MARMARA UNIVERSITY
FACULTY OF ENGINEERING

DEVELOPMENT OF FRAMEWORKS FOR AERODYNAMIC SHAPE OPTIMIZATION USING OPEN-SOURCE COMPUTATIONAL FLUID DYNAMICS (CFD) SOFTWARE AND ARTIFICIAL INTELLIGENCE

BERKAN DIKMEN, EMRE KALDIRAK,
IBRAHIM BATUHAN FILIZ, MURAT SELANIK

GRADUATION PROJECT REPORT
Department of Mechanical Engineering

Supervisor
Prof. Dr. Emre ALPMAN

ISTANBUL, 2024

MARMARA UNIVERSITY
FACULTY OF ENGINEERING

Development of Frameworks for Aerodynamic Shape Optimization Using Open-Source Computational Fluid Dynamics (CFD) Software and Artificial Intelligence

by

Berkan Dikmen, Emre Kaldirak, Ibrahim Batuhan Filiz,

Murat Selanik

Jun 7, 2024, Istanbul

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF

BACHELOR OF SCIENCE

AT

MARMARA UNIVERSITY

Signature of Author(s) .....

Department of Mechanical Engineering

Certified By .....

Project Supervisor, Department of Mechanical Engineering

Accepted By.....

Head of the Department of Mechanical Engineering

ACKNOWLEDGEMENT

We are deeply grateful to all those who contributed to the completion of this thesis. First and foremost, we would like to express our sincere gratitude to our advisor, Prof. Dr. Emre Alpman, for their unwavering support, insightful guidance, and invaluable feedback throughout this research journey. Their expertise and encouragement have been instrumental in shaping this work.

We would like to thank TUSAŞ for accepting us into this insightful and leveraging programme. Also, we cannot underestimate their guidance, support and feedback during this journey.

On the other hand, we would like to thank TÜBİTAK for letting us use their experience, knowledge and opportunities among our research.

We are also grateful to the faculty and staff of the Mechanical Engineering Department for providing a stimulating academic environment and the necessary resources to pursue this research.

To everyone who has contributed to our academic and personal growth, thank you. This thesis is a testament to your support and belief in our potential.

June, 2024 Murat Selanik, Emre Kaldirak, Berkan Dikmen, Ibrahim Batuhan Filiz

CONTENTS

ABSTRACT

This study examines a support vector regression hyperparameter optimization process aimed at improving the accuracy of aerodynamic coefficient predictions using airfoil data. The problem centers on the difficulty of accurately predicting aerodynamic coefficients. The research aims to enhance the support vector regression model's performance by identifying the optimal hyperparameters. Accordingly, GNU OCTAVE was used for database creation and preprocessing stages, while Python was utilized for hyperparameter tuning and model training.

In this study, the GridSearchCV method was employed to determine the best values for the C, gamma, and epsilon parameters at different angles of attack. This method facilitates a systematic search of hyperparameters and selection of the combinations that yield the best performance. The results showed that the optimized support vector regression model significantly improved the prediction of lift, drag, and moment coefficients. Model performance was evaluated using high R^2 scores, demonstrating that accurate hyperparameter tuning substantially enhances the model's predictive accuracy.

The findings underscore the critical role of iterative hyperparameter optimization in aerodynamic applications. This study highlights the importance of hyperparameter optimization for enhancing the performance of the support vector regression model and demonstrates how properly implementing this process can maximize the model's predictive accuracy. Consequently, the support vector regression model with aerodynamic shape optimization emerges as an effective method for obtaining accurate and faster predictions of aerodynamic coefficients. This contributes to achieving reliable results in engineering and aerodynamic studies by shortening the process.

Bu çalışma, kanat profili verilerini kullanarak aerodinamik katsayı tahminlerinin doğruluğunu artırmayı amaçlayan bir destek vektör regresyonu hiperparametre optimizasyonu sürecini incelemektedir. Problemin temelini, aerodinamik katsayıların doğru bir şekilde tahmin edilmesinin zorluğu oluşturmaktadır. Araştırmanın amacı, destek vektör regresyonu modelinin performansını artırmak için en uygun hiperparametrelerin belirlenmesidir. Bu doğrultuda, veritabanı oluşturma ve ön işleme aşamaları için GNU OCTAVE kullanılırken, hiperparametre ayarlaması ve model eğitimi için Python tercih edilmiştir.

Çalışma kapsamında, farklı hücum açılarında C, gamma ve epsilon parametrelerinin en iyi değerlerini belirlemek için GridSearchCV yöntemi uygulanmıştır. Bu yöntem, hiperparametrelerin sistematik bir şekilde taranmasını ve en iyi performansı veren kombinasyonların seçilmesini sağlamıştır. Elde edilen sonuçlar, optimize edilen SVM modelinin, kaldırma, sürükleme ve moment katsayılarının tahmininde önemli iyileşmeler sağladığını göstermektedir. Model performansı, yüksek R^2 skorları ile değerlendirilmiş ve bu skorlar, doğru hiperparametre ayarlamasının modelin tahmin doğruluğunu büyük ölçüde artırdığını kanıtlamıştır.

Bulgular, iteratif hiperparametre optimizasyonunun aerodinamik uygulamalarda kritik bir rol oynadığını ortaya koymaktadır. Bu çalışma, destek vektör regresyonu modelinin performansını artırmak için hiperparametre optimizasyonunun önemini ve bu sürecin doğru bir şekilde uygulanmasının modelin tahmin doğruluğunu nasıl maksimize edebileceğini göstermektedir. Sonuç olarak destek vektör regresyonu modeli ile aerodinamik şekil optimizasyonu, aerodinamik katsayı tahminlerinin doğru ve daha hızlı bir şekilde elde edilmesi için etkili bir yöntem olarak ön plana çıkmaktadır. Bu mühendislik ve aerodinamik çalışmalarında işlem sürecinin kısaltılarak güvenilir sonuçlar elde edilmesine katkı sağlamktadır.

SYMBOLS

C : Regularization parameter

C_D : Drag Coefficient

C_L : Lift Coefficient

C_M : Moment Coefficient

Re : Reynolds Number

Ma : Mach Number

n : Number of the Data Points

y_c : Camber Function

y_l : Lower Side Function

y_t : Thickness Function

y_u : Upper Side Function

a_i : Lagrangian Multiplier

a_n : Thickness Function Coefficients

acle : Leading Edge Camber Line Angle

acte : Trailing Edge Camber Line Angle

atte : Trailing Edge Thickness Line Angle

b : Bias

b_n : Camber Function Coefficients

c : Chord

w : Weight Vector

xcmc : X Position for Maximum Camber

xtle : X Position for Leading Edge Control Point

xtth : X Position for Maximum Thickness

y_i : Class Label; Test Value from The Database

$\hat{y}_i$: Predicted Value from The Program

$\bar{y}$: Mean Value of Test Values From The Database

ycmc : Maximum Camber

ycte : Trailing Edge Camber

ycth : Camber at Maximum Thickness

ytte : Trailing Edge Thickness

ytle : Y Position for Leading Edge Control Point

ytth : Maximum Thickness

$\boldsymbol{\xi_i}$: Slack Variables

$\boldsymbol{\forall}$: For All

$\boldsymbol{\varepsilon_{absolute}}$: Absolute Error

ABBREVIATIONS

2D : Two Dimensions

3D : Three Dimensions

AI : Artificial Intelligence

AOA : Angle of Attack

API : Application Programming Interface

CFD : Computational Fluid Dynamics

CST : Class-Shape Transformation

DBSCAN : Density Based Spatial Clustering of Applications with Noise

DQN : Deep Q Networks

GRIDSEARCHCV : Grid Search Cross-Validation

GNU : GNU's Not Unix

K-NN : K-Nearest Neighbors

LIBSVM : Library for Support Vector Machine

MA : Mach Number

ML : Machine Learning

MIT : Massachusetts Institute of Technology

MSE : Mean Squared Error

NACA : National Advisory Committee for Aeronautics

NAN : Not a Number

OPENFOAM : Open-Source Field Operation and Manipulation

PARSEC : Parametrization Based on Airfoil Shape Equations

PCA : Principal Component Analysis

PDE : Partial Differential Equation

RBF : Radial Basis Function

RMSE : Root Mean Squared Error

SU2 : Stanford University Unstructured

SVM : Support Vector Machine

SVR : Support Vector Regression

LIST OF FIGURES

PAGE

LIST OF TABLES

PAGE

1. INTRODUCTION

In the aviation industry, the aerodynamic performance of wing profiles is a critical factor when designing aircraft. As the demand for more efficient and high-performance aircraft increases, the necessity for using advanced techniques in wing profile design also grows. This report presents a comprehensive study integrating innovative wing profile parameterization methods, open-source Computational Fluid Dynamics (CFD) tools, and machine learning techniques to optimize wing profile performance. (1)

Wing profile parameterization is essential for reducing the complex shapes of wing profiles into manageable parameters, enabling more effective design and analysis processes. Among the various methods available, Sobieczky parameterization offers significant advantages due to its flexibility and efficiency. This method allows for rapid changes and accurate aerodynamic performance evaluations, making it an ideal choice for iterative design processes. (2)

In this study, the adoption of the open-source CFD tool XFOIL plays a critical role. These tools provide cost-effective, robust, and highly customizable solutions for simulating and analyzing the airflow around wing profiles. Their open-source nature not only reduces the costs associated with proprietary software but also provides a collaborative environment where continuous improvements and innovations can be integrated.

Machine learning, particularly Support Vector Machines (SVM), is used in this report to enhance predictive capabilities and performance evaluations. The ability of SVMs to handle high-dimensional data and make robust predictions makes them a valuable asset in the aerodynamic optimization process. By integrating machine learning with traditional CFD methods, this study aims to push the boundaries of what is possible in wing profile design and analysis.

This report is structured to provide a detailed exploration of these methodologies and their applications in wing profile optimization. By combining advanced parameterization techniques, open-source CFD tools, and machine learning, this study aims to contribute innovative solutions to the field of aerospace engineering, enhancing the aerodynamic performance of wing profiles.

2. THEORITICAL BACKGROUND

2.1. Airfoil Parameterization and Database Creation

2.1.1. Definition and Importance of Parameterization

Parametrization is a mathematical process used to represent geometric shapes or mathematical functions using parameters. In the context of aerodynamics and airfoil design, parametrization involves describing the shape of an airfoil using a set of parameters or variables.

Instead of specifying the exact coordinates of every point on the airfoil's surface, parametrization allows us to define the shape using a smaller set of parameters that control key features such as thickness, camber, and curvature. These parameters can be adjusted to create different airfoil shapes without having to redefine the entire geometry.

Parametrization simplifies the design process and allows for easier manipulation of airfoil shapes. It is commonly used in CFD simulations, where varying airfoil shapes need to be analyzed efficiently.

2.1.2. Most Common Parameterization Methods

In the field of aerospace engineering, particularly in the design and analysis of airfoils (airfoils), several parametrization methods are commonly used. These methods help in defining the shape of an airfoil precisely and facilitate optimization processes for improved aerodynamic performance. The most common parametrization methods include classical geometric parameters, the NACA series, the CST (Class-Shape Transformation) method, Bézier and B-spline curves, PARSEC (Parameterization Based on Airfoil Shape Equations), and Fourier series.

Classical geometric parameters include the thickness-to-chord ratio (t/c), which defines the maximum thickness of the airfoil relative to its chord length; the camber line, which is the curve that is equidistant from the upper and lower surfaces of the airfoil and represents its mean line; and the leading and trailing edge radii, which describe the curvature at the leading and trailing edges of the airfoil.

The NACA (National Advisory Committee for Aeronautics) series defines airfoils using a series of digits. For example, a NACA 2412 airfoil has a specific camber, thickness, and chord length. The digits in the series provide a systematic way to define airfoil shapes. (3)

The CST method represents the shape of an airfoil using a combination of class functions and shape functions. It provides a flexible and efficient way to describe complex airfoil geometry. (4)

Bézier and B-spline curves are used to define the shape of an airfoil through control points. These curves offer high flexibility and smoothness in the design process. (5)

PARSEC is a method that uses a set of parameters specifically chosen to describe airfoil shapes, such as leading-edge radius, maximum thickness, and camber position. This method allows for efficient aerodynamic optimization. (6)

Fourier series can be used to parametrize the airfoil surface by decomposing it into a series of sine and cosine functions. This method is useful for capturing periodic features of the airfoil shape. (7)

2.1.3. Chosen Parameterization Method in This Study

Sobieczky parametrization is utilized to represent a wide variety of aircraft wing profiles using a manageable number of parameters. This method relies on mathematical expressions known as shape functions to accurately model the geometry of the airfoil, capturing key features such as curvature, thickness distribution, and camber. The primary advantage of this parametrization lies in its flexibility and efficiency, enabling rapid modifications and evaluations of aerodynamic performance. Therefore, this method has been employed in this study. This method has been thoroughly described under the "3.1.1 Parameterization" section.

2.2. Open-Source Computational Fluid Dynamics (CFD)

2.2.1. The Necessity of Open-Source Software

The primary reasons for preferring open-source software in projects include cost advantages and extensive community support. Open-source software is usually free, allowing project managers to avoid high licensing fees for programs that often perform the same functions. Supported by broad and active communities, open-source software is continuously developed, helping users solve problems and improve the software. Lastly, open-source projects, with contributions from a wide developer base, evolve quickly, enabling rapid integration of innovations into the software.

2.2.2. Overview of Open Source CFD Programs

Open-source Computational Fluid Dynamics programs offer accessible and customizable solutions for researchers, engineers, and enthusiasts. Among the prominent open-source CFD programs are OpenFOAM, SU2, and FEniCS. In addition to these, XFOIL stands out as a specialized tool for analyzing subsonic isolated airfoils, making it a valuable asset in aerodynamic design.

OpenFOAM (Open-Source Field Operation and Manipulation) stands out as a leading open-source CFD software. It boasts a comprehensive suite of solvers and utilities for simulating a wide range of fluid flow problems, from incompressible to compressible, turbulent to laminar, and single-phase to multiphase flows. OpenFOAM's modular architecture and object-oriented design make it highly flexible and extensible, allowing users to customize and extend its capabilities for specific applications. Developed and maintained by the OpenFOAM Foundation, this software is widely used in academia, research institutions, and industry for both fundamental research and practical engineering simulations. (8)

SU2 (Stanford University Unstructured) is another notable open-source CFD code developed primarily by researchers at Stanford University. It offers a comprehensive suite of solvers and optimization tools for aerodynamic and multidisciplinary analysis and design optimization. SU2 is designed to handle a variety of flow regimes, including subsonic, transonic, supersonic, and hypersonic flows. Its unstructured mesh capabilities enable users to simulate complex geometries with ease, while its parallel computing capabilities enhance scalability and performance. SU2 is actively developed and supported by a vibrant community of researchers and engineers, making it a valuable resource for academic and industrial applications. (9)

FEniCS is an open-source finite element library for solving partial differential equations (PDEs), including those governing fluid flow. FEniCS provides a high-level interface for defining and solving complex PDE problems, making it particularly well-suited for researchers and developers who require flexibility and control over the mathematical formulations of their simulations. With its Python-based scripting interface, FEniCS offers a user-friendly environment for prototyping and exploring new algorithms and numerical methods. While FEniCS is not specifically tailored for CFD, its versatility and generality make it a powerful tool for a wide range of scientific computing applications, including fluid dynamics. (10)

2.2.3. XFOIL

XFOIL is an open-source program written in FORTRAN by Mark Drela, used for analyzing and optimizing airfoils at subsonic speeds. XFOIL is a specialized CFD tool specifically designed for the analysis of subsonic isolated airfoils. Developed by Mark Drela at MIT, XFOIL is highly efficient and widely used in the aerospace industry and academia for preliminary airfoil design and analysis. (11)

It can determine the lift and drag characteristics of airfoils, defined by 2D coordinates, resulting from pressure differences on the upper and lower surfaces. The program integrates a panel method for potential flow analysis with a boundary layer solver that accounts for viscous effects, providing accurate predictions of airfoil performance characteristics such as lift, drag, and pressure distribution. (11)

XFOIL provides quick results for low Reynolds numbers at subsonic speeds and is particularly valued for its speed and simplicity, making it an ideal tool for rapid airfoil evaluation and optimization during the early stages of aerodynamic design. Detailed analyses can be performed with parameters such as the thickness, angle of attack, and camber line of the airfoils. Being open-source, XFOIL can be easily installed by users.

The main advantages of XFOIL compared to other programs are as follows: it is open-source, provides quick results at low Reynolds numbers, and has a large user base. However, technical knowledge may be required to perform operations in the user interface, and the program is limited for 3D analyses and high Reynolds numbers.

2.3. Machine Learning Applications

2.3.1. Introduction to Machine Learning

Machine learning (ML) is a subset of artificial intelligence (AI) focused on developing algorithms and statistical models for computers to perform tasks without explicit instructions. Instead, ML systems learn from data, identifying patterns and making decisions based on observations. Bishop highlights the goal: enabling systems to enhance performance on tasks through experience, automating analytical model building. (12) ML finds applications in predictive analytics, natural language processing, image recognition, and autonomous systems.

Common ML algorithms include supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning.

Supervised learning:

Trained on labelled datasets, each example paired with an output label. It learns mappings from inputs to outputs, useful for tasks like classification and regression. Algorithms include linear regression, logistic regression, decision trees, support vector machines, and neural networks. (13)

Table 2.1. Common Machine Learning Algorithms and Their Descriptions

Algorithm	Description
Linear Regression	A simple algorithm used for predicting continuous values based on input features.
Logistic Regression	Used for binary classification problems, where the goal is to predict the probability of an instance belonging to a particular class.
Decision Trees	A versatile algorithm that can be used for both classification and regression tasks. It partitions the feature space into regions and makes predictions based on the majority class or average value in each region.
Random Forests	An ensemble method that builds multiple decision trees and combines their predictions to improve accuracy and reduce overfitting.
Support Vector Machines	A powerful algorithm for both classification and regression tasks. SVM finds the optimal hyperplane that separates data points of different classes with the maximum margin.
k-Nearest Neighbors (k-NN)	A simple algorithm that classifies data points based on the majority class of their nearest neighbors in the feature space.

Unsupervised learning:

Works with unlabeled data, aiming to learn data structures without predefined labels. Techniques include clustering (e.g., k-means, hierarchical clustering, DBSCAN) and association rule learning (e.g., Apriori), often for exploratory data analysis and pattern recognition. (12)

Table 2.2. Common Clustering and Dimensionality Reduction Algorithms

Algorithm	Description
K-Means Clustering	A popular algorithm for partitioning data into a predefined number of clusters based on similarity or distance metrics.
Hierarchical Clustering	Builds a hierarchy of clusters by recursively merging or splitting data points based on similarity or distance metrics.
Principal Component Analysis (PCA)	A dimensionality reduction technique used to project high-dimensional data onto a lower-dimensional space while preserving as much variance as possible.
t-Distributed Stochastic Neighbor Embedding	Another dimensionality reduction technique that focuses on preserving the local structure of the data, often used for visualizing high-dimensional datasets in lower-dimensional space.

