Detecting Missing Flow Separation and Predicting Error in Drag using Supervised Machine Learning

by

Amirpasha Kamalhedayat

B.Sc., Aerospace Engineering, Amirkabir University of Technology, 2019

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Mechanical Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

October 2022

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Detecting Missing Flow Separation and Predicting Error in Drag using Supervised Machine Learning

submitted by **Amirpasha Kamalhedayat** in partial fulfillment of the requirements for the degree of **Master of Applied Science** in **Mechanical Engineering**.

Examining Committee:

Carl Ollivier-Gooch, Professor, Mechanical Engineering, UBC
Supervisor

Rajeev K. Jaiman, Associate Professor, Mechanical Engineering, UBC
Supervisory Committee Member

Gwynn Elfring, Associate Professor, Mechanical Engineering, UBC
Supervisory Committee Member

Abstract

Accuracy of flow simulations is a major concern in Computational Fluid Dynamics (CFD) applications. A possible outcome of inaccuracy in CFD results is missing a major feature in the flow field. Many methods have been proposed to reduce numerical errors and increase overall accuracy, but these are not always efficient or even feasible. In this study, a purely data-driven approach is proposed to assess flow simulations both in a qualitative and a quantitative manner. In this regard, Principal Component Analysis (PCA) has been performed on compressible flow simulations around an airfoil to map the high-dimensional CFD data to a lower-dimensional PCA subspace. A machine learning classifier based on the extracted principal components has been developed to detect the simulations that miss the separation bubble behind the airfoil. The evaluative measures indicate that the model is able to detect most of the simulations where the separation region is poorly resolved. Besides the classifier, a machine learning regressor has been trained on the same PCA subspace to predict the error in the output drag coefficient. The predictions reveal that the regression model estimates accurate errors with a tight uncertainty bound. Further, more efficient models built on top of fewer PCA modes have been implemented that show similar performance. In addition, the developed models were used to inspect simulations solved on a different mesh configuration from the one the models were trained on. This generalization framework gives rise to some challenges that are thoroughly discussed. Overall, the results demonstrate that machine learning models based on the principal components of the data set are promising tools to detect possible missing flow features and predict numerical errors in CFD.

Lay Summary

Computer simulations are an inseparable component of engineering designs nowadays. Hence, the accuracy of the simulation plays an integral role in assuring the quality and the safety of the end product. The common tools used to check and increase this accuracy are rather expensive for real-world applications. This thesis proposes a machine learning method to inspect whether a fluid simulation correctly represents the actual phenomenon and captures important patterns in the flow field. Similar to a facial recognition system, the developed tool looks for certain patterns in the output of a simulation and alerts the engineers when those are missing.

Preface

The developed models presented in this thesis are the outcome of the research conducted by the author, Amirpasha Kamalhedayt, under the direct supervision of Dr. Carl Ollivier-Gooch, as part of the Master of Applied Science program in Mechanical Engineering at the University of British Columbia (UBC). Data set generation, machine learning, model evaluation, and manuscript preparation were done by Amirpasha Kamalhedayat with the insightful guidelines from Carl Ollivier-Gooch. Simulation data were generated using the codes in GRUMMP and ANSLib packages that have been developed by the members of the Advanced Numerical Simulation Laboratory at UBC. The following papers related to the current thesis have been or are in the preparation process to be submitted:

- Parts of Chapters 2 and 3 have been accepted to be presented in the Machine Learning and AI session in the AIAA SciTech 2023 Forum, titled: "Detecting Missing Flow Separation using Supervised Machine Learning".

- A comprehensive discussion on the machine learning classifier developed in this study (Chapters 2 and 3) is currently in preparation for submission to the AIAA Journal.

- As a second part for the previous item, the results of the machine learning regression (Chapters 2 and 4) are planned to be published in the AIAA Journal.

Also, the machine learning data set generated and used in this work is made available for public use on GitHub[1].

[1] `https://github.com/APHedayat/missing-flow-feature-dataset.git`

Table of Contents

List of Tables

List of Figures

List of Programs

Acknowledgements

First, I would like to express my gratitude and appreciation to my supervisor, Dr. Carl Ollivier-Gooch, for all the support and guidance I received throughout the course of this research. It has been a difficult time with an unexpected pandemic in the world that forced us to meet virtually for two years; but somehow, we managed to complete this novel work regardless of all the challenges.

I also acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), project number RGPIN-2020-04503. (Cette recherche a été financée par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), numéro de projet RGPIN-2020-04503.)

Moreover, I would like to thank my dear colleagues at the Advanced Numerical Simulation Laboratory for all the informative discussions we had at the lab and during our group meetings. In particular, I very much appreciate all the help and support I received from Mohammad, who also proofread this thesis before submission.

Last, but not least, my warm and heartful gratitude goes to my dedicated mom, Nazli, for her endless love and encouragement. She is the strongest, kindest, and most wonderful human being I have ever known in my life, and hopefully, I would make her proud one day by reaching my dreams.

To my mom.

Chapter 1

Introduction and Background

1.1 Motivation

Since ancient times, data have played an integral role in scientific research and discovery. From ancient Egyptians, who recorded and stored data on *Papyrus*, to the recent black hole exposition revealed by the Event Horizon Telescope, researchers have always based their studies upon data. Nowadays, with the unprecedented improvements in computational power of computers following Moore's law [1], our ability in collecting data has significantly increased. As a result of this data abundance, there are growing appeals in the scientific community for data-driven methods to explore and exploit data in recent years. Evidence for this is the emerging machine learning algorithms and deep neural networks to study huge amounts of data to draw meaningful conclusions. Many people believe that we are in a transition phase from the prior three modes of study, namely theoretical, experimental, and numerical methods, to the *fourth paradigm* [2], which is the use of data-driven models (Fig. 1.1). It should be emphasized that each mode of research was not invented to replace the former ones, but rather to augment those. The same thing is true for data-driven approaches; they are being used in conjunction with other scientific methods to improve our understanding of the surrounding world.

One of the most important means of research over the past half century has been numerical simulation. This approach consists of performing mathematical modeling on a computer, which provides a powerful framework to

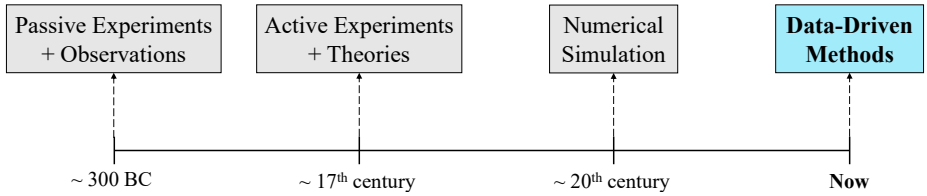


Figure 1.1: History of scientific methods.

study physical phenomena and to design engineering systems. Many fields of studies such as astronomy, chemistry, neuroscience, and economics have been affected by computer simulations. This trend is expected to continue with the increasing power of computers to simulate more complex phenomena. Among all fields of science, fluid mechanics has been presumably affected the most by the breakthroughs in computational power. Studying fluids is of utmost importance for us as living beings on earth; we live in a fluid, we breathe in a fluid, and almost all of our fabrications are exposed to a fluid: the *air*. It makes it extremely crucial for us to study and understand fluid dynamics and use that knowledge in major industries, from medicine to aerospace. Fluid movement is governed by a highly nonlinear and chaotic dynamical system which cannot be solved analytically. In fact, proving the existence of an analytical solution to this group of equations is one of the seven *Millennium Prize Problems*, indicating the complex nature of fluids. As a result, researchers in this area often employ computer simulations to study complicated phenomena occurring in fluids, among which the most mysterious one is the turbulence. Numerical simulation in fluid mechanics is known as Computational Fluid Dynamics (CFD), which is a key component in industrial applications. CFD aims at discretizing fluid flow equations, typically the Navier-Stokes equations, and solving the resulting system by simple algebraic methods. This approach substantially reduces the costs of design compared to experimental methods where a physical prototype model is required. Also, the fact that all data in the domain of simulation are available eliminates the need for sensors in certain locations, which is one of the challenges in experimental setups, reducing the cost even further and making the results more comprehensive. However, *every light*

has its shadow; although CFD might be exceptional in cost reduction, there are grave challenges attached to its use in action. These stem from the step where the governing differential equations are mapped from a continuous space to a discrete one. Here, meticulous care must be taken to prevent the simulation from divergence, and even worse, inaccurate results. The former is identified instantly and the researchers will be alerted to solve the problem to reach convergence. The latter, however, could stay hidden without showing any obvious *sign*, which eventually leads to misleading results and disastrous engineering designs. This inaccuracy in CFD methods can result in a simulation where one or more major flow features (such as separation bubble, shock wave, etc.) are either missed, or poorly resolved. These important flow patterns can significantly affect the main aerodynamic parameters such as lift and drag forces, and capturing those is one of the major goals of CFD applications.

Many methods have been proposed within CFD community to increase the accuracy of numerical simulations and to make sure that important phenomena and patterns in the flow field are represented correctly. Examples of these well-known approaches are grid refinement, where the resolution of the mesh is increased in certain areas of the domain to better compute gradients and capture patterns, and error estimators, that introduce additional equations to be solved alongside the flow equations as a correction. These methods do additional computational work to improve the solution with no a priori check to determine whether the solution is qualitatively correct. It could be the case that the available simulation already resolves all important flow features and computes parameters with an acceptable accuracy. By engaging traditional improvement methods, we need at least one step of validation by introducing additional computation into the simulation process. Only then can the integrity of the simulation be authenticated. In some cases involving flows around complex geometries, applying these techniques could take weeks or even months of simulations without causing a notable change in the output. In these scenarios, the mentioned approaches only waste time and computational resources and considerably delay the design phase in a project. In more complex situations, performing accuracy

improvement methods could even become infeasible. These all suggest the need for a method to examine the available simulation in hand, both qualitatively and quantitatively, to see whether it needs further improvement, before applying costly methods such as grid refinement and error estimators. In other words, CFD thirsts for an efficient way to find that *sign* that points to an inaccurate simulation.

CFD is one of the fields where massive amounts of data are being produced constantly. Although it makes it an ideal domain of interest to integrate data-driven and machine learning methods, we do not see many of such attempts in CFD and the aerospace industry compared to other fields of research such as economics or robotics. This could be related to two reasons:

- *Data Generation*: Gathering data in CFD is not as easy and straightforward as many other areas. There are many things that should be taken care of beforehand, such as turbulence modeling, possible stability issues, order of accuracy, and time advance schemes, to generate useful data for machine learning purposes.

- *Safety Regulations*: Due to stringent safety matters involved in the aerospace industry, entering machine learning into the design loop for decision making requires a minimum degree of confidence; something that the new ambitious data-driven methods lack.

Even though the mentioned reasons still hold and machine learning is not mature enough to be a reliable tool in the aerospace industry, researchers and engineers have begun to explore the use of such methods in aerospace applications (see e.g, [3]). In this thesis, we follow this movement and take advantage of the high volumes of CFD data to develop a machine learning classifier to accurately detect an under-resolved flow feature. Besides, we will go a step further to design a complementary machine learning regressor to estimate the error in the computed parameter of interest. The former could be used as a priori tool to qualitatively examine a simulation with regard to capturing major flow patterns, and the latter could be considered an efficient

alternative to the costly traditional accuracy improvement methods with the goal of quantifying uncertainties involved in a simulation. This work aims at creating a connection between CFD and the emerging powerful data-driven methods to tackle one of the most challenging aspects of numerical simulations.

The rest of this chapter is structured as follows: Sections 1.2 and 1.3 describe the building blocks of any numerical simulation, the mesh generator and the flow solver, respectively. Following these, Section 1.4 explains why inaccuracies exist in a simulation, and introduces the main types of error encountered in CFD. Also, a case study is presented in this section to show a possible effect of these errors in a simulation. The current conventional methods to deal with inaccuracies in CFD results and their drawbacks are mentioned in Section 1.5. We introduce our proposed data-driven method in Section 1.6, and finally, the test case and data set considered in this study are presented in Section 1.7.

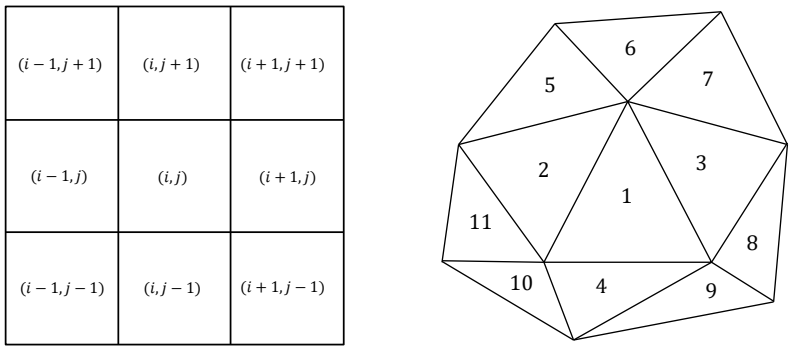
1.2 Mesh Generator

In numerical simulations, the spatial domain on which the physical process will be solved must be decomposed, or tessellated, into smaller pieces called control volumes, elements, or cells. This process is referred to as mesh generation and is one of the initial steps in any numerical simulation. It often requires direct human supervision, although CFD practitioners have proposed techniques to automate this time consuming task [4]. There are primarily two categories of meshes:

- *Structured*: This type of mesh is identified by the regular connectivity between the cells in the domain. This means that each control volume could be assigned with (i, j) or (i, j, k) indices in two-dimensional (2-D) or three-dimensional (3-D) space, respectively. Because of the regular arrangement of the cells in structured meshes, these are more favorable in formulating discretized equations. However, generating this type of mesh around geometries with complex configurations is a major

challenge and setback. Figure 1.2a depicts a fragment of a typical 2-D structured mesh.

- *Unstructured*: These meshes are more flexible in accepting arbitrary element shapes and have irregular connectivity between the cells in the domain. This irregularity makes it compulsory to store all connectivity information in a matrix to keep track of the neighbors surrounding each cell. While numerical discretization is more challenging compared to structured grids, unstructured meshes are much easier to generate in complex configurations, making them the most popular type of meshes in CFD nowadays. An illustration of a segment of a triangular unstructured mesh is presented in Fig. 1.2b.



(a) 2-D structured **(b) 2-D unstructured**

Figure 1.2: Main types of computational domain.

For the purpose of this work, our in-house mesh generating software called GRUMMP[2] [5] is used, which was originally developed to automate unstructured mesh generation and reduce human interference.

After the mesh is generated, the flow equations can be solved for each individual cell in the computational domain to form a large linear system of equations. The cells have common interfaces through which flow variables

[2]Generation and Refinement of Unstructured Mixed-Element Meshes in Parallel

and information are transferred by the means of convection and diffusion. The following section will introduce how the governing equations of the flow field are discretized on the computational mesh.

1.3 Finite Volume Flow Solver

Computers can only perform simple arithmetic operations. This means that a Partial Differential Equation (PDE) governing a physical phenomenon is a strange being to the computer that needs to be processed. This processing step involves discretizing a PDE on the numerical mesh introduced in the previous section. For a detailed description of this process in CFD, there exist many great resources (see e.g., [6]). The conventional approaches to perform discretization in science and engineering are finite-difference, finite-element, and finite-volume methods:

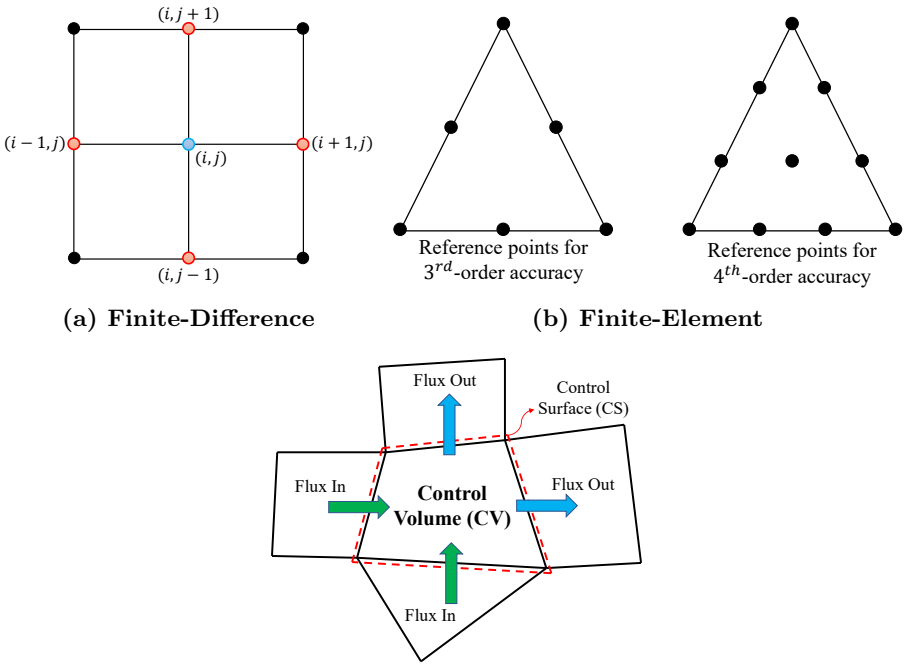
- *Finite-Difference Method (FDM)*
  FDM is the classical approach taken to discretize differential equations. It results in a point-wise representation of the computational domain, where derivatives are approximated using the Taylor series expansion of the solution around a certain point. This makes FDM an easy-to-derive discretization technique when applied on a structured mesh. Figure 1.3a shows a simple representation of the data stencil used in FDM where five neighboring nodes are engaged. Despite its simplicity, it becomes difficult to implement FDM for unstructured meshes where the cells are arranged irregularly. Moreover, conservation of mass, momentum, and energy are not guaranteed when performing FDM, making it an unsuitable choice in presence of discontinuities such as shock waves. Nevertheless, FDM is still useful for educational purposes and also for testing new methods on simple geometries.

- *Finite-Element Method (FEM)*
  FEM has been one of the key tools in numerical simulation, mainly to analyze solid structures. This method considers the weak formulation of governing equations and tries to approximate a trial solution in the

domain. This task is done by introducing local basis functions over the discrete elements (cells). The equations are then multiplied by these functions and integrated over the entire mesh. The goal is to minimize the numerical errors, also known as the residuals. Higher-order accurate solutions can be obtained simply by increasing the degree of polynomial basis functions. Also, this method is well suited for both structured and unstructured meshes with arbitrary element shapes. Figure 1.3b depicts an example of the nodal locations where FEM computes the solution, both for a 3^{rd}-order accurate approximation and a 4^{th}-order one. The downside of using FEM is that the conservation laws are not easily achievable, which could be problematic when facing flows with discontinuities.

- *Finite-Volume Method (FVM)*
  FVM method takes a more intuitive and physical approach by discretizing the integral form of governing equations over the domain. By doing so, conservation laws for each control volume are preserved, making FVM well suited for flows with discontinuities. Furthermore, FVM can handle both structured and unstructured meshes similar to FEM. However, reaching higher-order accuracy is more challenging compared to FEM, which will be addressed in the following subsections. An illustration of an FVM control volume is provided in Fig. 1.3c.

In this work, we take the finite-volume approach to discretize viscous compressible flow equations. In particular, we used the higher-order unstructured finite volume flow solver package (ANSLib), developed at the Advanced Numerical Simulation Laboratory at the University of British Columbia. FVM is widely used in commercial CFD software, making it the best choice for the purpose of this study. Nevertheless, the proposed data-driven approach in this thesis is expected to be applicable to any simulation, regardless of the discretization technique.



(a) Finite-Difference **(b) Finite-Element**

(c) Finite-Volume

Figure 1.3: Common discretization approaches.

1.3.1 Spatial Discretization

The PDEs governing the dynamics of fluids are the Navier-Stokes equations that describe the conservation of mass, momentum, and energy in a fluid domain. These equations should be first recast in a fully conservative formulation represented by:

$$\frac{\partial \vec{U}}{\partial t} + \nabla \cdot \vec{F} = 0 \tag{1.1}$$

to be discretized using FVM. In Eq. (1.1), $\vec{U}$ is the solution vector and $\vec{F}$ represents the total flux composed of convective flux, $\vec{F}_c$, and diffusive flux, $\vec{F}_d$:

$$\vec{F} = \vec{F}_c - \vec{F}_d \tag{1.2}$$

For the 2-D compressible viscous laminar Navier-Stokes, solution and flux vectors are written as:

$$\vec{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E_t \end{pmatrix}, \quad \vec{F}_c^x = \begin{pmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ u\left(E_t + P\right) \end{pmatrix}, \quad \vec{F}_c^y = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ v\left(E_t + P\right) \end{pmatrix}$$

$$\vec{F}_v^x = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ u\tau_{xx} + v\tau_{xy} + c_p\left(\frac{\mu}{Pr}\right)\frac{\partial T}{\partial x} \end{pmatrix} \tag{1.3}$$

$$\vec{F}_v^y = \begin{pmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ u\tau_{yx} + v\tau_{yy} + c_p\left(\frac{\mu}{Pr}\right)\frac{\partial T}{\partial y} \end{pmatrix}$$

where ρ is the fluid density, u and v are the Cartesian velocity components, P is the fluid pressure, E_t is the total energy, c_p is the specific heat at constant pressure, μ is the fluid dynamic viscosity, T is the fluid temperature, Pr is the Prandtl number, and τ_{ij} is the viscous stress tensor. Following FVM guidelines, if we pick an arbitrary control volume in the mesh and integrate Eq. (1.1) over that volume, we get:

$$\iiint_{CV} \left(\frac{\partial \vec{U}}{\partial t} + \nabla \cdot \vec{F}\right) dV = 0 \tag{1.4}$$

As will be noted in Section 1.7, the test case in this study is a 2-D flow simulation. Therefore, we turn our attention on the 2-D form of this formula, where the control volume (CV) is represented by the 2-D cell area, and cell edges constitute the surrounding control surface (CS). Applying these changes and using divergence theorem, we get the following as a combination

of volume and surface integrals:

$$\iint_{CV} \frac{\partial \vec{U}}{\partial t} dA + \oint_{CS} \vec{F} \cdot \hat{n} ds = 0 \tag{1.5}$$

where $\hat{n}$ is the outward unit normal vector on the control surface of the cell, and ds is an infinitesimal segment on cell boundaries. If cell shapes are independent of the time variable in the simulation (fixed grid), then $\vec{U}$ can be brought out from the volume integral, resulting in:

$$\frac{d\bar{U}_i}{dt} = -\frac{1}{A_{CV_i}} \oint_{CS_i} \vec{F} \cdot \hat{n} ds \tag{1.6}$$

where $\bar{U}_i$ is the average solution of the control volume, and A_{CV_i} is the area of the 2-D cell. The left hand side represents the change of average solution in control volume when advancing in time. The right hand side is the residual of the numerical simulation, which is more commonly known as the flux integral in CFD. The relation could be represented more compactly as:

$$\frac{d\bar{U}_i}{dt} = -R_i\left(\bar{U}_i\right) \tag{1.7}$$

for the i^{th} cell in the domain. Replacing the right hand side by proper algebraic approximations, we obtain an Ordinary Differential Equation (ODE) for each control volume, which is much easier to solve compared to the original PDE. We go deeper into this process in the following subsections.

1.3.2 *k*-Exact Reconstruction and High-Order Methods

After obtaining Eq. (1.7), the goal now is to approximate the residual in simple terms. For structured grids, where cells are systematically arranged, solution approximation in each cell is as easy as writing a Taylor expansion and integrating the terms over each control volume. This process will look similar to FDM, where guidelines on how to reach higher-order accurate approximations are well documented. Unstructured meshes, on the other hand, lack this property and a different approach must be taken. The method introduced here is known as the k-Exact Reconstruction, and a de-

tailed explanation is available in [7]. In this method, Taylor expansion is considered around an arbitrary reference point (usually the centroid of the cell, for simplicity) to reconstruct the following polynomial solution for each cell:

$$\begin{aligned} U_i^R(x-x_i, y-y_i) = U|_i &+ \left.\frac{\partial U}{\partial x}\right|_i (x-x_i) + \left.\frac{\partial U}{\partial y}\right|_i (y-y_i) \\ &+ \left.\frac{\partial^2 U}{\partial x^2}\right|_i \frac{(x-x_i)^2}{2} \\ &+ \left.\frac{\partial^2 U}{\partial x \partial y}\right|_i (x-x_i)(y-y_i) \\ &+ \left.\frac{\partial^2 U}{\partial y^2}\right|_i \frac{(y-y_i)^2}{2} + \dots \end{aligned} \tag{1.8}$$

where U_i^R is the reconstructed solution within the cell, and the derivatives are considered at the reference point $(x_i,\ y_i)$. The unknowns in this representation are $U|_i$ and the derivatives. Once these coefficients are calculated, any field variable at any location within the cell could be approximated by inserting the coordinates in Eq. (1.8). To solve for the unknown coefficients, we use the fact that if we integrate the reconstructed solution over the control volume, we should get back the average solution $\bar{U}_i$ on that cell:

$$\bar{U}_i = \frac{1}{A_i} \int_{V_i} U_i^R dA \tag{1.9}$$

Inserting moments of area defined by:

$$\overline{x^p y^q}_i = \frac{1}{A_i} \int_{V_i} (x-x_i)^p (y-y_i)^q \, dA \tag{1.10}$$

into the extended version of Eq. (1.9) when U_i^R is substituted from Eq. (1.8), we get the following:

$$\begin{aligned} \bar{U}_i = U|_i &+ \left.\frac{\partial U}{\partial x}\right|_i \bar{x}_i + \left.\frac{\partial U}{\partial y}\right|_i \bar{y}_i + \left.\frac{\partial^2 U}{\partial x^2}\right|_i \frac{\overline{x^2}_i}{2} \\ &+ \left.\frac{\partial^2 U}{\partial x \partial y}\right|_i \overline{xy_i} + \left.\frac{\partial^2 U}{\partial y^2}\right|_i \frac{\overline{y^2}_i}{2} + \dots \end{aligned} \tag{1.11}$$

To compute the unknown coefficients, Eq. (1.11) is rewritten for each control volume within the stencil (neighborhood) of the i^{th} cell. The number of cells in the stencil is chosen to be greater than the number of coefficients to be approximated to yield a least-squares system of equations. To reach higher-order of accuracy, defined as any order more than two, more terms in Taylor expansion should be retained, and hence, more cells in the stencil should be considered. The suggested number of cells in the stencil are three for 2^{nd}-order, nine for 3^{rd}-order, and fourteen for 4^{th}-order accurate solution reconstruction [7]. Figure 1.4 provides a representation of cells in the stencil of an arbitrary control volume in an unstructured mesh. Each cell is labeled with the desired order of accuracy for which that cell should be considered to find the unknowns in Eq. (1.8).

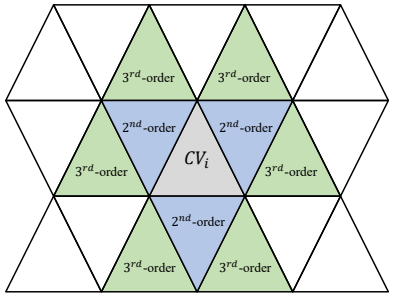
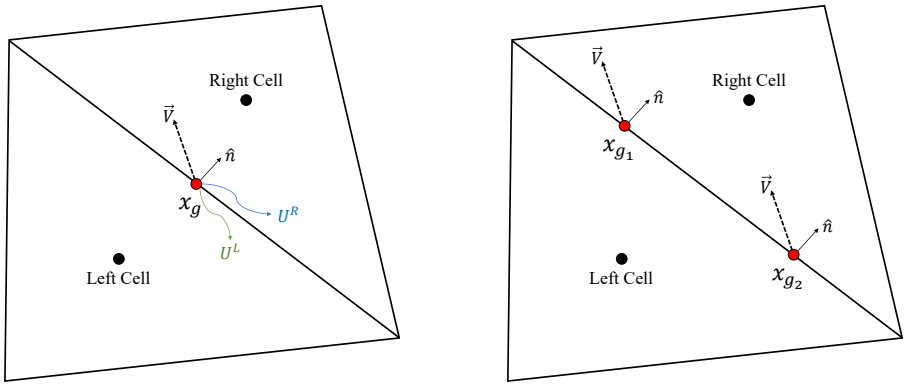


Figure 1.4: Stencils for *k*-Exact Reconstruction.

1.3.3 Flux Integration

Having reconstructed solutions in each cell using Eq. (1.8), flux values can be calculated and integrated on the edges of each control volume. The integration is done by the Gauss quadrature integration rule, where the integrand is evaluated at only a few points [8]. A 2^{nd}-order accurate flux integral can be obtained by a single quadrature point at the middle of the corresponding edge, whereas for higher-order accuracy, more points must be considered and weighted appropriately. It should be noted that the order of accuracy by which fluxes are calculated should be at least equal to the order

of solution reconstruction to retain the overall accuracy of the simulation. Figures 1.5a and 1.5b illustrate Gauss quadrature points respectively for 2^{nd}- and 3^{rd}- or 4^{th}-order approximation of the flux integral. Van Altena [9] provides a comprehensive overview of the process of flux integration. Fluxes are grouped as two major physical means of transportation, convection and diffusion, which will be dealt with separately in the following.



(a) Using one Gauss quadrature point (2^{nd}-order accurate integration)

(b) Using two Gauss quadrature points (3^{rd}- or 4^{th}-order accurate integration)

Figure 1.5: Flux integration on the common edge of two neighboring cells.

Convective Flux

Convection, also known as advection, is the process in which field variables are transported with the movement of fluid. Hence, fluid velocity plays in important role in this process. To calculate convective flux, we consider the general 2-D advection equation, which governs the transport of a conserved variable ϕ in a constant velocity field $\vec{V} = (u, v)$:

$$\frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial x} + v\frac{\partial \phi}{\partial y} = 0 \tag{1.12}$$

Here, following the fully conserved formulation represented by Eq. (1.1), solution U and convective flux $\vec{F}_c$ are defined as:

$$U = \phi \quad , \quad \vec{F}_c = \begin{bmatrix} u\phi \\ v\phi \end{bmatrix} \tag{1.13}$$

For simplicity, we take only one quadrature point as shown in Fig. 1.5a on the common edge of two neighboring cells. Using reconstructed solutions obtained for the left and the right cells, we get different values for the solution and its derivatives on the quadrature point, as no restrictions are set to ensure a smooth solution on the common edge. This requires us to combine the fluxes obtained from each reconstructed solution on the sides of the edge, $\vec{F}\left(U^L\right)$ and $\vec{F}\left(U^R\right)$, to get a single value. There are two approaches to define the combined flux, as explained in the following.

One common way is to take the average of the two fluxes on each side of the edge as the flux value:

$$\vec{F}_c = \frac{\vec{F}\left(U^L\right) + \vec{F}\left(U^R\right)}{2} \tag{1.14}$$

This representation of the convective flux is the central scheme, where both fluxes are weighted equally.

Another approach comes from a physical observation, where we argue that the values on the edge are affected solely by the information coming from the upstream cell. This method is known as the upwind scheme, and suggests that the direction of the flow should be calculated on the edge to decide which cell to use as the reference:

$$\vec{F}_c = \begin{cases} \vec{F}\left(U^L\right) & \vec{V} \cdot \hat{n} \geq 0 \\ \vec{F}\left(U^R\right) & \vec{V} \cdot \hat{n} < 0 \end{cases} \tag{1.15}$$

Diffusive Flux

Diffusion is the process of information spreading out at the molecular level. This time, we take the general 2-D diffusion equation:

$$\frac{\partial \phi}{\partial t} - \left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} \right) = 0 \tag{1.16}$$

In this case, solution U and diffusive flux $\vec{F}_d$ contain:

$$U = \phi, \quad \vec{F}_d = \begin{bmatrix} -\frac{\partial \phi}{\partial x} \\ -\frac{\partial \phi}{\partial y} \end{bmatrix} \tag{1.17}$$

We again consider the cell configuration in Fig. 1.5a. The common strategy to formulate a single diffusive flux on the edge is by arithmetic averaging the fluxes obtained from neighboring cells and adding an extra jump term [10] for stability purposes:

$$\vec{F}_d = \frac{\vec{F}\left(U^L\right) + \vec{F}\left(U^R\right)}{2} + \frac{\alpha}{\vec{r}_{LR} \cdot \hat{n}} \left(U^L - U^R\right) \hat{n} \tag{1.18}$$

The second term on the right hand side is the jump term, where $\vec{r}_{LR}$ denotes the vector connecting the reference points of the neighboring cells, and α is a constant.

1.3.4 Temporal Discretization

At this point, solution reconstructions are obtained for each cell and flux integrals are approximated on cell edges. It means that the residual value on the right hand side of Eq. (1.7) is calculated, and an ODE is obtained for each control volume in the domain. To solve this equation, there exist generally two techniques: the explicit and the implicit time advance.

In the explicit time advance, the residual terms are specified based on the known solution values obtained in the previous time step. Applying a

backward time difference scheme to the left hand side of Eq. (1.7) gives:

$$\frac{\bar{U}_i^{n+1} - \bar{U}_i^n}{\Delta t} = -R\left(\bar{U}_i^n\right) \tag{1.19}$$

where superscripts denote the time level. In this formula, the only unknown is the solution value at the next time step and can be easily calculated. Although this approach provides a simple formulation and code implementation, it imposes severe restrictions on the choice of time step Δt as a function of the smallest length scale in the mesh, to prevent solution instabilities.

The implicit time advance scheme, on the other hand, writes the residual terms as a function of unknown solution values of the next time step:

$$\frac{\bar{U}_i^{n+1} - \bar{U}_i^n}{\Delta t} = -R\left(\bar{U}_i^{n+1}\right) \tag{1.20}$$

This case requires more operations to solve for the unknown solution $\bar{U}_i^{n+1}$. To simplify the right hand side, the residual is linearized around the known solution state $\bar{U}_i^n$ as:

$$\begin{aligned} \frac{\bar{U}_i^{n+1} - \bar{U}_i^n}{\Delta t} &= -R\left(\bar{U}_i^{n+1}\right) \\ &= -\left[R\left(\bar{U}_i^n\right) + \frac{\partial R}{\partial \bar{U}}\left(\bar{U}_i^{n+1} - \bar{U}_i^n\right) + \mathcal{O}\left(\bar{U}_i^{n+1} - \bar{U}_i^n\right)^2\right] \end{aligned} \tag{1.21}$$

where $\dfrac{\partial R}{\partial \bar{U}}$ is the Jacobian matrix which can be obtained using the guidelines provided in [11]. For simplicity, we can define a solution difference as:

$$\delta \bar{U}_i = \bar{U}_i^{n+1} - \bar{U}_i^n \tag{1.22}$$

which when inserted into Eq. (1.21) yields:

$$\left(\frac{I}{\Delta t} + \frac{\partial R}{\partial \bar{U}}\right)\delta \bar{U}_i = -R\left(\bar{U}_i^n\right) \tag{1.23}$$

Despite the computation complexity of the implicit approach, this method puts no limit on the time step, making it ideal for steady-state simulations

where large Δt can be used for faster convergence. As will be mentioned in Section 1.7, our test case in this work is a steady simulation, and hence, the implicit time advance was implemented in this study.

Performing spatial discretization, solution reconstruction, flux integration, temporal discretization, and gathering individual equations derived for each cell together, we obtain a system of linear algebraic equations that can be solved for the solution vector $\vec{U}^{n+1}$ at each time step by employing matrix operations.

1.4 Source of Inaccuracy in Simulations

From describing a physical phenomenon using mathematical tools, to reading outputs on computer screen, there are many types of error introduced into the simulation along the way. These errors can be categorized into two main groups of modeling and numerical errors. Presence of different types of errors deviates the simulation output from reality. We will elaborate on the possible errors in the following.

1.4.1 Modeling Error

Modeling errors are defined as the mismatch between the exact solution to a mathematical model and the actual physical phenomenon it represents. These can be further subdivided into two groups: first, the physical modeling errors that arise from the inadequacy of the governing equations to accurately manifest the physical events; and second, the geometrical modeling errors that are discrepancies introduced by the model to correctly capture geometrical properties of the solution domain.

In CFD, if the full Navier-Stokes equations are discretized and solved on a mesh, the approach is called Direct Numerical Simulation (DNS), where every spatial and temporal length scales are resolved. This only works for a few cases, and becomes impractical when facing flows with medium to high Reynolds numbers. In these scenarios, chaotic terms in the Navier-Stokes equations are simplified using turbulence modeling techniques such

as Reynolds-Average Navier-Stokes (RANS) and Large-Eddy Simulation (LES). Wilcox [12] provided an overview of different turbulence modeling techniques. These methods introduce physical modeling errors into the simulation process, and are still one of the most challenging aspects of CFD. Recent efforts [13–15] have tried to model turbulence effects by modern machine learning techniques with success, which indicates the tendency to take advantage of high volumes of data generated in CFD simulations.

Although modeling errors are still an open problem in CFD applications, the focus of this thesis is on a different type of error, which is equally important.

1.4.2 Numerical Error

A different type of error comes from the discretization process, where governing PDEs are discretized to form a linear system of algebraic equations, which can be solved by a computer. These are known as numerical errors, and in most aerodynamic simulations, are the dominant source of inaccuracy [16]. Again, this type of error can be categorized in the following groups:

Discretization Error

Discretization error is usually the type of error we mean when we refer to the simulation error, and is the focus of this study. Therefore, throughout this thesis, the term error corresponds to discretization error. It is defined as the difference between the exact and approximated solutions to the governing equations. The approximated solution is the outcome of the steps mentioned in Section 1.3, when model equations have gone through discretization operations. The exact solution is often unknown in real-world cases; otherwise a simulation would not be of much help since we already know what we expect to get. Hence, calculating discretization error in most CFD cases requires implementing special tools, known as error estimators. A classic method to get discretization error is Richards extrapolation [17, 18], which uses a series of approximated solutions on a sequence of varying resolution meshes. In addition, two modern methods will be covered in the next section.

Truncation Error

When the exact operator in Eq. (1.1) is applied to the approximated discrete solution obtained by solving Eq. (1.23), it will not result in zero. This residual is known as the truncation error, which is one of the most important sources of inaccuracy affecting discretization error directly. Similarly, if the discrete operator is applied on the exact solution, it yields the same residual.

Iteration Error

The last two types of numerical errors we mention here are quite well understood. One of these comes from the common iterative techniques used to solve system of linear equations. Iterative methods aim at solving equations sequentially to reach a convergence point within a prespecified tolerance. This exact point is not typically reached, and the offset is known as the iteration error.

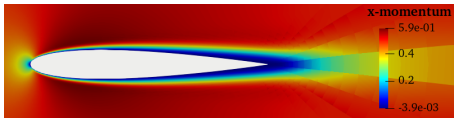
Round-Off Error

Another well-studied type of numerical errors is the round-off error, that is a result of finite-precision operations conducted at computer level. This error is generally quite small compared with discretization error.

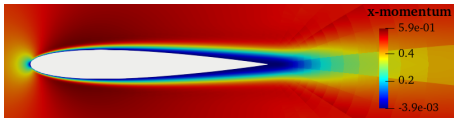
1.4.3 Possible Effects

Now that different types of errors involved in a simulation are introduced, we can present one of the main outcomes of these inaccuracies. As mentioned before, capturing all major flow features is an essential part of any simulation. These patterns have substantial effects on important aerodynamic parameters. It is proved that even one percent error in the calculated drag force on an airplane wing can have a significant impact on flight economics [19]. This clearly demonstrates the importance of accurate representation of all flow features in the flow field. In the following, we present a clear example where a simulation fails to capture one of these patterns.

Figure 1.6 represents the streamwise momentum solution around an airfoil, simulated using a 2^{nd}-order acccurate spatial discretization scheme (Figs. 1.6a and 1.6c) and a 4^{th}-order one (Figs. 1.6b and 1.6d). Detailed information on the mesh properties and solution techniques will be provided in Section 1.7. Here, the goal is to only observe the qualitative differences between the two simulations performed on the same object in the same flow conditions. By the first look at the overall flow field around the airfoil, the two solutions appear to be quite similar. However, a closer look at the trailing-edge area brings up a major concern in the lower-order simulation. In Figs. 1.6c and 1.6d, the colormap is manipulated so that only the negative velocity regions are illustrated. The pattern captured by the 4^{th}-order simulation is known as the separation bubble, which is the major flow feature in this case. It is revealed that the 2^{nd}-order solution fails to capture this important pattern. In this simple example, we have access to the higher-order simulation to compare the lower-order one to see whether it satisfies the goals of the simulation. In real-world applications, however, we generally have one simulation in hand, such as the 2^{nd}-order plot in Fig. 1.6a, and we need a way to evaluate it without having access to more accurate results. In many cases, even for the experts in aerodynamics it is impossible to make any decisive conclusions on the quality of the captured flow patterns merely based on a single simulation.

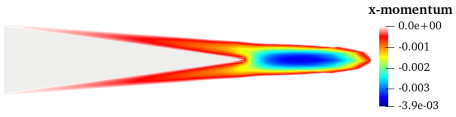


(a) 2^{nd}-order accurate field



(b) 4^{th}-order accurate field

(c) 2^{nd}-order accurate separation bubble



(d) 4^{th}-order accurate separation bubble

Figure 1.6: Flow simulation around a NACA 0012 airfoil. (Angle of Attack = 0 deg, Mach number = 0.5, Reynolds number = 5000)

In the following section, we present some common approaches to make sure that the simulation gives acceptable results. We will also discuss advantages and disadvantages linked to the use of each method.

1.5 Accuracy Improvement Methods

There have been many attempts in the past decades to tackle problems arise from inaccuracies in simulations, such as missing flow features presented in the previous section. In the following, we briefly mention a few of these efforts in minimizing numerical errors.

