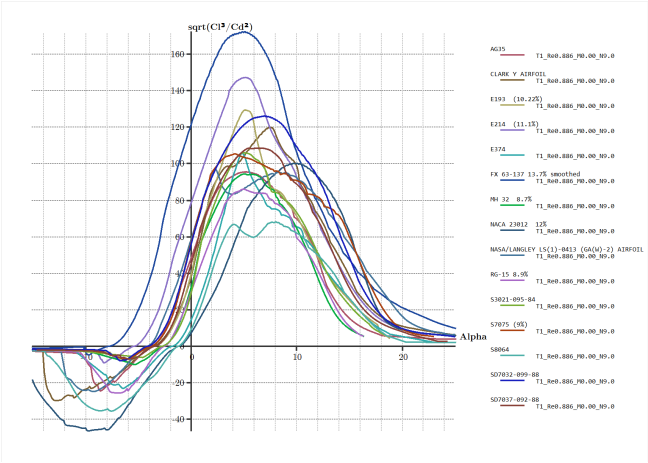

Figure 4.7: $(C_l{}^3/C_d{}^2)^{0.5}$ vs α

of the scenarios, the airfoils are fitted at zero angle of attack, so for cruise condition design, it is useful to look out for the angle of 0^o for manufacturing requirements considerations. Similarly the graph on the right, Fig 4.7, the metric $(C_l{}^3/C_d{}^2)^{0.5}$ is plotted against increasing angles of attack, α. Again, considering the ease of manufacturability, we look out for the value at zero angle of attack is considered for comparison of the value of metric so as the best performance is obtained at cruise conditions.

On the basis of the following selection criterion parameters for the overall aerodynamic performance considerations,

– Benign stall characteristics
– Maximum Lift ($C_{l,o}$) at 0_o angle of attack
– Maximum Lift Coefficient available ($C_{l,max}$)
– Widest Drag Bucket
– Maximum Deliverable Range and Endurance

we came to the conclusion that out of the 15 airfoils in our data set, **FX 63-137 13.7 % smoothed (fx63137sm-il)** has the best overall characteristics and hence was selected for further optimization processes to be carried out.

4.1 Comparison of Optimized Results

The selected airfoil's geometry was used as input in .dat format for geometry for both the optimization engines and the obtained results were pretty different as expected.

