Non-Intrusive Reduced Order Models for Aerodynamic Applications

Swansea University
Prifysgol Abertawe

Kensley Balla

Zienkiewicz Centre for Computational Engineering
Faculty of Science and Engineering

A thesis submitted in fulfillment of the requirements for the degree of

Doctor of Philosophy

August 26, 2021

Dedicated to my family . . .

Declaration of Authorship

Statement 1

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed : [redacted] (candidate)
Date : August 26, 2021

Statement 2

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed : [redacted] (candidate)
Date : August 26, 2021

Statement 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed : [redacted] (candidate)
Date : August 26, 2021

Acknowledgements

First of all, I would like to thank my supervisors, Prof. Oubay Hassan, Prof. Rubén Sevilla and Prof. Kenneth Morgan for giving me the opportunity to pursue research under their supervisions. I am grateful for their invaluable support and guidance they have provided me over the course of the PhD. Their passion for research and profound knowledge are inspiring. I cannot begin to list what I have learnt. Also, the present work would not have been possible without the financial support from my supervisors and Swansea University. I am very grateful for it.

The three years of study could not have been possible without the support of my friends from across the globe. A huge shout-out to Sanjay, Nidhal, German, Matheus, Leandro, Guillem, Paulo and many others. They made my stay enjoyable and are one of the reasons I could pull through to the day.

I am forever indebted to my family; Mami, Ato Kam, Aji, Kreti and Papi for making me the person I have become, pushing my limits and more importantly, believing in me. It is still difficult to forget my late grandparents who I have lost during the PhD. I really wish they were here. A special mention to Paul for his kind help and constant encouragement to believe in me when I was struggling during my studies, and especially during the pandemic which did not make it any easier.

Abstract

During the design and optimisation of aerodynamic components, the simulations to be performed involve a large number of parameters related to the geometry and flow conditions. In this scenario, the simulation of all possible configurations is not affordable. To overcome this problem, the present work proposes a novel multi-output neural network (NN) for the prediction of aerodynamic coefficients of aerofoils and wings using compressible flow data. Contrary to existing NNs that are designed to predict aerodynamic quantities of interest, the proposed network considers as output the pressure or stresses at a number of selected points on the aerodynamic surface. The proposed approach is compared against the more traditional networks where the aerodynamic coefficients are directly the outputs of the network. Furthermore, a detailed comparison of the proposed NN against the popular proper orthogonal decomposition (POD) method is presented. The numerical results, involving high dimensional problems with flow and geometric parameters, show the benefits of the proposed approach.

The proposed NN is used to accelerate the evaluation of the objective function in an inverse aerodynamic shape design problem. The optimisation algorithm uses the gradient-free modified cuckoo search method. Applications in two and three dimensions are shown, demonstrating the potential of the proposed framework in the context of both optimisation and inverse design problems. The performance of the proposed optimisation framework is also compared against existing frameworks where the more traditional NNs are employed.

Keywords: neural network; proper orthogonal decomposition; reduced order model; geometric parameters; NURBS; shape optimisation; inverse design

Research Output

Journal :

- **KB**, R Sevilla, O Hassan, K Morgan, *An application of neural networks to the prediction of aerodynamic coefficients of aerofoils and wings*, Applied Mathematical Modelling, 2021. https://doi.org/10.1016/j.apm.2021.03.019.
- **KB**, R Sevilla, O Hassan, K Morgan, *A comparison of novel neural networks for the prediction of aerodynamic coefficients of aerofoils and wings*, 2021. *(In preparation)*.

Book chapter :

- **KB**, R Sevilla, O Hassan, K Morgan, *Inverse aerodynamic design using deep neural networks*, ECCOMAS-Springer series, Computational Methods in Applied Sciences, 2021. *(Under review)*.

Conference :

- **KB**, R Sevilla, O Hassan, K Morgan, *Deep neural networks for fast aerodynamic predictions*, 28th Conference of the UK Association of Computational Mechanics, UK, 2021. https://doi.org/10.17028/rd.lboro.14587662.v1.
- **KB**, R Sevilla, O Hassan, K Morgan, *An investigation of neural networks for aerodynamic predictions*, 27th Conference of the UK Association of Computational Mechanics, UK, 2020. https://doi.org/10.17028/rd.lboro.12095931.v1.
  - ***Highly commended award*** at the UKACM Research Highlight Competition
- **KB**, R Sevilla, O Hassan, K Morgan, *Fast aerodynamic predictions using neural networks*, 14th World Congress in Computational Mechanics (WCCM), 2020.
  - Recipient of the ***International Association of Computational Methods (IACM) Scholarship***
- **Poster prize winner** at the annual Zienkiewicz Centre student workshop 2020 at Swansea University, UK.

Contents

List of Figures

List of Tables

List of Abbreviations

CFD	Computational Fluid Dynamics
FV	Finite Volume
LHS	Latin Hypercube Sampling
LSTM	Long Short Term Memory
MLP	Multi-Layer Perceptron
NS	Navier-Stokes
NN	Neural Network
NURBS	Non-Uniform Rational B-Splines
PGD	Proper Generalised Decomposition
POD	Proper Orthogonal Decomposition
RBF	Radial Basis Function
ROM	Reduced Order Model
RANS	Reynold Averaged Navier-Stokes
SA	Spalart-Allmaras

Chapter 1

Introduction

As we have moved from the great pioneers, such as Lanchaster, to the modern age of sophisticated computational methods and integrated ways of working, so we have moved from the art of compromise to the science of optimisation.

Jeff Jupp [1].

1.1 Motivation

Over the last decade, there has been a rapid and steady growth in the volume of air traffic, especially for commercial air transportation and this trend is anticipated to continue in the foreseeable future [2]. Due to the continuous growth of the aviation industry, the carbon dioxide emissions from aircraft are becoming a prominent contributor to climate change and henceforth raising public concerns [2]. In order to address the impact of greenhouse gas emissions from aircraft on the global climate, several international air transport associations have set targets on the carbon emission from air transports. It is reported that by 2050, carbon emissions should be reduced by 50% compared to the carbon level recorded in 2005 [2]. As such, researchers from both academia and the industry are making an effort to deliver innovative technologies to meet the set targets for the next generation of aircraft.

To address the challenge of reducing carbon emission, designers are developing new techniques and methods in various fields, such as employing more efficient jet engines or using composite materials to build the body of the aircraft. Of the wide considerations, one key aspect is to improve the aerodynamic efficiency of the airframe technology. The intrinsic complexity of modern aircraft designs is becoming more dependent on the development and assessment of new theoretical and numerical methodologies which are capable of reducing the cost of experiments or even replacing them. Moreover, these methods are often employed to explore the trade-offs at extreme conditions when a decision on the design path or a selection from a pool of candidates is considered. There are two main characteristics that are usually required in aircraft design analysis: high-fidelity and low-cost. High-fidelity is related to the capability of the theoretical method to replicate real-life phenomena with a high degree of accuracy. A few challenging examples of such flow phenomena include flow transition, separation, shock placement and aerodynamic stall. The application of theoretical or numerical representation of physical behaviours is generally preferred in the early design stage of design processes as global competitions push the increasing number of technical and commercial requirements in a posteriori stages. As a result, a vast number of degrees of freedom exists in the preliminary stage of the design cycle. Semi-empirical tools and rules, which are derived from experiments, are traditionally applied as they are computationally efficient. However, a loss in accuracy should generally be expected when the design path deviates from the conventional configurations. It is therefore relevant to explore unconventional configurations for which semi-empirical simulation tools are not applicable. Strong progress over the last two decades has led to the implementation of highly accurate design methods involving geometry representation and physics modelling [3]. Scientific computing enabled the prototyping of aircraft design through complex physics-based emulators that resulted in saving substantial costs compared to performing experiments in wind tunnels. In fact, the first large body aircraft was entirely designed from simulation based analyses in the 1980s [4]. However physics-based emulators also have their shortcomings. Numerically solving the fluid flow equations is challenging due to the transient and non-linear effects of fluid dynamics. Moreover, this implies a large amount of data storage, data handling and intensive processing cost, even when it is implemented on modern state-of-art computing platforms. Additionally, in the design of a full aircraft, the computational resources needed to support advanced-decision making at the conceptual design phase are prodigious due to the broadness of the flight envelope. It is reported that the complex aerodynamic systems can easily have millions of degrees of freedom requiring wall-clock times in the order of days to obtain a solution, even when resolved in parallel [4]. This turns out to be a much bigger problem when parametric studies or optimisation problems typically are of interest and the computational fluid dynamics (CFD) simulations must be run thousands of times. As a result, it is not feasible to perform the simulation of all required configurations in high-dimensional problems where geometric and flow parameters are involved.

Therefore, there is the need of robust and reliable methods to produce fast aerodynamic predictions while maintaining a high level of accuracy. Researchers in the engineering community across the world are increasingly focused on the use of reduced order models (ROMs). It has become a popular alternative to alleviate the computational burden associated with CFD simulations involving a large number of parameters [5]. ROMs are generally thought of as a computationally inexpensive representation of complex mathematical models that can offer the potential for near real-time analysis. Additionally, it should also provide an accurate and compact representation of the system [5]. Such methods have been heavily used across various fields and they can be broadly categorised into two main groups: the intrusive ROMs and the non-intrusive ROMs [5]. As the name suggests, the two classes differ from each on how intrusive the ROM is to the solver. Both types of ROMs tend to be sensitive to the so-called *training phase* which commonly consists of feeding the ROM with high-fidelity data and finding the set of parameters which best fit the model to the data seen in the training phase. Hence, an effective sampling method of the multi-dimensional design space is required. Designers usually employ design of experiments in an attempt to maximise the amount of information from the multi-dimensional space in the requested number of samples. The selected sampling points are evaluated with the high-fidelity tools and depending on the design objectives and constraints, a model is selected to construct the ROM. That said, sampling the design space is not a trivial task as a trade-off exists between the exploration of the design solutions and the improvement of the accuracy of the model.

1.2 Non-intrusive ROMs in aerodynamic design

Recent progress in aircraft development has witnessed the introduction of non-intrusive approaches as a rapid and cheaper way to address high-dimensional aerodynamic problems. Such approaches, part of the ROM family, are a popular alternative to alleviate the computational burden of CFD simulations in the early design stage of aerodynamic components. The attractive properties of non-intrusive approaches include its simplicity and the ability to deal with complex problems regardless of the nature of the problem. Additionally, it is entirely independent of the source of the data and relies only on the content present in the dataset. One major drawback of such methods includes the reproducibility of complex solutions using regression techniques to adequately represent equally complex systems of partial differential equations [6]. For instance, accurate predictions of highly non-linear aerodynamic behaviours, such as shocks, still remains a challenge [7, 8, 9]. As a result, it is important that the employed non-intrusive ROM is capable of capturing the very possible irregular solutions. Traditional non-intrusive approaches are generally said to be prone to more non-physical predictions as consistency is generally conserved in the solutions of intrusive ROMs. Nonetheless, intrusive ROMs tend to get complicated rapidly when the number of dimensions in parametric studies increases. Additionally, non-intrusive approaches are usually preferred for multi-physics problems to avoid the complication of coupling systems of equations [6].

This thesis focuses only on non-intrusive approaches. Some of the most popular non-intrusive ROMs in the engineering research community that are relevant to the context of this work, are the proper orthogonal decomposition (POD) [10, 11, 12, 13], reduced basis methods [5, 14, 15], eigensystem realisation algorithm [9, 16, 17] and machine learning techniques [18, 19, 20]. The more recent proper generalised decomposition is traditionally intrusive, however there are only few non-intrusive approaches in the literature [21]. The next section provides a brief discussion on the application of two non-intrusive ROMs, namely the POD and the artificial neural networks (NNs), in the aerodynamic design of aerofoils and wings.

1.2.1 The POD

The POD remains the most popular ROMs in the engineering community and is well documented [6]. It is capable of extracting an optimal set of modes and provides a means of truncating the expansion of modes to represent the original state at the prescribed level of accuracy. The POD has been widely adopted via the *method of snapshots*, proposed in [22], for the analysis of unsteady fluid flow problems [23, 24, 25, 26, 12]. The POD has been less commonly employed for solutions of steady fluid flow problems in parametric studies via the *method of snapshots*. The procedure usually begins by generating a set of instantaneous flow solutions or snapshots from the CFD solver in which the design variables are sampled, usually according to a design of experiment. The POD is then used to produce an optimal low-dimensional representation of these snapshots using a finite set of basis functions or modes. Given a new set of design variables, an independent set of equation is solved to obtain the POD coefficient in the low dimensional subspace. The new calculated coefficients are then used in the reconstruction to the original state. It is worth noting that the performance of non-intrusive models in general are sensitive to the sampling of the design space [27]. One of the first work in the construction of POD using steady state solutions for the design of aerofoils is in [28]. The authors generate a set of snapshots using the pressure defined at all the nodes in the mesh and employ least square techniques to interpolate the POD coefficient to obtain the solution of a modified geometry [28]. Other parametric studies focus on only flow conditions [29, 12, 30] or both flow and geometric parameters [31]. Various other works employ the POD in a similar manner, however they only differ in the way the continuous expansion of the POD coefficient is performed for a new set of design variables. A few popular choices include linear functions [30], cubic spline methods [32] least square regression [33] and radial basis functions (RBFs) based on Euclidean distance [12, 31, 34] or Gaussian processes [6]. It is worth noting that the methods employed thus far, have considered the pressure defined at all the mesh nodes or a subset of the nodes in the domain, in an attempt to reconstruct the pressure field.

Other variants to the original implementation of the POD exists. The *gappy* POD was developed to address the problem of missing data points in the snapshots used to build the ROM [35]. This is especially helpful when the data is obtained from experiments and least square fit is generally employed to approximate the missing data points locally [35]. A posteriori POD models have also been built by adapting the ROM with a greedy algorithm, in an attempt to minimise the computational cost of traditional non-intrusive models [36, 37]. The greedy algorithm provides an error indicator on the solutions being adaptively added to the ROM. However, the authors underline the exhaustive search of greedy algorithms when high dimensional problems are considered [36]. Other POD variant includes the so-called *balanced*-POD [39]. It is worth noting that there are a few works in the past which perform a direct interpolation rather than using a ROM, such as in [40, 41, 42, 43]. However the authors in [44] speculate that employing a ROM should be generally better than direct interpolation techniques.

1.2.2 Machine learning and NNs

Machine learning techniques have also been successfully employed as a non-intrusive ROM in many areas of science and engineering [45], including CFD [18, 46]. According to the authors in [47], the application of classic machine learning techniques for the prediction of aerodynamic coefficients at various flow conditions and geometric parameters can be considered as established. NNs, on the other hand, offer a feasible approach to aerodynamic design in the field of fast design space evaluation due to their economic computational consumption and accurate generalisation capabilities [46]. One of the first applications of NNs for the predictions of aerodynamic coefficients can be found in [20], where flight test data was obtained at various angles of attack. The last two decades have seen an explosion in the use of NNs in CFD applications. In [48], two multilayer perceptrons, the fully-connected NN model, were used to predict the lift and drag coefficients separately at different values of Mach number and angle of attack. One of the first applications of NN to study geometrically parametrised aerofoils was presented in the 1990s [49]. The use of the so-called PARSEC aerofoil representation has also been extensively employed to train NNs for the prediction of aerodynamic coefficients [50, 51, 52]. Other geometry parametrisation techniques have also been applied in aerodynamic design, in an attempt to reduce the number of geometric parameters required to define an aerofoil or wing shape. This is specially advantageous to ROMs as the lower the number of parameters defining the geometry, the lower is the computational effort in training the NN or machine learning techniques in general. For instance, the author in [53] employs a hybrid geometric parametrisation using the PARSEC representation and Bezier curves. A comparison of the parametrisation techniques on the accuracy of the aerodynamic predictions goes beyond the scope of this thesis, see [54, 55] for more information. More recent methods that have gained considerable attention in the last five years, are the use of modern NN architecture which are capable of reducing the number of dimensions to define an aerofoil profile [56].

One of the first works where the pressure is predicted over a geometrically parametrised wing is in [57]. The authors use as many NNs as the number of points on the surface to predict the pressure [57]. In [58], the authors train a NN which considers three inputs corresponding to the x, y and z coordinates defining the aerodynamic surface and one output corresponding to the pressure coefficient defined at that location. Hence, the size of the dataset scales with the number of points discretising the aerofoil as well as the number of aerofoils considered in the training. The authors remark that a drawback to this method is that the training of the NN becomes cumbersome as the number of training samples increases [58]. NN was also employed in parametric studies to reconstruct the pressure flow field in the entire domain [59, 60]. The main disadvantage of this approach lies in the number of outputs required to reconstruct the flow field, which scales with the total number of nodes in the mesh. Consequently, more weights need to be optimised and longer training time should be expected. More recently, approaches based on aerofoil images, rather than a mathematical description, have also been considered [61]. A more recent NN architecture, known as the convolutional NN, considers the graphical interpretation of aerofoil shape in the form of images as inputs and the aerodynamic coefficient of interest, the lift, drag or moment is considered as the output [61]. Similar work can be found in [47], however the authors predict all three aerodynamic coefficient using only one network. The author in [62] proposes a non-conventional way to predict the aerodynamic coefficients using convolutional NN. The approach considers images of modified aerofoils as inputs, however the outputs differ to conventional techniques. The pressure defined at the nodes in the computational mesh are discretised into a finite number of intervals or classes. The outputs correspond to the finite number of classes. Hence, the usual regression problem is turned into a classification one. The authors remark that this approach provides a prediction with a mean accuracy of only 80% [62]. Despite the remarkable progress in employing the convolutional NN, it has several limitations. A major drawback in employing the convolutional NN is the loss of information [63]. This is specially crucial in numerical examples where both flow and geometric parameters are synthesised onto an image as in [62, 47, 61]. Moreover, the authors in [64, 65] remark that a larger dataset is generally required in convolutional NN to provide similar accuracies compared to other simpler NN architectures. Consequently, longer training hours should be expected due to its complex architecture.

Physics-inspired NN models have also been built in which a NN is used to map the explanatory variables to the POD coefficients such that for a new set of inputs, the POD coefficients are obtained in the lower dimensional space of the ROM [66, 67, 68]. Using the POD coupled with the NN is relatively new and according to the author's knowledge, a comparison of the performance of the POD coupled with the NN and other interpolation techniques such as RBFs, still remains.

1.2.3 Aerodynamic shape optimisation

The modern way of improving the aerodynamic designs is to define a numerical optimisation problem where the solutions from full order models, such as CFD solvers, are coupled to an optimisation algorithm to effectively automate the process. According to the classification in [69], aerodynamic shape optimisation problems can be categorised into two groups, mainly optimisation design and inverse design. In the optimisation design problem, the optimum shape configuration is found by minimising or maximising an objective function related to a specified aerodynamic performance. It is usually subject to constraints in the geometry and the aerodynamic quantities of interest. Conversely, in the inverse design problem, the optimum shape configuration is usually determined by minimising the difference between a target and the computed pressure distribution. The authors in [69] remark that an ideal aerodynamic system must have both functionalities. The shape optimisation process can be classified into two main approaches, based on the type of the optimiser that is employed.

The first approach involves the use of a gradient-based method which computes the gradients of the cost function with respect to the design variables to determine the direction of descent. One of the earliest studies was performed in [70], where the authors couple the solver with the gradient-based optimiser. The main drawback of this method is the large computational cost associated with the computation of the gradients. To alleviate the curse of dimensionality in the direct computation of the gradient, the author in [71] introduced the concept of adjoint methods where an indirect evaluation of the gradient is performed. Although this method has been proven to be effective when a large number of design variables are used, there are two main drawbacks in employing this method. Firstly, the high development cost of adjoint methods is attributed to the linearisation of the complex systems and this requires modification each time the code is altered [72]. Secondly, this approach becomes more complicated to implement as the number of design variables is increased and is intolerant to noisy objective function spaces, inaccurate gradients, categorical variables and topology optimisation [72]. Additionally, an often-mentioned disadvantage is that gradient-based techniques are generally sensitive to the initial design and therefore may not reach the global optimum in high dimensional design problems. This is attributed to the noisy performance surface and gradient-based method may get stuck in a local minimum.

The second approach employs a gradient-free optimisation algorithm coupled with the solver to find the optimum shape. This approach alleviates the common difficulty in computing the sensitivities of gradient-based techniques. Some of the popular choices are genetic algorithm [73], particle swarm methods [74] and the modified cuckoo search [75]. These optimisers have minimal development cost and are also known to be tolerant to noise in the objective function [72]. The use of a gradient-free optimiser in shape optimisation has proven to be valuable in practice for finding the so-called *global minimum* [76, 77, 78]. It is worth noting that the key disadvantages of the gradient-based methods are the strengths of the gradient-free optimisers. However, these heuristic methods suffer from two main drawbacks, mainly slow convergence near an optimum and a termination criterion is not so straightforward [72]. The computational cost of employing gradient-free optimisation algorithms directly with the solver may become prohibitive due to the need of running a large number of full order CFD simulations [78].

To accelerate the evaluation of the objective function in shape optimisation problem, the most recent trend involves the use of a ROM constructed using the solutions of a full order model [79, 46, 80, 81]. One of the most attractive properties of ROMs is that, once the ROM is built, predictions can be performed in almost real time. ROMs were introduced to tackle the high computational cost of evaluating the full order models, faced by traditional techniques [79]. Traditional techniques include the use of an optimiser directly with the CFD solver as in [70, 71, 72, 77, 78]. Over the years, a number of optimisers and classic non-intrusive ROMs have been employed to obtain the optimum shape configuration, including Gaussian-based regression [80] and radial basis functions [81]. The use of NNs to perform directly the optimisation of aerofoils and wings has also recently attracted considerable attention [82, 53, 83]. A review of NNs in aerodynamic design is available in [46]. When the ROM is employed, there is the choice of gradient-based and gradient-free optimisers to be made. It is reported that the performance of gradient-free algorithms is superior to that of gradient-based algorithms when a ROM is employed due to the existing pitfalls of gradient-based techniques, especially in high dimensional problems, as mentioned earlier [84]. Therefore, it can be deduced that a combination of ROM and gradient-free algorithm can provide an excellent framework for shape optimisation problems due to two main reasons. Firstly, the ROM alleviates the computational cost of the gradient-free optimisation algorithm by accelerating the evaluation of the objective function. Secondly, the gradient-free algorithm is generally insensitive to the initial design and it may converge to the global optimum, provided sufficient information on the optimisation space is obtained in the first generation.

1.3 Scope of thesis

With the motivation and relevant background highlighted in the previous sections, the discussion leads to the key objectives and the contributions accomplished in the present work that are suitable to the aerodynamic design research community. The remaining part of the chapter is dedicated to present the outline of the thesis.

1.3.1 Aims and objectives

The primary objective of this work is to establish a reliable and robust non-intrusive ROM that enables the fast evaluation of aerodynamic coefficients of aerofoils in two dimensions and wings in three dimensions. A posteriori ROMs are considered and they require the construction of a database. Steady Euler and NS calculations are performed to build the database. To align the objectives of the current work with some of the challenges identified by NASA's 2030 CFD vision [85], the design of the aerodynamic components is aimed to have a higher degree of automation involved in geometry creation, mesh generation and adaption, creation of large simulation databases, extraction and understanding the vast amount of information generated, and lastly the ability to guide the process. Furthermore, the automated system is also anticipated to have high levels of reliability and robustness to minimise human intervention.

To these aims, a non-intrusive ROM is built using a multi-output NN for the fast predictions of aerodynamic coefficients on aerofoils and wings in numerical examples involving flow conditions and parametrised geometries. The output of the proposed NN consists of the pressure evaluated at a number of points on the aerodynamic surface. Here this is done using a single NN, rather than as many NNs as the number of points on the surface, as performed in [57].The performance of the proposed NN is compared with the most common approach found in the literature, where a NN with a single output, corresponding to one aerodynamic coefficient, the lift, drag or moment, is considered [86, 50, 87, 51, 49, 61]. The proposed NN is also compared against another existing NN where all aerodynamic coefficients are considered as the outputs, introduced in [47]. Additionally, the performance of the proposed approach is compared to the classic ROM in the literature, the POD, in the numerical examples in two dimensions.

In an attempt to improve the aerodynamic efficiency of aerofoils and wings, as well as addressing the challenge of reducing carbon footprints by 2050 [2], an efficient optimisation framework is presented using the proposed non-intrusive ROM and a global optimisation algorithm. It is applied to both optimisation and inverse design problems. The cost of evaluation of the objective function is alleviated by the use of the non-intrusive ROM, as in [82, 53, 83]. A comparison with an existing framework is also performed to identify the advantages of the proposed framework.

1.3.2 Outline

A brief outline of the structure of the thesis is given below:

- **Chapter 2 :** A brief introduction to the evolution of CFD and the selection of available approximation methods in aerodynamic design is provided. The formulations implemented in the full order CFD model is presented. Additionally, the mesh generation and the discretisation of the governing equations of the NS equations are detailed. Finally, an overview of the implemented solution procedure in the full order model is provided.
- **Chapter 3 :** The chapter aims at discussing ROMs which are commonly employed in CFD applications. Two concepts that are used as a non-intrusive ROM, are introduced. Firstly, a brief overview of the POD via the method of snapshots is provided. Some of most popular choices in interpolating the POD coefficients is written. Secondly, the NN is reviewed with the forward propagation and back-propagation detailed in steps. Several optimisation algorithms adapted to NNs are also considered. Finally, a brief discussion on popular design of experiment techniques are discussed.
- **Chapter 4 :** The chapter presents the application of the proposed NN using numerical examples that involves flow parameters. The wide range of flow conditions considered in the examples leads to the subsonic and transonic regime. The examples considered shows the benefits of the proposed NN against existing NNs and the POD using several strategies. The influence of the number of

training cases on the accuracy of the predictions for the NN and POD is demonstrated. The chapter also includes an extension to fast aerodynamic prediction on three dimensional wings using the proposed NN. Special emphasis is placed on the influence of the accuracy of the CFD data on the predictions of the employed ROM.

- **Chapter 5 :** The numerical examples presented in this chapter demonstrates the application of the proposed NN to the fast predictions of aerodynamic coefficient in problems involving geometric parameters. The examples have various levels of complexity and a comparison on the performance of the proposed NN against existing NNs and the POD is shown. The influence of the number of training cases on the accuracy of the predictions is also explored. An application to using the proposed NN and the modified cuckoo search is presented for an optimisation and inverse design problem. Additionally, the superior performance of the proposed technique in the context of optimisation design is demonstrated against existing NNs. Finally, an example on deforming wings over a wide range of conditions is shown to exemplify the robustness of the proposed method.

- **Chapter 6 :** The thesis ends with some concluding remarks. It recalls the contributions of the present work and comments on the future outlook on aerodynamic design using current technologies.

- **Appendix A :** The appendix provides additional supporting materials that have been employed in this work for completeness and consists of three sections. The first section provides a brief description of the non-uniform rational B-splines (NURBS). The second section describes the Delaunay graph mapping that was employed to deform a reference mesh. The third section presents an overview of the cuckoo search algorithm with remarks on the key implementation of the modified cuckoo search algorithm.

- **Appendix B :** The appendix shows the comparison of the performance of two NN architectures, mainly the extended multi-layer perceptron and the long short term memory network in a numerical example involving flow parameters.

Chapter 2

The Full Order Model

The fluid flow problems in this thesis are governed by the compressible Euler and NS equations for a calorically perfect gas in their stationary form. This chapter begins by providing a brief introduction to the evolution of CFD and the selection of available approximation methods. The work in this thesis employs the *Flite* system, which is an in-house CFD solver [88]. A summary of the formulations implemented in the full order model is presented. Furthermore, the mesh generation and the discretisation of the governing equations for the flow problems relevant to this work are detailed. Finally, an overview of the implemented solution procedure is provided.

2.1 Introduction

Before the development of transistors, the solutions of fluid problems relied on hand calculations by discretising the domain and applying basic numerical methods. An English mathematician Lewis F. Richardson was one of the pioneers in the field to numerically solve the governing equations by hand, for the purpose of weather predictions [89]. The numerical analysis performed on a cylinder at low Reynolds numbers was first documented in the 1930s [89]. The researcher described that the flow field should be broken down into a set of squares of any size and finer grid sizes would provide more accurate solutions where the gradients in the flow are large. Fortunately, due to the development of transistors in the second half of the 20th century, modern high performance computers were able to handle extremely high-speed calculations to produce reliable solutions. This has led CFD to become an important tool in the analysis of fluid flow problems in both academia and the industry. The wide range of applications has also caused the development of various methods to meet the need of

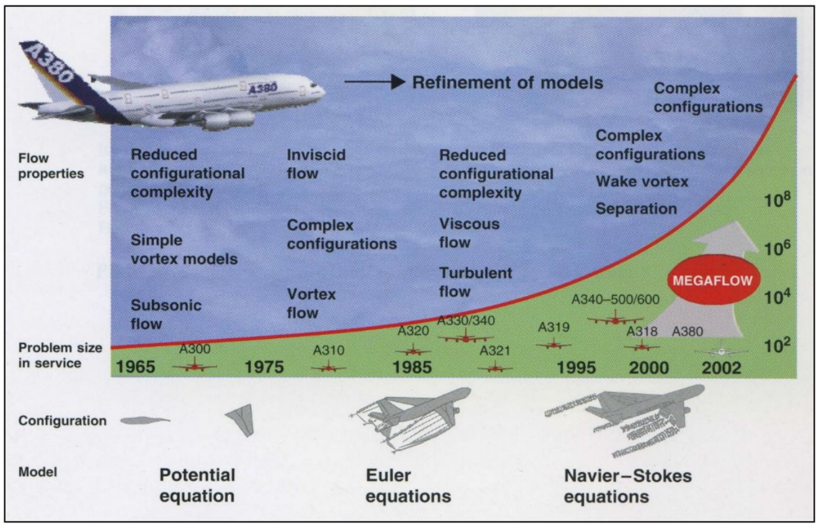


Figure 2.1: Evolution of the complexity of the flow solved over the 40 prominent years of CFD developments at Airbus, with an indication on the advancement of the applied methods, taken from [3].

users. Each approach differs from each other in the physical and computational accuracy, or precision and speed, based upon the assumptions that underlay the model and the quality of any potential discretisation.

The CFD study of aerodynamic flows has evolved to embrace the advancements over the years. Figure 2.1 shows the evolution in the uptake of CFD methods between 1965 and 2002 at Airbus. Accurate solutions obtained with the progress of numerical and computational capabilities have allowed new opportunities. For instance, it has allowed for improving aerodynamic designs, reduced expenditures in experimental testing and more importantly, broaden the understanding of physical phenomena. A recent report of NASA's CFD vision for 2030 has identified key areas that requires high-priority research in order to advance the fields and obtain accurate solutions to the NS equations [85]. Amongst the discussed strategies relevant to this work is a higher degree of automation in all steps involved at the early design stage of aerodynamic components. This involves geometry creation, mesh generation and adaptation, creation of large simulation databases, extraction and understanding of the databases, and the ability to computationally guide the process. The report also underlines that every step in the process should have high levels of reliability and robustness to minimise human intervention [85]. Another identified area is the flexibility to tackle capability and capacity of computing tasks in both industry and research environment. This involves large datasets of compact solutions or small datasets of large solutions that should be readily accomplished with the computational resources available [85].

Next, the computation of a flow solution requires the careful selection of a numerical or analytical approximation to the NS equations. The computational cost of the NS equations is approximately proportional to $\mathcal{O}(Re^3)$, where Re denotes the Reynolds number [90]. Direct numerical simulation relies on some of the most powerful computers in the world to produce solutions over simple geometries such as spheres and flat plates [91]. The standard CFD approximations shown in figure 2.2 are well known and can be found in [92].

Aerodynamic flows are physically approximated based on three main methods, namely, physical assumptions, mathematical reductions and modelling. The first method explicitly removes physical phenomena from the equations which consequently reduces the size of the system of equations to solve. An example is that of potential flow methods which assume that the flow is incompressible and irrotational. The second method allows a simplification of the governing equations without explicitly removing physical phenomena or performing a simplification on the assumptions made upon a certain behaviour. An example of this approach includes the time-averaging performed in the derivation of the Reynolds averaged Navier-Stokes (RANS) equations where the temporal variability of the viscous terms are first averaged and then modelled. The third method models any behaviour that are not well represented using the current methodology. The higher accuracy obtained with the last approach comes at a cost, however it is still significantly cheaper than solving the NS equations directly. An example is the large eddy simulation which models the micro-scale turbulence using a semi-analytical and empirical model.

For examples involving geometric parameters, potential flow methods are unable to represent arbitrary shapes and produce the correct approximations as the methods rely on the user specification of the locations of the stagnation point and separation point to enforce a flow pattern. On the other hand, the cost of large eddy simulation and direct numerical simulation makes their use impractical. In the case where only compressibility is required, solving the Euler equation is a popular choice and when both compressibility and viscosity is required, RANS model is usually selected as it can capture the correct trends between different inflow and geometric configurations [3].

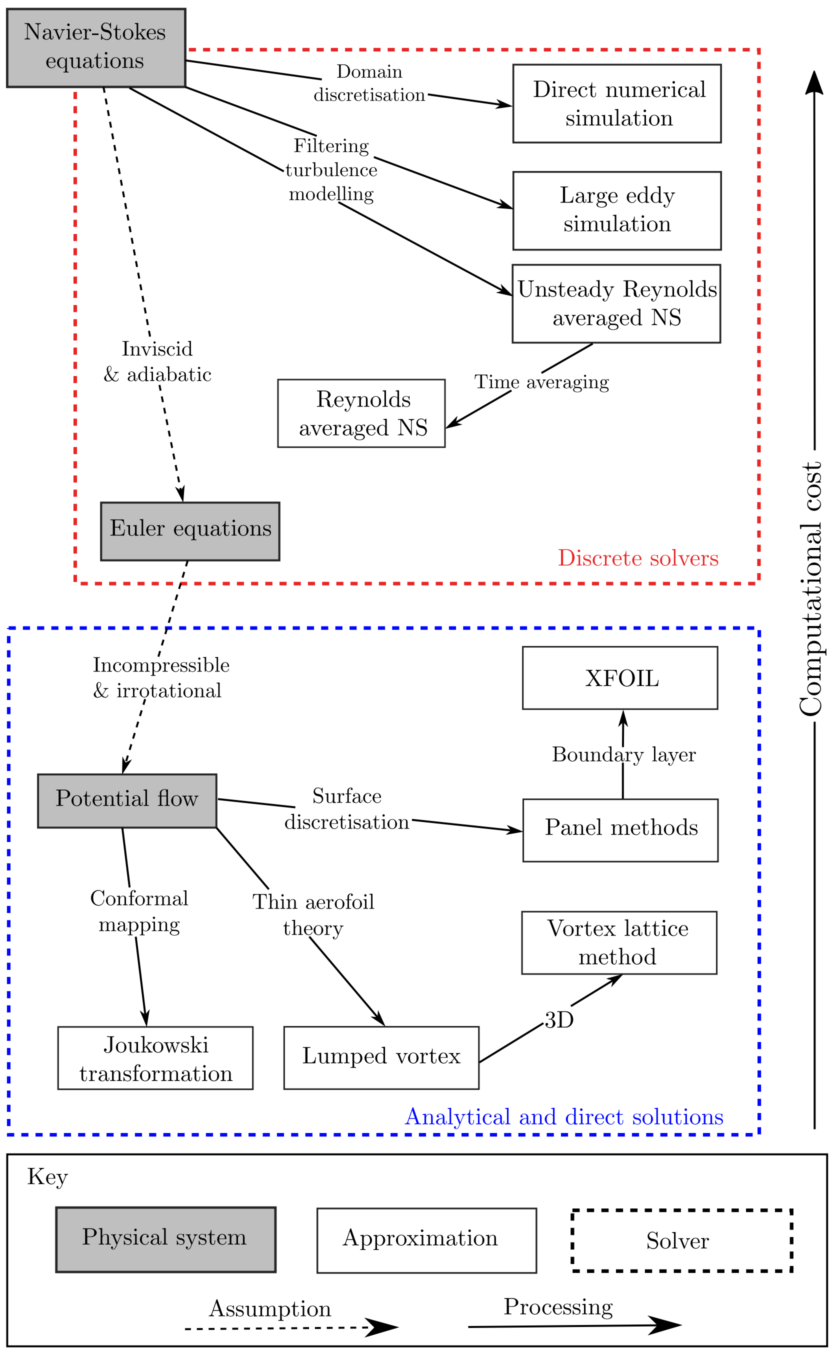


Figure 2.2: Standard CFD approximations to the NS equations [92].

The accuracy of a solution coming from an Eulerian formulation, a standard CFD approach for simulations without free-surfaces, is highly dependant on the quality of discretisation used for the surface and the domain. The discretisation can be achieved using two approaches, the structured and unstructured meshing techniques. The structured mesh has the benefits of having elements aligned with the principal directions of the flow which further simplifies the analysis of viscous flow problems. On the other hand, unstructured meshes allow for a wider range of element shapes and are boundary conforming to complex geometries [93, 94].

The selection of discretisation methods is also dependent on the equations being computed and the complexity of the expected flow features. For instance, the turbulence models used in RANS simulations are particularly sensitive to the rate of growth in the normal direction away from the no-slip wall boundaries. As a result, very small elements and smooth gradation to the outer boundary are required to accurately model the growth of the boundary layer. That said, mesh generation is not a trivial task. Moreover, it is of utmost importance to obtain an accurate description of the discretised geometry in the computational domain [95]. As a matter of fact, a low fidelity representation of the geometry is known to have a major impact on the solution accuracy, as well as the convergence to the correct solution [96, 97]. This has led the mesh generation community to produce fast and robust mesh generation tools [97]. However, providing automatic generation of high quality meshes with minimal user intervention is still considered a challenge in the research community [85, 95].

2.2 Formulation of compressible flows

This section is devoted to the full order formulation of the compressible NS equations in their stationary form and under the assumption of a calorically perfect gas. A brief overview of the turbulence model using the Favre-averaging procedure is provided. Furthermore, a short discussion of the boundary conditions relevant to this work is presented. Finally, the computation of aerodynamic quantities of interest is written.