1.5.1 Uniform and Adaptive Grid Refinement

The classical approach to make sure that a simulation gives legit results is the grid convergence study. Figure 1.7 depicts major steps taken in this analysis, when performed on the flow field simulation around an airfoil. As shown in the figure, a coarse mesh is generated as the computational domain, and the specific parameter of interest, in this case the pressure drag coefficient, is computed. Then the mesh is refined, resulting in more control volumes and degrees of freedom, which gives a more accurate approximation to the parameter. This process repeats until the parameter reaches a converged value which does not change by mesh refinement anymore (such as point **P** in Fig. 1.7). This process establishes a grid convergence study, which has been one of the crucial steps in any CFD simulation. Mesh refinement could be performed in a uniform [20–22] or an adaptive [23–25] manner. The former refines spatial grid cells uniformly, usually by a factor of two, while adaptive methods systematically improve grid resolution in certain locations, resulting in a more efficient approach.

Although effective in simple two-dimensional (2-D) cases, this approach comes with an enormous computational cost for some three-dimensional (3-D) applications dealing with complex geometries (see, e.g., [26]). The challenges posed by grid refinement have been a topic of interest for many years. In their comprehensive studies, Diskin et al. performed a grid convergence

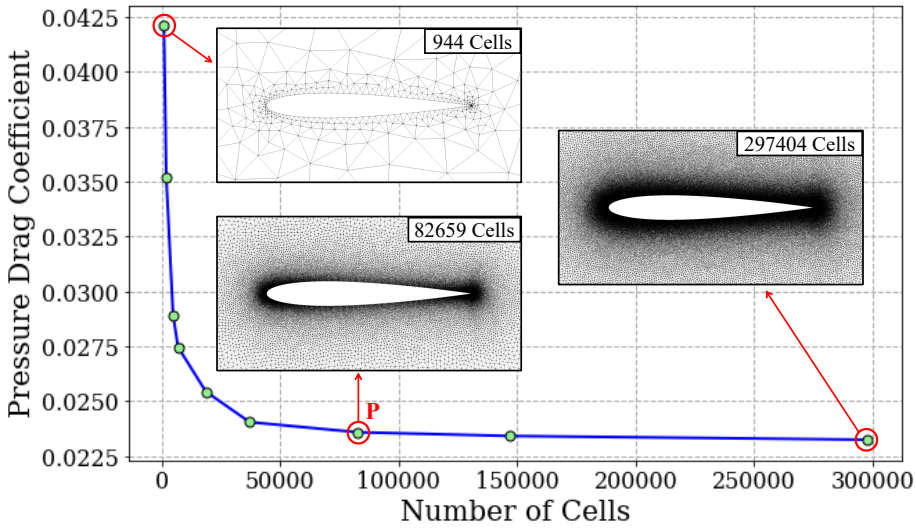


Figure 1.7: Grid convergence study for a 2-D flow simulation around a NACA 0012 airfoil.

study for some 2-D [27] and 3-D [28] benchmark flows, using three widely used CFD codes. They found that the convergence characteristics of flow parameters differed significantly between the grid families they used, and an asymptotic convergence order was not established. As a result, infinite-grid extrapolation could not be performed precisely to get an estimate for the exact value of flow parameters. Other studies (such as [29, 30]) draw the same conclusion, demonstrating that grid convergence is not an efficient way to improve the accuracy in CFD simulations.

1.5.2 Adjoint Error Estimation

Error estimators are an alternative approach to grid refinement, where the goal is to estimate the error in simulation and use that as a correcting term to compensate for the coarse-grid inaccuracy. A well-known error estimator is based on defining an auxiliary problem, called the adjoint [31–34], to compute the error in a solution functional such as lift or drag. Pierce and Giles [35] described the adjoint correction process in detail and employed a method to estimate functionals to an order that exceeded the discretiza-

tion order of accuracy. Adjoint error estimators can be even combined with grid refinement, like the work by Venditti and Darmofal where they implemented an adjoint-based grid adaptation for 2^{nd}-order 2-D finite volume discretization for both inviscid [36] and viscous [37] flows.

A common approach in CFD to implement adjoint correction is by calculating discrete adjoint. This variant is a simplified approximation to the more general continuous adjoint problem, which is comprehensively explained in [38]. In the discrete adjoint method, the available outputs of the simulation are directly used to solve for an adjoint solution, which then can be employed for estimating the error in one solution functional. Let us denote the Jacobian operator from Eq. (1.21) by $\mathbf{A}$ and the output solution vector from the simulation by $\vec{\bar{U}}$. This way, the adjoint vector $\vec{\psi}$ for a specific output functional J can be computed by:

$$\mathbf{A}^T \vec{\psi} = \frac{\partial J}{\partial \vec{\bar{U}}}^T \tag{1.24}$$

where superscript T denotes the matrix transpose. Having $\vec{\psi}$, the error δJ in the computed value for J can be approximated by taking the inner product of the adjoint and the truncation error $\vec{\tau}$:

$$\delta J = \left\langle \vec{\psi}, \vec{\tau} \right\rangle = \vec{\psi}^T \vec{\tau} \tag{1.25}$$

Guidelines on estimating truncation error in a simulation are provided in [39].

Although adjoint-based methods are more efficient compared to grid refinement techniques, they still add computational burden when dealing with multiple functionals. This is because for each functional of interest, a separate adjoint equation as Eq. (1.24) should be derived. The following subsection introduces an alternative error estimator that alleviates this limitation.

1.5.3 Error Transport Equation

A modern error estimator based on deriving an additional transport equation for discretization error has emerged in recent years [40–44]. This technique is called the Error Transport Equation (ETE), which aims at estimating the error in the field variables directly, instead of a single solution functional. In one instance, Yan and Ollivier-Gooch [45] applied the error transport method to unsteady compressible flows and they were able to obtain a higher-order accurate error estimate from a low-order solution. The main idea is to treat the discretization error as a physical property of the flow that follows the convection and the diffusion rules of the flow field. From this perspective, an additional transport equation for the discretization error can be derived and solved alongside the main flow equations.

To derive ETE, let u be the exact continuous solution to the primal flow equation presented in Eq. (1.1), and $\tilde{u}$ be the discrete solution from the simulation projected on the continuous space. We can define discretization error ε, mathematically as:

$$\varepsilon = u - \tilde{u} \tag{1.26}$$

If we substitute $u = \varepsilon + \tilde{u}$ back into Eq. (1.1), we get:

$$\partial_t(\varepsilon + \tilde{u}) + \nabla \cdot F(\varepsilon + \tilde{u}) = 0 \tag{1.27}$$

Rearranging both sides of this equation, and substituting truncation error as the resulting source term, ETE will be obtained as:

$$\partial_t \varepsilon + \nabla \cdot (F(\varepsilon + \tilde{u}) - F(\tilde{u})) = -\tau \tag{1.28}$$

This PDE governs the discretization error and can be solved using linearization techniques. A detailed description of this process is available in [46].

Both grid refinement and error estimation techniques are common in modern CFD applications. These are proved to be helpful in relatively simple 2-D simulations where the cost of additional computations is justifiable. However, for real-world applications, particularly complex 3-D cases, adding

more control volumes or equations in the simulation process could make it computationally inefficient or infeasible. In the next section, we will present our proposed approach to overcome challenges imposed by the current accuracy improvement methods in CFD, based on the advancements in artificial intelligence.

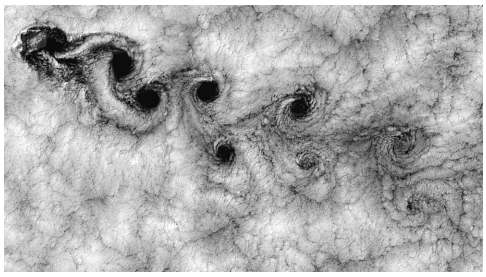
1.6 Proposed Data-Driven Approach

As mentioned in the previous section, although the current accuracy improvement methods are proved to be effective in some applications, they become extremely cumbersome in many cases involving complex 3-D geometries. These methods approach the problem purely from the numerical point of view, trying to improve calculated flow variables (grid refinement and ETE) or a functional (Adjoint), without directly analyzing resolved flow patterns in the flow field. The goal of this study is to provide an error analysis framework for CFD applications, where the focus is primarily devoted to the quality of major flow patterns captured in the flow field. With the advancements in data-driven methods and machine learning models, and also the ability of CFD techniques in producing huge amounts of insightful data, this task seems to be achievable more than ever. We will scratch the surface by explaining main ideas behind our proposed approach, and then we will delve into the details in the next chapters.

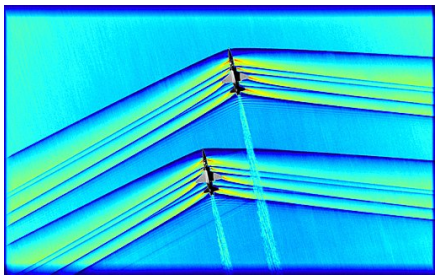
1.6.1 Pattern Recognition Perspective

Figure 1.8 provides some common patterns seen in the flow field around an object. Figure 1.8a depicts von Kármán vortex street, where swirling vortices are spotted behind a blunt body facing incoming flow due to the viscosity and pressure differences. Figure 1.8b shows shock waves that are discontinuities in the flow created near an object when traveling transonic or supersonic in a fluid domain. Capturing coherent flow structures, such as vortices and shock waves, is of utmost importance in CFD. The presence of numerical errors in simulations makes these important features prone

to be missed or poorly resolved, which will ultimately lead to inaccurate estimation of aerodynamic parameters. This could in turn result in a bad aerodynamic design, and sometimes, in disastrous events.



(a) Kármán vortex street caused by wind flowing around the Juan Fernández Islands. (Retrieved from public domain; captured by NASA in 1999.)



(b) Shock wave interaction between two aircraft. (Retrieved from public domain; captured by NASA in 2019.)

Figure 1.8: Common coherent structures seen in fluid flows.

In this thesis, we look at CFD simulations from the lens of pattern recognition, to try to predict whether major flow features are being missed, or insufficiently resolved for that matter. One of the main applications of machine learning and artificial intelligence has been face recognition [47–50]. Just like a *face detector* that recognizes different features on a face, we can develop a *flow feature detector* to scan for flow features in a fluid simulation. By doing that, we can set up an examination tool that expects to see certain features when analyzing specific flow fields, similar to a facial recognition system that looks for particular shapes on a human face. This way, this intelligent system can alert simulation engineers when a major pattern it anticipated to see is missing.

For image recognition (computer vision) applications, widely-used machine learning models are based on Convolutional Neural Networks (CNN) [51–53]. These algorithms work based on pixel analysis, where a filter moves across the image and detects features of interest. In CFD, however, we often deal with patterns ranging from very large to tiny cell-size length scales. Although it is possible to provide a CNN model with screenshots of the

flow field, it will not be effective for a detailed flow feature examination. Nonetheless, there exist studies conducted in recent years to incorporate CNN algorithms with CFD data to detect flow patterns [54, 55]. Such efforts have been successful in cases where the flow feature is large enough to be inspected by a human expert, and the goal was to speed-up and automate the workflow by removing the human-in-the-loop component of the postprocessing stage. CNN can detect most of the large patterns in the flow field, but it is likely to ignore smaller cell-size features that in fact have significant value in aerodynamics. To perform a thorough assessment of CFD results, we need a more quantitative tool able to analyze cell data, rather than a visual model such as CNN. This requires us to implement a more detailed detector to cover all length scales in the flow field.

In this study, Principal Component Analysis (PCA) is applied to the CFD data to decompose a flow field into a set of dominant modes. This step is the counterpart of the convolution operation in CNN where the filter is applied to the pixels to extract features. PCA, however, extracts the statistically dominant modes of the data set based on the variance it observes in cell data. It makes it an ideal tool since all important numerical information, even in the tiniest cells, is considered in the analysis. Since major patterns in the flow field make substantial contributions to the numerical data in each cell of the domain, it is expected that they show themselves in single or combinations of PCA modes. A formal introduction to PCA will be provided in Chapter 2. Here, we only point out that PCA creates a modal subspace from the high-dimensional space of CFD solutions, where all the information for a simulation can be presented concisely and much more efficiently in terms of extracted modes. These insightful modes can then be passed to a machine learning model for many purposes. The following subsection briefly explains how we have exploited the PCA modes for the purpose of this study, by the means of machine learning classification and regression.

1.6.2 Machine Learning Framework

As mentioned in the previous subsection, useful information about major patterns in a flow field is likely to be embedded in its PCA modes. Therefore, by using these modes as the input information to a machine learning model, we expect the learning algorithm to be able to find correlations in the PCA subspace to decide whether the simulation captures a particular flow feature. Even further, since these flow patterns have significant effect on aerodynamic parameters, using the same modes can be useful in predicting the error by which these parameters are calculated. The former model, where only a label is assigned to a simulation showing whether it misses a flow feature, is known as machine learning classification. The latter, where a continuous numerical value is predicted, is referred to as machine learning regression. These two types of machine learning models are introduced and explained in Chapters 3 and 4, respectively.

For now, we consider a machine learning model as a *black box*, which takes some input information and produces useful output. In later chapters, a comprehensive discussion of what happens inside this box will be provided. We can exhibit the overall workflow of our proposed learning system in Fig. 1.9. Looking at the figure, the first step in constructing such a feature detecting system is to gather simulations together as a big data set. This part of the workflow and the data set we used in this thesis will be covered in the next section. Then, PCA is applied on the data set of CFD simulations to extract dominant modes. These modes can be considered a compressed representation of the simulations, where each mode contains useful information about the flow field in a concise manner. The PCA modes are then passed into two machine learning models: a classifier, and a regressor. The classifier will predict, based on the PCA modes, whether a simulation misses a particular flow feature. The regressor, on the other hand, operates in the same information space to estimate the error in the calculated aerodynamic parameters to be used as a correction afterward.

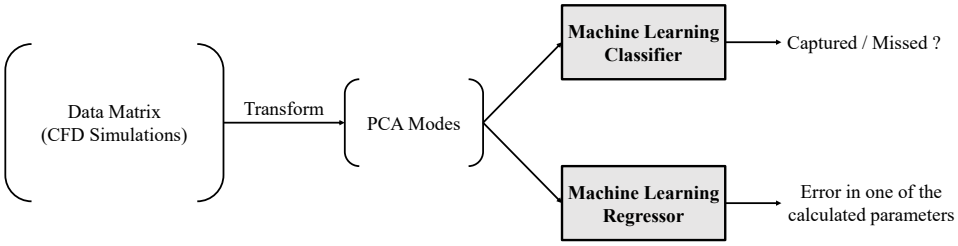


Figure 1.9: *Black box* representation of the proposed machine learning system.

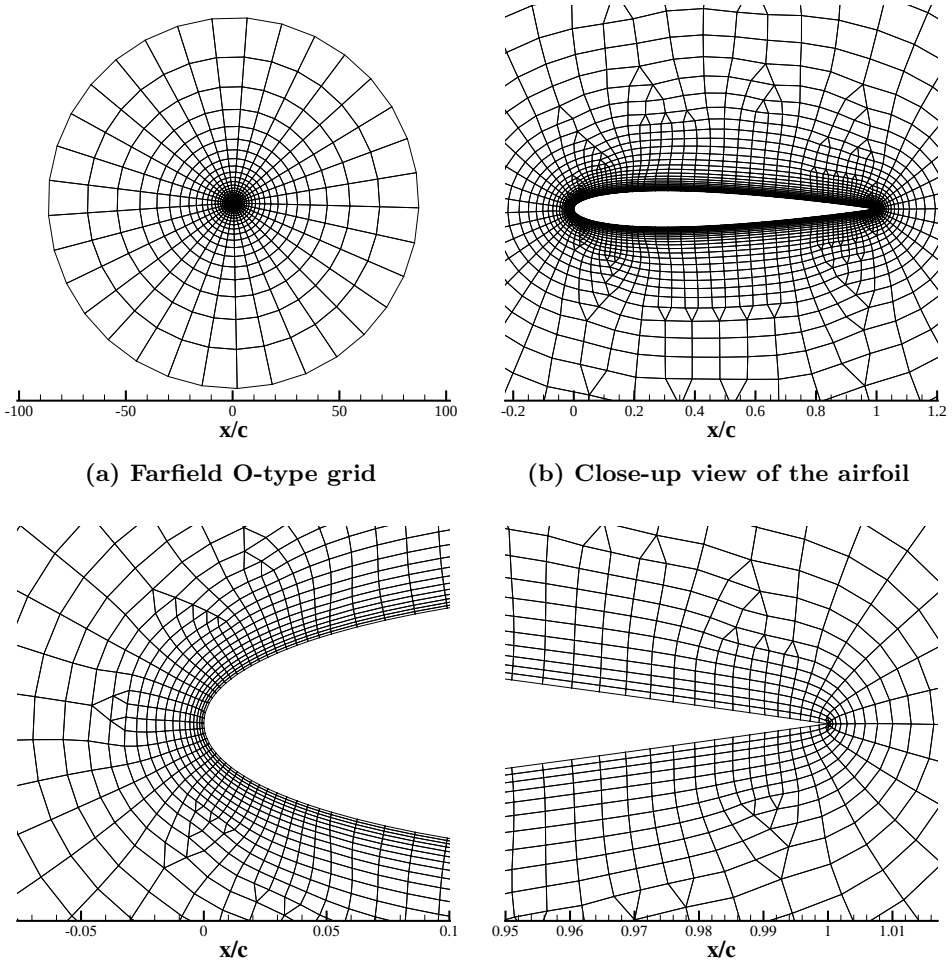
1.7 Test Case and Data Set

1.7.1 Test Case: Missing Flow Separation on NACA 0012

Flow past an airfoil is a canonical example in aerodynamics; thus, the test case in this study was chosen to be a simulated steady-state, compressible, subsonic, 2-D viscous flow, governed by the Navier-Stokes equations, around a NACA 0012 airfoil. When this particular body shape is placed in a flow with an Angle of Attack (AoA) of 0 deg, a Mach number (M) of 0.5, and a Reynolds number (Re) of 5000, a separation bubble is expected to be formed at the trailing-edge of the airfoil.

For this study, an unstructured coarse grid containing 4142 cells was generated using GRUMMP. As illustrated in Fig. 1.10, the grid was relatively coarse near the trailing-edge and inside the viscous boundary layer. The reason was to force the 2^{nd}-order accurate solution to miss the separation bubble in the specified flow conditions, whereas the 4^{th}-order one fully captured the separation region. The task was done by trial and error, testing multiple meshes to end up with a mesh that satisfied the goals of this study.

To solve the flow field, our in-house library (ANSLib) was used, and to handle the linear algebra operations, an external package called PETSC [56] was utilized. To obtain higher-order spatial accuracy for unstructured meshes, our solver uses the k-exact least-squares reconstruction method. For the advective fluxes, Roe's scheme [57] is implemented, whereas for the diffusive fluxes, the average of the reconstructed gradients with an additional jump term is used. The test case was solved two times; first, using a 2^{nd}-



(a) Farfield O-type grid

(b) Close-up view of the airfoil

(c) Leading-edge (10% of the chord)

(d) Trailing-edge (5% of the chord)

Figure 1.10: Unstructured mixed-element mesh around NACA 0012, containing a total of 4142 triangular and quadratic cells.

order accurate discretization scheme, and then, by a 4^{th}-order accurate one. The results are presented in Fig. 1.6 from Section 1.4, where the effect of inaccuracies in the simulation was discussed. As explained before, the lower-order simulation misses the main flow feature in this canonical case, which is the separation bubble, while the higher-order simulation captures it.

The purpose of this thesis is to develop two machine learning models

to detect the missing flow feature in the low-order simulation and predict the error in the drag coefficient, without having access to the higher-order solution for comparison. It should be noted that applying conventional accuracy improvement methods in this simple 2-D case is relatively simple and common. Our goal is to present a proof of concept, which could hopefully be generalized to more complex 3-D cases in the future. Therefore, although we train and test our intelligent model on a simple 2-D case, the target of this project is intended to be more challenging cases where the current improvement methods fail to perform well.

1.7.2 Raw Data Set

Any learning system, whether it is a human, an animal, or a computer, needs lots of data to be trained and understand patterns and correlations from the input space of information. To produce a full data set suitable for the purpose of this study, the following combinations of flow conditions were used:

- the angles of attack from -7 to $+7$ deg, with steps of 1 deg,
- the Mach numbers from 0.1 to 0.5, with steps of 0.1,
- and the Reynolds numbers from 3000 to 9000, with steps of 1000

that resulted in 525 unique flow conditions. Then, the governing flow equations were solved using 2^{nd}- and 4^{th}-order accurate discretization schemes, giving a total of 1050 solution vectors. For each unique case, the higher-order solution was assumed to be the exact solution (the ground truth), and the lower-order one was considered as the simulated flow. Among all the 2^{nd}-order simulations, 236 cases were either missing the separation or partially capturing it, compared to their 4^{th}-order counterparts.

To construct the data set, all the solution vectors are put together as columns of a large data matrix, as shown in Fig. 1.11. This gives a 16568 by 1050 matrix, where the dimensions represent the dimensionality of the solution vectors and the number of data samples, respectively. The large

number of values in each solution vector comes from the fact that each of the 4142 control volumes in the flow field contains 4 field variables that are solutions to the conservation of mass, momentum (we have two values coming out from this law, since it is a 2-D simulation), and energy, resulting in 16568 degrees of freedom for each vector. This data matrix is tagged *raw*, as it still needs some preprocessing operations before being prepared for PCA. These operations will be discussed in detail in the next chapter.

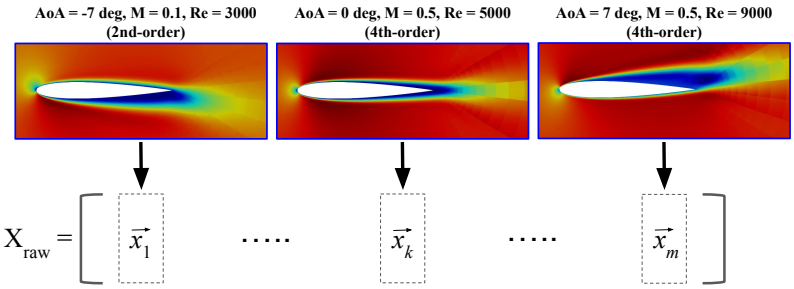


Figure 1.11: CFD solutions in different flow conditions around NACA 0012 gathered to create the *raw* 16568 by 1050 data set.

We have created a free open-source GitHub repository containing our data set, which can be accessed through: `https://github.com/APHedayat/missing-flow-feature-dataset.git`.

1.8 Outline

Figure 1.9 provides a convenient road-map for the rest of this thesis. Referring to this figure, the learning system we intend to develop consists of three main parts, namely PCA, classification, and regression. Chapter 2 introduces our modal decomposition method, where we create the basis space to train machine learning models for CFD data sets. Based on the extracted modes, we develop two intelligent models to identify missing flow separation and estimate the error in the drag coefficient in CFD simulations in Chapters 3 and 4, respectively. Finally, key outcomes of this study are discussed in Chapter 5, and guidelines for the future are provided. Moreover, additional information and results are presented in Appendices A to F.

Chapter 2

Principal Component Analysis of Aerodynamic Data

In this chapter, we introduce PCA and investigate its use in detecting CFD simulations with a missing separation bubble. PCA is a ubiquitous modal decomposition technique, based on Singular Value Decomposition (SVD), which is one of the most important tools in data analysis. PCA is labeled with a variety of terms in literature. In fluid mechanics, it is often known as Proper Orthogonal Decomposition (POD), while other terminologies such as Karhunen–Loève procedure and Hotelling analysis are common in other fields. This technique decomposes a data matrix, such as the one in Fig. 1.11, into linearly-independent basis functions, also called modes. The main objective of PCA is to find the most efficient set of modes, tailored to the specific data set, to obtain a low-rank approximation to the high-dimensional data. In other words, PCA extracts dominant patterns in a data set that capture most of the energy in the data. The extracted mode shapes can be used in a variety of applications, from dimensionality reduction methods [58–60], to pattern recognition techniques [61].

PCA methods were first, to the best of our knowledge, introduced by Pearson [62] and Hotelling [63], and then were presented as a tool for pattern recognition by Watanabe [64]. In the field of fluid mechanics, PCA was proposed by Lumley [65], where he was able to extract the coherent structures in a turbulent flow field. PCA has since been frequently used in a variety of applications to study fluid flows [66–69]. In their recent review

papers, Taira et al. [70, 71] provided a detailed overview of the applications of modal decomposition techniques, including PCA, in the field of fluid dynamics. The power of PCA in extracting dominant flow patterns makes it an attractive candidate to detect missing flow features, which is the goal of the present study.

Section 2.1 of this chapter presents a profound insight into what PCA actually does in practice, and why it is one of the key tools in data science. Then, Section 2.2 provides a brief overview of its mathematical foundation, and a very important preprocessing operation that should be performed on the data matrix is discussed. Following that, we apply PCA on our prepared data set in Section 2.3 to uncover the dominant modes of the CFD data. These modes are further studied to explore the underlying details and physics of the flow. Finally, Sections 2.4 and 2.5 describe our approach to isolating the mode responsible for the missing separation bubble.

2.1 Dimensionality Reduction

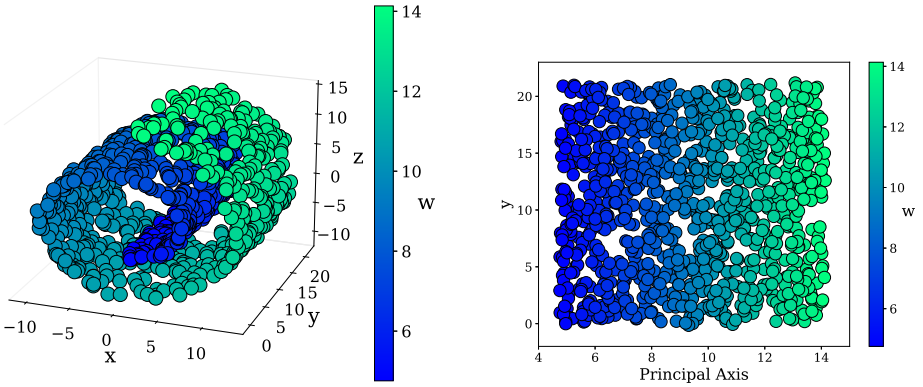
In all branches of science and engineering, data is being generated constantly. These data are typically high-dimensional as they have many degrees of freedom, which makes it impossible to gain insight solely based on the raw data. This problem is referred to as *the curse of dimensionality*, and there are multiple dimensionality reduction techniques such as Dynamic Mode Decomposition (DMD) for time-varying data sets, Linear Discriminant Analysis (LDA), and Autoencoders to handle it. The basic idea behind all these methods is to create a lower-dimensional subspace from the high-dimensional input data without significant information loss. The resulting subspace represents original data in a compact form, where all meaningful information is retained. Hence, operating in this subspace is much more efficient compared to analyzing the high-dimensional data directly.

In general, dimensionality reduction techniques can be divided into two major groups: feature selection, and feature extraction. Feature selection is the simplest approach one could take to decrease the complexity of the data.

In this approach, all individual components in the data vector are analyzed, either by a human expert or a computer, to discard less important entities. This way, only the most meaningful input information is kept, yielding a lower-dimensional data vector. When the number of entities in the original high-dimensional data vector is relatively low, this method can be utilized. We will show in later chapters how we used feature selection in our machine learning models, under the sections where we perform feature analysis.

For substantially higher-dimensional data sets with many degrees of freedom, feature selection becomes impractical as the number of components is beyond our capability to examine each individually. For these cases, which constitute almost all engineering data sets, another method called feature extraction, also known as feature projection, should be employed. This type of dimensionality reduction takes a statistical approach to map the high-dimensional data onto a lower-dimensional subspace. PCA falls into this category of dimensionality reduction techniques, where the directions along which the data samples show highest variance are extracted as a set of linearly-independent modes. These modes create an orthonormal basis, onto which data vectors in the data set can be projected. This action significantly reduces the dimensionality of the main data without losing much information. The process is comprehensively explained in the next section. An illustrative example of using PCA on a simple test case is provided in Fig. 2.1. This famous example is know as the *swiss roll*, where the data points are represented in the 3-D space of independent variables (x, y, z), and are colored by the dependent variable w (Fig. 2.1a). A close inspection reveals that the dependent variable can be represented by a single curved direction along the roll. Figure 2.1b demonstrates that PCA precisely detects this direction and unrolls the 3-D structure to represent the data set much more efficiently using a single principal axis.

In practice, both feature selection and feature extraction are applied to a high-dimensional data set to produce a much more efficient subspace for further analysis. First, a feature extraction technique such as PCA is implemented to give a lower-dimensional subspace from high-dimensional data with many degrees of freedom. Then, feature selection analysis can be



(a) Original data points

(b) Reduced-order representation after applying PCA

Figure 2.1: *Swiss roll* data set, demonstrating the power of PCA in dimensionality reduction.

applied on this low-dimensional subspace to create an even more efficient one.

CFD results are among the highest-dimensional data in science and engineering, which makes the use of dimensionality reduction techniques compulsory in these cases for data analysis. Accordingly, we apply PCA on the CFD data set presented in Fig. 1.11, to extract dominant modes and therefore substantially reduce the dimensionality and complexity of the data for our machine learning purposes. A detailed overview of PCA method is presented in the following section.

2.2 Methodology

Here, we present the mathematical foundation of PCA. Excellent overviews of this subject are available in the literature (refer to, e.g., [72]). Suppose that the data samples are stacked together as columns of a large data matrix, $\mathbf{X}_{\text{raw}} \in \mathbb{R}^{n \times m}$, as previously presented in Fig. 2.1:

$$\mathbf{X}_{\text{raw}} = \begin{bmatrix} | & & | \\ \mathbf{x}^{(1)} & \dots & \mathbf{x}^{(m)} \\ | & & | \end{bmatrix} \tag{2.1}$$

where m represents the number of samples and n defines the dimensionality of the data. Before it is ready for PCA, the raw data matrix needs a preprocessing step called centering, where the mean of the data vectors is subtracted from each column. This makes the results more interpretable since they are expressed as deviations from the mean solution. In fact, we often care only about off-mean fluctuations when studying fluid dynamics; turbulence models are good examples to show this tendency. The process of centering can be written mathematically as follows:

$$\mathbf{X} = \mathbf{X}_{\text{raw}} - \bar{\mathbf{x}} \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}_{1 \times m} \tag{2.2}$$

where $\bar{\mathbf{x}}$ is a column-vector representing the mean, and the second operation simply copies the mean data as columns of an n by m matrix.

Using *economy* SVD, which is the more efficient version of the classic SVD, the mean-subtracted data matrix can be decomposed into three matrices as follows:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_m \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_m \end{bmatrix} \begin{bmatrix} | & & | \\ \mathbf{v}_1 & \dots & \mathbf{v}_m \\ | & & | \end{bmatrix}^T \tag{2.3}$$

where the columns of $\mathbf{U} \in \mathbb{R}^{n \times m}$ and $\mathbf{V} \in \mathbb{R}^{m \times m}$ are the left and the right singular vectors of $\mathbf{X}$, respectively, and the superscript T denotes the matrix transpose. Moreover, $\mathbf{\Sigma} \in \mathbb{R}^{m \times m}$ is a non-negative diagonal matrix where the entries are the singular values, ordered hierarchically from the largest to the smallest. The columns of $\mathbf{U}$ and $\mathbf{V}$ follow the same rule and are arranged based on their energy content as defined by their corresponding

singular value. The left singular vectors are also called the modes or Principal Components (PCs) of the data set and span an orthonormal basis on which the data samples can be projected. In other words, these vectors capture the directions in the information space of $\mathbf{X}$ where data samples $\mathbf{x}^{(i)}$ show the highest variance. By putting decomposed data in the order described before, the most important directions are piled up as the first few columns of $\mathbf{U}$.

To obtain the left singular vectors, the eigensystem of the correlation matrix $\mathbf{X}\mathbf{X}^T$ can be solved:

$$\left[\mathbf{X}\mathbf{X}^T\right]\mathbf{U} = \mathbf{\Lambda}\mathbf{U} \tag{2.4}$$

In Eq. (2.4), the eigenvectors of $\mathbf{X}\mathbf{X}^T$ are the left singular vectors of the data matrix, and hence, are denoted by $\mathbf{U}$. Also, the singular values of $\mathbf{X}$ are related to the eigenvalues of $\mathbf{X}\mathbf{X}^T$ by:

$$\sigma_i = \sqrt{\lambda_i}$$

The same analogy can be made for the right singular vectors of the data matrix by solving the eigensystem of $\mathbf{X}^T\mathbf{X}$. This analysis points out the nature of SVD, and also demonstrates why the first few PCA modes are the most statistically important factors in describing the data set.

The strength of each mean-subtracted data sample, $\mathbf{x}^{(i)}$, in the direction of each PCA mode, $\mathbf{u}_k$, is computed by taking the inner product of the two vectors,

$$\alpha_{ik} = \left\langle \mathbf{x}^{(i)}, \mathbf{u}_k \right\rangle = {\mathbf{x}^{(i)}}^T \mathbf{u}_k \tag{2.5}$$

where α_{ik} is called the expansion coefficient that indicates the amount of energy in the i^{th} data sample captured by the k^{th} mode. This action gives the projected solution vector, $\mathbf{x}^{(i)}$, on the direction defined by the k^{th} PCA mode. Orthonormality of the PCA modes (i.e., $\langle \mathbf{u}_i, \mathbf{u}_j \rangle = \delta_{ij}$) enables us to reconstruct any data sample as a weighted summation over all the modes:

$$\mathbf{x}^{(i)} = \sum_{j=1}^{m} (\alpha_{ij}\mathbf{u}_j) \tag{2.6}$$

In most cases, using all the modes to represent a data sample is not efficient. Furthermore, the purpose of PCA is to provide an optimal low-dimensional basis for the high-dimensional data. In that regard, Eckart and Young [73] shared that the best rank-r approximation to any mean-subtracted sample, $\mathbf{x}^{(i)}$, in a least-square sense, can be written as the weighted summation over the first r modes:

$$\mathbf{x}^{(i)} \approx \sum_{j=1}^{r} (\alpha_{ij}\mathbf{u}_j) \tag{2.7}$$

where $r \ll m$. The choice of r is usually based on the amount of energy captured by using the first r modes, which will be discussed in detail in the results section. In simple terms, Eq. (2.7) picks the leading directions from $\mathbf{U}$ to create a remarkably efficient subspace to analyze the high-dimensional data in $\mathbf{X}$. The Eckart-Young theorem is the basis for all dimensionality reduction techniques based on PCA, where the level of complexity of an n-dimensional problem significantly reduces to r-dimensions, where $r \ll n$.

In this study, the data samples are the solution vectors of multiple CFD cases, defined in the previous chapter. CFD simulations are perfect examples of high-dimensional problems with many degrees of freedom, where multiple flow variables are assigned to numerous control volumes. In such cases, using PCA could substantially decrease the level of complexity from thousands or even millions of degrees of freedom, to only a few modes. This dimensionality reduction, as well as the power of PCA in extracting the statistically dominant modes of the solution, were the motives of the current study to develop an intelligent *flow feature detector*.

2.3 Results: Modal Decomposition

In this section, we present the results of applying PCA on our CFD data set presented in Fig. 1.11 after going through preprocessing step in Eq. (2.2), and thoroughly investigate the output mode shapes to uncover hidden information embedded in each mode. It is worth noting that our initial attempt for modal decomposition involved a different technique, called eigendecomposition, where we faced conditioning issues. A full discussion on this method and its associated limitations is provided in Appendix A.

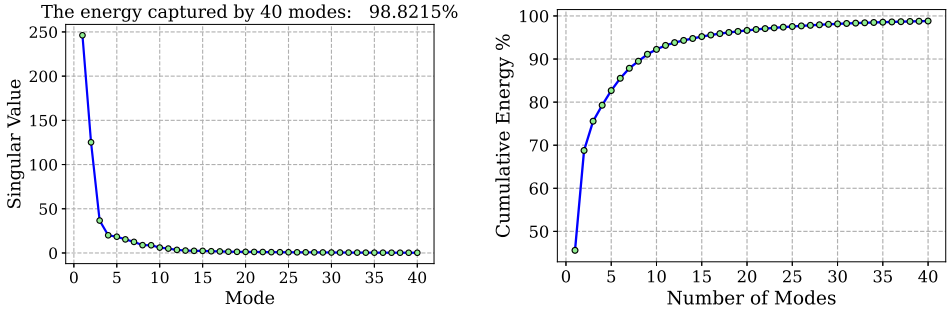
2.3.1 Energy Plot and Cut-Off Mode

The singular values, σ_k, stored in the diagonal matrix $\mathbf{\Sigma}$, defined in Eq. (2.3), can be interpreted as the amount of energy each mode contains. Figure 2.2a shows the evolution of the singular values associated with each mode of the solutions in our data set. The results clearly show that the first modes contain most of the energy in the whole data set, and the singular values (energy contents) drop significantly as the mode number is increased, as the theory of SVD suggests. The main usage of this plot is to decide how many modes should be preserved to capture as much energy as possible to describe the data set with acceptable accuracy (Eckart-Young theorem, Eq. (2.7)). In that regard, one popular method is to look at the *elbow* in Fig. 2.2a, after which the energy contents become nearly negligible. Applying this methodology in our case, it suggests that any number around 4 to 10 modes would be an appropriate choice. However, as the main purpose of this section is to identify the mode responsible for the missing separation bubble, and not merely dimensionality reduction, we should be more cautious when performing rank-r approximation, defined by Eq. (2.7). A better option, as mentioned by Taira et al. in their review paper [70], is to retain r modes of the solution such that:

$$\frac{\sum_{j=1}^{r} \sigma_j}{\sum_{j=1}^{m} \sigma_j} \approx 0.99 \tag{2.8}$$

In other words, we keep as many modes as to capture approximately 99% of the information in the data, and discard remaining less informative modes.

For this purpose, a more illustrative plot is the cumulative energy, presented in Fig. 2.2b. This plot accounts for the quality of the data reconstruction, as the number of PCA modes increases. As shown in the figure, using only 5 modes, we capture around 83% of the information embedded in the data. Although it might sound impressive to describe high-dimensional CFD data using only a few modes by such a good accuracy, this could also increase the chance of missing important details of the flow, particularly the separation region, in the remaining truncated modes. Using the first 40 modes of the solution, on the other hand, we capture almost 99% of the energy in the data set, which satisfies the criterion stated by Eq. (2.8). Therefore, in this study we kept and analyzed only the first 40 modes obtained by applying PCA on our data set. This clearly demonstrates the ability of PCA to significantly reduce the complexity of the data set; In our case, using PCA we represent each 16568-dimensional CFD simulation as a data point in a 40-dimensional PCA subspace, which is nearly 400 times more compressed. It provides a considerably more efficient framework to explore different aspects of such a complicated nonlinear dynamical system.



(a) Singular values for each mode of the solution

(b) The total energy captured in the data set as a function of the number of modes considered

Figure 2.2: The evolution of the singular values of the data set and the energy level of rank-r approximation given by Eq. (2.7).

2.3.2 Physical Interpretation

Now, we narrow down our investigation to understand the physical significance of the dominant PCA modes. All the first 40 solution modes obtained by applying PCA (Eq. (2.3)) on the data matrix (shown in Fig. 1.11) are presented in Fig. B.1 as part of Appendix B. The plots represent the modes associated with the streamwise momentum solution, since this flow variable better depicts the separation region. Scanning through the mode shapes, each mode represents a certain shape in the flow field, and there is a lot of information embedded in these patterns. It is helpful to think of each mode as a source of velocity and see which regions of the overall velocity field they affect when added together to reconstruct the simulation. This yields a physical insight behind each mode, which can be beneficial for many applications. Using this examination, we observe the following:

- The first mode appears to represent the general structure of the velocity field. Since it is primarily concerned with the overall velocity, it probably represents the effect of the Mach number. This mode is illustrated in Fig. 2.3.

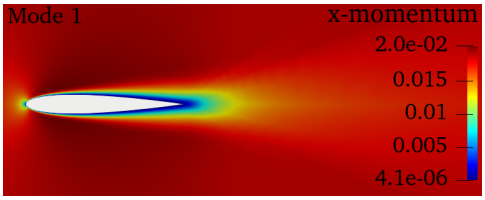


Figure 2.3: First PCA mode of the CFD data set in Fig. 1.11, representing the effects of the Mach number.

- The second PCA mode shows a huge difference between the upper and the lower regions of the airfoil, suggesting that it is the main source of the pressure difference between the surfaces of the airfoil, which ultimately produces lift. The major source of these effects for a symmetric airfoil is the angle of attack, and therefore we can easily link the second mode to this flow condition. The plot of the second PCA mode is represented by Fig. 2.4.

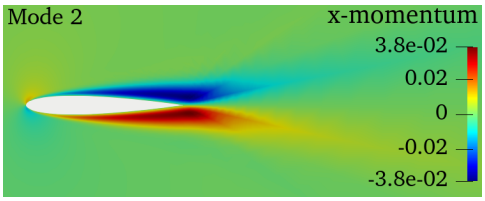


Figure 2.4: Second PCA mode of the CFD data set in Fig. 1.11, representing the effects of the angle of attack.

- Other interesting mode shapes are the fourth and the sixth ones, where the boundary layer is mostly affected. This demonstrates that these two modes contain information about the effects of viscosity and the Reynolds number. These mode shapes are pictured in Fig. 2.5.

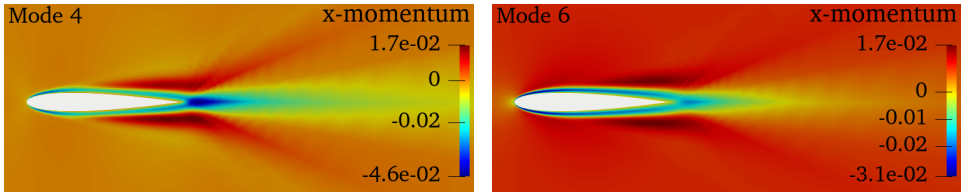


Figure 2.5: Fourth (left) and sixth (right) PCA modes of the CFD data set in Fig. 1.11, representing the effects of the Reynolds number.

- Other conclusions can be drawn using the same inspection method, although as the mode shapes become more complicated, physical interpretation becomes more difficult.

More complex flow features, such as separation bubble, are likely to be stored in more complicated modes, as mentioned in the last point above. Their effects can even be represented by a mixture of multiple modes. It suggests that visual investigation is not very effective for our purposes, and a different examination approach must be taken.