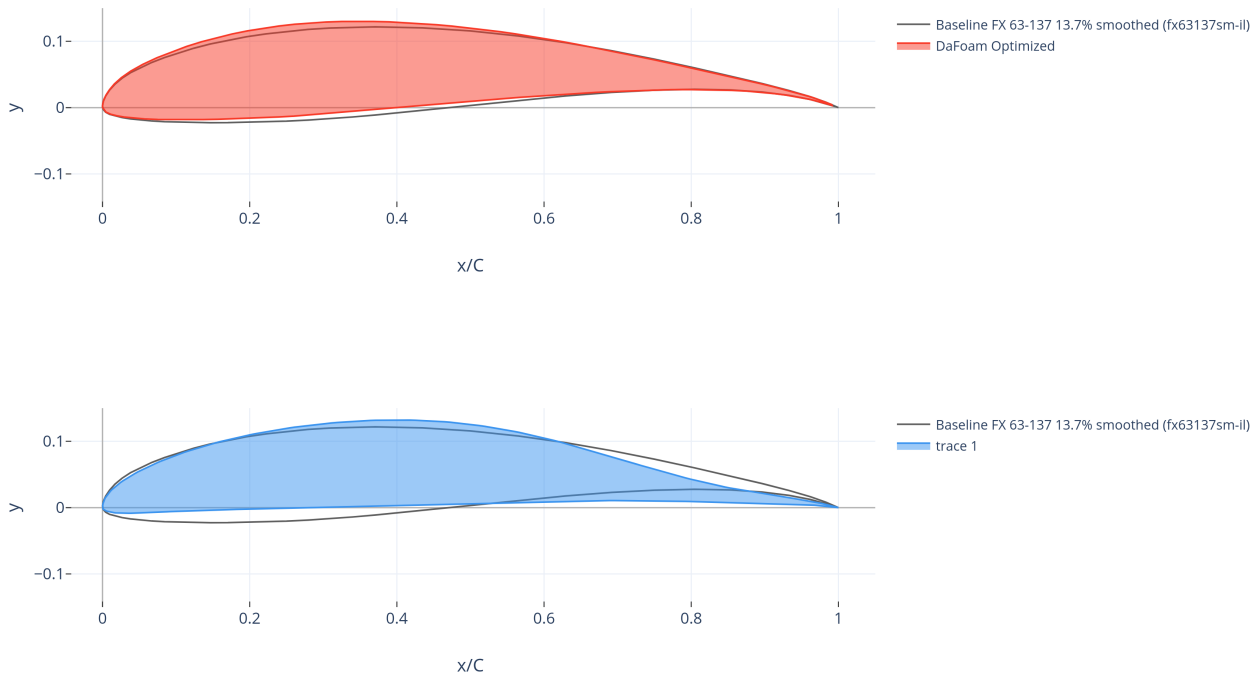

Figure 4.8: Comparison of Optimized Results

4.2 Optimization History

In the optimization history graph, the objective function is plotted against the number of iterations to illustrate the progression of optimization over time. Initially, the objective function starts at a relatively

high value, indicating poor aerodynamic performance. As the optimization algorithm begins exploring the design space, there is a rapid decrease in the objective function during the initial iterations, signifying significant improvements in aerodynamic performance.