2.2.1 Governing equations

The fluid flow problems considered in this work are governed by the NS equations for a compressible fluid. The strong form of the problem, in a computational domain $\Omega \subset \mathbb{R}^d$ in d spatial dimensions and in the absence of external volume forces, can be written as

$$\begin{aligned} \boldsymbol{U}_t + \boldsymbol{\nabla} \cdot \boldsymbol{F}(\boldsymbol{U}) - \boldsymbol{\nabla} \cdot \boldsymbol{G}(\boldsymbol{U}) &= \boldsymbol{0} \quad \text{in } \Omega \times (0, T_f]\,, \\ \boldsymbol{U} &= \boldsymbol{U}_0 \quad \text{in } \Omega \times \{0\}, \\ \boldsymbol{B}(\boldsymbol{U}, \boldsymbol{U}^\infty) &= \boldsymbol{0} \quad \text{in } \partial\Omega \times (0, T_f]. \end{aligned} \tag{2.1}$$

The vector of conservation variables, $\boldsymbol{U}$, the flux tensor, $\boldsymbol{F}$, and the viscous tensor, $\boldsymbol{G}$, are given by

$$\boldsymbol{U} := \begin{Bmatrix} \rho \\ \rho \boldsymbol{v} \\ \rho E \end{Bmatrix}, \qquad \boldsymbol{F} := \begin{bmatrix} \rho \boldsymbol{v}^T \\ \rho \boldsymbol{v} \otimes \boldsymbol{v} + p\mathbf{I}_d \\ (\rho E + p)\boldsymbol{v}^T \end{bmatrix}, \qquad \boldsymbol{G} := \begin{bmatrix} 0 \\ \mathbf{T} \\ (\boldsymbol{v}^T\mathbf{T} - \boldsymbol{q}^T) \end{bmatrix}, \tag{2.2}$$

Here, $\boldsymbol{U}_0$ denotes the initial condition, T_f is the final time and $\boldsymbol{B}$ is the generic flux used to define the boundary conditions over the inflow, outflow and wall boundaries. In the above expressions ρ is the density, $\rho\boldsymbol{v}$ is the momentum, ρE is the total energy per unit volume, p is the pressure and $\mathbf{I}_d$ is the identity matrix of dimension d. The contribution of the deviatoric stresses in the direction j is written as $\mathbf{T}_j = [\tau_{1j}, \tau_{2j}, \tau_{3j}]^T$ and contains the deviatoric stress components which can be expressed in terms of the velocity gradients as

$$\tau_{ij} = -\frac{2}{3}\mu \sum_{k=1}^{d} \frac{\partial v_k}{\partial x_k}\delta_{ij} + \mu\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right), \quad \text{where} \quad \begin{matrix} \delta_{ij} = 1 & \texttt{if} & i = j \\ \delta_{ij} = 0 & \texttt{if} & i \neq j \end{matrix} \tag{2.3}$$

and for $i, j = 1, 2, .., d$. v_l is the l-th component of the velocity vector. μ is the dynamic viscosity, δ_{ij} is the Kronecker delta, and the heat flux is given as

$$\boldsymbol{q} = -\kappa \frac{\partial \check{T}}{\partial \boldsymbol{x}}, \tag{2.4}$$

where κ is the thermal conductivity and $\check{T}$ is the absolute temperature, measured in Kelvin. It is assumed that the dynamic viscosity varies with temperature according to the Sutherland's law,

$$\frac{\mu}{\mu_\circ} = \left(\frac{\check{T}}{\check{T}_\circ}\right)^{1\backslash 2} \frac{\check{T}_\circ + 110}{\check{T} + 110}, \tag{2.5}$$

where $\mu_\circ$ and $\check{T}_\circ$ are the reference state. The Prandtl number is expressed as

$$Pr = \frac{c^p \mu}{\kappa}, \tag{2.6}$$

taken to be constant and $Pr = 0.72$. In this expression, c^p is the specific heat at constant pressure. The system of equations is closed with the equation of state for a perfect polytropic gas, in the form

$$p = (\gamma - 1)\rho \left(E - \frac{1}{2}\|v\|^2 \right), \tag{2.7}$$

where $\gamma = 1.4$ is the ratio of the specific heats for air. The flow around a given geometry with its corresponding boundary conditions can be fully described using the Prandtl, Reynolds, Mach, and Strouhal numbers, written as

$$Pr_\infty = \frac{c^p \mu_\infty}{\kappa_\infty}, \quad Re_\infty = \frac{\rho_\infty \|v\| l}{\mu_\infty}, \quad M_\infty = \frac{\|v\|}{c_s}, \quad St_\infty = \frac{\|v\| t_c}{l}, \tag{2.8}$$

going from left to right, respectively. In the above expressions, the subscript ∞ denotes the freestream values, l is a specified reference length scale, t_c is a characteristic time scale and $c_s = \sqrt{\gamma p / \rho}$ is the speed of sound.

2.2.2 Turbulence modelling

The most challenging part in CFD simulations is to accurately and efficiently capture the complex and non-linear features of turbulent flows. The necessity to solve turbulent flow problems resulted in the development of turbulence modelling. The selection of an appropriate turbulence model is required to accurately model any given problem [98]. There are three main approaches that are commonly employed to resolve turbulent flows, mainly, the direct numerical simulation, large eddy simulation and RANS.

The principle behind direct numerical simulation is to directly solve all of the turbulence length and time scales. An adequately fine mesh is needed, which subsequently requires large computational resources [99]. Due to this high computational demand, the author in [100] underlines that direct numerical simulation is only suitable to solve low Reynolds number flow problems or domains with smaller degrees of freedom. As most research in aerodynamics involves high Reynolds number flow problems and large domains, solving the governing equations directly is not feasible. In an attempt to reduce the computational efforts to solve a given flow problem, researchers have modelled or approximated the turbulent flows at various levels of accuracy. Moreover, it also depends on how much of the NS equation wants to be retained. The large eddy simulation is an example in which a semi-analytical and empirical approach is used to model the micro-scale turbulence. It is assumed at this level, turbulence is reasonably homogeneous and easier to model compared to large-scale chaotic flow. According to [98], the authors note that large eddy simulation compares well against direct numerical simulation as the CPU time for large eddy simulation is between five to ten percent of that needed for direct numerical simulation. However, the authors in [101] report that for high Reynolds number flows, large eddy simulation also requires large computational resources because of its time-dependent nature and hence is sometimes deemed too expensive for some aerodynamic analyses. For instance, large eddy simulation requires approximately 600 million elements to discretise the domain in the near-wall region of $y^+ < 20$ for a wall bounded flow around a simple vehicle [102]. Moreover, large eddy simulation requires precise level of turbulence at the inlet and regular isotropic elements with smooth and gradual spatial change in resolution outside the wall region in order to minimise the erosive effect on numerical dissipation [103].

The next level of approximation is the so-called RANS which represents the dominant approach in industrial applications. It involves the modelling of turbulence at all characteristic length scales by decomposing the flow variables into the mean and fluctuating parts, followed by an averaging procedure [104]. This is favourable in the context of aerodynamic design exploration where multi-query simulations are performed and the averaging procedure speeds up the process of obtaining a solution to the NS equations. It is usually recommended to apply the density weighted averaging procedure, also referred to as the *Favre averaging* [105], to certain flow quantities in compressible flows. This is usually performed to avoid the complications in correlation terms involving density fluctuations. Additionally, when a statistically averaged steady-state solution of compressible flows is sought, as in this work, Favre averaging is the most popular choice [94].

Favre averaging relies on the Morkovins hypothesis. It states that if fluctuations are small compared to the mean density in the flow, then the effects of the density fluctuations on the turbulence will also be small [106]. This assumption is however only valid for Mach numbers up to at most five [98]. The Favre averaging procedure can be described by first introducing the Reynolds time-averaging using a general variable u as

$$\overline{u} = \frac{1}{\Delta t} \int_{t}^{t+\Delta t} u(\boldsymbol{x}, t) \, \mathrm{d}x, \tag{2.9}$$

where $\overline{u}$ is the averaged quantity and Δt denotes the time over which the variable u

is averaged. The time should be large enough to alleviate small turbulent oscillations. The Favre averaging of the generic variable, $\tilde{u}$ can therefore be written as

$$\tilde{u} = \frac{\overline{\rho u}}{\overline{\rho}}, \tag{2.10}$$

which represents the density-weighted version of the Reynolds procedure. The fluctuations can now be defined using these definitions as

$$u' = u - \overline{u} \quad \text{and} \quad u'' = u - \tilde{u}, \tag{2.11}$$

for the Reynolds and Favre averaging procedure respectively.

By inserting the decomposed variables into the NS equations and performing the averaging, the same equations for the mean variables are obtained with the exception of two additional terms in the momentum and energy equations. The first term corresponds to the *Reynolds-stress tensor*, $\mathbf{T}^R = -\overline{\rho \boldsymbol{v}'' \otimes \boldsymbol{v}''}$, where $\boldsymbol{v}''$ denotes the density-weighted fluctuating parts of the velocity components $\boldsymbol{v}$. The second term corresponds to the turbulent heat flux, denoted as $\boldsymbol{q}^R = -\overline{\rho h'' \boldsymbol{v}''}$.

There are two major benefits of this approach. Firstly, coarser meshes can be used compared to large eddy simulations, and secondly stationary mean solution can be assumed for attached and moderately separated flows. These two features significantly reduces the computational effort in comparison to solving the NS equations directly or using large eddy simulation. As a result, the RANS approach is a popular choice in engineering designs and applications.

The two additional terms that were obtained by applying the Favre averaging procedure, that is the Reynolds-stress tensor and the turbulent heat flux, have to be approximated to close the system of equations. A large variety of turbulence models was devised to close the RANS equations. The models can be broadly categorised into two groups, the Reynolds-stress model and the turbulent viscosity model, respectively. The Reynolds-stress method models the Reynolds-stress tensor directly. On the other hand, the Boussinesq assumption is made in the turbulent viscosity model, where the turbulent effects are treated as diffusive effects in the NS equation. For instance, this assumption on the deviatoric stress contribution can be mathematically written as

$$\tau_{ij}^R = -\frac{2}{3}\mu^{tb}\sum_{k=1}^{d}\frac{\partial \tilde{v}_k}{\partial x_k}\delta_{ij} + \mu^{tb}\left(\frac{\partial \tilde{v}_i}{\partial x_j} + \frac{\partial \tilde{v}_j}{\partial x_i}\right) - \frac{2}{3}\overline{\rho}\kappa\delta_{ij}\,, \tag{2.12}$$

where μ^{tb} is the turbulent viscosity and the last term is added to ensure that the relation $\tau_{ii}^R = -2\overline{\rho}\kappa$ is respected. κ denotes the turbulent kinetic energy term. The analogy of Reynolds to relate the momentum and heat transfer can be written as

$$\boldsymbol{q}^R = -\mu^{tb}\frac{c^p}{Pr^{tb}}\frac{\partial \tilde{T}}{\partial \boldsymbol{x}}\,, \tag{2.13}$$

where c^p is the specific heats at constant pressure and the turbulent Prandtl number is usually taken to be $Pr^{tb} = 0.9$ for air. The full order model employs the turbulent viscosity model in the examples considered due to numerical difficulties caused by the stiffness in the Reynolds stress model [107].

Next, the turbulent viscosity coefficient μ^{tb} has to be determined with the aid of a turbulence model. There are various turbulent viscosity models within this paradigm, which can be grouped by the number of differential equations used to model the turbulent viscosity. The first turbulent viscosity model is known as the *zero-equation model*, which does not require the solutions to the differential equations to model the turbulent viscosity. It is generally inspired by Prandtl's mixing-length theory [108] which relates the turbulent viscosity to the momentum transfer in the shear layers. Two of the most commonly used methods in this class are the Cecebi-Smith [109] and Baldwin-Lomax [110] models. These models provide satisfactory results for flows involving simple geometries and small separations, however there are limitations to their applicability to complex geometries.

The second type of turbulence models uses one differential equation, usually referred to as *one-equation* model, and represents significant improvement in modelling the turbulent viscosity. Two popular choices of this scheme are the Baldwin-Barth [111] and the Spalart-Allmaras (SA) [112] models. These models were designed specifically for aerodynamic applications and compare well for a wide variety of problems under this category [113]. It is known that the SA model is superior to the Baldwin-Barth model as it exhibits mesh dependency [114]. One disadvantage of the SA model is that the specification of the trigger location where a laminar flow becomes turbulent, is required. However, it is argued that most turbulence models are incapable of finding the separating regions and the trigger location approach can be easily adapted to a separation-detection model to alleviate the need of user input [112].

The third type of turbulence models are the *two-equation* models which solves two differential equations to obtain the turbulent viscosity. The most popular choice of this type are the $\kappa - \varepsilon$ [115], $\kappa - \omega$ [116], and the shear-stress transport model [117]. The turbulent kinetic energy κ and dissipation rate ε are used to model the turbulent viscosity in the $\kappa - \varepsilon$ model. Conversely, the turbulent kinetic energy and the specific dissipation rate ω are used to model the turbulent viscosity in the $\kappa - \omega$ model. The shear-stress transport model blends both the $\kappa - \omega$ and $\kappa - \varepsilon$ models and has two distinct benefits than its counterparts when employed separately. Firstly, the shear-stress transport model takes advantage of the good stabilising properties and accurate predictions of adverse pressure gradients in flows using the $\kappa - \omega$ model near the surface [118]. Secondly, it also exploits the superior characteristics of the $\kappa - \varepsilon$ model near the boundary layer edge [118]. The author in [98] underlines that the two-equations models are superior to the one-equation models in engineering applications and that the latter shows marginal improvements to the zero-equation model. However, it is also reported in [113] that the overall performance of the SA model compares well with the $\kappa - \varepsilon$ and $\kappa - \omega$ models, and is inferior to the shear-stress transport model in regions of separation. One main drawback for this class of turbulence models is that they require more calculations in the boundary layers compared to other models.

It can therefore be deduced that there is no perfect approach in modelling turbulence accurately for all engineering problems. In this work, the SA model is employed as it offers numerical convenience over improved accuracy, which is often required in the early design stage of aerodynamic components.

A brief summary of the benefits and formulations of the SA model follows. The one-equation SA model was derived using arguments of dimensional analysis, empiricism, Galilean invariance and selective dependence on the molecular viscosity [119]. It was calibrated using results from two dimensional mixing layers, wakes and flat-plate boundary layers, respectively [107]. The model allows for reasonably accurate predictions of turbulent flows with adverse pressure gradients. Moreover, there is usually a smooth transition from laminar to turbulent flow. According to the authors in [107], the SA model is robust, fast to converge to steady-state solutions and only requires moderate mesh resolution in the near-wall regions. The kinematic turbulent

$c_{b1} = 0.1355$	$c_{b2} = 0.622$	$\varrho = 2/3$
$\varkappa = 0.41$	$c_{w2} = 0.3$	$c_{w3} = 2$
$c_{v1} = 7.1$	$c_{t1} = 1$	$c_{t2} = 2$
$c_{t3} = 1.1$	$c_{t4} = 2$	

Table 2.1: Constants defined in the Spalart-Allmaras turbulence model.

viscosity is related to an eddy viscosity, $\tilde{\nu}$ using the definition,

$$\nu^{tb} = \frac{\mu^{tb}}{\bar{\rho}} = \tilde{\nu} f_{v1}(\tilde{\nu}), \tag{2.14}$$

where the eddy viscosity is obtained using the transport equation, written as,

$$\begin{aligned}\tilde{\nu}_t + \tilde{\boldsymbol{v}} \cdot \boldsymbol{\nabla} \tilde{\nu} = c_{b1}(1 - f_{t2})\tilde{S}\tilde{\nu} + \frac{1}{\varrho}\left\{ \boldsymbol{\nabla} \cdot ((\nu + \tilde{\nu})\boldsymbol{\nabla}\tilde{\nu}) + c_{b2}(\boldsymbol{\nabla}\tilde{\nu})^2 \right\} \\ -(c_{w1} f_w - \frac{c_{b1}}{\varkappa^2} f_{t2})\left(\frac{\tilde{\nu}}{d}\right)^2 + f_{t1}\Delta\|\tilde{\boldsymbol{v}}\|_2.\end{aligned} \tag{2.15}$$

The left hand-side of the transport equation represents the Langrangian derivative of the eddy viscosity $\tilde{\nu}$. The terms on the right hand-side correspond to the production, diffusion, destruction and transition going from left to right, respectively. The unknown terms introduced in the transport equation are defined as

$$\begin{aligned} f_{v1} &= \frac{\chi^3}{\chi^3 + c_{v1}^3}, & \chi &= \frac{\tilde{\nu}}{\nu}, \\ \tilde{S} &= \tilde{\omega} + \frac{\tilde{\nu}}{\varkappa^2 d^2} f_{v2}, & f_{v2} &= 1 - \frac{\chi}{1 + \chi f_{v1}}, \\ f_w &= g\left(\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6}\right)^{1\backslash 6}, & g &= r + c_{w2}(r^6 - r), \\ r &= \min\left(\frac{\tilde{\nu}}{\tilde{S}\varkappa^2 d^2}, 10\right), & f_{t2} &= c_{t3} e^{-c_{t4}\chi^2}, \\ g_t &= \min\left(0.1, \frac{\Delta\tilde{v}}{\tilde{\omega}_t \Delta x}\right), & f_{t1} &= c_{t1} g_t e^{-c_{t2}\frac{\tilde{\omega}_t^2}{\Delta\tilde{v}^2}(d^2 + g_t^2 d_t^2)}, \\ c_{w1} &= \frac{c_{b1}}{\varkappa^2} + \frac{1 + c_{b2}}{\varrho}, \end{aligned} \tag{2.16}$$

where the constants introduced in the formulations are defined in Table 2.1. In the above expression, d is the distance from a given point to the nearest wall and $\tilde{\omega} = \|\boldsymbol{\nabla} \times \boldsymbol{v}\|_2$ is the vorticity magnitude, $\Delta\tilde{v}$ is the difference in velocity between a point and an associated user-defined trigger point in two dimensions or curve in three dimensions, d_t denotes the closest distance to that trigger location and Δx is the surface grid-spacing at the trigger location.

Hence, the resulting Favre-averaged NS equation to solve, reads as

$$\hat{\boldsymbol{U}}_t + \boldsymbol{\nabla} \cdot \hat{\boldsymbol{F}}(\hat{\boldsymbol{U}}) - \boldsymbol{\nabla} \cdot \hat{\boldsymbol{G}}^{tb}(\hat{\boldsymbol{U}}) = \boldsymbol{0} \quad \text{in}\, \Omega \times (0, T_f] \tag{2.17}$$

where $\hat{\boldsymbol{U}}$ and $\hat{\boldsymbol{F}}$ denote the Favre-averaged vector of conservation variables and flux tensor, respectively. The resulting viscous tensor $\hat{\boldsymbol{G}}^{tb}$, when combined with the correlation tensor, is written as

$$\hat{\boldsymbol{G}}^{tb} = \begin{bmatrix} 0 \\ \tilde{\mathbf{T}}^{tb} \\ (\tilde{\boldsymbol{v}}^T \tilde{\mathbf{T}} - \tilde{\boldsymbol{q}}^{tb\,T}) \end{bmatrix}, \tag{2.18}$$

where the turbulent deviatoric stress tensor and the turbulent heat flux is set as $\tilde{\mathbf{T}}^{tb} = \tilde{\mathbf{T}} + \mathbf{T}_{ij}^R$ and $\tilde{\boldsymbol{q}}^{tb} = \tilde{\boldsymbol{q}} + \boldsymbol{q}^R$, respectively. The turbulent viscosity, μ^{tb}, which appears in the turbulent deviatoric stress tensor $\tilde{\mathbf{T}}^{tb}$, is computed using the SA model.

2.2.3 Boundary conditions

The problem is well defined when boundary conditions are applied on the boundaries of the computational domain. There are four main types of boundary conditions, the wall, symmetry, inflow and the outflow boundary condition, respectively.

The *wall* boundary condition ensures that the relative normal component of the velocity at the wall must vanish to prevent flow traversing through the surface such that the relation $v_i n_i^w = v_i^w n_i^w$ holds in inviscid flows. The fluid particles are prescribed to stick to the wall in viscous flows, that is, $v_i = v_i^w$. This shows that the velocity at the wall is zero if it is stationary. In addition to the boundary conditions imposed on the velocity components, one boundary condition must be imposed on the temperature. Two popular choices are the assumption of an isothermal wall where $\check{T} = \check{T}_w$ or alternatively, a Neumann boundary condition assuming the wall to be isentropic, that is,

$$\frac{\partial \check{T}}{\partial \boldsymbol{n}^w} = \frac{\partial \check{T}}{\partial \boldsymbol{x}} \cdot \boldsymbol{n}^w = 0. \tag{2.19}$$

In this work, the assumption of an adiabatic wall is considered and the turbulent viscosity is set to zero at the wall boundaries as there are no fluctuations in velocity.

Next, the *symmetry* boundary is a boundary in the domain where the normal velocity component with respect to that boundary is zero and, the density and total energy have zero normal derivatives, that is,

$$\boldsymbol{u} \cdot \boldsymbol{n}^s = 0, \quad \frac{\partial \rho}{\partial \boldsymbol{n}^s} = \frac{\partial \tilde{e}}{\partial \boldsymbol{n}^s} = 0, \tag{2.20}$$

where $\boldsymbol{n}^s$ denotes the normal to the symmetry surface. The symmetry boundary condition is particularly useful for reducing the size of the problem when symmetry planes are known. An application of one such problem is the flow around a symmetric aircraft with no side-slip angle.

It is worth noting that the number of each boundary condition required at a particular point is determined by the number of characteristics entering the domain. As a result, the number of boundary conditions at the inflow and outflow regions may differ. Moreover, the number and type of boundary conditions relevant to a specified boundary in the domain depends on the system of equation considered and on the local Mach number. Only the boundary conditions relevant to this work is discussed, that is, for subsonic and transonic flows.

The numerical simulation of external flows are performed within a bounded domain. Artificial *farfield* boundary conditions are implemented to meet two requirements. Firstly, the truncation of the domain should have no sizeable effect on the flow solution compared to an infinite domain. Secondly, any outgoing disturbances cannot be reflected back into the flow field [120]. The concept of characteristic variables is employed when selecting boundary conditions for the farfield, as described in [121]. The farfield in a physical domain consists of two flow situations, the flow entering the domain referred to as the inflow and the flow leaving the domain which is referred to as the outflow.

The *inflow* boundary condition for inviscid flows has four boundary conditions required to fully define the system of equations since one characteristic is leaving the computational domain. In contrast with the inviscid flow, all the five boundary conditions are specified in viscous flows at the inflow. Moreover, the viscosity $\tilde{\nu}$ in the SA turbulence model is set to ten percent of the laminar viscosity at the inflow. In the *outflow* boundary condition, four flow variables, that is, density and the three velocity components, are extrapolated from the interior of the domain. The fifth variable, pressure, must be specified externally for inviscid flows. This results in only one boundary condition at the outflow. Conversely, four boundary conditions are required in viscous flows due to the inclusion of viscous terms in the governing equations. It is worth noting that the solution in viscous flows where the boundary of the physical domain is found far away from regions of significant gradients, behaves as though it is governed by the Euler equations [94]. On the other hand, for problems where the boundary is located near regions of significant gradients, such as in internal flows, the application of boundary conditions can be challenging. In this work, only external flows are considered and inviscid boundary conditions are employed at the farfield. The turbulent viscosity from the turbulence model is also extrapolated from the interior nodes.

2.2.4 Computation of the aerodynamic coefficients

The calculations of the lift, drag and moment coefficients for a given configuration are written as

$$C_L = \frac{1}{q_\infty S_\Gamma} \int_\Gamma (\boldsymbol{\sigma}^w \cdot \boldsymbol{n}^\infty) \, \mathrm{d}\Gamma, \tag{2.21}$$

$$C_D = \frac{1}{q_\infty S_\Gamma} \int_\Gamma (\boldsymbol{\sigma}^w \cdot \boldsymbol{t}^\infty) \, \mathrm{d}\Gamma, \tag{2.22}$$

$$\boldsymbol{C_M} = \frac{1}{q_\infty S_\Gamma l} \int_\Gamma (\boldsymbol{r} \times \boldsymbol{\sigma}^w) \, \mathrm{d}\Gamma, \tag{2.23}$$

where Γ is the integration surface, S_Γ is the reference area, taken to be the chord length in two dimensions or the planform area in three dimensions, q_∞ is the free-stream dynamic pressure, $\boldsymbol{t}^\infty$ and $\boldsymbol{n}^\infty$ are the tangential and normal unit vectors with respect to the direction of the free-stream velocity. $\boldsymbol{\sigma}^w$ is the wall stress vector and is given by

$$\boldsymbol{\sigma}^w := (-p\mathbf{I}_d + \mathbf{T}) \cdot \boldsymbol{n}^w, \tag{2.24}$$

where $\boldsymbol{n}^w$ denotes the outward unit normal vector to the wall surface.

The vector obtained in equation 2.23 represents the moment coefficients in the x, y and z directions. In this work, only the pitching moment is reported, corresponding to the moment about the y-axis in three dimensions. A length scale, l, is introduced and this is usually taken to be the chordlength of the aerofoil in two dimensions or the chordlength of the mid-span aerofoil cross section of the wing in three dimensions. The vector $\boldsymbol{r}$ denotes the position of a point, $\boldsymbol{x}$, on Γ with respect to a reference point $\boldsymbol{x}_{\mathtt{ref}}$, namely $\boldsymbol{r} = \boldsymbol{x} - \boldsymbol{x}_{\mathtt{ref}}$. The reference point is taken as the aerodynamic centre in two dimensions or the centre of gravity in three dimensions, unless stated otherwise.

2.3 Discretisation procedure

This section considers the construction of a finite set of discrete equations to approximate the NS equations. The section begins by providing a brief discussion on the types of domain discretisation procedures in fluid problems, followed by the mesh generation procedure used in this work. Next, a brief comparison of finite volume methods is presented. The formulation of the vertex-centred finite volume methods relevant to the discretisation used in this work is described for inviscid and viscous flow problems.

2.3.1 Domain discretisation

A key requirement in numerical simulation is the discretisation of the domain, where the geometry is discretised into $\mathtt{n_{el}}$ elements. The domain is generally discretised using triangles, quadrilaterals, hexahedrals or a combination of the three. Also referred to as the mesh generation procedure, it can be described as the process of breaking up the physical domain Ω into $\mathtt{n_{el}}$ sub-domains Ω_e in order to facilitate the computation of numerical solution of a partial differential equation. This can be mathematically represented as

$$\Omega = \bigcup_{e=1}^{\mathtt{n_{el}}} \Omega_e, \quad \Omega_e \cap \Omega_f = \emptyset, \quad \text{for} \quad e \neq f. \tag{2.25}$$

There are two main classes of mesh generation techniques, the structured and unstructured, respectively. The difference between the two is the valence of an internal mesh node, that is, the number of connectivities that an internal node has. In a structured mesh, the valence of an arbitrary node is constant, while in an unstructured mesh, the valence is not constant. Additionally, unstructured meshes require the construction of an element connectivity table which relate the nodes to each element.

The development of structured mesh generation started with the construction of simple quadrilaterals using algebraic methods or partial differential equations [107]. However, with the increasing complexity of geometries that needed to be simulated, the meshes had to be divided into a number of topologically simpler blocks, also known as the multi-block approach. Although there has been significant improvement in generation of structured meshes using the multi-block approach, it is known to be

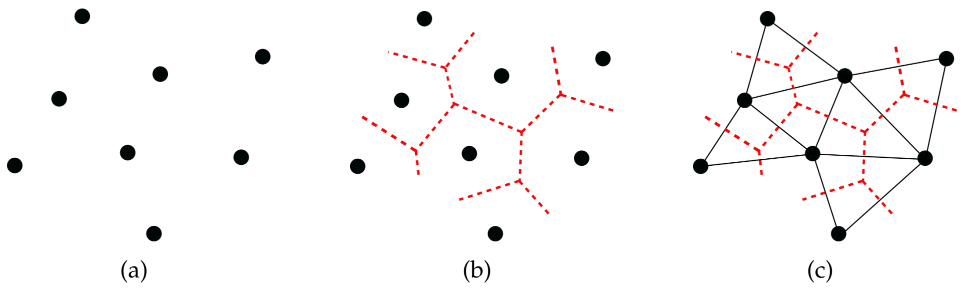


Figure 2.3: Illustration of the Delaunay scheme. The solid circle denotes the initial set of nodes and the dashed line is the Dirichlet tessellation. The solid line represents the edges constructed from the Delaunay triangulations.

time-consuming. This is attributed to the fact that setting up the connectivities at the interfaces of the blocks and ensuring such connections do not create overlapping elements can be a challenge [122]. Moreover, there are reasonable doubts about the possibility of ensuring a low cell skewness, a smooth point distribution and an adequate grid resolution in regions of interest [123]. In addition, it will be difficult to automate the structured mesh generation. The motivation to overcome these issues has led to the use of unstructured meshes comprising of triangles in two dimensions and tetrahedra in three dimensions [124]. The mesh can be adapted to irregular boundary and a smooth transition of the size of the elements can be achieved with minimal or zero distortion [124]. The solutions of Euler equations on complex geometries have efficiently been obtained using unstructured meshes [94, 125, 93, 126, 88].

In this thesis, unstructured meshes are generated due to its adaptive capabilities and relatively inexpensive mesh generation procedure. Moreover, it is a fully automated process with minimal or zero user interaction and has the ability to handle complex geometries. There are various ways to obtain an unstructured mesh. This thesis focuses on two methods, the Delaunay and advancing front methods, respectively. The two differ from each other in the way that the points and elements are introduced in the computational domain.

The Delaunay method [127] defines an arbitrary set of nodes in the computational domain to form a valid mesh consisting of triangles in two dimensions and tetrahedra in three dimensions [128]. As this method is not well-defined for closed domains, it requires recovery methods to ensure the validity of the interface at the boundary and internal mesh [94]. Figure 2.3 shows an illustration of the Delaunay scheme in two dimensions. The starting point of the Delaunay scheme is an initial set of nodes $\boldsymbol{p}_i \in \Re^d$, $i \in [1, N]$, as in figure 2.3(a). The Dirichlet tesellation is constructed by assigning a territory to each point, as shown in figure 2.3(b). The tessellation of a closed domain would result in a set of non-overlapping convex polygons known as the Voronoi regions [129], i.e., $\boldsymbol{V}_i \subset \Re^d$. Hence the region $\boldsymbol{V}_i$ is defined as a subset of the computational domain that is closer to node $\boldsymbol{p}_i$ than any other node in the domain. The territorial boundary, part of the side of the Voronoi polygon, must be midway due to construction of the perpendicular bisector from the line joining two nodes. The Delaunay triangulation now consists of joining all the nodes that have a common Voronoi face. Figure 2.3(c) illustrates the resulting triangulations. It is worth noting that each Delaunay triangle contains a unique vertex of a Voronoi diagram and there is no other vertex found within the circle centered at this vertex. This may however be an impractical approach to generate the triangulation efficiently. The approach used for the meshes generated in this work employs a sequential process of introducing new mesh nodes into the existing structure. This is followed by a check on the validity of the new Delaunay triangulation using an in-circle criterion. The criterion states that for the triangulation to be valid, the circle passing through the points in that triangle cannot contain any node and must have its centre at a Voronoi vertex. When the new triangulation violates this criterion, the element is split and a local Delaunay procedure is performed. One of the major advantage of Delaunay triangulations over other types is that it maximises the minimum angles of the triangles. This tends towards equiangularity in the triangles and therefore avoids triangles which are disproportional. Additionally, the triangulation is also independent of the order the points are processed.

In the advancing front scheme [130], the meshes are built from the computational boundaries [131]. The physical domain is divided into two parts at any given time of the process, mainly, regions where meshing is complete and regions where meshing has not been performed yet. The scheme is said to be local in nature due to this property and this results in a very memory efficient scheme with the ability to generate meshes of high quality [132]. Figure 2.4 provides an illustration of the advancing front scheme. A brief description of the advancing front scheme in two dimensions follows. The scheme utilises the concept of a generation front which uses a dynamic data structure that changes continuously during the procedure. The algorithm begins by discretising the boundary into N segments and is defined as the initial front, as shown in figure 2.4(a). The nodes forming the front are added to a list and are termed as *active*. The scheme finds the shortest segment on the boundary and introduces a new

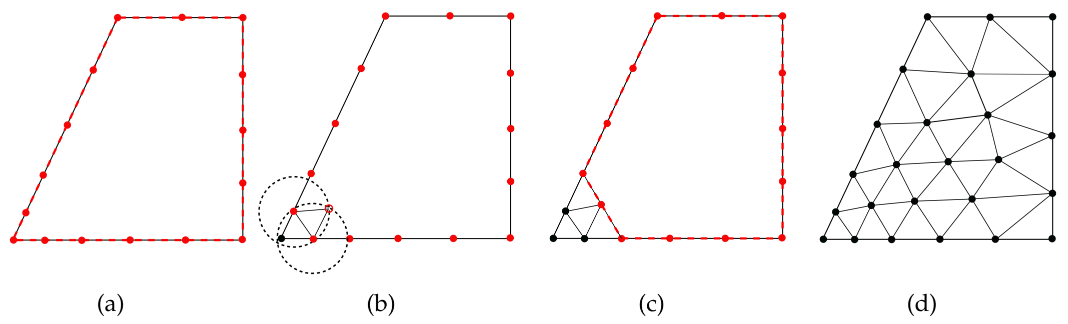


Figure 2.4: Illustration of the advancing front scheme. The generation front is represented using the red dashed line and solid circles, and the open circle denotes the newly inserted node. The solid black lines and circles represent the edges and nodes.

point into the unmeshed region. The position of this new point is determined from the desired point distribution, as described in figure 2.4(b). A check is performed to determine if any of the existing points from the front is within a specified tolerance from the ideal position. If the new point satisfies this check, the validity of the element is confirmed before adding the new point to the dynamic list of nodes forming the front. Figure 2.4(c) shows the new front once it has been added to the list. Conversely, if there are points within the specified tolerance, the closest of these points that forms a valid triangle, is selected. The procedure stops when the list containing the segments becomes empty. Figure 2.4(d) shows when the advancing front scheme is completed. In addition to being memory efficient, the advancing front scheme has the advantage of preserving boundary integrity and the capacity to create clusters of high aspect-ratio triangles in boundary layer regions.

In this work, the unstructured mesh procedure for the inviscid cases in two dimensions employs the advancing front scheme, while the mesh generator in three dimensions takes advantage of both the advancing front scheme and the Delaunay method. It can be summarised as follows:

1. The advancing front scheme is used to generate triangles on the surface segments. The meshing procedure are performed in the parametric space in two dimensions using the user-specified spacing.

2. An initial Delaunay triangulation is performed by connecting the nodes from the surface triangulation.

3. A boundary recovery technique is employed to ensure the validity of the interface between the volume and surface meshes. The elements found outside the

computational domain are deleted.

4. One node is introduced in the interior mesh at a time and a tetrahedral element is formed by creating connections with the newly inserted node using the Delaunay algorithm. The node is positioned according to a specified background mesh and user-defined sources.

5. The mesh quality is controlled by employing mesh cosmetic techniques such as diagonal swapping, element collapsing and Laplacian mesh smoothing.

More information on the unstructured mesh generation procedure for inviscid flows, part of the solver, can be found in [88].

The viscous terms in the NS equations complicates the domain discretisation as the normal gradients at the wall are considerably larger in high Reynolds number flows. As a result, a dense mesh is required in this direction to capture the flow features accurately in those regions. A larger spacing can be used in the tangential direction along the wall as the gradients along this direction are much smaller in magnitude. This would result in stretched elements along the boundary. In this work, an *unstructured hybrid mesh* is used to obtain solutions of viscous flow problems. The mesh generator employs an advancing layer method that creates meshes of structured appearance in the normal direction of the viscous wall. This regularity is exploited in the generation of the hybrid meshes where edges are removed in the boundary layers by merging elements [133]. According to the authors in [94, 133], this approach reduces the global number of edges by 30-45% of a given tetrahedral mesh. This becomes specially advantageous when the numerical discretisation is formulated over the edges in the mesh. Additionally, the diagonals of simplex meshes in the boundary layers have adverse effects on the accuracy of the solutions [133]. It is hence desirable to eliminate the diagonal edges in the boundary layers.

In the current work, viscous problems in two dimensions are considered and a brief overview of the hybrid mesh generation procedure follows. The procedure begins by discretising the solid wall boundaries using the spacings specified by the user via a mesh control function. The approach proceeds in a similar way to the advancing front scheme used in inviscid meshes. The advancing front technique generates a triangular mesh of double the desired element size. Quadrilaterals are formed by an indirect method of combining triangles coupled with a splitting scheme [133]. The advancing

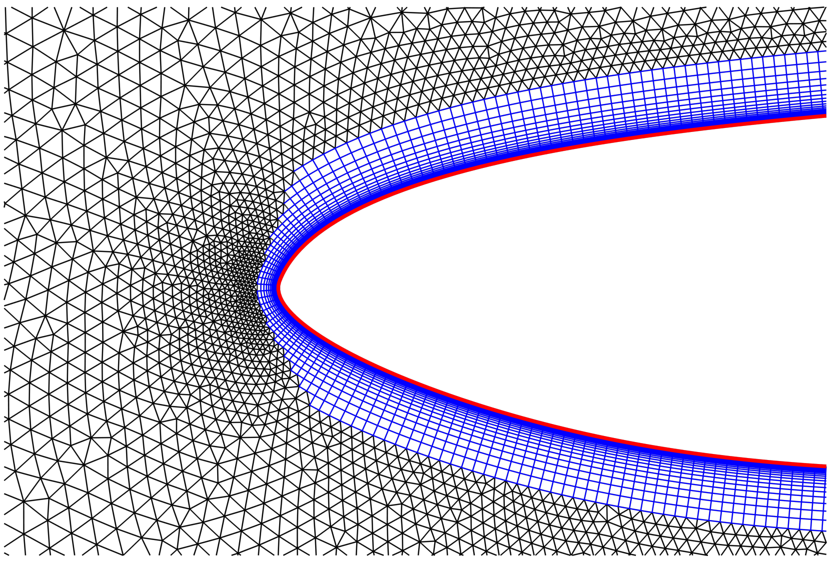


Figure 2.5: Example of a hybrid mesh in two dimensions.

layer method is employed to generate stretched elements adjacent to the boundary wall surface [133]. The height and number of layers are specified by the user. The layers are constructed by generating points along prescribed lines using advancing front mesh generation scheme and triangles are formed by connecting the generated points. For the first layer, the prescribed lines are normal to the surface, while for succeeding layers, smoothing of the line directions is performed. The generation of points stops before the prescribed number of layers is reached if an intersection appears or if the local mesh size is close to that prescribed user-defined mesh size. Once the viscous layers are discretised, the diagonals are deleted and the remaining part of the domain is meshed using the standard advancing front scheme, followed by mesh cosmetic techniques, as described in the inviscid mesh generation. Figure 2.5 shows an example of a hybrid mesh generated around an aerofoil geometry in two dimensions. More information on the generation of hybrid meshes used in this thesis can be found in [94, 133].

2.3.2 The vertex centred finite volume method

The finite volume (FV) method is a numerical technique that transforms the partial differential equations representing the conservation laws over differential volumes into discrete algebraic equations over discrete volumes. Due to the direct discretisation of the integral formulation into the physical space and the inherent conservative properties of the FV method, many industrial applications have made this approach their preferred choice [93, 94, 134]. Similar to the finite difference or the finite element method, the first step involves the discretisation of the computational domain in the form of a grid or mesh to create the discrete elements. In the FV method, it is required that the computational domain is discretised into non-overlapping finite volumes, referred to as control volumes. Next, the partial differential equations are discretised into algebraic relations by integrating over each control volume using quadrature rules of the desired accuracy. Lastly, a system of algebraic equations is obtained by choosing suitable interpolation profiles for approximating the values on the surface of the control volumes based on the values of the discrete control volumes of the conservative variables.

There are two main types of FV schemes, the vertex-based and the element-based. The difference between the two types is found in the construction of the control volumes. In the vertex-based method, the discrete flow variables are stored at the vertices and each control volume is constructed to have only one node. This virtual element connected by virtual edges is known as the dual mesh. It is constructed using the *median dual principle*, which involves the connection of the edge centres, face and the volume centroid of the elements. Moreover, the vertex-based schemes integrate the fluxes over the dual mesh, usually by looping over the edges of the actual mesh. On the other hand, element-based schemes do not require the construction of the dual mesh and the elements in the mesh are defined as the control volumes. The discrete dependent variables are stored within the element, usually at the centroid, and the fluxes are integrated over the edges in two dimensions and faces in three dimensions. This is known as the cell-centred FV method. Figure 2.6 illustrates the vertex-based and cell-based FV schemes using triangular elements in a two dimensional interior mesh.

It is worth noting that the number of calculations required on a given grid does not differ significantly between the two types of FVs schemes for problems in two dimensions. This is because both schemes loop over the edges. However, in three dimensions, the cell-based approach is computationally more expensive as the fluxes are integrated over the faces. In tetrahedral meshes, the ratio of the number of faces to the number of edges is approximately 1.7. Moreover, the number of elements is about 5.5 times the number of nodes. Hence, the vertex-based schemes prove to be more efficient in terms of storage requirements [135].