2.4 Results: Solution Projection

The goal of the PCA decomposition in this study is to identify the mode responsible for the missing separation bubbles observed in some of the 2^{nd}-

order solution vectors. Therefore, we look for a mode where the 2^{nd}- and the 4^{th}-order solution vectors behave in a completely different manner. Hence, we develop our alternative modal examination tool based on this discrepancy between the low- and the high-order simulation vector fields. As a starting point, all solution vectors can be projected onto the retained PCA modes of the data set, by taking the inner product mentioned in Eq. (2.5). The best way to picture the behavior of the 2^{nd}- and the 4^{th}-order simulations when going through this mapping transformation is by plotting those and look for major differences. For this purpose, the plots in Fig. C.1 in Appendix C are provided, where the y-axis shows the expansion coefficient, resulting from the projection of each solution vector on each mode, and the x-axis represents the index associated to each combination of flow conditions, ranging from 1 to 525. These indices are of least importance, and the sole purpose of defining such a numbering system is to help visualize the expansion coefficients for each mode in a 2-D plane. As shown in Fig. C.1, both curves of the 2^{nd}- and the 4^{th}-order solutions have been plotted for each mode. By doing so, the difference between the low- and the high-order solutions can be easily identified for each mode. It is helpful to keep in mind that the Reynolds number varies faster than the other flow conditions in these plots, then comes the Mach number, and finally the angle of attack. Looking at the plots, we mention the following notes:

- Leading PCA modes show exactly identical behaviors for the 2^{nd}- and the 4^{th}-order solutions, suggesting that these modes account for the general flow conditions, such as the angle of attack, the Mach number, and the Reynolds number. This is consistent with our previous inspection of the mode shapes. Figure 2.6 shows a few examples of such modes where the expansion coefficients for the lower- and the higher-order simulations coincide.

- Analyzing the remaining modes, although some of them demonstrate slight discrepancies between the curves, their overall behavior seems to be qualitatively identical. It appears that the 16^{th} mode is the bold one in representing most of the difference between the 2^{nd}- and the

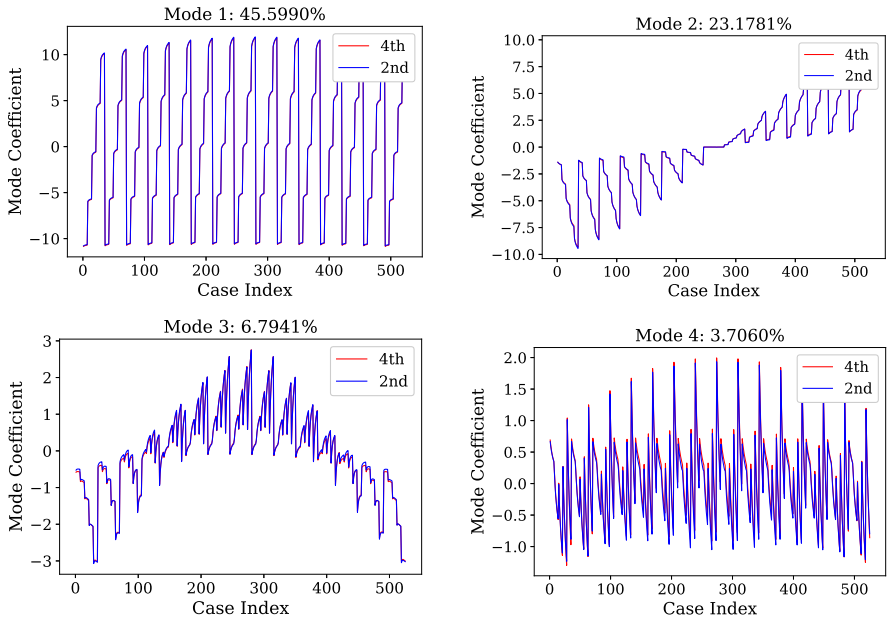


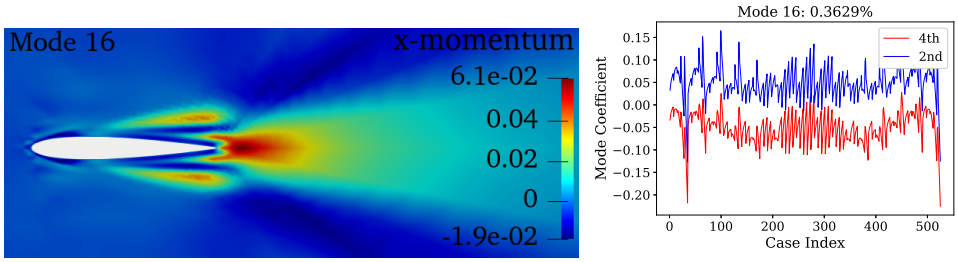
Figure 2.6: Expansion coefficients for the 2^{nd}- and the 4^{th}-order simulations, plotted for the four leading modes in the PCA subspace.

4^{th}-order solutions. It shows a huge gap between the low- and the higher-order curves, indicating that the 16^{th} mode contains much of the information about the order of accuracy of the solution. Since missing major flow features is a direct result of the order of accuracy, the 16^{th} mode is possibly the main source of the missing separation bubble observed in some of our cases.

- Some of the subsequent modes also show major differences between the curves; however, they contain less energy than the 16^{th} mode because of the hierarchical structure of SVD.

Consequently, the 16^{th} PCA mode of the data set is the foremost candidate to represent the missing separation bubble. Figure 2.7 depicts both the mode shape and the expansion coefficients of the solution vectors associated with the 16^{th} PCA mode side-by-side. It can be seen that the 16^{th} mode indeed recognizes the separation region (the dark red area near the trailing-

edge of the airfoil, which suggests positive streamwise velocity). Also, the fact that the 4^{th}-order expansion coefficients for this mode are mostly negative suggests that this mode tries to create a negative velocity bubble near the trailing-edge, which is one of the characteristics of the separation bubble; whereas, the 2^{nd}-order expansion coefficients for this mode are mainly positive, which indicates that the 16^{th} PCA mode prevents the separation bubble to be formed in some 2^{nd}-order solutions by injecting positive velocity into the trailing-edge region.



(a) Mode shape **(b) Expansion coefficients**

Figure 2.7: x-momentum component of the 16^{th} PCA mode alongside the expansion coefficients for the 2^{nd}- and the 4^{th}-order solution vectors in this mode, calculated from Eq. (2.5).

2.5 Results: Separation Bubble Reconstruction

To assess our findings in the previous section, we present the solution buildups for a test case that we already know misses the flow separation when solved to 2^{nd}-order accuracy. This is done by adding the modes weighted by appropriate expansion coefficients incrementally (using Eq. (2.7)), starting from the first mode, all the way to the 40^{th} one. The goal is to see how the solutions evolve as the PCA modes are added bit by bit. Based on the results of the previous section, we expect to see different behaviors between the 2^{nd}- and the 4^{th}-order accurate solution buildups regarding the creation of the separation bubble when adding the 16^{th} mode. A side-by-side comparison of the solution buildups for the low- and the high-order streamwise momentum fields are shown in Fig. 2.8, where the flow field is scaled and the colormap

is manipulated so that only the negative velocities are shown. By definition, these negative velocity regions illustrate the separation bubble, and we aim to only analyze the creation of these patterns. The results show that:

- Although minor differences exist between the modes as they reconstruct the solution, they all behave qualitatively the same before adding the 16^{th} mode. In other words, when a particular mode generates separation region at the trailing-edge for the higher-order simulation, it acts the same way on its low-order counterpart, although its effects can quantitatively differ.

- When the 16^{th} mode is taken into account, the separation region is reinforced on the 4^{th}-order plot, whereas it has no effect on the 2^{nd}-order one. By very close inspection, we notice that this mode even tries to reduce the intensity of the weak separation region in the low-order simulation. This is consistent with our observations in Fig. 2.7.

- The subsequent modes have minor influence on the separation, and no major change in the pattern of the separation bubble is seen after the 16^{th} mode. It confirms our previous observation that the 16^{th} PCA mode is the source of the missing separation bubble.

2.6 Summary

This chapter presented the results of PCA when performed on our CFD data set. We discussed the dimensionality reduction aspect of this modal decomposition technique, and how it facilitates our investigations. We also observed that the PCA modes are physically meaningful, providing a great framework to study CFD simulations from different perspectives. We were also able to mark a single mode that may be primarily responsible for the missing flow separation for the deficient cases in the data set. We can take advantage of PCA and all the useful information it provides in the development of our *flow feature detector*, which is the topic of the next chapter.

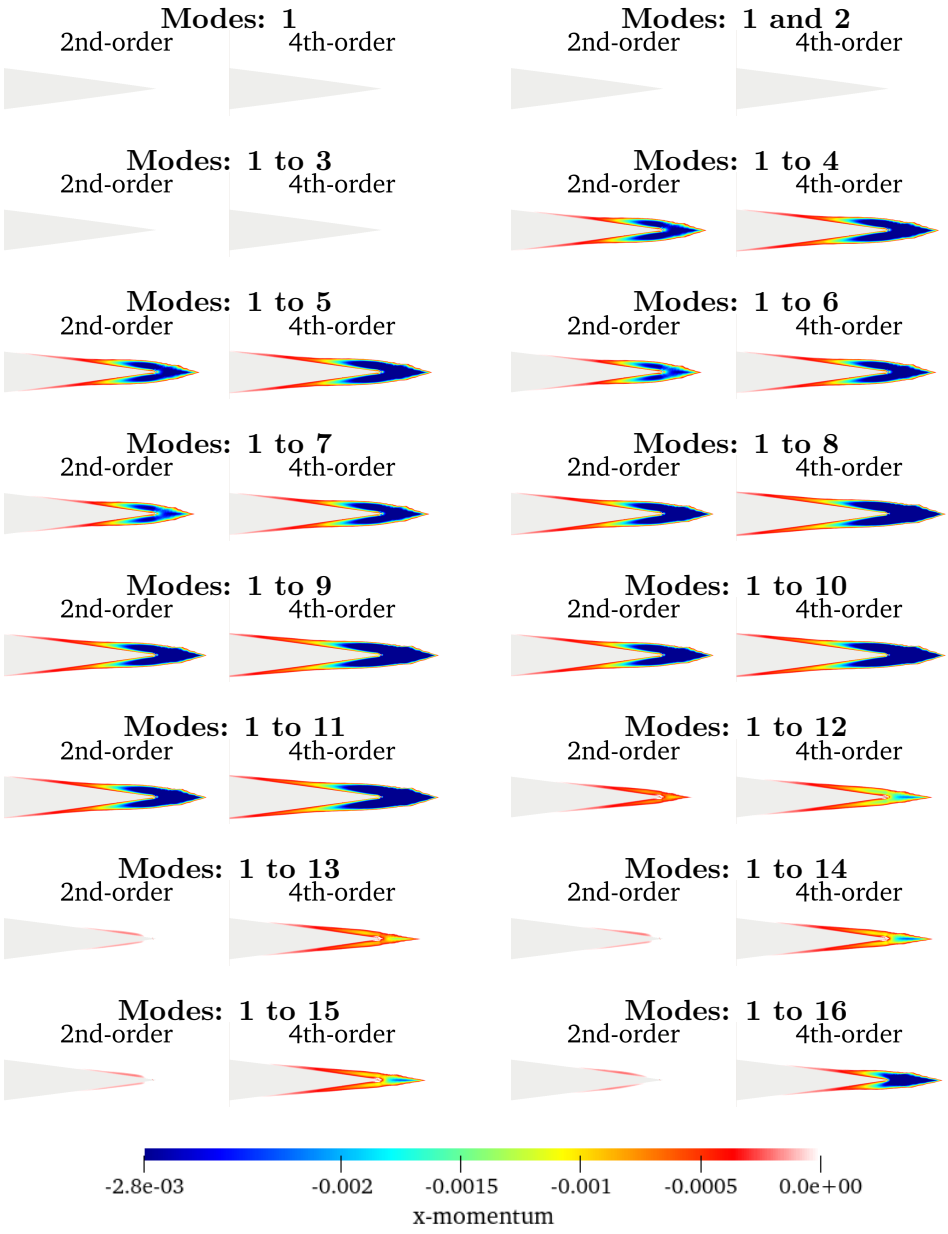


Figure 2.8: Solution buildup based on the PCA modes for the simulated flow around NACA 0012 at AoA = 0 deg, M = 0.5, and Re = 5000. Each portrait is a close-up view of the trailing-edge area (10% of total chord). The 2^{nd}- and the 4^{th}-order buildups are shown on the left and the right side of each portrait, respectively.

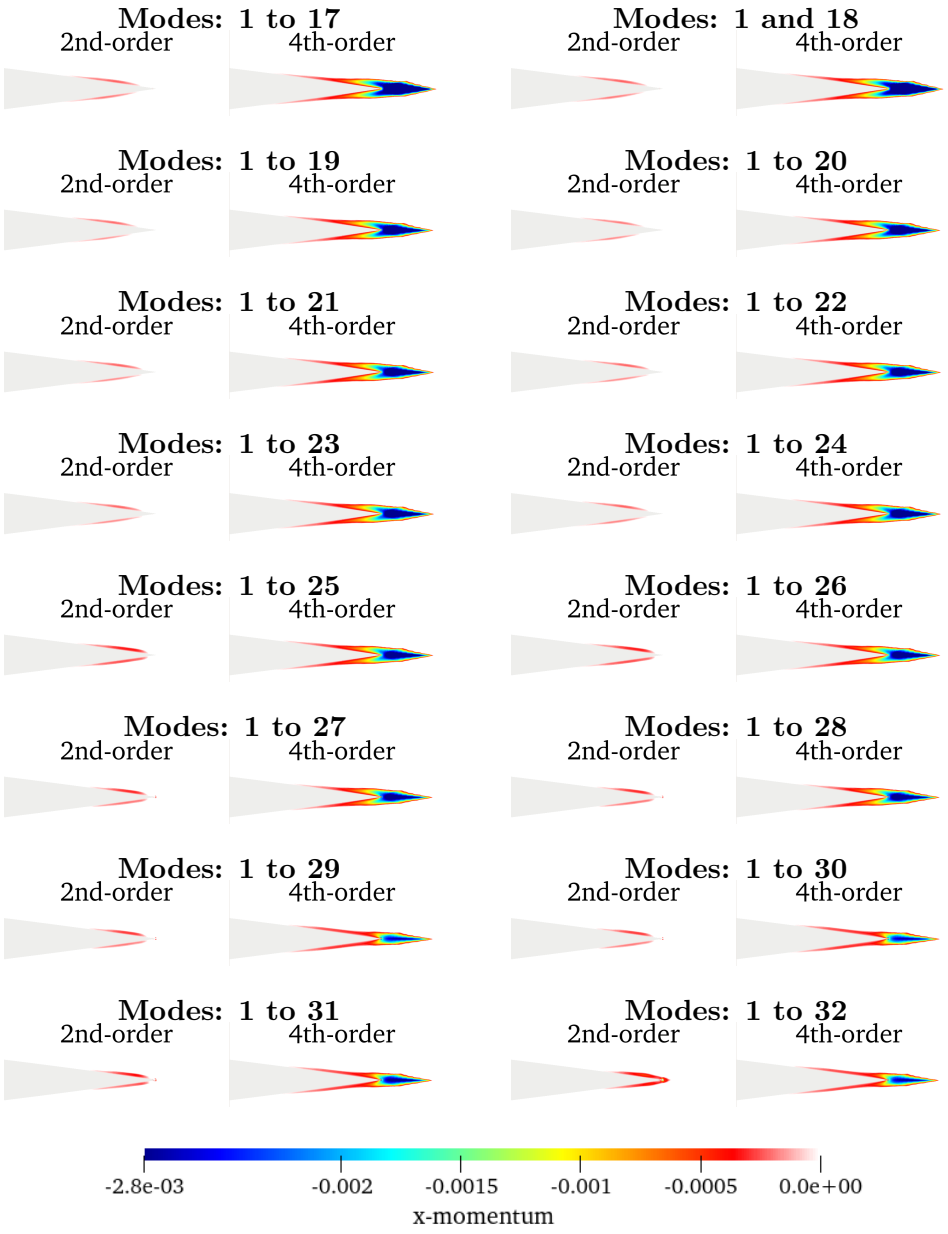


Figure 2.8: Solution buildup based on the PCA modes for the simulated flow around NACA 0012 at AoA = 0 deg, M = 0.5, and Re = 5000. Each portrait is a close-up view of the trailing-edge area (10% of total chord). The 2^{nd}- and the 4^{th}-order buildups are shown on the left and the right side of each portrait, respectively. (cont.)

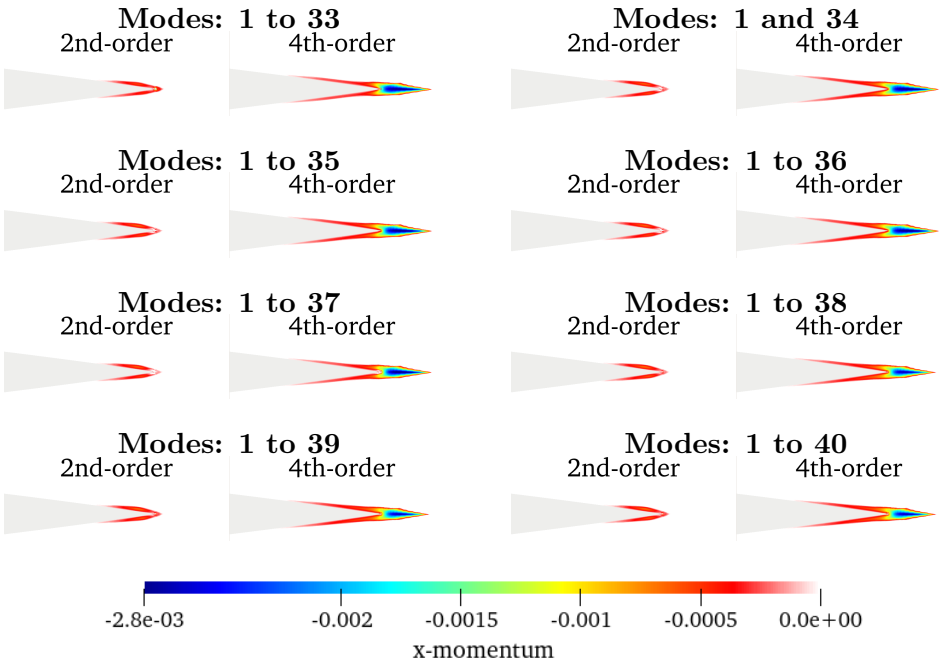


Figure 2.8: Solution buildup based on the PCA modes for the simulated flow around NACA 0012 at AoA = 0 deg, M = 0.5, and Re = 5000. Each portrait is a close-up view of the trailing-edge area (10% of total chord). The 2^{nd}- and the 4^{th}-order buildups are shown on the left and the right side of each portrait, respectively. (cont.)

Chapter 3

Separation Identification

In this chapter, we present our approach to developing an intelligent model to detect an under-resolved flow feature in CFD simulations. One of the primary applications of PCA is to preprocess the data to create an optimal subspace suitable for machine learning purposes. This is because PCA extracts the principal axes along which the data samples show the highest variance. In turn, it is much easier for machine learning algorithms to operate in such spaces where different data points are statistically more distinct, whereas the similar ones are clustered together. In addition, the power of PCA in extracting the mode responsible for the missing separation, which was proved to be the 16^{th} one in the previous chapter, suggests that the deficient cases could be identified by studying their 16^{th} expansion coefficient. These two points encourage the development of a machine learning classifier, which takes the expansion coefficients as input and detects the cases where the separation bubble is either missed, or partially captured.

The following sections provide a brief overview of how machine learning classifiers work, and how we implemented a supervised binary classifier for the purpose of this study. In Section 3.1, we highlight important aspects of machine learning classification in general, and introduce well known algorithms in this area. Sections 3.2 and 3.3 explain our logic in selecting input and output data for our machine learning classifier. Also, challenges related to imbalanced data sets are discussed in Section 3.3. Guidelines on evaluating classification systems are provided in Section 3.4, and parameter tuning is covered in Section 3.5. Overall steps taken to develop our classifier are described in Section 3.6, and the results of entering our CFD data matrix of Fig. 1.11 into the developed pipeline are presented in Sections 3.7 to 3.13.

3.1 Machine Learning Classification

A general definition for machine learning is provided by Samuel [74] as follows:

> *"Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed."*

and a more engineering description is given by Mitchel [75]:

> *"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."*

These definitions indicate that a machine learning model learns from previous observations to accurately act when applied to new instances. This process is not controlled by a human expert, in the sense that no direct command is given to the model, and the algorithm intelligently weighs related input information to make an accurate output prediction. In general, machine learning tasks are divided into two major groups: classification and regression. The current chapter explains the development of a classifier, while the next chapter will deal with a regression problem.

A machine learning classifier is a powerful algorithm that detects patterns and correlations in a collection of information, called features, to predict the class to which each data sample belongs. If the learning process is guided, meaning that the classes, also know as the labels, are already provided to the algorithm as a benchmark, it will be called a supervised classifier. In this study, we deal with a supervised binary classifier with two label outputs: captured (false, or 0), or missed (true, or 1). In a machine learning process, the full data set (such as the one we presented in Fig. 1.11) is divided into the training and the test sets, usually containing 80% and 20% of the data, respectively. The model is trained on the training set, to find all relative patterns and correlations between the features and labels, and is evaluated on the test set.

So far, a machine learning model was treated as a *black box*. Now, we turn our attention to the operations inside this box to correctly assign a label

to each data sample in the data set. In the following, we briefly introduce some common algorithms used for classification systems. For a detailed overview of different aspects of machine learning and deep learning, readers can refer to [76, 77].

3.1.1 Logistic Regression

Logistic Regression is a probability-based classification model, which essentially measures the probability that a data sample belongs to a class, and assigns labels accordingly. Let $\mathbf{x}^{(i)}$ be the vector of input features for the i^{th} data sample; the Logistic Regression algorithm operates based on a weighted summation over all elements of this vector as follows:

$$h_\theta^{(i)}(\mathbf{x}^{(i)}) = \theta_0 + \sum_{j=1} \theta_j x_j^{(i)} = \boldsymbol{\theta} \cdot \mathbf{x}^{(i)} \tag{3.1}$$

In the vector representation, we have added 1 at the beginning of $\mathbf{x}^{(i)}$ to properly take θ_0 into account. Probabilities are computed by applying a sigmoid function, defined by:

$$\sigma(t) = \frac{1}{1 + exp(-t)} \tag{3.2}$$

on Eq. (3.1), giving:

$$\hat{p}^{(i)}(\mathbf{x}^{(i)}) = \sigma\left(\boldsymbol{\theta} \cdot \mathbf{x}^{(i)}\right) \tag{3.3}$$

Equation (3.3) yields a value between 0 and 1, based on which the following labels are assigned:

$$\hat{y}^{(i)} = \begin{cases} 0 & \text{if} \quad \hat{p}^{(i)} < 0.5 \\ 1 & \text{if} \quad \hat{p}^{(i)} \geq 0.5 \end{cases} \tag{3.4}$$

The objective of Logistic Regression is to set $\boldsymbol{\theta}$ in Eq. (3.3) through a supervised learning on the training set to minimize prediction errors. It is

achieved by defining a cost function $J(\boldsymbol{\theta})$:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(\hat{p}^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-\hat{p}^{(i)}\right)\right] \tag{3.5}$$

and minimizing it using an optimization method, such as Gradient Descent (GD). In Eq. (3.5), $y^{(i)}$ is the actual label of the i^{th} data sample, and $\hat{p}^{(i)}$ is its predicted probability. Also, m represents the total number of the training instances. This function is convex, and therefore GD is guaranteed to find the global minimum.

3.1.2 Decision Trees

Decision Trees can be pictured as nested conditioning statements we use to program computers; the only difference is that the conditions are set by the algorithm itself, rather than being hard-coded as direct commands. Even so, developers have control over hyperparameters, and by changing these values the overall performance can be improved. These tunable parameters define the main structure of the Decision Tree, such as the maximum depth of the nested conditions. Individual conditions are intelligently calibrated by the machine learning model to accurately predict labels. The skeleton of a Decision Tree is depicted in Fig. 3.1, starting from the parent condition to its child nodes.

Decision Trees learn based on an impurity measure known as Gini, which is computed for each conditioning node. Let n_i be the number of total data samples entering the i^{th} conditioning node. Then, the following metric can be defined:

$$G_i = 1 - \sum_{k}\left(\frac{m_{i,k}}{n_i}\right)^2 \tag{3.6}$$

where $m_{i,k}$ is the number of instances belonging to the k^{th} label. The smaller G_i is, the purer data samples are in that specific conditioning node. We can use this metric to successively split the training set into two subsets as follows:

- Having (x, t_x) as a pair of one of the input features and a splitting

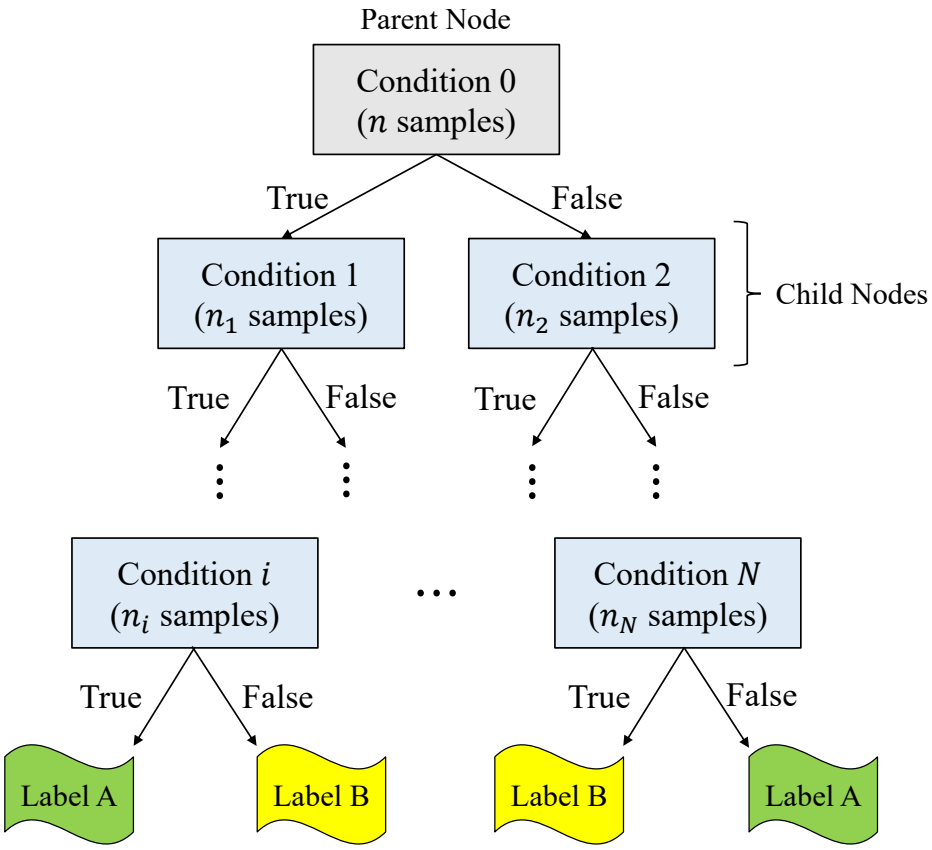


Figure 3.1: Overall structure of a binary Decision Tree classifier, consisting of multiple nested conditioning nodes to predict the output labels.

threshold, respectively, we split the whole training set into two left and right subsets.

- From Gini impurity function defined in Eq. (3.6), we construct the following cost function to be minimized:

$$J\left(x, t_x\right) = \frac{n_{\text{left}}}{n_{\text{total}}} G_{\text{left}} + \frac{n_{\text{right}}}{n_{\text{total}}} G_{\text{right}} \tag{3.7}$$

The algorithm loops through all input features and different thresholds to find a split that yields purest subsets. This process is essentially different than what Logistic Regression does by using optimization

techniques.

- This process is repeated for each subset, until either the maximum depth of the tree controlled by the user is reached, or the subsets cannot be subdivided into two purer subsets.

When the Decision Tree is created, the probability of each data sample in the i^{th} conditioning node belonging to the k^{th} label can be computed by the ratios $\frac{m_{i,k}}{n_i}$ defined in Eq. (3.6), and the labels can be distributed accordingly.

3.1.3 Random Forest Classification

The Random Forest algorithm is among the most powerful machine learning models one could use. It is based on a technique called the Ensemble Learning, where multiple machine learning models are mixed together to predict the output. In the case of Random Forest, multiple Decision Trees (shown in Fig. 3.1) are trained on distinct random subsets of the training set (hence the name *forest*). To predict the label, the data sample is passed to each of these models, and the label that gets the most votes will be assigned to the sample. In most cases, the Random Forest algorithm performs better than a single Decision Tree model, since it introduces more randomness and flexibility in growing trees.

Random Forest inherits all properties linked to Decision Trees, such as the ability to calculate probabilities. In addition, Random Forest makes meaningful calculations based on impurity values to assign an importance weight to each input feature. Having this information can help us reduce the dimensionality even further by discarding less important input values. This process is a type of feature selection, which was explained in Chapter 2.

3.1.4 Neural Networks and Deep Learning for Classification

Neural networks are our interpretation of how the brain works. The human brain is an impressive collection of neurons that work closely together

to process input information and make decisions accordingly. Figure 3.2 represents the structure of artificial neural networks we use to mimic the functionality of the brain.

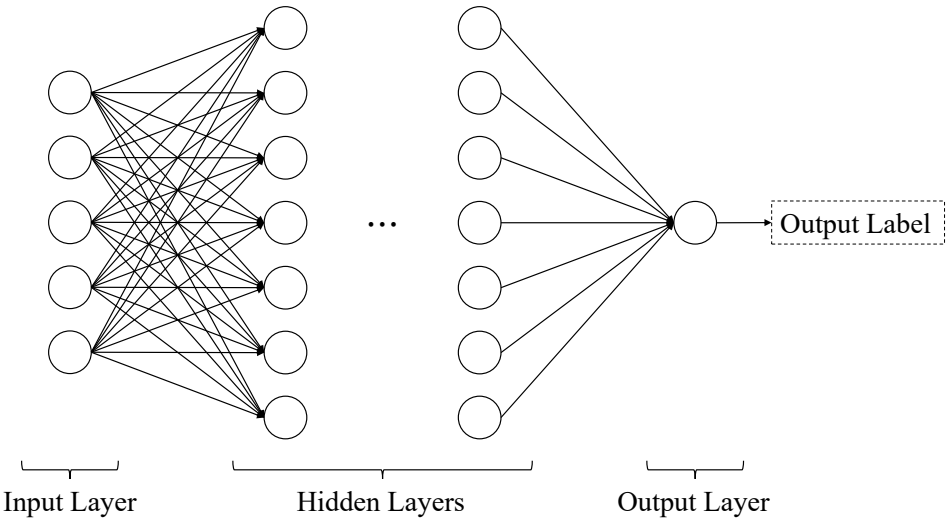


Figure 3.2: Overall structure of an artificial neural network for classification.

It consists of input, hidden, and output layers with respect to the direction in which information flows. The input layer holds all the input features based on which the desired output in the output layer is predicted. The hidden layers in between are responsible for processing data for decision making. If there are more than one hidden layer in the structure, it is called a deep learning system. Each neuron, except the ones in the input layer, takes weighted summation of all values in the previous layer, and apply an activation function on the resulting value. This activation function can be of different forms, such as threshold function, sigmoid, rectifier, and hyperbolic tangent. The purpose of applying such functions is to break the linearity of the weighted summation, allowing us to perform more complicated predictions. It is suggested in the literature [78] to use rectifier in the hidden layers. Also, since we intend to calculate probabilities at the end to assign proper labels to each data sample, we employ sigmoid activation for the output neuron, similar to Logistic Regression. In fact, by removing all hid-

den layers, the neural network acts exactly the same as Logistic Regression, indicating the flexibility of neural networks.

Unknown parameters in this structure are the weights between different layers that can be tuned by minimizing a cost function based on the predicted and actual labels. The neural network is trained using Adam optimizer [79], which is an extension to stochastic GD, through multiple back-propagation processes to update network weights accordingly.

3.1.5 Classifier Selection

We used the Random Forest algorithm [80, 81] within Scikit-Learn library [82] to train the model on the training set, and then evaluate it on the test set. As mentioned previously, Random Forest works based on developing multiple Decision Trees, giving it the following advantages over other machine learning algorithms for our purposes:

- it is commonly known to be more robust to possible outliers in the data set.
- it is less likely to over-fit the data.
- it is generally more accurate because of a process called cross-validation.
- it works well for problems with high dimensionality.
- it can be used for feature selection, which can lead to a more efficient classifier.

In comparison to neural networks, the Random Forest algorithm is computationally more efficient. Neural networks require much more data compared to the Random Forest models to yield reliable outputs. In our case, where the data set is not as enormous as usual data sets in deep learning, implementing a deep neural network is not justified. Nevertheless, the results of employing Logistic Regression and Neural Network will be presented in the results section for comparison.

Also, probability values extracted from the Random Forest classifier in our case can be considered as *how certain* the model is that the simulation

is capturing the separation bubble. This will become useful in identifying some challenging cases where the probability of being captured is estimated to be 50%.

3.2 Feature Selection

The goal of this chapter is to detect the CFD simulations where the separation bubble is poorly captured. It was demonstrated in the previous chapter that PCA is an effective method in extracting information regarding the missing separation. Therefore, it makes sense to use expansion coefficients obtained from PCA as the features of the model. Even though we found that the 16^{th} PCA mode is the possible source of the missing flow feature, we consider all the first 40 modes as inputs to the classifier. The reason is to make the classifier more flexible and let the algorithm decide which attributes matter the most for its learning purposes without being biased. The following 43-dimensional vector of features for the i^{th} solution vector is formed to be passed into the classifier:

$$\mathbf{x}^{(i)} = \Big[\text{AoA},\ \text{M},\ \text{Re},\ \alpha_{i1},\ \alpha_{i2},\ \ldots,\ \alpha_{i40}\Big] \tag{3.8}$$

Each vector of features contains the flow conditions as its first three entries, as well as the expansion coefficients for each solution vector.

3.3 Label Generation

3.3.1 Manual Labeling

To produce the labels for supervised learning, we took advantage of the availability of the high-order solution vectors. Since the 4^{th}-order simulation is assumed to be the exact solution to the Navier-Stokes equations, we labeled all the 4^{th}-order solution vectors as captured. This produced a collection of benchmark solutions for all the 525 groups of flow conditions around the airfoil. Each 2^{nd}-order solution was compared to its higher-order peer from the benchmark, and based on the quality of its separation region, was labeled ei-

ther captured or missed. It must be noted that there exist some challenging cases in the data set where it is not evident whether the low-order simulation captures the separation bubble compared to the benchmark solution. More discussion on this matter is provided in the results subsection. Statistical information about the generated labels is provided in Table 3.1.

Table 3.1: General statistical information about the data set.

total	captured	missed
1050	814	236

3.3.2 Physics-Oriented Labeling

It is worth mentioning that our first attempt to generate labels was based on a more physical study. In that process, we analyzed different characteristics of the simulation such as the error in the drag coefficient and the location of separation to decide whether a simulation captures the separation bubble properly. Although this examination provided useful initial labels for our manual labeling system, it was not proved to be a reliable label generator. In a sense, this can be a good sign as it strengthens our claim that we need to use machine learning to detect missing flow separation. Otherwise, if we could generate labels only based on simple flow physics such as the separation location, then we could simply use this label generator as our *flow feature detector*, without having to develop a machine learning model.

3.3.3 Unsupervised Labeling

The manual labeling system could become quite tedious as the number of data samples in data set increases. Another approach to produce labels for huge data sets is by using a clustering algorithm. Clustering is a type of unsupervised machine learning, where an intelligent model detects clusters (groups) of labels showing similarity, without being guided. From one perspective, we expect the computer to be able to detect patterns and clusters in the space of input features presented in Eq. (3.8) related to the missing separation. On the other hand, this approach could yield misleading labels,

as it is not guaranteed that the produced labels are linked to the missing flow feature. We do not specify our intention at any point along the clustering process, and the computer decides only based on the input space. Consequently, the generated labels may be associated with other kinds of similarities, such as the Mach number or the Reynolds number. Nonetheless, for huge data sets where the manual labeling in not practical, clustering could be used, but with great caution. We have employed a well-used clustering algorithm called k-means to generate the labels and compare those to the manually generated ones. The results are provided in Section 3.11 where we present label clusters in the 2-D plane of the most important features.

3.3.4 Imbalanced Data Sets

Having generated all the actual labels for each case, the captured labels outnumbered the missed ones by a ratio of around $7:2$ (refer to Table 3.1). This is the characteristic of an imbalanced data set, which could negatively affect the algorithm and make the learning biased towards the majority data. We handled this problem by randomly discarding some captured data and producing multiple copies of the missed data in the training set, provided as part of the Imbalanced-Learn python module [83]. Table 3.2 provides the quantity of the captured and the missed labels in each set, before and after balancing. It must be mentioned that the balancing operation is only applied to the training set for training purposes, since the test set is only meant for model validation, and it should represent the statistical characteristics of the whole data set.

Table 3.2: Statistical information about the training set and the test set, before and after applying the balancing algorithm.

	training set ($\sim 80\%$)			test set ($\sim 20\%$)		
	total	captured	missed	total	captured	missed
imbalanced	839	561	188	211	163	48
balanced	390	195	195	—	—	—

3.4 Evaluative Measures for Classification

After training the model on the training set, it should be evaluated on the test set before being considered as a sound *flow feature detector*. For binary classification, the following tools can be used for model evaluation:

- *Confusion Matrix*

  To evaluate the binary classifier, a matrix known as the confusion matrix can be formed based on the model predictions for the samples in the test set. The overall structure of this matrix is provided in Fig. 3.3. The confusion matrix provides an efficient way to explore the predictions made by the classifier. Each entry shows the number of data samples with actual and predicted labels defined by row and column tags, respectively. These are defined as:

  - *True Negative (TN)*

    TN entry accounts for the quantity of the solution vectors that actually capture the flow separation and are correctly detected as captured.

  - *True Positive (TP)*

    TP entry accounts for the quantity of the solution vectors that actually miss the flow separation and are correctly detected as missed.

  - *False Positive (FP)*

    FP entry accounts for the quantity of the solution vectors that actually capture the flow separation and are falsely detected as missed.

  - *False Negative (FN)*

    FN entry accounts for the quantity of the solution vectors that actually miss the flow separation and are falsely detected as captured.

  The same coloring convention shown in Fig. 3.3 will be used throughout this thesis to illustrate TN, TP, FP, and FN data. Based on the

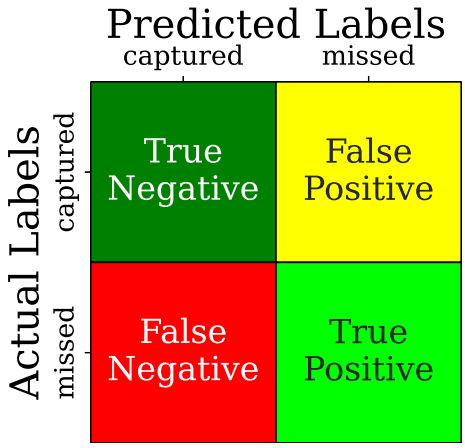


Figure 3.3: The structure of the confusion matrix for a supervised binary classifier.

confusion matrix, the following performance measures can be defined for a binary classifier.

- *Accuracy Score*

  The accuracy score measures, among all the samples, how many are correctly labeled. It can be defined as follows:

$$\text{accuracy score} = \frac{(\text{TP} + \text{TN})}{(\text{TP} + \text{TN} + \text{FP} + \text{FN})} \tag{3.9}$$

  It should be noted that this performance score is not an ideal choice for an imbalanced data set, such as the one we have. The reason is that it is fairly easy to come up with a poor classifier with a high accuracy score. For instance, a model that labels all of the samples as captured will record an accuracy of nearly 77%, since there are relatively fewer missed data than the captured ones in the test set. As alternatives, other measures can be used as mentioned in the following.

- *Precision Score*

  The precision score, also know as Positive Predictive Value, accounts for how many of the solutions predicted as missed, in fact miss the

separation bubble. It can be formulated as:

$$\text{precision score} = \frac{\text{TP}}{(\text{TP} + \text{FP})} \tag{3.10}$$

- *Recall Score*

  The recall score, also known as sensitivity or True Positive Rate, measures out of all cases where the flow separation is missed, how many samples are identified by our classifier:

$$\text{recall score} = \frac{\text{TP}}{(\text{TP} + \text{FN})} \tag{3.11}$$

- *F1 Score*

  Finally, the F1 score is simply the harmonic mean between the precision and the recall, and is defined as:

$$\text{F1 score} = \frac{\text{TP}}{\left(\text{TP} + \frac{\text{FP} + \text{FN}}{2}\right)} \tag{3.12}$$

FP samples indicate that the model suggests extra caution be taken, even though the simulation already resolves the separation region. However, FN data are more critical, as the model asserts that the simulation does not need improvement, while in reality, it misses an important flow pattern. Consequently, our focus in this thesis is more on the recall score, since the deficient simulations predicted as good simulations are more unpleasant than the good simulations detected as deficient ones. In other words, we prefer to develop a model with very few FN predictions, even though the cost of having such a model is to end up with extra FP data.

3.5 Parameter Tuning

Parameter tuning is an essential component of any machine learning workflow. As discussed, all learning algorithms define some tunable hyperparam-

eters that make models flexible to be used for different problems. For each particular case, these parameters can be tweaked to maximize prediction accuracy. Scikit-learn provides modules for this purpose, where the training set is divided into multiple bins. An iterative technique called cross-validation is implemented to evaluate different combinations of hyperparameters to find the best set of values. The steps of parameter tuning are provided here:

- We take a single set of hyperparameters to begin the iteration process.
- In each iteration, one of the bins is considered as a temporary test set and a model based on the current combination of hyperparameters is trained on the remaining samples.
- At the end of all iterations, the average performance measure is assigned to that specific combination of hyperparameters.
- We tweak the values of hyperparameters and repeat the above steps to get the new performance measure for the new set of values.
- We continue this process until we find the best combination of hyperparameters that records the highest score.