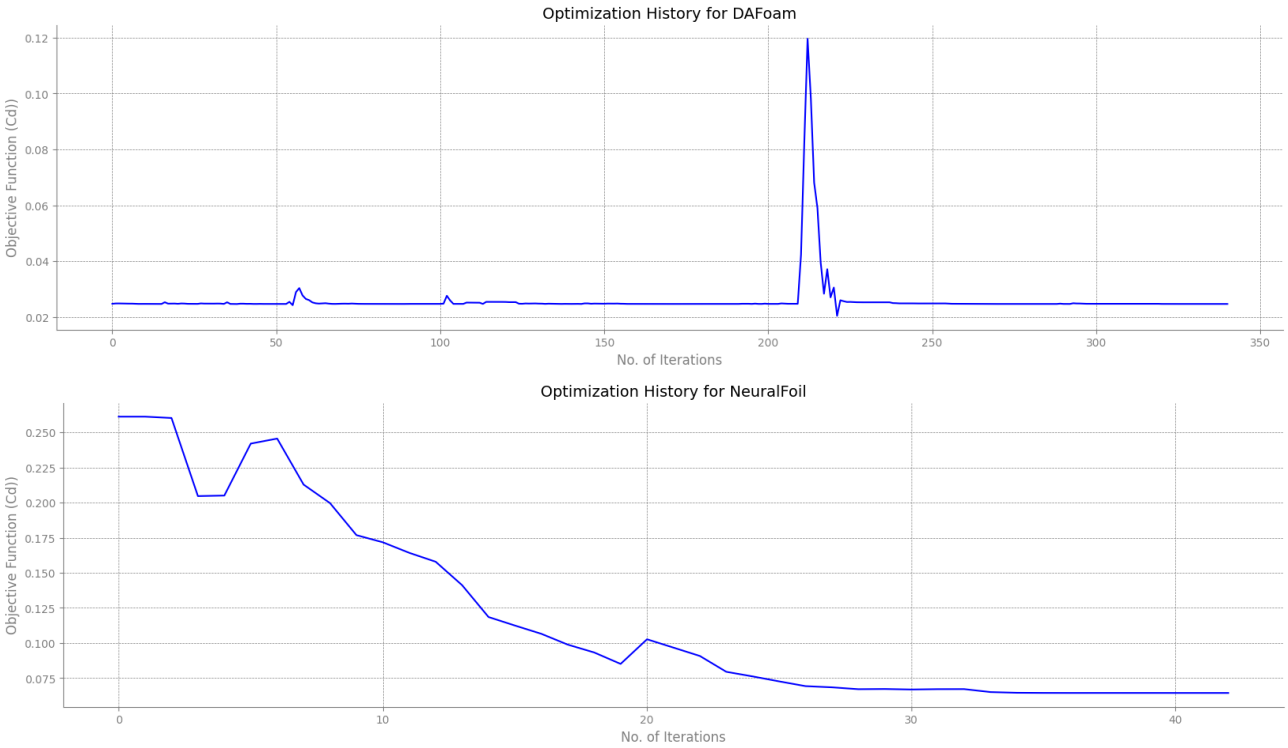

Figure 4.9: Comparison of Optimization History

This steep decline gradually slows down as the optimization progresses, with smaller adjustments made to refine the design. In the mid-iterations, the rate of improvement becomes more gradual, and by the later iterations, the objective function values stabilize as the algorithm converges towards an optimal or near-optimal solution. After a certain point, further iterations show minimal improvement, indicating convergence. The graph provides insights into the efficiency of the optimization process and the convergence behavior of the algorithm, helping engineers understand how the design evolves and improves over the course of optimization.

Chapter 5

Conclusions

We conclude that using DAFoam as optimization engine provides more robust design optimization and smoother control over both volume and thickness constraints. Using the leading and trailing edge points as control points for constraints of volumetric and thickness dimesnions provide more flexibility and control for overall optimization

- The total number of iterations incurred in NeuralFoil is significantly lesser than that of DAFoam.
- Since DAFoam uses CFD to evaluate conjugate gradient at each step of gradient descent, it consumes much more resource and time than NeuralFoil, which directly uses its Machine Learning Based Algorithm developed from XFOIL data to integrate Surrogate based optimization.
- The overall percentage improvement in aerodynamic efficiency for NeuralFoil blows away the comparison to DAFoam engine

Parameter	DAFoam	NeuralFoil
Iterations	340	42
Time(mins)	240	1.5
Efficacy	0.2	75

Table 5.1: Metric Comparison for both Optimization Engines

It might seem that NeuralFoil beats DAFoam in both time and efficacy matter, however there seems to be discrepancy in the initial value of objective function evaluated through both engines, therefore a future scope remains to completely validate the results against experimental data obtained from either wind tunnel testings or computational results obtained from robust CFD workflow.

Also, the concept of optimization is to be carried out to the shape morphing of the whole wing structure and include aero-structural parameters too as a part of future scope.