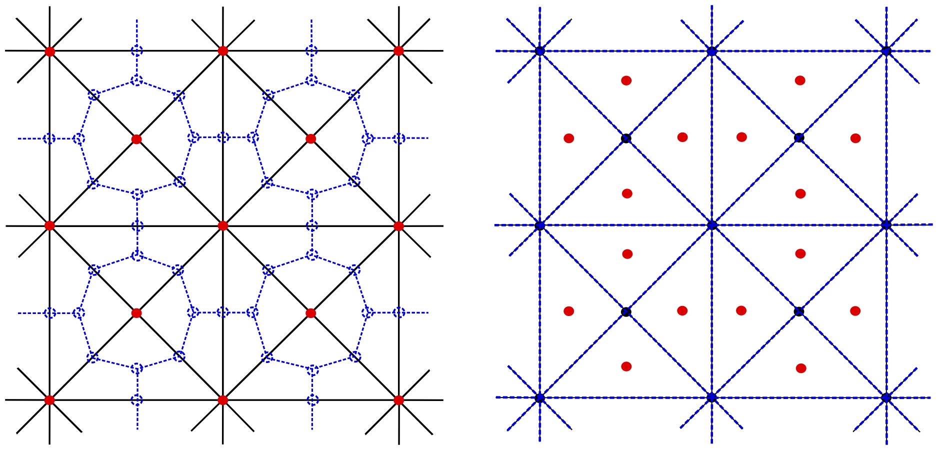


Figure 2.6: Illustration of two FV schemes in an interior mesh. The solid black lines and circles denote the edges and nodes in the original mesh. The control volume definition is represented using the blue dashed line and the location where the unknowns are stored is identified by the solid red circles.

The vertex-centred FV, part of the vertex-based family, has been extensively applied to Euler and NS equations as an efficient discretisation technique for steady flows [93, 136, 137]. A vertex-centred approach was shown to be equivalent to a linear finite element method [138]. Moreover, the vertex-centred FV method is known to ensure a minimum number of degrees of freedom in the vicinity of the aerodynamic shape when compared to other FV schemes such as the cell-centred or face-centred approach [139]. Additionally, the vertex-centred approach performs better than the element-based counterparts in regions of high distortion and anisotropy in the mesh [140]. This is attributed to the fact that the unknowns are stored on the boundary of the computational domain itself. As a result, this does not represent a complication when boundary conditions are applied, compared to element-based schemes.

A few nomenclatures related to the data structure of edge-based formulation in vertex-centred FV scheme are introduced. Let Ω_i be the control volume of an arbitrary node i in the FV domain discretised using $\mathtt{n_{cv}}$ mesh nodes. A set of facets $\{\Upsilon_i\}$ of size $\mathtt{n_f}$ that defines the boundary of the control volume $\partial\Omega_i$ is written as

$$\partial\Omega_i = \bigcup_{j=1}^{\mathtt{n_f}} \Upsilon_{i,j}. \tag{2.26}$$

In two dimensions, straight edge facets are formed by connecting the edge centre and element centroid, while in three dimensions, triangular facets are constructed by connecting the edge centre, face centroid and element centroid. For a mesh node lying on the boundary, the set of facets $\{\Upsilon_i^\Gamma\}$ contains facets constructed from nodes lying on the boundary, that is, $\Upsilon_i^\Gamma \in \partial\Omega$.

The *semi-discrete form* of the steady-state *NS equations* results in a non-linear system of equations, obtained by the assembly of each nodal contribution $\boldsymbol{R}_i$ as

$$\begin{aligned} \boldsymbol{R}_i := & \sum_{j\in\Lambda_i} C_l^{i,j}\tilde{\boldsymbol{F}}_l^{i,j} + \sum_{j\in\Lambda_i^\Gamma} C_l^{i,j,\Gamma}\tilde{\boldsymbol{F}}_l^{i,\Gamma} \\ & - \sum_{j\in\Lambda_i} C_l^{i,j}\tilde{\boldsymbol{G}}_l^{i,j} - \sum_{j\in\Lambda_i^\Gamma} C_l^{i,j,\Gamma}\tilde{\boldsymbol{G}}_l^{i,\Gamma} \\ & + \sum_{j\in\Lambda_i} \mathrm{m}_{i,j}\varphi_{i,j}(\mathbf{E}_j - \mathbf{E}_i) - \sum_{j\in\Lambda_i} \epsilon_2\varphi_{i,j}s_{i,j}(\boldsymbol{U}_j - \boldsymbol{U}_i), \end{aligned} \tag{2.27}$$

for $i = 1, 2, ..., \mathrm{n_{cv}}$, where Λ_i denotes the set of nodes connected to an arbitrary node i through an edge and, $C_l^{i,j}$ and $C_l^{i,j,\Gamma}$ are the flux coefficients of an interior edge and a boundary edge, respectively. The coefficients are given by

$$C_l^{i,j} = \sum_{\Upsilon_i\in\Upsilon_{i,j}} |\Upsilon_i| n_l^{\Upsilon_i}, \quad C_l^{i,j,\Gamma} = \sum_{\Upsilon_i\in\Upsilon_{i,j}^\Gamma} |\Upsilon_i| n_l^{\Upsilon_i}, \tag{2.28}$$

where $|\Upsilon_i|$ denotes the area of the facets Υ_i and $n_l^{\Upsilon_i}$ is the outward unit normal vector of the facet from node i. The first two terms of equation (2.27) correspond to the treatment of the inviscid fluxes. The discrete fluxes $\boldsymbol{F}$ at the interior and boundary nodes are expressed in terms of the discrete conservative variable $\boldsymbol{U}$ by using a *second-order* central difference scheme. Here, the trapezoidal rule across an edge is used and is given by

$$\tilde{\boldsymbol{F}}_l^{i,j} = \left(\frac{\boldsymbol{F}_l(\boldsymbol{U}_i) + \boldsymbol{F}_l(\boldsymbol{U}_j)}{2}\right), \quad \tilde{\boldsymbol{F}}_l^{i,j,\Gamma} = \frac{3\boldsymbol{F}_l(\boldsymbol{U}_i) + \boldsymbol{F}_l(\boldsymbol{U}_j)}{4}. \tag{2.29}$$

This ensures a second-order convergence of the method. The third and fourth terms of equation (2.27) corresponds to the viscous tensors for an interior node and boundary node, respectively. The solver uses an edge-based formulation with a compact stencil to discretise the velocity gradients which provides second-order accurate solutions. This compact stencil splits the evaluation of the derivatives into the tangential and normal components at an edge midpoint using flow variables computed at only two nodes. It is performed using finite difference. Moreover, the solver uses a first order upwind scheme to discretise the convective terms from the SA model when turbulent flows are modelled around the aerodynamic components. It avoids the numerical instabilities in the scheme. Any other terms from the viscous tensors are discretised using the standard finite volume procedure.

The central difference scheme described is known to be unstable for equations of hyperbolic nature. In order to stabilise such schemes, dissipation is usually employed to damp oscillations originating from the convective terms. The fifth term corresponds to a third order biharmonic dissipation term that is employed in the CFD model used in this thesis [88]. The scaled artificial dissipation flux $\mathbf{E}(\boldsymbol{U})$ at node i is expressed in terms of the conservative variables as

$$\mathbf{E}_i = \frac{1}{\sum_{j\in\Lambda_i} l_{i,j}^{-1}} \sum_{j\in\Lambda_i} l_{i,j}^{-1}(\boldsymbol{U}_j - \boldsymbol{U}_i), \tag{2.30}$$

where $l_{i,j}$ is the length of the edge connecting node i and j. This scaling is performed so that the artificial dissipation scheme is close to the discrete fourth-order differences. The coefficients $\mathrm{m}_{i,j}$ and $\varphi_{i,j}$ of the bi-harmonic term are written as,

$$\mathrm{m}_{i,j} = \max(0, \epsilon_4 - \epsilon_2 s_{i,j}), \quad \varphi_{i,j} = \frac{1}{|\Lambda_i| + |\Lambda_j|} \min\left(\frac{|\Omega_i|}{\Delta\tau_i}, \frac{|\Omega_j|}{\Delta\tau_j}\right), \tag{2.31}$$

where ϵ_2 and ϵ_4 are user-defined constants known as the biharmonic dissipation factors and taken to be 0.2 in this work. $|\Lambda_i|$ denotes the number of edges connected to node i and $\Delta\tau_i$ is the size of the local time step. The biharmonic term however, does not smooth out oscillations in regions of high gradients and therefore the scheme requires a local addition of harmonic dissipation which renders the scheme to be first order locally. This harmonic term corresponds to the sixth term in equation (2.27). It only contributes significantly to high pressure gradients due to presence of a pressure sensor $s_{i,j}$, given by,

$$s_{i,j} = \max(|\Delta p_i|, |\Delta p_j|), \qquad \Delta p_i = 12\frac{\sum\limits_{k\in\Lambda_i}(p_k - p_i)}{\sum\limits_{k\in\Lambda_i}(p_k + p_i)}, \tag{2.32}$$

where p_i is the pressure at an arbitrary node i. The pressure sensors are activated in regions of high gradients such as shocks in the flow. The third-order dissipative term is suppressed since it causes perturbations near shocks when it is activated. This is performed using the second-order difference term to obtain first-order accuracy locally.

2.4 Solution procedure

The unknowns in the set of discrete equations described in equations (2.27) are determined by a solution procedure. There are several ways to obtain a solution and the procedures can be broadly categorised into two main groups, the implicit and explicit solution schemes. In the context of implicit schemes, the procedure requires non-trivial matrix inversions directly or iteratively. This may become cumbersome when large meshes with tens of millions of unknowns are used. On the other hand, explicit schemes do not require the storage and inversion of matrices. The updated solution is explicitly given as a function of known entities through localised vector products and summations. Explicit schemes are naturally iterative and require many more iterations than its counterpart to achieve convergence. Two distinct advantages of the explicit schemes are the reduced storage requirements and reduced difficulty to parallelise the code.

Albeit the many choices available in the literature to obtain a solution explicitly [141, 142], the multi-stage Runge-Kutta time stepping scheme, described in [143], is used in the solver. The mathematical representation of the three stage Runge-Kutta method to obtain the explicit solution of the ordinary differential equation of the form, $|\Omega_i|\boldsymbol{U}_{t,i} + \boldsymbol{R}_i(\boldsymbol{U}_i) = \boldsymbol{0}$, can be written as

$$\begin{aligned}
\boldsymbol{U}_i^{r_0} &= \boldsymbol{U}_i^n, \\
\boldsymbol{U}_i^{r_1} &= \boldsymbol{U}_i^{r_0} - \vartheta_1 \text{CFL}\ \Delta\tau_i \boldsymbol{R}_i(\boldsymbol{U}_i^{r_0}) \\
\boldsymbol{U}_i^{r_2} &= \boldsymbol{U}_i^{r_1} - \vartheta_2 \text{CFL}\ \Delta\tau_i \boldsymbol{R}_i(\boldsymbol{U}_i^{r_1}) \\
\boldsymbol{U}_i^{n+1} &= \boldsymbol{U}_i^{r_2} - \vartheta_3 \text{CFL}\ \Delta\tau_i \boldsymbol{R}_i(\boldsymbol{U}_i^{r_2}),
\end{aligned} \tag{2.33}$$

for $i = 1, 2, ..., \mathtt{n_{cv}}$. In the above expression, the $\boldsymbol{U}_i^n$ denote the unknowns at iteration n, $\boldsymbol{U}_i^{r_1}$ are the unknowns at Runge-Kutta iteration r and sub-iteration 1 of the three stage scheme. $\Delta\tau_i$ is the local time-step and the coefficients ϑ are selected as 0.6, 0.6 and 1.0 [141]. The maximum allowable CFL number for stability is usually limited by the number of stages employed in the scheme [144].

One major drawback of explicit scheme is that several iterations are required for the correction of an arbitrary node to influence the computational domain. This is attributed to the fact that the domain of influence of an arbitrary node is dictated by the local stencil of the discretisation scheme and the number of stages involved in the explicit approach [141]. As a result, explicit schemes are not efficient in viscous flows of high Reynolds numbers which are of hyperbolic nature outside the boundary layers and have low error frequencies. The solver uses the multigrid method [141, 94, 144] to effectively dampen the errors of all frequencies at once while preserving a low operation count per iteration and low memory usage of the schemes. There are two main types of multigrid schemes, the algebraic and geometric scheme. The algebraic approach linearises the non-linear system of equations and generates several smaller matrix systems that are successively added. The effect of this allows the domain of influence of a mesh node to span to a larger part of the computational domain. However, this scheme has the complications of computing and storing the Jacobian matrix, which is not favourable when larger systems of equations are solved. On the other hand, the geometric approach consists of employing meshes at various coarseness levels to solve the discrete equations. Each level of mesh refinement removes different intervals in the frequency of the errors and the coarser meshes dampen the lower frequency errors. Similar to the algebraic scheme, as the spacing in the domain is increased, the domain of influence at an arbitrary node is also increased. Therefore, the explicit relaxation scheme on a coarse mesh smooths out effectively the low error frequencies which slows down convergence on the fine mesh. Moreover, the computational cost per iteration is considerably lower on a coarse mesh than a fine mesh. The solver uses the FAS multigrid scheme which is capable of handling non-linearities. More information on the implementation of the FAS scheme is available in [94].

Chapter 3

Reduced Order Models

This chapter aims at discussing ROMs which are commonly employed in CFD applications. Two concepts that are used as a ROM in this work are introduced. Firstly, the popular proper orthogonal decomposition coupled with the radial basis functions is formulated to use in a regression setting. Secondly, the NN is reviewed with the forward propagation and backpropagation detailed in steps. Moreover, several optimisation algorithms adapted to NNs are also considered. Lastly, a brief overview of various design of experiment techniques is presented.

3.1 Introduction

Despite the huge progress attained by current CFD simulation capabilities, a number of problems remains challenging. Numerical simulations of turbulent flows, flow separation, high-lift devices and other multi-physics problems involving complex geometries are still computationally demanding. The number of simulations required becomes unaffordable when designers have to perform simulations for a range of model parameters such as geometric parameters or flow conditions.

The use of ROMs has become a popular alternative to alleviate the computational burden associated with CFD simulations involving a large number of parameters [5]. This allows users to render full order simulations to become affordable by reducing the CPU time and memory requirements. Some of the more popular ROMs are the POD [10, 11, 12, 13], the reduced basis method [14, 5, 15], the eigensystem realisation algorithm [9, 16, 17] and the PGD [145, 146, 147].

The POD is a general technique for extracting the most significant energies of the behaviour of a system using a set of POD basis vectors that reflect a lower dimensional representation of the real system [39]. The application of POD as a data-driven technique in CFD can be found in [10, 11, 12, 13]. The reduced basis method is another family of ROMs which requires the use of a greedy algorithm and an *a posteriori* error function to construct the reduced bases. This method has an offline stage where the reduced basis model is constructed on a given set of data, followed by an online stage for solving different model parameters with control on the accuracy of the model. When the error becomes unacceptable, a greedy adaption strategy is established to enrich the reduced bases [5, 14, 15]. The eigensystem realisation algorithm is a linear ROM which is part of the system identification family [148]. The method is usually employed in modal analysis to construct a state-space dynamic model for modern control design. The eigensystem realisation algorithm has seen applications in model reduction of dynamic systems, for instance, in gust problems [17, 9]. See [148] for more information.

A more recent ROM, the PGD, is based on the separation of variables when solving a set of partial differential equations analytically [145, 146, 147]. The origin of the PGD can be traced back to the 1980s [149], where it was used to construct the space-time separated representations of transient solutions involved in strongly non-linear models. The concept of the PGD became popular when the original idea was extended to consider parameters as extra coordinates [149]. The application of the PGD can be found in optimisation, inverse analysis and simulation-based control [149]. Although non-intrusive approaches exist [150, 151, 21], the applicability of the PGD is often limited by the number of parameters to be considered. The limitations of the PGD is shown in the work of [152], by considering a time-dependent example with a non-separable solution. The authors report that the convergence of the error of the PGD relative to the reference known solution tends to decrease when the assumption of the separation of variables cannot respect the problem to solve [152]. Moreover, the authors speculate that parameter dependent problems may have the same behaviour [152].

The non-linear normal modes method is one of the techniques that are rarely employed in fluids and have seen more applications in the field of structural dynamics. The idea behind this method lies in the construction of the ROM using specific properties of the dynamic system and by adapting the mathematical reduction techniques of the centre manifold theorem and normal form theory into the governing equations [153]. The authors in [153] compare the POD with the non-linear normal mode technique, and report that the only advantage that the latter has over the POD is that it is non-linear. As a result, fewer modes can be utilised to capture the non-linearities in the system and therefore lower computational cost. However, given that non-intrusive empirical models are selected for the examples considered in this work, this technique is not employed. Next, the centroid voronoi tessellation method in the context of ROM is a data compression technique which considers a set of snapshots as in the POD [154]. The voronoi tessellation takes place in the higher dimensional space, spanned by the snapshot vectors, to determine the generators of the centroid voronoi tessellation method which constitute the reduced bases. A low dimensional representation of the complex system can then be obtained in a similar way as that obtained with the POD [154]. The authors in [154] compare the centroid voronoi tessellation method with the POD for an incompressible viscous flow problem, and report that neither technique proves to have an advantage over the other. This remark was based on the computational cost and errors of both techniques and was found to be of the same order. However, as it is less well tested in the literature and is still currently an active area of research, this approach is not considered in this work.

Machine learning techniques can also be employed as a data-driven ROM that improves automatically through experience. This field which lies at the intersection of computer science and statistics, has seen major advances over the last two decades with development of new algorithms and theories [155]. Machine learning has rapidly evolved in many areas of science and engineering, leading to more evidence-based decision making fields such as health care, manufacturing, education, financial modelling and marketing [156, 157, 158, 159, 160]. There are two dominant machine learning paradigms based on the extent that the training involves labelled data, mainly the *supervised* and *unsupervised learning*. In addition to the two dominant paradigms, reinforcement learning forms another type of learning which involves goal-oriented interactions of an agent with its environment for solving problems [182]. A classification of main machine learning techniques organised by tasks is shown in figure 3.1. The *semi-supervised* learning, a hybrid to the two main paradigms, is a type of machine learning technique which represents learning using labelled data as well as unlabelled data [161].

Four main stages can be described when employing a machine learning algorithm.

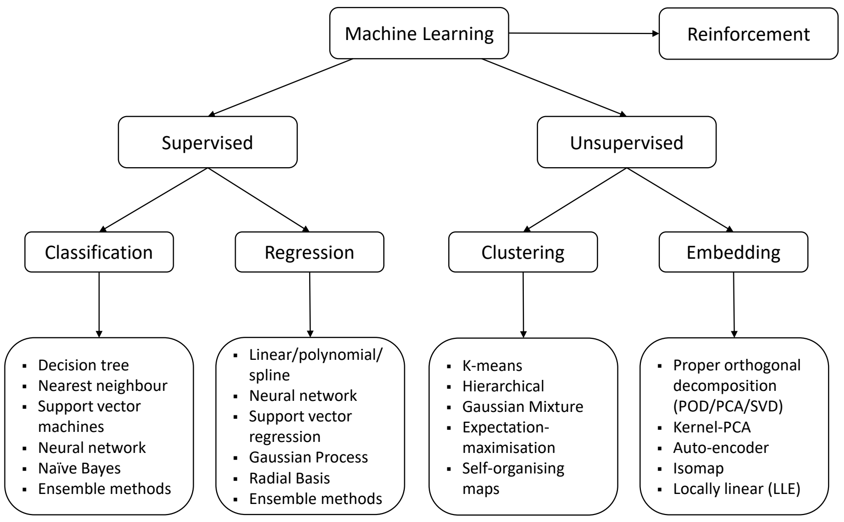


Figure 3.1: A classification of the main machine learning paradigms.

The first stage defines an objective. The datasets are then collected and curated in the second stage. Next, an appropriate model architecture and parametrisation are identified in the third stage. Lastly, an optimisation algorithm is chosen to obtain the parameters of the model. Human intelligence is required in each stage. Although considerable attention is usually given to the selection of various models, it is often the data collection and optimisation stages that are the main factors contributing to the computational cost and time. As a matter of fact, exploring a broad and diverse set of machine learning architectures as a *black-box* model is a hallmark feature of data-centric companies nowadays [4]. It is also worth noting that known physics such as invariances, symmetries, conservation laws and constraints may be incorporated in any of the stages [162].

In *supervised learning*, it is assumed that the training data has both the inputs, $\boldsymbol{x}$ and the target outputs, $\boldsymbol{y}$. If the targets are *categorical*, such as the categorical representation of images, then the learning task is referred to as *classification*. On the other hand, if the targets are *continuous*, then the task is referred as *regression*. The goal of the supervised algorithm is to train a model to minimise a cost function.

The predominantly used learning for real life applications are usually supervised and they are capable of automating complex tasks when the data is labelled with expert knowledge [4]. Some examples of supervised learning are the classification trees, random forests, support vector machines and NNs [163, 164, 165]. Classification trees and random forests fall under the tree algorithm paradigm and is the most popular algorithm in data mining. However, the authors in [166] underline that decision tree algorithms can be unstable, that is, a small change in the data may lead to a large change in the outputs of the system. It is worth noting that this is not Next, support vector machines have the ability to solve complex problems. However, as the size of the dataset or the dimensionality of the data is increased, long training time must be expected [166, 167]. Moreover, the authors in [167] underline that a concern of using support vector machines is to find a suitable kernel function which can sophistically represent the data in the Hilbert space. Additionally each kernel function comes with their respective hyperparameters that need to be tuned during the training [167]. The extensive use of these shallow machine learning techniques over the years has led to rigorous mathematical analyses in the literature. With the rise in computational capabilities, specialised hardware for matrix operations and the availability of larger datasets, NNs have gained considerable attention in both academia and the industry over the last decade [168]. NNs are said to surpass the performance of traditional methods, provided sufficient data is used in the training [168]. Moreover, the universal approximation property of the NN has been shown in the work of [169]. The primary disadvantage of NN is the hyperparameters, which are problem dependent and need to be fine-tuned during the training. This can consequently render the training stage to be time-consuming. However, fine-tuning hyperparameters of the NN model during the training stage is not unlike most other machine learning methods. More information on NN is provided in section 3.3. Researchers have also employed pre-trained models to partly alleviate the computational burden of optimising the weights from initial condition. This is referred to as *transfer learning* [170] in the literature and is a way of improving the learning of a new task through the transfer of knowledge from a related task that has been learnt in the pre-trained model. It has been proved to be beneficial in deep NN architectures where millions of weights are optimised [171, 170].

Unsupervised learning establishes the underlying structure of a dataset without a given set of target outputs. It is also referred to as data mining, pattern extraction or feature extraction across various fields. Two common classifications of this type of learning are *clustering*, where the task is to group data into distinct categories and *embedding*, where the task is to find a low-dimensional representation of a continuous distribution. It is tedious to perform such a task given that the algorithm is not guided by the targets. As a result, these algorithms seek for any kind of correlation that exists between the features of the dataset in an exploratory fashion and this new information is often used in future developments that are supervised.

The most common unsupervised clustering methods include k-means, hierarchical clustering and more recently, self-organising maps [164, 165, 172]. In each case, the user has to specify the number of distinct clusters to find in the data. This can also be considered as a numerical parameter in the training stage.

Unsupervised embedding is a technique that finds a low-dimensional representation of the input data and it is also referred as a dimension reduction technique. Such representation can be a linear or non-linear transformation to a low-dimensional subspace, parametrised by a latent variable v, which describes the original high-dimensional state x. For instance, the goal is to find two transformation functions, an encoder $v = \psi(x)$ and a decoder $\tilde{x} \approx \varphi(v)$ such that, $\tilde{x} = \varphi(\psi(x)) \approx x$. The classic method, the POD, belongs to this category of learning. This embedding consists of a linear transformation to a lower dimensional subspace. The mapping between the high-dimensional and low-dimensional state of the data can also be structured using non-linear functions such that the underlying structures in the non-linear manifolds are uncovered. The kernel principal component analysis is one example of such embedding method which are commonly used as a feature extraction method in the explanatory variables [173, 174]. Researchers report that reconstruction from their corresponding principal components to the higher-dimensional feature space is always possible, however there are still difficulties in the reconstruction to the input space [175]. The autoencoder algorithm is an alternative example of embedding algorithms. This algorithm makes use of two NNs as a means to first compress the data using the first network and to decompress the data with the second, for the purpose of information filtering bottleneck [176, 177].

Generative adversarial network is another example of unsupervised learning models which create dummy data that follows a probability distribution and aim to imitate the original data used in the training stage. The algorithm consists of two networks that compete with each other. The first network, termed as the generative network, creates the dummy data used in the training of the second network. The goal of the second network, termed as the discriminative network, is to optimise for a specific task [178].

Semi-supervised learning is a learning approach that operates under partial supervision. It makes an attempt to improve the performance of the supervised and unsupervised learning using labelled and unlabelled data, respectively [161]. This type of learning is particularly relevant to scenarios where labelled data is scarce [161]. It is usually referred to as either transductive learning. The goal of transductive learning is to infer the correct targets for the given unlabelled data. This is generally performed by adding a loss term which accounts for the learning on the unlabelled data and encourages the model to generalise better [179]. Two recent examples of this type of learning are the extended-generative adversarial networks [180] or variational auto-encoders [181]. More information on various methods in this type of learning can be found in [161].

Reinforcement learning is another area of machine learning and it has been used as a mathematical framework for solving problems that involves goal-oriented interactions of an agent with its environment [182]. The agent performs an action from a library of actions, perceives the state of the action and compensates the state with full, partial or negative rewards. In addition to that, the agent is not only concerned with uncovering patterns in its actions or in the environment, but also with maximising its long term rewards. Applications of reinforcement learning that exemplify its strengths and limitations are found in gaming [183]. It is worth noting that reinforcement learning requires a lot of computational resources due to the number of actions required through the trial and error interactions of the agent with the environment [184]. The authors in [185] underline that reinforcement learning may be prohibitive in experiments and flow simulations as it is computationally expensive for high-dimensional case studies. However, the authors also quote that this is rapidly changing as more computational resources are becoming available [185].

3.2 Proper orthogonal decomposition

The POD is a popular model order reduction technique that has successfully been applied in many areas of science and engineering. The POD has its roots in statistical analysis and has been referred as the principal component analysis, Karhunen-Loeve decomposition, Hotelling transform and empirical orthogonal eigenfunctions in other disciplines [186]. The goal of the POD is to identify an optimal coordinate system, here a set of linear basis, to represent an ensemble of empirical solutions by removing redundant information or noise [187]. The most common applications in CFD problems are found in transient fluid flow problems where the POD is constructed as a non-intrusive ROM [39, 10, 11, 12]. The application of POD to parametric problems in CFD, as considered in this thesis, is more recent and some examples can be found in [28, 35, 13].

According to the authors in [188], the striking property of the POD is its optimality as it is observed that POD is efficient at capturing dominant components. Moreover, the authors underline that when the modes are truncated and arranged from highest to lowest energies, it captures more relative energies with the selected modes than any other decomposition method truncated in the same way [188]. In addition, the POD is said to be an unbiased technique since it acts solely on the data and requires no prior information on which the optimal linear basis functions are constructed [189].

3.2.1 Computation of the POD modes

Let us consider a set of $\mathtt{n_{Tr}}$ training cases, usually called *snapshots* in a POD context, defined by a vector of N inputs, $\boldsymbol{x}^k = \{x_1^k, \dots, x_N^k\}$, and a vector of M outputs, $\boldsymbol{y}^k(\boldsymbol{x}^k) = \{y_1^k(\boldsymbol{x}^k), \dots, y_M^k(\boldsymbol{x}^k)\}$, for $k = 1, \dots, \mathtt{n_{Tr}}$. The matrix of snapshots is formed with all the available outputs as

$$\mathbf{Y}(\boldsymbol{x}^1, \boldsymbol{x}^2, \dots, \boldsymbol{x}^{\mathtt{n_{Tr}}}) = \begin{bmatrix} y_1^1(\boldsymbol{x}^1) & y_1^2(\boldsymbol{x}^2) & \cdots & y_1^{\mathtt{n_{Tr}}}(\boldsymbol{x}^{\mathtt{n_{Tr}}}) \\ y_2^1(\boldsymbol{x}^1) & y_2^2(\boldsymbol{x}^2) & \cdots & y_2^{\mathtt{n_{Tr}}}(\boldsymbol{x}^{\mathtt{n_{Tr}}}) \\ \vdots & \vdots & & \vdots \\ y_M^1(\boldsymbol{x}^1) & y_M^2(\boldsymbol{x}^2) & \cdots & y_M^{\mathtt{n_{Tr}}}(\boldsymbol{x}^{\mathtt{n_{Tr}}}) \end{bmatrix}. \tag{3.1}$$

This snapshot matrix can be approximated using a set of linear basis functions, $\boldsymbol{\varphi}$, written as,

$$\boldsymbol{y}^k(\boldsymbol{x}^k) = \sum_{j=1}^{M} \alpha_j^k(\boldsymbol{x}^k)\boldsymbol{\varphi}_j\,, \tag{3.2}$$

where α_j^k corresponds to the coefficients of the basis vectors, $\boldsymbol{\varphi}_j$. The mathematical statement of optimality is that the basis functions are chosen so that they maximise the averaged projection of the ensemble of snapshots onto the projected basis, written as,

$$\max_{\boldsymbol{\varphi}} \frac{\overline{|\boldsymbol{\varphi}^T \boldsymbol{y}^k|^2}}{||\boldsymbol{\varphi}||^2}\,, \quad \text{subject to} \quad ||\boldsymbol{\varphi}||^2 = 1, \tag{3.3}$$

where $\overline{\cdot}$ denotes the mean, $|\cdot|$ is the modulus and $||\cdot||$ denotes the L^2-norm. The mean is maximised to reflect the goal of representing a typical member of the snapshots and the normalisation factor $||\boldsymbol{\varphi}||^2$ is included to prevent $\boldsymbol{\varphi}$ from increasing without limit.

However, the functional in (3.3) can have multiple local maxima which would provide additional basis functions for the decomposition of equation (3.2). In order to constrain this optimisation problem under the condition $||\boldsymbol{\varphi}||^2 = 1$, the problem is reformulated to seek the maximum of

$$J(\varphi) = \overline{|\boldsymbol{\varphi}^T \boldsymbol{y}^k|^2} - \mathcal{L}(||\boldsymbol{\varphi}||^2 - 1), \tag{3.4}$$

where $\mathcal{L}$ is a Lagrange multiplier. After some algebra, as described in [186], it is shown the basis functions that are being sought, must satisfy the eigenvalue problem for a finite dimensional case as,

$$\frac{1}{\mathtt{n_{Tr}}} \mathbf{Y}\mathbf{Y}^T \boldsymbol{\varphi}_j = \mathcal{L}_j \boldsymbol{\varphi}_j, \tag{3.5}$$

where $\mathbf{Y}\mathbf{Y}^T$ represents the autocorrelation matrix of size $M \times M$. Their corresponding eigenvalues, $\mathcal{L}_j$ are real and positive, and represents the relative energy contained in each POD mode. The eigenvectors $\boldsymbol{\varphi}_j$ are mutually orthogonal and are termed the POD basis vectors [190]. Building a ROM from an ensemble of snapshots using POD implies that the constructed ROM will contain all the intrinsic properties found within the data as only linear operations are performed to obtain the basis [191]. This provides confidence that no information is lost in the process.

It is worth noting that even in the case where only a small number of modes is necessary to represent a new function which is not found in the snapshot matrix, an eigenvalue problem of order equal to the size of the autocorrelation matrix is still required to be solved. To improve this, iterative techniques can be used to find eigenvectors with the highest energy content first and then downward to lower energy contents.

A more elegant procedure that reduces the cost for computing these eigenvectors is the use of the singular value decomposition [22]. The singular value decomposition of the snapshot matrix is given by

$$\mathbf{Y} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{W}^*, \tag{3.6}$$

where $\mathbf{V}$ is a unitary matrix of dimension $M \times M$, whose columns, $\mathbf{V}_j$ for $j = 1, \dots, M$, are the left singular vectors of $\mathbf{Y}$ and corresponds to the eigenvectors $\boldsymbol{\varphi}$ in (3.2) and $\boldsymbol{\Sigma}$ is a rectangular diagonal matrix of dimension $M \times \mathtt{n_{Tr}}$ whose entries, ψ_j for $j = 1, \ldots, \min\{M, \mathtt{n_{Tr}}\}$, are the singular values of $\mathbf{Y}$ and equal to $\psi_j = \sqrt{\mathcal{L}_j}$. $\mathbf{W}$ represents a unitary matrix of dimension $\mathtt{n_{Tr}} \times \mathtt{n_{Tr}}$, whose columns, $\mathbf{W}_j$ for $j = 1, \ldots, \mathtt{n_{Tr}}$, are the right singular vectors of $\mathbf{Y}$ and the superscript $*$ denotes the conjugate transpose.

Since the columns of $\mathbf{V}$ form an orthonormal basis, each column of the snapshot matrix $\mathbf{Y}$, corresponding to a set of inputs $\boldsymbol{x}^k$, for $k = 1, \ldots, \mathtt{n_{Tr}}$, can be written as

$$\boldsymbol{y}^k(\boldsymbol{x}^k) = \sum_{j=1}^{M} \alpha_j^k(\boldsymbol{x}^k)\mathbf{V}_j, \tag{3.7}$$

where α_j^k are the POD coefficients and the vectors $\mathbf{V}_j$ are referred to as POD modes. The POD coefficients are computed as $\alpha_j^k = \mathbf{V}_j{}^T \boldsymbol{y}^k$ for $j = 1, 2, ..., M$.

The diagonal elements of $\boldsymbol{\Sigma}$ consist of $r = \min\{M, \mathtt{n_{Tr}}\}$ non-negative numbers which denotes the unique singular values of $\mathbf{Y}$ arranged in decreasing order, that is, $\psi_1 \geq \psi_2 \geq ... \geq \psi_r \geq 0$. The rank of $\mathbf{Y}$ is determined by the value of r. When the ensemble of snapshots considered has noise, a lower rank matrix approximation can be obtained where eigenvectors with the lower energy content are ignored or set to zero, resulting in a truncated set of basis $\mathbf{V}_j^\star$ for $j = 1, 2, ..., r^\star$, where $r^\star < r$. This yields an orthonormal basis that provides an efficient low-dimensional representation of the snapshot matrix. Among all the orthonormal bases of size $r^\star$, the POD basis minimises the least square error of the snapshot reconstruction,

$$\min_{\mathbf{V}_j} \left\| \mathbf{Y} - \mathbf{V}^\star \mathbf{V}^{\star T} \mathbf{Y} \right\|^2 = \sum_{j=r^\star+1}^{M} \psi_j{}^2. \tag{3.8}$$

The sum of squares of the singular values corresponding to the left singular vectors that are not included in the POD basis gives the square of the error in the snapshot representation. Thus, the singular values provide a quantitative guidance for selecting the lower rank order, $r^\star$ needed to approximate the snapshot matrix accurately. A typical approach used to select $r^\star$ is

$$\frac{\sum_{j=1}^{r^\star} \psi_j{}^2}{\sum_{j=1}^{M} \psi_j{}^2} > \kappa, \tag{3.9}$$

where κ is a user-defined tolerance and the authors in [192] underline that it should be chosen to be in the vicinity of the unity in order to capture most of the energy of the snapshot basis. The left-hand side of the above equation is referred as the relative energy captured by the first $r^{\star}$ POD modes.

3.2.2 Continuous extension of the POD coefficients

The ability of the POD to approximate the full order solution depends entirely on the information contained in the snapshot matrix used to generate the basis vectors. The continuous extension of the POD coefficients allows the user to approximate the outputs, $\boldsymbol{y}$, for a set of inputs, $\boldsymbol{x}$ that is not present in the set of training examples as

$$\boldsymbol{y}(\boldsymbol{x}) \approx \sum_{j=1}^{M} \alpha_j(\boldsymbol{x})\mathbf{V}_j, \tag{3.10}$$

where the POD coefficients α_j can be obtained using two approaches, the interpolation and projection methods, respectively. They both share the common eigenfunctions and differ from each other by the method to obtain the POD coefficients. The interpolation approach determines the POD coefficients for a new set of exploratory variables by solving an independent set of equations relating the explanatory variables in the training design space to its corresponding set of POD coefficients. On the other hand, the projection approach determines the coefficients by projecting the continuous governing equations onto a lower linear subspace spanned by the eigenfunctions. The POD with projection techniques are usually employed with Galerkin methods for generating solutions of unsteady problems [193, 194, 195]. This approach is not generally employed for steady problems, although few literatures are available [196]. It is worth noting that the projection approach is not adopted for problems which cannot be described in the form of continuous functions. The authors in [196] shows the implementation of POD with Galerkin projection for both steady and unsteady heat transfer problems.

Remark 1 (Local selection of eigenfunctions). *Following the parametric studies performed in [196], the authors deduced that the interpolation approach produces errors of magnitude lower than those produced with the projection approach, provided a local selection of the eigenvectors is performed to do the interpolation. As a result, this is applicable to the parametric studies performed in this thesis.*

In [197], the authors underline that the interpolation approach is accurate for strong non-linear problems and is commonly employed for steady problems. The POD with interpolation has been widely used as a data-driven technique and its low intrusiveness property has allowed it to be applied in various fields. Moreover, it appears to be well tested and a popular choice in the literature. As a result, due to the attractive properties of the POD with interpolation, the projection approach is not implemented in this work. For more information on the projection methods, see [196, 198].

It is worth noting that, in practice, the number of modes selected to construct the approximation $\boldsymbol{y}(\boldsymbol{x})$ is lower than M. However, in the examples shown in this work, all the POD modes are considered to ensure that the accuracy of the POD approximation is not influenced by the number of modes considered. In addition, the POD model does not have a global predictive feature, that is, large errors may be obtained when the POD model extrapolates outside the design space over which the lower subspace is built.

In the case of a one dimensional parametric problem, the continuous extension can be obtained by Lagrange linear interpolation, while in two dimensional cases, bilinear and spline interpolation can be performed. The expression for a single variable Lagrange linear interpolation is written as

$$\alpha_j(x) = \alpha_j(x_a) + \frac{\alpha_j(x_a) - \alpha_j(x_b)}{x_a - x_b}(x - x_a), \tag{3.11}$$

where the subscripts a and b denote the upside and downside values of the interpolation. When there are N number of independent parameters, researchers usually employ least square regression technique which performs a best fit approximation of the POD coefficients. The second-order least square method relating the POD coefficients to the design parameters $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_M\}$ can be written as

$$\alpha_j(\boldsymbol{x}) = w_0 + \sum_{i=1}^{M} w_i^{\hat{1},k} x_i + \sum_{j=1}^{M}\sum_{j=1}^{M} w_{ij}^{\hat{2},k} x_i x_j + \epsilon, \tag{3.12}$$

for $j = 1, \ldots, M$ and $k = 1, \ldots, \mathtt{n}_{\mathtt{Tr}}$. In the above expression, $w_i^{\hat{1},k}$ is the first i-th unknown coefficient of the k-th sample and ϵ is an estimate of the error of the approximation. The second model can be useful in the presence of strong non-linear behaviour in the input-output relation, however, there are cases where only a first-order least square is required to fit the system behaviour properly. For instance, when only a small number of model parameters is considered in the example or in subsonic flows.

For multi-dimensional parametric problems where an exact fit is not obtained with least square methods, researchers usually employ the radial basis functions (RBFs). The guiding principle behind this approach is to use translations of a single basis function $\Theta(r)$ that depends on the Euclidean distance from a specified center. This creates a multi-dimensional interpolant which is radially symmetric about that center [199]. RBFs are known to be an effective tool for solving partial differential equations of engineering and science problems [200, 198]. Some other applications can be found in surface reconstructions, mesh morphing, NNs, fuzzy system, pattern recognition and data visualisation [201, 202, 203].