We followed the same principles to tune the parameters of our Random Forest classifier by defining 5 bins over the training set. The hyperparameters considered in our analysis were {`'n_estimators'`, `'max_features'`, `'bootstrap'`, `'max_depth'`}, that regulate tree configuration. Scikit-learn documentation [84] provides a detailed overview of these parameters. The tuned values will be presented in the results section.

3.6 Classification Steps

The skeleton of our proposed *flow feature detector* is shown in Fig. 3.4. Every step covered so far are included in the figure, from gathering data samples, to evaluating the model. The final criterion based on which one decides whether the model is ready to be used for general simulations depends on the need and complexity of specific applications. We will provide

instructions on how evaluative measures can be interpreted to assess a classifier, which can be helpful in weighing our satisfaction with regard to a particular model.

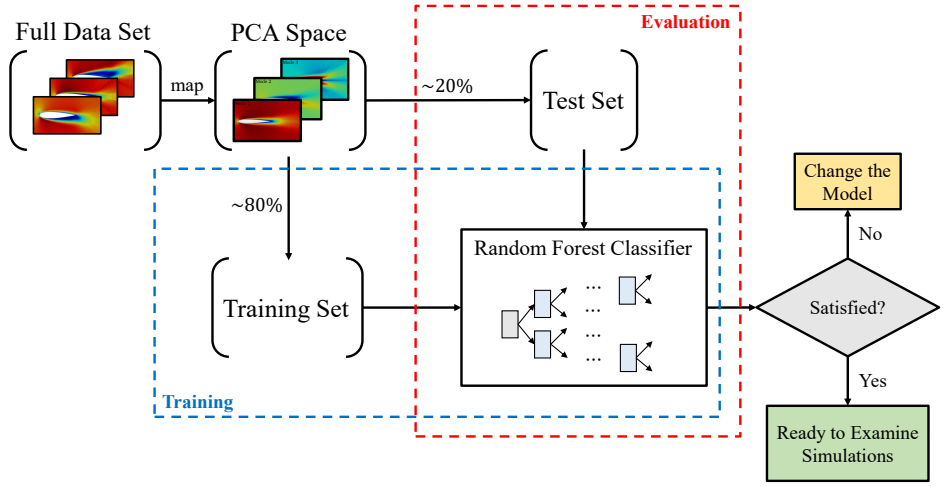


Figure 3.4: Proposed flowchart to develop a machine learning classifier to detect possible missing flow features in CFD simulations.

3.7 Results: Correlation Matrix for Separation Identification

Before training the model on the training set, we investigate how learning features are correlated with some of the important parameters of the problem. This is done by constructing a correlation matrix, as shown in Fig. 3.5, which helps us gain an insight into the possible role of each parameter in the decision making process. Each element of the correlation matrix is colored and filled with a weight that can be treated as the covariance of the parameters labeled in the corresponding rows and columns. Important points deduced from the correlation matrix are presented below:

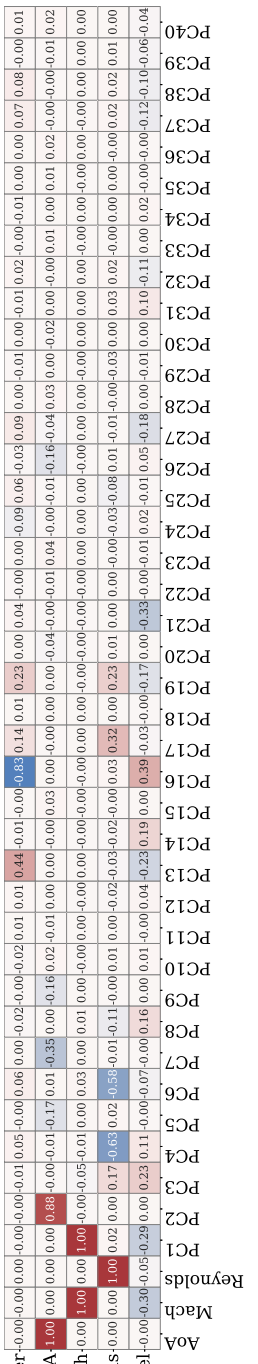


Figure 3.5: Correlation matrix for feature detection, where each entry represents the covariance of the corresponding row and column tags.

- PC_{16}, PC_{21}, the Mach number, and PC_1 contribute the most to the target label, respectively. This is consistent with our previous observation that the 16^{th} PCA mode is the possible source of the missing separation.
- PC_{13} and PC_{16} represent the effect of order of accuracy, which could be an interesting topic from the numerical analysis standpoint.
- PC_1 is directly correlated with the Mach number, meaning that the effect of the Mach number can be fully represented by the first mode.
- The angle of attack is highly correlated with PC_2.
- The effects of the Reynolds number can not be described by a single mode and is distributed mostly between PC_4 and PC_6.

We drew some of the same conclusions when inspecting the mode shapes in the previous chapter. However, for more complicated mode shapes, such as the 16^{th} one, the correlation matrix becomes the single descriptive source of information. These results indicate that using the flow conditions as features of the model could be redundant, since their influence is well defined by the PCA modes. This matter is further discussed when extracting the importance weights from the Random Forest classifier. For now, we keep the input features intact and let the machine learning model decide the importance of each feature.

3.8 Results: Classifier Predictions

3.8.1 Random Forest Classifier

A binary classifier based on the Random Forest machine learning algorithm was trained on the balanced training set and evaluated on the test set represented in Table 3.2, to detect the solution vectors lacking the separation bubble. The model was tuned using the guidelines discussed in Section 3.5, giving the optimized hyperparameters presented in Table 3.3. The performance of the default and the tuned models on the test set are shown as

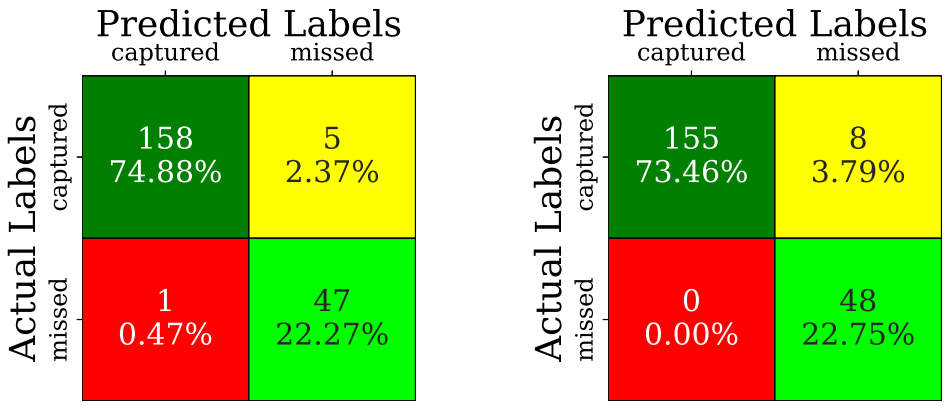
Table 3.3: Default and tuned hyperparameters for the Random Forest classifier, when optimized on the training set in Table 3.2.

	n_estimators	max_features	bootstrap	max_depth
Default	100	auto	True	None
Tuned	60	10	False	4

confusion matrices in Fig. 3.6. The confusion matrix in Fig. 3.6a demonstrates that the default model is able to label most of the solution vectors correctly before being tuned, and only a handful of cases are misidentified by the classifier. Comparing the results with the tuned version in Fig. 3.6b, we see that the latter correctly labels every simulation that misses the separation, while the model with the default parameters misses one of these deficient simulations. However, the tuned model fails to perform as well as the default model when assessing captured flow separations. Overall, tuning the model in our case does not make a significant difference in the output, as both models correctly tag simulations in most situations. Hence, we decided to pick the default model in our analysis, since it yields fewer incorrect predictions. Therefore, the results we provide for the classifier hereafter correspond to the Random Forest algorithm with the default parameters mentioned in Table 3.3. It is worth mentioning that the parameter tuning generally makes the model attain its best possible performance on both the training and the test sets when the training samples provide a good representation of the whole data set. In our case, however, we have access to a limited number of CFD simulations to train the classifier, and so optimizing the algorithm on the training set does not necessarily guarantee better performance on the test instances as well. It is expected that by adding more simulations into the data matrix, the tuned model will perform slightly better than the default one. Nevertheless, parameter tuning will not become a concern when the model shows small sensitivity to the change in parameters and performs well on both the tuned and the default models.

For a quantitative assessment of the model, we look at the performance scores provided in Table 3.4. As justified in the methodology of this section, we pay extra attention to the FN data and the recall score. The model

records an impressive recall score of 97.92%, since only one solution vector that actually misses the separation is mislabeled as captured. Moreover, the classifier shows a precision score of 90.38%, which means that having a very high recall score has not resulted in many FP predictions and the model correctly detects most of the high quality simulations. Also, it was found that the model is 97.16% accurate, which indicates that the overall performance of the classifier is very promising.



(a) Default Model **(b) Tuned Model**

Figure 3.6: Confusion matrix, resulted from testing the trained binary classifier on the test set presented in Table 3.2. The percentage values describe the quantity of each TN, TP, FN, and FP data relative to the total number of samples in the test set.

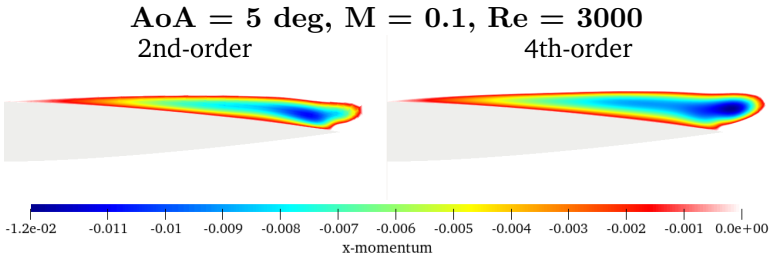
Table 3.4: Performance of the trained binary classifier on the test set, calculated based on the confusion matrix presented in Fig. 3.6 and the measures given by Eqs. (3.9-3.12).

	Accuracy	Precision	Recall	F1
Default Model	97.16%	90.38%	97.92%	94.00%
Tuned Model	96.21%	85.71%	100.00%	92.31%

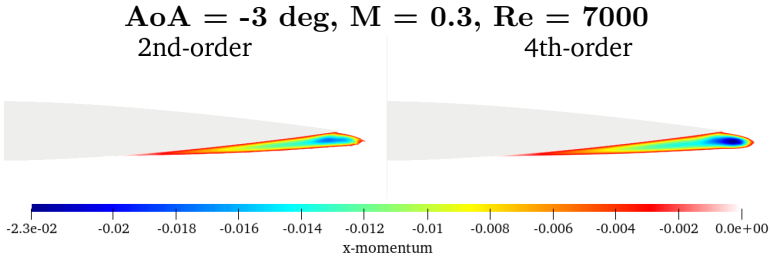
To investigate the performance of the classifier in more detail, we extracted the probabilities of its decisions for all the data samples in the test set. The way the Random Forest algorithm works is that it allocates a number to each data sample, which represents the probability of that case

capturing the separation. It also sets the threshold for decision making to probability = 50% and assigns labels accordingly. Figure 3.7 displays a side-by-side comparison of the 2^{nd}- and the 4^{th}-order separation bubble patterns for a few samples in the test set, as well as their actual label, predicted label, and the probability of those cases being captured. Again, the colormap for the streamwise momentum is manipulated so that only the negative velocity regions are illustrated. Examining the FN data in Fig. 3.7a, we observe that it is a challenging task to decide whether the 2^{nd}-order solution misses the separation region. In this case, although the 4^{th}-order solution depicts a larger separation region, it appears that the 2^{nd}-order solution correctly captures the overall pattern of the separation bubble. Nevertheless, assigning a single label to the low-order solution in this particular case seems to be a difficult task, even for the experts in the field. The model predicts that the low-order solution is solving for the separation bubble with 52% accuracy, hence it is labeled as captured. The extracted probability for this case is quite close to the boundary of decision making, meaning that the algorithm is confused and insecure to correctly assign a label to the data sample. The investigation suggests that the probability value is a logical way to describe *how certain* the model is that the separation region is captured in the simulation. The same point can be made for the solution provided in Fig. 3.7b, which is one of the FP data. Again, in this case it is not evident whether the low-order solution captures the separation. The separation bubble seems to be roughly the same size for both the 2^{nd}- and the 4^{th}-order solutions. However, the low-order separation region appears to be slightly different from a qualitative point of view. The classifier suggests that the separation bubble is 40% captured in the 2^{nd}-order simulation, which seems to be a proper estimate. To further analyze the validity of probability predictions, one example of a TN data, and two examples of TP predictions are presented in Figs. 3.7c, 3.7d, and 3.7e, respectively. It seems that the probability values are indeed good measures to be reported alongside the labels to exhibit the level of certainty of the model. Figure 3.8 shows a histogram of the probability values predicted by the model on the test set. It can be seen that the false predictions happen near the threshold of the

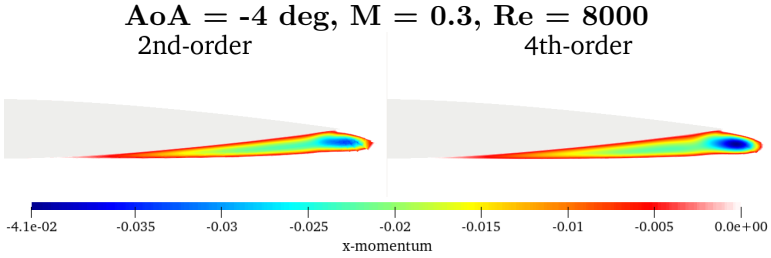
algorithm, probability = 50%, where the cases are more challenging. For the samples located farther from the threshold, in what we might call the confidence zone of the classifier, the model identifies the captured and the missed solution vectors correctly.



(a) Actual: missed, prediction: captured, probability of being captured: 52%, FN

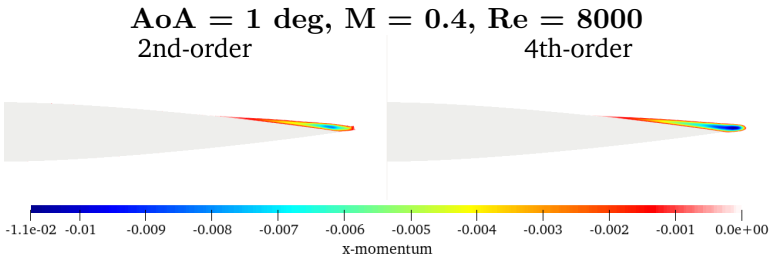


(b) Actual: captured, prediction: missed, probability of being captured: 40%, FP

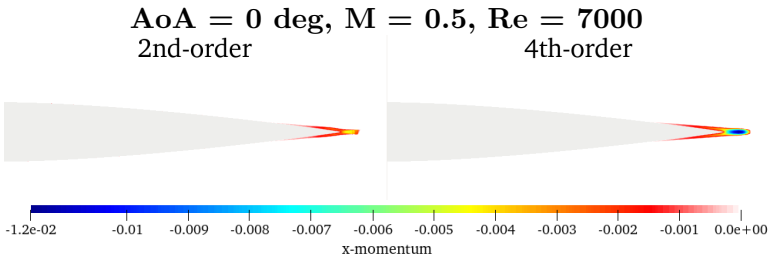


(c) Actual: captured, prediction: captured, probability of being captured: 92%, TN

Figure 3.7: Prediction information for some of the cases in the test set. Both the 2^{nd}- and the 4^{th}-order solution vectors are shown for comparison.



(d) Actual: missed, prediction: missed, probability of being captured: 18%, TP



(e) Actual: missed, prediction: missed, probability of being captured: 37%, TP

Figure 3.7: Prediction information for some of the cases in the test set. Both the 2^{nd}- and the 4^{th}-order solution vectors are shown for comparison. (cont.)

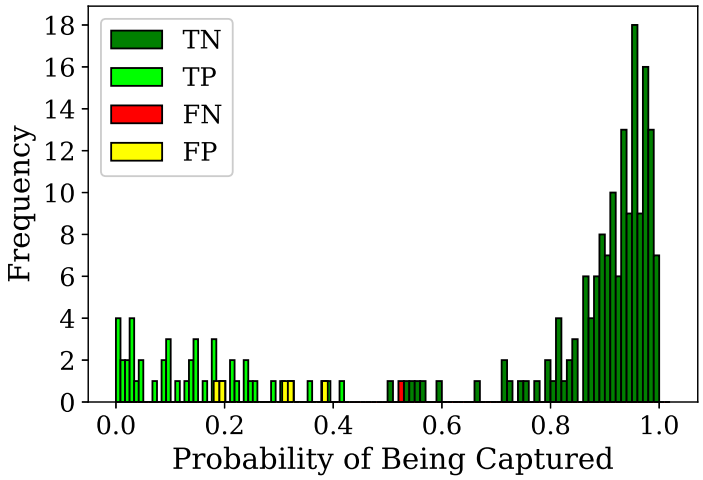


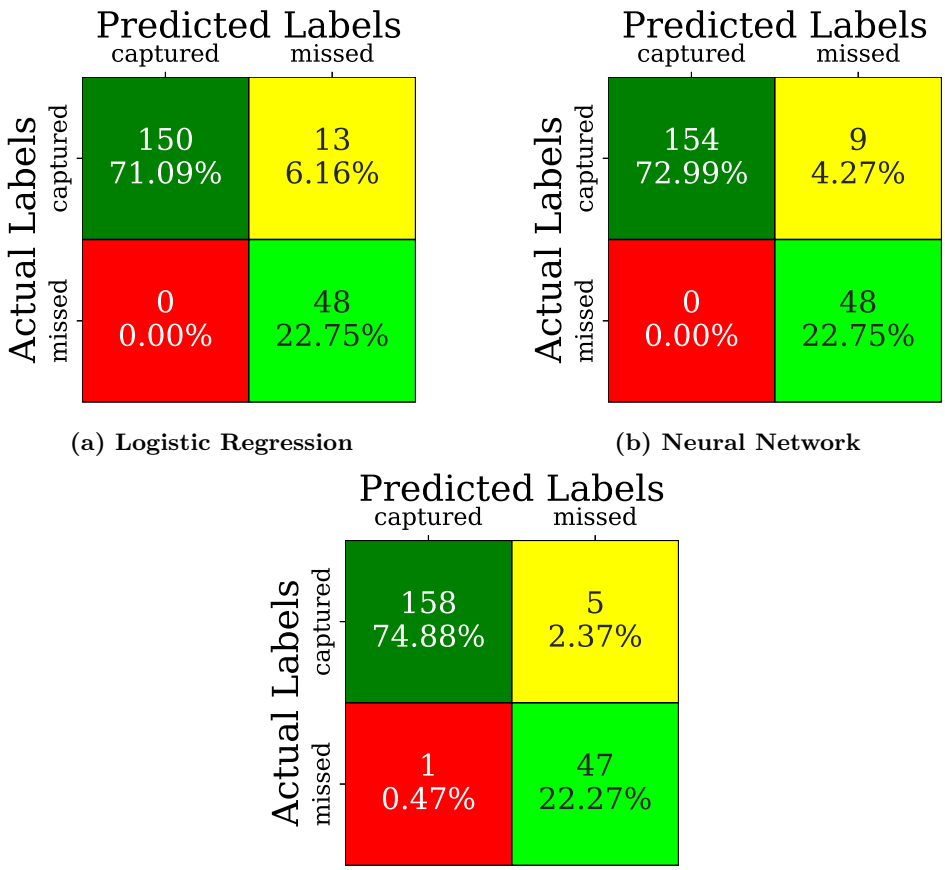
Figure 3.8: The predicted probabilities histogram.

3.8.2 Classifier Comparison

Here, we compare our developed Random Forest classifier to other common classification algorithms. In that regard, two models based on Logistic Regression and a deep neural network were trained on the same training set and evaluated on the test set. The neural network consisted of two hidden layers, each containing 10 neurons with rectifier activation functions, making it a powerful learning structure. Figure 3.9 and Table 3.5 present confusion matrices and evaluative measures for all the three models tested. We see that Random Forest performs significantly better than Logistic Regression in detecting missing separation bubble. The reason is that Logistic Regression is a very simple model that finds a linear decision boundary in the hyperspace of input features. Obviously, for a complicated decision making task where features are expected to show nonlinear relations, Logistic Regression is not a suitable choice. On the other hand, neural network performs comparable to the Random Forest model due to its deep structure. Neural networks provide an impressive flexible learning system, with many tunable parameters. Everything, from the number of hidden layers and neurons in each layer to the type of activation functions can be manipulated based on the task. However, as mentioned previously, training a deep neural network is more costly compared to a Random Forest classifier, and for cases where the number of the data samples is limited, Random Forest is generally a better choice. Nonetheless, as neural networks are becoming more popular in science and engineering, we should keep an eye on these powerful learning structures for more complex data sets involving 3-D simulations.

Table 3.5: Performance comparison of the trained binary classifiers.

	Accuracy	Precision	Recall	F1
Logistic Regression	93.84%	78.69%	100.00%	88.07%
Neural Network	95.73%	84.21%	100.00%	91.43%
Random Forest	97.16%	90.38%	97.92%	94.00%



(a) Logistic Regression

(b) Neural Network

(c) Random Forest

Figure 3.9: Confusion matrix comparison, resulted from testing three binary classifiers on the test set presented in Table 3.2.

3.9 Results: Evaluative Curves

To further explore the performance of the classifier, Learning, Receiver Operating Characteristic (ROC), and Precision-Recall curves can be used. A brief introduction for each, as well as the results of plotting these for the Random Forest classifier is presented in the following.

3.9.1 Learning Curve

The learning curve (Fig. 3.10) demonstrates the learning ability of the model over the experience it gains by gradually feeding in the training samples. The behavior of this plot is a convenient means to inspect whether the model is over-, under-, or well-fitted to the training data. The performance of the model over the training set shows perfect learning with zero false predictions throughout the learning process. This is not unusual, since the size of the balanced training set is relatively small. This makes the model able to learn all the training samples flawlessly, in a supervised learning procedure. This excellent performance of the model over the training set is not a concern as long as it executes with the expected accuracy over the test set. The accuracy measure over the test data is increased with experience, to a point of stability where it starts to show small oscillations around a fixed value. After the point of stability, the small gap between the training and the test curves is retained, demonstrating good performance on the test set. This is an indication that the model is a good fit to the data set and is sufficiently trained.

Figure 3.10: Learning curve of the developed Random Forest classifier.

3.9.2 Receiver Operating Characteristic Curve

The ROC curve (Fig. 3.11) is a tool to investigate if the binary classifier is a skilled predictor compared to a model that randomly labels the data in the test set. It is generated by changing the threshold of the Random Forest algorithm multiple times, obtaining various sets of precision and recall scores. Both the shape of the curve and the Area Under the Curve (AUC) are good measures to decide if the classifier is a skilled one. We also inspect the differences between the default Random Forest model and the tuned version, by showing both curves on the same plot. The general rule of thumb to interpret AUC is that values between 0.7 and 0.8 are considered acceptable, between 0.8 and 0.9 point to an excellent model, and above 0.9 show an outstanding classifier [85]. In our case, both the default and the tuned classifiers fall into the latter range with AUCs of 0.994 and 0.981, which outperform a random predictor with an AUC of 0.5. Comparing the results of the default and the tuned models, we notice that the former performs slightly better, justifying our decision on using that one. By the standards of AUC, the developed classifier in this work is considered a remarkable model to detect simulations with missing separations. However, for an imbalanced data set, the ROC curve can be misleading [86, 87], for the same reason that accuracy score is not a very good measure for such data sets.

3.9.3 Precision-Recall Curve

The Precision-Recall curve (Fig. 3.12) analyzes both the recall and the precision on the same plot. Again, the curves are generated for both the default and the tuned models by changing the threshold of the Random Forest algorithm and are compared to a random predictor. As shown in Fig. 3.12, both curves show good performances over the selected thresholds with AUCs of 0.97 and 9, reassuring that the models are indeed skilled predictors to detect the solution vectors with missing separation bubbles. Precision-Recall plot demonstrates that the default model shows a more natural behavior compared to the tuned one, which suggests that picking the default parameters would be a more prudent choice.

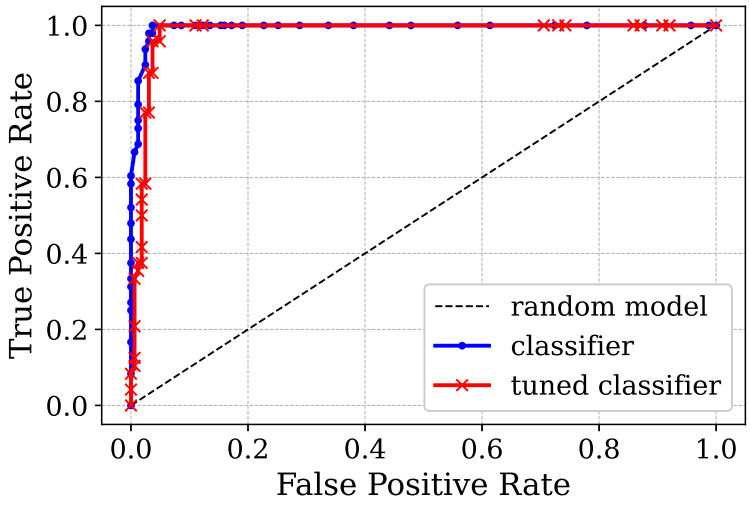


Figure 3.11: Receiver Operating Characteristic (ROC) curve of the developed Random Forest classifier; default AUC = 0.994, tuned AUC = 0.981.

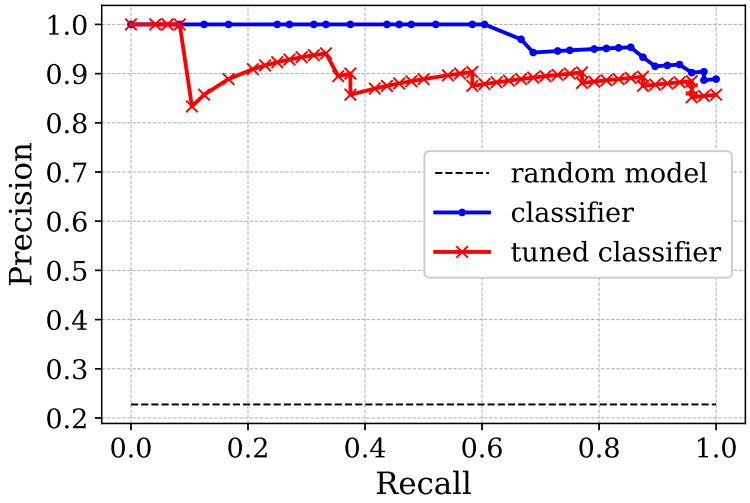


Figure 3.12: Precision-Recall curve of the developed Random Forest classifier; default AUC = 0.977, tuned AUC = 0.9.

3.10 Results: Feature Importance Analysis for Separation Identification

The Random Forest algorithm provides a set of weights assigned to the input features, to assess the importance of each parameter in the learning task. The feature importance plot is presented in Fig. 3.13, and demonstrates that:

- The classifier learns mostly from PC_{16}, as we expected from the results of PCA and the correlation matrix shown in Fig 3.5.
- After that, PC_{21}, PC_3, PC_2, PC_{13}, PC_8, PC_{14}, and PC_1 are proved to be the next major factors in detecting the missing separation, respectively.
- Unsurprisingly, flow conditions have little effect on the model, since their influence is already projected on the PCA modes.

Furthermore, it can be seen that most PCs have a very small effect on the output of the predictor, and decision making is centered around only a handful of the modes. This suggests the possibility of a more efficient classifier based only on the modes that matter. We expect that it will not affect the overall performance of the model significantly, since only the trivial parameters are discarded. This process of removing less important features is feature selection, which was introduced in the previous chapter. The results of this efficient classifier will be presented in Section 3.12.

3.11 Results: Label Clusters

Another outcome of the feature importance plot is that it helps us visualize the data set more effectively. Although PCA has significantly reduced the dimensionality of the data set, it is still not possible to picture the sample points in the 40-dimensional PCA subspace. The feature importance plot indicates that PC_{16} and PC_{21} are the most important parameters in decision making. The plot of the data points in a 2-D plane created by these two

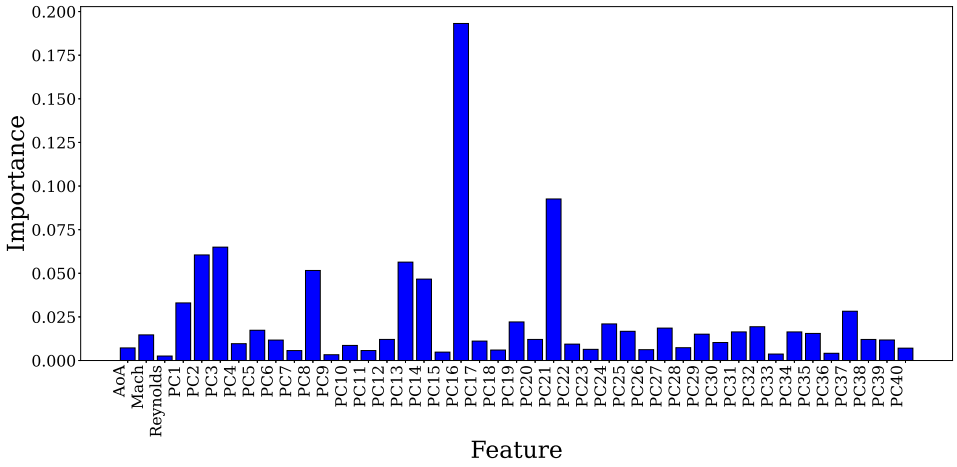
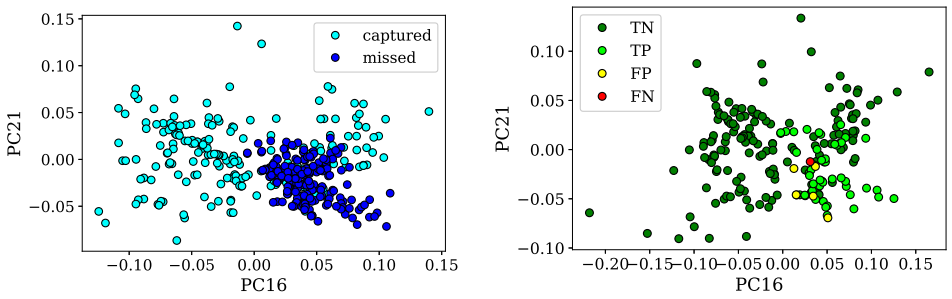


Figure 3.13: Feature importance of the classifier extracted from the Random Forest algorithm.

features is shown in Figs. 3.14a and 3.14b for the training and the test sets, respectively. It can be seen from the training set plot that the missed labels are clustered in a certain region when plotting the data in the plane of the two most important learning features. The clusters are not perfectly distinct and overlap, because other modes are also involved in decision making. It is not visually possible to plot the high-dimensional data; otherwise, we would have detected perfect clusters of the missed and the captured data in the hyper-dimensional space created by PCs. It once again demonstrates the power of PCA in extracting information about the missing separation bubble. It in fact creates a subspace from the high-dimensional data with more discrimination in different label clusters. This subspace is suitable for learning purposes where it is much easier for the classifier to detect label clusters and create a hyperplane as its decision boundary. We clearly observe that a linear decision boundary is not a good choice for our purpose, and a model capable to detect nonlinearities in the input data should be employed; this is another reason that Logistic Regression is not a proper option for this case.

Looking at the test set plot (Fig. 3.14b), it can be observed that the model is capable of detecting the regions assigned to each label. The perfor-

mance scores of the model on the test set have been mentioned previously in Table 3.4. Here, we only point out the placement of the falsely predicted samples in the 2-D plane of PC_{16} and PC_{21}. It is clear that the single FN data and most of the FP predictions are positioned near the boundary of the missed labels cluster. This explains the confusion of the classifier when facing such challenging data points. In such cases, the probability value extracted from the model is often a good signal to the uncertainty of the predictions.



(a) Training set samples

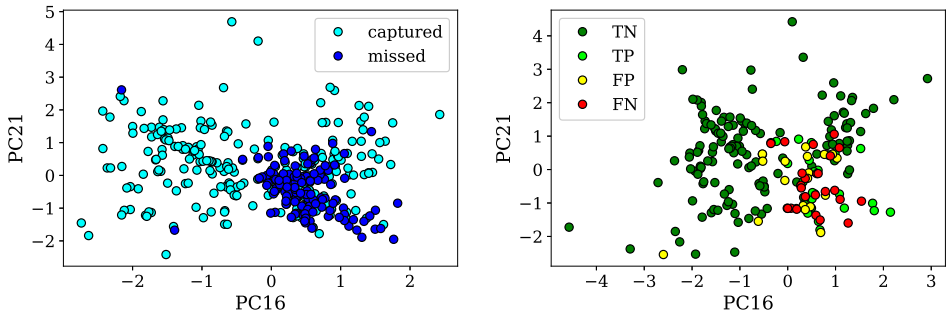
(b) Test set samples colored by model predictions

Figure 3.14: Label positions on the 2-D plane created by PC_{16} and PC_{21}.

Testing Unsupervised Labeling

Now that we have found the two most important features and are able to visually scan the label clusters in the data set, we can go back and assess the unsupervised clustering method for generating the true labels, introduced in Section 3.3. We applied a k-means clustering method to generate binary labels for the full data set in Fig. 1.11. Figure 3.15 depicts the resulting labels in the 2-D plane of PC_{16} and PC_{21}. We see that the clusters captured by the k-means algorithm matches our manual labels for the most part. However, compared to the manual labeling, k-means mislabels $\sim$ 30% of the data, which is concerning. Even so, clustering could become handy when the manual labeling becomes tedious as the number of data samples increases. Also, it can be treated as both the label generator and the *flow*

feature detector (results of which is presented in Fig. 3.15). However, the combination of the manual labeling and Random Forest still gives much better predictions.



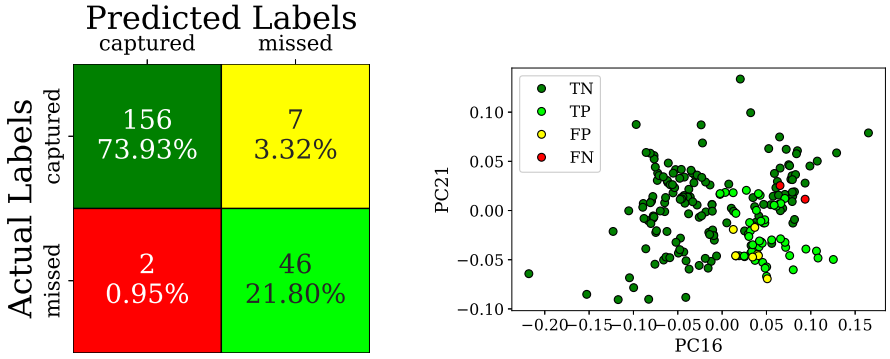
(a) Training set samples (detected by the algorithm itself) **(b) Test set samples colored by model predictions**

Figure 3.15: Label positions on the 2-D plane created by PC_{16} and PC_{21}, for an unsupervised classifier.

3.12 Results: Efficient Classifier

The results of the feature importance analysis, presented in Fig. 3.13, suggest that a more efficient classifier could be developed based on the most influential parameters in the vector of features. Here, we present the result of a binary classifier that was trained using PC_{16}, PC_{21}, PC_3, PC_2, PC_{13}, PC_8, PC_{14}, and PC_1, which were proved to be the major factors regarding the missed separation, respectively. Thus, instead of operating in a 43-dimensional space of information, the learning algorithm tries to find patterns and clusters in an 8-dimensional subspace, which makes the learning task much easier, and also faster. The confusion matrix and the performance scores are presented in Fig. 3.16a and Table 3.6. In addition, the label predictions for the test samples on the plane of the most important features, namely PC_{16} and PC_{21}, are plotted in Fig. 3.16b. The plot of the labels in the training set remains the same as Fig. 3.14a, since the data set is not changed. From the results, we conclude that the model is

still able to correctly label most of the solution vectors. Compared to the previous model, the efficient classifier produces a few more false predictions, but not enough to affect the overall performance of the model significantly. Again, the false predicted labels are positioned near the boundary of decision making, meaning that these cases are the challenging ones.



(a) Confusion matrix

(b) Test set samples colored by the efficient model predictions, shown on the 2-D plane of PC_{16} and PC_{21}

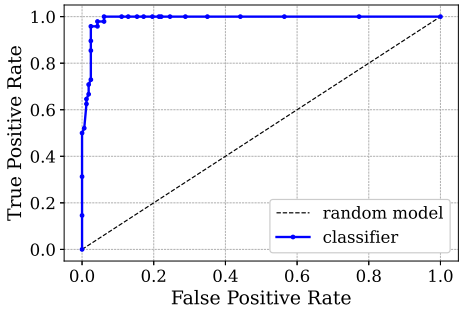
Figure 3.16: Results of testing the efficient classifier on the test set.

Table 3.6: Performance comparison of the main and the efficient Random Forest classifiers.

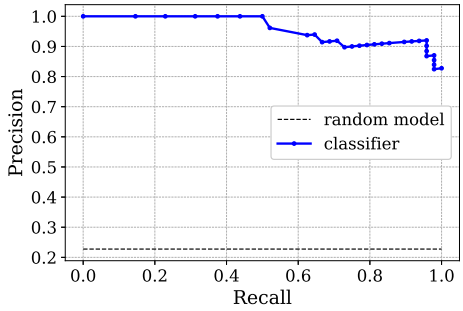
	Accuracy	Precision	Recall	F1
Main Model	97.16%	90.38%	97.92%	94.00%
Efficient Model	95.73%	86.79%	95.83%	91.09%

To ensure that this efficient classifier is a skilled and reliable model in detecting the simulations with missing separation, evaluative curves are plotted in Fig. 3.17. Again, the behavior of the test curve suggests that the efficient model is a good fit to the training samples, and performs well on the test set. Even though AUCs calculated for the ROC and the Precision-Recall curves are slightly smaller than the ones for the previous model, they still indicate that the efficient model is an outstanding predictor for our goals. Hence, it is proved that it is possible to detect the poorly simulated separation regions using the proposed efficient model with acceptable accuracy.

(a) Learning curve



(b) ROC curve, AUC = 0.989



(c) Precision-Recall curve, AUC = 0.959

Figure 3.17: Evaluative curves to study the performance of the efficient classifier.

3.13 Results: Classifier Generalization

One of the most important characteristics of a machine learning model to be considered suitable is its ability to be generalized to a wide range of data samples. The model developed in the previous sections is only applicable for simulations performed on the mesh shown in Fig. 1.10. This is because of the solution projection step where we take the inner product of a solution vector and the PCA modes to calculate expansion coefficients as the input information to the model. This operation requires the solution vector to be of the same dimension as the PCA modes. This imposes the following restrictions if we intend to examine a new 2-D simulation around an airfoil:

- The airfoil is limited to be NACA 0012, and the model is not applicable to other geometries.
- The computational domain should be exactly identical to the mesh pictured in Fig. 1.10, which we refer to as the base mesh henceforth.

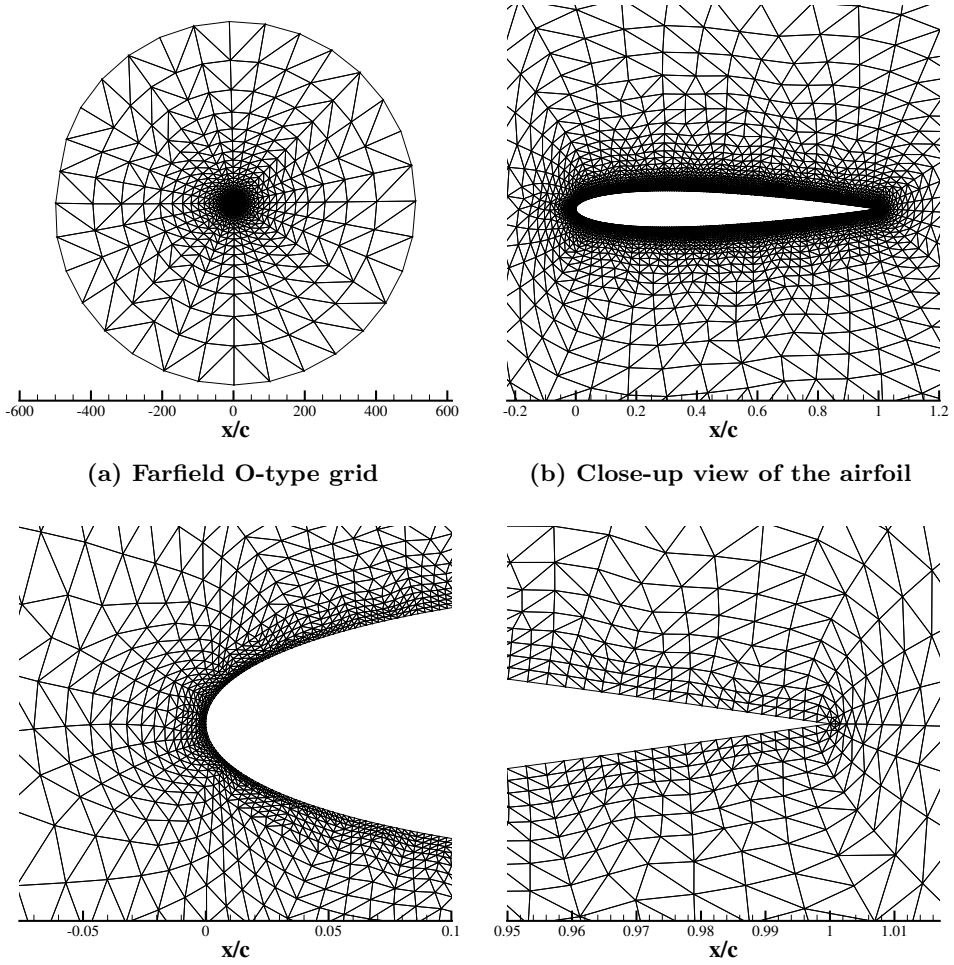
In this section, we address the second limitation and the first one is left for future studies.