Appendix A

Python Script used to run the DAFoam Optimization

A Python script was used to run the optimization module using DAFoam with initial and boundary conditions, constraints, and objective functions:

```
#!/usr/bin/env python
"""
DAFoam run script for the fx63137sm-il airfoil at low-speed
"""

# =============================================================
# Imports
# =============================================================
import os
import argparse
import numpy as np
import json
from mpi4py import MPI
import openmdao.api as om
from mphys.multipoint import Multipoint
from dafoam.mphys import DAFoamBuilder, OptFuncs
from mphys.scenario_aerodynamic import ScenarioAerodynamic
from pygeo.mphys import OM_DVGEOCOMP
from pygeo import geo_utils


parser = argparse.ArgumentParser()
# which optimizer to use. Options are: IPOPT (default), SLSQP, and SNOPT
parser.add_argument("-optimizer", help="optimizer to use", type=str, default="IPOPT")
# which task to run. Options are: opt (default), runPrimal, runAdjoint, checkTotals
parser.add_argument("-task", help="type of run to do", type=str, default="opt")
args = parser.parse_args()

# =============================================================
# Input Parameters
# =============================================================
U0 = 12.0
p0 = 0.0
nuTilda0 = 4.5e-5
CL_target = 1
aoa0 = 1.5
A0 = 0.1
# rho is used for normalizing CD and CL
rho0 = 1.0

# Input parameters for DAFoam
```

```
daOptions = {
    "designSurfaces": ["wing"],
    "solverName": "DASimpleFoam",
    "primalMinResTol": 1.0e-4,
    "primalBC": {
        "U0": {"variable": "U", "patches": ["inout"], "value": [U0, 0.0, 0.0]},
        "p0": {"variable": "p", "patches": ["inout"], "value": [p0]},
        "nuTilda0": {"variable": "nuTilda", "patches": ["inout"], "value": [nuTilda0]},
        "useWallFunction": True,
    },
    "objFunc": {
        "CD": {
            "part1": {
                "type": "force",
                "source": "patchToFace",
                "patches": ["wing"],
                "directionMode": "parallelToFlow",
                "alphaName": "aoa",
                "scale": 1.0 / (0.5 * U0 * U0 * A0 * rho0),
                "addToAdjoint": True,
            }
        },
        "CL": {
            "part1": {
                "type": "force",
                "source": "patchToFace",
                "patches": ["wing"],
                "directionMode": "normalToFlow",
                "alphaName": "aoa",
                "scale": 1.0 / (0.5 * U0 * U0 * A0 * rho0),
                "addToAdjoint": True,
            }
        },
    },
    "adjEqnOption": {"gmresRelTol": 1.0e-6, "pcFillLevel": 1, "jacMatReOrdering": "rcm"},
    "normalizeStates": {
        "U": U0,
        "p": U0 * U0 / 2.0,
        "nuTilda": nuTilda0 * 10.0,
        "phi": 1.0,
    },
    "designVar": {
        "aoa": {"designVarType": "AOA", "patches": ["inout"], "flowAxis": "x", "normalAxis": "y"},
        "shape": {"designVarType": "FFD"},
    },
}

# Mesh deformation setup
meshOptions = {
    "gridFile": os.getcwd(),
    "fileType": "OpenFOAM",
    # point and normal for the symmetry plane
    "symmetryPlanes": [[[0.0, 0.0, 0.0], [0.0, 0.0, 1.0]], [[0.0, 0.0, 0.1], [0.0, 0.0, 1.0]]],
}

# Top class to setup the optimization problem
class Top(Multipoint):
    def setup(self):