The primary advantage of using RBFs is its meshless nature as other interpolation techniques, such as Hermite interpolation, require gradients defined at each data point which result in the need of triangulation [12]. Those techniques are highly dependent on the accuracy of the derivative estimates [19]. The RBF is found to have a superior performance when compared to 31 competing interpolation techniques on a range of functions [19]. Moreover, it is reported that the computational cost of the RBF approximation increases non-linearly with the number of points in the dataset and linearly with the dimensionality of the parametric problem [203]. According to the authors in [204], the RBF approximation was demonstrated to be flexible, convenient and accurate for any dimensional scattered data. However, one major drawback of the RBF interpolation is the cost of computing the RBF weights which becomes cumbersome when the number of data points exceeds 100 [19]. This is attributed to the linear system that needs to be solved is dense and often ill-conditioned [205]. However, this value is constantly changing due to the availability of computational resources.

The radial basis function interpolation is given by

$$\alpha_j(\boldsymbol{x}) = \sum_{k=1}^{\mathtt{n_{Tr}}} w_j^k \Theta(\|\boldsymbol{x} - \boldsymbol{x}^k\|), \tag{3.13}$$

for $j = 1, \ldots, M$, where w_j^k, for $k = 1, \ldots, \mathtt{n_{Tr}}$, is the set of unknown coefficients and Θ is a radial basis function. The RBF can also be expressed in terms of radius of influence r, where $r = \|\boldsymbol{x} - \boldsymbol{x}^k\|$. Amongst the many available choices, some of the most popular

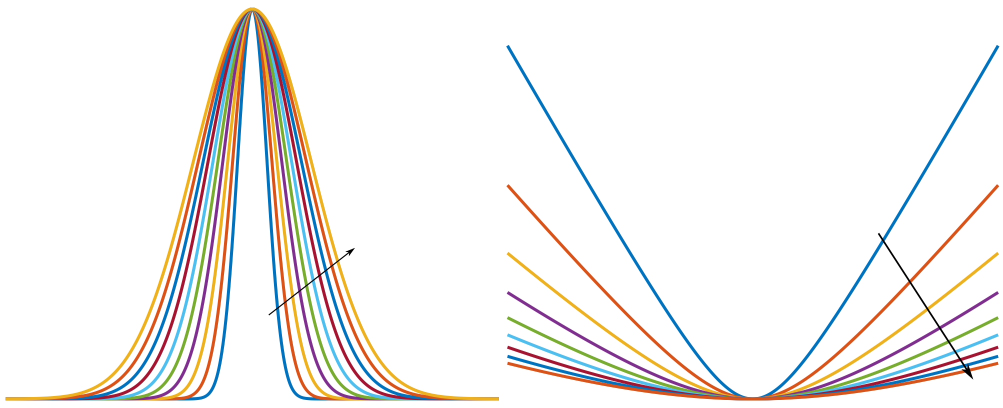

(a) Gaussian (b) Multi-quadric

Figure 3.2: The influence of the shape parameter ε on the shape of two popular basis functions that are employed in the RBF approximation. The arrow denotes the direction of increasing values of ε.

basis functions used in RBFs are,

$$\begin{aligned}
&\text{Gaussian:} && \Theta(r) = e^{-\dfrac{r}{2\varepsilon^2}}\,, \\
&\text{Multi-quadric:} && \Theta(r) = \sqrt{r^2+\varepsilon^2}\,, \\
&\text{Inverse multi-quadric:} && \Theta(r) = (r^2+\varepsilon^2)^{-1/2}\,, \\
&\text{Thin plate spline:} && \Theta(r) = r^2 \ln r\,,
\end{aligned} \tag{3.14}$$

where ε is a numerical parameter that dictates the radius of influence of the data. It is also commonly referred as the shape parameter and is subject to the constraint, $\varepsilon > 0$. The influence of the shape parameter on two common basis functions for the RBF approximations is shown in figure 3.2. In addition to the popular basis functions, there are also compactly supported RBFs, proposed in [206], that can be used. Their use is desirable due to their localisation properties. However, the author in [207] underlines that several compactly supported functions give rise to non-singular interpolation problems. Moreover, as it is more recent and therefore not well established in the literature compared to the traditional basis functions listed above, they are not considered in this work. Further information on the performance of the compactly supported radial basis functions is available in [208].

The coefficients w_j^k in equation (3.13) are computed by solving a set of independent linear system of equations,

$$\mathbf{A}\mathbf{w}^j := \mathbf{f}^j, \tag{3.15}$$

for $j = 1, \dots, M$, with $A_{IJ} = \Theta(\|\mathbf{x}^I - \mathbf{x}^J\|)$ and $f_I^j = \alpha_j^I(\mathbf{x}^I)$. I and J are the indices denoting the two inputs involved in the calculations of the RBF interpolant. When the value of ε is selected, the unknown coefficients $\mathbf{w}$ can be obtained provided it is non-singular, that is an inverse of the coefficient matrix can be obtained. The formulation presented in (3.13) implies the constraint of using as many RBFs as the number of data points. In this work, it is equal to the number of snapshots, $\mathtt{n_{Tr}}$. If this number is much larger than the number of degrees of freedom required to generate an acceptable fit, the linear system can become ill-conditioned which renders the matrix $\mathbf{A}$ as singular [209]. When these problems arise, researchers have employed the Moore-Penrose pseudo-inverse of matrix $\mathbf{A}$ which can be solved in a least square sense [210]. Polynomial terms have also been added to the RBF approximations to overcome the ill-conditioned matrix problem, referred as the augmented RBF method in the literature [210].

Following the work in [12], the RBF interpolation implemented with the multi-quadric basis function is selected. It was deduced in [19] that the multi-quadric function is the best technique for scattered data interpolation. Moreover, the author speculates that the coefficient matrix $\mathbf{A}$ is invertible and this approach is well-posed whenever the multi-quadric function is employed [19]. The accuracy of the multi-quadric RBF is dependent on the numerical parameter ϵ and to tackle this issue, the authors in [211] proposed an algorithm to find the optimum value for a particular data set. However due to the additional expense of performing such an analysis, a numerical experiment on the radius of influence is performed to obtain the optimum value in terms of the number of closest cases for the examples considered in this thesis.

3.3 Artificial neural networks

The artificial NN is a technique inspired by the human nervous system that allows learning by example from representative data that describes a physical phenomenon or a decision process [212]. The computational network relates the inputs and outputs of a system by forming a mathematical model using a composite of linear or non-linear functions to address a given problem. The authors in [213] emphasise that there are an infinite number of ways to arrange a NN, however only a few dozens have been explored and put to common usage.

3.3.1 Neural network architecture

The neuron of a NN represents a mathematical function and it is conceived as a model of biological neurons. Here the neuron in a NN takes a set of inputs, performs an operation using an activation function, and outputs a value. Each connection formed between the neurons has an associated weight.

A generic NN model can be described in terms of its *architecture* which defines the structure of the network. The most basic NN has three types of layered structure, the input layer, the hidden layers and the output layer, respectively. The input layer and the output layer consist of input neurons and output neurons. The layers that are found between these two layers are termed as the hidden layers and the neurons found within, as the hidden neurons. Each hidden layer can be chosen to perform a specific task and follow a specific connection between the layers in order to solve a given problem [214]. The type of connections between the layers forms two types of networks, the feed-forward NN and the feedback or recurrent NN, respectively [215]. The feed-forward NN has only forward connections from the input layer to the output layer. It is usually referred to as a static NN as only one set of outputs is obtained with one set of inputs. On the other hand, the feedback NN has both forward and feedback connections and is sometimes referred to as a dynamic NN, as several outputs can be obtained for the same set of inputs. Over the last decade, several other NN architectures have been invented that can be categorised under the same dichotomy. The multi-layer perceptron, convolutional NN, self-organising map, auto-encoders and decoders, and the more recent generative adversarial networks are a few examples of feed-forward networks [82, 172, 176, 178]. Recurrent NN, convolutional recurrent NN, long short term memory, convolutional memory networks are examples of feedback networks [216]. An illustration of the most basic NN with forward connections, the multi-layer perceptron NN, is shown in figure 3.3. The first and last layers correspond to the inputs and outputs, and the hidden layers, are numbered from $l = 1$ to $l = \mathtt{n}_\mathrm{L}$, with $\mathtt{n}_\mathrm{L}$ being the number of hidden layers. The number of hidden layers defines the *depth* of the NN. In practice, researchers classify NN with two or more hidden layers as deep networks and this field of study is loosely referred as *deep learning* [168].

According to the authors in [169], a regressor multilayer perceptron with at most two hidden layers can provide approximations to the desired degree of accuracy, provided

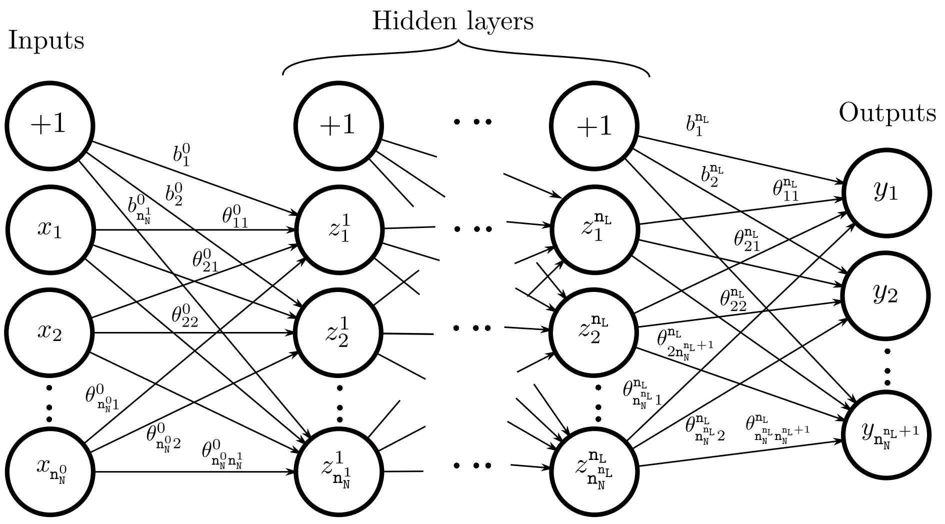


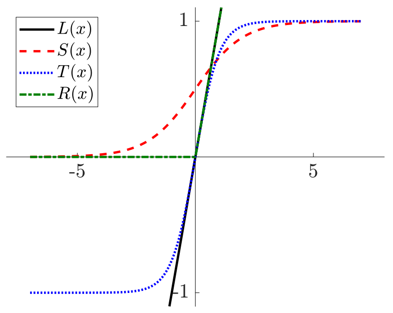
Figure 3.3: Schematic representation of a multi-layer perceptron NN.

sufficient hidden neurons are used and the optimised weights are not stuck in a local minimum. A more recent study in [217] also shows the universal approximation property of the multi-layer perceptron. It is also reported that for linear and quadratic functions only one hidden layer is required, while for approximations of higher orders, only two hidden layers should be used [217].

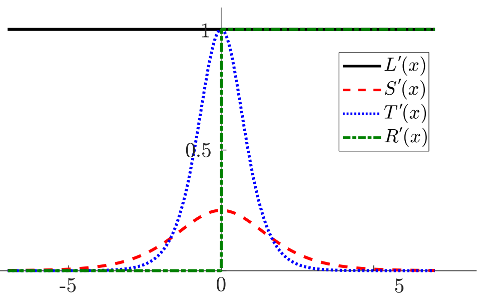
In this work, a static network is employed as steady state CFD solutions are used to build the datasets and following the findings of the authors in [169, 217], the multi-layer perceptron and its extendable deep-structured variant are selected and referred to as *NN* hereafter. It is worth noting that, the NN algorithm, *SwANN*, is employed to perform data analysis in this work. SwANN was written in Matlab by the author of this thesis. SwANN is a fully-connected NN with deep learning capabilities. In addition to the learning opportunity that comes with coding a NN toolbox from scratch, SwANN ensures it does not converge prematurely as three errors are monitored, namely the cost function, the finite precision errors propagated by the inputs and the derivatives of the cost function with respect to the weights.

3.3.2 Forward propagation

During the so-called *forward propagation*, the value associated to each neuron is computed by using the values associated to the connected neurons in the previous layer, the weights of the connections and an activation function F^l. More precisely, the value



(a) Activation function



(b) Derivative of activation function

Figure 3.4: The variation of four common types of activation functions employed in a NN and their corresponding derivatives.

of the j-th neuron in the layer $l+1$, denoted by z_j^{l+1}, is computed as

$$z_j^{l+1} = F^l \left(\sum_{i=1}^{\mathtt{n}_{\mathtt{N}}^l} \theta_{ij}^l z_i^l + b_j^l \right), \tag{3.16}$$

where b_j^l is a bias unit, θ_{ij}^l denotes the weight of the connection between the i-th neuron of the layer l and the j-th neuron of the layer $l+1$ and $\mathtt{n}_{\mathtt{N}}^l$ is the number of neurons in the layer l.

The activation function of the neurons in the output layer determines the objective of the network. It is essential to induce non-linearities in the NN as a composition of linear functions would only result in another linear transform. As a result, non-linear activation functions are used in the hidden layers to allow the NN to learn the complex mapping between the inputs and the outputs. There are various types of activation functions that have been employed over the years. Some of the popular functions are

Linear:	$L(x) = x$
Log-sigmoid:	$S(x) = \dfrac{1}{1+e^{-x}}$
Hyperbolic tangent sigmoid:	$T(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$
Rectified linear unit:	$R(x) = \max\{0, x\}$

Figure 3.4(a) shows the variation of the four types of activation functions that are commonly employed in NNs and their corresponding gradients. The linear activation function is usually used in the output layer when a continuous output is required while the sigmoid functions are by far the most popular choice when the outputs are discrete. When the rectified linear activation function is used, it can be either $R(x) = 0$

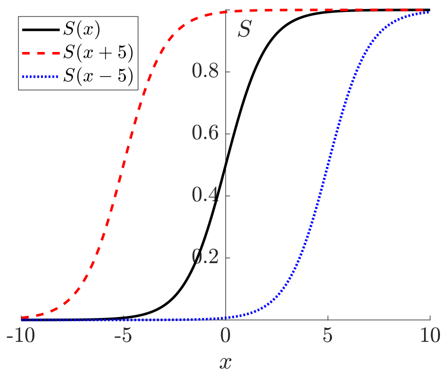


Figure 3.5: The effect of the added bias to the log-sigmoid function.

for $x \leq 0$ or $R(x) = x$ for $x > 0$. This results in the function to be non-differentiable for $x \leq 0$ and therefore the model cannot learn from the data when a gradient-based algorithm is employed to optimise the weights in the network.

Remark 2 (Inputs standardisation). *The log-sigmoid and the hyperbolic tangent function saturates as $|x|$ increases and it becomes difficult to update the weights of the network due to its vanishing gradient properties. As a result, experts always recommend to scale the inputs such that this problem is avoided [218].*

A comparison of the activation functions is performed in [219], to show the superiority of the sigmoid functions. The author underlines that the log-sigmoid function remains the most popular choice in NN models due to the ease of computing its derivatives [219].

Two classes of functions are considered in this work, namely the log-sigmoid function $S(x)$ and the linear function $L(x)$. When the NN is designed as a function approximant, the activation functions are selected as $F^l = S$ for $l = 0, \ldots \mathtt{n}_\mathrm{L} - 1$ and $F^{\mathtt{n}_\mathrm{L}} = L$. In contrast when the NN is designed as classifiers, all the activation functions are taken as the log-sigmoid, $F^l = S$ for $l = 0, \ldots \mathtt{n}_\mathrm{L}$. Figure 3.5 shows the effect of the added bias unit to the activation functions. This increases the flexibility of the NN model to fit the data by shifting to the left or right. The concept of the bias unit in NN is analogous to the role of the constant in the equation of a line. Without the constant, the line will only pass through the origin. Conversely, with the constant, the line can cross the y-axis at any position along the axis. The author in [220] carried out empirical studies to show the importance of the bias for accurate predictions in deep learning models.

3.3.3 Error evaluation of the NN model

A *training case* is defined by a vector of N inputs, $\boldsymbol{x} = \{x_1, \ldots, x_N\}^T$, and a vector of M outputs, $\boldsymbol{y}(\boldsymbol{x}) = \{y_1(\boldsymbol{x}), \ldots, y_M(\boldsymbol{x})\}^T$. Given a set of $\mathtt{n_{Tr}}$ training cases, $\boldsymbol{x}^k = \{x_1^k, \ldots, x_N^k\}$ and $\boldsymbol{y}^k = \{y_1^k, \ldots, y_M^k\}$, for $k = 1, \ldots, \mathtt{n_{Tr}}$, the *quadratic* cost function is defined as

$$C_Q(\boldsymbol{\theta}) = \frac{1}{M\mathtt{n_{Tr}}} \sum_{k=1}^{\mathtt{n_{Tr}}} \sum_{i=1}^{\mathtt{n_N^{n_L+1}}} \left[y_i^k(\boldsymbol{x}^k) - h_i^k(\boldsymbol{\theta}) \right]^2. \tag{3.17}$$

The values h_i^k correspond to the predicted outputs, computed using a forward propagation, starting from the input values $z_i^0 = x_i^k$ for $k = 1, \ldots, \mathtt{n_{Tr}}$ and $i = 1, \ldots, \mathtt{n_N^0}$. It is worth noting that the number of neurons in the input layer is taken as $\mathtt{n_N^0} = N$ and, similarly, the number of neurons in the output layer is taken as $\mathtt{n_N^{n_L+1}} = M$.

Equation (3.17) is the mean square error and it measures the discrepancy between the target outputs and the NN predictions when the NN is designed as a function approximant. Conversely, when the NNs are designed as classifiers, the *cross-entropy* cost function is the most popular option and is written as

$$C_{CE}(\boldsymbol{\theta}) = -\frac{1}{M\mathtt{n_{Tr}}} \sum_{k=1}^{\mathtt{n_{Tr}}} \sum_{i=1}^{\mathtt{n_N^{n_L+1}}} \left\{ y_i^k(\boldsymbol{x}^k) \log\left[h_i^k(\boldsymbol{\theta})\right] + [1 - y_i^k(\boldsymbol{x}^k)] \log\left[1 - h_i^k(\boldsymbol{\theta})\right] \right\} \tag{3.18}$$

In addition to the popular choices of cost function described in equations (3.17) and (3.18), there are several other options such as the exponential, Hellinger-distance, Kullback-Leibler divergence and Itakura-Saito distance cost functions [221]. These cost functions have been used in both supervised and unsupervised learning algorithms [221]. A comparison of the cost functions is beyond the scope of this work, see [222, 223] for more information.

3.3.4 Learning algorithms

The goal of the so-called *training stage* is to obtain the weights associated to all the connections of the NN that minimise the cost function. There are two main approaches, the gradient-based [215] and the gradient-free [224], respectively, each adapted to minimise the cost function of a NN model. Gradient-based methods which employ the backproprogation procedure to obtain the derivatives with respect to the weights, is by far the most commonly used technique for multi-layered NNs. This approach becomes more complicated to implement as the number of weights is increased and is intolerant to noisy objective function spaces, inaccurate gradients, categorical variables and topology optimisation [72]. Additionally, an often-mentioned disadvantage is that gradient-based techniques are generally sensitive to the initial set of weights and therefore may not reach the global optimum in high dimensional design problems. This is attributed to the noisy performance surface and gradient-based method may get stuck in a local minimum.

On the other hand, gradient-free optimisation does not rely on the calculations of derivatives with respect to the weights to move towards the global minimum. As a result it is known to be tolerant to noise in the objective function , however, it is reported that gradient-free algorithms such as genetic algorithm, require between five and 200 more times the number of function evaluations than the gradient-based approach for high dimensional problems [72]. Therefore, gradient-free algorithms in the context of weight optimisation in NN, are not employed as up to millions of weights are considered in this work.

The iterative process of the most general form of a gradient-based algorithm to compute the weights at iteration $r+1$ is given by

$$\theta_{ij}^{l,r+1} = \theta_{ij}^{l,r} + \tau s_{ij}^{l,r}, \tag{3.19}$$

or if the weights are re-arranged in a vector format for ease of understanding, it can be written as,

$$\boldsymbol{\theta}^{r+1} = \boldsymbol{\theta}^{r} + \tau \boldsymbol{s}^{r}, \tag{3.20}$$

where τ is the step length to move in the direction of the search direction $\boldsymbol{s}^r$. There are three main types of learning algorithms that are discussed in this section. The derivations of each one of them is based on a general quadratic cost function, expressed in a vector format as

$$C(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^T \boldsymbol{\mathcal{H}} \boldsymbol{\theta} + \boldsymbol{d}^T \boldsymbol{\theta} + c\,, \tag{3.21}$$

where $(\cdot)^T$ denotes the transpose operator of a tensor, $\boldsymbol{\mathcal{H}}$ is the symmetric Hessian matrix or the second derivative of the quadratic function, $\boldsymbol{d}$ is the free coefficient vector and c is the constant of the quadratic equation .

The first type of learning is the *Gradient descent*, which is a *first*-order optimisation algorithm for finding the minimum of a function iteratively. The aim of this task is to have $C(\boldsymbol{\theta}^{r+1}) < C(\boldsymbol{\theta}^{r})$ and to move in the search direction $\boldsymbol{s}$ by a step length τ. Consider the first order Taylor series expansion of the cost function, $C(\boldsymbol{\theta}^{r+1})$,

$$C(\boldsymbol{\theta}^{r+1}) = C(\boldsymbol{\theta}^{r} + \Delta\boldsymbol{\theta}^{r}) \approx C(\boldsymbol{\theta}^{r}) + \frac{\partial C}{\partial \boldsymbol{\theta}^{r}}^{T} \Delta\boldsymbol{\theta}^{r}, \tag{3.22}$$

where $\Delta\boldsymbol{\theta}^{r}$ represents a change in the values of the weights and using equation (3.20), it can be written as

$$\Delta\boldsymbol{\theta}^{r} = \boldsymbol{\theta}^{r+1} - \boldsymbol{\theta}^{r} = \tau \boldsymbol{s}^{r}, \tag{3.23}$$

and for the equality $C(\boldsymbol{\theta}^{r+1}) < C(\boldsymbol{\theta}^{r})$ to be valid,

$$\frac{\partial C}{\partial \boldsymbol{\theta}^{r}}^{T} \Delta\boldsymbol{\theta}^{r} = \tau \frac{\partial C}{\partial \boldsymbol{\theta}^{r}}^{T} \boldsymbol{s}^{r} < 0. \tag{3.24}$$

Since the learning rate τ is a small positive value, it implies that the dot product of the derivatives with respect to the weights and the search direction must be less than zero, and this inner product will be most negative when the derivatives are taken to be negative. Hence, the gradient descent algorithm can be written as,

$$\boldsymbol{\theta}^{r+1} = \boldsymbol{\theta}^{r} - \tau \frac{\partial C}{\partial \boldsymbol{\theta}^{r}}, \tag{3.25}$$

The derivatives with respect to weights in the input and hidden layers can be obtained using the so-called *backpropagation* procedure. As the error is not an explicit function with respect to the weights in the network, it can be difficult to compute the derivatives. The backpropagation procedure implements the chain rule, starting from the output layer to the input layer, to obtain the derivatives with respect to the weights at iteration $r+1$ and this can be written as

$$\frac{\partial C}{\partial \theta_{ij}^{r,l}} = \frac{2}{M \mathtt{n}_{\mathtt{Tr}}} \sum_{k=1}^{\mathtt{n}_{\mathtt{Tr}}} \sum_{j=1}^{\mathtt{n}_{\mathtt{N}}^{l+1}} \frac{\partial C}{\partial z_j^{k,l+1}} \frac{\partial z_j^{k,l+1}}{\partial \theta_{ij}^{r,l}}, \tag{3.26}$$

for $l = \mathtt{n}_{\mathtt{L}}, \mathtt{n}_{\mathtt{L}} - 1, ..., 0$, where the derivatives of the activation function $z_j^{k,l+1}$ of layer $l+1$, with respect to the weights, θ_{ij}^{l} of layer l at iteration $r+1$, can be expressed as

$$\begin{aligned} \frac{\partial z_j^{k,l+1}}{\partial \theta_{ij}^{r,l}} &= \frac{e^{-z_i^{k,l}\theta_{ij}^{r,l}}}{(1+e^{-z_i^{k,l}\theta_{ij}^{r,l}})^2} && \texttt{if} \quad F^l(\cdot) = S(\cdot), \\ \frac{\partial z_j^{k,l+1}}{\partial \theta_{ij}^{r,l}} &= z_i^{k,l} && \texttt{if} \quad F^l(\cdot) = L(\cdot). \end{aligned} \tag{3.27}$$

The learning rate in equation (3.25) is a positive scalar value and it can either be fixed, or it can be computed in each step to maximise the minimisation along every search direction. In the *fixed learning rate* approach, the maximum value of τ can be determined by expressing it in terms of the eigenvalues of the Hessian matrix of the cost function C. The derivative of equation (3.21) with respect to the weights in the output layer can be written as

$$\boldsymbol{g} = \frac{\partial C}{\partial \boldsymbol{\theta}} = \boldsymbol{\mathcal{H}}\boldsymbol{\theta} + \boldsymbol{d}, \tag{3.28}$$

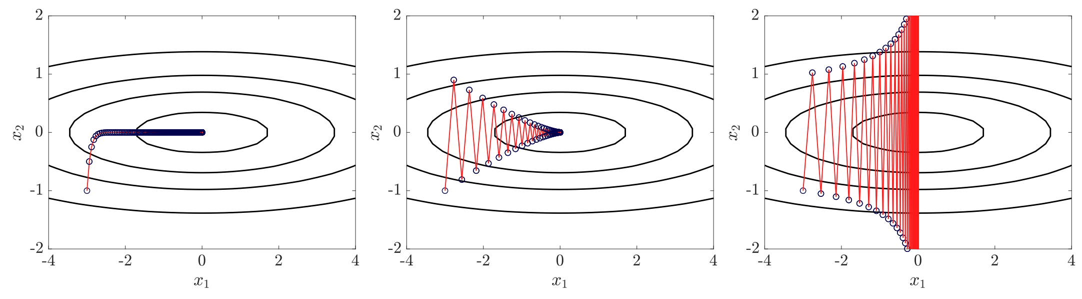
where $\boldsymbol{g}$ denotes the gradient vector. Substituting this expression in equation (3.25),

$$\boldsymbol{\theta}^{r+1} = [\mathbf{I} - \tau\boldsymbol{\mathcal{H}}]\boldsymbol{\theta}^{r} - \tau\boldsymbol{d}, \tag{3.29}$$

where $\mathbf{I}$ is the identity matrix. This is a linear dynamic system where the eigenvalues of $[\mathbf{I} - \tau\boldsymbol{\mathcal{H}}]$ is stable if it is less than one and that the eigenvectors of $[\mathbf{I} - \tau\boldsymbol{\mathcal{H}}]$ is the same as $[1 - \tau\gamma_i]$ where γ_i denotes the eigenvalue of the Hessian matrix, $\boldsymbol{\mathcal{H}}$ [225]. If it is assumed that the quadratic function has a strong minimum, then its eigenvalues must be positive [215] and the equation reduces to,

$$\tau < \frac{2}{\gamma_{\max}}, \tag{3.30}$$

where $\gamma_{\max}$ denotes the maximum eigenvalue of the Hessian matrix. The maximum stable learning rate is therefore inversely proportional to the maximum curvature of the quadratic function. If $\tau > \tau_{\max}$, that is, the learning rate is too large, the algorithm may jump past the minimum point and if $\tau = \tau_{\max}$, the algorithm will converge the quickest. To illustrate the sensitivity of the learning parameter, τ, consider the quadratic function $f(\boldsymbol{x}) = {x_1}^2 + 25{x_2}^2$, with a strong minimum at the origin. The eigenvalues of the quadratic equation are 2 and 50. According to the equation in (3.30), the maximum stable learning rate is 0.04. Figure 3.6 shows the influence of the learning parameter on the convergence of the gradient descent algorithm, with a fixed initial position at $x_1 = -3$ and $x_2 = -1$. It illustrates that the learning rate is limited by a particular value which corresponds to the maximum eigenvalue of the Hessian matrix. Figure 3.6(a) shows that with a smaller learning rate $\tau < 0.040$, the convergence is guaranteed. However, more iterations are required to reach the global minimum. The gradient descent algorithm converges the quickest when the learning parameter is equal to the largest eigenvalue, that is, $\tau = 0.040$, moving in the direction of the



(a) $\tau = 0.010$ (b) $\tau = 0.040$ (c) $\tau = 0.041$

Figure 3.6: The influence of the learning parameter τ on the convergence of the gradient descent algorithm for a quadratic function with a strong minimum.

eigenvector, as shown in figure 3.6(b). In the case where $\tau > 0.040$, the algorithm may get past the minimum as shown in figure 3.6(c). As a result, the gradient descent algorithm requires a careful selection of the learning parameter. A practical approach that researchers usually employ is to use small values of τ in the case where the function is unknown which may however affect the time for the algorithm to converge.

The variable learning rate approach minimises the cost function with respect to $\boldsymbol{\theta}^r + \tau \boldsymbol{s}^r$ along the search direction in the context of gradient descent, termed hereafter as the *variable gradient descent*. For a quadratic function, the derivative of the cost function with respect to the learning rate at iteration r can be written as

$$\frac{\partial C(\boldsymbol{\theta}^r + \tau \boldsymbol{s}^r)}{\partial \tau^r} = \frac{\partial C}{\partial \boldsymbol{\theta}^r}^T \boldsymbol{s}^r + \tau^r \boldsymbol{s}^{r\,T} \frac{\partial^2 C}{\partial \boldsymbol{\theta}^{r\,2}} \boldsymbol{s}^r \,, \tag{3.31}$$

The learning rate at iteration r can be obtained by setting equation (3.31) to zero and solve for τ^r,

$$\tau^r = -\frac{\dfrac{\partial C}{\partial \boldsymbol{\theta}^r}^T \boldsymbol{s}^r}{\boldsymbol{s}^{r\,T} \dfrac{\partial^2 C}{\partial \boldsymbol{\theta}^{r\,2}} \boldsymbol{s}^r} = -\frac{\boldsymbol{g}^{r\,T} \boldsymbol{s}^r}{\boldsymbol{s}^{r\,T} \boldsymbol{\mathcal{H}} \boldsymbol{s}^r}. \tag{3.32}$$

In the case of the gradient descent algorithm, the search direction is equal to the derivative of the cost function with respect to the weights. By minimising along the line, this allows the weights to move to the minimum along the search direction $\boldsymbol{s}^r$ at iteration r. This point is tangent to the contour line and since the gradient is orthogonal to the contour line, the gradient in the next iteration will also be orthogonal to the current iteration. The concept of orthogonality is also shown in another optimisation algorithm discussed later in this section.

To alleviate the common difficulty of gradient-based methods to reach a global minimum in problems where multiple local minimums are present, a heuristic modification to the simple gradient descent is applied in which a low pass filter is used. More precisely, the *momentum gradient descent* approach computes the optimum weights at the iteration $r+1$ as

$$\boldsymbol{\theta}^{r+1} = (1+\eta)\boldsymbol{\theta}^{r} - \eta\boldsymbol{\theta}^{r-1} - \tau(1-\eta)\boldsymbol{g}^{r}, \tag{3.33}$$

where $\eta \in [0,1)$ is the momentum coefficient of the first order smoothing filter.

Remark 3 (Momentum gradient descent)**.** *The momentum gradient descent allows the user to use a larger learning rate while maintaining the stability of the optimisation [215]. This could be possible as the oscillations on the performance surface are smoothed out using a first order filter. Another prime feature of this heuristic modification is that it tends to accelerate convergence when the trajectory is moving in a consistent direction [215].*

Next, the second type of learning is the *Newton's method* which is based on the *second*-order Taylor series expansion and has the *quadratic termination* property. In other words, it minimises a quadratic function in a finite number of iterations. The second order Taylor expansion of the quadratic cost function in equation (3.21) can be expressed as

$$C(\boldsymbol{\theta}^{r+1}) = C(\boldsymbol{\theta}^{r} + \Delta\boldsymbol{\theta}^{r}) \approx C(\boldsymbol{\theta}^{r}) + \boldsymbol{g}^{r^T}\Delta\boldsymbol{\theta}^{r} + \frac{1}{2}\Delta\boldsymbol{\theta}^{r^T}\boldsymbol{\mathcal{H}}^{r}\Delta\boldsymbol{\theta}^{r}. \tag{3.34}$$

Finding the derivatives with respect to the $\boldsymbol{\theta}^{r}$ and setting it to zero reduces the equation to

$$\boldsymbol{g}^{r} + \boldsymbol{\mathcal{H}}^{r}\Delta\boldsymbol{\theta}^{r} = 0\,, \tag{3.35}$$

and, rearranging the formula to obtain the Newton's method

$$\boldsymbol{\theta}^{r+1} = \boldsymbol{\theta}^{r} - [\boldsymbol{\mathcal{H}}^{r}]^{-1}\boldsymbol{g}^{r}. \tag{3.36}$$

Equation (3.36) will always march to the minimum of a quadratic function with a strong minimum in just one step. However, the performance surface in NN contains several local minima and therefore the learning algorithm may terminate in more than one step. Moreover, as the number of weights increases in the NN, the size of the Hessian matrix scales proportionally to the squared of the number of weights. On the other hand, the gradient vector scales proportionally to the number of weights in the NN. As a result, it may be impractical to perform the computation and storage of the Hessian matrix and its inverse.

To avoid this issue, researchers use the Jacobian matrix, $\boldsymbol{\mathcal{J}}$ to approximate the Hessian matrix [226]. For instance, the quadratic cost function in (3.21) can be further simplified to

$$C = \sum_{i=1}^{N} e_i(\boldsymbol{\theta})^2 = \boldsymbol{e}^T(\boldsymbol{\theta})\boldsymbol{e}(\boldsymbol{\theta}), \quad \text{where} \quad \boldsymbol{e}(\boldsymbol{\theta}) = \boldsymbol{y} - \boldsymbol{h}(\boldsymbol{\theta}), \tag{3.37}$$

where $\boldsymbol{y}$ and $\boldsymbol{h}$ are taken to be a vector of the target and predicted outputs respectively for ease of understanding. The derivative can be expressed as

$$\frac{\partial C}{\partial \boldsymbol{\theta}} = 2\boldsymbol{\mathcal{J}}^T(\boldsymbol{\theta})\boldsymbol{e}(\boldsymbol{\theta}), \tag{3.38}$$

where the Jacobian matrix, $\boldsymbol{\mathcal{J}}$ is expressed in its matrix form as

$$\boldsymbol{\mathcal{J}}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial e_1}{\partial \theta_1} & \frac{\partial e_1}{\partial \theta_2} & \cdots & \frac{\partial e_1}{\partial \theta_n} \\ \frac{\partial e_2}{\partial \theta_1} & \frac{\partial e_2}{\partial \theta_2} & \cdots & \frac{\partial e_2}{\partial \theta_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_N}{\partial \theta_1} & \frac{\partial e_N}{\partial \theta_2} & \cdots & \frac{\partial e_N}{\partial \theta_n} \end{bmatrix}. \tag{3.39}$$

Next, the Hessian matrix can be found using the Jacobian as

$$\boldsymbol{\mathcal{H}} = \frac{\partial^2 C(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^2} = 2\boldsymbol{\mathcal{J}}^T(\boldsymbol{\theta})\boldsymbol{\mathcal{J}}(\boldsymbol{\theta}) + \boldsymbol{\mathcal{S}}(\boldsymbol{\theta}), \tag{3.40}$$

where

$$\mathcal{S}_{k,j}(\boldsymbol{\theta}) = 2\sum_{i=1}^{N} e_i(\boldsymbol{\theta})\frac{\partial^2 e_i(\boldsymbol{\theta})}{\partial \theta_k \partial \theta_j} \tag{3.41}$$

It is assumed the contribution of $\boldsymbol{\mathcal{S}}(\boldsymbol{\theta})$ is comparably smaller than the product of the Jacobian and can be ignored. Therefore, the Hessian matrix can be approximated to be

$$\tilde{\boldsymbol{\mathcal{H}}}(\boldsymbol{\theta}) \approx 2\boldsymbol{\mathcal{J}}^T(\boldsymbol{\theta})\boldsymbol{\mathcal{J}}(\boldsymbol{\theta}) \tag{3.42}$$

Gauss-Newton method is obtained if the equations in (3.42) and (3.38) are substituted in (3.36),

$$\boldsymbol{\theta}^{r+1} = \boldsymbol{\theta}^r - [\boldsymbol{\mathcal{J}}^T(\boldsymbol{\theta}^r)\boldsymbol{\mathcal{J}}(\boldsymbol{\theta}^r)]^{-1}\boldsymbol{\mathcal{J}}^T(\boldsymbol{\theta}^r)\boldsymbol{e}(\boldsymbol{\theta}^r) \tag{3.43}$$

It is worth noting that the advantage of Gauss-Newton method is that it does not require the computation of the Hessian matrix. However in some cases, this matrix may not be invertible and therefore computing the Hessian would not be possible. For this operation to be possible, it can be made into a positive definite matrix by using, $\tilde{\boldsymbol{\mathcal{H}}} + \mu \mathbf{I}$, where μ is a positive scalar value. This value is increased until the eigenvalues, $(\gamma_i + \mu) > 0$ for all i, which leads to the *Levenberg-Marquardt* algorithm [227], written as,

$$\boldsymbol{\theta}^{r+1} = \boldsymbol{\theta}^r - [\boldsymbol{\mathcal{J}}^T(\boldsymbol{\theta}^r)\boldsymbol{\mathcal{J}}(\boldsymbol{\theta}^r) + \mu^r \mathbf{I}]^{-1}\boldsymbol{\mathcal{J}}^T(\boldsymbol{\theta}^r)\boldsymbol{e}(\boldsymbol{\theta}^r) \tag{3.44}$$

An important feature of this algorithm is when the value of μ^r is increased, it approximates the gradient descent algorithm with a small learning rate and as μ^r approaches zero, the algorithm approximates the Gauss–Newton method. For more information on the application of this algorithm within a NN framework, see [215].

Finally, the *conjugate gradient* method is yet another learning approach that has the quadratic termination property of Newton's method but does not require the computation of the Hessian matrix. The computation of the Jacobian matrix is still required and a set of mutually conjugate vectors with respect to the Hessian matrix is used as search directions. Here the conjugacy condition, $\boldsymbol{s}^{r^T}\boldsymbol{\mathcal{H}}\boldsymbol{s}^{r^\star}$ for $r \neq r^\star$, must hold.