Let us consider a flow simulation around NACA 0012 airfoil on the mesh shown in Fig. 3.18. We call this new unstructured mesh, containing 6176 cells, the arbitrary mesh, as opposed to the base mesh. The aforementioned limitations suggest that to analyze this new simulation, our already-trained model is not suitable as we cannot map the new 24704-dimensional solution vector on the available 16568-dimensional PCA modes. Hence, we need to repeat the whole PCA decomposition by generating a new data set of flow simulations on the arbitrary mesh to create an acceptable PCA subspace. This operation is computationally exorbitant and we are better off performing the current accuracy improvement methods mentioned in Chapter 1.

One approach to generalize our previously-trained model to be functional on new simulations on any arbitrary mesh is to map the new solution vector to our base mesh. This is done by utilizing a built-in function in ANSLib, which takes a solution on a mesh and transforms it to another. By doing that, the new mapped solution vector can be projected to the previously-created PCA subspace to get the input features for the classifier. Therefore, our suggested generalization method is as simple as concatenating the aforementioned mapping functionality at the very beginning of the classification pipeline shown in Fig. 3.4.

To test the generalization, we simulated the flow field on the arbitrary mesh (Fig. 3.18) using the following group of flow conditions:

- the angles of attack (AoA): {-4, -1, 0, 2, 3},
- the Mach numbers (M): {0.2, 0.5},
- and the Reynolds numbers (Re): {4000, 8000}



(a) Farfield O-type grid

(b) Close-up view of the airfoil

(c) Leading-edge (10% of the chord)

(d) Trailing-edge (5% of the chord)

Figure 3.18: Unstructured mesh around NACA 0012, containing a total of 6176 triangular cells.

to cover various flow configurations, and manually labeled each solution vector using the guidelines provided in Section 3.3. Statistical information about this new set of flow simulations is provided in Table 3.7.

Using the efficient classification model on these new simulations, which have gone through the mapping operation described above, results into the confusion matrix and evaluative measures presented in Fig. 3.19 and Table

Table 3.7: General statistical information about the data set generated to generalize the classifier.

total	captured	missed
40	23	17

3.8, respectively. The results indicate that although the performance of the model is not as good as before, generalization can be still considered successful. When we map a solution vector from a mesh to another, there will be an inevitable information loss along the way. This introduces additional inaccuracy that can alter expansion coefficients, and therefore reduce the performance. However, the model still remarkably marks most of the deficient CFD simulations. Note that the generalization set presented in Table 3.7 is a very small group of simulations and is only generated to test our proposed method to assess solution vectors obtained on arbitrary meshes. To perform an accurate comparison between the performance of the model when applied on different mesh configurations, a more comprehensive data set would be required. This comes with the challenge of manually labeling a large number of simulations, which could become burdensome. At this point, we believe that the small data set considered here is sufficient for the purpose of this section.

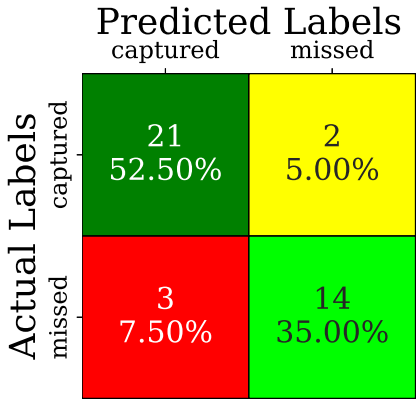


Figure 3.19: Confusion matrix, resulted from testing the efficient binary classifier on the generalization set presented in Table 3.7.

Table 3.8: Performance of the trained binary classifier on the generalization set, calculated based on the confusion matrix presented in Fig. 3.19.

Accuracy	Precision	Recall	F1
87.50%	87.50%	82.35%	84.85%

3.14 Summary

This chapter presented the development of a machine learning classifier to detect CFD simulations where flow separation is missing, or poorly-resolved. It was demonstrated that PCA subspace is a suitable information space to build a classifier to evaluate CFD solution vectors. The results of the developed classifier is a single label, either missed or captured, accompanied by a probability value signifying its certainty. Either scenario can be interpreted as follows:

- A captured label specifies that the separation region in the simulation is well-resolved and no further accuracy improvement is needed. In these cases, the probability of being captured should be checked as an indicator to the level of certainty. If this value is close to 50%, it is a good idea to act cautiously in the next steps of the design.
- On the other hand, a missed label alerts us that the separation bubble is either missed or poorly-resolved. Therefore, accuracy improvement methods should be utilized.

In the second scenario, we have not achieved a major improvement in our analysis and we still need to implement costly methods such as grid refinement or error estimators. The classifier only tells us that improving the simulation is necessary in these cases. An alternative error estimator can be developed based on the similar machine learning approach we took in this chapter. This new alternative accuracy improvement model will be introduced in the next chapter.

Chapter 4

Error Prediction

In the previous chapter, we demonstrated that the expansion coefficients obtained from PCA can play an important role in detecting missing flow separation in CFD simulations. A separation bubble has a direct impact on the drag coefficient of the airfoil. This is because flow separation changes the pressure distribution around the airfoil, which directly changes the pressure drag. Hence, we expect the same inputs that signaled us to the poorly-resolved separation bubble to be usable for estimation of the error in the computed drag coefficient. Thus, in this chapter, we explore the implementation of a machine learning regressor to accurately quantify uncertainties in CFD, which is the error in drag in our case.

There are many theoretical and practical overlaps between regression and classification. Therefore, in this chapter we only provide necessary information about the main differences between these two types of machine learning approaches. New algorithms and evaluative measures specific to regression are presented and discussed in the first four sections, and the subsequent sections cover the ability of our intelligent regressor in estimating the error in the drag coefficient.

4.1 Machine Learning Regression

In this chapter, we turn our attention to a different type of machine learning called regression. Unlike classification, where the goal is to predict a categorical label, regression aims at estimating a continuous numerical value. There are numerous possibilities for the value of the error in a simulation, and therefore a machine learning regression must be employed for estimating this numerical parameter. Regression and classification are based on the

same principles described in the previous chapter and they only differ in the algorithms used to produce the output from the input features. While classifiers generate a label as the output, regressors give a number, known as the target value. The following presents a brief overview of some of the algorithms specific to machine learning regression.

4.1.1 Linear Regression

Linear regression is the simplest form of curve fitting, where the relation between the input features and the target value is represented by a flat hyperplane (or a straight line when there is only one input feature). To perform linear regression, let $\mathbf{x}^{(i)}$ be the vector of input features and $y^{(i)}$ be the actual target value for the i^{th} data sample. We can predict the target value as a weighted summation over the input values:

$$\hat{y}^{(i)} = \theta_0 + \sum_{j=1} \theta_j x_j^{(i)} = \boldsymbol{\theta} \cdot \mathbf{x}^{(i)} \tag{4.1}$$

where $\boldsymbol{\theta}$ is the vector of unknown weights, and a constant value equal to 1 is added as the first entity to $\mathbf{x}^{(i)}$ to yield a clean vectorized form. Gathering all the manipulated features (containing 1 as the first entity) of the training samples as columns of a data matrix $\mathbf{X}$, and the target values as a column vector $\mathbf{y}$, we can construct the following least-square system of equations:

$$\mathbf{X}^T \boldsymbol{\theta} = \mathbf{y} \tag{4.2}$$

To find the best set of unknown weights, a cost function should be defined and minimized similar to the classification process. For linear regression, the mean squared error is used:

$$\text{MSE}\left(\mathbf{X}, \boldsymbol{\theta}\right) = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2 \tag{4.3}$$

where m is the number of the training samples, and $\hat{y}^{(i)}$ is substituted from Eq. (4.1). There are two general approaches to minimize this cost function:

- *The Normqal Equation*
  For linear regression represented in Eq. (4.2), there exists a closed-form solution called the normal equation written as follows:

$$\boldsymbol{\theta} = \left(\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{X}\mathbf{y} \tag{4.4}$$

  which minimizes the cost function in Eq. (4.3). This equation directly yields the best possible values for the unknown weights. A somewhat different representation of this equation is:

$$\boldsymbol{\theta} = \mathbf{X}^{\dagger}\mathbf{y} \tag{4.5}$$

  where $\mathbf{X}^{\dagger}$ is the pseudoinverse of $\mathbf{X}^T$, also known as the Moore-Penrose inverse, computed using singular value decomposition. The normal equation results in relatively expensive operations, and generally is not used for real-world applications.

- *Optimization*
  Optimization gives a more efficient approach, where the cost function is minimized through an iterative process. A well-known optimization algorithm is the Gradient Descent (GD) and its variants such as the Batch and the Stochastic GD. This approach is much more efficient compared to the normal equation when dealing with high-dimensional data sets, and is used for minimization purposes throughout this work.

Comparing linear regression (Eq. (4.1)) with Logistic Regression (Eq. (3.3)), we notice that the only difference is the presence of a sigmoid function in the latter. The reason is that for classification, we need to map the result of the linear summation to a range between 0 and 1 to assign labels accordingly, whereas in regression problems, the target value is estimated directly using the weighted summation in Eq. (4.1).

4.1.2 Polynomial Regression

For more complex data sets that cannot be well represented linearly, a polynomial regression can be used to take nonlinear relations into account. This algorithm is a derivative of linear regression rather than a stand-alone regression technique. Let us set the polynomial degree as z, and add all nonlinear combinations of the n-dimensional input features up to the z^{th}-degree to the input vector:

$$\mathbf{x}_{\text{new}} = \left\{ \forall x_j \in \mathbf{x} \text{ and } k_j \in \mathbb{N} : \prod_{j=1}^{n} x_j^{k_j};\ \text{where: } \sum_{j=1}^{n} k_j \leq z \right\} \tag{4.6}$$

Using this new set of the input features, a linear regression model can be developed using the same guidelines provided in the previous subsection.

One major drawback of employing polynomial regression is that it substantially increases the dimensionality of the input vector by adding nonlinear terms. It negates the efforts of PCA in creating an efficient space to perform machine learning. To be more specific, if we take the same 43-dimensional input space presented in Eq. (3.8), we will get:

- a 990-dimensional input vector, if we use a 2^{nd}-degree polynomial,
- a 15180-dimensional input vector, if we use a 3^{rd}-degree polynomial,
- and a 178365-dimensional input vector, if we use a 4^{th}-degree polynomial.

It is clearly shown that polynomial regression completely wastes the achievements of PCA in dimensionality reduction. Furthermore, polynomial regression is prone to over-fitting the training data, as it introduces many degrees of freedom into the model. It all makes this regression algorithm an unfitting tool in this project.

4.1.3 Random Forest Regression

Decision Tree, and its advanced version Random Forest, can be used for both classification and regression tasks. The structure of the regressor is

exactly identical to the Random Forest classifier pictured in Fig. 3.1, where the learning system is composed of multiple conditioning nodes. The only difference is that in the Random Forest regression, instead of assigning a label to each child node, the arithmetic average of actual target values of data samples entering the node is considered as their estimated target values.

To create the overall structure of the Decision Tree, the Random Forest classifier branches off from the parent node to yield minimum Gini impurity for each of its child nodes (refer to Eq. (3.7)). However, the Random Forest regressor does the same based on minimization of the mean squared error, introduced in Eq. (4.3), in each of its child branches. The cost function for such a system is provided as:

$$J\left(x, t_x\right) = \frac{n_{\text{left}}}{n_{\text{total}}}\text{MSE}_{\text{left}} + \frac{n_{\text{right}}}{n_{\text{total}}}\text{MSE}_{\text{right}} \tag{4.7}$$

$$\text{where } \begin{cases} \text{MSE}_{\text{left/right}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{left/right}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{left/right}} = \frac{1}{n_{\text{left/right}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

In this equation, $\hat{y}_{\text{left/right}}$ represents the estimated target value in each conditioning node, and $y^{(i)}$ is the actual target value of each sample entering the node. The Random Forest regression is considered an advanced model that is often used in cases where linear and polynomial regressors are incapable of predicting accurate target values.

4.1.4 Neural Networks and Deep Learning for Regression

Neural networks are another learning algorithm that are applicable for both classification and regression tasks. The structure of a neural network regressor is the same as the one depicted in Fig. 3.2, consisting of an input layer, one or more hidden layers, and an output layer. The output layer contains only one neuron that gives the estimated target value. Similar to the classification case, we apply rectifier activation functions to the neurons in hidden layers. However, unlike neural network classifier where a sigmoid function was applied to the output neuron to yield probability, a neural network regressor does not need such an activation function. Linear regression

can be thought as a neural network without any hidden layers in between its input and output layers.

The common cost function used in the neural network regression is the mean squared error defined in Eq. (4.3). A variant of GD optimization techniques called the Adam optimizer [79] is employed similar to neural network classification to minimize this cost function.

4.1.5 Regressor Selection

So far in this chapter, we mentioned that there are lots of similarities between classification and regression models. In particular, Random Forest and neural network are capable to tackle both learning tasks as their structure allows them. We also introduced two models specific to regression problems, namely linear and polynomial regressors that are suitable for relatively simple applications. We argued in the previous chapter that our CFD data set is inherently complex, having nonlinear correlations between the features; it automatically makes the use of simple models obsolete for such complicated learning tasks. Also, we discussed in detail why polynomial regression is a wasteful model that tends to over-fit the data.

To stay consistent with the classifier we developed in the previous chapter, which was proved to be a successful one, we trained the regressor using the Random Forest algorithm. We can take the same arguments made in Section 3.1 as the logic behind this choice. The results of employing other methods are also presented and compared to ultimately decide which model performs best when trying to estimate the error in the drag coefficient.

4.2 Features and Target Values

Missing flow separation directly affects the drag coefficient of the airfoil. Hence, the same input features used to detect separation are expected to be proper choices to estimate the error in the drag coefficient. Consequently, the same vector of input features shown in Eq. (3.8) is passed to the regressor for the purpose of this chapter. We will analyze the importance of each of

these entities when training the model to see if we can further reduce the dimensionality of the input space.

The output values, however, are no longer the binary labels (captured or missed) defined for the classifier. Instead, we compute the drag coefficient for each of the 2^{nd}- and the 4^{th}-order simulations in the same flow condition, and take the relative error in the value computed for the lower-order case as its target value. This relative error is calculated as:

$$C_{d_{error}} = \frac{|C_{d_{Low\text{-}Order}} - C_{d_{High\text{-}Order}}|}{C_{d_{High\text{-}Order}}} \tag{4.8}$$

Note that the drag coefficients correspond to the total drag, taking both the pressure and the viscous drags into account. Although it is expected that the missing flow separation affects the pressure drag the most, we take the error in the total drag as the target value since it is the quantity we often appraise for performance evaluation. For the higher-order simulation, target value is set to zero, since it is assumed to be the exact solution. Unlike classification where label generation was a challenging step, the guidelines for setting target values are straightforward for the regression task. Generating these values for our data set, relative errors for 2^{nd}-order solutions ranged from 0.0156 to 0.1052 (or 1.56% to 10.52% when presented in percentage). The goal is to estimate these errors based on the input features defined in Eq. (3.8) using machine learning regression. The estimated relative error in total drag can then be used as a correcting term for the simulation output.

4.3 Evaluative Measures for Regression

Since classification and regression predict different data types, evaluative measures employed for these differ as well. For regression tasks, we deal with continuous numerical values instead of labels and metrics capable of handling these types of data must be used. There are generally two evaluative measures for regression, as explained in the following:

- *Root Mean Squared Error (RMSE)*
  RMSE, which directly corresponds to the l_2 norm[3] denoted by $\|.\|$, is the typical performance measure for regression problems. It is obtained by taking the second root of the mean squared error defined in Eq. (4.3):

$$\mathrm{RMSE}\,(\mathbf{X}) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2} \tag{4.9}$$

  where $\mathbf{X}$ is the data set containing all the input features, m is the total number of samples, and y and $\hat{y}$ represent the true and the predicted target values, respectively. RMSE is more intuitive than the mean squared error, as it has the same unit as the target values.

- *Coefficient of Determination*
  This measure is commonly denoted by R^2, and indicates how well the model performs compared to an estimator that always takes the mean of the target values as its prediction. Let $\mathbf{y}$ be the vector of true target values and $\hat{\mathbf{y}}$ be the vector of predictions, each containing n entities. R^2 score for this system is defined as:

$$R^2 = 1 - \frac{SS_{\text{residual}}}{SS_{\text{total}}} \tag{4.10}$$

$$\text{where } \begin{cases} SS_{\text{residual}} = \sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 \\ SS_{\text{total}} = \sum_{i=1}^{n}(y^{(i)} - \bar{y})^2 \end{cases}$$

  In the above equations, $\bar{y}$ represents the arithmetic mean of the true target values. R^2 score shows how much of the variation in the target value can be represented by the input features. A perfect model that predicts all the true target values without any error yields $R^2 = 1$, while a naive estimator that sets all predictions as the mean of the target values results in $R^2 = 0$. Therefore, we intend to develop a model with an R^2 score closer to 1. This score is usually between 0 and 1, but negative values could be obtained for the models that

[3] In general, the l_k norm of vector $\mathbf{v}$ is written as: $\|\mathbf{v}\|_k = \left(|v_0|^k + |v_1|^k + \cdots + |v_n|^k\right)^{\frac{1}{k}}$

exhibit extremely bad behavior.

4.4 Regression Steps

Similar to the classification process depicted in Fig. 3.4, Fig. 4.1 presents our attempt to develop an intelligent model to predict the error in an output parameter of a CFD simulation. We notice that the suggested pipelines for both the feature detector and the error estimator are exactly identical, indicating the correlation between these two tasks. The only difference between the two is the algorithm used to generate the output.

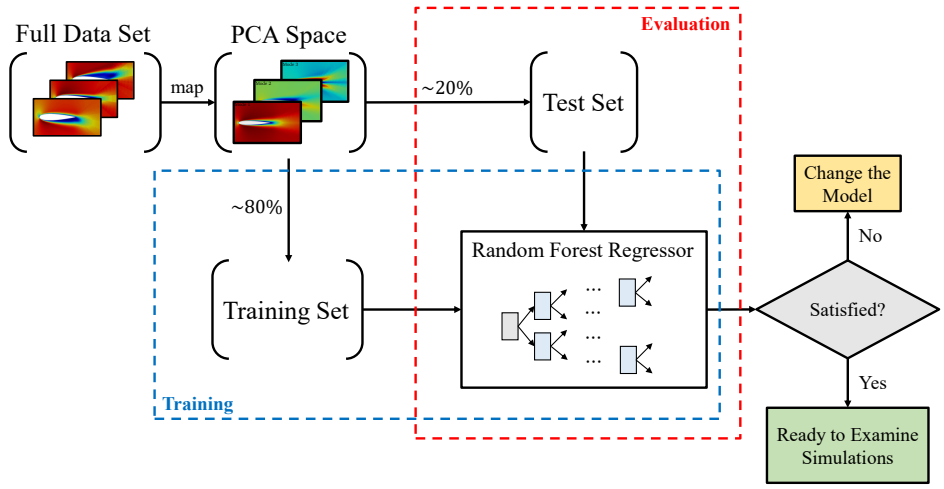


Figure 4.1: Proposed flowchart to develop a machine learning regressor to predict the error in CFD output parameters.

4.5 Results: Correlation Matrix for Error Estimation

Before passing the input features to the model to observe how it performs, we inspect the correlations between the input data and some of the important parameters of the simulation. This gives a general insight into the importance of each feature in estimating the error in the drag coefficient, and

provides an additional means to examine the behavior of the model. Correlation information is provided in Fig. 4.2. The structure of this matrix was previously described in Section 3.7.

Since we use the same data set as we considered for classification, general correlation data remains the same. For instance, the same conclusions we drew for the role of each input feature in capturing the variance in the general flow parameters and the order of accuracy apply here as well. In this chapter, our attention is on the capability of input variables in predicting the error in the drag coefficient. Looking at the corresponding row in the correlation matrix, we observe that PC_{16}, PC_{13}, and PC_1 represent most of the target value. We omitted the effects of the Mach number, as it is directly correlated with PC_1. Inspecting the correlation matrix for the missing separation in Fig. 3.5, we see the aforementioned features have also considerable contributions to the missing flow separation. This reinforces our earlier argument that the flow separation and the drag coefficient are directly related, as their variance is well captured by the same input features. It also increases our hope in developing the regressor based on the defined input features, since the classifier exhibited outstanding performance in the previous chapter. It is worth mentioning that not all important features in predicting the missing separation bubble are necessary for error estimation. This is because the error in the computed drag is only one of the side effects of the missing flow separation. Other outcomes of the missing pattern are captured by other features that appear to be less important for error prediction, such as PC_{21}. We will compare the results of the feature importance analysis to the arguments we made in this section for further assessment.

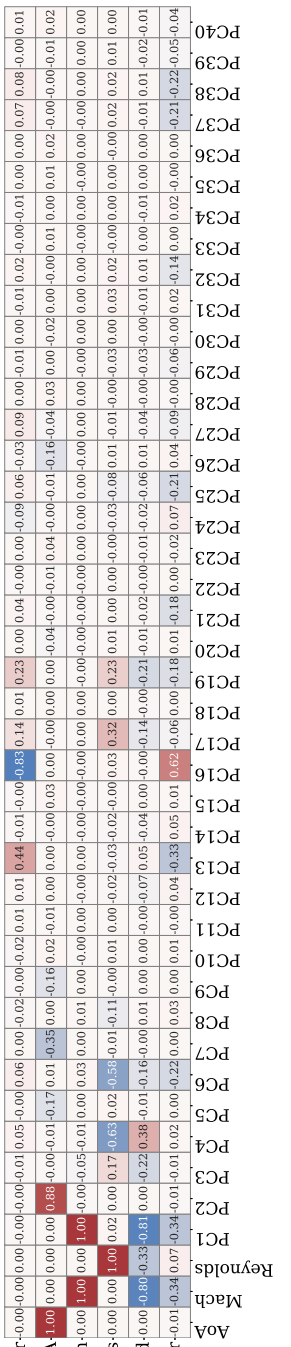


Figure 4.2: Correlation matrix for the error estimation, where each entry represents the covariance of the corresponding row and column tags.

4.6 Results: Regressor Estimations

4.6.1 Random Forest Regressor

A machine learning regressor was developed to take the input information in Eq. (3.8) and estimate the relative error in the total drag coefficient. It was trained and evaluated on the training and the test data sets shown in Table 3.2, before balancing. The regressor was tuned following the steps provided in Section 3.5, which results in the values presented in Table 4.1. Using

Table 4.1: Default and tuned hyperparameters for the Random Forest regressor, when optimized on the imbalanced training set in Table 3.2.

	n_estimators	max_features	bootstrap	max_depth
Default	100	auto	True	None
Tuned	320	20	True	None

evaluative measures in Eqs. (4.9) and (4.10), we get the results provided in Table 4.2. Unlike classification, where the tuned model failed to make predictions better, here the tuned regressor is slightly improved. Nevertheless, both the default and the tuned models exhibit comparable results, indicating that the regressor is not highly sensitive to the hyperparameters. The results presented hereafter correspond to the tuned Random Forest regressor. We observe that the model performs very well on the training set, as we expected. It shows that the model does not under-fit the data, and is able to detect patterns and correlations in the input space for its predictions. For ultimate assessment, we must inspect the performance of the regressor when applied to the test samples. The results demonstrate that the regressor remarkably estimates the error in the drag coefficient for the previously unseen simulations. It suggests that the model is not over-fitting the training set, and can be considered a well-fit intelligent model for uncertainty quantification. More specifically, the RMSE we get from the test data indicate that the model predicts the target values with an average error of ± 0.0024. Note that the true target values range from 1.56% to 10.52%, when written in percentage form, and estimating these values with only ± 0.24 uncertainty is quite an achievement. Also, this is an outstanding performance consider-

ing the fact that the model predicts the error with such accuracy without introducing additional computational burdens imposed by the current error estimation techniques.

Table 4.2: Evaluative measures of the Random Forest regressor when applied to the imbalanced data set presented in Table 3.2.

	Default Model		Tuned Model	
	training	test	training	test
RMSE	0.0007796	0.002627	0.0007424	0.002446
R^2	0.9989	0.9872	0.9990	0.9889

RMSE provides significant information about the output of the regressor and how well the model is fitted to the data. To inspect the relation between the input and the output space of the regressor, we must analyze the coefficient of determination denoted by R^2 in Table 4.2. As we argued before, values closer to 1 are favorable and signal that the input features capture most of the variance observed in the target values. Referring to the measures, our model shows a remarkable R^2 score of 0.9889 when applied on the test set, which illustrates perfect correspondence between the input and output spaces.

The combination of RMSE and R^2 scores demonstrate that our intelligent regressor based on the Random Forest algorithm is a potent tool for predicting the error in the drag coefficient.

4.6.2 Regressor Comparison

We performed the same regression using two other machine learning algorithms, namely linear and deep neural network regression. As discussed before, polynomial regression is not a wise choice in this case as it greatly increases the dimensionality of the input space of features. The results are mentioned here to examine whether Random Forest is indeed the most suitable choice for the purpose of this chapter. To better compare the performance of different algorithms in predicting the error, evaluative measures are all put together and shown in Table 4.3. We only focus on the performance of the models when applied to the test instances, since these samples

mimic the real-world simulations that the models have not seen before.

Table 4.3: Performance comparison of the selected regressors when applied to the imbalanced data set presented in Table 3.2.

	Linear Regression	Neural Network	Random Forest
RMSE	0.0076	0.0058	0.0024
R^2	0.8939	0.9377	0.9889

Surprisingly, linear regression does not show a poor performance when attempting to estimate the error in the drag coefficient. We expected to see worse outputs since we already know there are many nonlinear correlations between the input data and a linear model is not generally a good choice for these cases. Yet the linear regression model apparently detects many correlations correctly based on the values of RMSE and R^2. However, Random Forest still outperforms this simple linear regression due to its more advanced and capable structure.

Now, we compare the performance measures of the Random Forest regressor with a deep neural network structure. General configuration of the neural network is the same as the one we used for classification, with 2 hidden layers and 10 neurons in each. A rectifier activation function was utilized for these layers, while no activation was imposed on the output layer. The results demonstrate that the neural network performs better than the simple regression models such as the linear algorithm. The structure of the neural network is very flexible and one can tune the regressor by changing the number of hidden layers and neurons in each to further improve the predictions. It is likely to beat the performance of the Random Forest model by fine-tuning these parameters. However, for the same reasons mentioned in the previous chapter when developing the classifier, using a deep neural network when a more efficient algorithm such as Random Forest gives such high quality results does not sound rational.

4.7 Results: Learning Curve

Similar to classification, plotting the learning curve is a proper way to inspect the fitting condition of the regressor. General tips and guidelines on how to interpret this curve are provided in Section 3.9. Figure 4.3 presents the training and the test curves for the Random Forest regressor as the training samples are fed in. Unlike classification, the performance measure considered to plot the learning curve is RMSE. The performance of the model on the training set starts off from zero when only one training sample is considered. It then jumps to a large RMSE as more instances are introduced to the training process. Then, the performance improves as the number of the training data increases until it converges to a fixed value. This convergence demonstrates that the model is sufficiently trained and no additional training data are needed. The performance of the model on the test set starts from a large RMSE and gradually enhances as the model becomes more experienced. This curve reaches a point of stability where a small gap between the training and the test curves retain. This satisfactory behavior indicates that the Random Forest model is well-fitted to the training data and performs skillfully on the test samples.

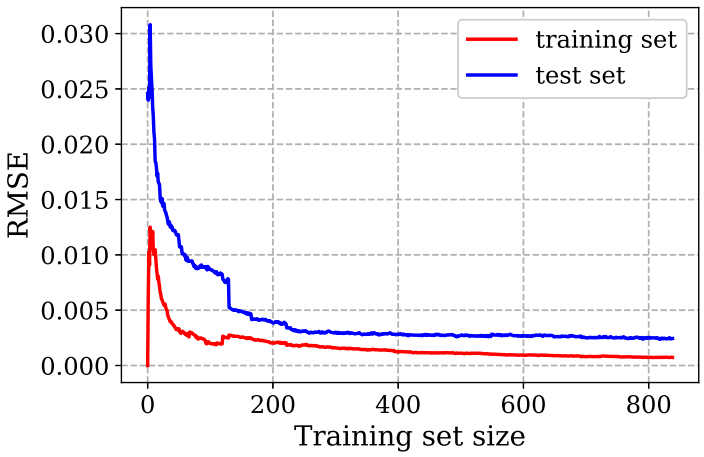


Figure 4.3: Learning curve of the developed Random Forest regressor.

4.8 Results: Feature Importance Analysis for Error Estimation

Beneficial properties of the Random Forest classification are transferred to its regression algorithm as well. One of these is the ability to assign weights to the input features to compare their contribution to decision making. Figure 4.4 illustrates this feature importance analysis for our regression model. From the figure, we derive the following conclusions:

- Random Forest identifies PC_{16} as the major decisive factor to estimate the error in the drag coefficient. PC_1 and PC_{13} are respectively marked as the next important features. The Mach number is fully represented by PC_1, and hence we ignore its importance shown in the plot. These results are consistent with our previous analysis on the correlation matrix presented in Fig. 4.2. We see that relatively fewer parameters appear to be significant for the regression task compared to the *flow feature detector*. This is because the target value considered here is only one of the many outcomes associated with the missing flow pattern. This automatically makes our regression less complex than our classification. It is worth noting that the main feature in decision making for both the regressor and the classifier was proved to be PC_{16}. It indicates that the error in the drag is the major effect of missing flow separation in a simulation around a 2-D airfoil.

- Similar to the classifier results in Fig. 3.13, we notice that the angle of attack and the Reynolds number play negligible roles in estimating the error. It appears that the other flow condition, namely the Mach number, is an important feature for the regressor. This is a false impression as we already know that the Mach number is directly correlated with PC_1, and having one of these in the set of important features suffices. This is where the correlation matrix helps us to avoid possible misinterpretations when examining the feature importance plot.

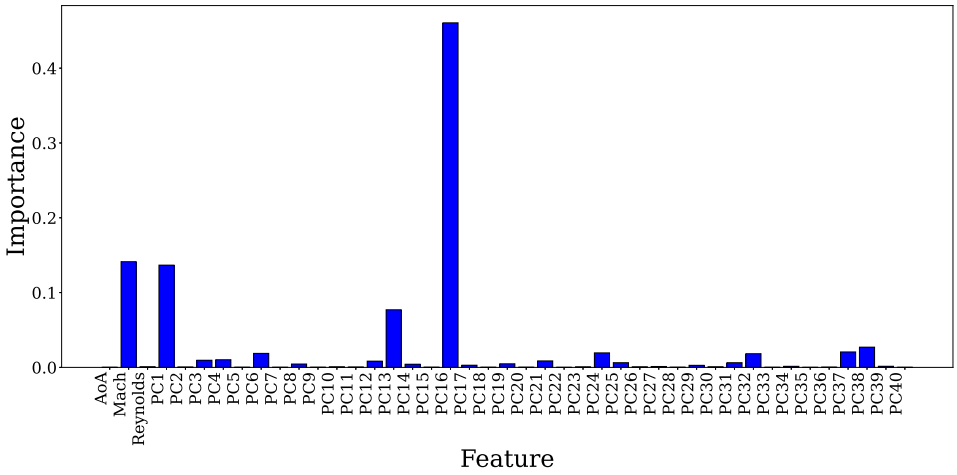


Figure 4.4: Feature importance of the Random Forest regressor.

As a comparison, we provide the feature importance plot of the linear regression algorithm applied to the data set in Fig. 4.5. The weights in the summation written for linear regression, presented in Eq. (4.1), show the importance of their corresponding input features. We notice that the linear model decides mainly based on the value assigned to PC_1 (or, the Mach number). This is inconsistent with the correlation matrix where we tagged PC_{16} as the most important input parameter, while linear regressor completely neglects the input features other than PC_1. In our case, the Mach number is indeed a major factor ruling the error, and hence the result of the linear regression model presented in Table 4.3 is not very poor. However, linear regression totally misses the most significant point of this study, that PC_{16} is the main source of the missing flow separation and its consequent error in drag. Hence, once again Random Forest is proved to be the suitable choice for this study.

These results help us to perform some postprocessing assessments and also in development of an efficient regressor, as explained in the following.

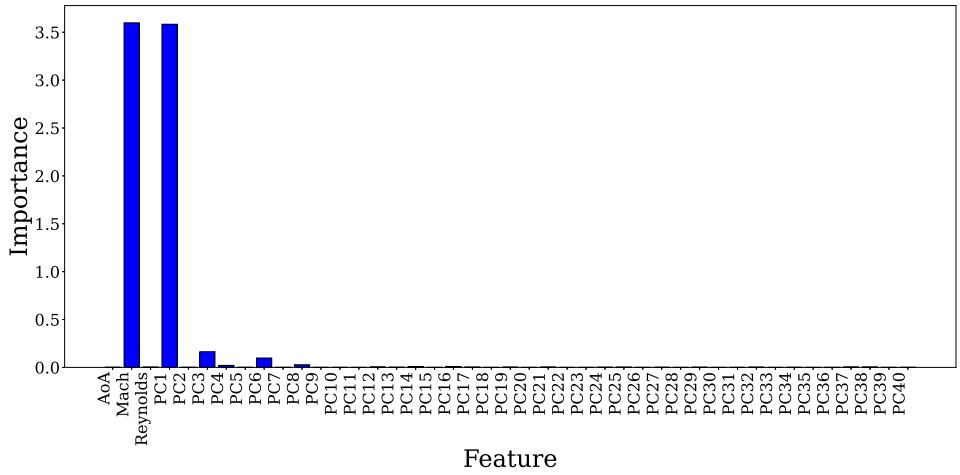
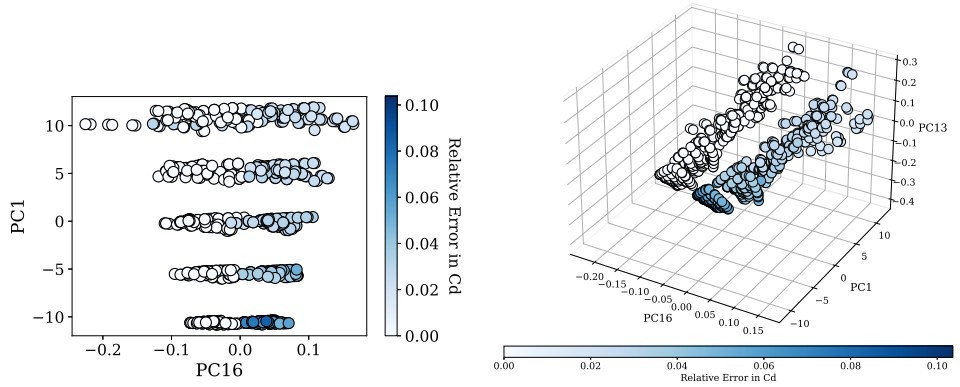


Figure 4.5: Feature importance of the linear regressor.

4.9 Results: Data Clusters

The feature importance analysis narrowed down the 43-dimensional vector of input features to only three major parameters, namely PC_{16}, PC_1, and PC_{13}, ordered by their relative importance. This becomes extremely useful when trying to inspect data points visually. Figures 4.6a and 4.6b depict the position of the data samples in the 2-D plane of PC_{16} and PC_1 and the 3-D space of all important features. The points are colored by the true value of their relative error in the drag coefficient. We clearly see a trend in both plots, where certain regions exhibit higher errors. We can also notice why PC_{16} is considered as the main feature, as it separates data points having lower-value errors from higher-value ones. These plots roughly visualize how the Random Forest regressor operates in these subspaces to assign a value to a new data point.

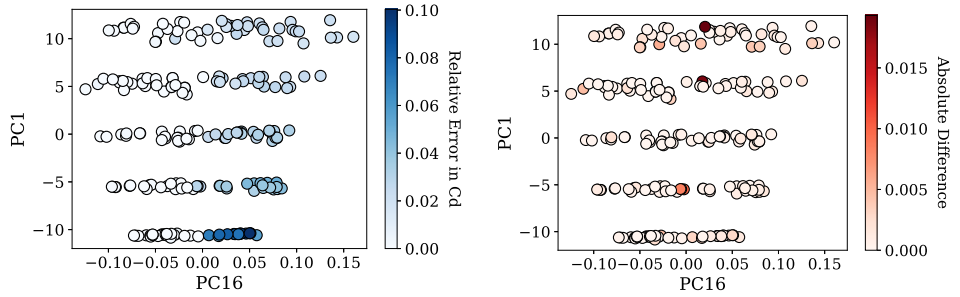
In Figs. 4.7a and 4.8a, the test samples colored by the error values predicted by the Random Forest regressor are plotted in the same 2-D and 3-D spaces mentioned before. These plots indicate that the model correctly identifies clusters shown in Fig. 4.6. To better visualize how the model performs in assigning error values to each of the data points, we must compare the predicted values with the actual ones. This is done by computing



(a) 2-D plane of PC_{16} and PC_1 **(b) 3-D space of PC_{16}, PC_{13}, and PC_1**

Figure 4.6: Positions of the data samples, colored by their corresponding true target values.

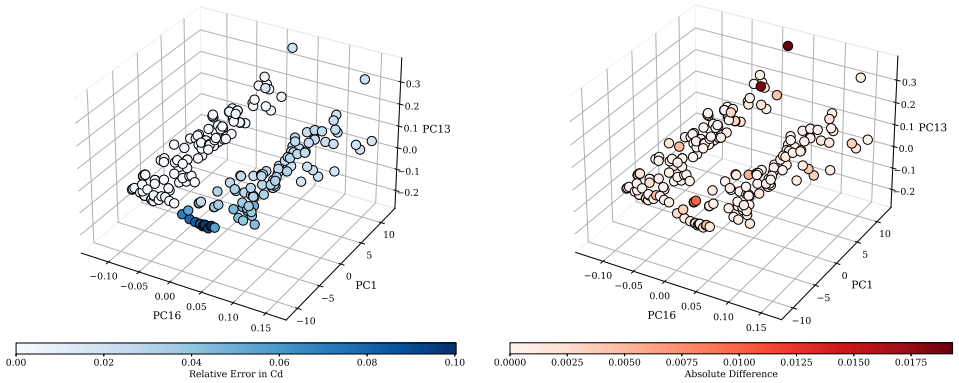
the absolute difference between these values and picturing the same data points colored by this difference. The results of such analysis are shown in Figs. 4.7b and 4.8b. We see that only a handful of data points show large differences between the predicted and the true target values, and in most cases, the model does an outstanding job in predicting the error.



(a) Colored by the predicted values

(b) Colored by the absolute difference between the predicted and the actual values

Figure 4.7: Performance of the Random Forest regressor on the test samples, visualized in the 2-D plane of PC_{16} and PC_1.



(a) Colored by the predicted values

(b) Colored by the absolute difference between the predicted and the actual values

Figure 4.8: Performance of the Random Forest regressor on the test samples, visualized in the 3-D space of PC_{16}, PC_{13}, and PC_1.

4.10 Results: Efficient Regressor

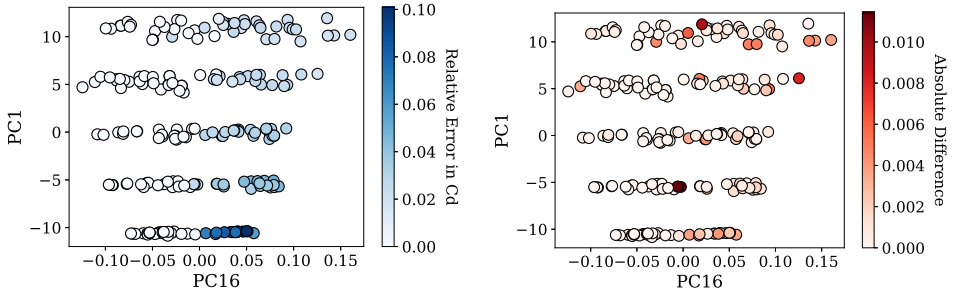
We can perform a feature selection on the input vector based on the results we got from the feature importance analysis. Therefore, we select PC_1, PC_{13}, and PC_{16} to develop a more efficient Random Forest regressor, and discard all other features. The combination of feature extraction and feature selection reduces the dimensionality of the 16568-dimensional complex CFD outputs to a 3-dimensional vector of features, which is quite impressive. It is substantially more convenient for a machine learning algorithm to operate in this low-dimensional space to detect relevant patterns shown in Fig. 4.6.

We train and evaluate this efficient model using the Random Forest algorithm on the same training and test sets as before. The performance of the model on the test samples is tabulated in Table 4.4. The evaluative measures demonstrate that the efficient regressor performs impressively well on the test set, and can be considered an acceptable error estimator. The value of RMSE suggests that the true target values are approximated with an average error of ± 0.21, when presented in percentage. In fact, it outperforms our main regressor where 43 input features were used instead of 3.

This interesting behavior is not unusual, since in many cases as we increase the number of irrelevant input information, the model could be confused by misleading correlations between the data. Here, when we use only the relevant input features, the model can predict the error values without any possible noise from other futile information. Overall, the model performs well enough to be marked as a reliable error estimator for our CFD data set. Predicted data clusters for the efficient model are plotted in Figs. 4.9 and 4.10.

Table 4.4: Performance comparison of the efficient Random Forest with the main model when applied to the imbalanced data set presented in Table 3.2.