```

```
        # create the builder to initialize the DASolvers
        dafoam_builder = DAFoamBuilder(daOptions, meshOptions, scenario="aerodynamic")
        dafoam_builder.initialize(self.comm)

        # add the design variable component to keep the top level design variables
        self.add_subsystem("dvs", om.IndepVarComp(), promotes=["*"])

        # add the mesh component
        self.add_subsystem("mesh", dafoam_builder.get_mesh_coordinate_subsystem())

        # add the geometry component (FFD)
        self.add_subsystem("geometry", OM_DVGEOCOMP(file="FFD/wingFFD.xyz", type="ffd"))

        # add a scenario (flow condition) for optimization, we pass the builder
        # to the scenario to actually run the flow and adjoint
        self.mphys_add_scenario("cruise", ScenarioAerodynamic(aero_builder=dafoam_builder))

        # need to manually connect the x_aero0 between the mesh and geometry components
        # here x_aero0 means the surface coordinates of structurally undeformed mesh
        self.connect("mesh.x_aero0", "geometry.x_aero_in")
        # need to manually connect the x_aero0 between the geometry component and the cruise
        # scenario group
        self.connect("geometry.x_aero0", "cruise.x_aero")

    def configure(self):
        # configure and setup perform a similar function, i.e., initialize the optimization.
        # But configure will be run after setup

        # add the objective function to the cruise scenario
        self.cruise.aero_post.mphys_add_funcs()

        # get the surface coordinates from the mesh component
        points = self.mesh.mphys_get_surface_mesh()

        # add pointset to the geometry component
        self.geometry.nom_add_discipline_coords("aero", points)

        # set the triangular points to the geometry component for geometric constraints
        tri_points = self.mesh.mphys_get_triangulated_surface()
        self.geometry.nom_setConstraintSurface(tri_points)

        # define an angle of attack function to change the U direction at the far field
        def aoa(val, DASolver):
            aoa = val[0] * np.pi / 180.0
            U = [float(U0 * np.cos(aoa)), float(U0 * np.sin(aoa)), 0]
            # we need to update the U value only
            DASolver.setOption("primalBC", {"U0": {"value": U}})
            DASolver.updateDAOption()

        # pass this aoa function to the cruise group
        self.cruise.coupling.solver.add_dv_func("aoa", aoa)
        self.cruise.aero_post.add_dv_func("aoa", aoa)

        # select the FFD points to move
        pts = self.geometry.DVGeo.getLocalIndex(0)
```

```
        indexList = pts[:, :, :].flatten()
        PS = geo_utils.PointSelect("list", indexList)
        nShapes = self.geometry.nom_addLocalDV(dvName="shape", pointSelect=PS)

        # setup the symmetry constraint to link the y displacement between k=0 and k=1
        nFFDs_x = pts.shape[0]
        nFFDs_y = pts.shape[1]
        indSetA = []
        indSetB = []
        for i in range(nFFDs_x):
            for j in range(nFFDs_y):
                indSetA.append(pts[i, j, 0])
                indSetB.append(pts[i, j, 1])
        self.geometry.nom_addLinearConstraintsShape("linearcon", indSetA, indSetB, factorA=1.0, factorB=-1.0)

        # setup the volume and thickness constraints
        leList = [[0.0005, 0.0015, 1e-4], [0.0005, 0.0015, 0.1 - 1e-4]]
        teList = [[0.99, 0.00425, 1e-4], [0.99 - 1e-4, 0.00425, 0.1 - 1e-4]]
        self.geometry.nom_addThicknessConstraints2D("thickcon", leList, teList, nSpan=2, nChord=10)
        self.geometry.nom_addVolumeConstraint("volcon", leList, teList, nSpan=2, nChord=10)
        # add the LE/TE constraints
        self.geometry.nom_add_LETEConstraint("lecon", volID=0, faceID="iLow", topID="k")
        self.geometry.nom_add_LETEConstraint("tecon", volID=0, faceID="iHigh", topID="k")