From equation (3.21) and (3.28), the change in gradient vector at iteration $r+1$ can be written as

$$\Delta \boldsymbol{g}^r = \boldsymbol{g}^{r+1} - \boldsymbol{g}^r = \frac{\partial C}{\partial \boldsymbol{\theta}^{r+1}} - \frac{\partial C}{\partial \boldsymbol{\theta}^r} = \boldsymbol{\mathcal{H}}\Delta\boldsymbol{\theta}^r, \tag{3.45}$$

and similarly, the change in weights using equation (3.20) is,

$$\Delta\boldsymbol{\theta}^r = \boldsymbol{\theta}^{r+1} - \boldsymbol{\theta}^r = \tau \boldsymbol{s}^r. \tag{3.46}$$

The conjugacy condition can be restated as

$$\tau^r \boldsymbol{s}^{r^T}\boldsymbol{\mathcal{H}}\boldsymbol{s}^{r^\star} = \Delta\boldsymbol{\theta}^{r^T}\boldsymbol{\mathcal{H}}\boldsymbol{s}^{r^\star} = \Delta \boldsymbol{g}^{r^T}\boldsymbol{s}^{r^\star} = 0, \quad \texttt{for} \quad r \neq r^\star \tag{3.47}$$

Since the conjugacy condition is restated in terms of the gradient vector, there is no need to calculate the Hessian matrix iteratively. The search direction at iteration r will be conjugate to the search directions at any previous iterations if they are orthogonal to the changes in gradient. In order to maintain this conjugacy condition, a procedure similar to Gram-Schmidt orthogonalisation [215] can be implemented and this is

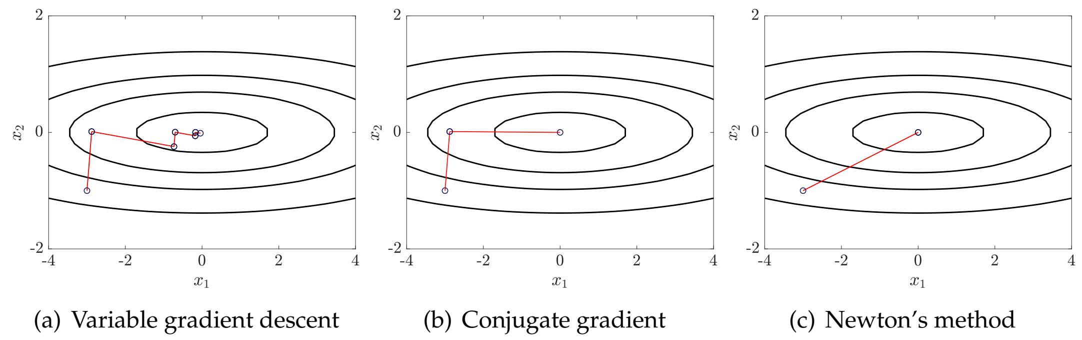


Figure 3.7: The comparison of the convergence of three learning algorithms using the quadratic function $f(\boldsymbol{x}) = {x_1}^2 + 25{x_2}^2$ and, the starting position, $x_1 = -3$ and $x_2 = -1$.

expressed as

$$\boldsymbol{s}^r = -\boldsymbol{g}^r + \beta^r \boldsymbol{s}^{r-1}, \tag{3.48}$$

where the constant β^r, due to the authors in [228], is defined as

$$\beta^r = \frac{{\boldsymbol{g}^r}^T \boldsymbol{g}^r}{{\boldsymbol{g}^{r-1}}^T \boldsymbol{g}^{r-1}}. \tag{3.49}$$

There are other alternatives for choosing the scalars of β^r and are discussed in [227]. It is worth noting that the first search direction, $\boldsymbol{s}^1$ can be arbitrary and it is usually taken to be the negative of the first gradient, $\boldsymbol{g}^1$. The search directions in the successive iterations, $\{\boldsymbol{s}^2, \boldsymbol{s}^3, ..., \boldsymbol{s}^r\}$, will be orthogonal to each other and the iterative formulation of the conjugate gradient method can be written as

$$\boldsymbol{\theta}^{r+1} = \boldsymbol{\theta}^r - \tau^r \boldsymbol{g}^r + \tau^r \beta^r \boldsymbol{s}^{r-1}. \tag{3.50}$$

Figure 3.7 shows the convergence of three learning algorithms using the quadratic function, $f(\boldsymbol{x}) = {x_1}^2 + 25{x_2}^2$, and the same initial coordinates, $x_1 = -3$ and $x_2 = -1$. Figure 3.7(a) shows the orthogonality properties of the search directions in the variable gradient descent approach. Next, the quadratic termination property and the conjugacy property of the conjugate method to terminate in only two steps are illustrated in figure 3.7(b). lastly, the quadratic termination property of the second-order Newton's method which terminates in only one step in figure 3.7(c).

The simple gradient descent is the simplest first order algorithm and it is often reported to have slow convergence when using a small learning rate or an oscillatory behaviour when a much larger learning rate is used, as shown in figure 3.6. As a result, it is hard to find the optimum value for a particular problem and can be regarded as a numerical parameter of the optimisation algorithm. The variable learning rate approach allows users to move to the minimum in each search direction and hence the algorithm converges in a smaller number of steps when compared to the fixed learning rate approach. However, to be able to compute the learning rate in each iteration of the variable gradient descent, the Hessian matrix needs to be computed, as shown in equation (3.32). The momentum gradient descent allows the use of a larger learning rate and hence a faster convergence by using a low pass first-order filter to damp down the oscillations found in the performance surface of a NN.

Newton's method is much faster and provides the user with quadratic convergence. However, it requires the computation and storage of the Hessian matrix and the calculations of its inverse. Although Levenberg Marquardt algorithm of Newton's method converges in only one step for a quadratic function, as seen in figure 3.7(c), with increasing number of parameters to optimise non-quadratic functions, the method may require more iterations to converge. As a result, it may seem to be an impractical approach. The conjugate gradient method, on the other hand, is a compromise as it does not require the computation of the second derivatives, and yet achieves the quadratic termination property with at most n iterations for functions of n-th orders. Figure 3.7(b) shows that the conjugate gradient approach converges in two steps for a quadratic function. However, in the application of NN, the performance surface is not of quadratic type due to the induced non-linearities of the activation functions in the system. As a result, the conjugate gradient method does not terminate in two steps. Two common steps are usually employed in conjunction with the conjugate gradient method, the *interval location* and *interval reduction* respectively, which works when a moderate number of parameters are optimised. The purpose of the interval location is to perform a linear search along the search direction to find some initial intervals that contains a local minimum. The interval reduction step then reduces these intervals to obtain a minimum at the desired degree of accuracy. More information on this adaptation to the conjugate gradient method can be found in [227, 215]. However, as the NN architecture becomes more pronounced, these approaches can become cumbersome. The author in [215] underlines that the conjugate gradient converges in fewer iterations compared to most optimisation algorithms, however each iteration performs more function evaluations than any other methods.

In this work, the backpropagation approach with momentum gradient descent is employed as it has demonstrated a good performance for training NNs of moderate size [229]. The algorithm also requires the choice of the parameters η and τ for the iterative process given by equation (3.33). This choice is usually based on numerical experiments and some recommendations can be found in the literature [215]. For the numerical examples presented in this thesis, the values selected are $\eta = 0.9$ and $\tau = 0.5$, and are close to the values found in other studies.

In addition to the selection of the learning algorithm, stopping criteria have to be defined to stop the training when the defined criteria satisfy a specified tolerance. The traditional stopping criterion is to check if the absolute value of the cost function falls below the specified tolerance. This can however be a drawback as the tolerance is chosen before the training is started. Such a choice is highly subjective to the dataset and involves trial and error. To circumvent this issue, researchers always work with standardised datasets and more than one criterion are usually defined to ensure convergence. To provide another quantitative measure of the error during the training, the error propagated by the inputs are calculated. Consider a non-linear function $F(z, \theta)$, which has two input variables, z and θ, each has an finite precision error and whose sources are known prior to the data manipulations. Similarly, the multiplication of two input variables incur another error in the operation. The errors in the inputs to each neurons is propagated forward through the hidden neurons of the hidden layers to the neurons of the output layer [230]. The first-order Taylor series approximation of the error E propagated by the inputs of the neuron z_j^{l+1} of layer $l+1$ can be written as

$$\begin{aligned} E\left(z_j^{l+1}\right) &= E\left(F^l\left(\sum_{i=1}^{\mathtt{n}_{\mathtt{N}}^l} \theta_{ij}^l z_i^l\right)\right) \\ &= \left(F^l\right)'\left(\sum_{i=1}^{\mathtt{n}_{\mathtt{N}}^l} \theta_{ij}^l z_i^l\right) E\left(\sum_{i=1}^{\mathtt{n}_{\mathtt{N}}^l} \theta_{ij}^l z_i^l\right), \end{aligned} \tag{3.51}$$

where $(F^l)'$ is the derivative of the activation function and,

$$\begin{aligned} E\left(\sum_{i=1}^{\mathrm{n}_{\mathrm{N}}^{l}} \theta_{ij}^{l} z_i^l\right) &= \sum_{i=1}^{\mathrm{n}_{\mathrm{N}}^{l}} E\left(\theta_{ij}^{l} z_i^l\right) \\ &= \sum_{i=1}^{\mathrm{n}_{\mathrm{N}}^{l}} \theta_{ij}^{l} E(z_i^l) + E(\theta_{ij}^{l}) z_i^l + E(\theta_{ij}^{l}) E(z_i^l) \\ &\approx \sum_{i=1}^{\mathrm{n}_{\mathrm{N}}^{l}} \theta_{ij}^{l} E(z_i^l) + E(\theta_{ij}^{l}) z_i^l \,, \end{aligned} \tag{3.52}$$

where the errors $E(\theta_{ij}^l)$ and $E(z_i^l)$ are small numbers and their product is taken to be negligible. This error is used as one of the stopping criteria in the optimisation algorithm. Beside the cost function and the error propagated by the inputs, the derivatives of the cost function with respect to the weights are also monitored and used as a stopping criteria in this work as it provides an indication whether the NN is learning or not. The tolerances of all the stopping criteria used in this thesis are reported in later chapters.

3.3.5 Methods to improve generalisation

One of the most common problem with training NNs is *overfitting*, which is the consequence of a trained model that performs well on the training data and poorly on the unseen test data [231]. A trained model is said to be ideal when the network generalises well in both the training and test datasets. There are several approaches that have been proposed to address this problem, some of which includes, growing, global searches, drop-outs, early stopping, addition of noise and regularisation.

Growing methods start with the minimum number of hidden neurons and/or hidden layers in the NN and subsequently more hidden neurons or layers are added until an adequate performance is achieved. This represents one of the most common technique used to improve generalisation in the test set. *Global searches* are meta-heuristic or gradient free algorithms such as genetic algorithms and cuckoo search which perform a search of all possible network architectures to locate the simplest model that can generalise well. A well-known drawback of this method involves the high number of evaluations that are required to locate the minimum.

The *drop-out* technique has also proven to be an effective method in addressing overfitting in NNs. This approach preserves the size of the training dataset. The co-adaptation of hidden neurons which contributes to overfitting, is prevented by stochastically selecting the hidden neurons and setting the value of their activation function to zero during the training [232]. This approach is also conceived as a bagging approach that implicitly induces sparsity in the NN model. Several authors deem this technique as promising, especially in deep NN [233, 234]. However it is worth noting that the drop-out technique is not accustomed as researchers do not have clear understanding why this method works well in deep NN.

When the *early stopping* approach is used during the training, another dataset in addition to the training and test datasets, known as the validation set, is used to prevent the network from memorising the training data. This set is usually built by taking a subset of the training data, usually about $[10\%, 25\%]$. The error on the training and validation set is monitored during the training stage. The aim is to record the set of weights when the error on the validation set starts to diverge and that of the training still converges. It is assumed that the network is about to memorise the data and the predictions will not generalise well on the test dataset at this *clipping point* and beyond [235]. As a result, this clipping point is used as a stopping criterion. The use of validation set reduces the size of the training data available for learning and this can be a major setback when there is not enough data such as in experiments. The multi-fold cross-validation techniques, proposed in [236], provides a way to train on scarce datasets. It involves partitioning the dataset into a number of subsets, among which one subset is left out for validation and the rest is used for training. Multiple rounds of the different selections of the subsets are used for training and the validation results are usually averaged to obtain an overall performance of the model.

Next, there are three main approaches how the *addition of noise* can be used to overcome overfitting. Artificial noise has been added to the input data with the hope that the added noise will perturb the training [237]. This makes the learning more difficult for the NN to perfectly fit the training data. A common way to model the artificial noise is to use a Gaussian distribution and this would introduce two numerical parameters in the training, the mean and standard deviation of the perturbation, respectively [237]. The authors in [238] have added artificial noise to the weights of the NN and report that such an approach comes with rigorous weight updates which can lead to convergence issues. The authors in [239] propose a method to add noise during the training to alleviate this problem. Alternatively, researchers have also successfully addressed overfitting by adding artificial noise to the derivatives of the cost function with respect to the weights [240]. Literature where this method is used in deep NN and recurrent NN are available in [240, 241]. As this method is not well established in the literature, it is not implemented in this work.

Finally, *regularisation* is a method which is employed to penalise the weights to reduce the non-linear learning capabilities of the NN models. It corresponds to one of the most popular techniques used in NNs to avoid overfitting [242, 243]. This approach has been shown to marginally discourage overfitting in the training stage by adding a penalty term or regularisation term to the cost function of the NN model [244]. The two most common types of weight penalisation schemes are the L^1 and L^2 regularisation, respectively. For a NN model as a function approximant, the cost function with the L^1 regularisation is written as

$$\tilde{C}(\boldsymbol{\theta}) = \frac{1}{M\mathtt{n}_{\mathtt{Tr}}} \sum_{k=1}^{\mathtt{n}_{\mathtt{Tr}}} \sum_{i=1}^{\mathtt{n}_{\mathtt{N}}^{\mathtt{n}_{\mathtt{L}}+1}} \left[y_i^k(\boldsymbol{x}^k) - h_i^k(\boldsymbol{\theta})\right]^2 + \frac{\lambda}{\mathtt{n}_{\mathtt{Tr}}} \sum_{l=0}^{\mathtt{n}_{\mathtt{L}}} \sum_{i=1}^{\mathtt{n}_{\mathtt{N}}^{l+1}} \sum_{j=1}^{\mathtt{n}_{\mathtt{N}}^{l}} |\theta_{ij}^l|, \tag{3.53}$$

where $|\cdot|$ denotes the absolute value and λ is the overfitting parameter which controls the impact of the weight penalisation. It is worth noting that the weights to the bias term are not regularised as it has been shown that this usually has a negative effect on the NN approximation properties [245]. The cost function with the L^2 regularisation is,

$$\tilde{C}(\boldsymbol{\theta}) = \frac{1}{M\mathtt{n}_{\mathtt{Tr}}} \sum_{k=1}^{\mathtt{n}_{\mathtt{Tr}}} \sum_{i=1}^{\mathtt{n}_{\mathtt{N}}^{\mathtt{n}_{\mathtt{L}}+1}} \left[y_i^k(\boldsymbol{x}^k) - h_i^k(\boldsymbol{\theta})\right]^2 + \frac{\lambda}{2\mathtt{n}_{\mathtt{Tr}}} \sum_{l=0}^{\mathtt{n}_{\mathtt{L}}} \sum_{i=1}^{\mathtt{n}_{\mathtt{N}}^{l+1}} \sum_{j=1}^{\mathtt{n}_{\mathtt{N}}^{l}} (\theta_{ij}^l)^2. \tag{3.54}$$

Both approaches shrink the weights to low values and some to zeros, however more attention has been given to L^2 regularisation. The authors in [246] provides a discussion on the effectiveness of the two types of regularisation.

The drawback of using this method is that it is difficult to determine the optimum value of the overfitting parameter, λ. As a result, researchers consider this parameter as another hyperparameter of the NN and needs to be tuned for a specific problem. If this parameter is too small, the trained network may overfit and conversely if the parameter is too large, the network may underfit. To alleviate this problem, the author in [247] proposes a method to determine the overfitting parameter in an automated fashion using the Bayesian framework.

In this work, two methods are selected to improve predictions of the trained NN models, the growing method, and the L^2 regularisation respectively. Moreover, the performance surface of highly dimensional problems usually contains several local minima and saddle points. When the weights are initialised using the same initial weights, for instance zero value, it will converge to the same minima using gradient-based algorithms. On the other hand, if the weights are large, the solution can fall on a flat part of the performance surface due to saturation of the sigmoid activation function [215]. Following [248], the initialisation of the weights for a layer l uses a uniform distribution to break symmetry within the interval $[-\sqrt{a^l}, \sqrt{a^l}]$, with $a^l = 6/(\mathtt{n}_{\mathtt{N}}^l + \mathtt{n}_{\mathtt{N}}^{l+1})$. The authors remark that in order to avoid the multiplicative effects through layers, this initialisation procedure approximately satisfies the objectives of maintaining activation variances and back-propagated variances as one moves up or down the network [248]. Moreover, to mitigate the possibility of being stuck in a local minima, a practical approach is to train the network using more than one random initialisation and the result with the minimum error is taken.

3.4 Design of experiment

In order to use the various types of non-intrusive ROMs mentioned in this chapter, the training stage requires a selection of data points. This selection is usually obtained using *design of experiments*. It was originally introduced to plan and conduct experiments with the aim of understanding the behaviour of a system [27]. With the advent of numerical methods and computing facilities, researchers applied the same concept of performing experiment through computer-based simulations. Sampling the design space or selecting a good set of training data points is one of the key issues in numerical experiments as the goal is to maximise the amount of information from a limited number of samples.

The central composite design is a classic example of design of experiment which is usually employed for building second order models without the need to populate a three-level full factorial experiment. A drawback to this method is that the number of points increases exponentially with the number of design variables [249]. As a result, it becomes inefficient for high dimensional design problems, for instance, parametrised geometries in the area of aerodynamic design exploration. Another example is the

D-optimal design, which has also been widely utilised [249]. This method requires a smaller number of data points than the central composite design. However, the author in [27] underlines that D-optimal design has a model dependent efficiency which does not address the prediction variance. The author also remarks that a good experimental design for deterministic computer analyses should fill the design space rather than focusing on the boundary [27]. Two of the popular space filling methods are the orthogonal arrays [250] and latin hypercube samplings (LHS) [251, 252]. LHS designs are usually preferred over random sampling and stratified sampling, as it was found to estimate the mean, variance and distribution functions of an output more accurately. Moreover, LHS ensures that each of the input variables is well represented over the range considered, even when the number of input variables are large. As a result, this renders the method computationally inexpensive in comparison to other methods.

The process to generate samples with LHS involves two main steps, firstly the range of each input variable is divided into n intervals and secondly, a datapoint within each interval is chosen from a normal distribution. In real life applications, some combinations of the variables are not computationally feasible and LHS design allows to adjust a variable without undermining its fundamental properties [253]. Moreover, LHS provides the flexibility on the size of the sample as different applications may have different constraints such as time and budget. The authors in [254] report that the number of samples for building a response surface model with LHS is approximated to be between 20% and 50% of the size of sample generated using D-optimal designs. Figure 3.8 shows a comparison of the full factorial design using 3^3 variable combinations and the LHS design using three samples. The authors in [255] report that the sampling strategy should be based on the function to be approximated. However in the case where the function to be approximated is unknown, a sensitivity analysis based on the size of the samples is usually recommended. This ensures that a minimum number of samples is generated using the selected design of experiment such that the ROM accurately represents the *black-box* function the designer is attempting to model. A survey of the various types of design of experiments can be found in [27].

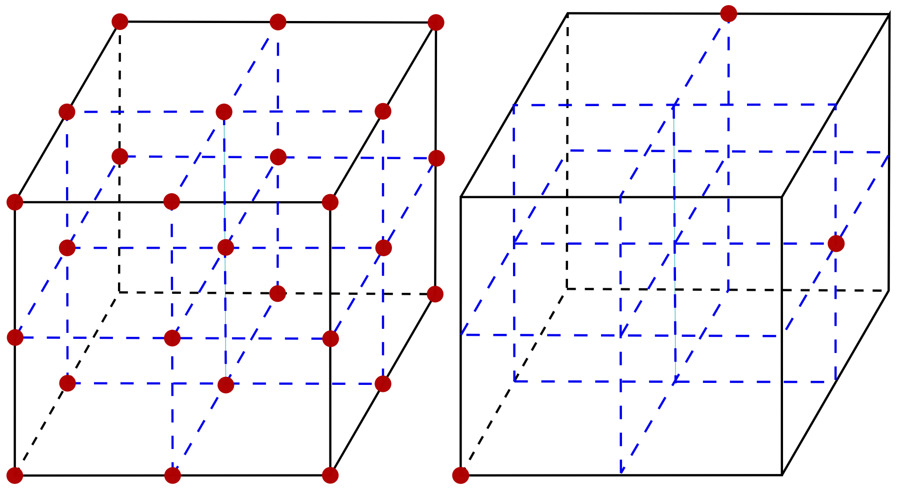

(a) Full factorial sampling (b) Latin hypercube sampling

Figure 3.8: An illustration of two design of experiment methods using three design variables.

3.5 Workflow

A summary of the workflow employed to perform prediction using a trained ROM follows. There are five main steps involved to obtain a trained ROM and the workflow used in this thesis is automated on high performance computing facilities.

1. The first step begins by generating a dataset for the training and test set using predefined intervals of the explanatory variables and Latin hypercube sampling.
2. The aerofoil and wing geometries over which the flow needs to be resolved are modelled in the second step. NURBS is used in the case where geometric parameters are involved.
3. The third step involves the domain discretisation using unstructured meshes around the geometry. To avoid the computational burden of generating meshes around several geometries, mesh morphing schemes such as Delaunay graph method and Laplacian smoothing are employed.
4. Next, the full order solution is obtained using the Flite system [88], which is a second-order vertex-centred FV solver.
5. The last step involves training a ROM using the training set and testing the performance of the ROM using the test set.

The *a posteriori* nature of this work allows the choice of multiple ROMs, however here the in-house NN or POD coupled with RBF is employed. It is worth noting that the training and test set are standardised, to avoid saturation in the log-sigmoid function employed in the hidden neurons, as noted in remark 2.

When choosing either the NN or the POD, there is the choice of several hyperparameters leading to various trained models with different predicting capabilities. As a result, a numerical experiment is performed to explore the accuracy of the predicted aerodynamic coefficients on the test set as a function of the hyperparameters of the employed ROM. For instance, in the case of NN there is the choice of the number of hidden neurons, $\mathtt{n_N}$, the number of hidden layers, $\mathtt{n_L}$and the overfitting parameters, λ. The NN is trained for each combination of the three listed hyperparameters in a predefined range and, the quadratic and cross-entropy cost function in equations (3.17) and (3.18) are employed to evaluate the errors in the training when the NN is used as a regressor and classifier, respectively. Moreover, the cost function is implemented with the regularisation term, as in equation (3.53), to avoid the need to divide the dataset into validation and training cases. This choice implies that the over-fitting parameter has to be tuned, but it ensures that the whole set of available data can be used for training. The NN is trained 10 times for each combination of hidden layer, hidden neurons and the overfitting parameter. This is performed in an attempt to obtain a trained model that is not stuck in a local minimum. The weights of the trained model with the lowest value of the error measured in the test set is recorded and the next combination of hyperparameters is trained. To facilitate the interpretation of the performance of each trained model, the errors are expressed in terms of lift, drag or moment counts, as they are more *user-friendly* and are generally used in the industry [256]. For instance, the error in the lift prediction of the k-th test case, measured in lift counts, is defined as

$$\varepsilon_{C_L,k} = |C_L(k) - C_L^\star(k)| \times 10^3, \tag{3.55}$$

where C_L is the lift coefficient from the CFD solver and $C_L^\star$ is the lift coefficient predicted by the ROM or NN. In the case when the drag or moment coefficient is predicted, the error measured in drag counts or moment counts is calculated in the ten thousandths decimal. In the numerical examples, two error measures are considered to assess the overall performance of the NNs, namely the mean value of the error measured in the test set, defined as

$$\overline{\varepsilon_{C_L}} = \frac{1}{\mathtt{n_{Te}}} \sum_{k=1}^{\mathtt{n_{Te}}} \varepsilon_{C_L,k}, \tag{3.56}$$

and the maximum value of the error measured in the test set, defined as

$$\varepsilon_{C_L,\mathtt{max}} = \max_{k=1,\ldots,\mathtt{n}_{\mathtt{Te}}} \left\{\varepsilon_{C_L,k}\right\} . \tag{3.57}$$

When a comparison of the performance of the ROM is performed, the trained model with the lowest mean value of the error measured in counts is selected.

Chapter 4

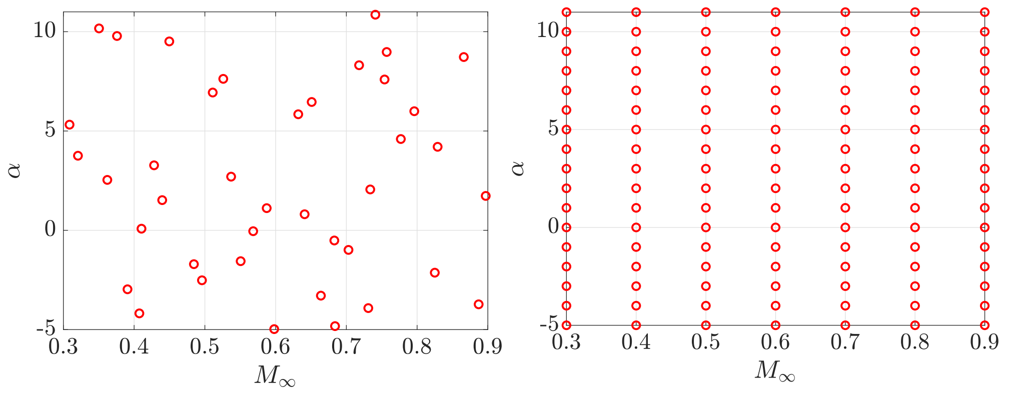
Aerodynamic predictions using flow parameters

The present chapter demonstrates the application of ROMs for the fast aerodynamic predictions in numerical examples involving flow parameters. There are four examples in this chapter. The first example shows the benefits of using the proposed NN against existing NNs, where the aerodynamic coefficients are predicted directly as the outputs, using two dimensional inviscid compressible data in the training. The second example provides a comparison of the NN with the POD using several strategies. The influence of the training examples on the accuracy of the predictions is also explored. Moreover, the choice of outputs in the proposed NN is justified by performing an analysis on the effect of the number of outputs on the accuracy of the predictions.

The multi-output NN is compared against existing NNs in the third example by using two dimensional viscous compressible solutions in the training. Finally, the fourth example considers the computation of the aerodynamic coefficients on wings by using inviscid compressible solutions in the training. Special emphasis is placed on the influence of the accuracy of the CFD data on the predictions of the ROM. The wide range of inflow conditions considered in all the examples of this chapter, leads to the subsonic and transonic flow regimes.

4.1 Benefits of multi-output NN

The first example considers the computation of the aerodynamic coefficients of a NACA0012 aerofoil at a free stream Mach number, M_∞, and an angle of attack, α, in predefined intervals $I_M = (0.3, 0.9)$ and $I_\alpha = (-5^\circ, 11^\circ)$, respectively.

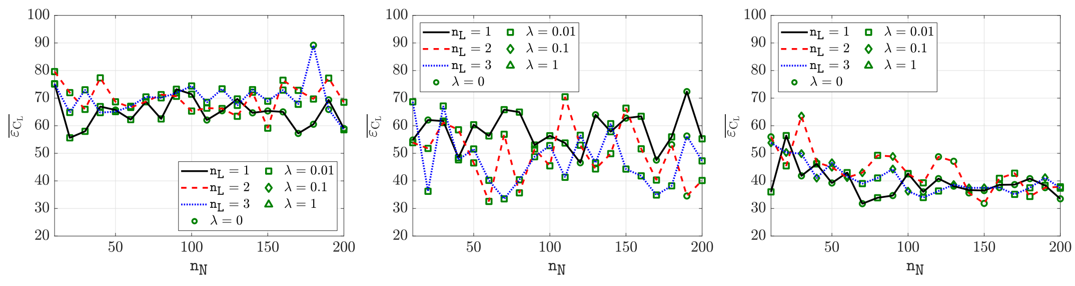


(a) Training set (b) Test set

Figure 4.1: The sampling space used to define the training and test dataset using $\mathtt{n}_{\mathtt{Tr}} = 40$ and $\mathtt{n}_{\mathtt{Te}} = 119$ cases respectively.

Three NNs are considered and compared. The first network considers M_∞ and α as inputs and, the lift, drag or moment coefficient as a single output. The second NN also considers M_∞ and α as inputs and has three outputs, corresponding to the three aerodynamic coefficients. The third network considers M_∞ and α as inputs and the output is the pressure at a user defined set of points on the aerofoil. The set of points considered corresponds to the 300 mesh nodes used to discretise the aerofoil. As the geometry is not changing, the pressure obtained from the full order solution is computed at those 300 points for each combination of explanatory variables. Additionally, this allows the user to have a consistent number of points at which the pressure is defined and therefore 300 outputs are used in the NN. The first and second networks have previously been considered [86, 50, 87, 51, 49, 61], whereas the third network, to the authors' best knowledge, has not been investigated previously.

All three networks are trained using a dataset of $\mathtt{n}_{\mathtt{Tr}} = 40$ simulations obtained from compressible Euler calculations. The training set is selected by using the latin hypercube sampling [251] in $I_M \times I_\alpha$. The test set consists of $\mathtt{n}_{\mathtt{Te}} = 119$ cases in $I_M \times I_\alpha$ and employs an equally-spaced distribution in I_M with step 0.1. Similarly, an equally-spaced distribution in I_α with step 1° is considered for each value of M_∞. Figure 4.1 shows the design sampling space used to define the training and test data. It can be observed that the test set contains a number of points that will induce an extrapolation, i.e. they are outside of the convex hull defined by the training set. Therefore, this example is also useful to evaluate the performance of the different networks when the predictions involve extrapolation. The learning rate and the momentum coefficient



(a) Network 1 (b) Network 2 (c) Network 3

Figure 4.2: Mean value of the error measured in lift counts, $\overline{\varepsilon_{C_L}}$, as a function of the number of hidden neurons, n_N, and the number of hidden layers, n_L.

of the networks considered in this work, are taken as $\tau \in [0.005, 0.5]$ and $\eta = 0.9$ respectively. These two parameters have been obtained after performing experiments with the two networks considered, and are similar to the parameters used in other studies [215].

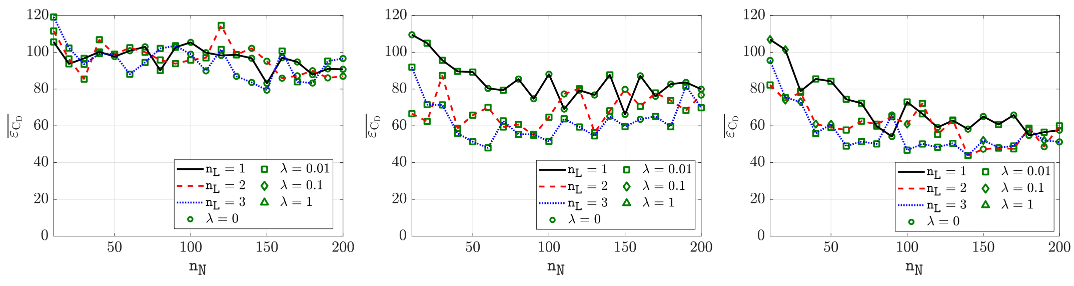
The first numerical experiment explores the accuracy of the predicted aerodynamic coefficients as a function of the number of hidden neurons, n_N, and the number of hidden layers, n_L, for the three networks considered and for different values of the over-fitting parameter, λ. It should also be noted that the third network outputs the pressure on the aerofoil and the computation of the lift is performed after the pressure distribution is predicted.

Figure 4.2 shows the mean value of the error, measured in lift counts, as a function of the number of hidden neurons, n_N, and the number of hidden layers, n_L. The results in figure 4.2(a) show that for the first network, with a single output corresponding to the lift, the highest accuracy is obtained with one hidden layer and a large number of neurons, namely $n_N \in [20, 30]$ and $n_N \in [170, 200]$. The best accuracy obtained with this network is $\overline{\varepsilon_{C_L}} = 55$. Similar accuracy is also provided by using three hidden layers and 200 hidden neurons. However, the results show that a small variation in the number of neurons can lead to a substantial increase in the error, suggesting a lack of robustness.

The second network, with three aerodynamic coefficients as outputs, provide more accurate results with two or three hidden layers over the range of number of neurons covered, $n_N = [60, 70, 170, 190]$, however, higher deviations from the mean error are observed in the first network. This indicates that this network has a higher dependence on the number of hidden layers and neurons.

Table 4.1: The selected NN configurations for the prediction of lift using the three networks considered.

	Network		
	1	2	3
Input	M, α		
Output	C_L	C_L, C_D, C_M	C_p
Hidden layer, $\mathtt{n_L}$	1	2	1
Hidden neuron, $\mathtt{n_N}$	20	60	70
Overfitting, λ	0.01	0.01	0



(a) Network 1 (b) Network 2 (c) Network 3

Figure 4.3: Mean value of the error measured in drag counts, $\overline{\varepsilon_{C_D}}$, as a function of the number of hidden neurons, $\mathtt{n_N}$, and the number of hidden layers, $\mathtt{n_L}$.

The results for the third network, with multiple outputs corresponding to the pressure on the aerofoil, are shown in figure 4.2(c). It can be observed that the accuracy is less dependent on the number of neurons in the hidden layers and the mean error has a tendency to decrease as the number of neurons is increased. The best accuracy obtained with this network is $\overline{\varepsilon_{C_L}} = 31$, when one hidden layer with 70 hidden neurons is used. There are no significant differences in the accuracy provided by the network with one hidden layer and a number of neurons $\mathtt{n_N} \in [120, 180]$ or with three hidden layer and a number of neurons in a similar range. This illustrates its robustness when compared to the first two networks, that is, the mean value of the error shows less variation with the number of hidden layers and hidden neurons. For the third network, there is no advantage in using more than one hidden layer. This shows that the level of non-linearity required to model the pressure on the aerofoil is lower than the level of non-linearity required to directly model the lift as an output. Table 4.1 shows a summary of the selected NN configurations for the predictions of lift in the test set using the three networks considered in this example. Figure 4.3 shows the mean value of the error, measured in drag counts, as a function of the number of hidden neurons, $\mathtt{n_N}$, and the number of hidden layers, $\mathtt{n_L}$ for the three different NNs. The first NN provides less accurate results, whereas the proposed NN provides the most

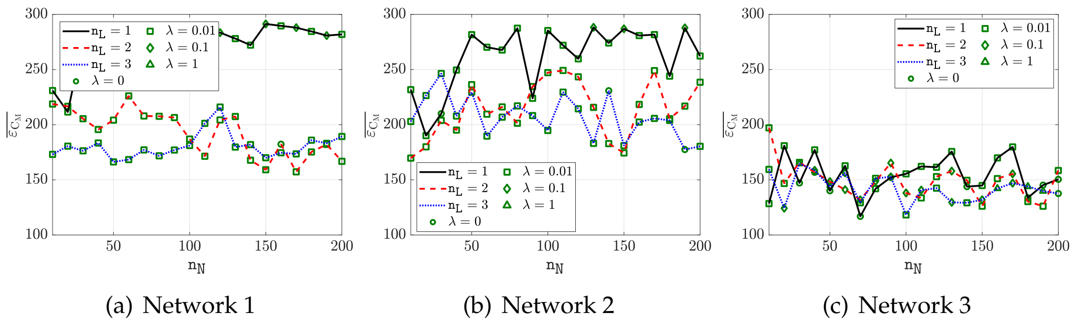


Figure 4.4: Mean value of the error measured in moment counts, $\overline{\varepsilon_{C_M}}$, as a function of the number of hidden neurons, $\mathtt{n_N}$, and the number of hidden layers, $\mathtt{n_L}$.

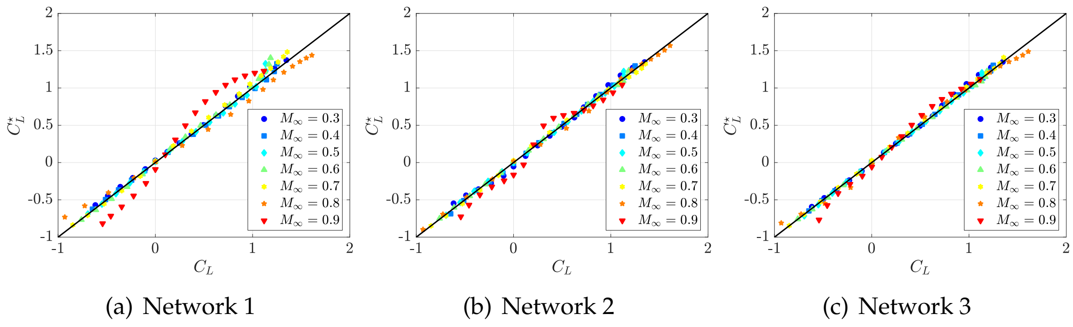


Figure 4.5: Regression plot for the lift coefficient as a function of the free-stream Mach number M_∞.

accurate results and is less sensitive to the choice of the hyperparameters. The second NN is also capable of providing accurate results if only two and three hidden layers are used.

Finally, figure 4.4 shows the mean value of the error, measured in moment counts, as a function of the number of hidden neurons, $\mathtt{n_N}$, and the number of hidden layers, $\mathtt{n_L}$ for the three different NNs. For this aerodynamic quantity of interest, the superiority of the proposed NN is clearly observed. The first and second NNs are not able to provide a single case where the prediction provides an error below 150 counts, whereas the proposed NN is consistently providing much lower errors. For this quantity of interest it seems advantageous to employ more than one hidden layer.

To illustrate the performance of the networks in the different flow regimes considered, figure 4.5 shows the regression plots for the lift coefficient using the three types of network employed. Different symbols are used for different free-stream Mach numbers to show the accuracy of the predictions in terms of the flow regime. The results shown correspond to the best accuracy measured in the mean value of the error in lift counts obtained with the networks considered. The first network employs one hidden layer and 20 hidden neurons, the second network has two hidden layers and 60 hidden neurons, and finally, the third network is selected to have one hidden layer and 70 hidden neurons. The results for the first and second networks, reported in figures 4.5(a), and 4.5(b), show a large deviation for $M_\infty = 0.8$ and $M_\infty = 0.9$. The higher accuracy of the third network is clearly observed in figure 4.2(c), with a much lower deviation in all cases. The results also show that the only sizeable deviation in the predictions with the proposed NN are observed for $M_\infty = 0.9$. This corresponds to cases which contains strong shocks and where an extrapolation is performed, as seen in figures 4.1.

4.2 Comparison of NN with the POD

The second example considers the prediction of the lift on RAE2822 aerofoil at a free stream Mach number, M_∞, and an angle of attack, α, in predefined intervals $I_M = (0.3, 0.9)$ and $I_\alpha = (-5^\circ, 12^\circ)$ respectively. It is worth noting that solutions of inviscid compressible data are used to build the datasets.

This example investigates the effect of performing a global or local binary classification, separating the subsonic and transonic cases, before training the NN or employing the POD. The influence of the number of training cases considered on the accuracy of the predictions is also studied by considering training sets ranging from $\mathtt{n_{Tr}} = 20$ up to $\mathtt{n_{Tr}} = 160$ cases. Moreover, this example is used to compare the performance of the NN with multiple outputs proposed in the previous example against a popular reduced order modelling technique, the POD [257], coupled with RBF. The performance of the NN is also compared with a physics-inspired NN [68], where the POD is coupled with a NN. In addition, the effect of the choice of number of pressure outputs on the accuracy of the lift predictions is investigated.

The first approach considers the network with multiple outputs proposed in section 4.1 and it utilises the whole set of training cases, including subsonic and transonic cases, to perform the predictions. This is the strategy proposed in the previous example that was shown to outperform the traditional techniques where the aerodynamic coefficients are directly predicted as the outputs of the network. It is referred to as *global* in this example.