	Main Model	Efficient Model
RMSE	0.0024	0.0021
R^2	0.9889	0.9918

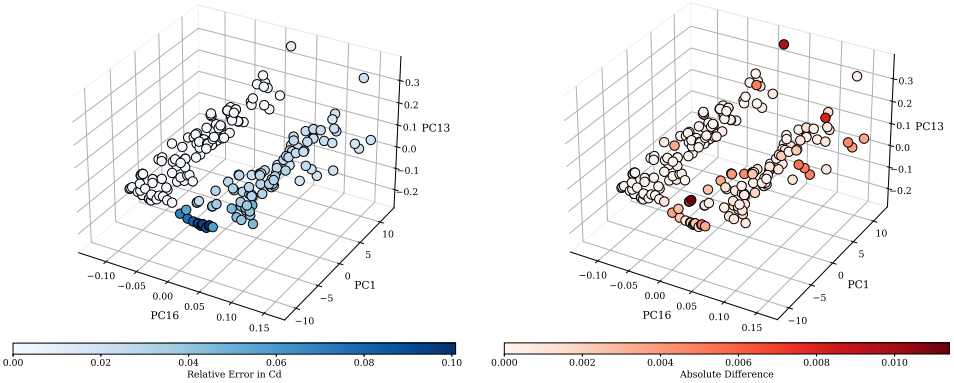


(a) Colored by the predicted values

(b) Colored by the absolute difference between the predicted and the actual values

Figure 4.9: Performance of the efficient Random Forest regressor on the test samples, visualized in the 2-D plane of PC_{16} and PC_1.

To assess whether the model is over-, under-, or well-fitted to the data, we analyze the output of the learning curve presented in Fig. 4.11. The training and test curves for the efficient model exhibit similar behavior to our main regressor, indicating that the model is a good fit to the data set. This proves that the efficient regressor developed based on only three input parameters is a skilled model in estimating the error in the drag coefficient.



(a) Colored by the predicted values

(b) Colored by the absolute difference between the predicted and the actual values

Figure 4.10: Performance of the efficient Random Forest regressor on the test samples, visualized in the 3-D space of PC_{16}, PC_{13}, and PC_1.

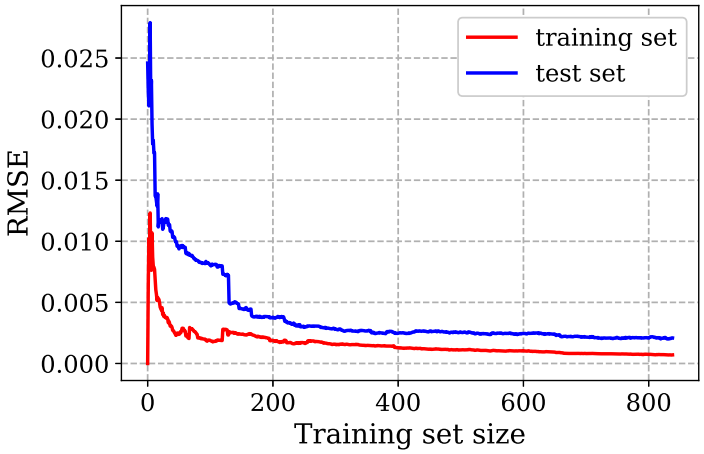


Figure 4.11: Learning curve of the efficient Random Forest regressor.

4.11 Results: Regressor Generalization

For the same reasons mentioned in the classification chapter, our developed regressor is only applicable to simulations performed on the base mesh presented in Fig. 1.10. To generalize the model, we again consider the arbitrary

mesh shown in Fig. 3.18. The generalization steps are exactly the same as before, where we need to map the solution from the arbitrary mesh onto the base mesh for dimension consistency. For a detailed explanation of the mapping process, refer to Section 3.13. Here, we only present the results of the generalization to see whether the model is able to estimate the error in the drag coefficient for a 2-D flow simulation around NACA 0012 performed on an arbitrary mesh.

We take the same simulations on the arbitrary mesh as the ones we used to generalize the classifier. The flow conditions are mentioned here again:

- the Angles of Attack (AoA): {-4, -1, 0, 2, 3},
- the Mach numbers (M): {0.2, 0.5},
- and the Reynolds numbers (Re): {4000, 8000}

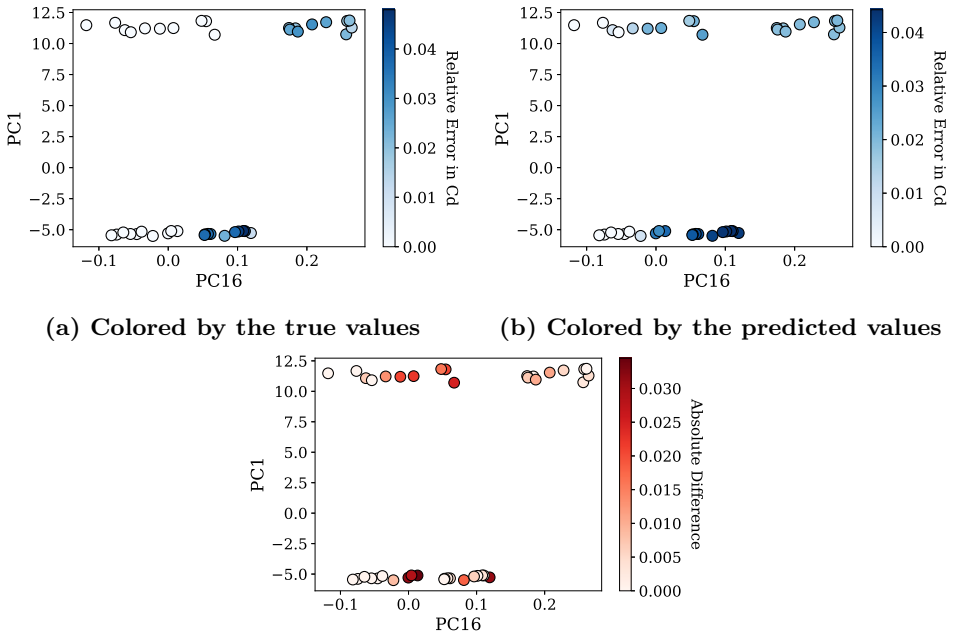
that yield 40 data samples to test generalization. The true target value is set to the relative error in the drag coefficient for each of the simulations, using the guidelines provided in Section 4.2. The results of applying the efficient Random Forest regressor on this new data set are provided in Table 4.5.

Table 4.5: Performance of the efficient Random Forest when applied to the generalized data samples.

RMSE	0.0135
R^2	0.3127

Evaluative measures indicate that the model is not performing as expected on the new solution vectors obtained on a different mesh than the one it was trained on. More specifically, the model yields an RMSE = 0.0135, or equivalently, an average ± 1.36 error margin when specified in percentage. This is a very poor performance compared to the results presented in the previous sections. Looking at the R^2 score, we conclude that the expansion coefficients do not represent the error in the drag coefficient for the new simulations sufficiently. It suggests that the PCA subspace created using the simulations performed on the base mesh is not a suitable information space

to estimate the error on other arbitrary meshes. This is surprising since our classification that shares similar properties to our regression showed acceptable results on the same generalized data set. We inspect the predicted values the same way we analyzed data clusters in previous sections, by presenting Figs. 4.12 and 4.13. Comparing the positions of the generalization data set in Figs. 4.12a and 4.13a with the main data set in Fig. 4.6, we detect a possible source of our failed attempt in generalizing the regressor. In Fig. 4.6a, we can easily identify a hard threshold at around $PC_{16} = 0.0$. Looking at the placement of the generalization data points in the same 2-D plane in Fig. 4.12a, we see that the aforementioned threshold does not exist anymore, as if the data points are slightly shifted to the right. This is concerning since the hard threshold we identified in the main data set plays an integral role in the decision making process of the model.

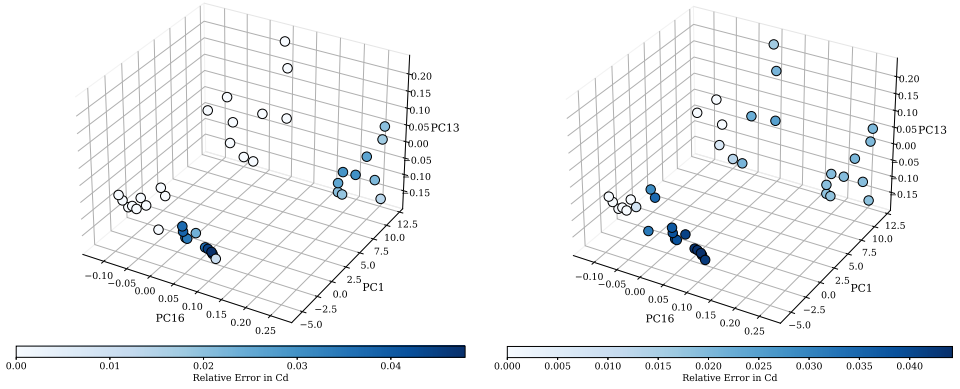


(a) Colored by the true values

(b) Colored by the predicted values

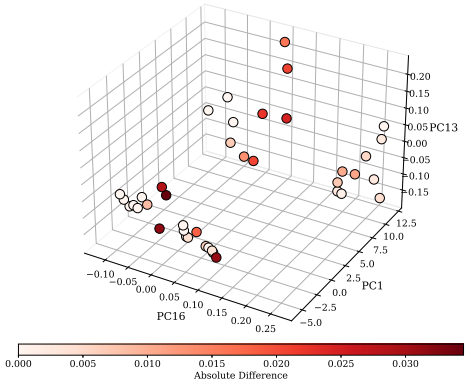
(c) Colored by the absolute difference between the predicted and the actual values

Figure 4.12: Performance of the efficient Random Forest regressor on the generalization samples, visualized in the 2-D plane of PC_{16} and PC_1.



(a) Colored by the true values

(b) Colored by the predicted values



(c) Colored by the absolute difference between the predicted and the actual values

Figure 4.13: Performance of the efficient Random Forest regressor on the generalization samples, visualized in the 3-D space of PC_{16}, PC_{13}, and PC_1.

In the following, we provide possible reasons for the poor performance we observed when generalizing the regressor:

- One possible source of the performance contrast between the classifier and the regressor could be related to the different evaluative measures for these two types of machine learning models. RMSE is a much more strict measure than the confusion matrix and its derived perfor-

mance metrics when dealing with poor individual predictions. In the case of classification, when a single data point is incorrectly labeled, it has minor effects on the overall performance of the model. When the number of incorrect labels increases, then we start to notice major differences in performance measures. On the other hand, RMSE penalizes the regressor severely even if only a single target value is poorly estimated. This comes from the numerical nature of the regression tasks. In fact, if we look at the predictions of the regressor when applied to the generalization samples, it shows good estimations in most of the simulations, while a few samples exhibit large discrepancies between the estimated and the true target values. It turns out that these large errors are sufficient to hugely affect the RMSE score, and mark the generalized model as a poor estimator.

- Another reason could be that the PCA subspace created based on the base mesh is fundamentally inconsistent with the new simulations. One major assumption in our analysis is that the same PCA subspace we used to develop the model can represent the arbitrary simulations just as accurate. However, it could be the case that the new set of simulations result in a different PCA subspace and projecting those on the old PCA modes is meaningless. To check this possible source of error, we repeated the whole PCA decomposition for the simulations on the arbitrary mesh shown in Fig. 3.18. We considered the same flow conditions as before to generate 1050 solution vectors, and performed the decomposition on this new data set. The goal is to compare this new PCA subspace with the one we developed our models on. Thus, we provide mode shapes and PCA coefficient plots for the new PCA subspace in Appendices D and E, respectively. Comparing the mode shapes and the coefficient plots with the ones provided in Appendices B, and C, we notice that the dominant modes appear the same for both spaces. In particular, the first and the second modes that respectively represent the Mach number and the angle of attack are identical. In the third one, we observe similar patterns for the two spaces with

a sign flip, where negative regions in one space appear positive in the other, and vice versa. In general, this is not a concern since the PCA algorithm has correctly detected one of the high-variance axes for both data sets, with opposite directions. As we compare less dominant modes, it appears that these modes do not match anymore. Especially, the 16^{th} mode in the old PCA subspace that was the major factor in our decision making does not have an identical counterpart in the new PCA subspace. This could be one of the reasons that the regressor performs poorly on the simulations on different meshes.

Chapter 5

Concluding Remarks

5.1 Summary

Missing flow features in CFD simulations, originating from inaccuracies involved in the numerical solver, impose a major challenge upon the application of CFD methods in the industry. The inability of the simulation in properly capturing these major patterns results in unreliable output parameters and directly affects the design and performance assessment in the postprocessing stage. The current accuracy improvement methods proposed to fill this gap, such as grid refinement and error estimators, can become computationally inefficient. Also, these methods do not provide a priori assessment of the quality of the features captured in the computational domain. Hence, an efficient alternative tool capable of evaluating the accuracy of CFD simulations both qualitatively and quantitatively could make a huge difference in computational aerodynamics.

In this thesis, we took advantage of the availability of the massive output data from CFD simulations to construct a purely data-driven framework to address the aforementioned challenges. The goal was to develop two intelligent machine learning models to:

- first, predict whether a simulation properly captures the dominant flow pattern,
- and second, estimate the error in one of the aerodynamic parameters calculated by the simulation.

In our analysis, the major flow pattern was the separation bubble behind a NACA 0012 airfoil and the drag coefficient was selected as the aerodynamic parameter as it is more likely to be affected by the separation.

CFD data sets suffer from the *curse of dimensionality*, as the output solutions usually contain many degrees of freedom. This makes it quite difficult for machine learning algorithms to detect patterns and identify useful correlations in this high-dimensional space of information. Therefore, we applied a dimensionality reduction technique, called Principal Component Analysis (PCA), to map the high-dimensional CFD solution vectors onto a low-dimensional PCA subspace. The resulting subspace consisted of a set of orthonormal modes, tailored to our specific data set, which created an ideal space for machine learning purposes. Apart from making the data considerably more efficient to work with, PCA was proved to be a powerful tool in extracting meaningful information from the flow field. In particular, we were able to identify the modes representing the effects of the angle of attack, the Mach number, the Reynolds number, and also the order of accuracy of the simulation. Furthermore, a single mode primarily responsible for the missing separation bubble was uncovered. PCA sheds light on every detail of the flow physics as well as the numerical aspects of the simulation, which in turn makes it a potent tool in many applications in aerodynamics, provided sufficient data are available.

For the qualitative part of our analysis, a machine learning classifier based on the extracted PCA modes was implemented with the aim of detecting the simulations where the separation bubble behind the airfoil was not well captured. The performance measures demonstrated that the supervised binary classifier developed in this study is an outstanding model in detecting the solution vectors lacking separation. A comparison between the performance of common machine learning algorithms was provided, and it was proved that Random Forest is the best model to train the *flow feature detector*. By performing a feature importance analysis, a more efficient classifier was developed with less input information, which was proved to be a comparable tool. We attempted to generalize the model to handle simulations on different meshes by mapping solution vectors obtained on an arbitrary mesh onto our base mesh. The results demonstrated that the generalized classifier makes acceptable predictions, and can be used for any 2-D flow simulations around NACA 0012 airfoil regardless of the mesh configu-

raion. The proposed approach could be used as a priori examination tool to identify deficiencies in a CFD simulation, before attempting to improve its accuracy.

We extended the classifier and developed a supervised machine learning regressor based on the same information space for uncertainty quantification. In particular, we set the relative error in the drag coefficient as the target value to implement an alternative approach to the current error estimators such as Error Transport Equation (ETE) and Adjoint methods. The performance measures indicated that our regressor estimates the errors with remarkable accuracy. The regression was trained using the Random Forest algorithm, and a comparison between this model and other machine learning techniques was provided. Similar to the classification, we performed a feature importance analysis to reduce the dimensionality of the input vector even further using feature selection. The resulting efficient regressor exhibited better performance compared to the main model, possibly because of the better input information. Our generalization to make the regressor accept flow simulations on different meshes, however, was not as successful. We thoroughly discussed possible issues related to regressor generalization. This new error estimator based on machine learning regression has the potential to replace the current costly accuracy improvement methods, if properly generalized. It can be used concurrently with the classifier to assign a predicted error value to the CFD simulations that miss a major flow pattern. This way, our machine learning models that operate based on PCA modes provide both qualitative and quantitative assessment tools.

5.2 Key Outcomes

This work is very much at the proof of concept stage, with many features that are not scalable to large industrial problems. Nonetheless, we think there are valuable lessons to be learned from our progress thus far. Here, we discuss the key findings of the present study:

- We performed a modal decomposition based on PCA on the CFD sim-

ulations around a NACA 0012 airfoil. The resulting PCA subspace provides a suitable framework to train different machine learning and deep learning models efficiently. Besides learning tasks, PCA modes are physically interpretable and each holds a great amount of information about the simulated flow field. Since PCA yields an orthonormal basis, the modes are linearly independent, and therefore, each mode can be analyzed separately. This might become extremely useful when a single aspect of the simulation is to be studied. Using PCA, we can filter out all redundant information and only focus on the mode(s) representing the particular characteristic we are interested in. Our full data set and its PCA modes, as well as other useful information are all available in an open-source GitHub repository[4]. This data set can be used to develop various machine learning models for several applications or to study different aspects of the simulation by inspecting PCA modes.

- We developed two supervised machine learning models on the PCA subspace to tackle one of the grave challenges in CFD. The models were successful in predicting the missing flow feature and estimating the error in the drag coefficient with outstanding accuracy. The proposed data-driven approach examines inaccuracies involved in CFD simulations without introducing additional equations and complexities into the process; all we need to do is to get the simulation output and pass it to our machine learning pipeline to yield a qualitative and quantitative assessment of the simulation. Although there still remain challenges regarding generalization, the developed models manifested great potential for efficient accuracy analysis in numerical simulations.

5.3 Future Directions

The next steps to further develop the proposed machine learning approach can be listed as follows:

[4] `https://github.com/APHedayat/missing-flow-feature-dataset.git`

- In this project, the focus was on the missing separation bubble behind an airfoil. However, the proposed method is not limited to this particular flow feature, and similar classifiers and regressors can be trained to deal with other major flow patterns, such as shock waves. It is expected that the effects of any dominant flow feature would be well captured in its PCA subspace. This is because PCA scans all values in every single cell in the domain, and the major patterns in the flow field show themselves within these values. As a result, if the flow feature has enough energy to considerably affect the flow field, PCA will represent it among its leading modes. Therefore, using the same flowcharts presented in Figs. 3.4 and 4.1 for a set of simulations with different missing features, we can develop other models to deal with inaccuracies in those particular cases.

- We faced serious issues when attempting to generalize the regressor to make it independent of the mesh configuration. By performing further analysis provided in Appendices D and E, we noticed that the PCA modes for similar simulations performed on different meshes differ. This weakens our generalization idea where we map the arbitrary solution vector on the base mesh on which PCA was performed. There are still uncertainties attached to our generalization, and a successful idea to make our machine learning models independent of the mesh would be delightful.

- The ultimate target of this project are costly three-dimensional (3-D) applications. Such 3-D simulations with complex geometries will reveal the true potential of this method, since utilizing accuracy improvement techniques in these cases becomes computationally heavy, and in some cases, infeasible. Having a classifier to decide whether to apply such costly methods would save us valuable time. On the other hand, developing a regressor to work in parallel with the classifier and act as an alternative to the current improvement techniques can eliminate such costly methods from the simulation cycle and save us further time in 3-D cases.

- The created data set with labeled simulations (captured or missed) can be a reference for other data-driven or pattern recognition approaches. Labeling such a huge number of solution vectors manually is a tedious task, and therefore we have provided the data set in a GitHub repository for anyone interested in participating (URL address was provided previously). This work only explores one data-driven approach to tackle the problem of missing flow features in CFD simulations. We do not claim that our developed models are the best possible solutions to this problem, and further endeavors using various methods to provide a comparing platform with the models developed in this thesis will advance the use of data-driven approaches in CFD.

- The logical path to take from now on, in our opinion, is to focus more on the interesting modes we find during the process, instead of developing other machine learning models that are not scalable to real-world problems. In this thesis, for instance, we found one particular PCA mode that was the source of the missing flow separation in our test cases. Using the same data-driven approach, we can find different modes related to other missing patterns, such as shock waves. The reason that these modes could become helpful in the future is that they can be thought of as footprints of the numerical errors involved in the simulation that are causing certain patterns to be missed. Therefore, we believe there is more value in further studying these modes and discover possible ways to take advantage of these to reduce numerical errors in general flow simulations.

Overall, the data-driven framework developed in this study was proved to have a great potential to be used for accuracy improvement in numerical simulations, and hopefully, this work opens more opportunities to develop capable machine learning models or other novel approaches for CFD applications.

Bibliography

[1] Moore, G. E., "Cramming more components onto integrated circuits," *Electronics*, Vol. 38, No. 8, 1965, p. 114.

[2] Hey, T., Tansley, S., and Tolle, K., *The Fourth Paradigm: Data-Intensive Scientific Discovery*, Microsoft Research, Redmond, WA, 2009.

[3] Brunton, S. L., Kutz, J. N., Manohar, K., Aravkin, A. Y., Morgansen, K., Klemisch, J., Goebel, N., Buttrick, J., Poskin, J., Blom-Schieber, A. W., Hogan, T., and McDonald, D., "Data-Driven Aerospace Engineering: Reframing the Industry with Machine Learning," *AIAA Journal*, Vol. 59, No. 8, 2021, pp. 2820–2847. https://doi.org/10.2514/1.J060131.

[4] Ollivier-Gooch, C., "An Unstructured Mesh Improvement Toolkit with Application to Mesh Improvement, Generation and (De-)Reinement," *36th AIAA Aerospace Sciences Meeting and Exhibit*, American Institute of Aeronautics and Astronautics, Reston, Virigina, 1998. https://doi.org/10.2514/6.1998-218.

[5] Ollivier-Gooch, C. F., "GRUMMP Version 0.7.0 User's Guide," Tech. rep., Department of Mechanical Engineering, The University of British Columbia, 2016.

[6] Hirsch, C., *Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics*, 2nd ed., Elsevier, 2007.

[7] Ollivier-Gooch, C. F., and Van Altena, M., "A High-Order Accurate Unstructured Mesh Finite-Volume Scheme for the Advection-Diffusion Equation," *Journal of Computational Physics*, Vol. 181, No. 2, 2002, pp. 729–752. https://doi.org/10.1006/jcph.2002.7159.

[8] Stroud, A. H., and Secrest, D., *Gaussian Quadrature Formulas*, Englewood Cliffs, N.J., 1966.

[9] Van Altena, M., "High-Order Finite-Volume Discretizations for Solving a Modified Advection-Diffusion Problem on Unstructured Triangular Meshes," Ph.D. thesis, The University of British Columbia, 1999.

[10] Nishikawa, H., "Beyond Interface Gradient: A General Principle for Constructing Diffusion Schemes," *40th AIAA Fluid Dynamics Conference and Exhibit*, 2010. https://doi.org/10.2514/6.2010-5093.

[11] Michalak, C., and Ollivier-Gooch, C., "Globalized matrix-explicit Newton-GMRES for the high-order accurate solution of the Euler equations," *Computers and Fluids*, Vol. 39, No. 7, 2010, pp. 1156–1167. https://doi.org/10.1016/j.compfluid.2010.02.008.

[12] Wilcox, D., "Turbulence modeling - An overview," *39th Aerospace Sciences Meeting and Exhibit*, AIAA, Reno, NV, 2001. https://doi.org/10.2514/6.2001-724.

[13] Ling, J., Kurzawski, A., and Templeton, J., "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance," *Journal of Fluid Mechanics*, Vol. 807, 2016, pp. 155–166. https://doi.org/10.1017/jfm.2016.615.

[14] Duraisamy, K., Iaccarino, G., and Xiao, H., "Turbulence Modeling in the Age of Data," *Annual Review of Fluid Mechanics*, Vol. 51, 2019, pp. 357–377. https://doi.org/10.1146/annurev-fluid-010518-040547.

[15] Beck, A., and Kurz, M., "A perspective on machine learning methods in turbulence modeling," *GAMM Mitteilungen*, Vol. 44, No. 1, 2021, pp. 1–27. https://doi.org/10.1002/gamm.202100002.

[16] Jalali, A., Sharbatdar, M., and Ollivier-Gooch, C., "Accuracy analysis of unstructured finite volume discretization schemes for diffusive fluxes," *Computers and Fluids*, Vol. 101, 2014, pp. 220–232. https://doi.org/10.1016/j.compfluid.2014.06.008.

[17] Richardson, L. F., "The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, Vol. 210, No. 459-470, 1911, pp. 307–357.

[18] Richardson, L. F., and Gaunt, J. A., "The deferred approach to the limit. Part I. Single lattice. Part II. Interpenetrating lattices," *Philosophical Transactions of the Royal Society London A*, Vol. 226, 1927, pp. 299–361.

[19] Vassberg, J., DeHaan, M., and Sclafani, T., "Grid Generation Requirements for Accurate Drag Predictions Based on OVERFLOW Calculations," *16th AIAA Computational Fluid Dynamics Conference*, American Institute of Aeronautics and Astronautics, Orlando, Florida, 2003. https://doi.org/10.2514/6.2003-4124.

[20] Roache, P. J., "Perspective: A Method for Uniform Reporting of Grid Refinement Studies," *Journal of Fluids Engineering, Transactions of the ASME*, Vol. 116, No. 3, 1994, pp. 405–413. https://doi.org/10.1115/1.2910291.

[21] Roache, P. J., "Verification of Codes and Calculations," *AIAA Journal*, Vol. 36, No. 5, 1998, pp. 696–702. https://doi.org/10.2514/2.457.

[22] Vassberg, J. C., and Jameson, A., "In Pursuit of Grid Convergence for Two-Dimensional Euler Solutions," *Journal of Aircraft*, Vol. 47, No. 4, 2010, pp. 1152–1166. https://doi.org/10.2514/1.46737.

[23] Mavriplis, C., "Adaptive mesh strategies for the spectral element method," *Computer Methods in Applied Mechanics and Engineer-*

ing, Vol. 116, No. 1-4, 1994, pp. 77–86. https://doi.org/10.1016/S0045-7825(94)80010-3.

[24] Nemec, M., Aftosmis, M., and Wintzer, M., "Adjoint-Based Adaptive Mesh Refinement for Complex Geometries," *46th AIAA Aerospace Sciences Meeting and Exhibit*, American Institute of Aeronautics and Astronautics, Reno, Nevada, 2008. https://doi.org/10.2514/6.2008-725.

[25] Hartmann, R., Held, J., and Leicht, T., "Adjoint-based error estimation and adaptive mesh refinement for the RANS and k–x turbulence model equations," *Journal of Computational Physics*, Vol. 230, No. 11, 2011, pp. 4268–4284. https://doi.org/10.1016/j.jcp.2010.10.026.

[26] Paudel, S., and Saenger, N., "Grid refinement study for three dimensional CFD model involving incompressible free surface flow and rotating object," *Computers and Fluids*, Vol. 143, 2017, pp. 134–140. https://doi.org/10.1016/j.compfluid.2016.10.025.

[27] Diskin, B., Thomas, J. L., Rumsey, C. L., and Schwöppe, A., "Grid-Convergence of Reynolds-Averaged Navier–Stokes Solutions for Benchmark Flows in Two Dimensions," *AIAA Journal*, Vol. 54, No. 9, 2016, pp. 2563–2588. https://doi.org/10.2514/1.J054555.

[28] Diskin, B., Anderson, W. K., Pandya, M. J., Rumsey, C. L., Thomas, J. L., Liu, Y., and Nishikawa, H., "Grid Convergence for Three Dimensional Benchmark Turbulent Flows," *AIAA Aerospace Sciences Meeting*, 2018, pp. 1–35. https://doi.org/10.2514/6.2018-1102.

[29] Mavriplis, D. J., "Results from the 3rd Drag Prediction Workshop Using the NSU3D Unstructured Mesh Solver," *Journal of Aircraft*, Vol. 45, No. 3, 2008, pp. 750–761. https://doi.org/10.2514/1.29828.

[30] Spalart, P. R., and Venkatakrishnan, V., "On the role and challenges of CFD in the aerospace industry," *The Aeronautical Journal*, Vol. 120, No. 1223, 2016, pp. 209–232. https://doi.org/10.1017/aer.2015.10.

[31] Giles, M. B., and Pierce, N. A., "Adjoint equations in CFD: duality, boundary conditions and solution behaviour," *13th Computational Fluid Dynamics Conference*, 1997, pp. 182–198. https://doi.org/10.2514/6.1997-1850.

[32] Pierce, N. A., and Giles, M. B., "Adjoint Recovery of Superconvergent Functionals from PDE Approximations," *SIAM Review*, Vol. 42, No. 2, 2000, pp. 247–264. https://doi.org/10.1137/S0036144598349423.

[33] Sharbatdar, M., and Ollivier-Gooch, C., "Adjoint-Based Functional Correction for Unstructured Mesh Finite Volume Methods," *Journal of Scientific Computing*, Vol. 76, No. 1, 2018, pp. 1–23. https://doi.org/10.1007/s10915-017-0611-8.

[34] Venditti, D. A., and Darmofal, D. L., "Adjoint Error Estimation and Grid Adaptation for Functional Outputs: Application to Quasi-One-Dimensional Flow," *Journal of Computational Physics*, Vol. 164, No. 1, 2000, pp. 204–227. https://doi.org/10.1006/jcph.2000.6600.

[35] Pierce, N. A., and Giles, M. B., "Adjoint and defect error bounding and correction for functional estimates," *Journal of Computational Physics*, Vol. 200, No. 2, 2004, pp. 769–794. https://doi.org/10.1016/j.jcp.2004.05.001.

[36] Venditti, D. A., and Darmofal, D. L., "Grid Adaptation for Functional Outputs: Application to Two-Dimensional Inviscid Flows," *Journal of Computational Physics*, Vol. 176, No. 1, 2002, pp. 40–69. https://doi.org/10.1006/jcph.2001.6967.

[37] Venditti, D. A., and Darmofal, D. L., "Anisotropic grid adaptation for functional outputs: application to two-dimensional viscous flows," *Journal of Computational Physics*, Vol. 187, No. 1, 2003, pp. 22–46. https://doi.org/10.1016/S0021-9991(03)00074-3.

[38] Kast, S. M., "An Introduction to Adjoints and Output Error Estimation in Computational Fluid Dynamics," Ph.D. thesis, University of Michigan, 2016. URL http://arxiv.org/abs/1712.00693.

[39] Sharbatdar, M., "Error Estimation and Mesh Adaptation Paradigm for Unstructured Mesh Finite Volume Methods," Ph.D. thesis, The University of British Columbia, 2017.

[40] Tyson, W. C., Świrydowicz, K. K., Derlaga, J. M., Roy, C. J., and de Sturler, E., "Improved Functional-Based Error Estimation and Adaptation without Adjoints," *46th AIAA Fluid Dynamics Conference*, 2016, pp. 1–18. https://doi.org/10.2514/6.2016-3809.

[41] Kin Yan, G. K., and Ollivier-Gooch, C., "Discretization Error Estimation by the Error Transport Equation on Unstructured Meshes — Applications to Viscous Flows," *54th AIAA Aerospace Sciences Meeting*, 2016, pp. 1–18. https://doi.org/10.2514/6.2016-0836.

[42] Yan, G., and Ollivier-Gooch, C. F., "Towards higher order discretization error estimation by error transport using unstructured finite-volume methods for unsteady problems," *Computers and Fluids*, Vol. 154, 2017, pp. 245–255. https://doi.org/10.1016/j.compfluid.2017.06.012.

[43] Derlaga, J. M., Park, M. A., and Rallabhandi, S. K., "Application of Exactly Linearized Error Transport Equations to AIAA CFD Prediction Workshops," *AIAA SciTech Forum - 55th AIAA Aerospace Sciences Meeting*, 2017, pp. 1–24. https://doi.org/10.2514/6.2017-0076.

[44] Wang, H., Tyson, W. C., and Roy, C. J., "Discretization Error Estimation for Discontinuous Galerkin Methods using Error Transport Equations," *AIAA Scitech Forum*, 2019, pp. 1–13. https://doi.org/10.2514/6.2019-2173.

[45] Yan, G., and Ollivier-Gooch, C., "Applications of the unsteady error transport equation on unstructured meshes," *AIAA Journal*, Vol. 56, No. 11, 2018, pp. 4463–4473. https://doi.org/10.2514/1.J057024.

[46] Yan, G., "Numerical Estimation of Discretization Error on Unstructured Meshes," Ph.D. thesis, The University of British Columbia, 2018. URL https://open.library.ubc.ca/collections/24/items/1.0364237.

[47] Zhao, W., Chellappa, R., Phillips, P. J., and Rosenfeld, A., "Face Recognition: A Literature Survey," *ACM computing surveys*, Vol. 35, No. 4, 2003, pp. 399–458. https://doi.org/10.1145/954339.954342.

[48] Tolba, A., El-Baz, A., and El-Harby, A., "Face Recognition: A Literature Review," *International Journal of Signal Processing*, Vol. 2, No. 2, 2006, pp. 88–103. https://doi.org/10.5281/zenodo.1334652.

[49] Li, S. Z., and Jain, A. K., *Handbook of Face Recognition*, Springer, 2011.

[50] Parkhi, O. M., Vedaldi, A., and Zisserman, A., "Deep Face Recognition," *Proceedings of the British Machine Vision Conference (BMVC)*, BMVA Press, 2015, pp. 41.1–41.12. https://doi.org/10.5244/C.29.41.

[51] LeCun, Y., and Bengio, Y., "Convolutional Networks for Images, Speech, and Time-Series," *The handbook of brain theory and neural networks*, 1998, pp. 255–258.

[52] Rawat, W., and Wang, Z., "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review," *Neural Computation*, Vol. 29, No. 9, 2017, pp. 2352–2449. https://doi.org/10.1162/neco_a_00990.

[53] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., and Chen, T., "Recent advances in convolutional neural networks," *Pattern Recognition*, Vol. 77, 2018, pp. 354–377. https://doi.org/10.1016/j.patcog.2017.10.013.

[54] Abras, J., and Hariharan, N. S., "Application of Machine Learning to Automate Vortex Core Extraction Computations in Hovering Rotor Wakes," *AIAA Aviation 2021 Forum*, American Institute of Aeronautics and Astronautics, 2021. https://doi.org/10.2514/6.2021-2595.

[55] Abras, J., and Hariharan, N. S., "Machine Learning Based Physics Inference from High-Fidelity Solutions: Vortex Core Data Extraction,"

AIAA Scitech 2022 Forum, American Institute of Aeronautics and Astronautics, San Diego, CA, 2022. https://doi.org/10.2514/6.2022-1683.

[56] Balay, S., Gropp, W., McInnes, L. C., and Smith, B. F., "PETSc, the portable, extensible toolkit for scientific computation," *Argonne National Laboratory*, Vol. 2, No. 17, 1998.

[57] Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, No. 2, 1981, pp. 357–372. https://doi.org/10.1016/0021-9991(81)90128-5.

[58] Rowley, C. W., "Model Reduction for Fluids, using Balanced Proper Orthogonal Decomposition," *International Journal of Bifurcation and Chaos*, Vol. 15, No. 03, 2005, pp. 997–1013. https://doi.org/10.1142/S0218127405012429.

[59] Berguin, S. H., and Mavris, D. N., "Dimensionality Reduction Using Principal Component Analysis Applied to the Gradient," *AIAA Journal*, Vol. 53, No. 4, 2015, pp. 1078–1090. https://doi.org/10.2514/1.J053372.

[60] Salih Hasan, B. M., and Abdulazeez, A. M., "A Review of Principal Component Analysis Algorithm for Dimensionality Reduction," *Journal of Soft Computing and Data Mining*, Vol. 02, No. 01, 2021, pp. 20–30. https://doi.org/10.30880/jscdm.2021.02.01.003.

[61] Zhang, Q., Liu, Y., and Wang, S., "The identification of coherent structures using proper orthogonal decomposition and dynamic mode decomposition," *Journal of Fluids and Structures*, Vol. 49, 2014, pp. 53–72. https://doi.org/10.1016/j.jfluidstructs.2014.04.002.

[62] Pearson, K., "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, Vol. 2, No. 7-12, 1901, pp. 559–572.

[63] Hotelling, H., "Analysis of a complex of statistical variables into principal components," *Journal of Educational Psychology*, Vol. 24, 1933, pp. 417–441.

[64] Watanabe, S., *Pattern Recognition: Human and Mechanical*, John Wiley & Sons, Inc., 1985.

[65] Lumley, J. L., "The Structure of Inhomogeneous Turbulent Flows," *Proceedings of the International Colloquium on the Fine Scale Structure of the Atmosphere and Its Influence on Radio Wave Propagation*, Moscow, 1967.

[66] Lang, Y. D., Malacina, A., Biegler, L. T., Munteanu, S., Madsen, J. I., and Zitney, S. E., "Reduced Order Model Based on Principal Component Analysis for Process Simulation and Optimization," *Energy and Fuels*, Vol. 23, No. 3, 2009, pp. 1695–1706. https://doi.org/10.1021/ef800984v.

[67] Lange, A., Voigt, M., Vogeler, K., and Johann, E., "Principal component analysis on 3D scanned compressor blades for probabilistic CFD simulation," *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, 2012. https://doi.org/10.2514/6.2012-1762.

[68] Aversano, G., Bellemans, A., Li, Z., Coussement, A., Gicquel, O., and Parente, A., "Application of reduced-order models based on PCA & Kriging for the development of digital twins of reacting flow applications," *Computers and Chemical Engineering*, Vol. 121, 2019, pp. 422–441. https://doi.org/10.1016/j.compchemeng.2018.09.022.

[69] Scherl, I., Strom, B., Shang, J. K., Williams, O., Polagye, B. L., and Brunton, S. L., "Robust principal component analysis for modal decomposition of corrupt fluid flows," *Physical Review Fluids*, Vol. 5, No. 5, 2020. https://doi.org/10.1103/PhysRevFluids.5.054401.

[70] Taira, K., Brunton, S. L., Dawson, S. T., Rowley, C. W., Colonius, T., McKeon, B. J., Schmidt, O. T., Gordeyev, S., Theofilis, V., and Ukeiley, L. S., "Modal Analysis of Fluid Flows: An Overview," *AIAA Journal*, Vol. 55, No. 12, 2017, pp. 4013–4041. https://doi.org/10.2514/1.J056060.

[71] Taira, K., Hemati, M. S., Brunton, S. L., Sun, Y., Duraisamy, K., Bagheri, S., Dawson, S. T., and Yeh, C. A., "Modal Analysis of Fluid Flows: Applications and Outlook," *AIAA Journal*, Vol. 58, No. 3, 2020, pp. 998–1022. https://doi.org/10.2514/1.J058462.

[72] Brunton, S. L., and Kutz, J. N., *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*, Cambridge University Press, New York, NY, 2019.

[73] Eckart, C., and Young, G., "The approximation of one matrix by another of lower rank," *Psychometrika*, Vol. 1, No. 3, 1936, pp. 211–218.

[74] Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development*, Vol. 3, No. 3, 1959, pp. 210–229. https://doi.org/10.1147/rd.33.0210.

[75] Mitchell, T. M., *Machine Learning: A multistrategy approach*, McGraw-Hill Science, Engineering, & Math, 1997.

[76] Goodfellow, I. J., Bengio, Y., and Courville, A., *Deep Learning*, MIT Press, Cambridge, MA, 2016. URL http://www.deeplearningbook.org.

[77] Geiron, A., *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems*, 2nd ed., O'Reilly, CA, 2019.

[78] Glorot, X., Bordes, A., and Bengio, Y., "Deep Sparse Rectifier Neural Networks," *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, Vol. 15, PMLR, 2011, pp. 315–323. https://doi.org/10.1002/ecs2.1832.

[79] Kingma, D. P., and Ba, J. L., "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015. https://doi.org/10.48550/arXiv.1412.6980.

[80] Ho, T. K., "Random decision forests," *Proceedings of 3rd International Conference on Document Analysis and Recognition*, Vol. 1, IEEE, 1995, pp. 278–282. https://doi.org/10.1109/ICDAR.1995.598994.

[81] Breiman, L., "Random Forests," *Machine Learning*, Vol. 45, 2001, pp. 5–32. https://doi.org/10.1023/A:1010933404324.

[82] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, Vol. 12, No. 85, 2011, pp. 2825–2830. URL http://jmlr.org/papers/v12/pedregosa11a.html.

[83] Lemaître, G., Nogueira, F., and Aridas, C. K., "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning," *Journal of Machine Learning Research*, Vol. 18, No. 17, 2017, pp. 1–5. URL http://jmlr.org/papers/v18/16-365.html.

[84] learn developers, S., "scikit-learn user guide Release 0.18.2," Tech. rep., 2017. URL https://scikit-learn.org/0.18/_downloads/scikit-learn-docs.pdf.

[85] Hosmer Jr, D. W., Lemeshow, S., and Sturdivant, R. X., *Applied logistic regression*, Vol. 398, John Wiley & Sons, 2013.

[86] Davis, J., and Goadrich, M., "The Relationship Between Precision-Recall and ROC Curves," *Proceedings of the 23rd International Conference on Machine Learning*, Vol. 148, 2006, pp. 233–240. https://doi.org/10.1145/1143844.1143874.

[87] Saito, T., and Rehmsmeier, M., "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets," *PLoS ONE*, Vol. 10, No. 3, 2015. https://doi.org/10.1371/journal.pone.0118432.

Appendix A

Eigendecomposition

Eigendecomposition provides another means to break down a CFD simulation into a set of modes. Unlike PCA, this approach does not require a data set, and can be applied directly on the Jacobian operator in the linearized system of equations. Therefore, eigendecomposition is considerably easier to implement as we do not have to simulate the flow field multiple times to get a large set of solution vectors. That being said, there are severe limitations associated with eigendecomposition as opposed to PCA, which will be explained at the end of this appendix. We will briefly provide the implementation and possible usage of this method in detecting missing flow features, and also point out the possible conditioning problem in the following.