        # add the design variables to the dvs component's output
        self.dvs.add_output("shape", val=np.array([0] * nShapes))
        self.dvs.add_output("aoa", val=np.array([aoa0]))
        # manually connect the dvs output to the geometry and cruise
        self.connect("aoa", "cruise.aoa")
        self.connect("shape", "geometry.shape")

        # define the design variables to the top level
        self.add_design_var("shape", lower=-1.0, upper=1.0, scaler=1.0)
        self.add_design_var("aoa", lower=0.0, upper=10.0, scaler=1.0)

        # add objective and constraints to the top level
        self.add_objective("cruise.aero_post.CD", scaler=1.0)
        self.add_constraint("cruise.aero_post.CL", equals=CL_target, scaler=1.0)
        self.add_constraint("geometry.thickcon", lower=0.5, upper=3.0, scaler=1.0)
        self.add_constraint("geometry.volcon", lower=1.0, scaler=1.0)
        self.add_constraint("geometry.tecon", equals=0.0, scaler=1.0, linear=True)
        self.add_constraint("geometry.lecon", equals=0.0, scaler=1.0, linear=True)
        self.add_constraint("geometry.linearcon", equals=0.0, scaler=1.0, linear=True)


# OpenMDAO setup
prob = om.Problem()
prob.model = Top()
prob.setup(mode="rev")
om.n2(prob, show_browser=False, outfile="mphys.html")

# initialize the optimization function
optFuncs = OptFuncs(daOptions, prob)

# use pyoptsparse to setup optimization
prob.driver = om.pyOptSparseDriver()
```

```
prob.driver.options["optimizer"] = args.optimizer
# options for optimizers
if args.optimizer == "SNOPT":
    prob.driver.opt_settings = {
        "Major feasibility tolerance": 1.0e-5,
        "Major optimality tolerance": 1.0e-5,
        "Minor feasibility tolerance": 1.0e-5,
        "Verify level": -1,
        "Function precision": 1.0e-5,
        "Major iterations limit": 1000,
        "Nonderivative linesearch": None,
        "Print file": "opt_SNOPT_print.txt",
        "Summary file": "opt_SNOPT_summary.txt",
    }
elif args.optimizer == "IPOPT":
    prob.driver.opt_settings = {
        "tol": 1.0e-5,
        "constr_viol_tol": 1.0e-5,
        "max_iter": 1000,
        "print_level": 5,
        "output_file": "opt_IPOPT.txt",
        "mu_strategy": "adaptive",
        "limited_memory_max_history": 10,
        "nlp_scaling_method": "none",
        "alpha_for_y": "full",
        "recalc_y": "yes",
    }
elif args.optimizer == "SLSQP":
    prob.driver.opt_settings = {
        "ACC": 1.0e-5,
        "MAXIT": 1000,
        "IFILE": "opt_SLSQP.txt",
    }
else:
    print("optimizer arg not valid!")
    exit(1)

prob.driver.options["debug_print"] = ["nl_cons", "objs", "desvars"]
prob.driver.options["print_opt_prob"] = True
prob.driver.hist_file = "OptView.hst"

if args.task == "opt":
    # solve CL
    optFuncs.findFeasibleDesign(["cruise.aero_post.CL"], ["aoa"], targets=[CL_target
    ])
    # run the optimization
    prob.run_driver()
    # write the optimal design variable values to disk
    OptDesignVars = {var: val.tolist() for var, val in prob.model.geometry.DVGeo.
    getValues().items()}
    with open('OptDesignVars.json', 'w') as f:
        json.dump(OptDesignVars, f, indent=4)
elif args.task == "runPrimal":
    # just run the primal once
    prob.run_model()
elif args.task == "runAdjoint":
    # just run the primal and adjoint once
    prob.run_model()
    totals = prob.compute_totals()
    if MPI.COMM_WORLD.rank == 0:
        print(totals)
elif args.task == "checkTotals":
    # verify the total derivatives against the finite-difference
```

```
    prob.run_model()
    prob.check_totals(
        of=["cruise.aero_post.CD", "cruise.aero_post.CL"], wrt=["shape", "aoa"],
    compact_print=True, step=1e-3, form="central", step_calc="abs"
    )
else:
    print("task arg not found!")
    exit(1)
```