Semi-supervised learning:

Utilizes both labeled and unlabeled data for training, beneficial when obtaining labeled data is challenging. Methods like self-training, co-training, and graph-based approaches improve learning accuracy by leveraging labeled data to understand unlabeled data. (14)

Reinforcement learning:

Trains agents to make decisions by rewarding desired actions and penalizing undesired ones. The agent learns a policy to maximize cumulative reward, applicable in robotics, game playing, and navigation. Key algorithms include Q-learning, deep Q-networks (DQN), and policy gradient methods. (15)

Table 2.3. Common Reinforcement Learning Algorithms and Their Descriptions

Algorithm	Description
Q-Learning	A model-free reinforcement learning algorithm that learns to make decisions by iteratively updating a Q-value function based on rewards received from actions taken in the environment.
Deep Q-Networks (DQN)	A variant of Q-learning that uses deep neural networks to approximate the Q-value function, enabling it to handle high-dimensional state spaces.
Policy Gradient Methods	Reinforcement learning algorithms that directly learn a policy function that maps states to actions, often used in continuous action spaces.

2.3.2. Fundamentals of Support Vector Machines

A Support Vector Machine is a supervised machine learning algorithm used primarily for classification and regression tasks. SVMs are particularly well-suited for binary classification problems, where the objective is to separate two classes with a clear margin. The main idea behind SVM is to find the optimal hyperplane that maximizes the margin between the two classes in the feature space.

In a typical SVM, the training data is represented as points in a high-dimensional space, and the algorithm constructs a hyperplane or set of hyperplanes in this space. The best hyperplane is the one that has the maximum distance (margin) from the nearest data points of any class, known as support vectors. This approach helps ensure that the classifier has good generalization capabilities on unseen data. (16)

SVM can handle both linear and non-linear classification. For linear classification, it finds the hyperplane directly in the feature space. However, for non-linear classification, SVM uses a technique called the kernel trick. The kernel trick involves transforming the original feature space into a higher-dimensional space where a linear separator can be found. Common kernel functions include the polynomial kernel, radial basis function (RBF) kernel, and sigmoid kernel. These functions enable SVM to fit the optimal hyperplane in complex spaces, making it versatile for a variety of classification problems. (17)

Moreover, SVMs are known for their robustness and effectiveness in high-dimensional spaces. They are particularly effective in cases where the number of dimensions exceeds the number of

samples, as they are less prone to overfitting compared to other classifiers. This characteristic makes SVM a popular choice in fields such as bioinformatics, text categorization, and image recognition. (18)

Despite their advantages, SVMs can be computationally intensive, especially with large datasets. The training process involves solving a quadratic optimization problem, which can be slow. Additionally, choosing the right kernel and tuning hyperparameters such as the regularization parameter (C) and kernel parameters is crucial for optimal performance. (19)

2.3.3. Mathematical Framework of SVM

Support Vector Machines are a powerful set of supervised learning algorithms used for classification and regression tasks. (16) The mathematical foundation of SVM is grounded in concepts from linear algebra, optimization, and statistical learning theory.

At its core, an SVM aims to find the optimal hyperplane that separates data points of different classes in a high-dimensional space. For linearly separable data, the goal is to identify the hyperplane that maximizes the margin between two classes. The margin is defined as the distance between the hyperplane and the nearest data point from either class. These nearest points are known as support vectors. The equation of the hyperplane can be written $w \cdot x + b = 0$ where w is the weight vector and b is the bias term. (17)

Mathematically, the problem can be formulated as an optimization problem:

$$min_{w,b} \frac{1}{2} ||w||^2 \tag{2-1}$$

subject to the constraints:

$$y_i(w \cdot x_i + b) \geq 1 \quad \forall i \tag{2-2}$$

Where y_i is the class label of data point x_i.

For cases where the data is not linearly separable, SVM introduces a soft margin approach, allowing some misclassifications by incorporating slack variables ξ_i:

$$min_{w,b} \frac{1}{2} ||w||^2 + C \sum_{i=1}^{n} \xi_i \tag{2-3}$$

subject to:

$$y_i(w \cdot x + b) \geq 1 - \xi_i \quad \forall i \tag{2-4}$$

Here, C is a regularization parameter that controls the trade-off between maximizing the margin and minimizing the classification error. (16)

In scenarios where the data is not linearly separable even with a soft margin, SVM employs the kernel trick. A kernel function $K(x_i, x_j)$ maps the input data into a higher-dimensional feature space where a linear separation is possible. Common kernel functions include the polynomial kernel, Gaussian (RBF) kernel, and sigmoid kernel. (20)

The optimization problem in the dual form, incorporating the kernel function, is:

$$\sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=i}^{n} a_i a_j y_i y_j K(x_i, x_j) \tag{2-5}$$

subject to:

$$0 \leq a_i \leq C \quad \forall i \tag{2-6}$$

$$\sum_{i=1}^{n} a_i y_i = 0 \tag{2-7}$$

Here, a_i are the Lagrange multipliers.

SVM's robustness and flexibility come from its solid mathematical foundation, which ensures that it finds a global optimum due to its convex optimization problem formulation.

2.3.4. Justification for Using SVM in This Study

Support Vector Machine (SVM) is a powerful classification and regression analysis method used in machine learning and data mining. SVM classifies data points by separating them with a hyperplane and finds the hyperplane that provides the best separation for classification. There are several reasons for preferring SVM. First, its ability to perform multivariate analysis makes it effective in environments with multiple variables in aerodynamic analyses, which is a significant advantage. In terms of overall performance, SVM generally provides high accuracy with small and medium-sized data sets and is resistant to overfitting. Its ability to find the hyperplane that best separates the data allows it to perform classification tasks with high accuracy. Additionally, SVM's strong mathematical foundation ensures theoretically

sound and reliable results. The purpose of SVM is to perform classification and regression analyses by optimally separating different data sets. In projects such as aerodynamic shape optimization, SVM can be used to predict and optimize the performance of various airfoil profiles. This allows for the development of more efficient and optimized designs.

2.4. Overview of SVM Libraries and Scikit-learn

2.4.1. Comparison of Various Libraries with a Focus on Scikit-learn

Support Vector Machine (SVM) algorithms are widely used in machine learning and are supported by several powerful libraries. The main libraries that provide implementations of SVM include scikit-learn, LIBSVM, and TensorFlow. Each of these libraries offers unique features, advantages, and use cases.

Scikit-learn is a widely used machine learning library in Python, which provides simple and efficient tools for data mining and data analysis. It is built on NumPy, SciPy, and matplotlib. Scikit-learn offers an easy-to-use implementation of SVM through the `svm` module, which supports various kernels, such as linear, polynomial, and RBF kernels. The library is known for its simplicity and integration with other data processing libraries in Python, making it a favorite among beginners and professionals alike. (21)

LIBSVM is a library specifically designed for SVM. It is written in C++ and provides a command-line interface, as well as interfaces for various programming languages, including Python, Java, and MATLAB. LIBSVM is known for its efficiency and accuracy in handling SVM problems. It supports classification, regression, and distribution estimation. LIBSVM is particularly useful for researchers and developers who require a highly optimized and flexible SVM implementation. (19)

TensorFlow is an open-source machine learning framework developed by Google. Although TensorFlow is primarily known for deep learning, it also supports traditional machine learning algorithms, including SVM. TensorFlow's SVM implementation is part of the `tf.estimator` API, which allows for easy integration with other machine learning models and tools within the TensorFlow ecosystem. TensorFlow is advantageous for users who need to deploy SVM models in production environments or combine them with deep learning models. (22)

Below is a comparison table of these libraries based on various factors:

Table 2.4. Feature Comparison: Scikit-learn, LIBSVM, and TensorFlow for SVM

Feature	Scikit-learn	LIBSVM	TensorFlow
Language	Python	C++, with bindings for Python, Java, MATLAB	Python, C++
Ease of Use	Very user-friendly, simple API	Moderate, command-line interface and programming interfaces	Moderate, requires understanding of TensorFlow
Kernel Support	Linear, Polynomial, RBF, Sigmoid	Linear, Polynomial, RBF, Sigmoid	Linear, Polynomial, RBF, Custom
Performance	Good for small to medium datasets	Excellent, optimized for performance	Good, scalable with TensorFlow infrastructure
Integration	Excellent, integrates well with NumPy, SciPy, pandas, etc.	Good, can be integrated with various languages	Excellent, integrates with deep learning models
Documentation and Community	Extensive documentation, large community support	Good documentation, smaller community	Extensive documentation, large community support
Use Case	General machine learning tasks	Specialized SVM tasks, research	Combining SVM with deep learning models, deployment
Flexibility	High, with many parameters and easy customization	High, with support for various kernels and parameters	High, with extensive API for customization

These libraries are instrumental in implementing and utilizing SVM algorithms effectively across different domains and applications. Each library offers distinct advantages depending on the specific requirements of the task at hand, whether it is ease of use, performance, integration capabilities, or flexibility.

2.4.2. Specific Use Case of Scikit-learn in This Study

Firstly, scikit-learn's ease of use is a significant advantage. Its intuitive API allows quick implementation and experimentation with SVM models, reducing the time and effort required for data preprocessing, training, and evaluation. Pedregosa emphasize that scikit-learn's design focuses on simplicity and efficiency, which can streamline your workflow. (21) Secondly, scikit-learn's integration capabilities are invaluable. It seamlessly integrates with other Python libraries like NumPy and pandas, facilitating smooth handling of airfoil geometry data and XFOIL performance metrics. This enhances productivity and allows for sophisticated data processing and feature engineering.

Moreover, scikit-learn provides comprehensive kernel support for SVM, including linear, polynomial, radial basis function (RBF), and sigmoid kernels. This variety enables experimentation with different kernel functions to find the best fit for your dataset, leading to better model performance and more accurate predictions for airfoil design optimizations. Performance-wise, scikit-learn efficiently handles datasets of your size, with approximately 5000 training samples. While not as optimized as specialized libraries like LIBSVM for extremely large datasets, it balances computational efficiency and ease of use for typical machine learning tasks in engineering projects. Additionally, scikit-learn's extensive documentation and community support are critical. The detailed documentation provides guidance on implementing SVM and other algorithms, and the active community offers support, tutorials, and examples, facilitating smoother learning and development. (21)

Lastly, scikit-learn's general-purpose nature allows for extending your analysis beyond SVM if needed. It provides a robust framework for exploring additional machine learning techniques, such as regression, clustering, or dimensionality reduction, within the same environment.

In summary, scikit-learn's ease of use, integration capabilities, comprehensive kernel support, adequate performance, extensive documentation, and versatility make it an ideal choice for training an SVM on airfoil geometries and XFOIL data.

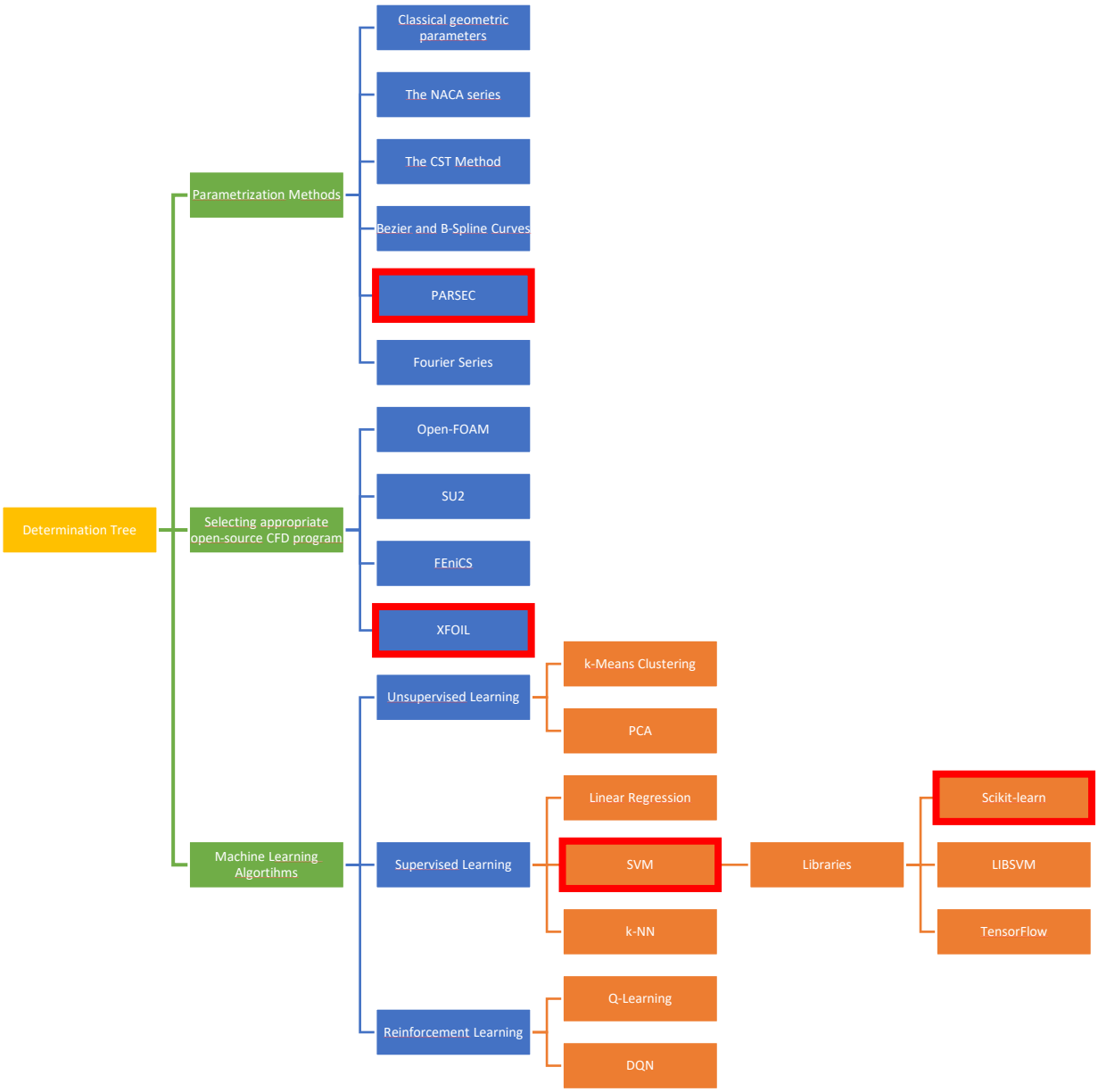


Figure 2.1. Determination Tree

3. PROCESS

In this section, the detailed examination and determination process of parameters used to represent various airfoil types are initially addressed. Subsequently, the process of creating a database based on these parameters and preparing the data is elaborately explained. Following that, the process of determining and evaluating hyperparameters, crucial for machine learning models such as Support Vector Machines, is discussed. Automatic hyperparameter detection methods are examined, and their performance is assessed using R^2 scores. Lastly, while examining the details of the SVM model, emphasis is placed on the separation of training and test data, potential error types, and their causes. Additionally, the potential of ensemble methods such as Adaboost, Bagging, and Gradient Boosting to enhance the performance of the SVM model is explored.

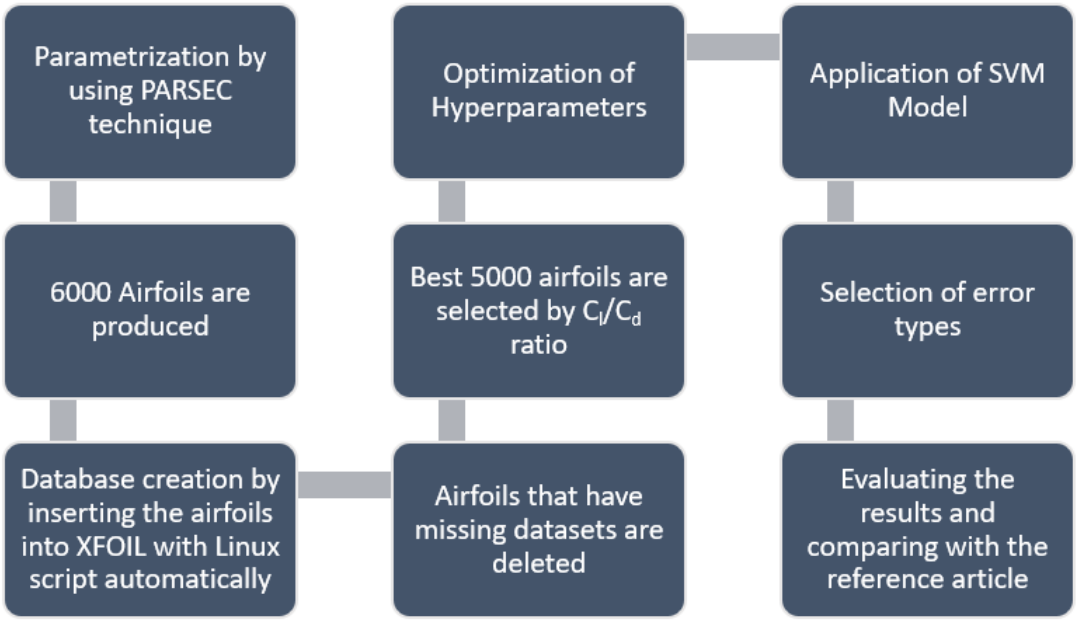


Figure 3.1. Process Flowchart

3.1. Airfoil Parameterization and Database Creation

3.1.1. Parameterization

Sobieczky parametrization is used, which is a widely utilized method in aerodynamics for representing airfoil shapes. (23) This technique is particularly valued for its ability to encapsulate a diverse range of airfoil geometries using a manageable number of parameters. By

employing specific mathematical expressions known as shape functions, Sobieczky parametrization facilitates the accurate modelling of the complex contours of airfoil surfaces.

Shape functions in this context are mathematical formulas that define the geometry of an airfoil's surface. These functions are designed to capture the essential characteristics of airfoil shapes, including curvature, thickness distribution, and camber. The parametrization allows for the efficient adjustment and optimization of airfoil designs, making it a powerful tool in both theoretical studies and practical applications in aerospace engineering.

The primary advantage of Sobieczky parametrization lies in its flexibility and efficiency. By reducing the complexity of airfoil representation to a few key parameters, it enables engineers to perform rapid modifications and assessments of aerodynamic performance. This capability is crucial in the iterative process of airfoil design, where multiple variations must be evaluated to achieve optimal performance characteristics.

This is accomplished using polynomial functions for the airfoil thickness (y_t) and camber (y_c) lines:

$$y_t = a_1\sqrt{x} + a_2x + a_3x^2 + a_4x^3 + a_5x^4 \tag{3-1}$$

$$y_c = b_1x + b_2x^2 + b_3x^3 + b_4x^4 + b_5x^5 + b_6x^6 \tag{3-2}$$

The upper and lower side y-coordinates at a given chord location are given equations below.

$$y_u = y_c + \frac{1}{2}y_t \tag{3-3}$$

$$y_l = y_c - \frac{1}{2}y_t \tag{3-4}$$

The geometric parameters defining the airfoil are: position of the leading edge control point, position and airfoil maximum thickness, trailing edge thickness line angle, trailing edge thickness, leading edge camber line angle, camber at maximum thickness, position and

maximum camber, camber at maximum thickness, trailing edge camber line angle and trailing edge camber. Figure 3.2 shows the parameters used for airfoil definition. Note that the mean curvature line (colored in red) has been multiplied by 10 only for presentation purposes.