A.1 Methodology

Let $\mathbf{A}$ be the n by n square Jacobian operator of a CFD simulation. By performing an eigenanalysis on this matrix, we can extract the left and the right eigenvectors alongside their corresponding eigenvalues:

$$[\mathbf{A}]\mathbf{X} = \mathbf{\Lambda}\mathbf{X} \tag{A.1a}$$

$$\mathbf{Y}^T[\mathbf{A}] = \mathbf{\Lambda}\mathbf{Y}^T \tag{A.1b}$$

where $\mathbf{X}$ and $\mathbf{Y}$ contain the right and the left eigenvectors as their columns, respectively, and $\mathbf{\Lambda}$ is a diagonal matrix containing the eigenvalues. The Jacobian matrix is generally not symmetric, and therefore the left and the right eigenvectors span different spaces. The right eigenvectors are called the eigenmodes, and similar to PCA modes, they span a basis on which we can represent any field vector. Here, we consider the truncation and the

discretization error vectors respectively as τ and ε as a combination of the eigenmodes:

$$\tau = \sum_{i=1}^{n} a_i \mathbf{x}_i \tag{A.2a}$$

$$\varepsilon = \sum_{i=1}^{n} b_i \mathbf{x}_i \tag{A.2b}$$

where $\mathbf{x}_i$ are the right eigenvectors and a_i and b_i are the truncation and the discretization error eigencoefficients, respectively. The two types of errors are linked through the Jacobian matrix as follows (from Error Transport Equation presented in Eq. (1.28)):

$$[\mathbf{A}]\ \varepsilon = \tau \tag{A.3}$$

By substituting Eqs. (A.2a) and (A.2b) in Eq. (A.3), we get:

$$[\mathbf{A}] \sum_{i=1}^{n} b_i \mathbf{x}_i = \sum_{i=1}^{n} a_i \mathbf{x}_i \implies \sum_{i=1}^{n} b_i [\mathbf{A}] \mathbf{x}_i = \sum_{i=1}^{n} a_i \mathbf{x}_i \tag{A.4}$$

and using the definition of eigenvalues λ_i:

$$\sum_{i=1}^{n} b_i \lambda_i \mathbf{x}_i = \sum_{i=1}^{n} a_i \mathbf{x}_i \tag{A.5}$$

The right eigenvectors are linearly independent, and therefore, Eq. (A.5) indicates:

$$a_i = \lambda_i b_i \tag{A.6}$$

which relates the eigendecomposition of the truncation and the discretization errors through the eigenvalue in each mode. Unlike the PCA modes, the eigenmodes are not orthonormal for general non-symmetric matrices; hence, the same approach we presented to compute the expansion coefficient in Eq. (2.5) does not apply here. Instead, we use the orthonormality property between the left and the right set of eigenvectors (i.e., $\langle \mathbf{x}_i, \mathbf{y}_j \rangle = \delta_{ij}$) to

compute a_i as:

$$a_i = \langle \mathbf{y}_i, \tau \rangle = \mathbf{y}_i^T \tau \tag{A.7}$$

where $\mathbf{y}_i$ are the left eigenvectors. Having a_i and λ_i, we can then use Eq. (A.6) to calculate b_i.

We can combine eigendecomposition results with the Adjoint error estimator, where we represent the error in a single functional J as the inner product of the truncation error τ and the adjoint vector ψ:

$$\delta J = \langle \psi, \tau \rangle \tag{A.8}$$

We discussed how to get the adjoint vector in Section 1.5. If we substitute Eq. (A.2a) in Eq. (A.8), we get:

$$\delta J = \sum_{i=1}^{n} a_i \langle \psi, \mathbf{x}_i \rangle = a_1 \langle \psi, \mathbf{x}_1 \rangle + a_2 \langle \psi, \mathbf{x}_2 \rangle + \cdots + a_n \langle \psi, \mathbf{x}_n \rangle \tag{A.9}$$

The above expression writes the total error in a solution functional as a combination of errors resulting from each eigenmode. This way, we can mark the mode that shows the largest contribution to the error and tag it as the source of inaccuracy in the simulation. In our case, the solution functional we are interested in is the drag coefficient of the airfoil, since the missing separation bubble mostly affects this parameter. Consequently, performing eigendecomposition in conjunction with the adjoint error estimation can yield a single mode responsible for the missing separation bubble.

A.2 Conditioning Problem

A crucial assumption in eigendecomposition is that the left and the right set of eigenvectors are orthogonal. This is in general true for any square matrix, regardless of it being symmetric or not. We took advantage of this property when computing the eigencoefficients of the truncation error in Eq. (A.7).

A possible challenge arises when the inner product of the left and the right eigenvectors corresponding to a particular eigenvalue, which is known

as the condition number of the eigenvalue λ_i and denoted by $s(\lambda_i)$, is nearly zero:

$$s(\lambda_i) = \langle \mathbf{x}_i, \mathbf{y}_i \rangle \approx 0 \tag{A.10}$$

In other words, the corresponding left and right eigenvectors are very close to being orthogonal to each other. To make the sets of left and right eigenvectors orthonormal, we need to augment this insignificant inner product to be equal to 1. This is done by applying a constant scaling factor equal to $1/s(\lambda_i)$ (which becomes very large when $s(\lambda_i) \approx 0$) to one of the eigenvectors. In an ideal situation, where the inner product of orthogonal vectors are exactly zero, the scaling factor will not affect the off-diagonal entities in the diagonal matrix of $\mathbf{Y}^T\mathbf{X}$. However, since the computer zero is slightly deviated from the exact zero, a gigantic scaling factor inevitably makes some off-diagonal entities attain significant values and the orthogonality condition breaks. This makes our computations in Eq. (A.7) invalid, and we can no longer calculate the eigencoefficients.

In these cases, square matrix $\mathbf{A}$ is ill-conditioned with regard to its eigensystem, and the eigenvalues that show a very low condition number are called the ill-conditioned eigenvalues (note that the condition number of a single eigenvalue is different from the overall condition number assigned to a matrix). The Jacobian matrix in our CFD simulation exhibited this problematic behavior, and therefore, performing an eigendecomposition became impractical for our analysis.

We provide the following points to compare PCA and eigendecomposition for the purpose of this work. The single advantage of eigendecomposition over PCA is that it can be performed on a single simulation by taking only the Jacobian operator. Therefore, in initial implementation stage, the modes from eigendecomposition are achieved much easier and faster compared to the PCA modes. However, there are many reasons to consider PCA as the decomposition method for flow feature detection:

- PCA shows more robustness compared to eigendecomposition, for which the matrix could be ill-conditioned.

- For machine learning purposes, we always need lots of data. Therefore, the mentioned advantage of eigendecomposition over PCA becomes neutralized, as we still need to gather multiple CFD simulations to form a learning system.

- The hierarchical structure of PCA makes it the ideal choice for machine learning as it substantially reduces the dimensionality of the data.

- PCA captures the most important physical and numerical aspects of the simulation, whereas eigendecomposition only focuses on the numerical characteristics.

- Applying PCA on a CFD data set is generally more efficient than performing a full eigenanalysis on the Jacobian operator. If we take the dimensionality of the CFD simulations as n, then the Jacobian matrix is a square n by n matrix; whereas the data set is a rectangular n by m matrix where m is the number of simulations and often we have $m \ll n$. Also, we normally implement the *economy* SVD when performing PCA, which further reduces involved computations. Note that we assume here that we want to use machine learning for which we need to create a full data set, regardless of the decomposition technique; otherwise, if an alternative feature detector can be proposed using eigenanalysis without the need for a full data set, then eigendecomposition becomes much more efficient compared to PCA, for which the existence of a full data set is inevitable.

Appendix B

Principal Modes (Base Mesh)

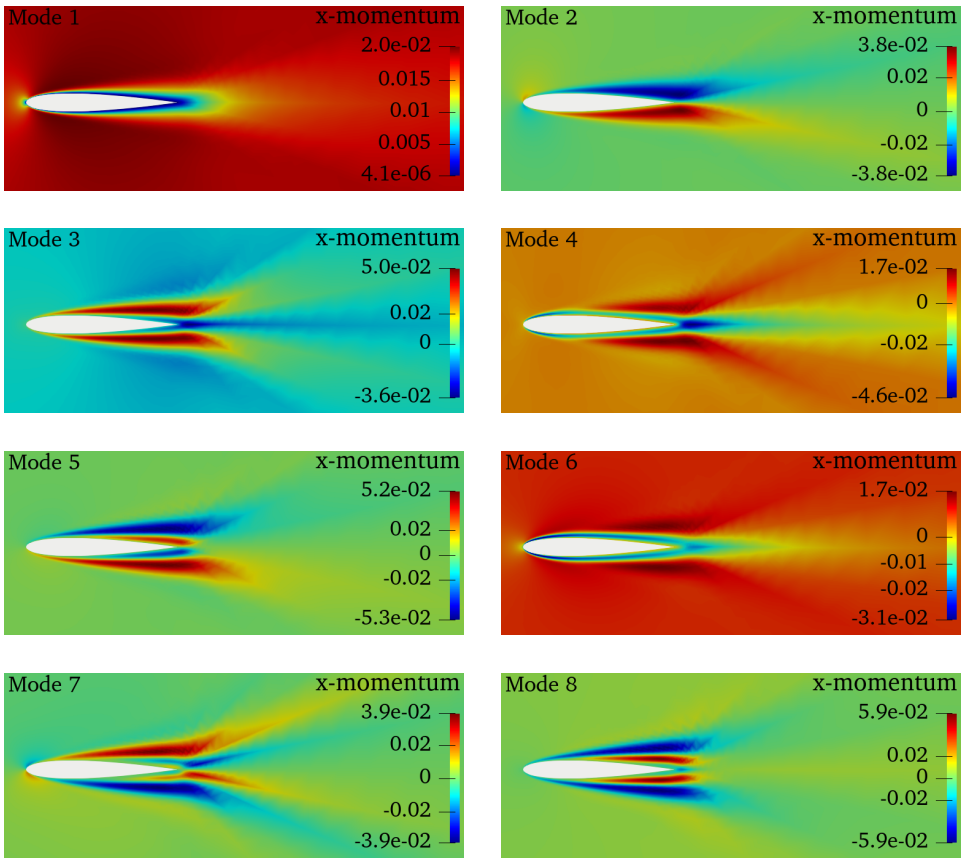


Figure B.1: x-momentum component of solution modes, extracted by performing PCA on the CFD data matrix shown in Fig. 1.11.

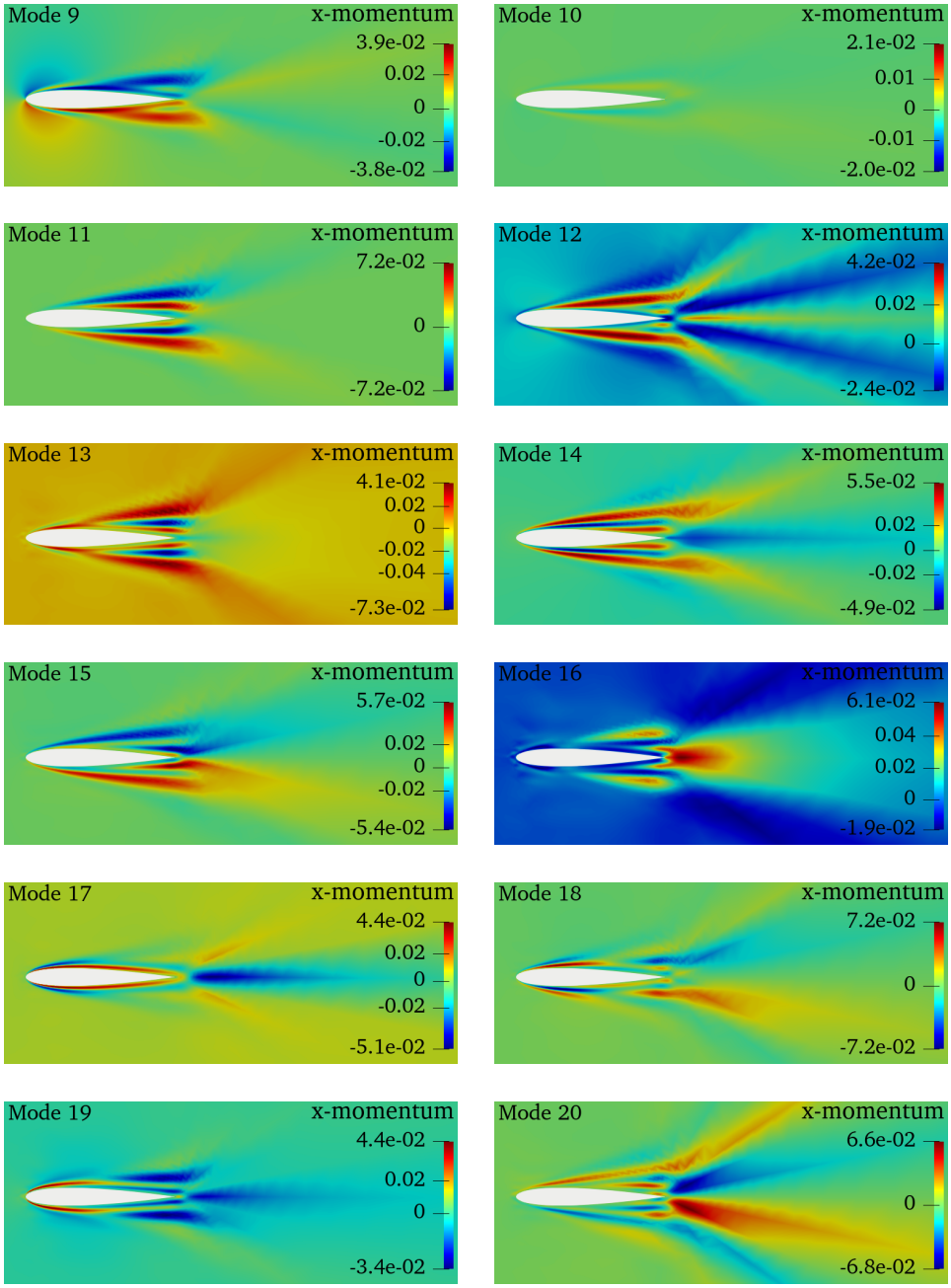


Figure B.1: x-momentum component of solution modes, extracted by performing PCA on the CFD data matrix shown in Fig. 1.11. (cont.)

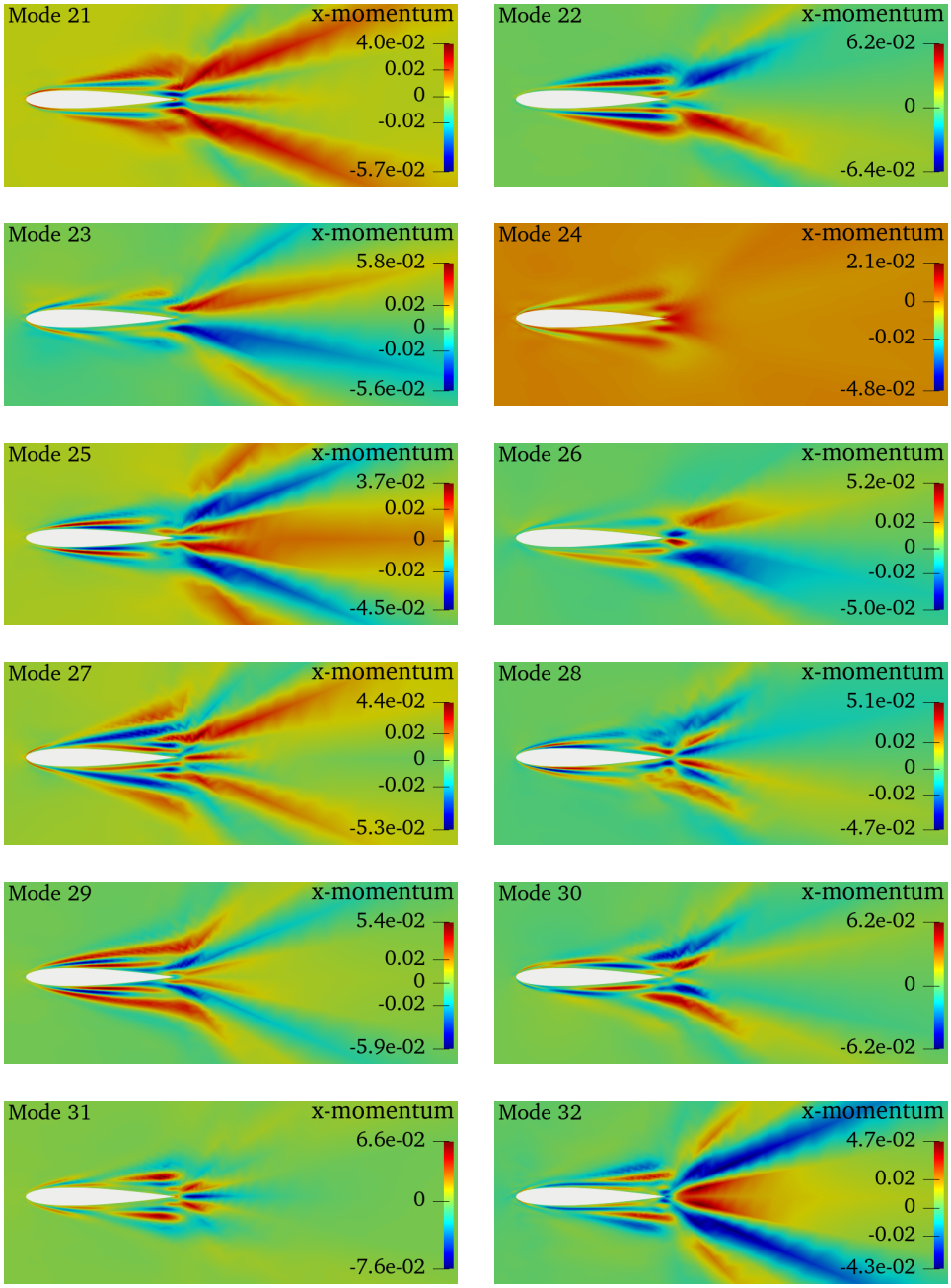


Figure B.1: x-momentum component of solution modes, extracted by performing PCA on the CFD data matrix shown in Fig. 1.11. (cont.)

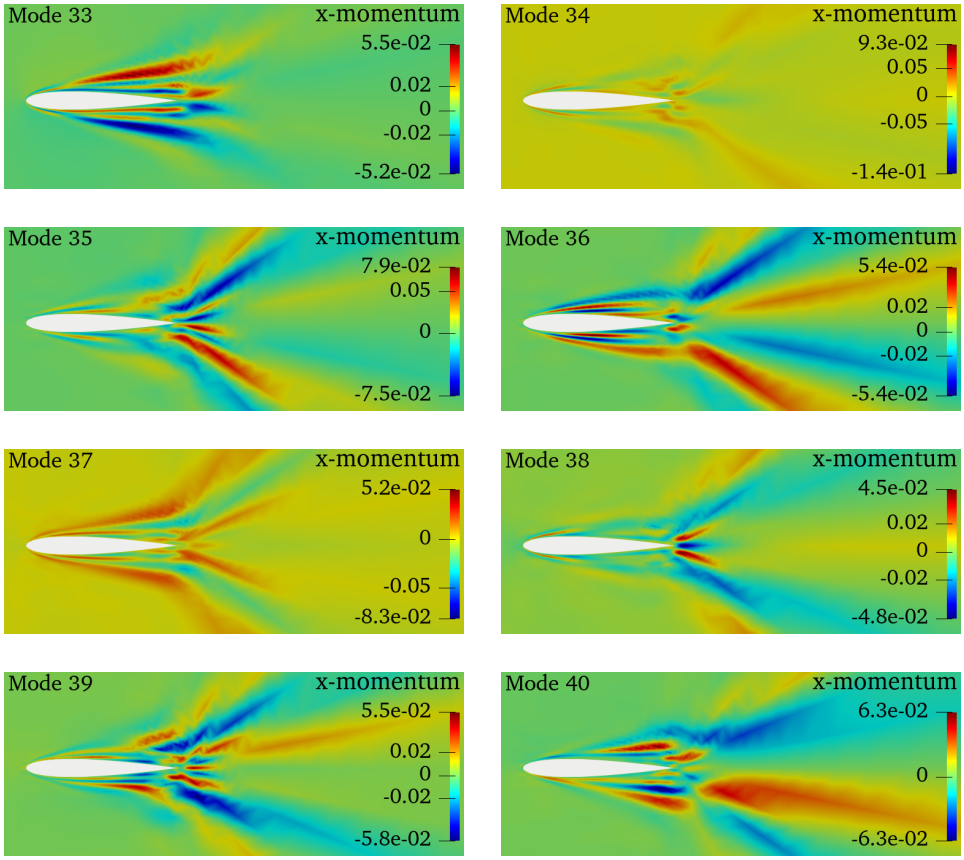


Figure B.1: x-momentum component of solution modes, extracted by performing PCA on the CFD data matrix shown in Fig. 1.11. (cont.)

Appendix C

Principal Coefficient (Base Mesh)

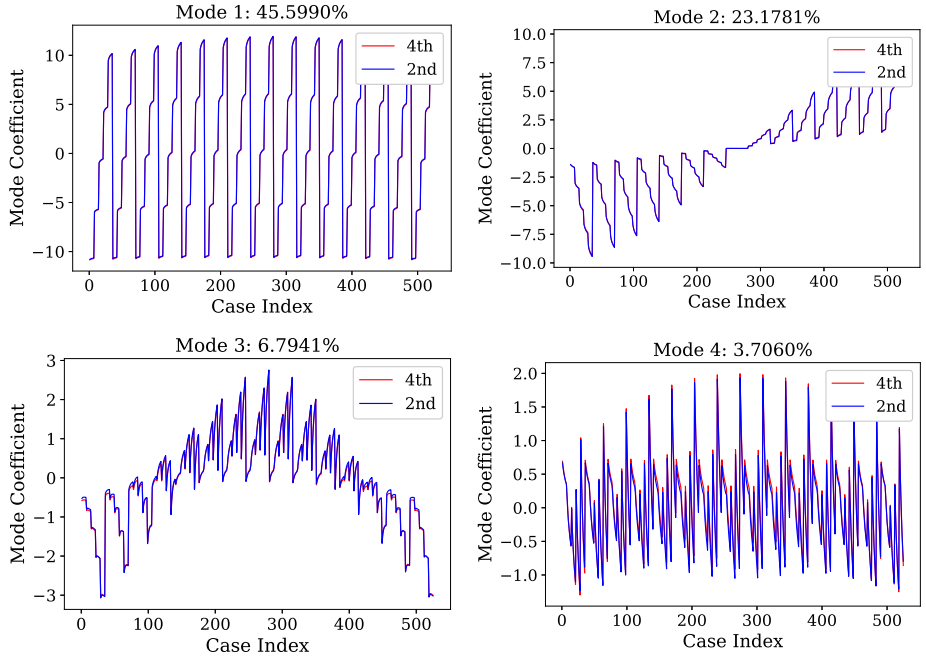


Figure C.1: PCA expansion coefficients of the 2^{nd}- and the 4^{th}-order solution vectors, calculated by taking the inner product of the solution vectors with PCA modes (Eq. (2.5)). The amount of energy captured by each mode is presented on top of its associated plot, and is calculated by $\sigma_i / \sum_{j=1}^{m} \sigma_j$, where σ represents the singular value.

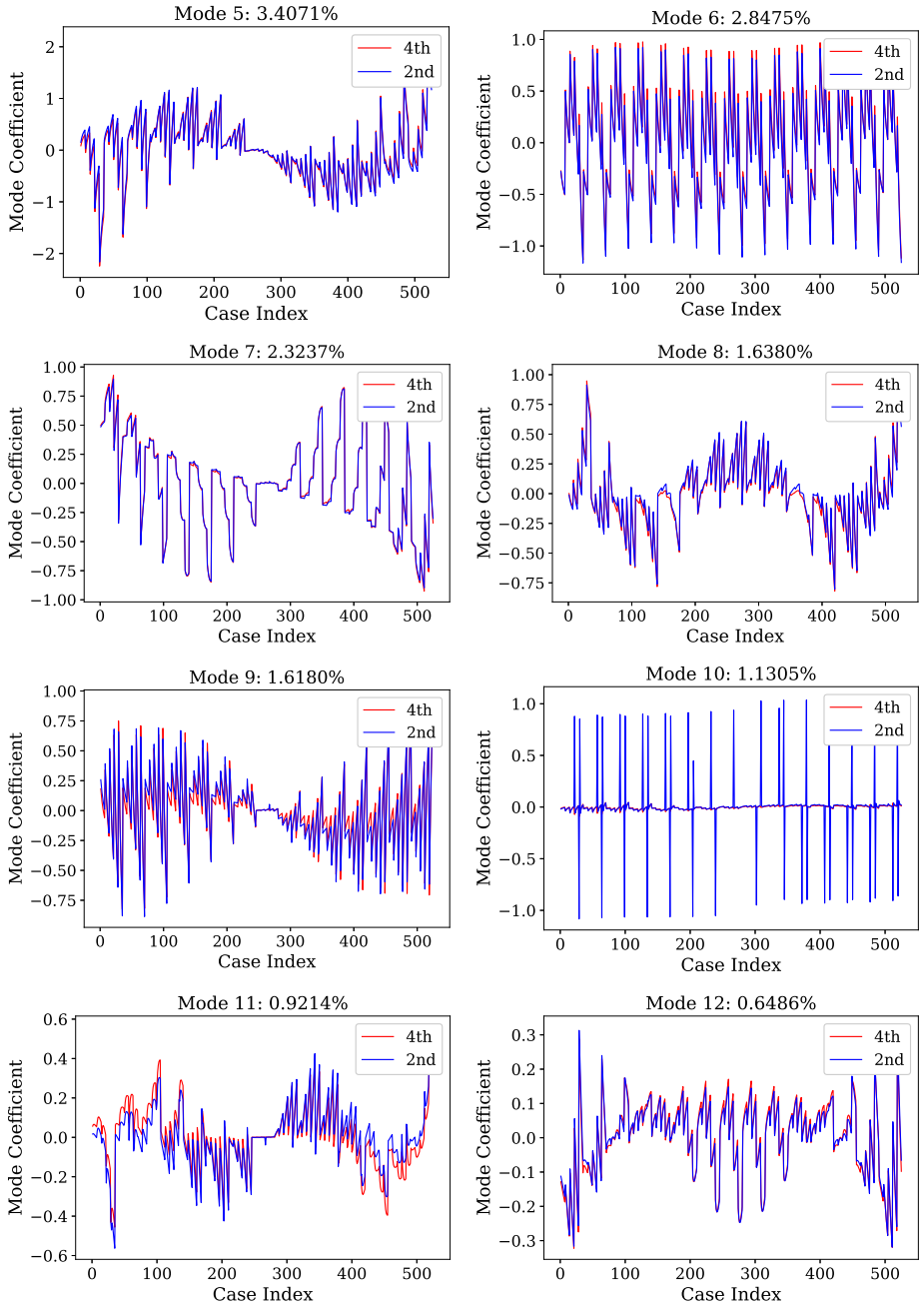


Figure C.1: PCA expansion coefficients of the 2^{nd}- and the 4^{th}-order solution vectors, calculated by taking the inner product of the solution vectors with PCA modes (Eq. (2.5)). (cont.)

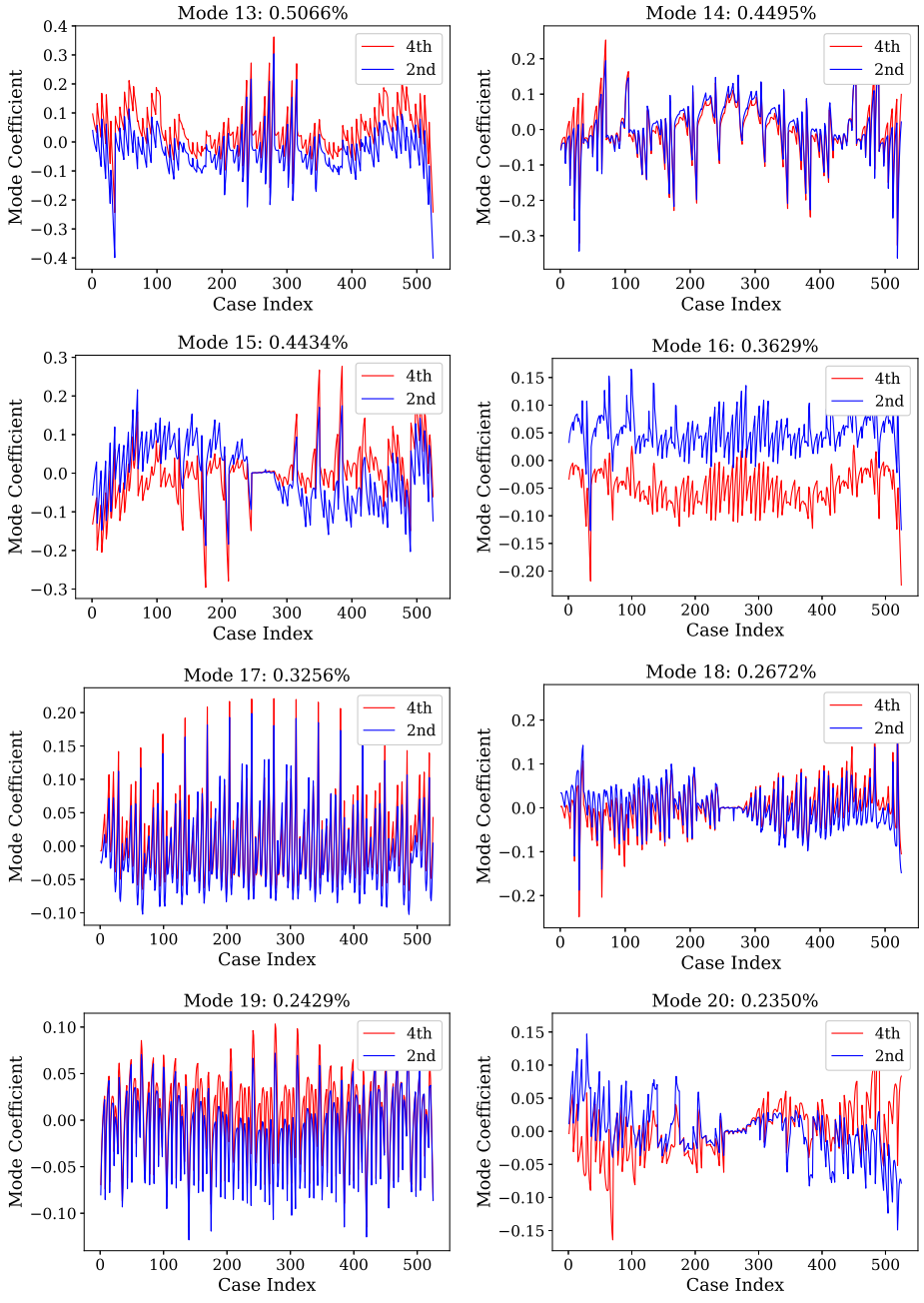


Figure C.1: PCA expansion coefficients of the 2^{nd}- and the 4^{th}-order solution vectors, calculated by taking the inner product of the solution vectors with PCA modes (Eq. (2.5)). (cont.)

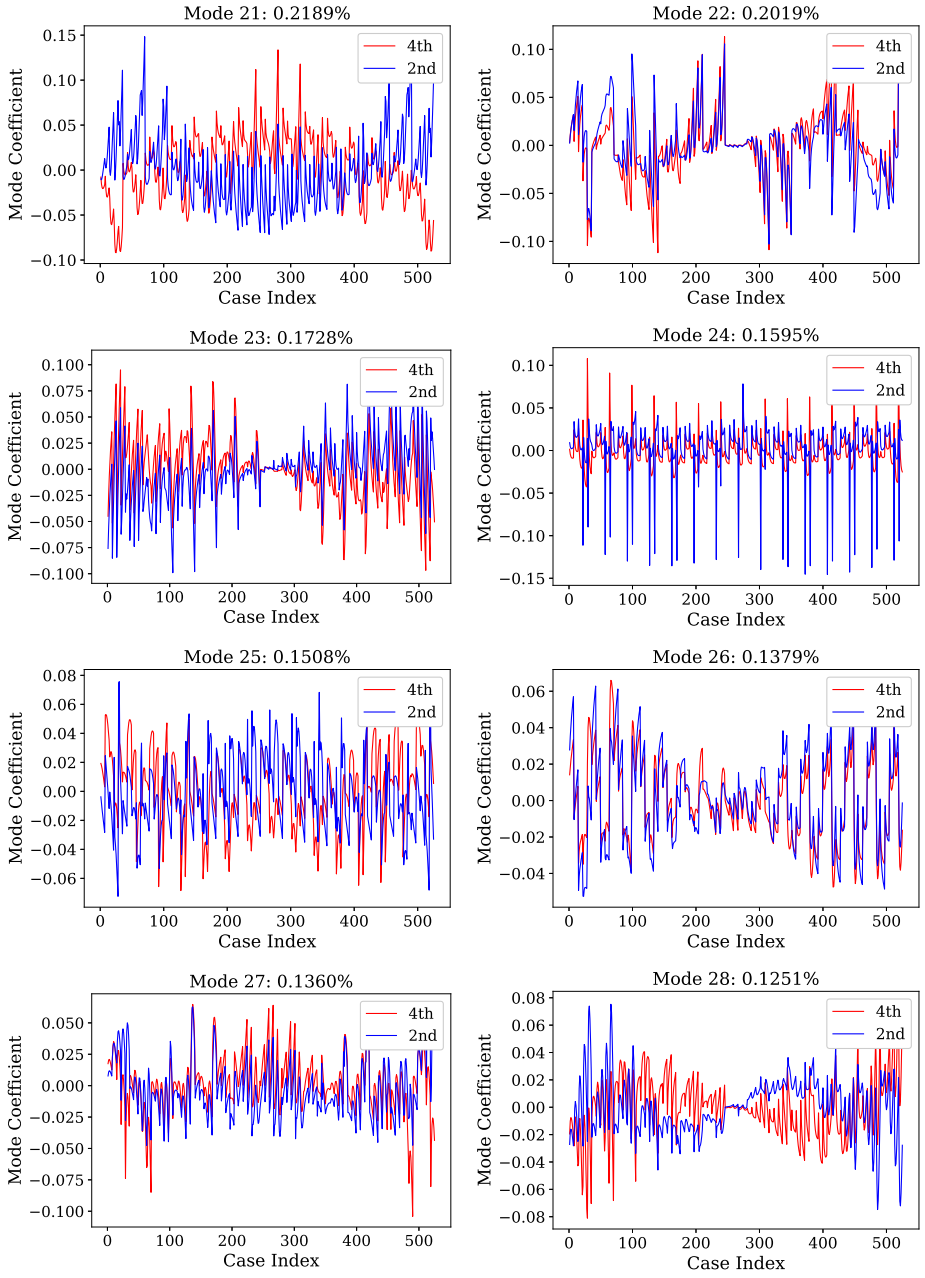


Figure C.1: PCA expansion coefficients of the 2^{nd}- and the 4^{th}-order solution vectors, calculated by taking the inner product of the solution vectors with PCA modes (Eq. (2.5)). (cont.)

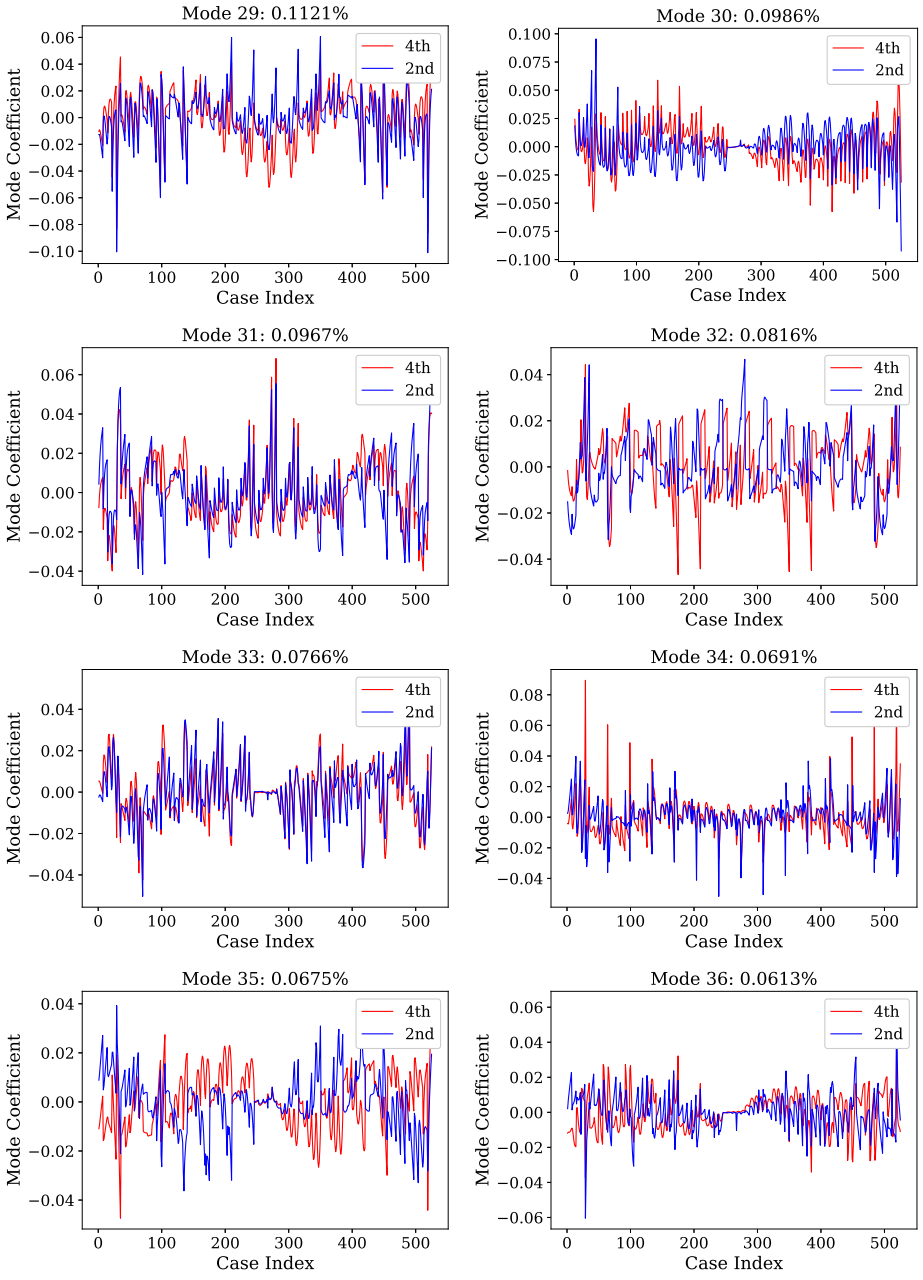


Figure C.1: PCA expansion coefficients of the 2^{nd}- and the 4^{th}-order solution vectors, calculated by taking the inner product of the solution vectors with PCA modes (Eq. (2.5)). (cont.)

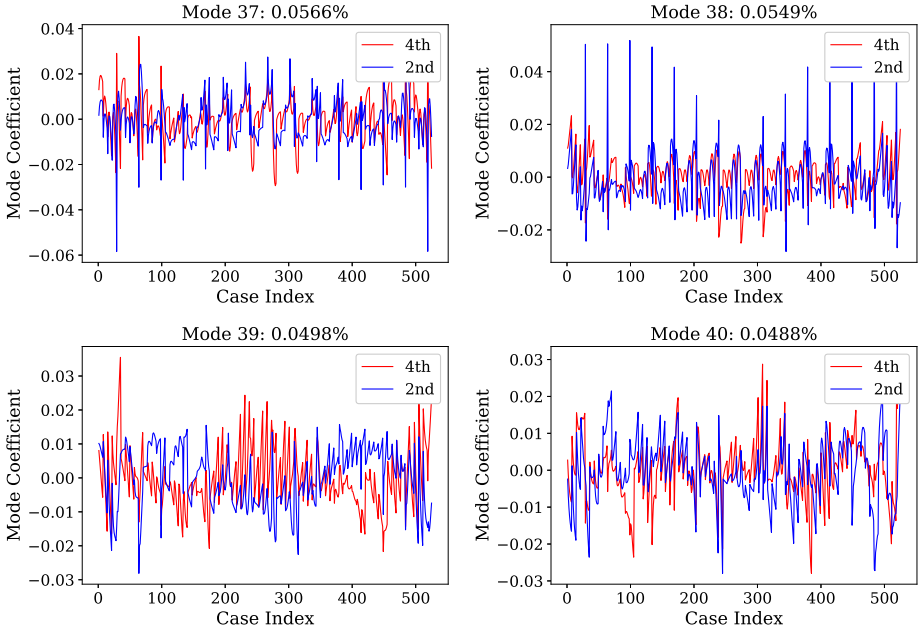


Figure C.1: PCA expansion coefficients of the 2^{nd}- and the 4^{th}-order solution vectors, calculated by taking the inner product of the solution vectors with PCA modes (Eq. (2.5)). (cont.)