Listing A.1: DAFoam run script for the fx63137sm-il airfoil at low-speed

Appendix B

Python Script used in NeuralFoil via AeroSandbox

In the NeuralFoil Framework, all of the optimization is carried out using a single python script :

```
import aerosandbox as asb
import aerosandbox.numpy as np
import matplotlib.pyplot as plt
import aerosandbox.tools.pretty_plots as p
from pprint import pprint  # "pretty print"
import matplotlib
import numpy as np
from matplotlib.animation import ArtistAnimation
from matplotlib.colors import LinearSegmentedColormap


filename = (r"C:\Users\shrey\OneDrive\Desktop\Phinal year project\scrapping\airfoil
    dat files\new\fx63137sm-il.dat")
coordinate_airfoil = asb.Airfoil(filename)
coordinate_airfoil

fig, ax = plt.subplots(figsize=(6, 2))
coordinate_airfoil.draw()

#converting to kulfan airfoil
kulfan_airfoil = coordinate_airfoil.to_kulfan_airfoil()
kulfan_airfoil

fig, ax = plt.subplots(figsize=(6, 2))
kulfan_airfoil.draw()

#pprint(kulfan_airfoil.kulfan_parameters)

CL_multipoint_targets = np.array([0.8, 1.0, 1.2, 1.4, 1.5, 1.6])
CL_multipoint_weights = np.array([5, 6, 7, 8, 9, 10])


Re = 800000 * (CL_multipoint_targets / 1.25) ** -0.5
mach = 0.0
initial_guess_airfoil = asb.KulfanAirfoil(filename)
initial_guess_airfoil.name = "Initial Guess (fx63137sm-il)"
opti = asb.Opti()

optimized_airfoil = asb.KulfanAirfoil(
    name="Optimized",
    lower_weights=opti.variable(
        init_guess=initial_guess_airfoil.lower_weights,
```

```
        lower_bound=-0.5,
        upper_bound=0.25,
    ),
    upper_weights=opti.variable(
        init_guess=initial_guess_airfoil.upper_weights,
        lower_bound=-0.25,
        upper_bound=0.5,
    ),
    leading_edge_weight=opti.variable(
        init_guess=initial_guess_airfoil.leading_edge_weight,
        lower_bound=-1,
        upper_bound=1,
    ),
    TE_thickness=0,
)

alpha = opti.variable(
    init_guess=np.degrees(CL_multipoint_targets / (2 * np.pi)),
    lower_bound=-20,
    upper_bound=25
)

aero = optimized_airfoil.get_aero_from_neuralfoil(
    alpha = alpha,
    Re=Re,
    mach=mach,
)

opti.subject_to([
    aero["analysis_confidence"] > 0.90,
    aero["CL"] == CL_multipoint_targets,
    np.diff(alpha) > 0,
    aero["CM"] >= -0.133,
    optimized_airfoil.local_thickness(x_over_c=0.33) >= 0.128,
    optimized_airfoil.local_thickness(x_over_c=0.90) >= 0.014,
    optimized_airfoil.TE_angle() >= 6.03, # Modified from Drela's 6.25 to match DAE-11 case
    optimized_airfoil.lower_weights[0] < -0.05,
    optimized_airfoil.upper_weights[0] > 0.05,
    optimized_airfoil.local_thickness() > 0
])


get_wiggliness = lambda af: sum([
    np.sum(np.diff(np.diff(array)) ** 2)
    for array in [af.lower_weights, af.upper_weights]
])

opti.subject_to(
    get_wiggliness(optimized_airfoil) < 2 * get_wiggliness(initial_guess_airfoil)
)

opti.minimize(np.mean(aero["CD"] * CL_multipoint_weights))
airfoil_history = []
aero_history = []

def callback(i):
    airfoil_history.append(
        asb.KulfanAirfoil(
            name="in-progress",
            lower_weights=opti.debug.value(optimized_airfoil.lower_weights),
            upper_weights=opti.debug.value(optimized_airfoil.upper_weights),
```

```
            leading_edge_weight=opti.debug.value(optimized_airfoil.
    leading_edge_weight),
            TE_thickness=opti.debug.value(optimized_airfoil.TE_thickness),
        )
    )
    aero_history.append({
        k: opti.debug.value(v) for k, v in aero.items()
    })

sol = opti.solve(
    callback=callback,
    behavior_on_failure="return_last",
    options={
        "ipopt.mu_strategy": 'monotone',
        "ipopt.start_with_resto": 'yes'
    }
)

optimized_airfoil = sol(optimized_airfoil)
aero = sol(aero)

optimized_filename = "optimized_airfoil.dat"
with open(optimized_filename, 'w') as f:
    f.write("Optimized Airfoil\n")
    for x, y in zip(optimized_airfoil.x(), optimized_airfoil.y()):
        f.write(f"{x:.6f}  {y:.6f}\n")
print(f"Optimized airfoil coordinates saved to {optimized_filename}")
fig, ax = plt.subplots(figsize=(6, 2))
optimized_airfoil.draw()
print(f"L / D = {sol(np.mean(aero['CL']) / np.mean(aero['CD'])):.1f}")
print(f"CD = {sol(np.mean(aero['CD'])):.1f}")


fig, ax = plt.subplots(1, 2, figsize=(10, 4), dpi=300)
p.show_plot(show=False, rotate_axis_labels=False)
ax[0].set_title("Airfoil Shape")
ax[0].set_xlabel("$x/c$")
ax[0].set_ylabel("$y/c$")

ax[1].set_xlabel("Iteration")
ax[1].set_ylabel("Lift-to-Drag Ratio $C_L/C_D$ [-]")
plt.tight_layout()

cmap = LinearSegmentedColormap.from_list(
    "custom_cmap",
    colors=[
        p.adjust_lightness(c, 0.8) for c in
        ["orange", "darkseagreen", "dodgerblue"]
    ]
)

colors = cmap(np.linspace(0, 1, len(airfoil_history)))
ims = []
for i in range(len(airfoil_history)):
    plt.sca(ax[0])
    plt.plot(
        airfoil_history[i].x(),
        airfoil_history[i].y(),
        "-",
        color=colors[i],
        alpha=0.2,
    )
    plt.axis('equal')
```

```
    plt.sca(ax[1])
    if i > 0:
        p.plot_color_by_value(
            np.arange(i),
            np.array([
                np.mean(aero_history[j]["CL"] / aero_history[j]["CD"])
                for j in range(i)
            ]),
            ".-",
            c=np.arange(i),
            cmap=cmap,
            clim=(0, len(airfoil_history)),
            alpha=0.8
        )

    plt.suptitle(f"Optimization Progress")

    ims.append([
        *ax[0].get_children(),
        *ax[1].get_children(),
        *fig.get_children(),
    ])
ims.extend([ims[-1]] * 30)
ani = ArtistAnimation(fig, ims, interval=100)
print(ani)
writer = matplotlib.animation.PillowWriter(fps=10)
ani.save("assets/airfoil_optimization.gif", writer=writer)
writer = matplotlib.animation.FFMpegWriter(fps=10)
try:
    ani.save("assets/airfoil_optimization.mp4", writer=writer)
except FileNotFoundError as e:
    raise FileNotFoundError(
        "You likely do not have ffmpeg on PATH and need to install it / set up PATH accordingly."
    ) from e

del ani
print('Done!')
```