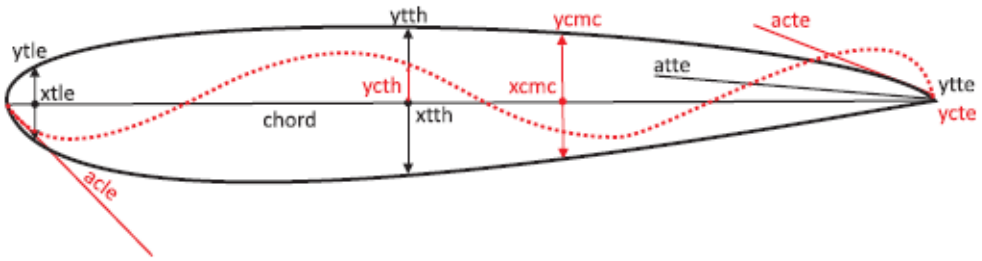


Figure 3.2. Sobieczky's Parametrization for Airfoil Definition **(1)**

Defining of a_n and b_n coefficients in Eqns. (3-1) and (3-2) is solved with airfoil's geometric variables described in **Error! Reference source not found.** as it is shown below.

Table 3.1. Airfoil's Geometric Variables **(1)**

Short Name	Variable	Min.	Max.
xtle	X position for leading edge control point	0.015	0.015
ytle	Y position for leading edge control point	0.036	0.036
xtth	X position for maximum thickness	0.300	0.450
ytth	Maximum thickness	0.100	0.170
atte	Trailing edge thickness line angle	-10.000	-4.500
ytte	Trailing edge thickness	0.006	0.006
acle	Leading edge camber line angle	-7.500	5.000
ycth	Camber at maximum thickness	-0.008	0.005
xcmc	X position for maximum camber	0.700	0.800
ycmc	Maximum camber	-0.010	0.020
acte	Trailing edge camber line angle	-15.000	0.000
ycte	Trailing edge camber	0.000	0.000

Using of Sobieczky's Parametrization, an airfoil shape is defined by basic geometric parameters, instead of the coefficient of shape functions directly. There are two shape functions for thickness distribution and camber line of airfoil. The parameters are set the equation, replaced x and y variables.

To obtain the a_n coefficients, the thickness distribution equations are given by between Eqns. (3-5) and (3-9) below.

$$0 = \frac{a_1}{2\sqrt{xtth}} + a_2 + 2a_3 xtth + 3a_4 xtth^2 + 4a_5 xtth^3 \tag{3-5}$$

$$ytth = a_1\sqrt{xtth} + a_2 xtth + a_3 xtth^2 + a_4 xtth^3 + a_5 xtth^4 \tag{3-6}$$

$$atte = \frac{a_1}{2} + a_2 + 2a_3 + 3a_4 + 4a_5 \tag{3-7}$$

$$ytte = a_1 + a_2 + a_3 + a_4 + a_5 \tag{3-8}$$

$$ytle = a_1\sqrt{xtle} + a_2 xtle + a_3 xtle^2 + a_4 xtle^3 + a_5 xtle^4 \tag{3-9}$$

From Eqn. (3-10) to (3-15), they are used to calculation of b_n coefficients for the camber line below.

$$acle = b_1 \tag{3-10}$$

$$ycth = b_1 xtth + b_2 xtth^2 + b_3 xtth^3 + b_4 xtth^4 + b_5 xtth^5 + b_6 xtth^6 \tag{3-11}$$

$$0 = b_1 + 2b_2 xcmc + 3b_2 xcmc^2 + 4b_4 xcmc^3 + 5b_5 xcmc^4 + 6b_6 xcmc^5 \tag{3-12}$$

$$acte = b_1 + 2b_2 + 3b_3 + 4b_4 + 5b_5 + 6b_6 \tag{3-13}$$

$$ycte = b_1 + b_2 + b_3 + b_4 + b_5 + b_6 \tag{3-14}$$

$$ycmc = b_1 xcmc + b_2 xcmc^2 + b_3 xcmc^3 + b_4 xcmc^4 + b_5 xcmc^5 + b_6 xcmc^6 \tag{3-15}$$

Within the ranges shown in Table 3.1, these design parameters are varied to create the geometries used for training and validation. To compare the results with earlier research, the trailing edge thickness, trailing edge camber, and leading-edge control point are all kept unchanged, as shown in the Table 3.1. Figure 3.3 shows the airfoils' maximum, minimum and mean thickness distribution. A big set of airfoils are included in the database to profit by the huge range of kinds. Unusual and unused airfoil geometries are also implicated in the database to increase the range of dataset.

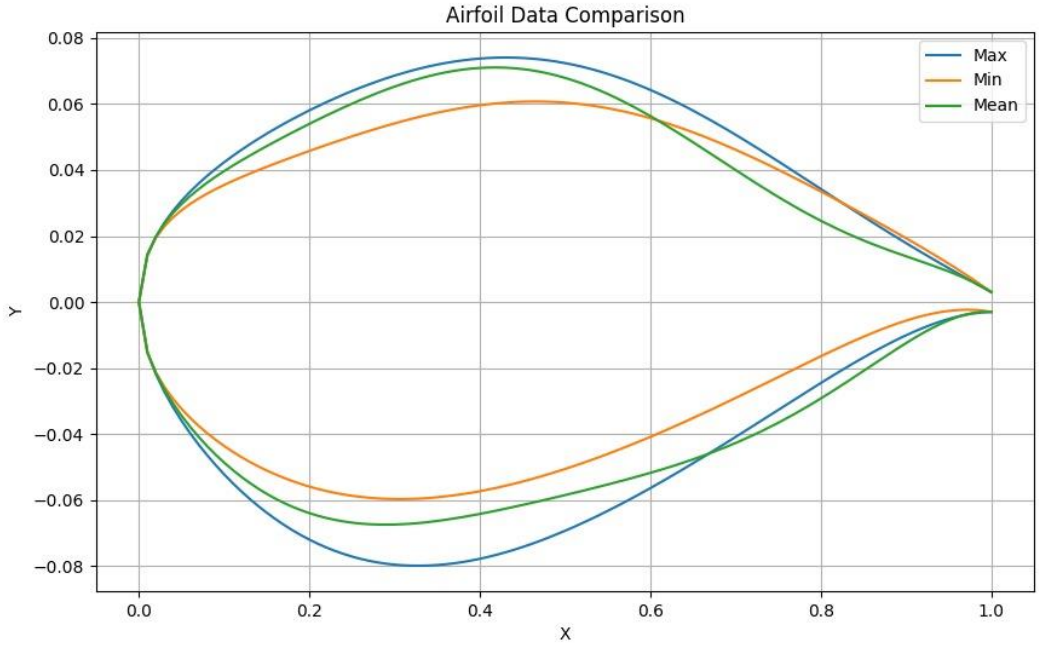


Figure 3.3. The Maximum, Minimum and Mean Thickness Distribution of Airfoil Database

Figure 3.4 shows that the 10 airfoil geometries from the database, which is randomly selected. It is plotted to obtain airfoil geometries' range. The numbers in the legend represent the airfoil numbers from database.

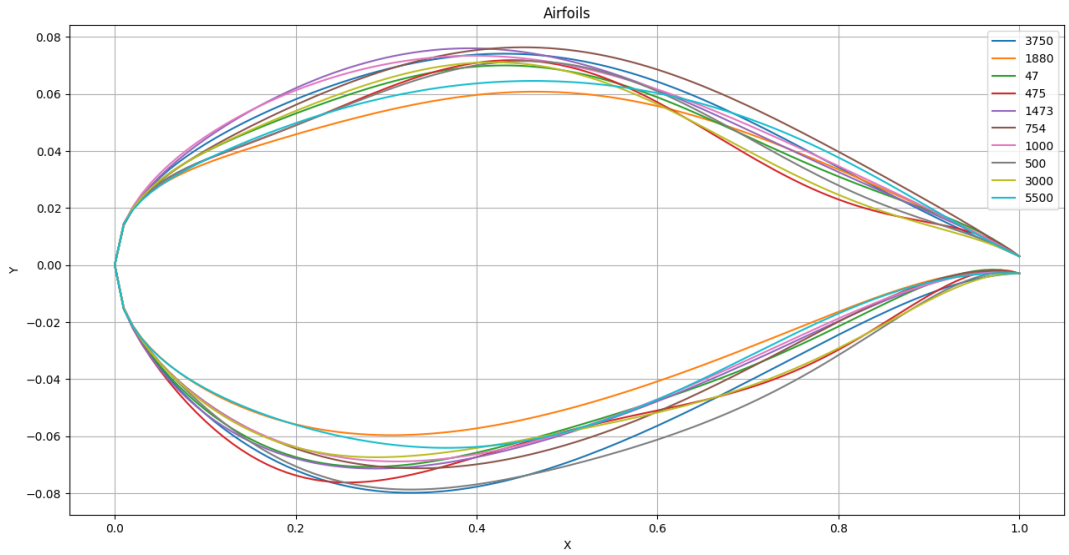


Figure 3.4. The 10 Airfoils from Database

3.1.2. Database Creation

With the given equations above, parametrization is done with a script automatically. The script is coded in GNU OCTAVE, which is an open-source program for the LINUX operating system. The airfoil geometries are constructed solution of above equations and variables are selected randomly for every airfoil separately.

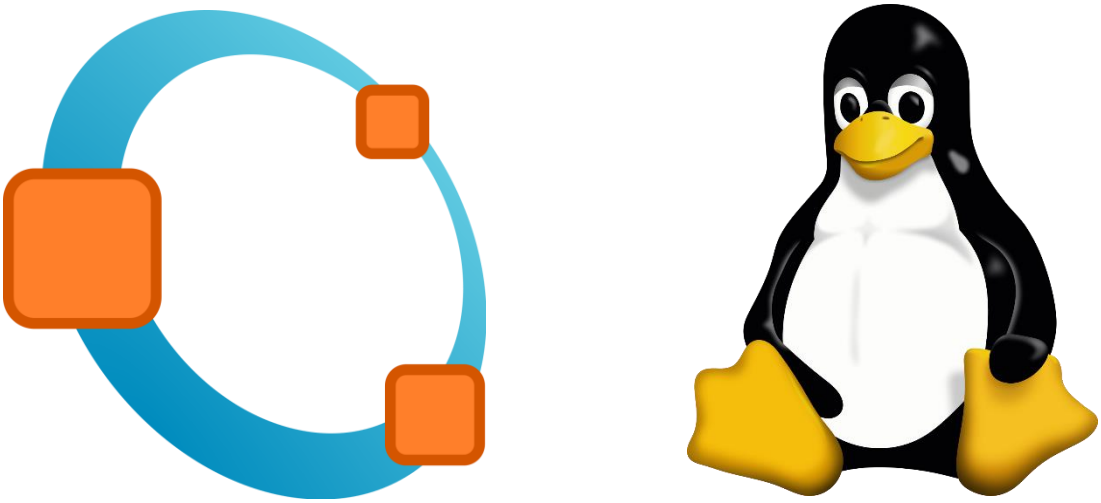


Figure 3.5. GNU OCTAVE and LINUX Logos **(24) (25)**

The loop is constructed before defining of variables, which are xtth, ytth, ycmc, xcmc, acte. The equations are solved again at each airfoil and the airfoils have separated each other in this way. The coordinates of airfoils, x and y are written in text files and airfoil's parameters

like xtth, ytth are written in one text file to identify airfoils in database later. There is some figures from the 'Airfoil Shape Parametrization' script below. (see for codes, section 7.1.1)

```
for m=1:6000
   xtle = 0.015;                              % X Position for leading edge control point (.015)
   ytle = 0.036;                              % Y Position for leading edge control point (.036)
   xtth = .337 + (rand * .075);               % X Position for maximum thicknes (.300<xtth<.450)
   ytth = .117 + (rand * .035);               % Maximum thickness (.100<ytth<<170)
   atte = tand(-8.625 + (rand * 2.75));       % Trailing edge thickness line angle (-10.000<atte<-4.500)
   ytte = .006;                               % Trailing edge thickness (.006)
   acle = tand(-4.375 + (rand * 3.125));      % Leading edge camber line angle (-7.500<acle<5.000)
   ycth = -.004 + (rand * .008);              % Camber ar maximum thickness (-.008<ycth<.005)
   xcmc = .725 + (rand * .05);                % X Position for maximum camber (.700<xcmc<.800)
   ycmc = -.005 + (rand * .017);              % Maximum camber (-.010<ycmc<.020)
   acte = tand(-11.250 + (rand * 7.50));      % Trailing edge camber line angle (-15<acte<0)
   ycte = 0.000;                              % Trailing edge camber (0.000)
```

Figure 3.6. Defining Process of Airfoil Parameters from Script

As shown in Figure 3.6, the script is set to produce 6000 airfoil geometry.

```
eqn_5 = [(1/(2*(xtth)^.5)), (1), (2*xtth), (3*xtth^2), (4*xtth^3)];
eqn_6 = [((xtth)^.5), (xtth), (xtth^2), (xtth^3), (xtth^4)];
eqn_7 = [(1/2), (1), (2), (3), (4)];
eqn_8 = [(1) (1) (1) (1) (1)];
eqn_9 = [((xtle)^.5), (xtle), (xtle^2), (xtle^3), (xtle^4)];
eqn_10 = zeros(1, 6);
eqn_10(1, 1) = 1;
eqn_11 = [(xtth), (xtth^2), (xtth^3), (xtth^4), (xtth^5), (xtth^6)];
eqn_12 = [(1), (2*xcmc), (3*xcmc^2), (4*xcmc^3), (5*xcmc^4), (6*xcmc^5)];
eqn_13 = [(1), (2), (3), (4), (5), (6)];
eqn_14 = [(1), (1), (1), (1), (1), (1)];
eqn_15 = [(xcmc), (xcmc^2), (xcmc^3), (xcmc^4), (xcmc^5), (xcmc^6)];
A = [eqn_5; eqn_6; eqn_7; eqn_8; eqn_9];
A_inverse = inv(A);
B = [eqn_10; eqn_11; eqn_12; eqn_13; eqn_14; eqn_15];
B_inverse = inv(B);
```

```
% Equation 5 from the article
% Equation 6 from the article
% Equation 7 from the article
% Equation 8 from the article
% Equation 9 from the article
% Equation 10 from the article
% Defining of first member of first line as acle
% Equation 11 from the article
% Equation 12 from the article
% Equation 13 from the article
% Equation 14 from the article
% We write one more equation from article to define 6x6 matrix
```

Figure 3.7. Setting Equations to Solve for Coefficients a and b

For every airfoil geometry, the functions are constructed as a matrix and then they are solved for Camber Line and Thickness Line separately. After the defining of coefficients a_n and b_n, functions of Camber and Thickness lines combined for above equations for upper y_u, and lower side y_l of airfoil. The other value of coordinate is completed with this solution and saved in a airfoils text file.

```
y_t = zeros(1, length(x));            % Thickness function
y_c = zeros(1, length(x));            % Camber function
y_u = zeros(1, length(x));            % Upper surface function
y_l = zeros(1, length(x));            % Lower surface function
i = 1;
while i <= length(x)
  y_t(i) = (a_1*((x(i)^.5)))+(a_2*x(i))+(a_3*x(i)^2)+(a_4*x(i)^3)+(a_5*x(i)^4);          % Calculation of thickness values each x values
  y_c(i) = (b_1*x(i))+(b_2*x(i)^2)+(b_3*x(i)^3)+(b_4*x(i)^4)+(b_5*x(i)^5)+(b_6*x(i)^6);  % Calculation of thickness values each x values
  y_u(i) = y_c(i) + 0.5*y_t(i);                                                          % Calculation of upper surface
  y_l(i) = y_c(i) - 0.5*y_t(i);                                                          % Calculation of lower surface
  i = i + 1;
end
```

Figure 3.8. Determination of Airfoil Geometry Coordinates

The coordinate files are saved in a 'Airfoils' file to continue process.

The analysis of airfoils is solved in XFOIL, which is another open-source program. The automation of this process is made with a LINUX bash script from terminal. First, the XFOIL is run to define optimum analysis conditions. Then, the analysis steps recorded in a text file

also, Figure 3.9 shows the optimum conditions, and they are saved as ‘Commands’ in a text file.

```
load
../airfoil1.DAT
pane
ppar

oper
vpar
xtr
0.0
0.1

iter
200
visc
1e6
m
0.3
pacc
polars/polar1.dat

aseq
1
10
1

quit
```

Figure 3.9. Commands for Optimum Analysis Conditions in XFOIL

The airfoil geometries are loaded from external source, so “load” command provides it. and second row defines the file path of airfoil geometry’s coordinate file. The number of airfoil is initialized and increased automatically with bash script. The purpose of using “pane” command is used to define or update the paneling of the airfoil geometry. Paneling is a crucial step in the numerical analysis performed by XFOIL because it discretizes the airfoil surface into small panels, allowing the software to solve the flow equations around the airfoil. The “ppar” command in XFOIL is used to adjust the parameters of the paneling process, allowing for more detailed control over how the airfoil surface is discretized into panels. This is particularly useful when the default paneling generated by the “pane” command needs refinement to capture specific geometric features or flow characteristics more accurately. They are all preprocessing of analysis. The spaces in ‘Command’ file, acts like enter button, when the program is run manually. The “oper” command is started the analysis and in XFOIL, the

“vpar” command is used to set and modify parameters related to the viscous flow analysis, specifically the boundary layer and wake model. These parameters influence how XFOIL computes the effects of viscosity on the airfoil's aerodynamic performance, including drag and separation. The “xtr” command in XFOIL, to specify the chordwise positions where the boundary layer transitions from laminar to turbulent flow on both the upper and lower surfaces of the airfoil. The numbers 0.0 and 0.1 represent specified chordwise positions for airfoil geometry. The space is used to quit this section.