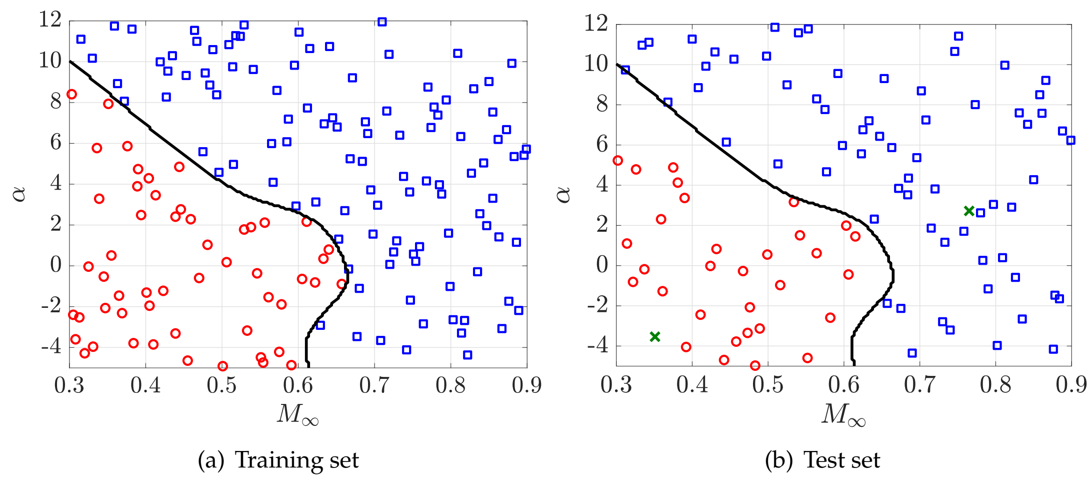


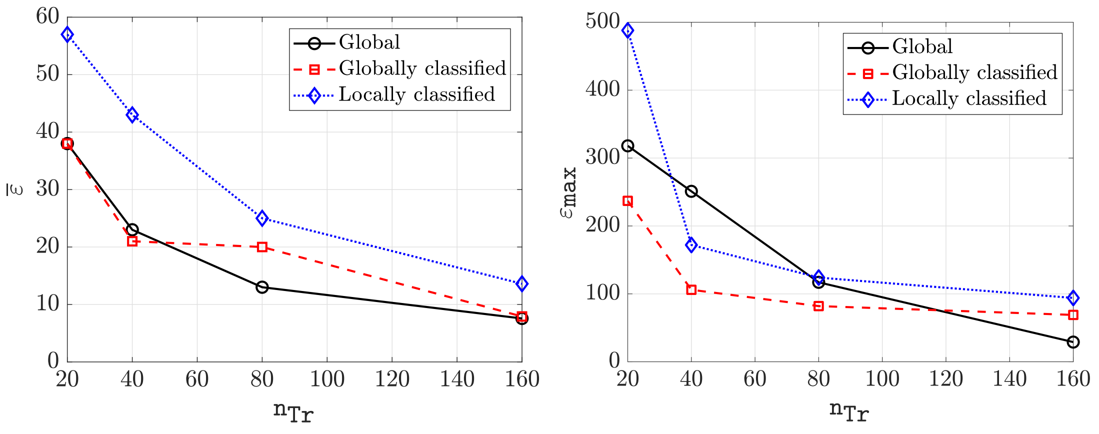
Figure 4.6: Binary classification of subsonic (denoted by circles) and transonic (denoted by squares) cases using the local Mach number M for a training set with $\mathtt{n_{Tr}} = 160$ cases. The continuous line denotes the classification boundary and it is shown with (a) the training cases and with (b) the $\mathtt{n_{Te}} = 100$ test cases. The highlighted crosses denote the two test cases used to further illustrate the performance of the three strategies later in this section.

The second approach proposed here, referred to as *globally classified*, consists of two stages. In the first stage, a binary classification of the training data is performed to separate subsonic and transonic cases, by considering the local Mach number, M. In the second stage two neural networks, or POD models, are built by using the subset of the training cases corresponding to subsonic and transonic cases respectively. When a prediction is performed, the inputs are used first to estimate if the case of interest is subsonic or transonic and the network, or POD basis, corresponding to the subsonic or transonic training cases is utilised. Figure 4.6(a) shows the results of the binary classification performed using a training set with $\mathtt{n_{Tr}} = 160$ cases obtained from the latin hypercube sampling in $I_M \times I_\alpha$. In this example, 53 cases are classified as subsonic and the remaining 107 cases as transonic.

The performance of the classification approach is illustrated in figure 4.6(b), where the classification boundary is shown together with a set of $\mathtt{n_{Te}} = 100$ test cases obtained from the latin hypercube sampling in $(0.35, 0.85) \times (-4.5, 11.5)$. The test cases have been selected in a smaller space compared to the training cases to minimise the adverse effects observed in the previous example when performing an extrapolation. The figure 4.6(b) shows the results of binary classification and table 4.2 demonstrates the accuracy of the binary classification approach in the test set. The results show that the classifier is able to predict all the subsonic cases correctly in the test set. On the other hand, the classifier predicts 67 out of the 68 transonic cases correctly and has

Table 4.2: The accuracy of the binary classification approach in the test set using the confusion matrix.

		Predicted		
		Subsonic	Transonic	Total
Targets	Subsonic	32	0	32
	Transonic	1	67	68
	Total	33	67	100



(a) Mean error (b) Maximum error

Figure 4.7: Evolution of the mean and maximum error on the test set, measured in lift counts, as the number of training examples is increased for the three strategies proposed using neural networks.

only one wrong prediction in the test set.

The third approach, referred to as *locally classified*, also consists of two stages. In the first stage, a binary classification of the training data is performed at each point of the aerofoil, to separate the cases that lead to a local Mach number lower or higher than one, at that point. In the second stage, as many networks or POD models are built as points on the aerofoil. All the examples considered here employ 300 points on the aerofoil. When a prediction is performed, the inputs are used first to estimate which network or POD model is to be utilised to predict the pressure at every point of the aerofoil.

Figure 4.7 compares the performance of the three strategies previously described by reporting the mean and maximum error as a function of the number of training cases, n_{Tr}. The results in figure 4.7(a) show that the global and the globally classified strategies provide very similar accuracy when the mean error is considered. In both cases, the accuracy obtained by using the set of $n_{Tr} = 160$ training cases is below 10 lift counts. In contrast, the locally classified strategy provides less accurate results for all

Table 4.3: The selected NN configurations for the predictions of lift using the three strategies considered.

	Strategy		
	Global	Globally classified	Locally classified
Input	M, α		
Output	C_p		
Number of NNs	1	2	600
Hidden layer, $\mathtt{n_L}$	1	1	1
Hidden neuron, $\mathtt{n_N}$	98	120	40
Overfitting, λ	0	0	0

the cases investigated. It can be observed that the accuracy of the global strategy with $\mathtt{n_{Tr}} = 80$ is similar to the accuracy of the locally classified strategy with $\mathtt{n_{Tr}} = 160$.

The accuracy in terms of the maximum error is represented in figure 4.7(b). For NNs trained with relatively low number of cases, i.e. $\mathtt{n_{Tr}} \leq 80$, the maximum error is around 100 lift counts and above, whereas using a set of $\mathtt{n_{Tr}} = 160$ training cases, the first strategy produces accurate results, with a maximum error of 29 lift counts. For the same set of training cases, the second strategy shows a substantially higher error, 64 lift counts, whereas the third approach produces a maximum error of 94 lift counts. Table 4.3 summarises the selected NN configurations for the predictions of lift in the test set using the three strategies considered.

To further illustrate the performance of the three strategies proposed, figure 4.8 shows a comparison of the pressure coefficient, C_p, obtained with the CFD solver and the predicted C_p using the three strategies considered. The two different cases considered correspond to the cases highlighted with a cross in figure 4.6(b) and they involve a subsonic and a transonic flow. The results show that all methods provide an accurate prediction of the pressure coefficient for the subsonic case. The prediction for the transonic case is substantially more challenging. The three strategies show a deviation with respect to the reference CFD results in the region near the strong shock on the top curve describing the aerofoil. The strategies that use the binary classification show a more pronounced oscillatory solution near the strong shock. This behaviour is attributed to the possibility of wrongly classifying the cases as subsonic or transonic, leading to the use of the incorrect NN to perform the prediction, as shown in table 4.2. It can be observed that the locally classified strategy shows oscillations even in the region near the leading edge. This behaviour is attributed to the amount of networks that are employed to perform the prediction. For each point on the aerofoil a different

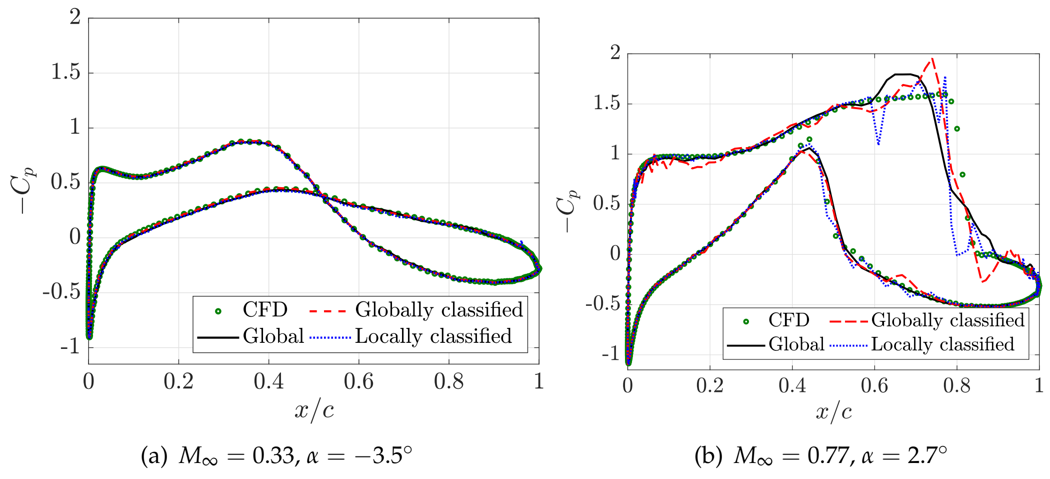


Figure 4.8: Comparison of the pressure coefficient, C_p, obtained with the CFD solver and the predicted C_p using the three strategies considered.

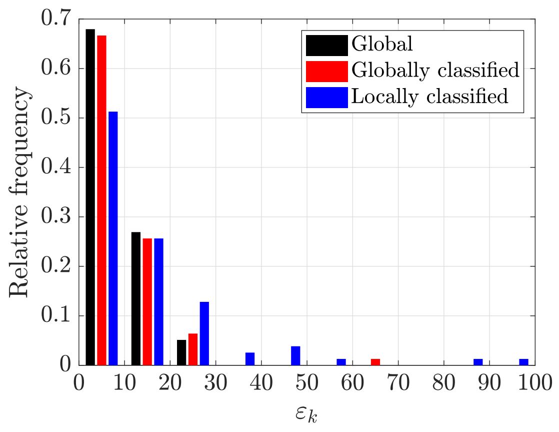
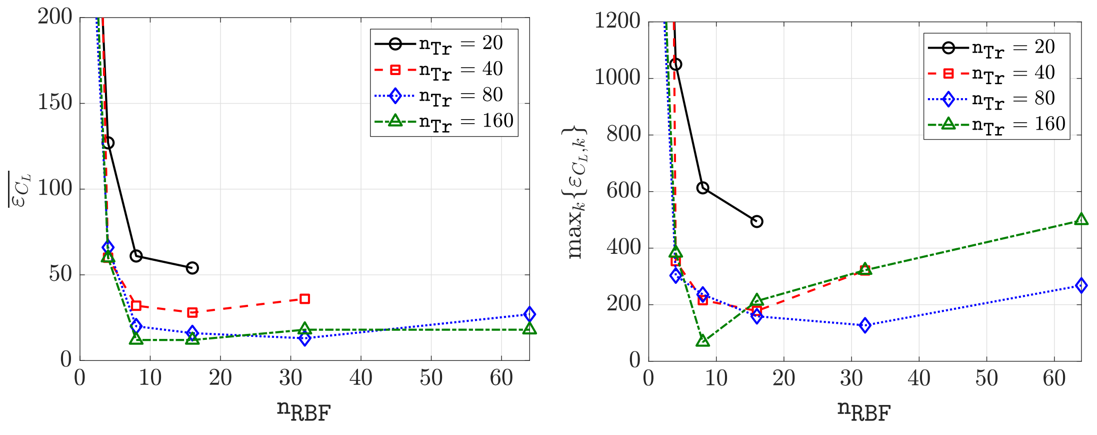


Figure 4.9: Relative frequency of the error on the test set, measured in lift counts, for all the test cases using neural networks with the three strategies considered.

network is used, leading to the possibility of having nodes where the binary classification fails to accurately predict the subsonic or transonic character of the solution.

Figure 4.9 quantifies the accuracy of the three NNs proposed. The histogram represents the relative frequency of the error, measured in lift counts, for all the test cases. The results show that the first strategy predicts almost 70% of the cases with an error lower than 10 lift counts and almost 30% of the remaining cases with an error lower than 20 lift counts. The second strategy is able to predict a similar number of cases with an error lower than 20 lift counts. However, it predicts a larger number of cases with higher errors and, more importantly, there are cases where the lift prediction has an error between 60 and 70 lift counts. The third strategy shows a significant lower percentage of cases where the prediction provides an error with less than 10 lift counts. In addition, it can be observed that the locally classified strategy leads to a number of



(a) Mean error (b) Maximumn error

Figure 4.10: Evolution of the mean and maximum error on the test set, measured in lift counts, as a function of the number of cases used to perform the interpolation with RBFs, $\mathtt{n}_{\mathtt{RBF}}$.

cases where the error is substantially higher.

The performance of the POD method with RBFs used to perform the interpolation [12], is evaluated. In all cases, POD modes are computed from the training cases, also referred to as snapshots in the context of the POD, corresponding to the pressure over the 300 points describing the aerofoil, rather than estimating the value of the lift directly. To predict the value of the pressure for a given test case, the RBF interpolation requires the selection of the radius of influence, that is the parameter ε in the multiquadric function of equation (3.14). To study the effect of this numerical parameter, figure 4.10 shows the evolution of the mean and maximum error, measured in lift counts, as a function of the number of cases that are included in the interpolation, $\mathtt{n}_{\mathtt{RBF}}$, for an increased value of the radius of influence. The results show that the optimal value of the radius, when the number of snapshots is large enough, i.e. $\mathtt{n}_{\mathtt{Tr}} = 160$, is such that the RBFs use the values of the closest eight cases to perform the interpolation. Larger values of the radius do not significantly affect the mean value of the error, as shown in figure 4.10(a), but lead to a loss of accuracy when the maximum error is considered, as shown in figure 4.10(b).

Figure 4.11 compares the performance of the three strategies previously described by reporting the mean and maximum error as a function of the number of training cases, or snapshots, in the context of the POD. The results show that the global strategy provides the best accuracy in all cases when the accuracy is measured as the mean or the maximum error. With the first approach, the accuracy is also less dependent

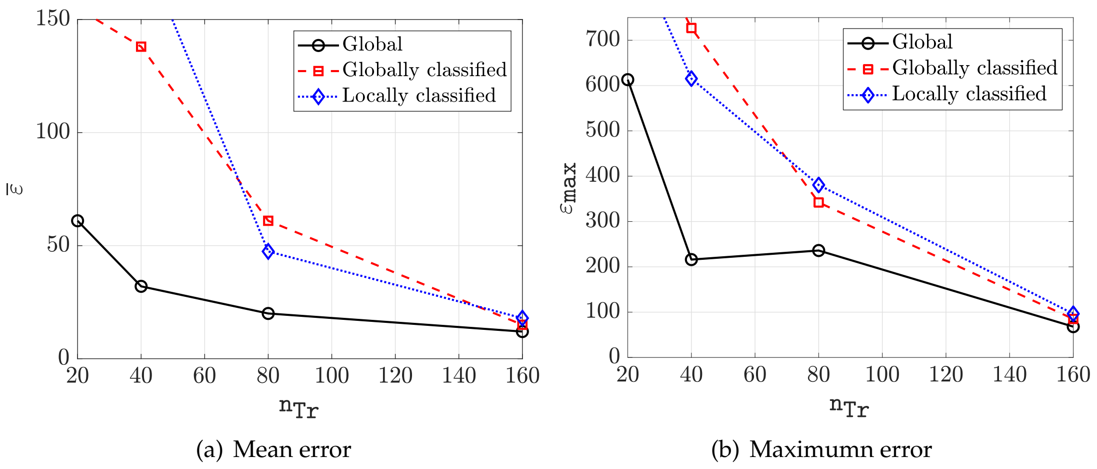


Figure 4.11: Evolution of the mean and maximum error on the test set, measured in lift counts, as the number of training cases, or snapshots, is increased for the three strategies proposed using the POD.

on the number of snapshots, whereas with the other two strategies the accuracy is highly dependent on the number of snapshots. For a large number of training cases, the accuracy of the three approaches is comparable.

The accuracy of the proposed NN and the POD with RBFs is compared. In both cases the global approach is adopted, as previous numerical examples show that this strategy provides the highest accuracy when predicting the lift coefficient. The NN employed has only one hidden layer and 98 neurons and the over-fitting parameter is set to $\lambda = 0$. For the POD, eight closest cases are considered to perform the interpolation using the RBFs. Additionally, the two methods are also compared with a third strategy. The third strategy builds a POD model using $n_{Tr} = 160$ simulations and employs the NN to interpolate the POD coefficients in the low dimensional subspace of the eigenfunctions, as proposed in [68]. The third strategy is referred to as *POD-NN* in this example. As before, a numerical experiment on the number of hidden layers and hidden neurons is performed to obtain the NN configuration with the lowest error. The NN predicting the POD coefficients employs one hidden layers with $n_N = 150$ neurons and the overfitting parameter is selected as $\lambda = 0.1$. In all the three cases, the prediction of the pressure is performed at the 300 points used to discretise the aerofoil and the lift coefficient is computed after the ROM makes a prediction.

Figure 4.12 compares the accuracy of the proposed NN, the POD coupled with RBFs and the POD coupled with NN. Figure 4.12(a) shows the evolution of the mean error on the test set, measured in lift counts, as the number of training examples is increased

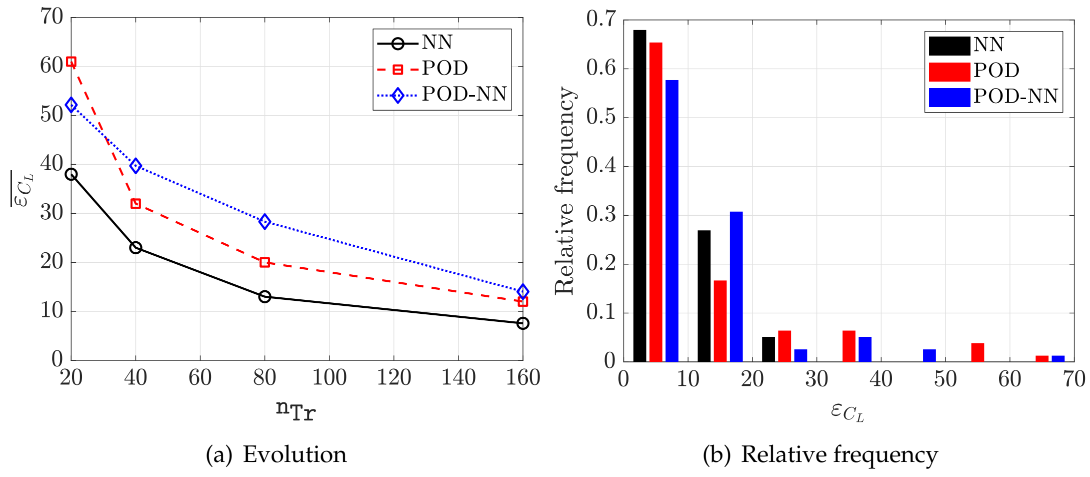


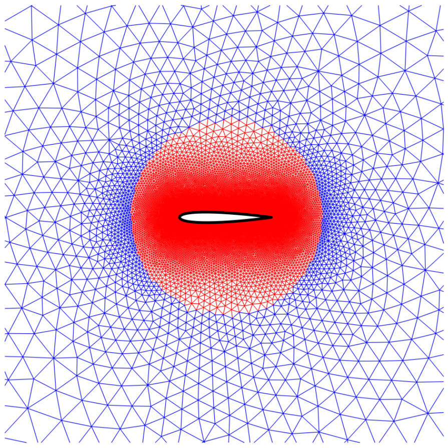
Figure 4.12: The evolution of the mean error on the test set, measured in lift counts, as the number of training cases is increased in (a) and the relative frequency of the error on the test set in (b), also measured in lift counts, when $\mathtt{n_{Tr}} = 160$ training examples are used, for the three global approaches.

from $\mathtt{n_{Tr}} = 20$ to $\mathtt{n_{Tr}} = 160$ simulations. The results show that the mean value of the error decreases as the number of training examples is increased for all three methods. However, NN can be seen to consistently produce more accurate results compared to the POD or POD-NN. The mean value of the error when $\mathtt{n_{Tr}} = 160$ simulations are used in the training is 8, 12 and 14 lift counts for the NN, POD, and the POD-NN, respectively. The histogram in figure 4.12(b) represents the relative frequency of the error, measured in lift counts, for all the test cases. The results show that the NN and the POD are able to provide an error below 10 lift counts for almost 70% of the test cases, with the NN achieving a marginally higher percentage. On the other hand, the POD-NN can be observed to predict with an error below 10 lift counts for only 58% of the test cases. The first significant difference is that the NN provides a prediction with an error below 30 lift counts for 100% of the test cases, whereas almost 90% of the test cases using the other two methods. The NN, POD and POD-NN predict with a maximum error of 29, 65 and 68 lift counts, respectively. It is worth noting that the overall performance of the POD coupled with RBFs is better than the POD coupled with NN. This is attributed to the local approximation property of RBFs.

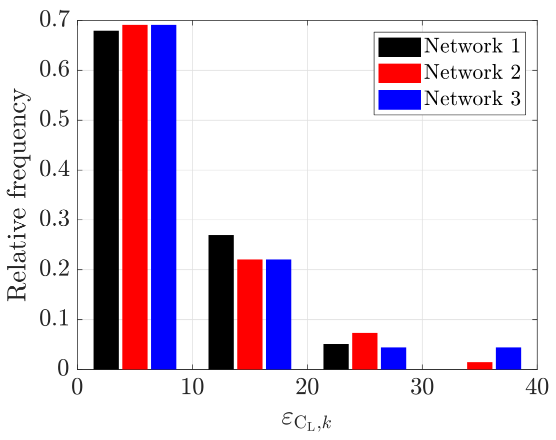
Lastly, with the aim to defend the choice of the proposed NN predicting on the aerodynamic surface, two other NNs are employed to investigate the effect of the choice of the NN outputs and the results are compared. All three NNs consider M_∞ and α as inputs. The first network corresponds to the proposed NN which considers the pressure defined at the mesh node discretising the aerofoil, as outputs. This corresponds to 300

Table 4.4: The selected NN configurations for the predictions of lift using the three networks considered.

	Network		
	1	2	3
Input	M, α		
Output	C_p		
Selected nodes	Surface	One chordlength	Full domain
Hidden layer, $\mathtt{n_L}$	1	3	3
Hidden neuron, $\mathtt{n_N}$	98	170	290
Overfitting, λ	0	0	0



(a) Domain



(b) Relative frequency

Figure 4.13: The domain of selection by colours and relative frequency of the error on the test set, measured in lift counts, for the three NNs.

outputs. The second network considers mesh nodes within one chord length from the midpoint position of the aerofoil as outputs and corresponds to 6808 mesh nodes. The third network considers all the nodes used to discretise the mesh and corresponds to 8004 outputs in the computational domain. Again, the networks are trained using $\mathtt{n_{Tr}} = 160$ simulations and tested with $\mathtt{n_{Te}} = 100$ simulations.

The first network employs one hidden layer and 100 hidden neurons, the second network employs three hidden layers and $\mathtt{n_N} = 170$ hidden neurons and lastly, for the third network, three hidden layers and 290 hidden neurons are selected. The three NNs select the overfitting parameter to be $\lambda = 0$. Table 4.4 summarises the selected NN configurations for the predictions of lift in the test set using the three networks considered.

Figure 4.13 shows the domain of selection by colours and the histogram quantifies the accuracy, measured in lift counts in the test set, for the three NNs. The results show that all three approaches predict with an error below 20 lift counts for almost

70% of the test set. However, significant differences can be clearly observed in the remaining 30% of the test data. The first network produces a maximum error below 30 lift counts, while the second and third networks have a maximum error of 33 and 39 lift counts, respectively in the test set. In addition, it is worth noting that the second and third networks require more hidden layers and neurons to be able to match the accuracy obtained using the first network. Additionally, the problem consists of optimising 30,398, 1,222,818 and 2,498,814 weights in the first, second and third networks, respectively. Due to the higher errors in the test set and the higher cost of training the second and third NNs, the first network is preferred.

4.3 Benefits of the multi-output NN for viscous flows

This example considers the computation of the aerodynamic coefficients on the NACA0012 aerofoil at a free stream Mach number, M_∞, an angle of attack, α, and Reynolds number, Re, in predefined intervals, $I_M = (0.3, 0.9)$, $I_\alpha = (0^\circ, 12^\circ)$ and $I_{Re} = (2, 6) \times 10^6$.

This example is used to compare four strategies using numerical experiments as a function of the number of hidden layers, $\mathtt{n_L}$, and hidden neurons, $\mathtt{n_N}$. The influence of the number of training cases on the accuracy of the predictions is also studied by considering training sets of sizes ranging from $\mathtt{n_{Tr}} = 20$ up to $\mathtt{n_{Tr}} = 320$ cases. The first strategy employs one NN which considers M_∞, α and Re as inputs and one aerodynamic coefficient, the lift, drag or moment, as output. The second strategy also employs one NN which considers M_∞, α and Re as inputs and the three aerodynamic coefficients directly as the outputs. The third strategy employs three NNs and consider M_∞, α and Re as their inputs. The first network considers the pressure coefficient defined at the 223 mesh nodes used to discretise the aerofoil. The second and third networks output the stresses, given as $\tilde{\boldsymbol{\tau}} = \mathbf{T} \cdot n^w$, in the x and y directions respectively. Finally, the fourth strategy employs only one NN which considers M_∞, α and Re as inputs and, the pressure and the stresses in the x and y directions as outputs using a single NN. This leads to three times the number of outputs compared to one of the NN employed in the third strategy.

Latin hypercube sampling is used to train the networks with a dataset of $\mathtt{n_{Tr}} = 320$ simulations performed with the Favre-averaged NS solver [94], in $I_M \times I_\alpha \times I_{Re}$. The test dataset consists of $\mathtt{n_{Te}} = 100$ simulations in $I_M \times I_\alpha \times I_{Re}$.

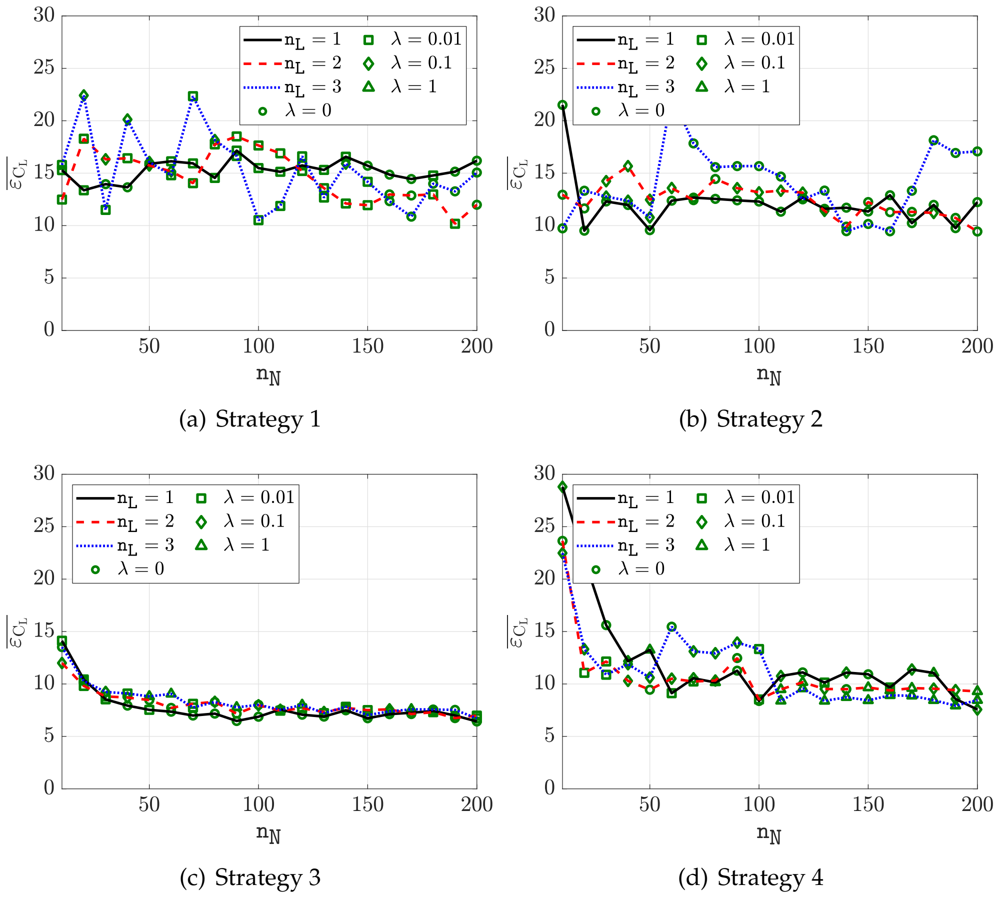


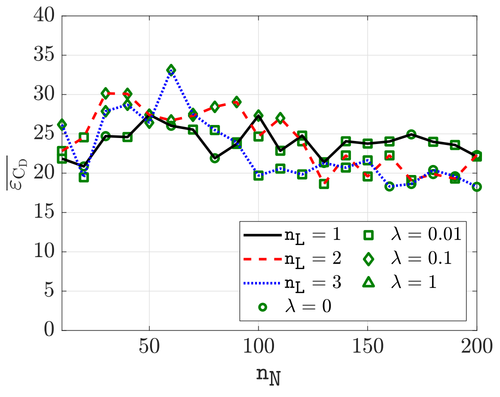
Figure 4.14: Mean value of the errors, measured in lift counts, as a function of the number of hidden neurons, n_N, and the number of hidden layers, n_L.

Figure 4.14 shows the mean value of the error, measured in lift counts, as a function of the number of hidden layers, n_L and hidden neurons, n_N. Figure 4.14(a) shows the results for the first strategy corresponding to a trained NN model with one single output, here the lift. The results show that a small change in the number of neurons can lead to a substantial change in the error when two or three hidden layers are selected. Conversely, one hidden layer tend to be less sensitive to the hidden neurons. The highest accuracy with this configuration is $\overline{\varepsilon_{C_L}} = 10$, obtained with two hidden layers and $n_N = 190$ neurons. Similar accuracy is also obtained with three hidden layers and $n_N = 100$ neurons. Figure 4.14(b) corresponds to the results of the second strategy with three aerodynamic outputs. The results show that the predictions are also highly dependent on both the number of hidden layers and neurons, demonstrating a lack of robustness. The highest accuracy is obtained with multiple NN configurations, such as with one hidden layers and a number of neurons, $n_N = \{20, 50, 170, 190\}$, with two hidden layers and $n_N = \{140, 200\}$ neurons, or with three hidden layers and

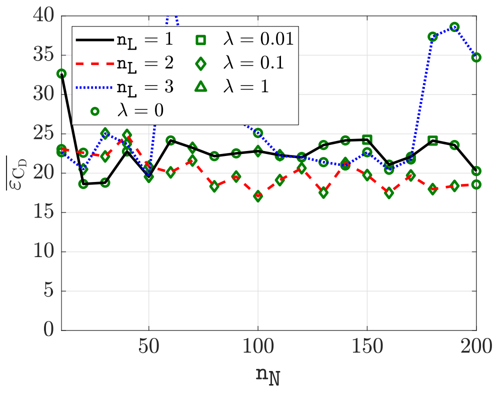
$\mathtt{n_N} = \{10, 140, 150, 160\}$ neurons. It is worth noting that this model cannot produce a prediction with an error less than $\overline{\varepsilon_{C_L}} = 9$ lift counts.

The results for the third strategy, corresponding to the training of three NNs predicting the pressure and the stresses in the x and y direction, are shown in Figure 4.14(c). It can be observed that the accuracy is weakly dependent on the number of neurons in all the three hidden layers and for neurons in the range $\mathtt{n_N} \in [100, 200]$. The highest accuracy is $\overline{\varepsilon_{C_L}} = 6$, obtained with one hidden layer and $\mathtt{n_N} = 90$ neurons. It is worth noting that there are no significant variation in the errors for neurons in the range, $\mathtt{n_N} \in [100, 200]$ and for any number of hidden layers considered. This indicates the robustness of this method. Finally, figure 4.14(d) shows the results of the fourth strategy, corresponding to the predictions of the pressure and the stress vector in the x and y direction using only one NN. It can be observed that the mean value of the error has a higher dependence on the number of hidden layers and neurons than the third strategy, but a lower dependence compared to the first and second strategies. The highest accuracy is obtained with one hidden layer and $\mathtt{n_N} = 200$ neurons. The mean value of the error at this configuration is $\overline{\varepsilon_{C_L}} = 7$. It is worth noting that a larger number of hidden neurons is required to model the pressure and stresses in the same network compared to the third strategy where the quantities are predicted separately using three NNs. Moreover, only 90 hidden neurons or less are required in the third strategy to provide predictions with similar accuracy.

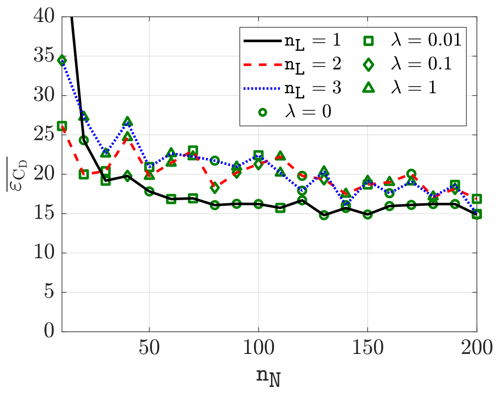
Figure 4.15 shows the mean value of the error, measured in drag counts, as a function of the number of hidden layers, $\mathtt{n_L}$ and hidden neurons, $\mathtt{n_N}$, for the four strategies employed. The first and second strategies in figures 4.15(a) and 4.15(b) are observed to be dependent on the number of hidden layers and hidden neurons. The best accuracy obtained with the first strategy is $\overline{\varepsilon_{C_D}} = 18$ when three hidden layers and $\mathtt{n_N} = 200$ are used. The second strategy provides a mean error of $\overline{\varepsilon_{C_D}} = 17$ when two hidden layers and $\mathtt{n_N} = 100$ neurons are selected. Figure 4.15(c) shows the results of third strategy. It shows that the strategy is almost insensitive to the number of hidden neurons in the range $\mathtt{n_N} \in [100, 200]$ when one hidden layer is employed. The lowest mean value of the error $\overline{\varepsilon_{C_D}} = 15$, is obtained with one hidden layer and $\mathtt{n_N} = 200$ hidden neurons. Similar accuracy can also be obtained with three hidden layers and the same number of hidden neurons. Finally, the fourth strategy produces slightly higher mean errors than the third strategy. Moreover, it is observed that the errors across the three hidden



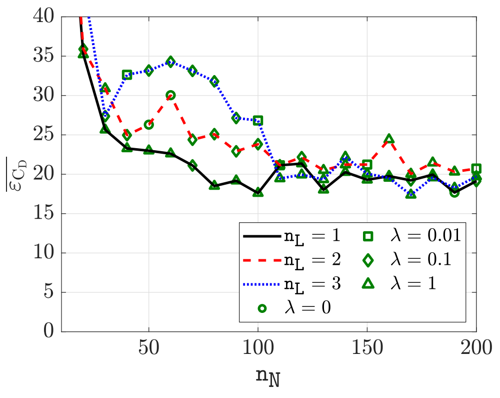
(a) Strategy 1



(b) Strategy 2



(c) Strategy 3



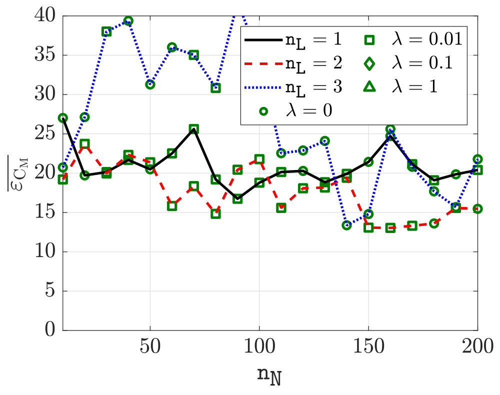
(d) Strategy 4

Figure 4.15: Mean value of the errors, measured in drag counts, as a function of the number of hidden neurons, n_N, and the number of hidden layers, n_L.

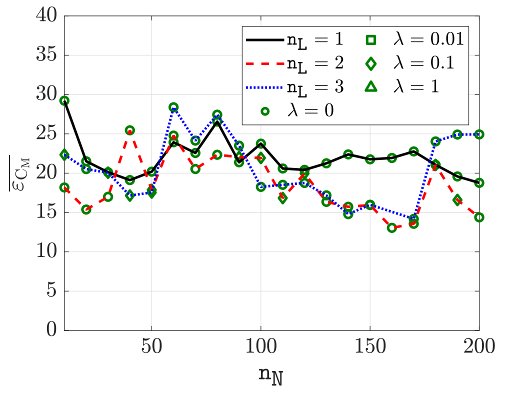
layers do not vary significantly for hidden neurons in the range, $n_N \in [110, 200]$.

Figure 4.16 shows the mean value of the error, measured in moment counts, as a function of the number of hidden layers, n_L and hidden neurons, n_N, for the four strategies employed. The results show that there are no significant difference between the highest accuracies obtained across the four strategies. However, there is a strong dependence on the number of hidden layers and neurons for the first, second and fourth strategies. Conversely, the third strategy is observed to be less sensitive for all the number of hidden layers considered in the higher end of the range of hidden neurons, that is $n_N \in [100, 200]$. The best accuracy, $\overline{\varepsilon_{C_M}} = 12$, is obtained with three hidden layers and $n_N = 120$ neurons. Similar accuracies can also be obtained with one hidden layer and $n_N = 60$ neurons. It is worth noting that for all other strategies, the lowest mean values of the error, measured in moment counts, are obtained with two or three hidden layers and a larger number of hidden neurons, $n_N \in [150, 180]$.

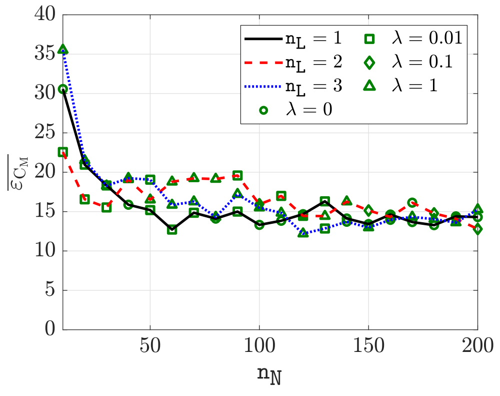
Figure 4.17 compares the accuracy of the four strategies, measured in lift, drag and



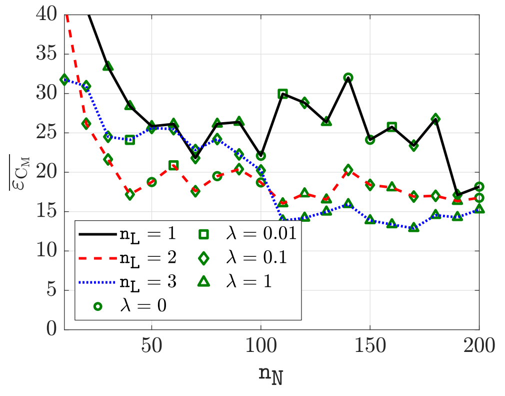
(a) Strategy 1



(b) Strategy 2

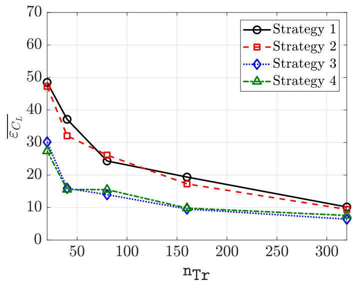


(c) Strategy 3

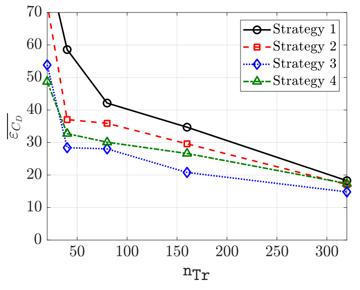


(d) Strategy 4

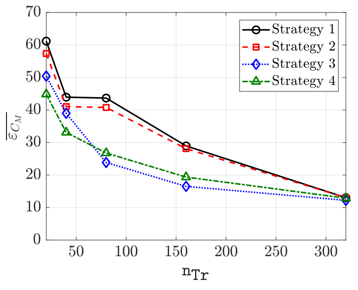
Figure 4.16: Mean value of the errors, measured in moment counts, as a function of the number of hidden neurons, n_N, and the number of hidden layers, n_L.