Listing B.1: NeuralFoil via AeroSandbox run script for the fx63137sm-il airfoil at low-speed

Bibliography

[1] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[2] Somdatta Goswami, Cosmin Anitescu, Souvik Chakraborty, and Timon Rabczuk. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *Theoretical and Applied Fracture Mechanics*, 106:102447, 2020.

[3] Thomas P Dussauge, Woong Je Sung, Olivia J Pinon Fischer, and Dimitri N Mavris. A reinforcement learning approach to airfoil shape optimization. *Scientific Reports*, 13(1):9753, 2023.

[4] Klapa Antonion, Xiao Wang, Maziar Raissi, and Laurn Joshie. Machine learning through physics–informed neural networks: Progress and challenges. *Academic Journal of Science and Technology*, 9(1):46–49, 2024.

[5] Gustavo LO Halila, Joaquim RRA Martins, and Krzysztof J Fidkowski. Adjoint-based aerodynamic shape optimization including transition to turbulence effects. *Aerospace Science and Technology*, 107:106243, 2020.

[6] Xueyi Song, Lin Wang, and Xianwu Luo. Airfoil optimization using a machine learning-based optimization algorithm. In *Journal of Physics: Conference Series*, volume 2217, page 012009. IOP Publishing, 2022.

[7] Marco Zanichelli. Shape optimization of airfoils by machine learning-based surrogate models. 2020.

[8] Efstratios L Ntantis and Vasileios Xezonakis. Aerodynamic design optimization of a naca 0012 airfoil: An introductory adjoint discrete tool for educational purposes. *International Journal of Mechanical Engineering Education*, page 03064190241254020, 2024.

[9] Ping He, Charles A Mader, Joaquim RRA Martins, and Kevin J Maki. Dafoam: An open-source adjoint framework for multidisciplinary design optimization with openfoam. *AIAA journal*, 58(3):1304–1319, 2020.

[10] Peter D Sharpe and R John Hansman. *Aerosandbox: A differentiable framework for aircraft design optimization*. PhD thesis, 2021.

[11] M Alizadeh and SM Sadrameli. Numerical modeling and optimization of thermal comfort in building: Central composite design and cfd simulation. *Energy and Buildings*, 164:187–202, 2018.

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]