$Re: 10^6$

$0.3\ Ma$

The “iter” command in XFOIL, is used to set the maximum number of iterations allowed for the viscous flow solver to converge during the boundary layer and wake analysis. This command is essential for controlling the convergence behavior of the iterative solver used in viscous flow calculations. The “visc” command in XFOIL activates the viscous flow calculations, enabling the analysis of airfoil performance considering the effects of viscosity. The number after this line represents Reynolds Number and its value is 1000000, one million. The “m” means in file Mach Number, and it is valued 0.3, which is thirty percent of local speed of sound. The “pacc” command is used to save polar file of analysis and below of this line includes name of polar file and location of polar file. The space after this command is used skip the saving of dump file of analysis. The finals step is determination of angle of attack range. The “aseq” command means alpha sequence and it is valued at between 1 and 10, where starts with angle of 1 and finishes with angle of 10. (1)

The bash script is constructed for 6000 airfoils. It is run at terminal in project file. The script is set from the first airfoil’s polar, and it passes to another. Figure 3.10 shows the bash script.

```
#!/bin/bash
rm -rf polars/*
for k in {1..6000}
do
        str=$(printf "%d" $k)
        str1="..\/airfoil${str}.DAT"
        str2="2s/.*/${str1}/"
        sed -i "${str2}" commands.dat
        str3="polars\/polar${str}.dat"
        str5="19s/.*/${str3}/"
        sed -i "${str5}" commands.dat
        xfoil < commands.dat #> "${name}"
done
```

Figure 3.10. The Bash Script to Automatize Analysis Process in XFOIL (see for code, section 7.1.1)

Parametrization and analysis processes, both are made automatically with these scripts. Then, the number of airfoils in reference paper is 5000, so the number of airfoils must be dropped from 6000 to 5000 and it is done two ways, which are missing values and objective function with ratio of C_L and C_D coefficients. Finally, the database is constructed.

3.1.3. Data Preparation

In the process of refining the database, two distinct scripts were employed. Under this heading, we will elaborate on these two sets of code.

Deleting Polar Files with Missing Data

Due to XFOIL, some airfoils may have missing solution data for certain angles. The following code was used to identify and remove polar files with missing data. Additionally, these identified files were noted for the modification of the "parameters" file containing airfoil parameters.

```
      XFOIL         Version 6.99

Calculated polar for: airfoil_10

1 1 Reynolds number fixed          Mach number fixed

xtrf =   0.000 (top)        0.100 (bottom)
Mach =   0.300     Re =     1.000 e 6     Ncrit =   9.000

  alpha    CL        CD       CDp       CM     Top_Xtr  Bot_Xtr
 ------ -------- --------- --------- -------- -------- --------
  1.000   0.2622   0.01187   0.00289  -0.0577   0.0000   0.1000
  2.000   0.3811   0.01224   0.00328  -0.0586   0.0000   0.1000
  3.000   0.4982   0.01275   0.00385  -0.0589   0.0000   0.1000
  4.000   0.6128   0.01344   0.00463  -0.0585   0.0000   0.1000
  5.000   0.7244   0.01434   0.00566  -0.0573   0.0000   0.1000
  6.000   0.8326   0.01548   0.00699  -0.0551   0.0000   0.1000
  7.000   0.9373   0.01692   0.00867  -0.0519   0.0000   0.1000
  8.000   1.0382   0.01876   0.01082  -0.0478   0.0000   0.1000
  9.000   1.1340   0.02115   0.01365  -0.0427   0.0000   0.1000
 10.000   1.2156   0.02442   0.01748  -0.0352   0.0000   0.1000
```

Figure 3.11. A complete polar file

```
      XFOIL         Version 6.99

Calculated polar for: airfoil_40

1 1 Reynolds number fixed          Mach number fixed

xtrf =   0.000 (top)        0.100 (bottom)
Mach =   0.300     Re =     1.000 e 6     Ncrit =   9.000

  alpha    CL        CD       CDp       CM     Top_Xtr  Bot_Xtr
 ------ -------- --------- --------- -------- -------- --------
  1.000   0.2731   0.01139   0.00248  -0.0558   0.0000   0.1000
  2.000   0.3944   0.01175   0.00286  -0.0566   0.0000   0.1000
  3.000   0.5148   0.01225   0.00342  -0.0570   0.0000   0.1000
  4.000   0.6339   0.01292   0.00418  -0.0569   0.0000   0.1000
  6.000   0.8670   0.01492   0.00653  -0.0547   0.0000   0.1000
  7.000   0.9802   0.01637   0.00825  -0.0525   0.0000   0.1000
  8.000   1.0899   0.01826   0.01049  -0.0493   0.0000   0.1000
  9.000   1.1940   0.02079   0.01351  -0.0450   0.0000   0.1000
 10.000   1.2867   0.02444   0.01781  -0.0389   0.0000   0.1000
```

Figure 3.12. A polar file with missing data

As seen in the Figure 3.12, there is no solution available for angle-5 for the 40th airfoil. A total of 89 files with similar issues in the 6000-airfoil database were deleted using the following code.

```python
import pandas as pd
import os

def check_and_delete_files(directory):
    # Check all files in the specified directory
    for filename in os.listdir(directory):
        if filename.startswith('polar') and filename.endswith('.dat'):
            file_path = os.path.join(directory, filename)
            try:
                # Read the file, skip the first 11 lines
                data = pd.read_csv(file_path, skiprows=10, sep=r'\s+')

                # Check the angle values in the "alpha" column
                if 'alpha' in data.columns:
                    alpha_values = data['alpha'].dropna()  # Drop empty values
                    print(f"{len(alpha_values)} alpha values found in {filename}.")
                    if len(alpha_values) == 11:
                        print(f"{filename} file is valid.")
                        continue  # Skip valid files
                # Delete files that do not meet the criteria
                os.remove(file_path)
                print(f"{filename} file deleted because there are not enough data in the alpha column.")
            except Exception as e:
                # Delete files with errors
                os.remove(file_path)
                print(f"{filename} file could not be read and was deleted. Error: {e}")

directory = 'C:/directory/of/files'
check_and_delete_files(directory)
```

Figure 3.13. Deleting Polar Files with Missing Data (see for code section 7.1.2)

This code contains a Python function that checks and deletes specific types of files within a directory. The function iterates through the files in the directory one by one in a loop. It identifies files starting with "polar" and ending with ".dat".

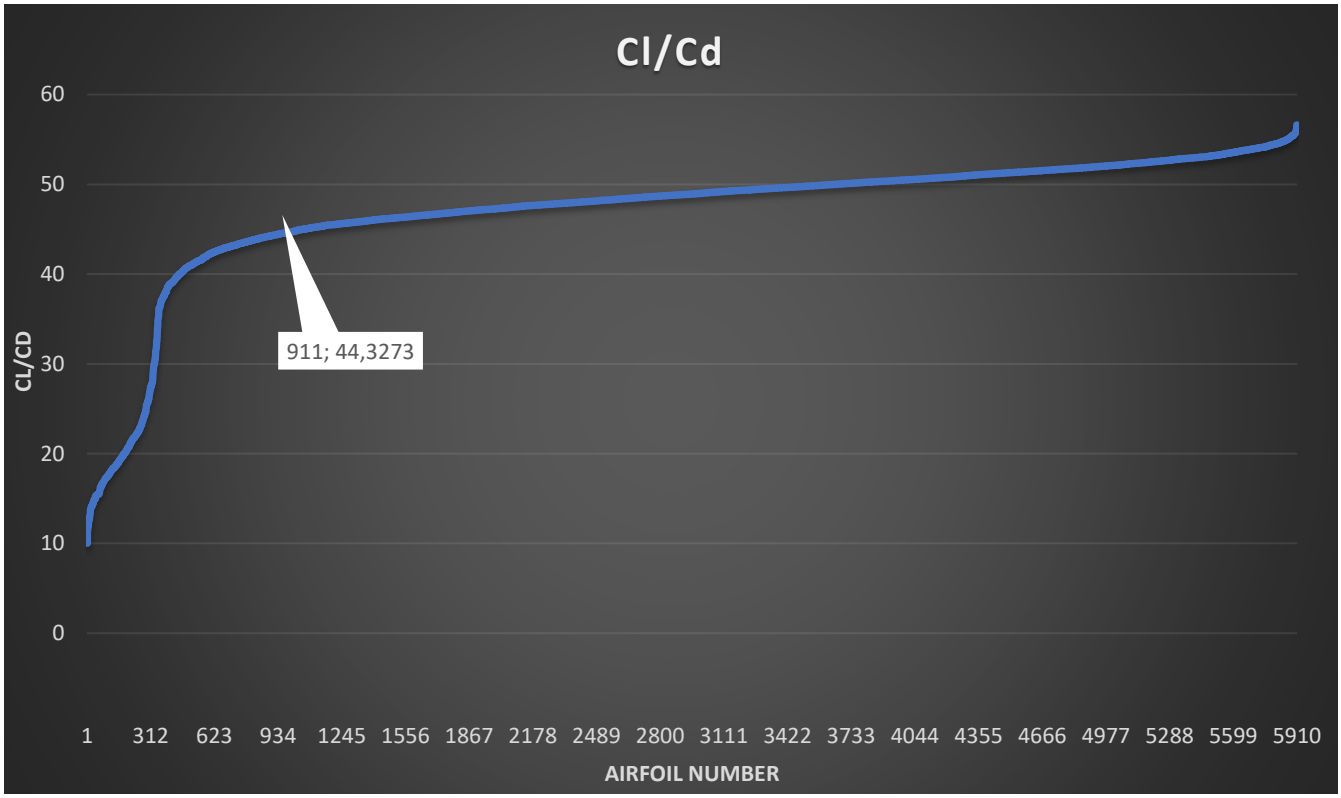
For each file, it first constructs the file path and then reads the file using the Pandas library. It skips the first 11 lines while reading the file because these lines typically contain header or description information. Next, it checks if there are values in the "alpha" column of the file. If values exist, it extracts them and removes any empty values (NaNs).

If the "alpha" column contains exactly 11 values, the file is deemed appropriate, and the process is skipped for that file. However, if there are not 11 values, the file is deleted, and this operation is printed to the screen.

If the file cannot be read (for example, if the data format is different from expected or if the file does not exist), an error is raised. In this case, the relevant file is deleted, and the error message is printed to the screen.

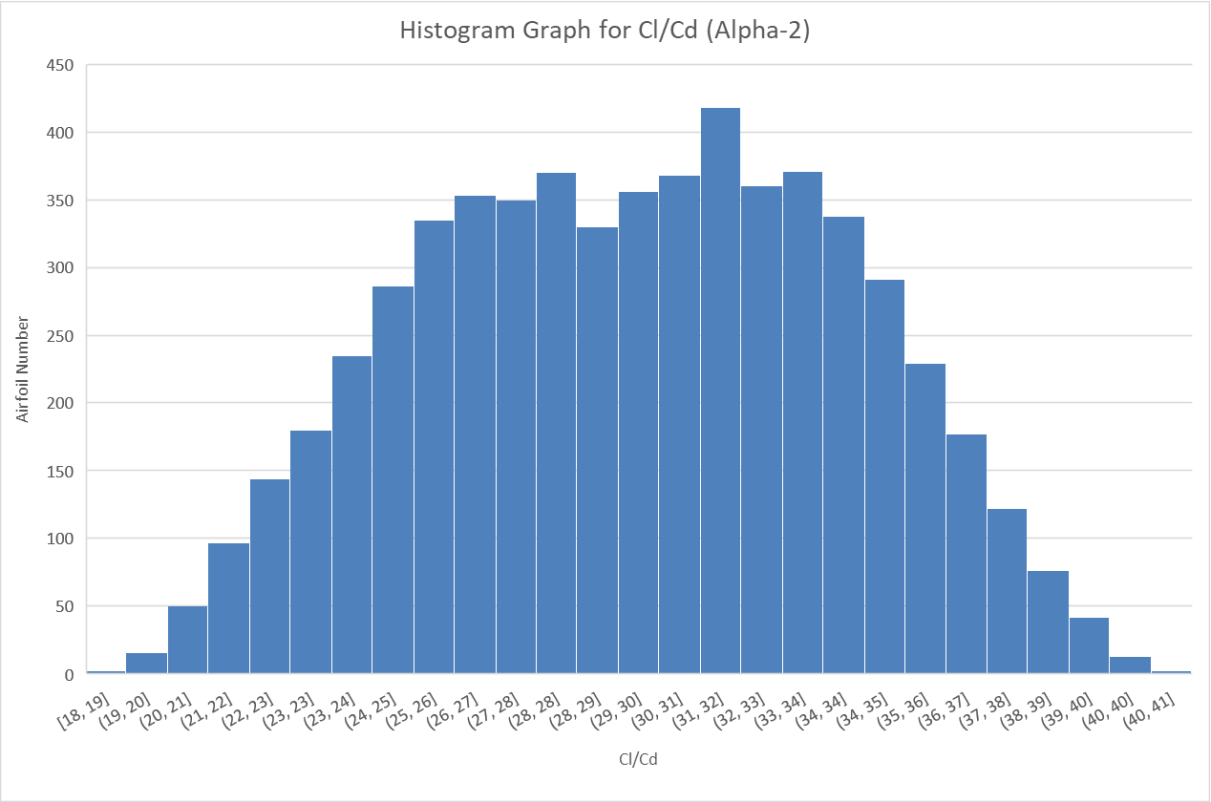
Reducing the Database to 5000 Entries Based on the C_l/C_d Ratio

After removing the files with missing data, we reduced the number of airfoils in the database to 5911. To further reduce the database to 5000 airfoils, we conducted an analysis based on the C_l/C_d ratio. At this point, to improve the database, C_l/C_d values were sorted for 10 degrees and necessary eliminations were made. This elimination helped to remove some airfoils with very low and unrealistic C_l/C_d values.



Graph 3.1. The distribution of airfoils based on the Cl/Cd ratio

The graph shows the distribution of C_l/C_d values for airfoils at 10 degrees before any eliminations were made. In this distribution, C_l/C_d values are ordered from the lowest to the highest. As illustrated in the graph, airfoils with a C_l/C_d ratio below 40 were excluded from the database. This distribution was created for 10 degrees, but the database was also examined for other angles. The C_l/C_d distribution at Alpha 2, 0.3 Ma number, and 1e6 Re conditions is as follows.



Graph 3.2. The distribution of airfoils based on the Cl/Cd ratio for Alpha 2

The code that performs the described process is provided in the section 7.1.2.

3.2. Determination of Hyperparameters

3.2.1. Explanation of SVM Hyperparameters

In this section, the model and parameter definitions are carried out. This step involves creating the Support Vector Regression (SVR) model and determining the hyperparameter range using GridSearchCV. First, a structure named params is defined. This structure contains the model's hyperparameters and specifies the parameter ranges to be evaluated by GridSearchCV. The params structure has four keys: C, gamma, kernel, and epsilon.

The C parameter is known as the penalty parameter in SVR and controls the model's tolerance to errors. Small C values create a softer margin by allowing some errors, thereby improving the model's generalization ability. Large C values create a stricter margin, aiming for fewer errors on the training data, but this can increase the risk of overfitting. For the C parameter, a range such as [0.1, 1, 10, 100] is typically used.

The gamma parameter, when using the RBF (Radial Basis Function) kernel, determines the influence of a single training example. Small gamma values mean a larger influence area for each data point, resulting in a smoother decision boundary. Large gamma values mean a smaller

influence area, which can capture more details from the data but also increase the risk of overfitting. The common range for gamma is [0.001, 0.01, 0.1, 1, 10].

The kernel parameter specifies the kernel function to be used by the SVR model. In this example, only the 'rbf' (Radial Basis Function) kernel is chosen because the RBF kernel generally yields good results in SVR models.

The epsilon parameter specifies the margin of error in the model. It defines a tube around the predicted function within which errors are not penalized. In other words, if the difference between the predicted value and the actual value falls within this epsilon margin, the model ignores this error. Small epsilon values make the model more sensitive and consider even small differences between the predicted and actual values. Large epsilon values allow the model to be more general and ignore these small errors. The common range for epsilon is [0.1, 0.2, 0.5, 1.0].

After defining these parameters, the SVR model is created. The svr variable creates an SVR model object by calling the svm.SVR() function. This model object will be trained with GridSearchCV.

GridSearchCV is a tool used to optimize the hyperparameters of the model. This tool tries different model configurations using the specified parameter ranges and selects the configuration that performs the best. The use of GridSearchCV in this example is realized through the regr variable. regr is defined with the expression GridSearchCV(svr, params) and takes the SVR model and the 'params' structure.

Then, the model is trained with the expression regr.fit(X_tr, Y). The fit function performs the model training using the scaled independent variable matrix X_tr and the dependent variable series Y. This process allows GridSearchCV to try all possible combinations over the specified parameter ranges and select the parameters that provide the best performance.

```
params = {'C': [0.1, 1, 10], 'gamma': [0.01, 0.1, 1], 'kernel': ['rbf'], 'epsilon': [0.005]}

def tune_hyperparameters(X, Y, params):
    """Tune SVR hyperparameters using GridSearchCV."""
    svr = svm.SVR()
    regr = GridSearchCV(svr, params)
    regr.fit(X, Y)
    return regr.best_params_, regr.best_score_
```

Figure 3.14. Adjusting interval of the hyperparameters (see for code section 7.2)

3.2.2. Automatic Detection of Hyperparameters

After completing the training process, the best_params_ and best_score_ variables store the best parameters selected by GridSearchCV and the best score obtained with these parameters. best_params_ includes the best C, gamma, and epsilon values. best_score_ indicates the R2 score obtained by the model with the best parameters.

This step is critical for optimizing the model's performance and finding the best parameters. Hyperparameter optimization enhances the generalization ability of machine learning models and reduces the risk of overfitting. Especially in complex models like SVR, selecting the correct hyperparameters significantly affects the model's accuracy and reliability. Therefore, tools like GridSearchCV are widely used in machine learning projects and are indispensable for maximizing model performance. These processes ensure that the model will also be successful on future datasets and guarantee the overall success of the modelling process.

```
best_params, best_score = tune_hyperparameters(X_tr, Y, params)
```

Figure 3.15. Selection of hyperparameters (see for code section 7.2)

In this step, an iterative search process is carried out to improve the parameters. The reason for doing this is to automatically determine the optimal value for each hyperparameter instead of manually, saving time.

```python
def adjust_hyperparameters(params, best_params):
    """Adjust hyperparameters based on the best params obtained."""
    # Adjust C
    c_values = [best_params['C'] / 100, best_params['C'], best_params['C'] * 100]
    # Adjust gamma
    gamma_values = [best_params['gamma'] / 100, best_params['gamma'], best_params['gamma'] * 100]
    return {'C': c_values, 'gamma': gamma_values, 'kernel': ['rbf'], 'epsilon': [0.005]}
```

Figure 3.16. Changing interval of the hyperparameters according to result (see for code section 7.2)

First, the current values of the C and gamma parameters are assigned to the c_values and gamma_values variables, and the median index of these values is determined. The loop checks whether the best C and gamma parameter values match the median values in the current range. If these values match the median values, the loop terminates. Otherwise, the C and gamma values are updated. If the median C value is less than the best C value, the new C values array is updated as exponential multiples of the median C value; if greater, it is updated as exponential divisions of the median C value. Similarly, the same operations are applied to the gamma values to update the ranges. These updates redefine the parameter ranges more narrowly, and GridSearchCV is run again. GridSearchCV is restarted with the svr model object and the new params dictionary, and the model is retrained. This process allows the best_params_ and best_score_ variables to be updated and the best parameters to be determined more precisely. This iterative process continues to improve the model's performance and find the optimal parameter set. This process continues until the best C and gamma parameter values exactly match the median values, aiming to provide the best model performance.