(a) Lift counts



(b) Drag counts



(c) Moment counts

Figure 4.17: Evolution of the mean error on the test set, measured in counts, as the number of training examples is increased for the four strategies.

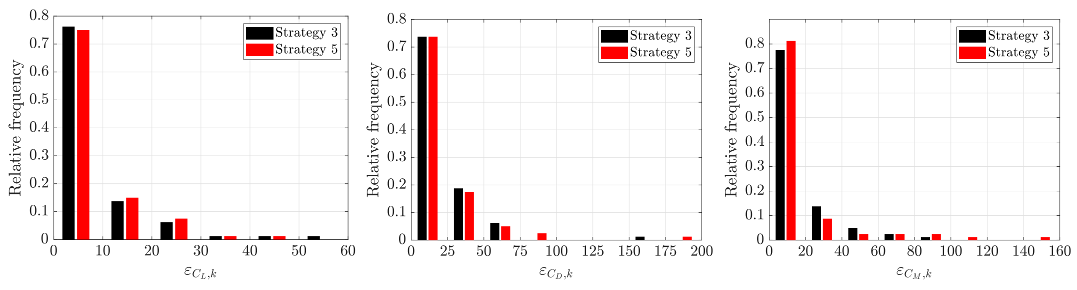
moment counts, as the number of training examples is increased. It is worth noting that the best accuracy of the NNs is reported in terms of the mean error of the predictions in the test set for each number of training examples. Figure 4.17(a) shows the evolution of the mean value of the error, measured in lift counts, as a function of the number of training cases. The results show that as the number of training cases is increased from $n_{Tr} = 20$ to $n_{Tr} = 320$ simulations, the mean error decreases for all four strategies. It can be observed that the first and second strategies provide predictions with similar accuracies at all number of training examples. The best accuracy obtained with the first and second strategies when $\mathtt{n_{Tr}} = 320$ simulations are employed is 10 and 9 lift counts, respectively. A similar deduction can be made for the third and fourth strategies. However, lower errors compared to the first and second strategies are consistently obtained for all number of training examples considered. The highest accuracy obtained with the third and fourth strategies is 6 and 7 lift counts, respectively, when $\mathtt{n_{Tr}} = 320$ simulations are used. It can also be deduced that the multi-output NN used in the third and fourth strategies in this example requires $\mathtt{n_{Tr}} = 320$ simulations to match the accuracy obtained in section 4.2 where $\mathtt{n_{Tr}} = 160$ solutions of the inviscid compressible flow are used to train the NN.

Figure 4.17(b) shows the evolution of the mean value of the error, measured in drag counts, as a function of the number of training cases. The results show that the mean errors decrease as the number of training examples is increased for all four strategies. It can be observed that the first and second strategies provide predictions with larger errors compared to the multi-output NN used in the third and forth strategies, at all number of training examples. The highest accuracy when $\mathtt{n_{Tr}} = 320$ simulations are used in the training, is obtained with the third strategy. It corresponds to a mean error of 15 drag counts. Finally, figure 4.17(c) shows the results measured in moment counts. A similar deduction to the mean errors in lift and drag counts can be made. The third and forth strategy provides a consistent decrease of the mean value of the error as the number of training examples is increased. The highest accuracy of 12 moment counts is obtained with the third strategy when $\mathtt{n_{Tr}} = 320$ simulations are used.

In addition to the strategies explored in this example, another strategy is introduced. It uses a NN which considers the pressure evaluated at the discretised mesh nodes in inviscid flow and the corresponding Reynolds number of each case defined in $I_M \times I_\alpha \times I_{Re}$, as inputs. The outputs of the NN corresponds to the pressure and the stresses in the x and y directions, defined at the mesh nodes on the NACA0012 aerofoil, for the M_∞ and α evaluated in viscous flow. The same training and test datasets of size, $\mathtt{n_{Tr}} = 320$ and $\mathtt{n_{Te}} = 100$ simulations, in $I_M \times I_\alpha$, are obtained by performing steady Euler calculations. This strategy is referred to as *strategy 5* in this section. It is compared with the third strategy where M_∞, α and Re are considered as inputs and three multi-output networks are considered to predict the pressure and the stress distributions

Table 4.5: The selected NN configurations for the predictions of lift using the four strategies considered.

	Strategy			
	1	2	3	4
Input	M, α, Re			
Output	C_L	C_L, C_D, C_M	$C_p / \tilde{\tau}_x / \tilde{\tau}_y$	$C_p, \tilde{\tau}_x, \tilde{\tau}_y$
Hidden layer, $\mathtt{n_L}$	2	1	1	1
Hidden neuron, $\mathtt{n_N}$	190	20	90	200
Overfitting, λ	0.01	0	0	0

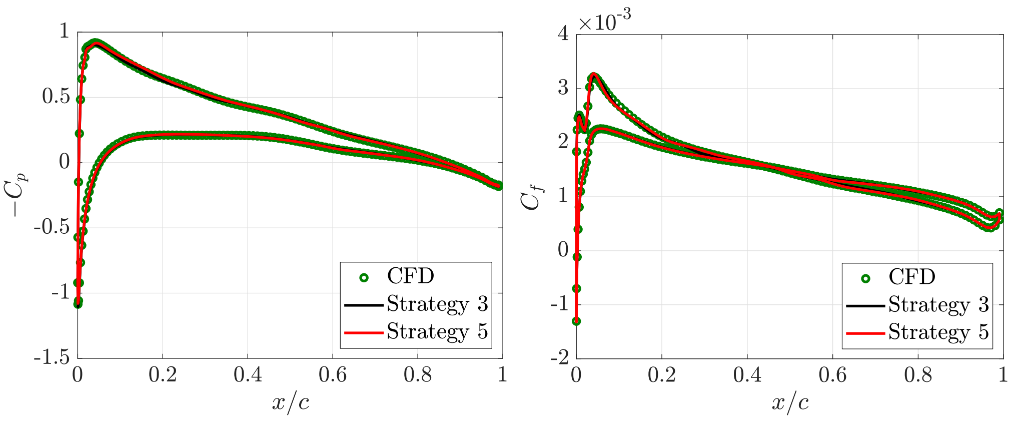


(a) Lift counts (b) Drag counts (c) Moment counts

Figure 4.18: Comparison of the relative frequencies of the error for strategy 3 and 5, measured in counts in the test set.

separately. The third strategy is selected due to its superior performance compared to the three other strategies. Table 4.5 summarises the selected NN configurations for the predictions of lift in the test set using the four strategies considered.

Figure 4.18 compares the relative frequency of the errors of the third and fifth strategies on the test set when $\mathtt{n_{Tr}} = 320$ simulations are used in the training. The NNs in the third strategy employ three hidden layers and 130 hidden neurons, and the NN in the fifth strategy considers two hidden layers and 200 hidden neurons. An overfitting parameter, $\lambda = 1$ is selected for both strategies. The results show that both strategies produce an error below 10 lift counts for 76% and 75% respectively. The highest error obtained in the test set is 41 and 51 lift counts for the third and fifth strategies, respectively. Figure 4.18(b) shows the relative frequency of the errors measured in drag counts. It is observed that both networks have 74% of the test cases with an error below 25 drag counts. The lowest accuracies obtained are 156 and 189 drag counts respectively. Finally, figure 4.18(c) shows the relative frequency of the errors measured in moment counts. There are 78% and 81% of the test cases producing an error below 20 moment counts. Although the fifth strategy has 3% more test cases with a prediction error below 20 moment counts, there is a significant difference in the maximum error between the two. The maximum errors obtained with the third and



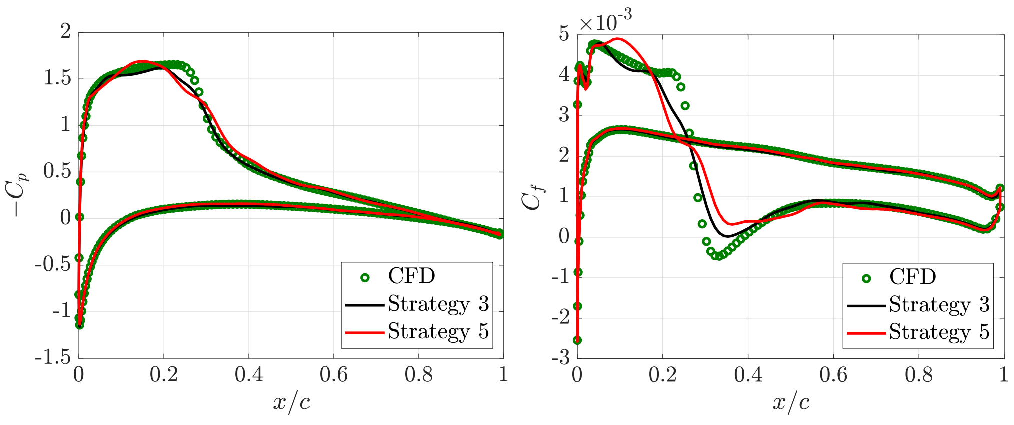
(a) Pressure coefficient (b) Skin friction coefficient

Figure 4.19: Comparison of the pressure and stress distributions obtained with the CFD solver and the predicted distributions from the third and fifth strategies, for a test case in subsonic flow, $M_\infty = 0.49$, $\alpha = 2.2°$ and $Re = 4.2 \times 10^6$.

fifth strategies are 95 and 153 moment counts, respectively.

To further illustrate the performance of the two strategies, figure 4.19 shows the comparison of the pressure and skin friction distributions, obtained with the CFD solver and predicted distributions from the two strategies, for a test case in the subsonic regime. It is worth noting that the skin friction coefficients are calculated after the predictions of the stresses in the x and y directions are made. The results show that both strategies provide accurate predictions for the pressure and stress distributions in the subsonic regime. The NNs are also capable of predicting the skin friction distribution accurately over the aerofoil. There are only minor discrepancies between the two predictions and the CFD data at $x/c = 0.3$.

Figure 4.20 shows the comparison of the pressure and skin friction distributions, obtained with the CFD solver and the predicted distributions from the third and fifth strategies, for a test case in the transonic regime. It is observed that predictions in transonic flows are more challenging. For instance, deviations from the reference CFD results can be observed at $x/c = 0.3$ for both strategies. The NNs are not capable of predicting the sharp gradient of the shock in figure 4.20(a). The fifth strategy show more deviations from the reference results on the upper surface of the aerofoil as well as in the shock region. The deviations from the CFD results are clearer in figure 4.20(b). Both strategies struggle to predict the sharp gradient in the skin friction distribution around $x/c = 0.3$. However, the fifth strategy shows more pronounced deviations than its counterpart for $x/c \in [0.1, 0.5]$.



(a) Pressure coefficient (b) Skin friction coefficient

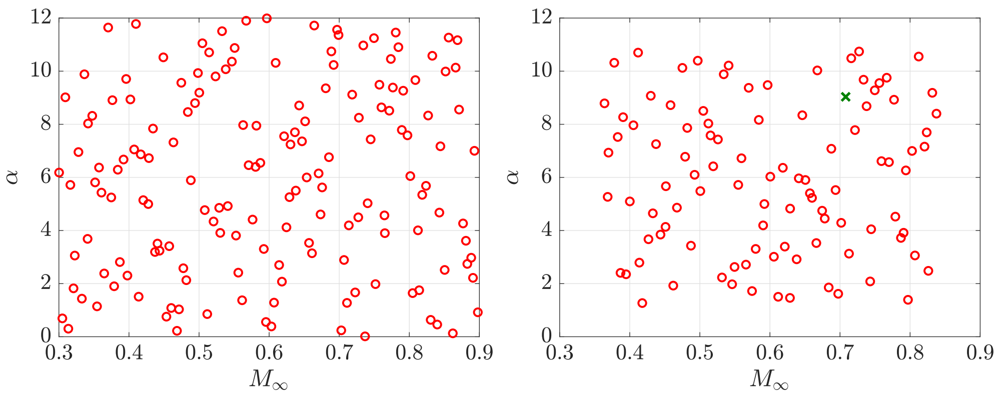
Figure 4.20: Comparison of the pressure and stress distributions obtained with the CFD solver and the predicted distributions from the third and fifth strategies, for a test case in transonic flow, $M_\infty = 0.69$, $\alpha = 4.0°$ and $Re = 3.1 \times 10^6$.

4.4 Influence of the accuracy of the CFD data on the NN predictions

This example considers the computation of the aerodynamic coefficients of the ONERA M6 wing [258] at a free stream Mach number, M_∞, and an angle of attack, α, in predefined intervals $I_M = (0.3, 0.9)$ and $I_\alpha = (0°, 12°)$, respectively.

This example is used to compare the accuracy of the predictions made by three NNs. The first network considers M_∞ and α as inputs and one aerodynamic coefficient, the lift, drag or moment, as a single output. The second NN also considers M_∞ and α as inputs and the three aerodynamic coefficients as outputs. The third network considers the same inputs as the other two networks and the output is the pressure at a user defined set of points on the wing. The set of points considered corresponds to the mesh nodes used to discretise the wing.

The networks are trained using a dataset of $\mathtt{n_{Tr}} = 160$ simulations, obtained by performing steady Euler calculations using the vertex-centred finite volume scheme [94]. Every trained network is then tested using a dataset of $\mathtt{n_{Te}} = 100$ simulations. Both dataset is selected by using the latin hypercube sampling [251] in $I_M \times I_\alpha$. Figure 4.21 shows the sampling space of the training and the test sets. In contrast to figures 4.1, more training data is used to ensure the sampling space is well covered. Moreover, the test set is defined using a smaller interval as $I_M = (0.35, 0.85)$ and $I_\alpha = (1°, 11°)$, so that there is no extrapolation. The volume mesh used in this example consists of



(a) Training set (b) Test set

Figure 4.21: The sampling space used to define the training and test dataset using $\mathtt{n_{Tr}} = 160$ and $\mathtt{n_{Te}} = 100$ simulations respectively. The highlighted cross denotes the test case used to further illustrate the performance of the strategy employed in this section.

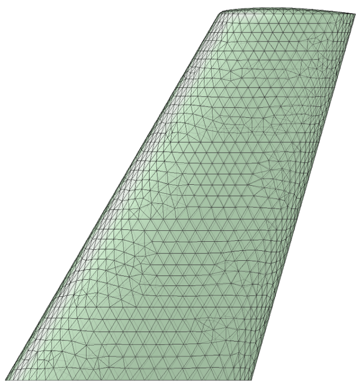
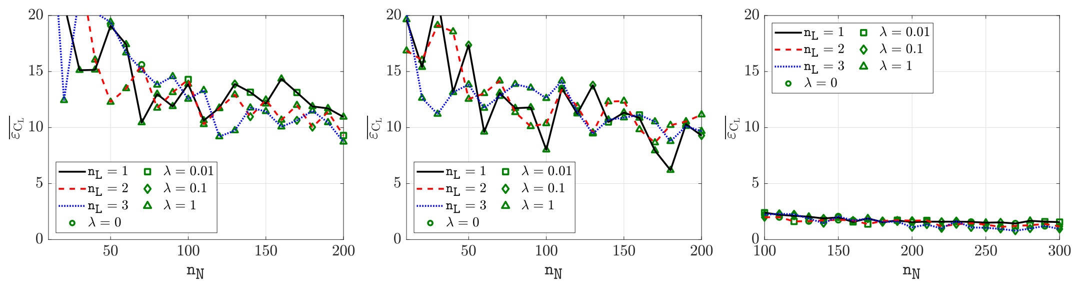


Figure 4.22: The surface mesh used to obtain CFD data in $I_M \times I_\alpha$, consisting of 1,977 nodes and 3,900 triangles.

167,824 elements and 29,025 nodes. Figure 4.22 shows the surface mesh used in this analysis, which consists of 1,977 nodes and 3,900 triangles.

The first numerical experiment explores the accuracy of the predicted aerodynamic quantities as a function of the number of hidden neurons, $\mathtt{n_N}$, and the number of hidden layers, $\mathtt{n_L}$, for all the three networks considered and for different values of the over-fitting parameter, λ. It is worth noting that the third network explores the accuracy on the test set in a higher range of hidden neurons, namely in the interval $I_{\mathtt{n_N}} = (100, 300)$, compared to the first and second network in the interval $I_{\mathtt{n_N}} = (10, 200)$. This is attributed to the fact that the first and second network produce predictions using one and three output neurons, respectively, and the third network has to predict at 1,977 output neurons corresponding to pressure defined at the surface mesh nodes.



(a) Network 1 (b) Network 2 (c) Network 3

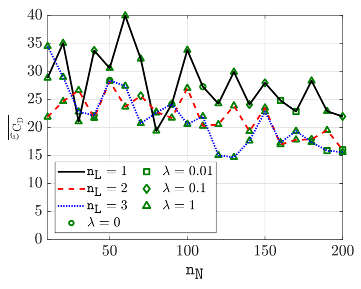
Figure 4.23: Mean value of the lift counts, $\overline{\varepsilon_{C_L}}$, measured in the test set as a function of the number of hidden neurons, $\mathtt{n_N}$, and the number of hidden layers, $\mathtt{n_L}$.

Table 4.6: The selected NN configurations for the prediction of lift using the three networks considered.

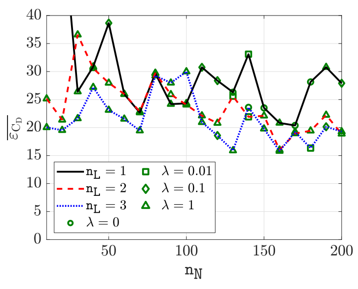
	Network		
	1	2	3
Input	M, α		
Output	C_L	C_L, C_D, C_M	C_p
Hidden layer, $\mathtt{n_L}$	3	1	3
Hidden neuron, $\mathtt{n_N}$	200	180	270
Overfitting, λ	1.0	1.0	0.1

Figure 4.23 shows the evolution of the mean error in the lift as a function of the number of hidden layer, $\mathtt{n_L}$, and hidden neurons, $\mathtt{n_N}$. The results show that the accuracy of the first two networks is comparable, with an error between 10 to 15 lift counts in the majority of cases. The second network is able to provide higher accuracy, with an error near five lift counts, with one hidden layer and $\mathtt{n_N} = 180$. Finally, the third network provides the most accurate results, with an error below three lift counts for any combination of the values of $\mathtt{n_L}$ and $\mathtt{n_N}$. This experiment shows the robustness of the third NN model. Table 4.6 summarises the selected NN configurations for the predictions of lift in the test set using the three networks considered in this example.

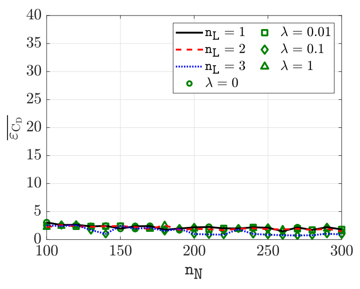
Figure 4.24 shows the mean value of the errors of the three types of trained NNs, measured in drag counts, as function of hidden layers, $\mathtt{n_L}$ and hidden neurons, $\mathtt{n_N}$. The first two networks are observed to have a higher level of oscillations, indicating a strong dependence on the number of hidden neurons and layers. The networks are not able to produce an accuracy lower than 15 drag counts, both achieved with three hidden layers and a comparatively larger number of neurons, namely $\mathtt{n_N} \in [120, 200]$. The third network provides again the most accurate results and it also shows a weak dependence on the hyperparameters.



(a) Network 1

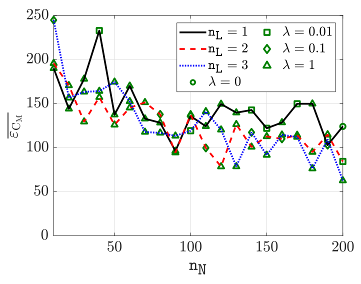


(b) Network 2

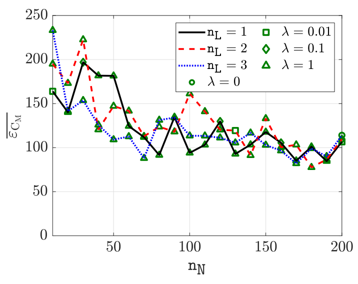


(c) Network 3

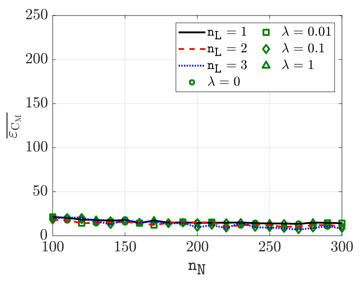
Figure 4.24: Mean value of the drag counts, $\overline{\varepsilon_{C_D}}$, measured in the test set as a function of the number of hidden neurons, $\mathtt{n_N}$, and the number of hidden layers, $\mathtt{n_L}$.



(a) Network 1



(b) Network 2

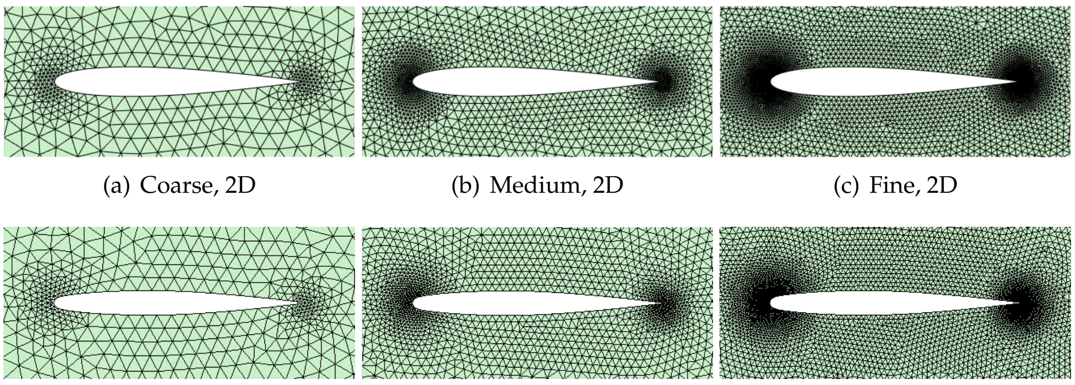


(c) Network 3

Figure 4.25: Mean value of the moment counts, $\overline{\varepsilon_{C_M}}$, measured in the test set as a function of the number of hidden neurons, $\mathtt{n_N}$, and the number of hidden layers, $\mathtt{n_L}$.

Finally, figure 4.25 shows the mean value of the error, measured in moment counts, as a function of the number of hidden layers, $\mathtt{n_L}$ and the number of hidden neurons, $\mathtt{n_N}$ for the three different types of networks considered. Similar conclusions can be drawn as both the first and second networks are observed to have more oscillations with a maximum accuracy of no less than 50 counts for any combination of hyperparameters. In this figure, the superiority of the third network is clearer with a marginally lower error measured in the test set. The highest accuracy for this network leads to an error of only six moment counts, achieved with three hidden layers and $\mathtt{n_N} = 220$ neurons. This corresponds to the same configuration used to predict the lift and drag coefficients.

It is worth noting that the network proposed in this work provides greater accuracy for this three dimensional example, compared to the two dimensional results of previous examples. To better understand this phenomenon, two studies are performed. First, the influence of the accuracy of the CFD data on the accuracy of the NN predictions is studied. Next, the flow features present in the two and three dimensional cases are analysed and compared to a three dimensional example that considers an


(a) Coarse, 2D (b) Medium, 2D (c) Fine, 2D

(d) Coarse, 3D (e) Medium, 3D (f) Fine, 3D

Figure 4.26: Detailed view of the three meshes used for the NACA 0012 aerofoil (top) and on the symmetry plane for the ONERA M6 wing (bottom).

extruded wing, rather than the ONERA M6 swept wing. Only the third network is used because, as previously shown, it provides the most accurate results.

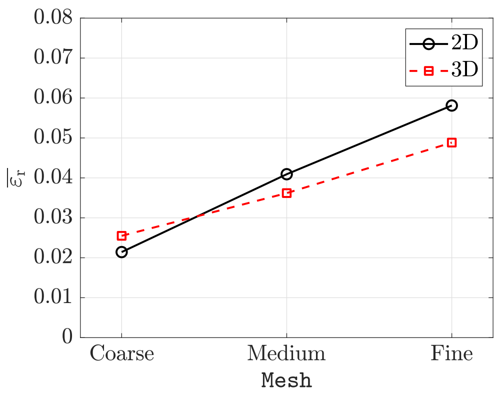
To study the influence of the accuracy of the CFD data on the accuracy of the NN predictions, three levels of mesh refinement are considered in two and three dimensions. Figure 4.26 shows the meshes employed in this example. There are approximately the same number of mesh nodes across any xz-plane in the spanwise direction of the ONERA M6 wing as in the corresponding meshes in two dimensions.

The two dimensional meshes have 1,936, 5,810 and 12,964 elements, and 51, 101 and 167 nodes are used to discretise the aerofoil surface, respectively. The three dimensional volume mesh used previously in this example, represents the coarse mesh. The medium and fine volume meshes in three dimensions have 1,304,444 and 4,485,190 elements, and 7,307 and 16,259 nodes, respectively. For each mesh a set of $\mathtt{n}_{\mathtt{Tr}} = 160$ training cases and a set of $\mathtt{n}_{\mathtt{Te}} = 100$ test cases are generated. The pressure at the nodes used to discretise the aerofoil and wing, in two and three dimensions respectively, is used as the output of the NN model.

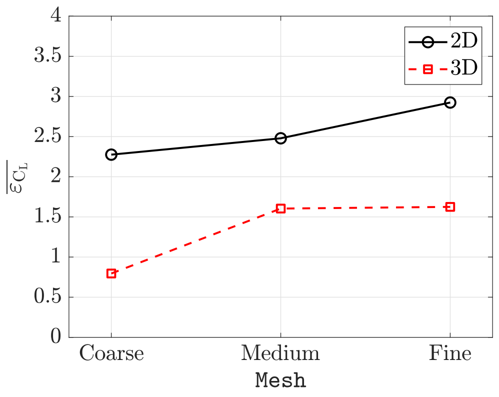
To measure the accuracy of the NNs, the relative error of the pressure coefficient, defined as

$$\varepsilon_{\mathtt{r}} = \left[\frac{\displaystyle\int_{\Gamma} \left[C_p(\boldsymbol{x}) - C_p^{\star}(\boldsymbol{x}) \right]^2 d\Gamma}{\displaystyle\int_{\Gamma} \left[C_p(\boldsymbol{x}) \right]^2 d\Gamma} \right]^{1/2}, \tag{4.1}$$

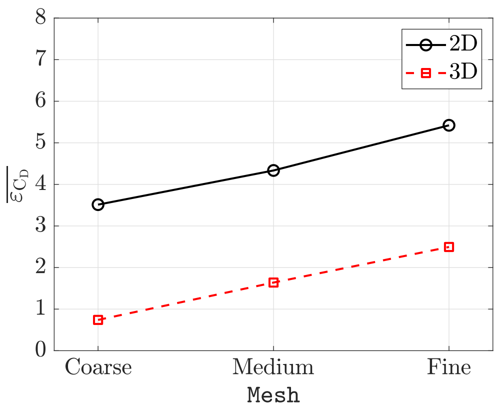
where $C_p(\boldsymbol{x})$ denote the target pressure coefficient distribution over the surface describing the aerofoil/wing, $C_p^{\star}$ denotes the corresponding predicted pressure coefficient and Γ is the surface of the aerofoil/wing. In addition, the accuracy on the three



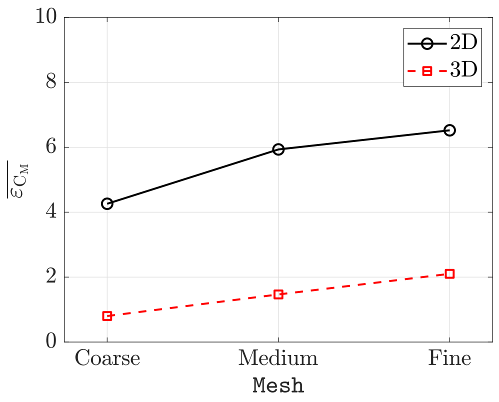
(a) Relative error in C_p



(b) Error in C_L



(c) Error in C_D

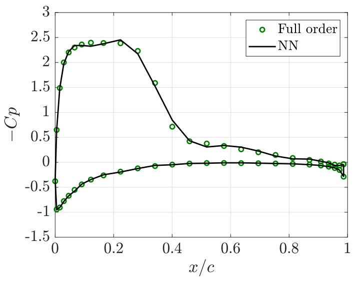


(d) Error in C_M

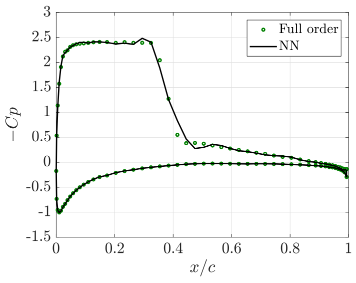
Figure 4.27: Mean value of the errors, measured in the test set, for the three levels of mesh refinement in two and three dimensions.

aerodynamic quantities of interest is also considered. To offer a better comparison between the two and three dimensional results, the moment coefficient in three dimensions is computed about the aerodynamic centre of each section.

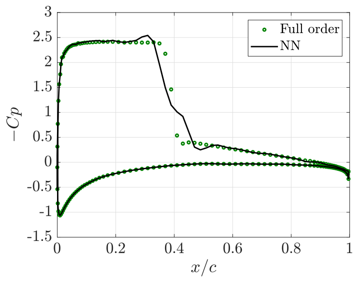
Figure 4.27 shows the comparison of the mean value of the errors in the test dataset. Every point in this figure represents the lowest error obtained by selecting the number of hidden layers, $\mathtt{n}_\mathtt{L}$, the number of hidden neurons, $\mathtt{n}_\mathtt{N}$ and the over-fitting parameter, λ, as done in previous experiments. The results show that the accuracy of the networks when predicting values of the pressure over the aerodynamic shape is almost identical in two and three dimensions. Moreover, it is observed that the error in the prediction increases as the mesh is refined. This behaviour is attributed to the fact that coarser meshes tend to smooth sharp gradients in the solution, such as shocks. As shown in previous studies of sections 4.2, NN produces more accurate predictions when the flow field is smooth (e.g subsonic flow), compared to cases with sharp gradients, (e.g transonic flow).



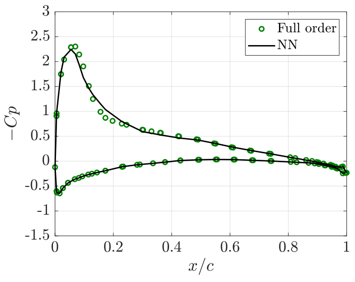
(a) Coarse, 2D



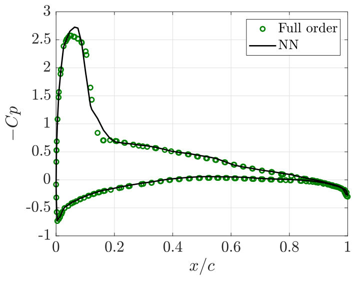
(b) Medium, 2D



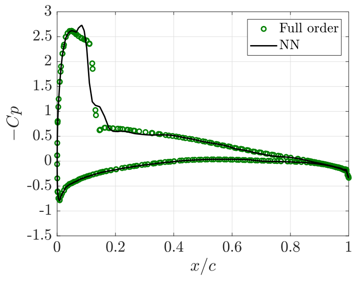
(c) Fine, 2D



(d) Coarse, 3D



(e) Medium, 3D



(f) Fine, 3D

Figure 4.28: Comparison of the pressure coefficient, C_p, obtained with the CFD solver and the predicted C_p using NN for a transonic test case at a free stream Mach number, $M_\infty = 0.71$ and an angle of attack, $\alpha = 9.1°$, for the meshes employed in two and three dimensions.

To further illustrate the difference between the predictions in two and three dimensions, figure 4.28 shows a comparison of the pressure coefficient, C_p, obtained with the CFD solver and the predicted C_p using NN for a test case in the transonic regime, highlighted with a cross in figure 4.21(b). The results show how the shock is smeared leading to a smooth variation of the pressure that is accurately predicted by the NN. As the level of mesh refinement is increased, the shock becomes sharper and the prediction made by the NN presents some overshoots and undershoots near the shock. It can therefore be deduced that as the meshes become finer, the predicted solution from the trained models may contain oscillations near regions of sharp gradients, such as shocks. However this may or may not affect the results measured in the count metrics as it is a calculated integral computed on the surface of the aerofoil or wing.

When the error is measured in the lift, drag or moment, the results of figure 4.27 show that more accurate predictions are obtained in three dimensions. This behaviour is attributed to the different flow features induced by the two dimensional NACA0012 aerofoil (equivalent to an extruded three dimensional wing) and the ONERA M6, which is a swept wing. To confirm this hypothesis, the second level of mesh refinement is considered and the NACA0012 profile is used to built an extruded wing in three dimensions. The generated mesh maintains the same spacing over the wing to

Table 4.7: Mean value of the errors, measured in the test dataset, for the medium level of mesh refinement when $\mathtt{n_{Tr}} = 160$ simulations are used in the training of the third network.

Mesh	$\overline{\varepsilon_{\mathtt{r}}}$	$\overline{\varepsilon_{C_L}}$	$\overline{\varepsilon_{C_D}}$	$\overline{\varepsilon_{C_M}}$
2D	0.041	2.5	4.3	6.0
3D-X	0.042	2.6	4.4	6.1
3D	0.036	1.6	1.6	1.7

ensure that the results between the extruded geometry and the two dimensional case can be compared.

Table 4.7 shows the comparison of the mean value of the errors obtained from three meshes when the same training and test datasets, as shown in figure 4.21, are employed. The results show that the accuracy of the predictions for the two dimensional case is almost identical to the three dimensional case with the extruded wing (3D-X). For the three dimensional swept wing, the accuracy is higher. A further analysis of the flow features indicates that the strength of the shock in the simulation with the ONERA M6 wing is not as strong as in the two dimensional and three dimensional extruded wing. In addition, the higher accuracy obtained in the ONERA M6 predictions is also attributed to the fact that swept wings are known to delay the appearance of shocks. Hence, more cases in the datasets contain smoother solutions for the wide range of flow conditions considered in this example and thus lower mean values of the errors in the predictions are obtained.

A summary of the findings in this chapter follows. The accuracy of the predictions using the proposed multi-output NN has been shown to be higher when compared to a NN where the output consists of the three aerodynamic coefficients. The results also show the superiority when compared to a NN where only one aerodynamic coefficient is predicted. The superior performance has been demonstrated for both two and three dimensional examples involving free stream Mach number and the angle of attack as the inputs of the NNs. Moreover, the performance of the proposed NN has been compared against NNs which predict pressure at various regions in the computational domain to defend the choice of the number of outputs in the NN. It was deduced that the NNs predict with similar accuracies. However, the NNs which predict at nodes other than the surface nodes are more expensive to train as more weights in the network need to be optimised.

In addition, the proposed NN has also been compared to the POD with RBF and the results show the better performance of the NN, especially when predicting flows that involve shocks. The superior accuracy of the proposed NN has been shown against POD coupled with a NN. It was deduced that the due to the global nature of the NN, the POD coupled with the NN is inferior to both the proposed NN and the POD with RBF. An extensive set of numerical studies has been presented to show the effect of the NN hyperparameters and numerical parameters of the POD on the accuracy of the predictions. Furthermore, the effect of increasing the number of cases in the training set on the accuracy of both the NNs and the POD has also been studied.

A NN was also trained to predict the viscous flow solutions using a database of inviscid flow solutions. The network considers the pressure obtained by solving the Euler equations and defined at the mesh nodes discretising the aerofoil as the inputs. The output consists of pressure and stresses obtained using the Favre-averaged NS solver and also defined at the mesh nodes discretising the aerofoil surface. The trained NN model can be employed to reduce the computational effort of solving the NS equations as only the Euler equation need to be resolved, leading to accurate viscous predictions.

The potential of the proposed NN has also been demonstrated for three dimensional examples involving flow conditions. The influence of the accuracy of the CFD data on the accuracy of the NN predictions has been studied in two and three dimensions. It was found that the higher the accuracy in the CFD data, the more challenging it is for the NNs to perform accurate predictions. This is attributed to the stronger shocks present in finer meshes due to the increased resolution of the flow field. The accuracy of the predictions in two and three dimensions has also been compared. It has been shown that for a swept wing the NN offers a greater accuracy when compared to a two dimensional aerofoil. This is explained by the weaker shocks in three dimensions and the delayed appearance of shocks in swept wings, leading to a greater set of training cases containing smooth solutions.

Chapter 5

Aerodynamic predictions using geometric parameters

The present chapter demonstrates the application of ROMs for the fast predictions of aerodynamic coefficients in numerical examples involving geometric parameters. Four examples at various levels of complexity are considered. The first example compares the performance of the proposed NN against the POD coupled with RBFs by using two dimensional inviscid compressible data at the subsonic and transonic flow conditions in the training. The influence of the number of training examples on the accuracy of the NN predictions is also explored. The second example shows the comparison of the performance of the proposed NN against existing NNs by using two dimensional viscous compressible data in the training. Moreover, the influence of the number of training cases on the accuracy of the NN predictions is also explored for all the strategies. The third example performs the comparison of the proposed NN against existing NNs for the predictions of aerodynamic coefficients on three dimensional wings using the solutions of steady Euler calculations in the training. Finally, an example on deforming wings over a wide range of flow conditions is demonstrated to exemplify the robustness of the proposed NN method.

5.1 Comparison of NN with the POD

The first example of this chapter involves the prediction of the lift coefficient for an aerofoil that is parametrised using the control points of the NURBS describing the aerofoil. The base geometry corresponds to an approximation of the NACA0012 aerofoil using two cubic B-splines with eight control points to define the top and bottom

Table 5.1: Control points of the top curve of the base geometry approximating a NACA0012 aerofoil.

i	1	2	3	4	5	6	7	8
x_i	0.500	0.297	-0.003	-0.317	-0.437	-0.474	-0.496	-0.500
y_i^+	0.000	0.030	0.061	0.062	0.040	0.030	0.013	0.000

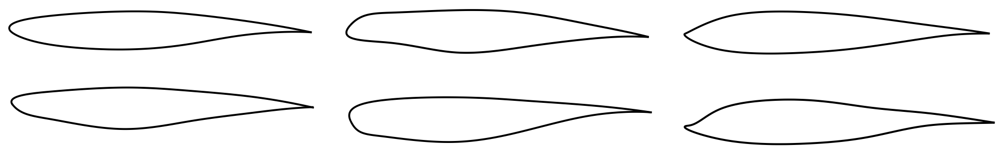


Figure 5.1: Examples of aerofoil shapes generated with NURBS.

curves respectively, as proposed in [95]. The knot vector of both NURBS is given by

$$\Lambda = \{0, 0, 0, 0, 0.615, 0.904, 0.941, 0.979, 1.019, 1.019, 1.019, 1.019\} \tag{5.1}$$

and the set of control points for the top curve $\{\mathbf{P}_i^+ = (x_i, y_i^+)\}_{i=1,\dots,8}$ is detailed in table 5.1. The control points for the bottom curve are symmetrically located with respect to the x axis and are defined as, $\mathbf{P}_i^- = (x_i, y_i^-)$ with $y_i^- = -y_i^+$, for $i = 1, \dots, 8$.