Appendix D

Principal Modes (Different Mesh)

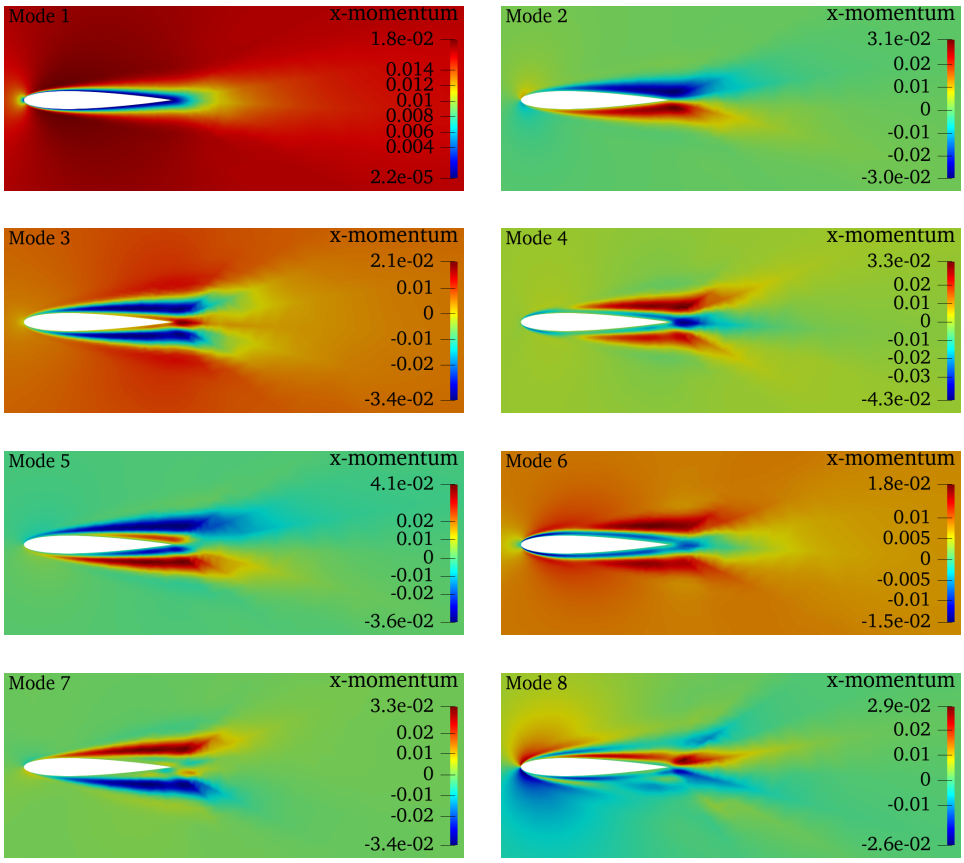


Figure D.1: x-momentum component of PCA modes, extracted from the simulations performed on the mesh shown in Fig. 3.18.

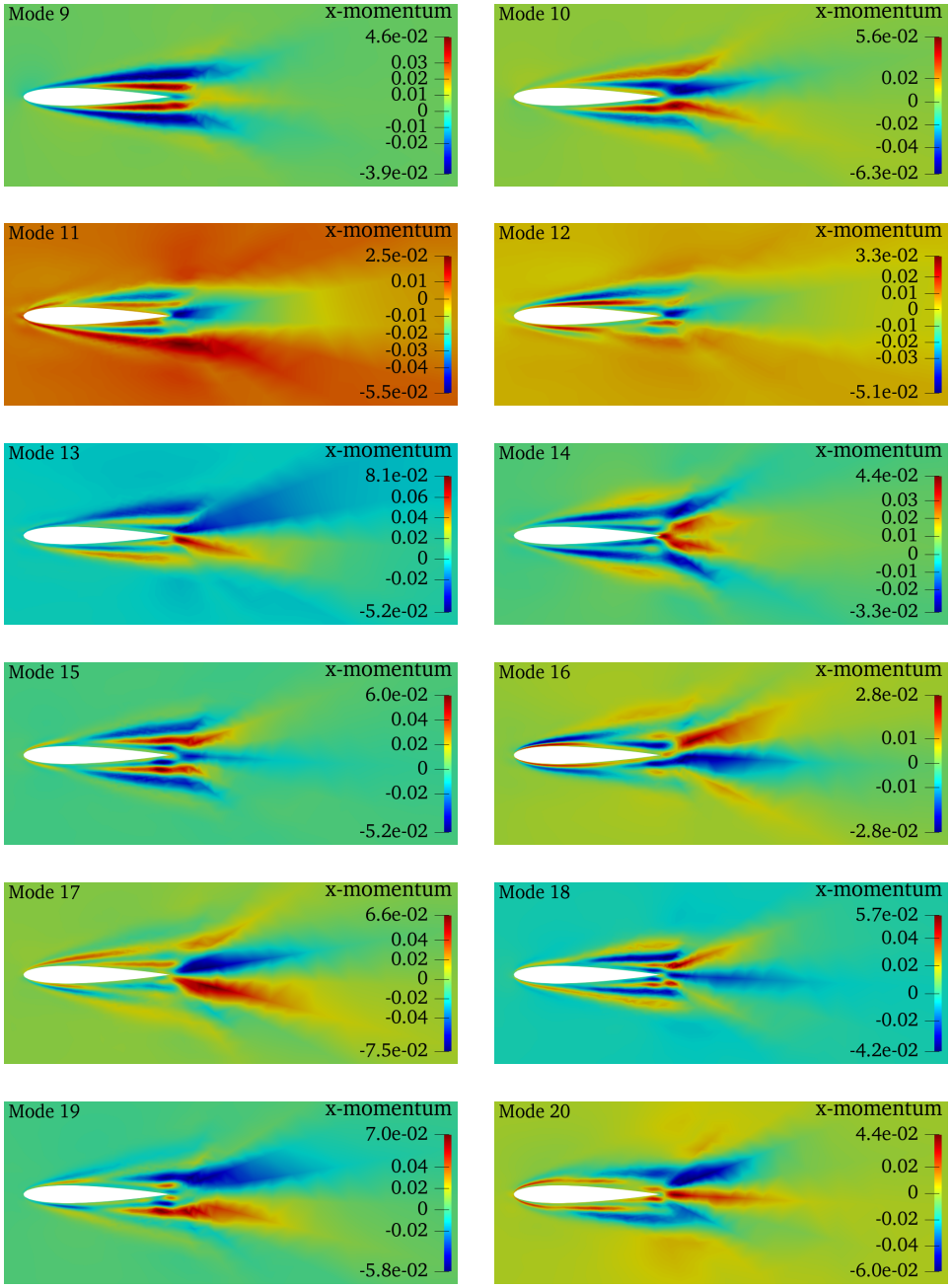


Figure D.1: x-momentum component of PCA modes, extracted from the simulations performed on the mesh shown in Fig. 3.18. (cont.)

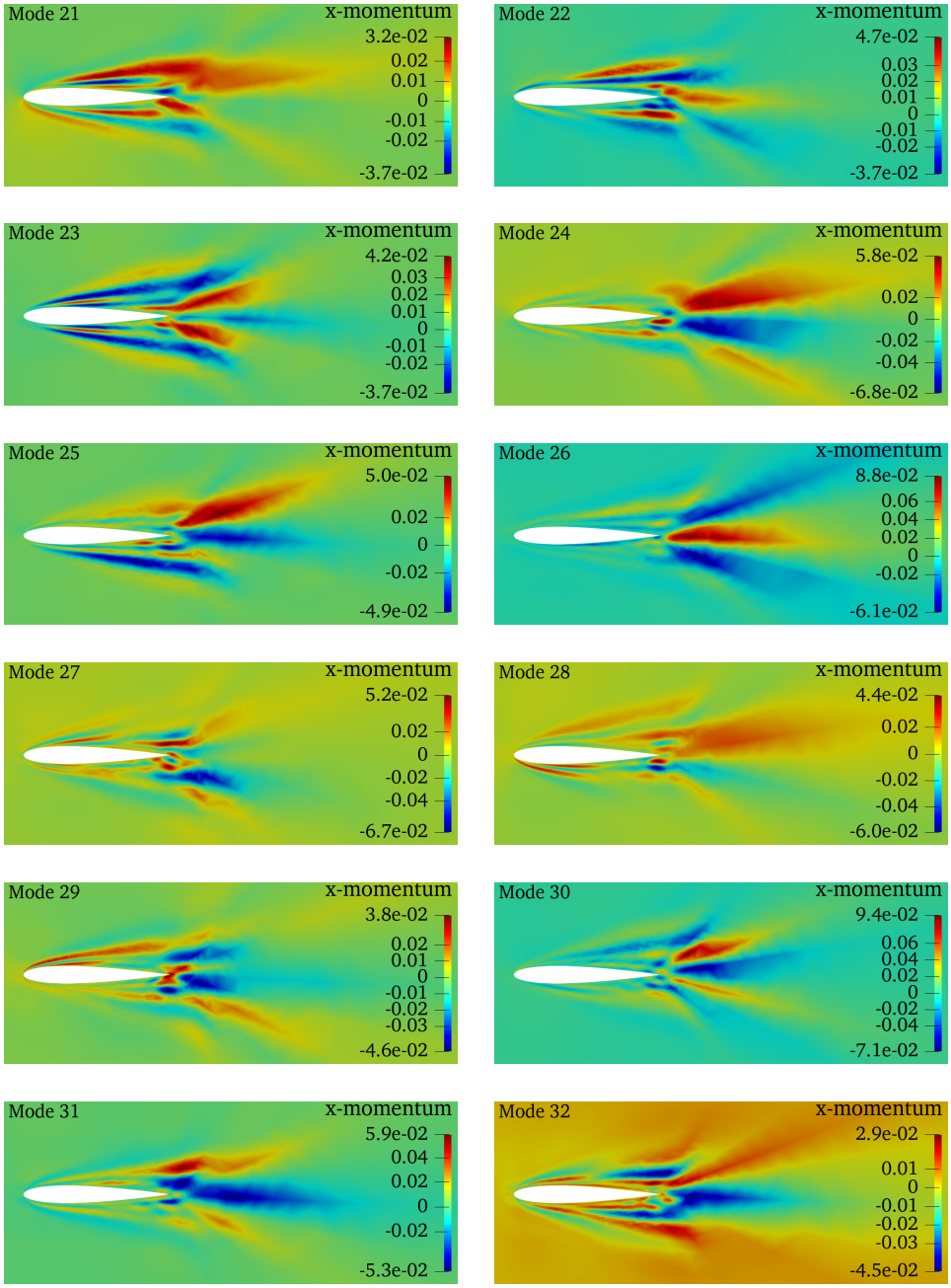


Figure D.1: x-momentum component of PCA modes, extracted from the simulations performed on the mesh shown in Fig. 3.18. (cont.)

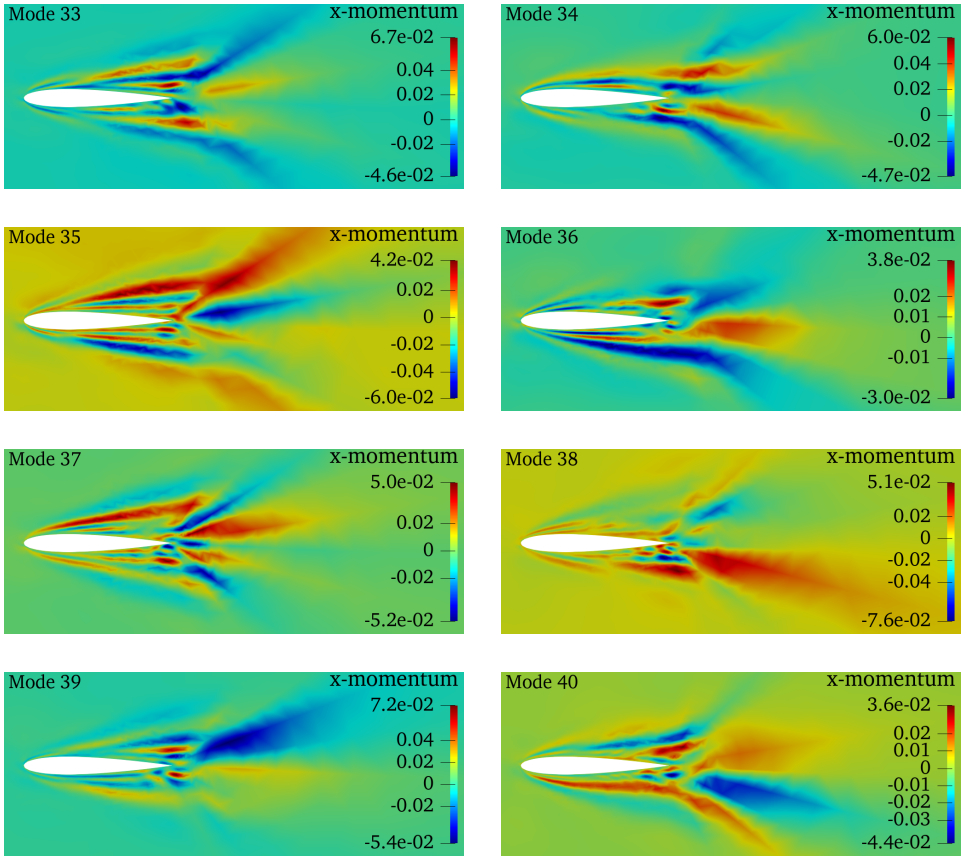


Figure D.1: x-momentum component of PCA modes, extracted from the simulations performed on the mesh shown in Fig. 3.18. (cont.)

Appendix E

Principal Coefficients (Different Mesh)

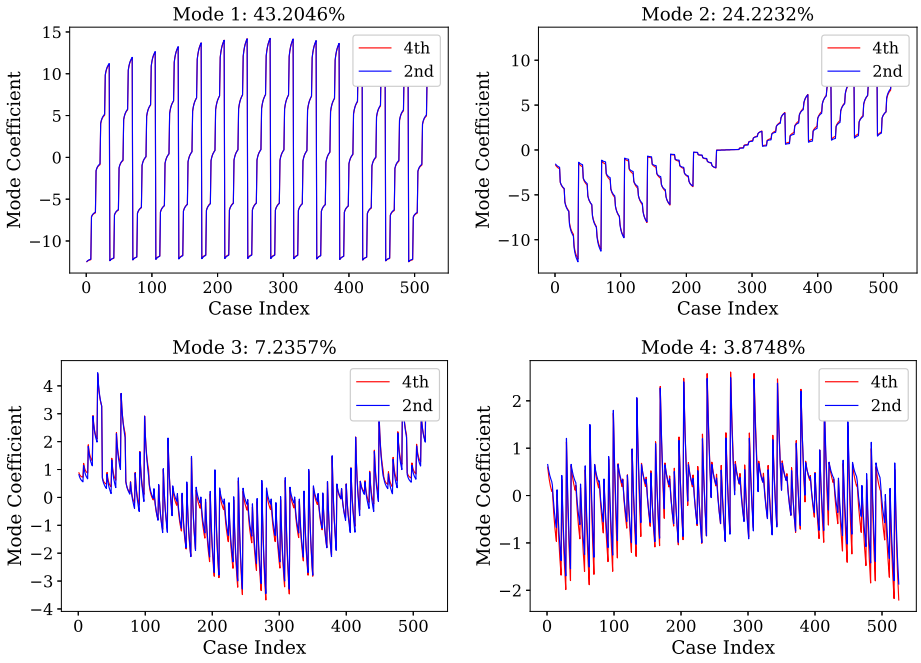


Figure E.1: PCA expansion coefficients of the 2^{nd}- and the 4^{th}-order solution vectors, corresponding to the mesh shown in Fig. 3.18. The amount of energy captured by each mode is presented on top of its associated plot, and is calculated by $\sigma_i / \sum_{j=1}^{m} \sigma_j$, where σ represents the singular value.

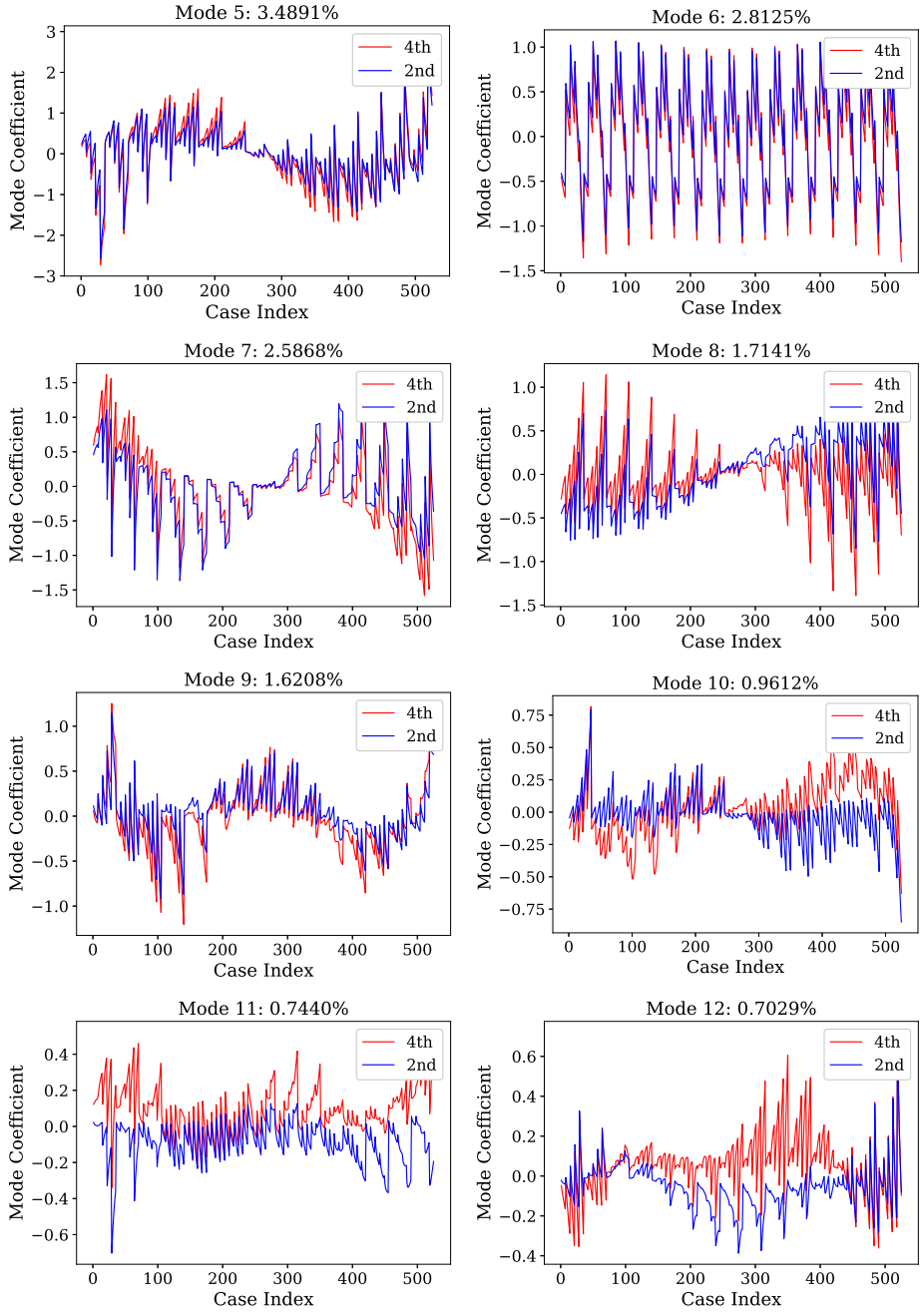


Figure E.1: PCA expansion coefficients of the 2^{nd}- and the 4^{th}-order solution vectors, corresponding to the mesh shown in Fig. 3.18. (cont.)

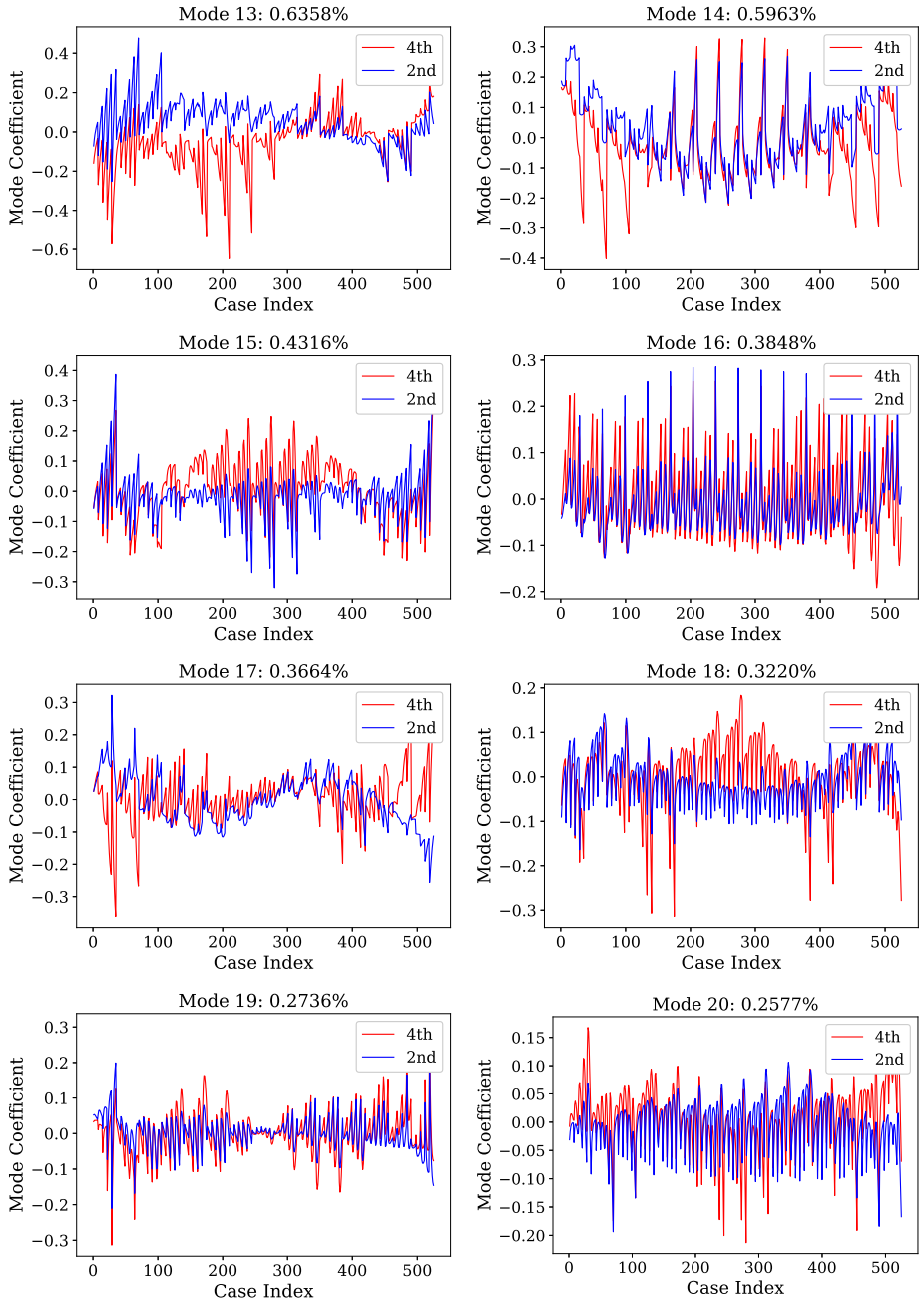


Figure E.1: PCA expansion coefficients of the 2^{nd}- and the 4^{th}-order solution vectors, corresponding to the mesh shown in Fig. 3.18. (cont.)

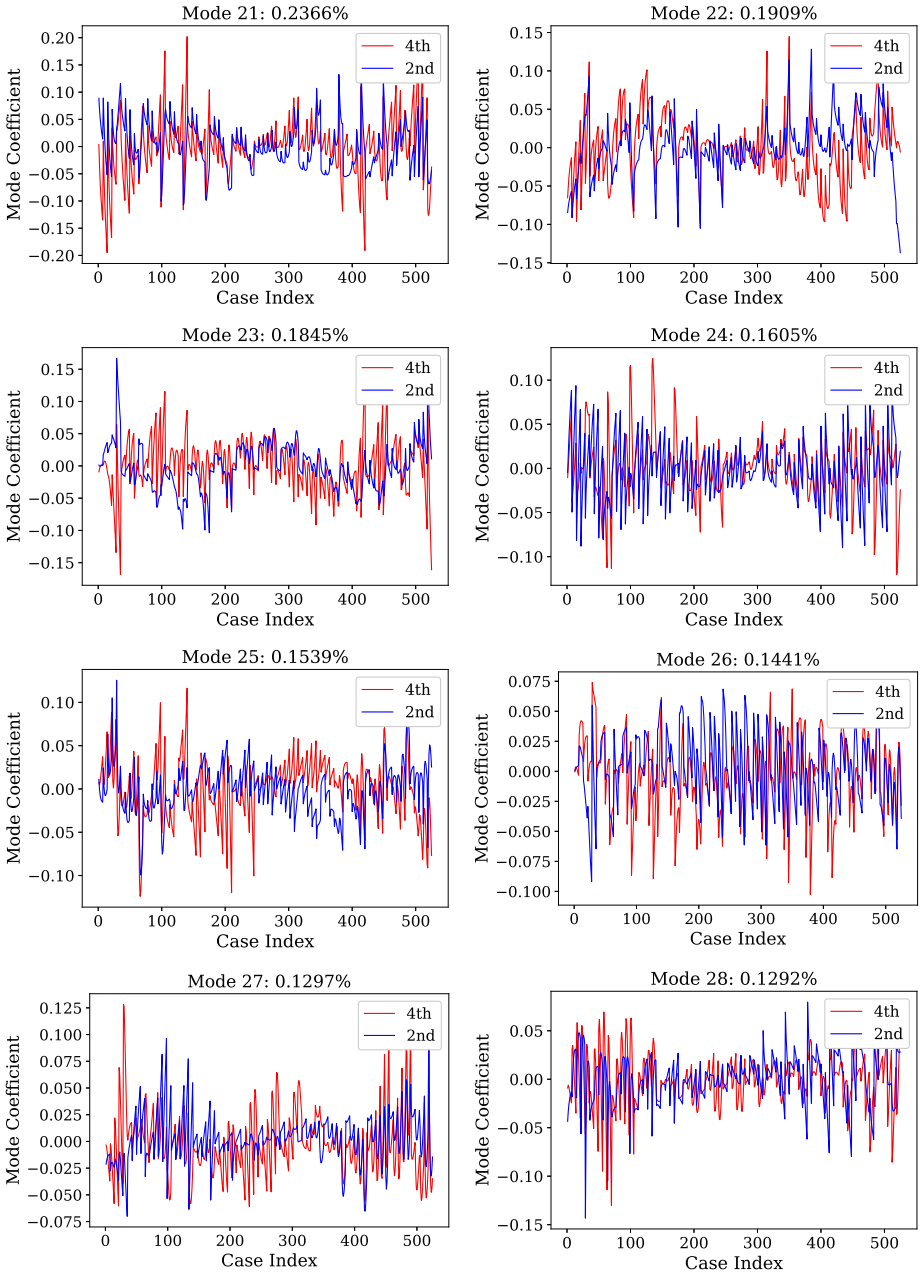


Figure E.1: PCA expansion coefficients of the 2^{nd}- and the 4^{th}-order solution vectors, corresponding to the mesh shown in Fig. 3.18. (cont.)

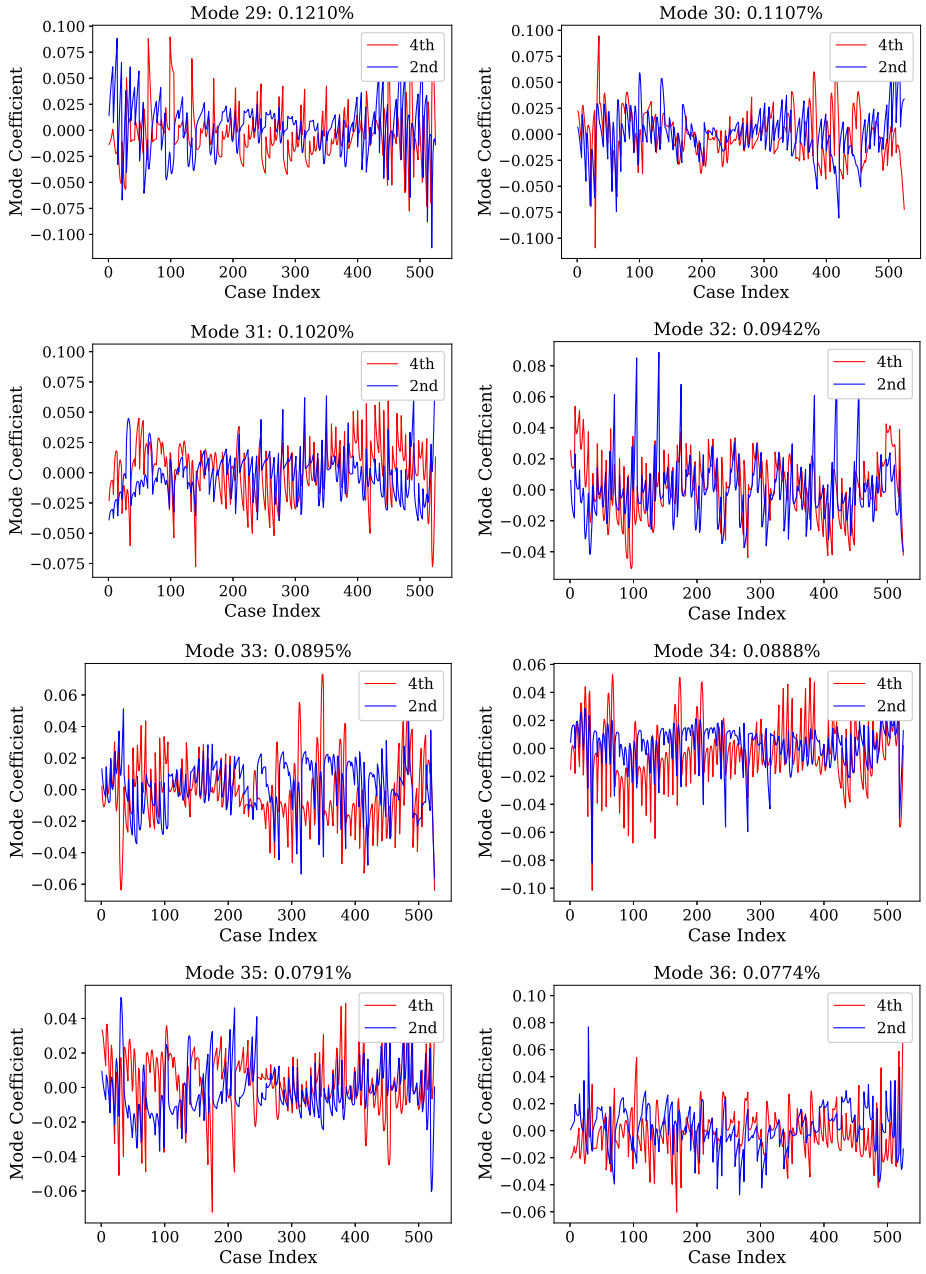


Figure E.1: PCA expansion coefficients of the 2^{nd}- and the 4^{th}-order solution vectors, corresponding to the mesh shown in Fig. 3.18. (cont.)

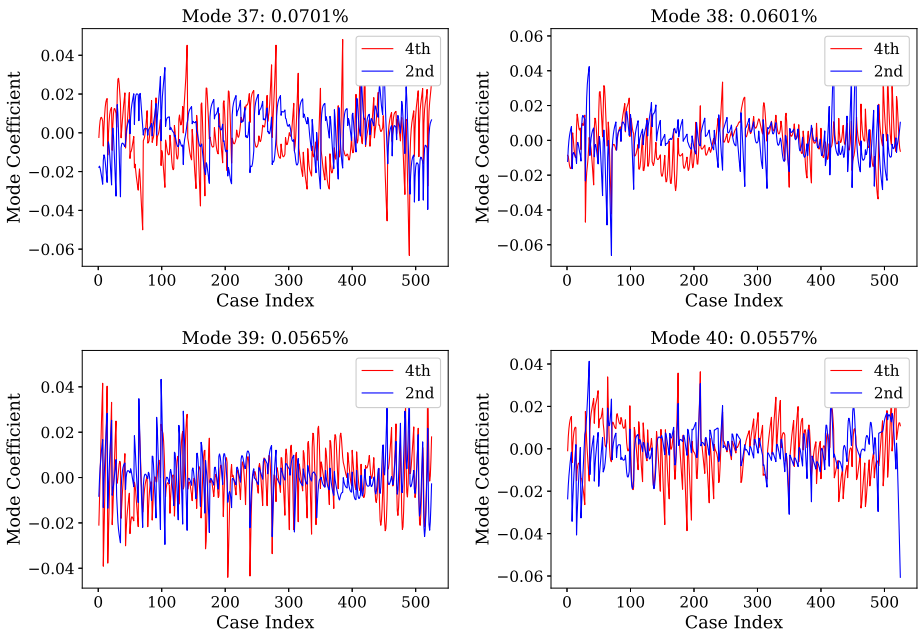


Figure E.1: PCA expansion coefficients of the 2^{nd}- and the 4^{th}-order solution vectors, corresponding to the mesh shown in Fig. 3.18. (cont.)

Appendix F

Code Snippets and Shell Commands

This Appendix provides important terminal commands and python functions used to generate the results presented in this thesis. All programs have been executed on a linux workstation running on Ubuntu 20.04.4 LTS.

F.1 GRUMMP Shell Commands

To generate the computational domains in this study, we used GRUMMP, which is a linux package written in C++ and runs based on the commands given in the terminal window. The following commands were used to generate the base and the arbitrary meshes:

```
$ {PATH_TO_GRUMMP}/apps/edam/2D/edam2d -i {BDRY_FILE} -R 8 -G 8
    -x 0.001 -g 1.2 -o 0012 -l {LEN_FILE} -s -n 55
```

Program F.1: GRUMMP terminal command to generate the base mesh shown in Fig. 1.10.

```
$ {PATH_TO_GRUMMP}/apps/edam/2D/edam2d -i {BDRY_FILE} -R 16 -G
   48 -x 0.002 -g 1.2 -o 0012 -l {LEN_FILE} -t
```

Program F.2: GRUMMP terminal command to generate the arbitrary mesh shown in Fig. 3.18.

Although these meshes are basically unstructured, they share similarities to structured configurations. To generate a fully-unstructured mesh, the following command may be used:

```
$ {PATH_TO_GRUMMP}/src/programs/tri -i {BDRY_FILE} -r 4 -g 4 -o
    0012 -l {LEN_FILE}
```

Program F.3: GRUMMP terminal command to generate a fully-unstructured mesh.

F.2 ANSLib Shell Commands

To solve the flow field on the domains created using GRUMMP, ANSLib package for linux that is also written in C++ was utilized. This package takes input commands from the terminal, similar to GRUMMP. To generate proper data for our machine learning purposes, we developed an application specific to this task. The code can be run by entering the following command:

```
$ {PATH_TO_ANSLIB}/apps/feature-identification/feature-
   identification -f {MESH_FILE} -physics RoeVisc2D -mach {M}
   -angle {AOA} -reynolds {RE} -Adjoint -drag -pqr 244 -C 0.1
   -mesh_type v
```

Program F.4: ANSLib terminal command to solve the flow field using 2^{nd}- and 4^{th}-order discretization schemes.

F.3 Python Code Snippets

To create the machine learning structures depicted in Figs. 3.4 and 4.1, we used python v3.8.10, which is the most popular programming language for machine learning nowadays. The learning pipeline can be divided into three steps: modal decomposition, classification, and regression. Important code snippets and modules used in each step are provided in the following.

Principal Component Analysis

We used Linear Algebra module from NumPy library to decompose the data matrix shown in Fig. 2.1 (after being processed) into PCA modes. The code is provided as follows:

```
# Import NumPy
import numpy as np
# Compute economic SVD of the data matrix
U, S, V = np.linalg.svd(data_matrix, full_matrices=False)
# Outputs:
# U -> left singular vectors (PCA modes)
# S -> singular values
# V -> right sinular vectors
```

Program F.5: Performing Principal Component Analysis on the data matrix.

Classification - Missing Flow Feature Identification

To perform machine learning classification, we utilized Scikit-learn and Tensorflow libraries. Scikit-learn encompasses most conventional machine learning algorithms, such as Logistic Regression and Random Forest, while Tensorflow provides a framework suitable for creating deep neural networks. The following python snippets present different steps taken to train a classifier on the imbalanced data set presented in Table 3.2:

```
# Import Pandas and Scikit-learn modules
import pandas as pd
from sklearn.model_selection import train_test_split
# Separate "MISSED" and "CAPTURED" cases
dataset_capt = dataset[ dataset["Label"] == "CAPTURED" ]
dataset_miss = dataset[ dataset["Label"] == "MISSED" ]
# Randomely create the training and the test sets for each "
    MISSED" and "CAPTURED" labels
train_capt, test_capt = train_test_split(dataset_capt,
    test_size=0.2, random_state=20)
train_miss, test_miss = train_test_split(dataset_miss,
    test_size=0.2, random_state=20)
# Gather "MISSED" and "CAPTURED" cases back together
train_data = pd.concat([train_capt, train_miss], axis = 0)
test_data = pd.concat([test_capt, test_miss], axis = 0)
# shuffle the data if needed
NEED_SHUFFLE = True
if NEED_SHUFFLE:
```

```
    train_data = train_data.sample(frac=1, random_state=20).
    reset_index(drop=True)
    test_data = test_data.sample(frac=1, random_state=20).
    reset_index(drop=True)
```

Program F.6: Splitting the full data set into the training and the test subsets for classification.

```
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
# over-sample minority so that: minor/major = sampling_strategy
over_sampler = RandomOverSampler(sampling_strategy = 0.3,
    random_state = 20)
X_train, y_train = over_sampler.fit_resample(X_train, y_train)
# under-sample majority so that: minor/major =
    sampling_strategy
under_sampler = RandomUnderSampler(sampling_strategy= 1.0,
    random_state = 20)
X_train, y_train = under_sampler.fit_resample(X_train, y_train)
```

Program F.7: Balancing the imbalanced data set.

```
# Import classification algorithms
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
# Select the classification algorithm
model = RandomForestClassifier(random_state=20, n_jobs = -1)
# model = LogisticRegression(random_state=20, solver = 'lbfgs')
# Training
model.fit(X_train, y_train)
# Test (prediction)
y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)
```

Program F.8: Training and testing the classifier. The same code structure is used for both Logistic Regression and Random Forest.

```
# Import tuning module
from sklearn.model_selection import GridSearchCV
# Select the parameters and the range on which the model will
    be tuned
```

```
param_grid = [
        {'n_estimators': [60, 70, 80],
         'max_features': [10],
         'bootstrap': [True, False],
         'random_state': [20],
         'criterion': ["gini", "entropy"],
         'max_depth': [2, 3, 4],
         'min_samples_split': [2],
         'min_samples_leaf': [1]
        }
    ]
# Perform tuning
grid_search = GridSearchCV(model, param_grid, cv=5,
                           scoring='accuracy',
                           return_train_score=True,
                           n_jobs = -1)
grid_search.fit(X_train, y_train)
# Extract the tuned model
model_tuned = grid_search.best_estimator_
```

Program F.9: Parameter tuning for the Random Forest classification.

```
# Import Tensorflow (ignore possible warning messages)
import tensorflow as tf
# Initialize the network
ann = tf.keras.models.Sequential()
# Add the hidden layers
ann.add(tf.keras.layers.Dense(units = 10, activation = 'relu'))
ann.add(tf.keras.layers.Dense(units = 10, activation = 'relu'))
# Add the output layer
ann.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'
    ))
# Train the network
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy',
    metrics = ['accuracy'])
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
# Test the network (inspect predictions)
y_proba = ann.predict(X_test)
y_pred = (y_proba > 0.5)
```

Program F.10: Creating a deep neural network structure for classification.

```
# Import evaluative metrics
from sklearn.metrics import (confusion_matrix, precision_score,
    recall_score, f1_score, accuracy_score)
# Confusion matrix
conf_matrix = confusion_matrix(actual_labels, predicted_labels)
# Precision score
prec_score = precision_score(actual_labels, predicted_labels)
# Recall score
rec_score = recall_score(actual_labels, predicted_labels)
# F1 score
f_score = f1_score(actual_labels, predicted_labels)
# Accuracy score
acc_score = accuracy_score(actual_labels, predicted_labels)
```

Program F.11: Compute evaluative measures for the classifier.

Regression - Error Prediction

Similar to classification, we used Scikit-learn and Tensor flow libraries to develop our regressors. Creating the training and the test sets is easier than the classification task, since we do not have to separate captured and missed cases. Also, there is no need to perform balancing step, since the target values are continuous numbers in regression. Everything else in splitting the data set remains similar, and `train_test_split()` function can be used similar to classification. The code snippets to develop the regressors in this study are provided below:

```
# Import Scikit-learn modules
from sklearn.model_selection import train_test_split
# Randomely create the training and the test sets
train_data, test_data = train_test_split(dataset, test_size
    =0.2, random_state=20)
```

Program F.12: Splitting the full data set into the training and the test subsets for regression.

```
# Import regression algorithms
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
# Select the regression algorithm
model = RandomForestRegressor(random_state=20, n_jobs=-1)
# model = model = LinearRegression()
# Training
model.fit(X_train, y_train)
# Test (prediction)
y_pred = model.predict(X_test)
```

Program F.13: Training and testing the regressor. The same code structure is used for both Linear Regression and Random Forest.

```
# Import tuning module
from sklearn.model_selection import GridSearchCV
# Select the parameters and the range on which the model will
    be tuned
param_grid = [
        {'n_estimators': [200,300,400],
         'max_features': [20],
         'bootstrap': [False,True],
         'random_state': [20],
         'criterion': ["mse", "mae", "poisson"],
         'max_depth': [6,10,None]
        }
    ]
# Perform tuning
grid_search = GridSearchCV(model, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True,
                           n_jobs = -1)
grid_search.fit(X_train, y_train)
# Extract the tuned model
model_tuned = grid_search.best_estimator_
```

Program F.14: Parameter tuning for the Random Forest regression.

```
# Import Tensorflow (ignore possible warning messages)
import tensorflow as tf
```

```
# Initialize the network
ann = tf.keras.models.Sequential()
# Add the hidden layers
ann.add(tf.keras.layers.Dense(units = 10, activation = 'relu'))
ann.add(tf.keras.layers.Dense(units = 10, activation = 'relu'))
# Add the output layer
ann.add(tf.keras.layers.Dense(units = 1, activation = None))
# Train the network
ann.compile(optimizer = 'adam', loss = 'mse', metrics = [tf.
    keras.metrics.RootMeanSquaredError()])
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
# Test the network (inspect predictions)
y_pred = ann.predict(X_test)
```

Program F.15: Creating a deep neural network structure for regression.

```
# Import evaluative metrics
from sklearn.metrics import (mean_squared_error, r2_score)
import numpy as np
# RMSE
mse = mean_squared_error(true_target, predicted_target)
rmse = np.sqrt(mse)
# R2
r2 = r2_score(true_target, predicted_target)
```

Program F.16: Compute evaluative measures for the regressor.