The obtained results are written to separate Excel files for each angle value.

```python
    results.append([alpha, best_params['C'], best_params['gamma'], best_params['epsilon'], best_score])
    print(f"For data_alpha_{alpha}.csv: Best SVR with params: {best_params} and R2 score: {best_score:.4f}")

results_df = pd.DataFrame(results, columns=['alpha', 'C', 'gamma', 'epsilon', 'best_score'])
results_df.to_excel( excel_writer: "parameter_results_CD.xlsx", index=False)


t_end = process_time()

print('Elapsed time : ', t_end - t_start)
```

Figure 3.17. Writing best of hyperparameters to Excel (see for code section 7.2)

3.2.3. Table of Detected Hyperparameters and R^2 Scores

The optimization of hyperparameters was performed using the code described in section 3.2.2. The obtained hyperparameters were written into an Excel file. In section 3.3, during the execution of the program, these parameters were automatically read from the Excel file. The results are presented in Table 3.2 and Table 3.3. Although it does not affect the operation of the program, the duration value has been determined.

Table 3.2. Hyperparameter Values for Moment Coefficient

C_M Values					
alpha	**C**	**gamma**	**epsilon**	**Best Score(R^2)**	**total elapsed time**
1	0,1	0,1	0,005	0,982873	
2	0,1	0,1	0,005	0,982548	
3	0,1	0,1	0,005	0,982193	
4	0,1	0,1	0,005	0,981713	
5	0,1	0,1	0,005	0,981067	
6	0,1	0,1	0,005	0,979860	
7	0,1	0,1	0,005	0,978475	
8	0,1	0,1	0,005	0,976837	
9	0,1	0,01	0,005	0,982000	
10	0,1	0,1	0,005	0,969238	176,25 s

Table 3.3. Hyperparameter Values for Lift and Drag Coefficient

C_L & C_D Values					
alpha	**C**	**gamma**	**epsilon**	**Best Score(R^2)**	**total elapsed time**
1	1	0,01	0,005	0,998639	
2	1	0,01	0,005	0,998557	
3	1	0,01	0,005	0,998456	
4	1	0,01	0,005	0,998343	
5	1	0,01	0,005	0,998243	
6	1	0,01	0,005	0,998107	
7	1	0,01	0,005	0,998085	
8	1	0,01	0,005	0,997136	
9	1	0,1	0,005	0,985194	
10	1	0,1	0,005	0,956983	494,0625 s

Since the C_D values are found within a small range, hyperparameter optimization can lead to inaccurate results. In the program described in section 3.3, the C_D values have been adjusted using normalization techniques. Due to the satisfactory results obtained, it has been decided to treat the C_D hyperparameters in the same manner as the C_L values.

3.3. SVM Model

3.3.1. Separation of Training and Test Data

The purpose of making a train-test split in machine learning is to evaluate a model's performance and generalization ability. This division involves separating the data into training and testing sets to measure the model's performance on new and unseen data, thereby detecting overfitting situations. While the training set (usually 70-80% of the data) is used for the model to learn, the test set (typically 20-30%) is reserved for evaluating the model. The selection of these ratios may vary depending on the size of the dataset and the complexity of the model. This process enables the calculation of model performance metrics, hyperparameter tuning, and comparison of different models. The practice continued with 20% test data in this application.

3.3.2. Types of Errors and Their Rationale

Numerical errors are an inherent aspect of computational methods and numerical analysis. These errors arise from various sources, including approximations, round-off errors, truncation errors, and algorithmic limitations. Understanding and quantifying numerical errors is essential for assessing the accuracy, stability, and reliability of numerical computations. By analysing these errors, we can improve the design and implementation of numerical methods, ensuring that they produce results that are as close to the true values as possible. Numerical errors are a critical consideration in fields ranging from scientific computing to engineering simulations, where precise and reliable calculations are paramount. Some of the error types, which are used to analyse the program, are represented below.

Absolute Error

Absolute error is a measure of the difference between the true value and the approximate or measured value in numerical computations. It provides a straightforward way to quantify the magnitude of the error without considering the scale or size of the true value. Absolute error is expressed mathematically as:

$$\varepsilon_{absolute} = |test\ value - predicted\ value| \tag{3-16}$$

Where the $\varepsilon_{absolute}$ means Absolute Error. In the program, absolute error is used to define maximum and minimum errors. It is done with NumPy Library in Python Packages.

R^2 Score

R^2, also known as the coefficient of determination, is a statistical measure used to evaluate the performance of a regression model. It indicates how well the model's predictions approximate the actual data points. The R^2 score ranges from 0 to 1, where:

- An R^2 of 1 indicates that the regression model perfectly fits the data.
- An R^2 of 0 indicates that the model does not explain any of the variability in the data.

The formula for calculating R^2 is:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3-17}$$

y_i: Test value from the database.

$\hat{y}_i$: Predicted value from the program.

$\bar{y}$: Mean value of test values from the database.

n: Number of the data points.

Root-Mean-Squared Error

Root Mean Squared Error (RMSE) is a commonly used metric to evaluate the performance of a regression model. It measures the average of the squares of the errors, which are the differences between the actual values and the predicted values. MSE is always non-negative, and values closer to zero indicate a better fit of the model to the data. The formulation of RMSE is below. (26)

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3-18}$$

Low value of RMSE indicates that the predicted values are close to the actual values, suggesting a good fit of the model. And high values of RMSE indicates that the predicted values deviate significantly from the actual values, suggesting a poor fit of the model.

There are two main limitations for RMSE:

- Scale dependency: RMSE is sensitive to the scale of the data. Larger values in the data can lead to larger RMSE values, which can make interpretation difficult across different datasets.
- Outliers: Because errors are squared, outliers can disproportionately affect the RMSE, making it very high if there are large deviations.

The outliers are detected this way, and it shows the prediction capabilities of program.

Cross-Validated Score

Cross-validation is a powerful technique used to assess the performance and generalizability of a machine learning model. Instead of evaluating the model on a single train-test split, cross-validation involves dividing the dataset into multiple folds and performing training and evaluation multiple times. This helps to ensure that the model's performance is not dependent on a particular split of the data and provides a more reliable estimate of its generalization error. It is used again RMSE for each split of the data.

3.3.3. Ensemble Methods: Adaboost, Bagging, and Gradient Boosting

Ensemble methods are machine learning techniques that aim to create a stronger and more generalizable model by combining various learning algorithms. These methods aggregate the predictions of base learners, or weak learners, to make a collective prediction, achieving a performance boost that each individual learner may not achieve on its own. The advantage of ensemble methods lies in their ability to adapt to various data structures and patterns by combining different algorithms and/or different learning parameters of the same algorithm. Popular techniques within these methods include Adaboost, Bagging, and Gradient Boosting. Adaboost, bagging, and gradient boosting are ensemble methods used in classification and regression problems to enhance the performance of base learners, such as Support Vector Machines.

Adaboost aims to create a strong learner by combining weak learners. Although simple models are generally used as weak learners, SVM can also be employed in this role. Adaboost increases the weights of misclassified examples in each iteration, ensuring that the next weak learner focuses more on these examples. The final model is formed by taking the weighted sum of all weak learners, thereby enhancing the performance of SVM to yield a stronger classifier.

Bagging, or bootstrap aggregating, involves generating multiple different training sets through random sampling (with replacement) from the original dataset. Each bootstrap sample is used to train a separate SVM model. The predictions from these SVM models are then combined; in classification problems, majority voting is used, while in regression problems, the average is taken. Bagging leverages the diversity of the dataset to reduce the variance of SVM models and mitigate the risk of overfitting.

Gradient boosting aims to build a strong model by incrementally reducing errors. It starts with a simple model, which can be an SVM. The errors made by this model are calculated, and a new SVM model is trained to correct these errors. This process is iterative, with each new model being added to the previous one. Thus, SVM models are iteratively improved, resulting in more accurate predictions.

In summary, Adaboost combines weak SVM models by weighting them and focusing on correcting misclassifications, while bagging trains SVM models on different bootstrap samples and combines their predictions to reduce variance. Gradient boosting sequentially trains SVM models to correct errors, forming a robust model. Each of these methods effectively enhances the performance of SVM and improves the generalization capability of the model.

3.3.4. Explaining the Code Written for The SVM Model

Importing Hyperparameters

After importing the necessary libraries,

In the first step, one of the coefficients, C_L, C_D, or C_M, is selected, and the hyperparameters of the database are obtained from an Excel file. These hyperparameters are based on the results obtained from the initial script, 'script_0' (see in the section 3.2). The final step for this section is importing the database before starting the loop. Attributes identify the airfoils and are shown in Figure 3.18 below.

```
# Get the desired coefficient
coefficient = input("Please enter coefficient ('CM', 'CD' or 'CL'): ")

# Load the parameters
titles = ['alpha', 'C', 'gamma', 'epsilon']
data_params = pd.read_excel( io: f"hyperparameter/parameter_results_{coefficient}.xlsx", names=titles, usecols=[0, 1, 2, 3])

# Define data and features
titles_data = ['xtth', 'ytth', 'atte', 'acle', 'ycth', 'xcmc', 'ycmc', 'acte', 'CL', 'CD', 'CM']
attributes = ['xtth', 'ytth', 'atte', 'acle', 'ycth', 'xcmc', 'ycmc', 'acte']
```

Figure 3.18 Selection of Coefficient and Importing Hyperparameters and Database (see for code section 7.3)

Applications C_L, C_M and Normalized C_D

The for loop is constructed for each angle value where is between 1 and 10. There are two different sections because C_D coefficient uses normalization different from other two coefficients. They are separated with if construction from each other, first if statement is processing the C_D if the selection is the drag coefficient. The second if statement process the other two coefficients, C_L and C_M The time is initialized after the data is read from file. The Figure 3.19, Figure 3.20 and Figure 3.21 below show each process in the script.

```
# Loop over alpha values
for alpha in [round(x, 1) for x in list(np.arange(1.0, 10.1, 1))]:
    file_path = f"database/data_alpha_{alpha}.csv"
    data = pd.read_csv(file_path, delimiter=',', names=titles_data)
```

Figure 3.19 The loop used for alpha values (see for code section 7.3)

```
if coefficient == "CD":

    CD = data['CD']
    min_value = CD.min()
    max_value = CD.max()
    dif = max_value - min_value

    1 usage
    def normalization(a):
        return (a - min_value) / dif

    CDN = [normalization(a) for a in CD]
    data['CDN'] = CDN

    train_set, test_set = train_test_split( *arrays: data, test_size=0.2, random_state=42)

    X = train_set[attributes]
    y_train = train_set["CDN"]

    x = test_set[attributes]
    y_te = test_set["CDN"]

    scaler = preprocessing.StandardScaler().fit(X)
    X_tr = scaler.transform(X)
    x_tr = scaler.transform(x)

    t_start = process_time()

    # Find parameters and train the model
    target_row = data_params[data_params['alpha'] == alpha]
```

Figure 3.20. if Statement for the Drag Coefficient (see for code section 7.3)

```
if not target_row.empty:
    C_value = target_row['C'].values[0]
    gamma_value = target_row['gamma'].values[0]
    epsilon_value = target_row['epsilon'].values[0]

    regr = svm.SVR(kernel="rbf", C=C_value, gamma=gamma_value, epsilon=epsilon_value)

    cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
    scores = cross_val_score(regr, X_tr, y_train, cv=cv, scoring='r2')

    print(f"Alpha: {alpha}")
    print("Cross-validated R2 scores:", scores)

    regr.fit(X_tr, y_train)
    y_pr = regr.predict(x_tr)
```

Figure 3.21. Revalued Missing Hyperparameters from Script (see for code section 7.3)

For other C_L and C_M coefficients, the normalization process is not needed so the data just split, and the program is run directly. The whole process is the same as C_D coefficients. The errors and plot are recorded to a project file. Figure 3.22 shows the difference of two coefficients from drag coefficient.

```
else:
    train_set, test_set = train_test_split( *arrays: data, test_size=0.2, random_state=42)

    X = train_set[attributes]
    y_train = train_set[coefficient]

    x = test_set[attributes]
    y_test = test_set[coefficient]

    scaler = preprocessing.StandardScaler().fit(X)
    X_tr = scaler.transform(X)
    x_tr = scaler.transform(x)

    t_start = process_time()

    # Find parameters and train the model
    target_row = data_params[data_params['alpha'] == alpha]
```

Figure 3.22 'Lift and Moment Coefficients' Process in Split Section from Script (see for code section 7.3)

Plotting Datas, Calculating Types of Errors, Process Time

The normalized C_D values are revalued with inverse of first normalization function and calculation of the errors are defined this part; time of the process is calculated here also in Figure 3.23 below.

```
2 usages
def original(b):
    return b * dif + min_value

y_pred = [original(b) for b in y_pr]
y_test = [original(b) for b in y_te]

# Convert y_test and y_pred to numpy arrays
y_test = np.array(y_test)
y_pred = np.array(y_pred)

r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
max_err = max_error(y_test, y_pred)
min_err = np.min(np.abs(y_test - y_pred))

t_end = process_time()
computation_time = t_end - t_start
```

Figure 3.23. Inverse of Normalization Function and Calculations of Error from Script (see for code section 7.3)

The calculated errors and output graph of program is shown at screen while program is running with the script in Figure 3.24 below. There is a code also to save results in an Excel file below.

```
    print(f"R2 score: {r2}")
    print(f"RMSE: {rmse}")
    print(f"Max Error: {max_err}")
    print(f"Min Error: {min_err}")
    print(f"Elapsed time: {computation_time:.2f} seconds\n")

    plt.figure()
    plt.plot( *args: y_test, y_pred, 'bo', y_test, y_test)
    plt.title(f"Alpha: {alpha}")
    plt.xlabel(f"Actual {coefficient}")
    plt.ylabel(f"Predicted {coefficient}")
    plt.grid(True)
    plt.savefig(f"outputs/outputs_results/outputs_{coefficient}/figure_alpha_{coefficient}_{alpha}.png")
    plt.close()

    # Save metrics to Excel
    metrics_df = pd.DataFrame(
        {'R2 Score': [r2], 'RMSE': [rmse], 'Max Error': [max_err], 'Min Error': [min_err]})
    metrics_df.to_excel( excel_writer: f"outputs/metrics/outputs_metrics_{coefficient}/metrics_{coefficient}_{alpha}.xlsx",
                        index=False)

    # Save prediction results to Excel
    result_df = pd.DataFrame(
        {'Data_Num': test_set.index, f'{coefficient}_test': y_test, f'{coefficient}_pred': y_pred})
    result_df.to_excel(
         excel_writer: f"outputs/outputs_results/outputs_{coefficient}/output_results_{coefficient}_{alpha}.xlsx", index=False)

else:
    print(f"Parameters not found for alpha value: {alpha}")
```

Figure 3.24. Plotting Figures and Writing the Errors at Screen from Script (see for code section 7.3)

And finally, Figure 3.25 shows the calculating of total process time.

```
# Record the program end time and calculate the total runtime
program_end_time = process_time()
total_computation_time = program_end_time - program_start_time

print(f'Total elapsed time for the entire program: {total_computation_time:.2f} seconds')
```

Figure 3.25. Calculation and Saving the Total Time of Program (see for code section 7.3)

Why Did We Need Normalization for Drag Coefficient?

Normalization is a crucial preprocessing step in machine learning, aiming to adjust the data to a standard scale without distorting differences in the ranges of values. It helps to improve the performance and training stability of machine learning models. This report will cover the concept of normalization, its importance, methods, and practical application in machine learning projects.

Normalization refers to the process of adjusting values measured on different scales to a common scale. In the context of machine learning, this often means scaling features so that they contribute equally to the model's learning process. Without normalization, features with larger ranges can dominate the model's learning process, leading to biased results. Some essential purposes of using normalization in data is explained below.

- Improved Convergence Speed: Gradient-based optimization algorithms (e.g., gradient descent) converge faster on normalized data and time is the one of the parameters while processing the algorithm’s performance.
- Reduced Bias: It prevents features with larger scales from disproportionately influencing the model.
- Enhanced Model Performance: Models like k-nearest neighbors (KNN) and SVM are sensitive to the scale of data. Normalized data ensures better performance.
- Avoiding Numerical Instability: Normalizing features can prevent numerical issues during computation, ensuring stability and reliability

There are many types of normalization methods but in this project the min-max normalization method is used. Min-max normalization scales the data to fixed range, typically [0-1] or [-1-1]. The data of the project is scaled between 0 and 1 to get better outputs from algorithm. The formulation of Min-Max Normalization is below.

$$X' = \frac{X - X_{minimum}}{X_{maximum} - X_{minimum}} \tag{3-19}$$

Where, X is the original value, $X_{minimum}$ is the minimum value in the database for C_D, $X_{maximum}$ is the maximum value of c_d in database and X' is the normalized value of C_D coefficient. (27)

After the normalization of the C_D coefficient in the database, the results are better than the older program.

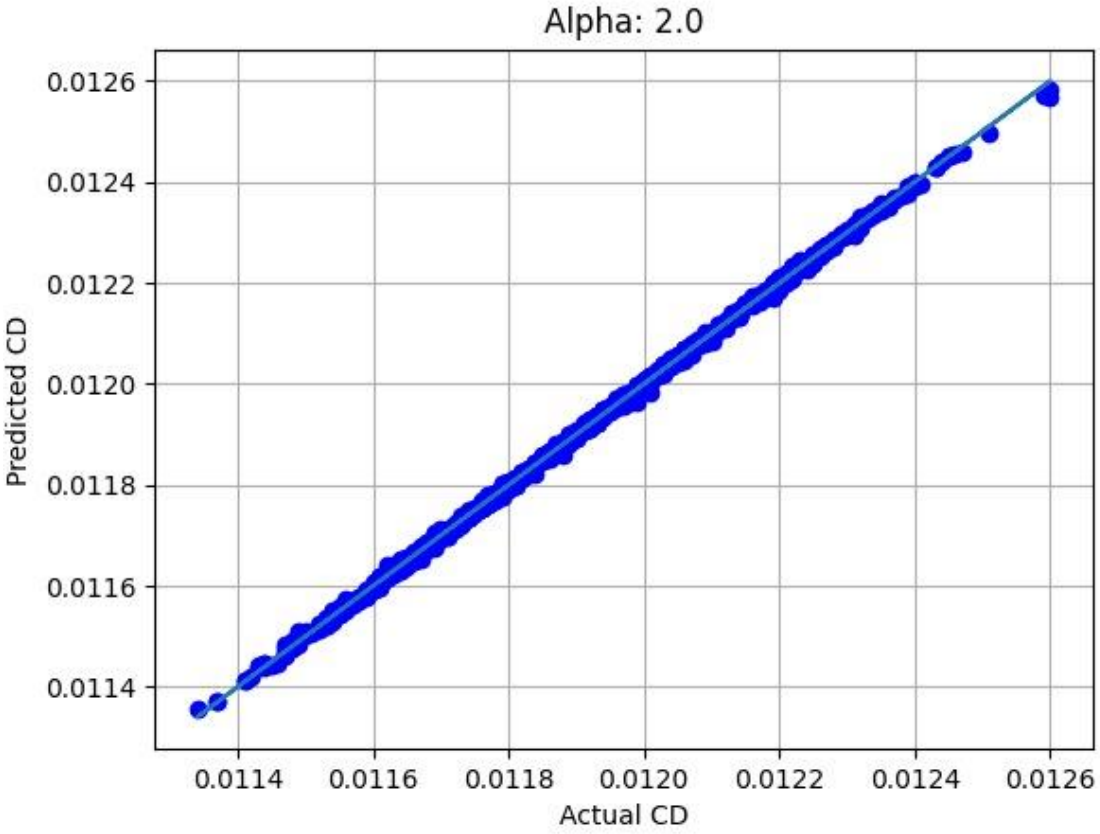


Figure 3.26. Test and Prediction Graph

4. RESULTS AND DISCUSSION

As a result, the project was run completely and analyzed with details. The predicted values matching test data with less than one per cent of error. When the results were examined, the error values increased rapidly at the high angle of attack values as 9 and 10 degrees. It is sourced from corruption of linear graph. Figure 4.1, Figure 4.2 and Figure 4.3 shows the outputs of the lift coefficient and it seems that the prediction quality is decreasing while angle of attack is going high levels. When the airfoils graph for one of the coefficients, corruption of linearity can be seemed clearly.

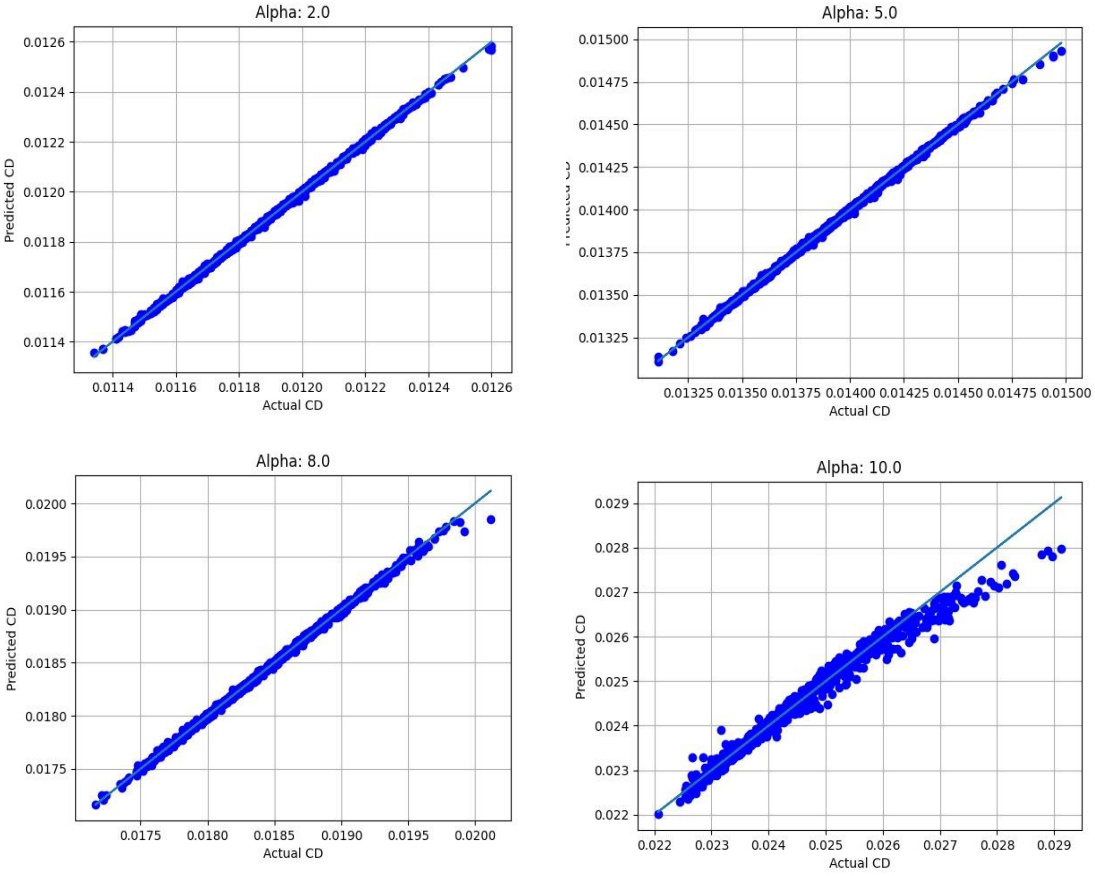


Figure 4.1 Drag Coefficient (C_D) – Angle of Attack (α) Graph

Predicted and actual values for Drag Coefficient at 2, 5, 8 and 10 degrees were obtained and graphed, respectively. When the graphs are examined, the values obtained by SVM at 2, 5 and 8 degrees overlap with the XFOIL data. However, at 10 degrees, it was observed that there was more deviation in the predicted and actual values due to the AoA value being very high.

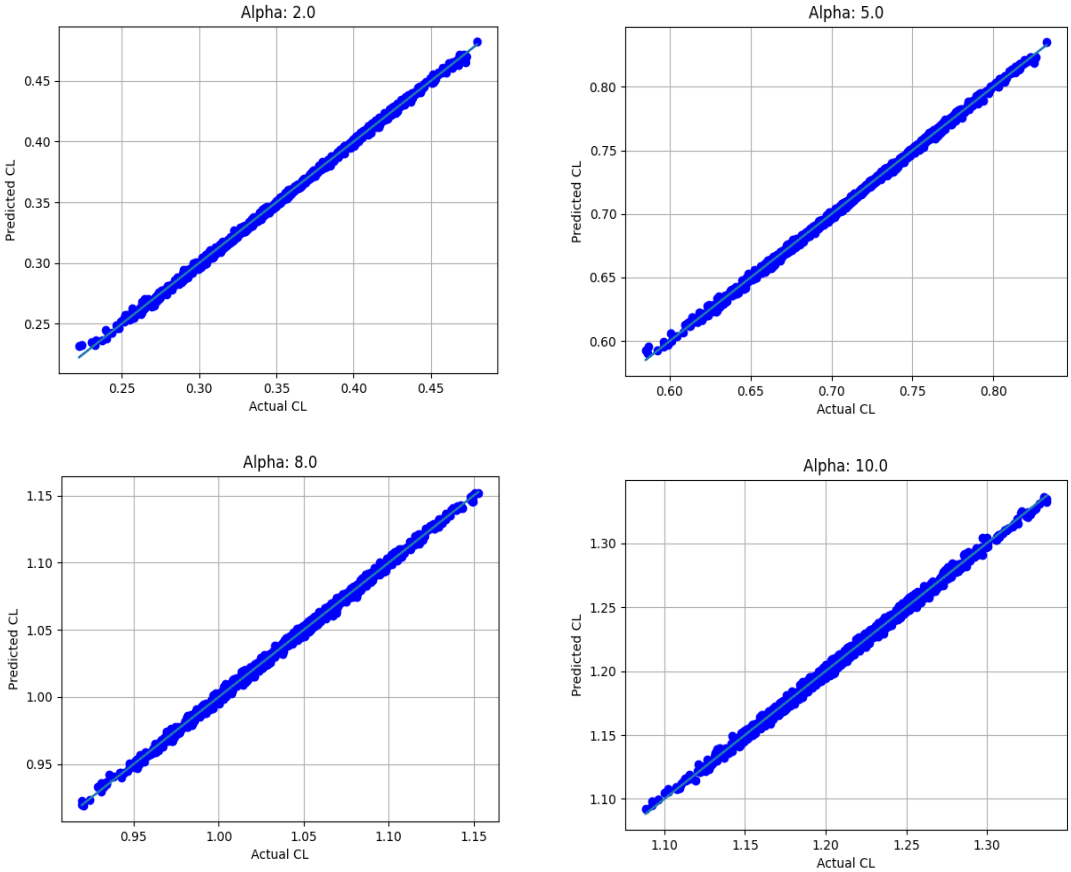


Figure 4.2 Lift Coefficient (C_L) – Angle of Attack (α) Graph

Lift Coefficient was also examined with the same method. As in Drag Coefficient, SVM and XFOIL data are consistent with each other at 2, 5 and 8 degrees in Lift. At 10 degrees, unlike Drag Coefficient, the values overlap with each other, as seen in the graph.

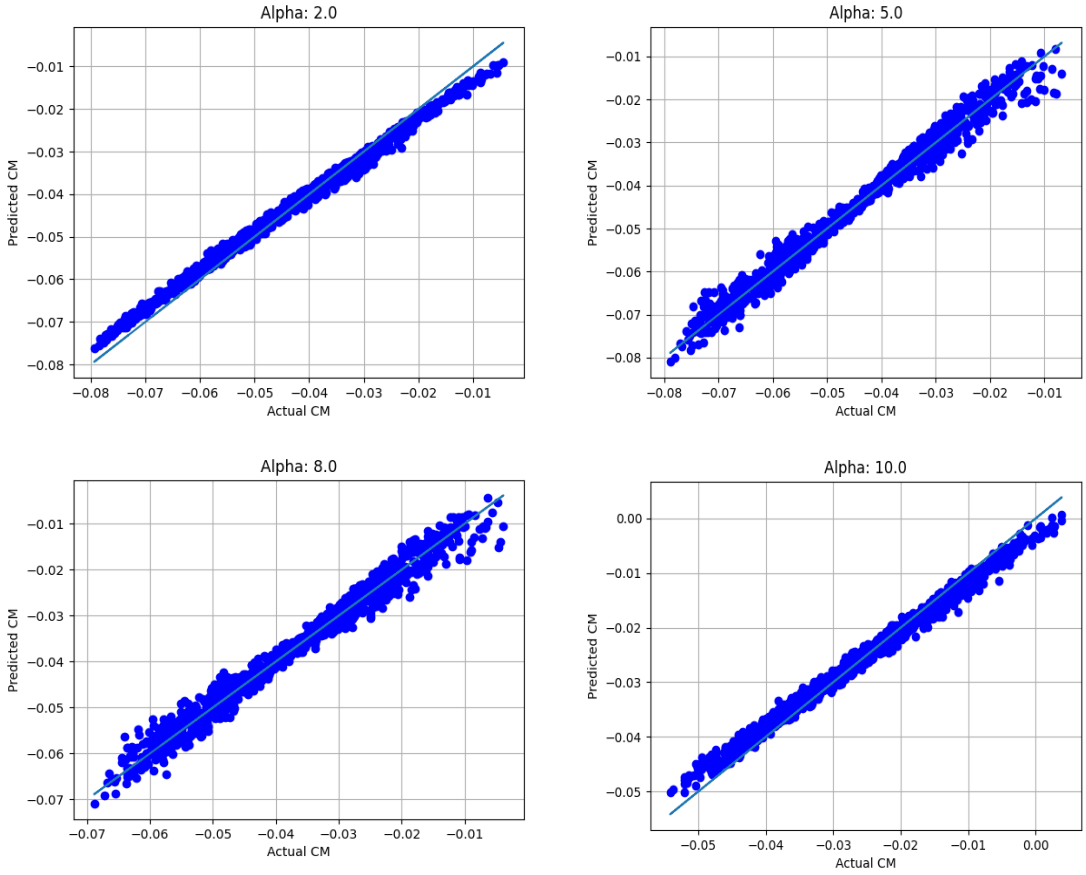
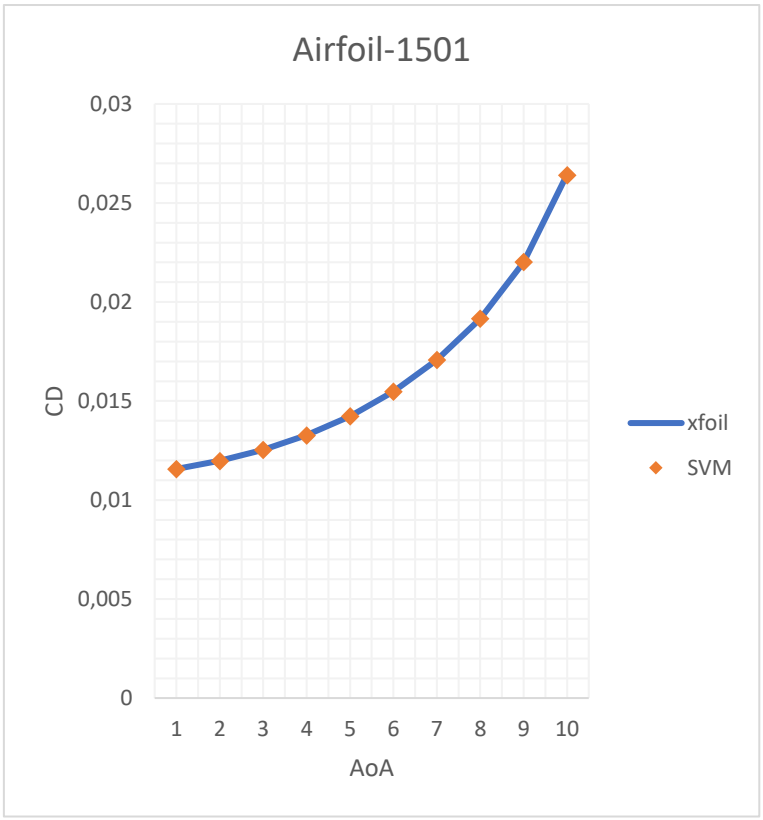


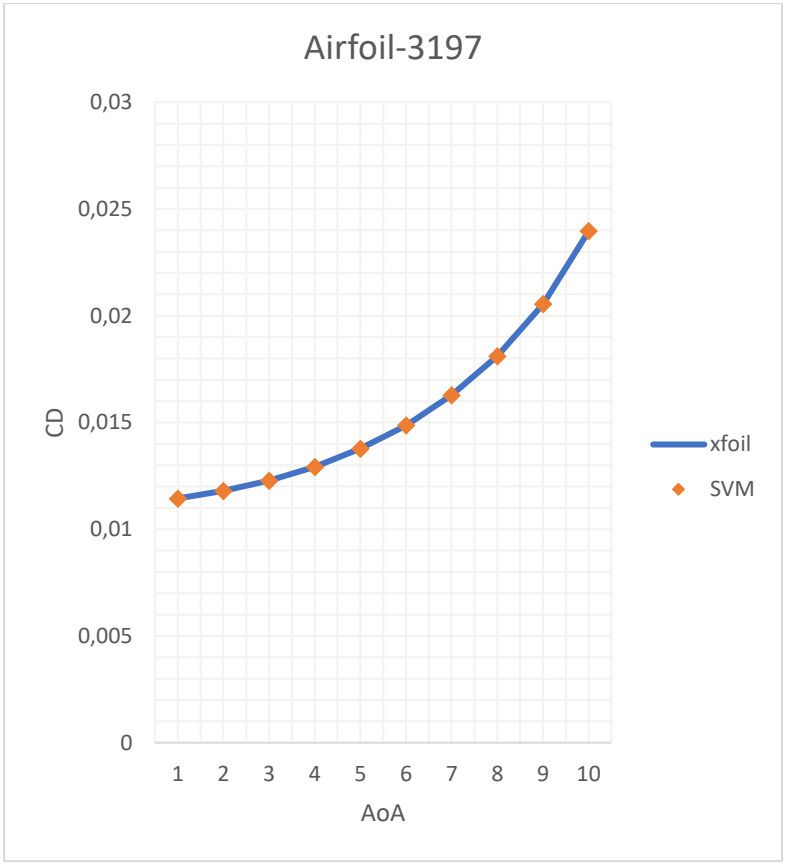
Figure 4.3. Moment Coefficient (C_M) – Angle of Attack (α) Graph

The coefficient of moment has been examined. Unlike the other two coefficients, the SVM and XFOIL data do not match significantly at any angle value.

To examine all coefficient values, a good airfoil and an airfoil approximately in the middle when error values were ranked from low to high were selected.

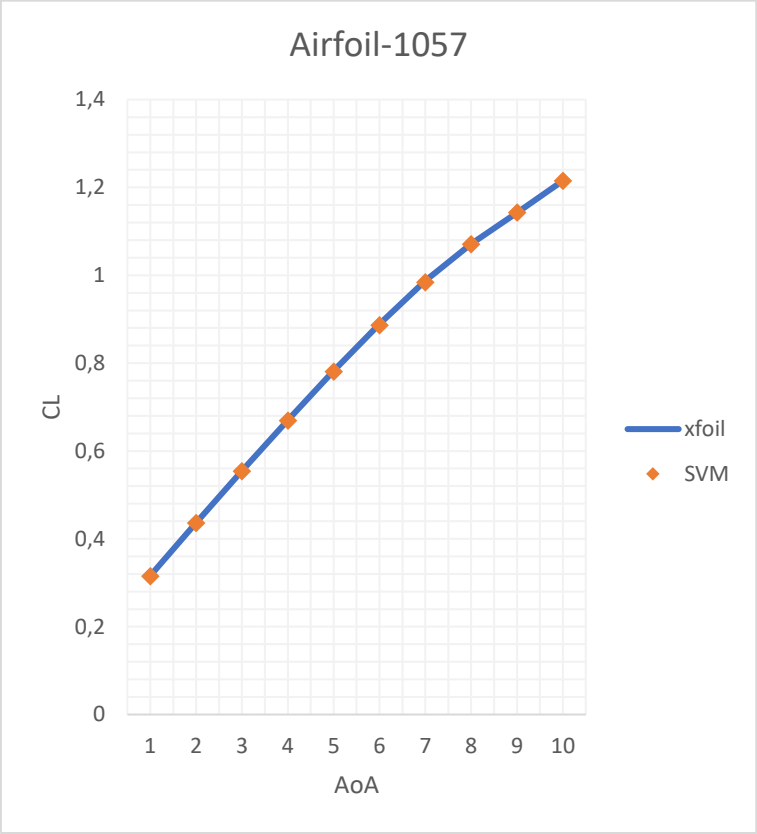


Graph 4.1. C_D-AoA Graph for Airfoil-1501

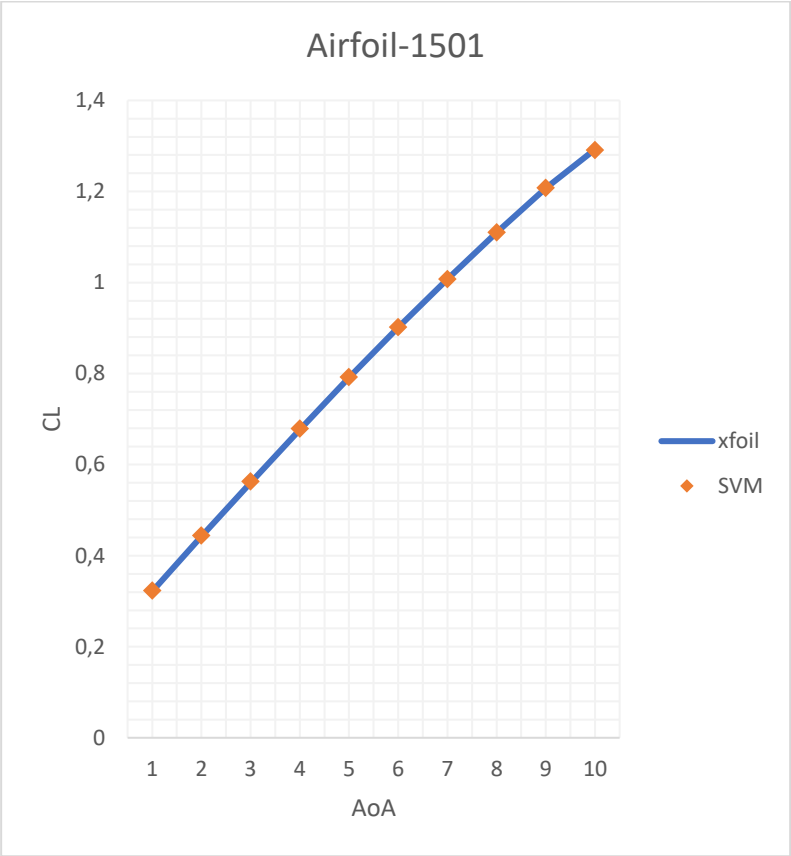


Graph 4.2. C_D-AoA Graph for Airfoil-3197

Airfoils 1501 and 3197 were selected to examine the Drag Coefficient value. When the graphs were examined, it was observed that each angle value of the SVM and XFOIL data was highly consistent.

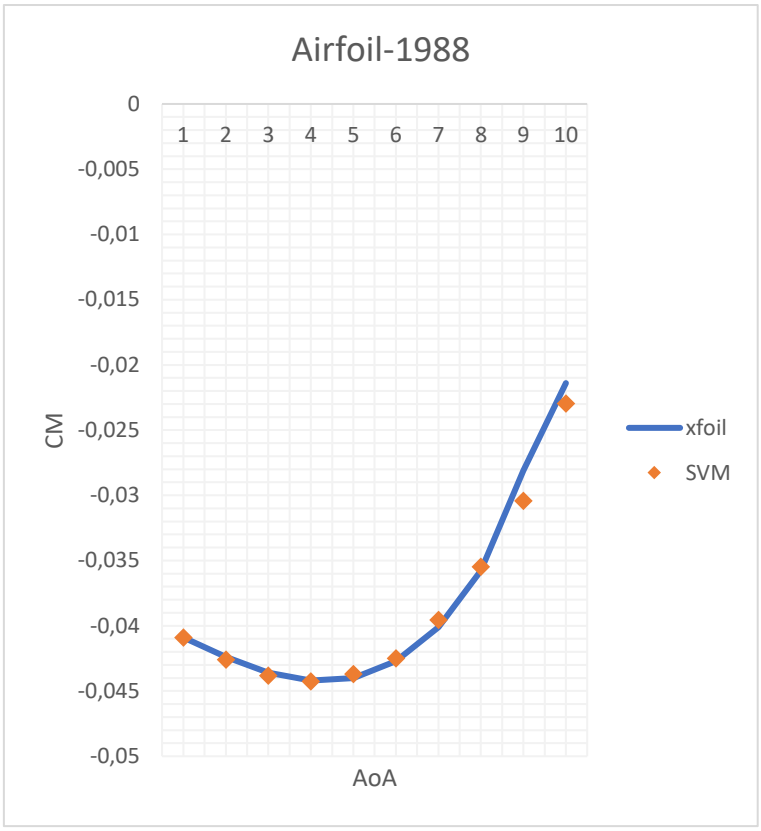


Graph 4.3. C_L-AoA Graph for Airfoil-1057

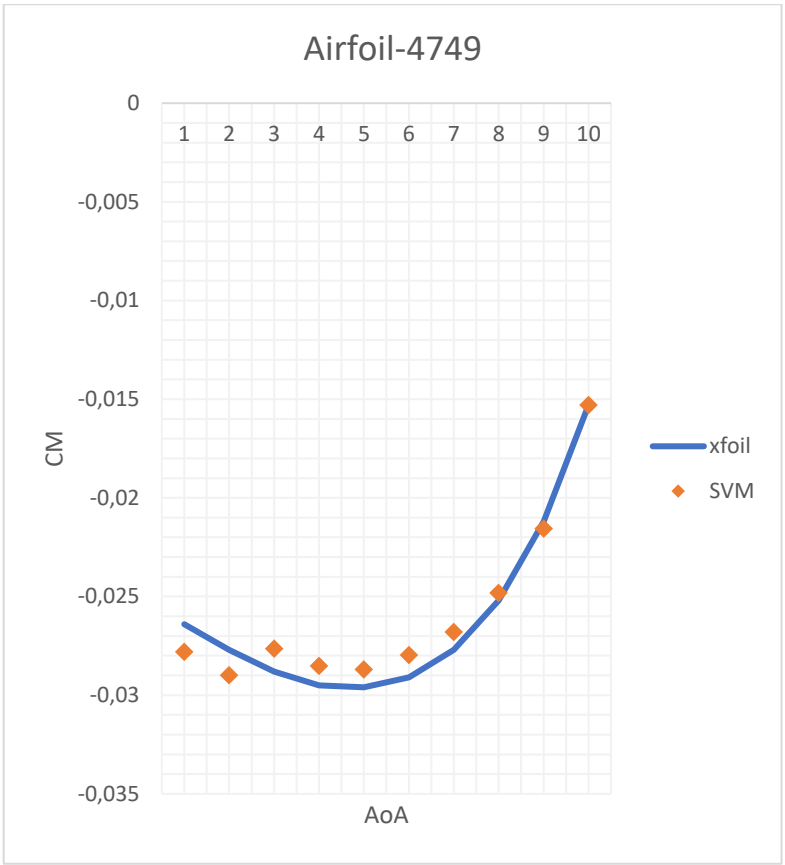


Graph 4.4. C_L-AoA Graph for Airfoil-1501

To examine the Lift Coefficient value, 1057 and 1501 airfoils were selected. When the graphs are examined, it is observed that each angle value of SVM and XFOIL data is highly consistent, as is the Drag Coefficient value.



Graph 4.5. C_M-AoA Graph for Airfoil-1988



Graph 4.6. C_M-AoA Graph for Airfoil-4749

Finally, 1988 and 4749 airfoils were selected to examine the Moment Coefficient value. As can be seen from the moment graphs, different consistent ratios were obtained from the Drag and Lift Coefficient graphs. For Moment Coefficient, the differences between the error values obtained as a result of SVM and XFOIL data are small. However, since the data was very small, the graphs were obtained this way due to scaling while transferring them to the source graph.

To get better results, the program is run with different algorithms, Adaboost, Bagging, and Gradient Boosting. The program has already errors less than the one per cent for test value and the average of the errors under the fifteen per thousand. When the results were compared to reference article, lift and drag coefficients have better error values, unlike the moment coefficients. The moment coefficient error values are so close to the reference article, but it is not caught with any method that in used project. The drag coefficient error values are less than the reference article. The normalization affects the results directly, expansion of range and clearance of the data to analyze. The lift coefficient gave already less value of error, when the program was run at the first time. All values of error are improved with the calculating of hyperparameters with a different code and it is a reason for decreasing the values of error. Table 4.1 shows the used algorithms, their error and time scores for overall program and the Table 4.2 shows the overall error scores from reference paper to comparison.

Table 4.1. The Overall Results for Comparison

Overall Error	Algorithms	R2 Score	RMSE	Max. Error	Min Error	Time (s)
C_L	Normal	0.998097	**0.002177417**	0.008644	4.405E-06	**5.78**
	Adaboost	0.998078	**0.002191821**	0.008758	5.6E-06	**16.11**
	Bagging	0.998052	**0.002204799**	0.009294	4.288E-06	**40.48**
	Gradient Boosting	0.998933	**0.001615929**	0.007571	2.459E-06	**19.62**
C_M	Normal	0.980031	**0.002194545**	0.008778	2.966E-06	**3.52**
	Adaboost	0.978911	**0.002255739**	0.009068	2.518E-06	**9.41**
	Bagging	0.978739	**0.002266627**	0.009554	1.53E-06	**19.94**
	Gradient Boosting	0.992125	**0.001280408**	0.00593	9.973E-07	**18.36**
C_D	Normal	0.995754	**3.30539E-05**	0.000219	1.457E-08	**25.27**
	Adaboost	0.996708	**2.9231E-05**	0.000156	3.264E-08	**204.58**
	Bagging	0.995717	**3.32071E-05**	0.00022	6.34E-08	**377.77**
	Gradient Boosting	0.996511	**3.00811E-05**	0.000171	1.179E-08	**56.58**

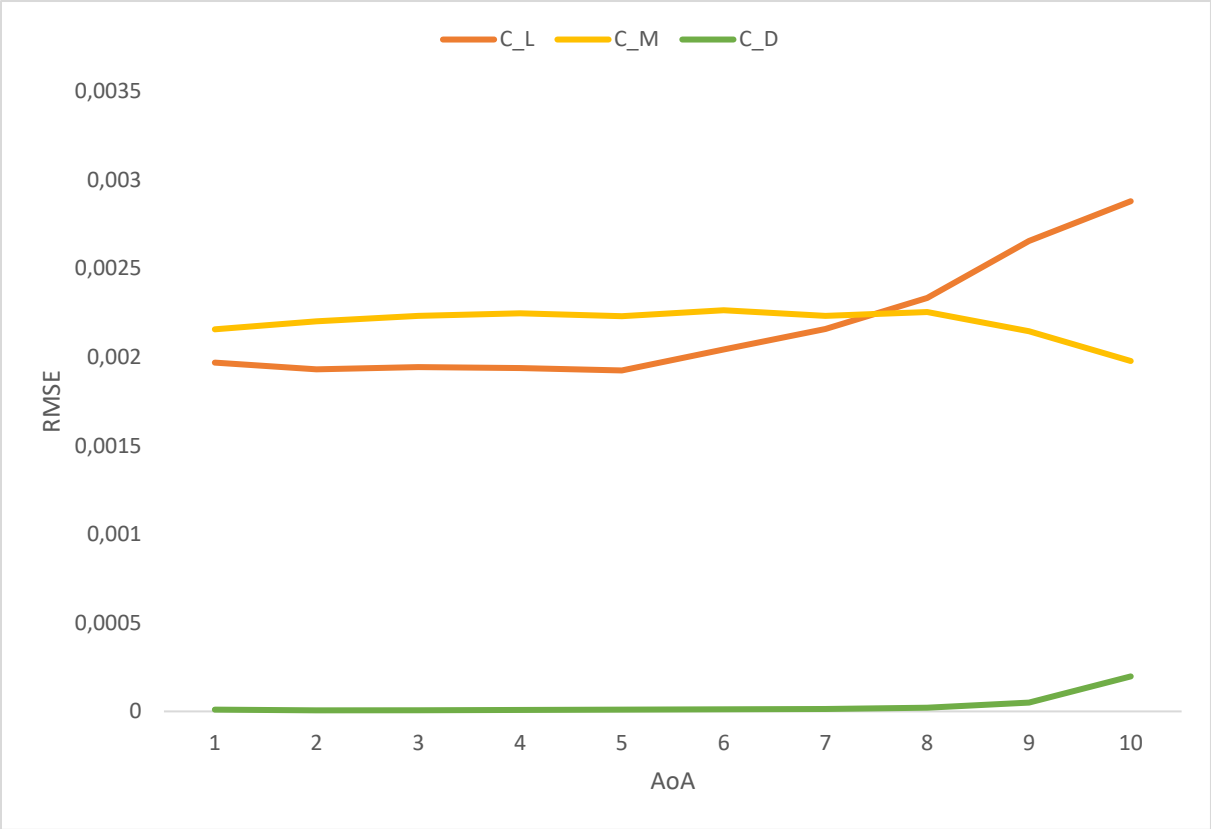
Table 4.2. Table 3 from Reference Paper **(1)**

	RMSE	Maximum Error
C_L	0.00475	0.20473
C_M	0.00055	0.03380
C_D	0.00099	0.00756

Table 4.3. Outputs of RMSE

RMSE			
alpha	**C_D**	**C_L**	**C_M**
1	9,44E-06	0,001969	0,002156
2	6,01E-06	0,00193	0,002202
3	6,29E-06	0,001944	0,002233
4	7,55E-06	0,001938	0,002248
5	9,17E-06	0,001925	0,002231
6	1,12E-05	0,002042	0,002265
7	1,31E-05	0,002159	0,002233
8	2,12E-05	0,002333	0,002255
9	4,9E-05	0,002655	0,002146
10	0,000198	0,00288	0,001978
overall	**3,30539E-05**	**0,002177417**	**0,002194545**

The RMSE values obtained for the drag coefficient (C_D), lift coefficient (C_L), and moment coefficient (C_M) across various angles of attack (alpha) through hyperparameter optimization with the SVR model demonstrate the model's predictive accuracy. The overall RMSE values for C_D, C_L, and C_M were found to be 3,30539E-05, 0,002177417, and 0,002194545, respectively. These values indicate that the model makes predictions with high precision. These results reflect the effectiveness of the iterative optimization process and the capability of the SVR model to accurately predict aerodynamic coefficients. The model's ability to consistently produce low RMSE values across different angles of attack validates its robustness and reliability in aerodynamic performance evaluations.



Graph 4.7. AoA to RMSE Value for each Drag Coefficient, Lift Coefficient and Moment Coefficient Graph

5. CONCLUSION

This project has provided innovative solutions for accurately predicting aerodynamic coefficients using Support Vector Regression with optimized hyperparameters. The GridSearchCV method was systematically used to optimize the performance of the SVR model. The application of GridSearchCV enabled the determination of the best values for the C, gamma, and epsilon parameters, leading to significant improvements in the predictions of lift, drag, and moment coefficients.

The optimized SVR model demonstrated high accuracy and robustness, as evidenced by superior R^2 scores and low RMSE values. These results underscore the effectiveness of proper hyperparameter tuning in enhancing prediction performance. The integration of GNU OCTAVE for database creation and preprocessing, along with Python for hyperparameter tuning and model training, offered a comprehensive and efficient approach to model optimization.

For practical applications, the findings of this project suggest that the optimized SVR model can be used to achieve faster and more accurate aerodynamic predictions, which are critical for engineering and aerodynamic studies. Future evaluations could focus on further improving the model and exploring its applicability to different aerodynamic shapes and conditions. To enhance the model further, various artificial intelligence methods or libraries could be employed. Additionally, an open-source CFD program other than XFOIL could be used, and all these results could be compared.

In summary, this project not only addresses the issue of accurately predicting aerodynamic coefficients but also pioneers the use of hyperparameter optimization in similar prediction modeling tasks. The innovative methodologies and tools used here pave the way for advancements in aerodynamic studies and related engineering fields.

6. REFERENCES

1. Andrés, E., Salcedo-Sanz, S., Monge, F., & Pérez-Bellido, A. M. (2012). Efficient Aerodynamic Design Through Evolutionary Programming and Support Vector Regression Algorithms. Expert Systems with Applications, 39, 10700-10708.
2. Sobieczky, H. (1999). Parametric Airfoils and Wings. In Recent Development of Aerodynamic Design Methodologies (pp. 71-88).
3. Abbott, I. H., & Von Doenhoff, A. E. (1959). Theory of Wing Sections: Including a Summary of Airfoil Data. Dover Publications.
4. Kulfan, B. M. (2007). A Universal Parametric Geometry Representation Method - 'CST' (Class Shape Transformation) Method. AIAA Paper 2007-62.
5. Farin, G. (2001). Curves and Surfaces for CAGD: A Practical Guide. Morgan Kaufmann.
6. Sobieczky, H. (1999). Parametric Airfoils and Wings. In Notes on Numerical Fluid Mechanics, 68. Springer.
7. Jameson, A. (1988). Aerodynamic Design via Control Theory. Journal of Scientific Computing, 3(3), 233-260.
8. Jasak, H., et al. (2007). OpenFOAM: A C++ Library for Complex Physics Simulations. International Workshop on Coupled Methods in Numerical Dynamics.
9. Economon, T. D., Palacios, F., Copeland, S. R., Lukaczyk, T. W., & Alonso, J. J. (2018). SU2: An open-source suite for multiphysics simulation and design. AIAA Journal, 56(1), 1-16.
10. Logg, A., et al. (2012). Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book. Springer.
11. Drela, M. (1989). XFOIL: An Analysis and Design System for Low Reynolds Number Airfoils. In T.J. Mueller (Ed.), Low Reynolds Number Aerodynamics (Lecture Notes in Engineering, Vol. 54).
12. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
13. Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. MIT Press.

14. Zhu, X., & Goldberg, A. B. (2009). Introduction to Semi-Supervised Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, 3(1), 1-130.

15. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.

16. Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. Machine Learning, 20(3), 273-297.

17. Cristianini, N., & Shawe-Taylor, J. (2000). An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods. Cambridge University Press.

18. Joachims, T. (1998). Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In Proceedings of the 10th European Conference on Machine Learning (ECML-98) (pp. 137-142).

19. Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A Library for Support Vector Machines. ACM Transactions on Intelligent Systems and Technology (TIST), 2(3), 1-27.

20. Schölkopf, B., & Smola, A. J. (2002). Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press.

21. Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

22. Abadi, M., et al. (2016). TensorFlow: A System for Large-Scale Machine Learning. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16).

23. Li, R. S. P. (1998). Manual Aerodynamic Optimization of an Oblique Flying Wing. AIAA, 98-0598, 98-108.

24. The Linux Foundation. (n.d.). Retrieved from https://www.linuxfoundation.org/.

25. GNU Octave. (n.d.). Retrieved from https://octave.org/.

26. Scikit-learn. (n.d.). Retrieved from https://scikit-learn.org/stable/.

27. Han, J., & Kamber, M. (2012). Data Mining Concepts and Techniques. Elsevier.

7. APPENDIX

7.1. Airfoil Parameterization and Database Creation

7.1.1. Database Creation

Database Creation

GNU OCTAVE

```
clc;
clear;
system('rm airfoil*');
file1 = fopen('parameters.dat','wt');
fprintf(file1, '  m      xtth       ytth       atte       acle       ycth
xcmc       ycmc       acte  \n');
for m=1:6000
  xtle = 0.015;                                % X Position for leading edge
control point (.015)
  ytle = 0.036;                                % Y Position for leading edge
control point (.036)
  xtth = .337 + (rand * .075);                 % X Position for maximum
thicknes (.300<xtth<.450)
  ytth = .117 + (rand * .035);                 % Maximum thickness
(.100<ytth<<170)
  atte = tand(-8.625 + (rand * 2.75));      % Trailing edge thickness line
angle (-10.000<atte<-4.500)
  ytte = .006;                                 % Trailing edge thickness
(.006)
  acle = tand(-4.375 + (rand * 3.125));        % Leading edge camber line
angle (-7.500<acle<5.000)
  ycth = -.004 + (rand * .008);                % Camber ar maximum thickness
(-.008<ycth<.005)
  xcmc = .725 + (rand * .05);                % X Position for maximum camber
(.700<xcmc<.800)
  ycmc = -.005 + (rand * .017);                % Maximum camber (-
.010<ycmc<.020)
  acte = tand(-11.250 + (rand * 7.50));     % Trailing edge camber line
angle (-15<acte<0)
  ycte = 0.000;                                % Trailing edge camber (0.000)
  params = [xtth ytth atte acle ycth xcmc ycmc acte];
  fprintf(file1, '%4d %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f
%10.5f\n',m, params');
  eqn_5 = [(1/(2*(xtth)^.5)), (1), (2*xtth), (3*xtth^2), (4*xtth^3)];
% Equation 5 from the article
  eqn_6 = [((xtth)^.5), (xtth), (xtth^2), (xtth^3), (xtth^4)];
% Equation 6 from the article
  eqn_7 = [(1/2), (1), (2), (3), (4)];
% Equation 7 from the article
  eqn_8 = [(1) (1) (1) (1) (1)];
% Equation 8 from the article
  eqn_9 = [((xtle)^.5), (xtle), (xtle^2), (xtle^3), (xtle^4)];
% Equation 9 from the article
  eqn_10 = zeros(1, 6);
% Equation 10 from the article
  eqn_10(1, 1) = 1;
% Defining of first member of first line as acle
  eqn_11 = [(xtth), (xtth^2), (xtth^3), (xtth^4), (xtth^5), (xtth^6)];
% Equation 11 from the article
  eqn_12 = [(1), (2*xcmc), (3*xcmc^2), (4*xcmc^3), (5*xcmc^4), (6*xcmc^5)];
```

```
% Equation 12 from the article
  eqn_13 = [(1), (2), (3), (4), (5), (6)];
% Equation 13 from the article
  eqn_14 = [(1), (1), (1), (1), (1), (1)];
% Equation 14 from the article
  eqn_15 = [(xcmc), (xcmc^2), (xcmc^3), (xcmc^4), (xcmc^5), (xcmc^6)];
% We write one more equation from article to define 6x6 matrix
  A = [eqn_5; eqn_6; eqn_7; eqn_8; eqn_9];
  A_inverse = inv(A);
  B = [eqn_10; eqn_11; eqn_12; eqn_13; eqn_14; eqn_15];
  B_inverse = inv(B);
  coefficients_a = [0; ytth; atte; ytte; ytle];
  as = A_inverse * coefficients_a;
  coefficients_b = [acle; ycth; 0; acte; ycte; ycmc];
  bs = B_inverse * coefficients_b;
  a_1 = as(1);
  a_2 = as(2);
  a_3 = as(3);
  a_4 = as(4);
  a_5 = as(5);
  b_1 = bs(1);
  b_2 = bs(2);
  b_3 = bs(3);
  b_4 = bs(4);
  b_5 = bs(5);
  b_6 = bs(6);
  x = 0:.01:1;                          % X coordinate with step size of
.01
  y_t = zeros(1, length(x));             % Thickness function
  y_c = zeros(1, length(x));             % Camber function
  y_u = zeros(1, length(x));             % Upper surface function
  y_l = zeros(1, length(x));             % Lower surface function
  i = 1;
  while i <= length(x)
    y_t(i) =
(a_1*((x(i)^.5)))+(a_2*x(i))+(a_3*x(i)^2)+(a_4*x(i)^3)+(a_5*x(i)^4);
% Calculation of thickness values each x values
    y_c(i) =
(b_1*x(i))+(b_2*x(i)^2)+(b_3*x(i)^3)+(b_4*x(i)^4)+(b_5*x(i)^5)+(b_6*x(i)^6)
;   % Calculation of thickness values each x values
    y_u(i) = y_c(i) + 0.5*y_t(i);
% Calculation of upper surface
    y_l(i) = y_c(i) - 0.5*y_t(i);
% Calculation of lower surface
    i = i + 1;
  end
  %Writing airfoil coordinates to a text file with four decimal points
  x_combined = [1:-0.01:0, 0.01:0.01:1];
  y_combined = [fliplr(y_u), (y_l(2:end))];
  file_name_txt = sprintf('airfoil%d.DAT', m);
  %Open the file for writing
  fid = fopen(file_name_txt, 'w');
  %Write the airfoil identifier
  fprintf(fid, 'airfoil_%d\n', m);
  %Write the coordinates with four decimal points
  fprintf(fid, '%8.5f  %8.5f\n', [x_combined; y_combined]);
  %Close the file
  fclose(fid);
  %dlmwrite(file_name_txt, [x_combined; y_combined]', 'delimiter', '\t',
'precision', 8);
```

```
end
fclose(file1);
```

Bash Script

The Linux script format is shown as a text file:

run_script.sh

```
#!/bin/bash
rm -rf polars/*
for k in {1..5000}
do
    str=$(printf "%d" $k)
    str1="..\/airfoil${str}.DAT"
    str2="2s/.*/${str1}/"
    sed -i "${str2}" commands.dat
    str3="polars\/polar${str}.dat"
    str5="19s/.*/${str3}/"
    sed -i "${str5}" commands.dat
#   name="log${str}"
    xfoil < commands.dat #> "${name}"
done
```

7.1.2. Data Preparation

Deleting Polar Files with Missing Data

Python

```
import pandas as pd
import os

def check_and_delete_files(directory):
    # Check all files in the specified directory
    for filename in os.listdir(directory):
        if filename.startswith('polar') and filename.endswith('.dat'):
            file_path = os.path.join(directory, filename)
            try:
                # Read the file, skip the first 11 lines
                data = pd.read_csv(file_path, skiprows=10, sep=r'\s+')

                # Check the angle values in the "alpha" column
                if 'alpha' in data.columns:
                    alpha_values = data['alpha'].dropna()  # Drop empty
values
                    print(f"{len(alpha_values)} alpha values found in
{filename}.")
                    if len(alpha_values) == 11:
                        print(f"{filename} file is valid.")
                        continue  # Skip valid files
                # Delete files that do not meet the criteria
                os.remove(file_path)
                print(f"{filename} file deleted because there are not
enough data in the alpha column.")
            except Exception as e:
                # Delete files with errors
                os.remove(file_path)
```

```
                print(f"{filename} file could not be read and was deleted.
Error: {e}")

directory = 'C:/directory/of/file'
check_and_delete_files(directory)
```

Reducing the Database to 5000 Entries Based on the C_l/C_d Ratio

```
import pandas as pd
import os
import re
from natsort import natsorted
from time import process_time

t_start = process_time()

def parse_polar_file(file_path):
    with open(file_path, 'r') as file:
        content = file.readlines()

    airfoil_data = []
    airfoil_number = None

    airfoil_pattern = re.compile(r'Calculated polar for: airfoil_(\d+)')
    data_pattern = re.compile(r'^\s*([\d\.\-]+)\s+([\d\.\-]+)\s+([\d\.\-
]+)\s+([\d\.\-]+)\s+([\d\.\-]+)\s+([\d\.\-]+)\s+([\d\.\-]+)')

    for line in content:
        airfoil_match = airfoil_pattern.search(line)
        data_match = data_pattern.match(line)

        if airfoil_match:
            airfoil_number = int(airfoil_match.group(1))
        elif data_match and airfoil_number is not None:
            try:
                alpha = float(data_match.group(1))
                Cl = float(data_match.group(2))
                Cd = float(data_match.group(3))
                airfoil_data.append((airfoil_number, alpha, Cl, Cd))
            except ValueError:
                continue

    return airfoil_data

def filter_data_by_alpha(data, alpha_value):
    return [item for item in data if item[1] == alpha_value]

def create_dataframe(data):
    df = pd.DataFrame(data, columns=['Airfoil Number', 'Cl', 'Cd'])
    df['Cl/Cd'] = df['Cl'] / df['Cd']
    return df

def process_files_in_directory(directory, alpha_values):
    for alpha_value in alpha_values:
        all_data = []

        # Sort file names using natsort
        filenames = natsorted(os.listdir(directory))

        for filename in filenames:
            if filename.startswith('polar') and filename.endswith('.dat'):
```

```
                file_path = os.path.join(directory, filename)
                airfoil_data = parse_polar_file(file_path)
                filtered_data = filter_data_by_alpha(airfoil_data,
alpha_value)
                for entry in filtered_data:
                    airfoil_number, _, Cl, Cd = entry
                    all_data.append((airfoil_number, Cl, Cd))

        final_df = create_dataframe(all_data)
        final_df.to_excel(f'cl_cd/airfoil_data_cl_cd_{alpha_value}.xlsx',
index=False)
        print(f"Excel file created for alpha = {alpha_value}")

# Define the directory containing the polar files
directory = 'C:/directory/of/file'
# Define the alpha values
alpha_values = list(range(1, 11))  # [1, 2, 3, ..., 10]

# Process all files for each alpha value and create separate Excel files
process_files_in_directory(directory, alpha_values)

t_end = process_time()

print('Elapsed time : ', t_end - t_start)
```

7.2. Determination of Hyperparameters

```
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import GridSearchCV
from time import process_time
import openpyxl

# CL parameters can be used for the CD value
coefficient = input("Please enter the coefficient (e.g., 'CM'): ")

titles = ['xtth', 'ytth', 'atte', 'acle', 'ycth', 'xcmc', 'ycmc', 'acte',
'CL', 'CD', 'CM']
attributes = ['xtth', 'ytth', 'atte', 'acle', 'ycth', 'xcmc', 'ycmc',
'acte']

def tune_hyperparameters(X, Y, params):
    """Tune SVR hyperparameters using GridSearchCV."""
    svr = svm.SVR()
    regr = GridSearchCV(svr, params)
    regr.fit(X, Y)
    return regr.best_params_, regr.best_score_

def adjust_hyperparameters(params, best_params):
    """Adjust hyperparameters based on the best params obtained."""
    # Adjust C
    c_values = [best_params['C'] / 100, best_params['C'], best_params['C']
* 100]
    # Adjust gamma
    gamma_values = [best_params['gamma'] / 100, best_params['gamma'],
best_params['gamma'] * 100]
    return {'C': c_values, 'gamma': gamma_values, 'kernel': ['rbf'],
```

```
'epsilon': [0.005]}

t_start = process_time()

results = []  # List to store results

for alpha in np.arange(1, 10.1, 1):
    filename = f"C:/directory/of/file/database/data_alpha_{alpha}.csv"
    data = pd.read_csv(filename, delimiter=',', names=titles)

    X = data[attributes]
    Y = data[coefficient]

    scaler = StandardScaler().fit(X)
    X_tr = scaler.transform(X)

    params = {'C': [0.1, 1, 10], 'gamma': [0.01, 0.1, 1], 'kernel':
['rbf'], 'epsilon': [0.005]}

    best_params, best_score = tune_hyperparameters(X_tr, Y, params)

    while True:
        params = adjust_hyperparameters(params, best_params)
        best_params, best_score = tune_hyperparameters(X_tr, Y, params)
        if params['C'][1] == best_params['C'] and params['gamma'][1] ==
best_params['gamma']:
            break

    results.append([alpha, best_params['C'], best_params['gamma'],
best_params['epsilon'], best_score])
    print(f"For data_alpha_{alpha}.csv: Best SVR with params: {best_params}
and R2 score: {best_score:.4f}")

results_df = pd.DataFrame(results, columns=['alpha', 'C', 'gamma',
'epsilon', 'best_score'])
results_df.to_excel(f"C:/directory/of/file/hyperparameter/parameter_results
_{coefficient}.xlsx", index=False)

t_end = process_time()

print('Elapsed time : ', t_end - t_start)
```

7.3. SVM Model

```
import pandas as pd
import numpy as np
from sklearn import svm
from sklearn.metrics import max_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split, cross_val_score,
ShuffleSplit
from sklearn import preprocessing
import matplotlib.pyplot as plt
from time import process_time

# Record the start time of the program
program_start_time = process_time()

# Desired coefficient to obtain
coefficient = input("Please enter the coefficient ('CM', 'CD', or 'CL'): ")

# Load the parameters
```

```
titles = ['alpha', 'C', 'gamma', 'epsilon']
data_params =
pd.read_excel(f"C:/directory/of/file/hyperparameter/parameter_results_{coef
ficient}.xlsx", names=titles, usecols=[0, 1, 2, 3])

# Define the data and features
titles_data = ['xtth', 'ytth', 'atte', 'acle', 'ycth', 'xcmc', 'ycmc',
'acte', 'CL', 'CD', 'CM']
attributes = ['xtth', 'ytth', 'atte', 'acle', 'ycth', 'xcmc', 'ycmc',
'acte']

# Loop over alpha values
for alpha in [round(x, 1) for x in list(np.arange(1.0, 10.1, 1))]:
    file_path = f"C:/directory/of/file/database/data_alpha_{alpha}.csv"
    data = pd.read_csv(file_path, delimiter=',', names=titles_data)

    if coefficient == "CD":

        CD = data['CD']
        min_value = CD.min()
        max_value = CD.max()
        diff = max_value - min_value

        def normalization(a):
            return (a - min_value) / diff

        CDN = [normalization(a) for a in CD]
        data['CDN'] = CDN

        train_set, test_set = train_test_split(data, test_size=0.2,
random_state=42)

        X = train_set[attributes]
        y_train = train_set["CDN"]

        x = test_set[attributes]
        y_te = test_set["CDN"]

        scaler = preprocessing.StandardScaler().fit(X)
        X_tr = scaler.transform(X)
        x_tr = scaler.transform(x)

        t_start = process_time()

        # Find the parameters and train the model
        target_row = data_params[data_params['alpha'] == alpha]

        if not target_row.empty:
            C_value = target_row['C'].values[0]
            gamma_value = target_row['gamma'].values[0]
            epsilon_value = target_row['epsilon'].values[0]

            regr = svm.SVR(kernel="rbf", C=C_value, gamma=gamma_value,
epsilon=epsilon_value)

            cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
            scores = cross_val_score(regr, X_tr, y_train, cv=cv,
scoring='r2')

            print(f"Alpha: {alpha}")
            print("Cross-validated R2 scores:", scores)
```

```
            regr.fit(X_tr, y_train)
            y_pr = regr.predict(x_tr)

            def original(b):
                return b * diff + min_value

            y_pred = [original(b) for b in y_pr]
            y_test = [original(b) for b in y_te]

            # Convert y_test and y_pred to numpy arrays
            y_test = np.array(y_test)
            y_pred = np.array(y_pred)

            r2 = r2_score(y_test, y_pred)
            rmse = np.sqrt(mean_squared_error(y_test, y_pred))
            max_err = max_error(y_test, y_pred)
            min_err = np.min(np.abs(y_test - y_pred))

            t_end = process_time()
            computation_time = t_end - t_start

            print(f"R2 score: {r2}")
            print(f"RMSE: {rmse}")
            print(f"Max Error: {max_err}")
            print(f"Min Error: {min_err}")
            print(f"Elapsed time: {computation_time:.2f} seconds\n")

            plt.figure()
            plt.plot(y_test, y_pred, 'bo', y_test, y_test)
            plt.title(f"Alpha: {alpha}")
            plt.xlabel(f"Actual {coefficient}")
            plt.ylabel(f"Predicted {coefficient}")
            plt.grid(True)

plt.savefig(f"C:/directory/of/file/outputs/outputs_results/outputs_{coeffic
ient}/figure_alpha_{coefficient}_{alpha}.png")
            plt.close()

            # Save metrics to Excel
            metrics_df = pd.DataFrame(
                {'R2 Score': [r2], 'RMSE': [rmse], 'Max Error': [max_err],
'Min Error': [min_err]})

metrics_df.to_excel(f"C:/directory/of/file/outputs/metrics/outputs_metrics_
{coefficient}/metrics_{coefficient}_{alpha}.xlsx",
                                index=False)

            # Save prediction results to Excel
            result_df = pd.DataFrame(
                {'Data_Num': test_set.index, f'{coefficient}_test': y_test,
f'{coefficient}_pred': y_pred})
            result_df.to_excel(

f"C:/directory/of/file/outputs/outputs_results/outputs_{coefficient}/output
_results_{coefficient}_{alpha}.xlsx", index=False)

        else:
            print(f"Parameters not found for alpha value: {alpha}")

    else:
```

```
        train_set, test_set = train_test_split(data, test_size=0.2,
random_state=42)

        X = train_set[attributes]
        y_train = train_set[coefficient]

        x = test_set[attributes]
        y_test = test_set[coefficient]

        scaler = preprocessing.StandardScaler().fit(X)
        X_tr = scaler.transform(X)
        x_tr = scaler.transform(x)

        t_start = process_time()

        # Find the parameters and train the model
        target_row = data_params[data_params['alpha'] == alpha]

        if not target_row.empty:
            C_value = target_row['C'].values[0]
            gamma_value = target_row['gamma'].values[0]
            epsilon_value = target_row['epsilon'].values[0]

            regr = svm.SVR(kernel="rbf", C=C_value, gamma=gamma_value,
epsilon=epsilon_value)

            cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
            scores = cross_val_score(regr, X_tr, y_train, cv=cv,
scoring='r2')

            print(f"Alpha: {alpha}")
            print("Cross-validated R2 scores:", scores)

            regr.fit(X_tr, y_train)
            y_pred = regr.predict(x_tr)

            # Convert y_test and y_pred to numpy arrays
            y_test = np.array(y_test)
            y_pred = np.array(y_pred)

            r2 = r2_score(y_test, y_pred)
            rmse = np.sqrt(mean_squared_error(y_test, y_pred))
            max_err = max_error(y_test, y_pred)
            min_err = np.min(np.abs(y_test - y_pred))

            t_end = process_time()
            computation_time = t_end - t_start

            print(f"R2 score: {r2}")
            print(f"RMSE: {rmse}")
            print(f"Max Error: {max_err}")
            print(f"Min Error: {min_err}")
            print(f"Elapsed time: {computation_time:.2f} seconds\n")

            plt.figure()
            plt.plot(y_test, y_pred, 'bo', y_test, y_test)
            plt.title(f"Alpha: {alpha}")
            plt.xlabel(f"Actual {coefficient}")
            plt.ylabel(f"Predicted {coefficient}")
            plt.grid(True)
```

```
plt.savefig(f"C:/directory/of/file/outputs/outputs_results/outputs_{coeffic
ient}/figure_alpha_{coefficient}_{alpha}.png")
            plt.close()

            # Save metrics to Excel
            metrics_df = pd.DataFrame(
                {'R2 Score': [r2], 'RMSE': [rmse], 'Max Error': [max_err],
'Min Error': [min_err]})

metrics_df.to_excel(f"C:/directory/of/file/outputs/metrics/outputs_metrics_
{coefficient}/metrics_{coefficient}_{alpha}.xlsx",
                                index=False)

            # Save prediction results to Excel
            result_df = pd.DataFrame(
                {'Data_Num': test_set.index, f'{coefficient}_test': y_test,
f'{coefficient}_pred': y_pred})
            result_df.to_excel(

f"C:/directory/of/file/outputs/outputs_results/outputs_{coefficient}/output
_results_{coefficient}_{alpha}.xlsx", index=False)

        else:
            print(f"Parameters not found for alpha value: {alpha}")

# Record the end time of the program and calculate the total runtime
program_end_time = process_time()
total_computation_time = program_end_time - program_start_time

print(f'Total elapsed time for the entire program:
{total_computation_time:.2f} seconds')
```