The end control points of both curves are fixed, i.e. $\mathbf{P}_1^- = \mathbf{P}_1^+ = (0.5, 0)$ and $\mathbf{P}_8^- = \mathbf{P}_8^+ = (-0.5, 0)$. In addition, the control point $\mathbf{P}_2^-$ is restricted to be aligned with $\mathbf{P}_1^- = \mathbf{P}_1^+$ and $\mathbf{P}_2^+$. More precisely, it is assumed that $\mathbf{P}_2^- = \mathbf{P}_1^+ + \mu(\mathbf{P}_1^+ - \mathbf{P}_2^+)$ with μ being a parameter defined in $[0.5, 1.5]$. This restriction ensures $\mathcal{G}^1$ continuity of the aerofoil geometry and leads to a problem with 25 independent geometric parameters. The variation of the position of the control points with respect to the base geometry, namely $(\pm\delta x_i, \pm\delta y_i)$ is considered to be the input of the NN, where $(\delta x_i, \delta y_i) \in [0, 0.1c]^2$ and c denotes the chord of the aerofoil. Figure 5.1 shows examples of aerofoil shapes generated by varying the positions of the control points of the NURBS in the range considered.

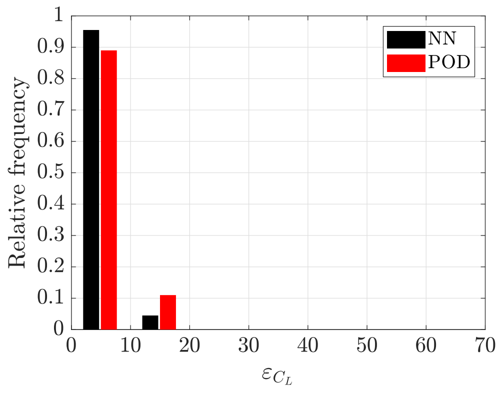
Two flow conditions are considered to explore the performance of the proposed NN and the POD with RBFs for subsonic and transonic flows using inviscid data. The subsonic case corresponds to a free-stream Mach number $M_\infty = 0.4$ and an angle of attack $\alpha = 2°$, whereas the transonic case corresponds to $M_\infty = 0.8$ and $\alpha = 2°$.

As before, the trained ROM with the lowest mean value of the error, measured in counts, in the test set is selected by performing a numerical experiment to explore the accuracy of the predicted aerodynamic coefficients as a function of the hyperparameters of the employed ROM. For instance, in the case of NN there is the choice of the number of hidden neurons, $\mathtt{n_N}$, the number of hidden layers, $\mathtt{n_L}$ and the overfitting parameters, λ. The NN is trained for each combination of the three listed hyperparameters in a predefined range and, the quadratic cost function is implemented with the regularisation term, as in equation (3.53) is employed to evaluate the errors in the training and to avoid the need to divide the dataset into validation and training cases. The NN is trained 10 times for each combination of hidden layer, hidden neurons and the overfitting parameter. This is performed in an attempt to obtain a trained model that is not stuck in a local minimum. The weights of the trained model with the lowest value of the error measured in the test set is recorded and the next combination of hyperparameters is trained. The trained model with the lowest mean value of the error, measured in counts, is selected and compare with other trained ROMs.

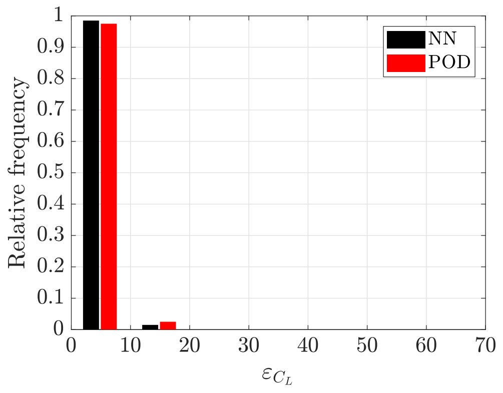
The number of training cases, $\mathtt{n_{Tr}}$, varies from 400 to 1,600 and 200 test cases are considered. Both the training and test sets are obtained by using the latin hypercube sampling. Following the numerical experiment performed on the number of hidden layers and hidden neurons, the NN employs a single hidden layer with 98 neurons and the over-fitting parameter is selected as $\lambda = 0$. For the POD approach, all the snapshots are used in the interpolation. It has been found that, contrary to the example involving a wide range of flow conditions, this option leads to the most accurate results here. This is attributed to the fact that the training cases do not contain a change of regime from subsonic to transonic flow. Additionally, the performance of the two ROMs employed in this example is not compared against *POD-NN* in section 4.2, as it was demonstrated to be inferior to both methods.

Figure 5.2 compares the performance of the proposed NN and the POD with RBFs for both subsonic and transonic flows. The histograms compare the relative frequency of the error, measured in lift counts, for all the test cases using the proposed NN and the POD method. In all the experiments performed, the proposed NN outperforms the POD with RBFs.

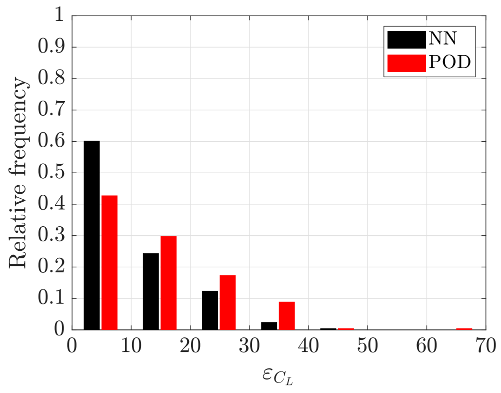
For the subsonic example and using $\mathtt{n_{Tr}} = 400$ cases, the lift is predicted with an error below 10 lift counts in 96% of cases with the NN and in 89% of cases with the POD. When the number of training cases or snapshots is increased to $\mathtt{n_{Tr}} = 1,600$, the lift is predicted with an error below 10 lift counts in 99% of cases with the NN and in 98%



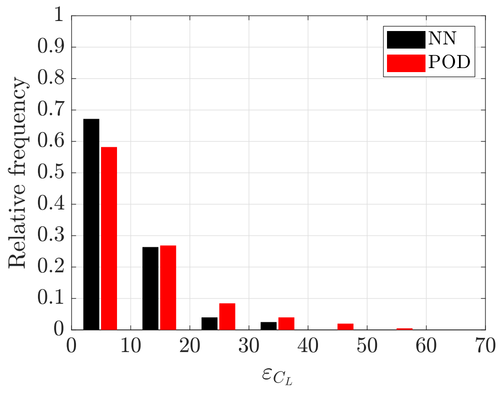
(a) Subsonic, $\mathtt{n_{Tr}} = 400$



(b) Subsonic, $\mathtt{n_{Tr}} = 1,600$



(c) Transonic, $\mathtt{n_{Tr}} = 400$

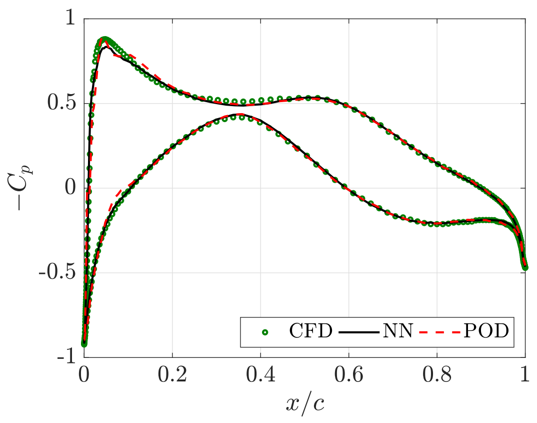


(d) Transonic, $\mathtt{n_{Tr}} = 1,600$

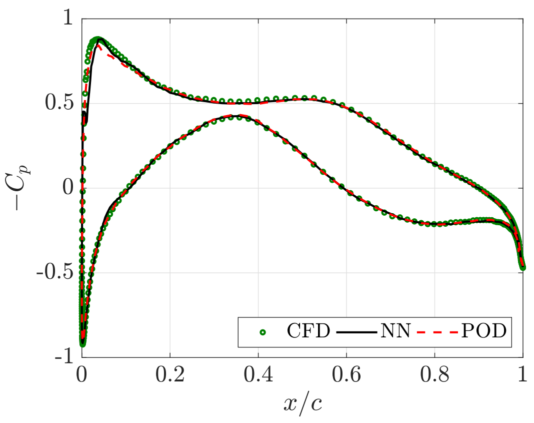
Figure 5.2: Relative frequency of the error on the test set, measured in lift counts, for the proposed NN and the POD for subsonic and transonic flows and with different number of training cases, $\mathtt{n_{Tr}}$.

of cases with the POD. In both cases, the worst predictions made by both the NN and the POD correspond to cases with an error below 20 lift counts.

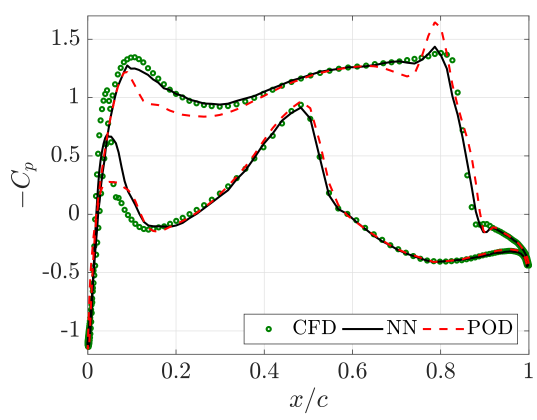
The transonic example is much more challenging, as it requires an accurate prediction of the shock position. Using $\mathtt{n_{Tr}} = 400$ cases, the lift is predicted with an error below 10 lift counts in 60% of cases with the NN and in 43% of cases with the POD. The number of cases that are predicted with an error between 10 and 30 lift counts with both the NN and the POD are 37% and 47% respectively. The main difference between the NN and POD approaches is that, with the NN only 6 predictions have an error higher than 30 lift counts, whereas with the POD, 20 predictions have an error higher than 30 lift counts. In addition, the maximum error with the POD is 20 lift counts higher than the maximum error of the NN. Using $\mathtt{n_{Tr}} = 1,600$ cases, the lift is predicted with an error below 10 lift counts in 68% of cases with the NN and in 58% of cases with the POD. Once more, the main difference is that the maximum error is 37 and 58 lift counts for the NN and the POD approaches respectively.



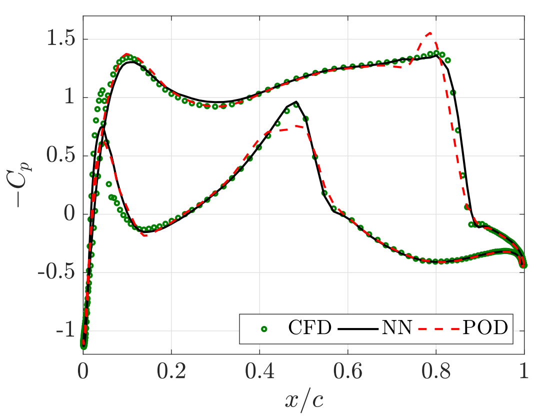
(a) Subsonic, $\mathtt{n_{Tr}} = 400$



(b) Subsonic, $\mathtt{n_{Tr}} = 1,600$



(c) Transonic, $\mathtt{n_{Tr}} = 400$



(d) Transonic, $\mathtt{n_{Tr}} = 1,600$

Figure 5.3: Pressure coefficient over the aerofoil of figure 5.4 compared to the predictions by the NN and the POD.

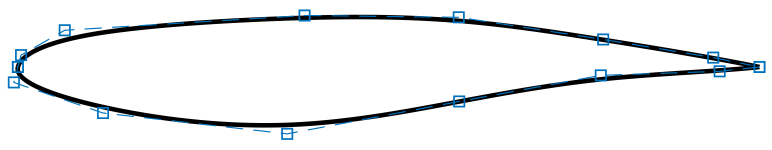


Figure 5.4: Geometry of the aerofoil chosen to compare the NN and POD prediction capability in figure 5.3. The squares denote the control points of the NURBS and the discontinuous line is the control polygon.

To illustrate the performance of the NN and POD approaches, figure 5.3 compares the predicted pressure distribution over the aerofoil configuration shown in figure 5.4 for a subsonic and a transonic case, both using the proposed NN and the POD and for different number of training cases. The results in figures 5.3(a) and 5.3(b) clearly show that both the NN and POD approaches are capable of producing accurate predictions of the pressure coefficient in a subsonic regime. For the transonic regime, when $\mathtt{n_{Tr}} = 400$ there is a sizeable difference between the POD results and the reference results. This is particularly noticeable in the oscillatory character of the prediction near the strong shock on the upper curve. The discrepancy is also visible in the pressure coefficient distribution over the upper curve near the leading edge. The NN is capable of predicting the shock much better without showing oscillations and it also produces a better approximation of the pressure coefficient distribution near the leading edge. Both the NN and the POD show some discrepancies with respect to the reference results for the pressure coefficient distribution over the lower curve near the leading edge. When the number of training cases is increased to $\mathtt{n_{Tr}} = 1,600$, the POD still shows a poor approximation near the strong shock on the top curve as well as near the weaker shock in the bottom curve, whereas the NN shows a very good agreement with respect to the reference results.

5.1.1 Inverse shape design for a target pressure distribution

This section presents an application of the proposed NN to inverse shape design. The goal is to show the potential and reliability of the proposed NN for the fast evaluation of the objective function in an optimisation process.

A pressure coefficient distribution, defined over the 300 points used to discretise an aerofoil, is considered as the target. The problem consists of finding the geometric configuration, given by the position of the control points of a set of two NURBS describing the aerofoil, that leads to the given pressure coefficient distribution. To recall, the two NURBS describing the aerofoil are defined as cubic B-Splines with eight control points, denoted by $\{\mathbf{P}_i^+\}$ and $\{\mathbf{P}_i^-\}$ for the top and bottom curve respectively and for $i = 1, \dots, 8$. It is worth noting that, as described in section 5.1, $\mathbf{P}_1^- = \mathbf{P}_1^+ = (0.5, 0)$ and $\mathbf{P}_8^- = \mathbf{P}_8^+ = (-0.5, 0)$ and $\mathbf{P}_2^-$ is restricted to be aligned with $\mathbf{P}_1^- = \mathbf{P}_1^+$ and $\mathbf{P}_2^+$.

The objective function is defined as

$$f(\{\mathbf{P}_i^{\pm}\}) = \frac{\displaystyle\int_0^c \left(\left[C_p^+ - C_p^\star(\{\mathbf{P}_i^+\}) \right]^2 + \left[C_p^- - C_p^\star(\{\mathbf{P}_i^-\}) \right]^2 \right) d\ell}{\displaystyle\int_0^c \left(\left[C_p^+ \right]^2 + \left[C_p^- \right]^2 \right) d\ell}, \tag{5.2}$$

where C_p^+ and C_p^- denote the pressure coefficient distribution over the top and bottom curves describing the aerofoil, respectively. Analogously, $C_p^\star(\{\mathbf{P}_i^+\})$ and $C_p^\star(\{\mathbf{P}_i^-\})$ denote the predicted pressure coefficient distribution over the top and bottom curves describing the aerofoil, respectively. In this work, the NN built in section 5.1 is employed to predict the pressure distribution for a given set of control points. This is in contrast with traditional approaches where, each evaluation of the objective function requires not only the running of the CFD solver, but also the generation of a mesh for each geometric configuration.

The minimisation problem is formally stated as

$$\begin{aligned}
\min_{\mathbf{P}_2^+,\dots,\mathbf{P}_7^+,\mu,\mathbf{P}_3^-,\dots,\mathbf{P}_7^-} \quad & f(\{\mathbf{P}_i^\pm\}_{i=1,\dots,8}) \\
\text{subject to} \quad & \mathbf{P}_1^- = \mathbf{P}_1^+ = (0.5, 0) \\
& \mathbf{P}_8^- = \mathbf{P}_8^+ = (-0.5, 0) \\
& \mathbf{P}_2^- = \mathbf{P}_1^+ + \mu(\mathbf{P}_1^+ - \mathbf{P}_2^+) \\
& \mu \in [0.5, 1.5],
\end{aligned} \tag{5.3}$$

where the target pressure distribution corresponds to the RAE2822 aerofoil.

The modified cuckoo search (MCS), proposed in [75], is used to solve the optimisation problem (5.3). This method belongs to the family of gradient-free optimisation algorithms and it has demonstrated a better performance when compared to other optimisation algorithms, especially in the context of high dimensional problems. The parameters selected for the MCS algorithm, taken from [75], are 0.75 for the fraction of nests to be discarded, 0.5 for the power step size of the random walk, 1 for the number of eggs to discard in every generation, 40 for the number of eggs in the first generation and 200 for the total number of generations.

To study the influence of the randomness of the MCS algorithm, figure 5.5 shows the statistics of the value for the objective function after 100 executions of the MCS optimisation are performed. The minimum, maximum, mean and standard deviation of the value of the objective function are represented for an increasing number of cases used for training the NN. The results show that the randomness in the MCS is only relevant for a relatively low number of training cases, $\mathtt{n}_{\mathtt{Tr}} = 400$ in this example. For $\mathtt{n}_{\mathtt{Tr}} \geq 800$, the influence of the randomness in the MCS is minimal and both the minimum and maximum value of the objective function is below 0.02, meaning a discrepancy below 2% between the predicted and the target pressure coefficient distribution over the aerofoil.

Figure 5.6 shows the result of the inverse shape design. Figure 5.6(a) offers a visual comparison of the target geometry, the initial geometry used in the MCS algorithm

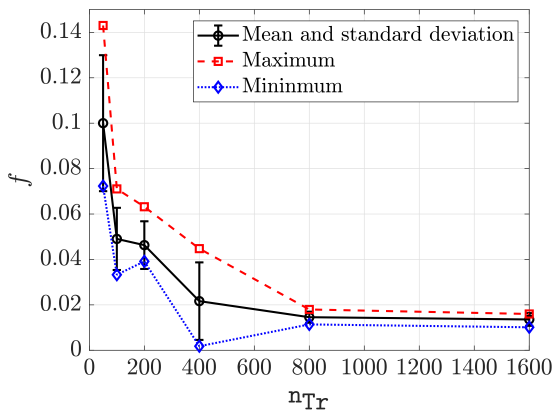
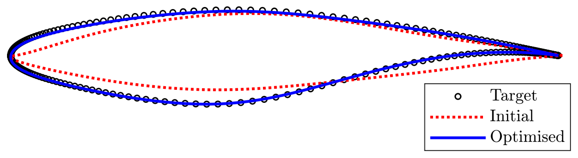
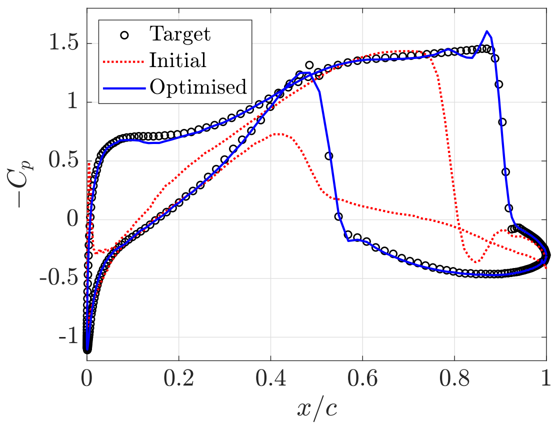


Figure 5.5: Value of the objective function after the minimisation problem is solved with the MCS as a function of the number of cases used for training the NN.



(a) Geometry



(b) Pressure coefficient

Figure 5.6: Target, initial and optimised geometry and pressure coefficient obtained using the MCS and the proposed NN to predict the values of the objective function.

and the final optimised geometry. A very good agreement is observed between the optimised and the target geometry. Figure 5.6(b) shows the pressure coefficient, again for the target geometry, the initial geometry used in the MCS algorithm and the final optimised geometry. As previously mentioned, the relative error in the $\mathcal{L}^2((0,c))$ norm, as measured by the objective function, is below 2%. It is worth mentioning that the optimisation process using the MCS algorithm took only 55 seconds, due to the almost negligible cost of evaluating the objective functions using the proposed NN. Conversely, traditional optimisation frameworks, where the optimisation algorithm is employed directly with the CFD solver, can take several hours to obtain a solution to the inverse design problem.

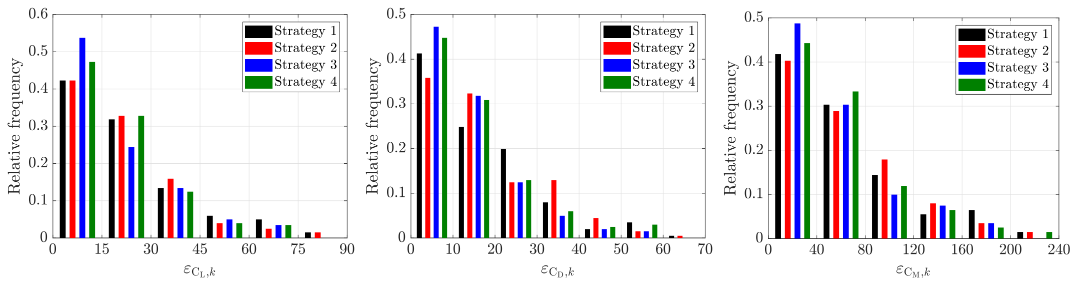
5.2 Benefits of the multi-output NN for viscous flows

This example considers the computation of the aerodynamic coefficients of an aerofoil that is parametrised using the control points of the NURBS describing the aerofoil. It is an extension of section 5.1, performed in viscous flow. There are 25 geometrical parameters used to define the NURBS curves. The variation of the position of the control points with respect to the base geometry, namely $(\pm\delta x_i, \pm\delta y_i))$ is considered to be the input of the NN, where $(\delta x_i, \delta y_i) \in [0, 0.1c]^2$ and c denotes the chord of the aerofoil.

The same design spaces defined in section 5.1 are used, namely, $\mathtt{n}_{\mathtt{Tr}} = 1{,}600$ and $\mathtt{n}_{\mathtt{Te}} = 200$. The cases are simulated at a free stream Mach number, $M_\infty = 0.8$, an angle of attack, $\alpha = 2^\circ$ and Reynolds number, $Re = 2 \times 10^6$, corresponding to flows in the transonic regime. The CFD solver uses the Favre-averaged NS solver, implemented with the SA turbulence model to obtain the viscous compressible solutions.

Four strategies are considered and the performance are compared in terms of counts for the three aerodynamic coefficients. All four strategies employ NNs and consider the 25 geometric parameters as inputs. The first strategy employs one NN that predicts one aerodynamic coefficient, the lift, drag or moment, as output. The second strategy also employs one NN and has all the three aerodynamic coefficients as outputs. The third strategy considers three separate NNs which predicts the pressure coefficient C_p, and the stresses in the x and y direction, $\tilde{\tau}_x$ and $\tilde{\tau}_y$, respectively, at a user defined set of points on the aerofoil. This set of point corresponds to 300 points used to define the aerofoil and is different to the nodes used to discretise the aerofoil in the computational mesh. Finally, the fourth strategy considers only one network which predicts the pressure and stresses on the aerofoil surface, leading to three times the number of outputs compared to a NN in the third strategy.

Figure 5.7 shows the relative frequency of the error for the four strategies considered, measured in counts on the test set. Figure 5.7(a) corresponds to the error measured in the lift. As before, a numerical experiment is performed as a function of the number of hidden layers and hidden neurons for different values of the overfitting parameters,



(a) Lift counts (b) Drag counts (c) Moment counts

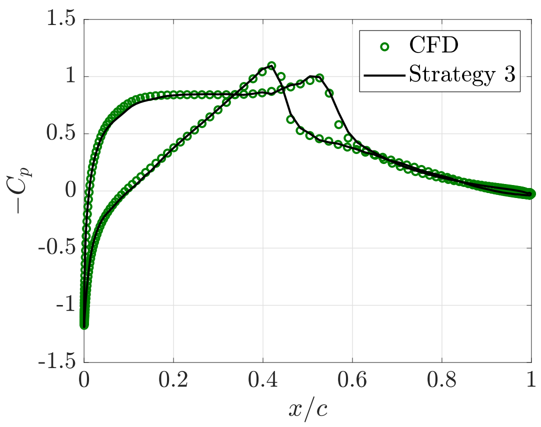
Figure 5.7: Relative frequency of the error on the test set for the four strategies employed, measured in counts.

Table 5.2: The selected NN configurations for the predictions of lift using the four strategies considered.

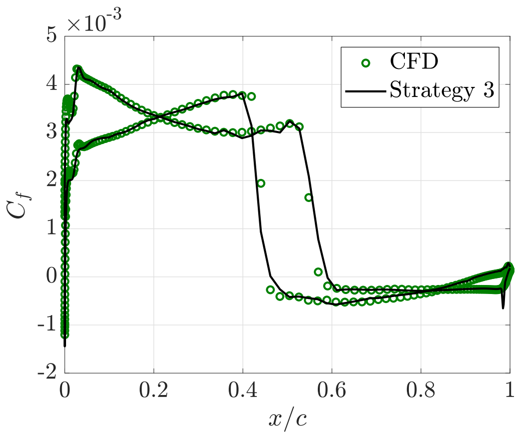
	Strategy			
	1	2	3	4
Input	$\mathbf{P}^{\pm}$			
Output	C_L	C_L, C_D, C_M	$C_p/\tilde{\tau}_x/\tilde{\tau}_y$	$C_p, \tilde{\tau}_x, \tilde{\tau}_y$
Hidden layer, $\mathtt{n}_\mathtt{L}$	2	2	1	1
Hidden neuron, $\mathtt{n}_\mathtt{N}$	80	20	150	200
Overfitting, λ	0	0	0	0

and the NN configuration with the lowest error is reported. The NN in the first strategy is selected to have two hidden layers and $\mathtt{n}_\mathtt{N} = 80$ neurons. The second strategy employs two hidden layers and $\mathtt{n}_\mathtt{N} = 20$ neurons. The third strategy employs one hidden layer and $\mathtt{n}_\mathtt{N} = 150$ hidden neurons, and finally, the fourth strategy employs one hidden layer and 200 hidden neurons. The overfitting parameter is set as $\lambda = 0$ for all four strategies. Table 5.2 shows a summary of the selected NN configurations for the predictions of lift in the test set using the four strategies considered. The results show that the first and second strategies predict with an error below 15 lift counts for about 42% of the test cases. The fourth strategy performs slightly better with 47%. The third strategy provides the most accurate results, with 54% of the test cases below 15 lift counts. The first and second strategies predict a maximum error above 75 lift counts, compared to the multi-output strategies which predicts with a maximum error of 74 and 72 lift counts, respectively.

Figure 5.7(b) shows the relative frequency of the errors measured for the drag. The network in the first strategy is selected to have one hidden layer and 30 hidden neurons. The NNs in the other strategies maintain the same configurations. The results show that the third strategy provides the most accurate results, with nearly 50% of the test cases with an error below 10 drag counts, compared to 42%, 36% and 45% of



(a) Pressure coefficient



(b) Skin friction coefficient

Figure 5.8: Comparison of the pressure and stress distributions obtained with the CFD solver and the predicted distributions from the third strategy, for an example case in the test set.

the cases for the first, second and fourth strategies, respectively. Finally, figure 5.7(c) shows the relative frequency of the errors measured for the moment. The first strategy employs two hidden layers and 100 neurons. The results again show the superiority of the third strategy with nearly 50% of the test cases producing an error below 40 lift counts.

Figure 5.8 compares the pressure and skin friction distributions obtained with the CFD solver and the predicted distributions from the third strategy for a test case. This example corresponds to the aerofoil shape shown in figure 5.4. Figure 5.8(a) shows comparison of the predicted pressure distribution with the reference results. It is observed that the NN predicts well with minor discrepancies over the top surface of the aerofoil at around $x/c = 0.1$. NN places the shock on the top and bottom surface at the correct position with no visible oscillation. Figure 5.8(b) compares the skin friction distribution obtained from the NN in the third strategy against the CFD data. It is worth noting that the skin friction is a calculated quantity from the predicted outputs of the NNs, namely the stresses $\tilde{\tau}_x$ and $\tilde{\tau}_y$. It can be observed that there are slight deviations at the leading edges where NN is incapable of predicting sharp gradients in the distribution. A small overshoot is also present near the trailing edges. Despite these minor deviations, the NN predictions compare well in the region of the shock as there are only a sizeable deviation from the CFD data in the range $x/c \in [0.4, 0.5]$.

5.2.1 Inverse shape design for a target pressure distribution

This section presents an application of the proposed NN to inverse shape design using viscous compressible data in two dimensions. The goal is to show the potential and reliability of the proposed NN for the fast evaluation of the objective function in an optimisation process.

A pressure coefficient distribution, defined over the 300 points used to discretise an aerofoil, is considered as the target. The problem consists of finding the geometric configuration, given by the position of the control points of a set of two NURBS describing the aerofoil, that leads to the given pressure coefficient distribution. To recall, the two NURBS describing the aerofoil are defined as cubic B-Splines with eight control points, denoted by $\{\mathbf{P}_i^+\}$ and $\{\mathbf{P}_i^-\}$ for the top and bottom curve respectively and for $i = 1, \dots, 8$. It is important to note that, as in section 5.1, $\mathbf{P}_1^- = \mathbf{P}_1^+ = (0.5, 0)$ and $\mathbf{P}_8^- = \mathbf{P}_8^+ = (-0.5, 0)$ and $\mathbf{P}_2^-$ is restricted to be aligned with $\mathbf{P}_1^- = \mathbf{P}_1^+$ and $\mathbf{P}_2^+$. The objective function defined in equation (5.2) is also implemented in this example.

The minimisation problem is stated as

$$\begin{aligned} \min_{\mathbf{P}_2^+,\dots,\mathbf{P}_7^+,\mu,\mathbf{P}_3^-,\dots,\mathbf{P}_7^-} \quad & f(\{\mathbf{P}_i^\pm\}_{i=1,\dots,8}) \\ \text{subject to} \quad & \mathbf{P}_1^- = \mathbf{P}_1^+ = (0.5, 0) \\ & \mathbf{P}_8^- = \mathbf{P}_8^+ = (-0.5, 0) \\ & \mathbf{P}_2^- = \mathbf{P}_1^+ + \mu(\mathbf{P}_1^+ - \mathbf{P}_2^+) \\ & \mu \in [0.5, 1.5], \end{aligned} \tag{5.4}$$

where the target pressure distribution corresponds to the RAE2822 aerofoil.

The MCS is used to solve the optimisation problem (5.4). The parameters selected for the MCS algorithm, taken from [75], are 0.7 for the fraction of nests to be discarded and 0.5 for the power step size of the random walk. Additionally, the number of eggs to discard in every generation is set as 1, the number of eggs in the first generation is selected as 50 and finally 100 generations are used in the MCS.

To study the influence of the randomness of the MCS algorithm, figure 5.9 shows some statistics of the value for the objective function after 10 executions of the MCS optimisation are performed. The minimum, maximum, mean and standard deviation of the value of the objective function are represented for an increasing number of cases used

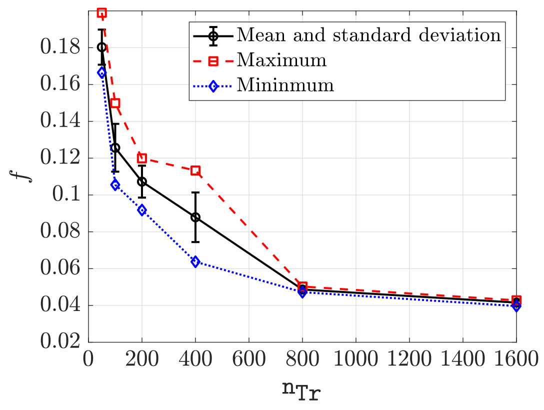
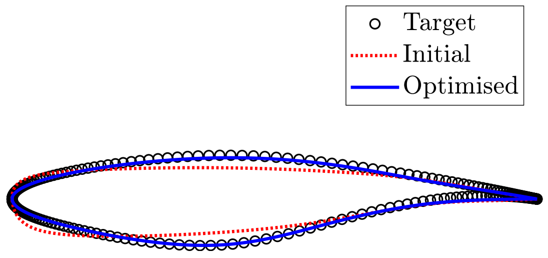


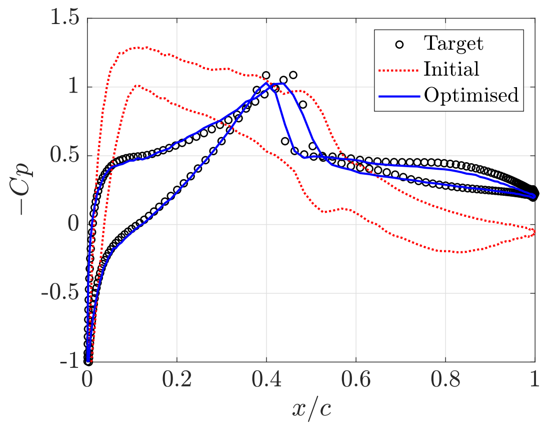
Figure 5.9: Value of the objective function after the minimisation problem is solved with the MCS as a function of the number of cases used for training the NN.

for training the NN. The results show that the randomness in the MCS is only relevant for a relatively low number of training cases, $n_{Tr} = 400$ in this example. For $n_{Tr} \geq 800$, the influence of the randomness in the MCS is minimal and both the minimum and maximum value of the objective function are below 0.05, meaning a discrepancy below 5% between the predicted and the target pressure coefficient distribution over the aerofoil.

Figure 5.10 shows the result of the inverse shape design. The plots corresponds to predictions of the NN when $n_{Tr} = 1600$ cases are used and, two hidden layers and 190 neurons are employed. The overfitting parameter is selected to be $\lambda = 1$. Figure 5.10 offers a visual comparison of the target geometry, the initial geometry used in the MCS algorithm and the final optimised geometry. There is a good agreement between the optimised and the target geometry. Figure 5.10(b) shows the pressure coefficient, again for the target geometry, the initial geometry used in the MCS algorithm and the final optimised geometry. As previously mentioned, the relative error in the $\mathcal{L}^2((0,c))$ norm, as measured by the objective function, is below 5%. In addition to the pressure distribution, the trained NN model also predicts the stresses. Although the skin friction distribution was not set as the target in this minimisation problem, it is worth comparing the predictions given the negligible cost of performing one prediction of the NN. Figure 5.11 shows the skin friction coefficient distribution of the target, initial and optimised geometry obtained using the MCS. The results again show that the predictions of the NN are in good agreement with the skin friction distribution of the target RAE 2822 aerofoil. It is worth mentioning that the optimisation process using the MCS algorithm takes just over one minute due to the almost negligible cost of evaluating the objective functions using the proposed NN.



(a) Aerofoil shape



(b) Pressure prediction

Figure 5.10: Target, initial and optimised geometry and pressure coefficient obtained using the MCS and the NN to predict the values of the objective function.

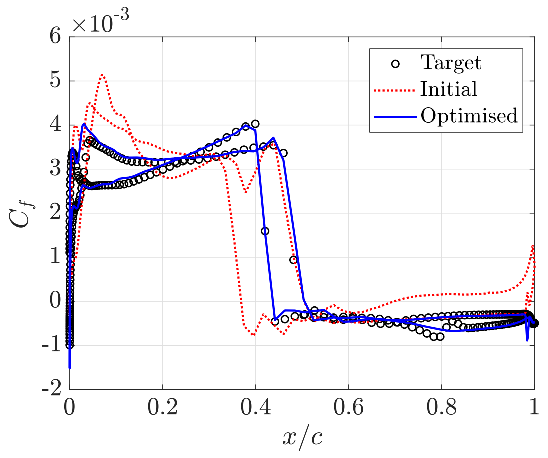


Figure 5.11: The skin friction coefficient distribution of the target, initial and optimised geometry obtained using the MCS and the NN.

5.3 Comparison of the multi-output NN with existing NNs

This example involves the prediction of the aerodynamic coefficients on a wing that is parametrised using the control points of the NURBS describing the surface. The base geometry corresponds to an approximation of the ONERA M6 wing [258] and consists of two surfaces, the top and bottom surface, each using six cubic B-splines with eight

Table 5.3: Control points of the top curve at the root and tip aerofoil section of the base geometry approximating the ONERA M6 wing.

i	1	2	3	4	5	6	7	8
$x_{i,1}$	0.0	0.0	31.8	136.9	332.6	571.0	738.1	805.9
$y_{i,1}$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$z^+_{i,1}$	0.0	9.5	26.3	34.6	43.9	26.6	8.4	0.0
$x_{i,6}$	690.6	690.6	708.5	767.6	877.7	1011.8	1105.8	1143.9
$y_{i,6}$	1196.3	1196.3	1196.3	1196.3	1196.3	1196.3	1196.3	1196.3
$z^+_{i,6}$	0.0	5.3	14.8	19.4	24.7	14.9	4.7	0.0

control points to define the surface. In this example, a planar surface is considered at the wingtip. The knot vectors of NURBS curves representing the surfaces are given by

$$\begin{aligned}\Lambda_u &= \{0,0,0,0,0.031,0.126,0.384,0.749,1,1,1,1\}\\ \Lambda_v &= \{0,0,0,0,0,0,1,1,1,1,1,1\}\end{aligned} \tag{5.5}$$

and the set of control points for the top curve $\{\mathbf{P}^+_{i,j} = (x_{i,j}, y_{i,j}, z^+_{i,j})\}_{i=1,\dots,8,\ j=1,6}$ are detailed in table 5.3. The control points for the bottom surface are symmetrically located with respect to the xy-plane and are defined as, $\mathbf{P}^-_{i,j} = (x_{i,j}, y_{i,j}, z^-_{i,j})$ with $z^-_{i,j} = -z^+_{i,j}$, for $i = 1, \dots, 8$.

The control points on the leading and trailing edges are fixed, i.e. $\mathbf{P}^-_{1,j} = \mathbf{P}^+_{1,j}$ and $\mathbf{P}^-_{8,j} = \mathbf{P}^+_{8,j}$. In addition, the control point $\mathbf{P}^-_{2,j}$ is restricted to be aligned with $\mathbf{P}^-_{1,j} = \mathbf{P}^+_{1,j}$ and $\mathbf{P}^+_{2,j}$. More precisely, it is assumed that $\mathbf{P}^-_{2,j} = \mathbf{P}^+_{1,j} + \mu(\mathbf{P}^+_{1,j} - \mathbf{P}^+_{2,j})$ with μ being a parameter defined in $[0.5, 1.5]$. This restriction ensures $\mathcal{G}^1$ continuity at the leading edge of the wing and leads to a problem with 25 independent geometric parameters in one aerofoil section. Figure 5.12 shows the 96 control points in the parametrised geometry used to approximate the ONERA M6 wing. However, in this example, only the control points in the root and tip chord of the wing, coloured red in figure 5.12 are varied in the xz-plane, that is, $\delta y_{i,j} = 0$. This amounts to a total of 50 independent geometric parameters in the analysis. The variation of the positions of these control points with respect to the base geometry, namely $(\pm\delta x_{i,j}, \pm\delta z_{i,j}))$, is considered to be the input to the neural network, where $(\delta x_{i,j}, \delta z_{i,j}) \in [0, 0.1c]^2$ and c denotes the chord of the aerofoil. To ensure that the variation of the control points does not lead to unrealistic geometries with very localised sudden change on the surface definition, the control points located between the root and tip chord, coloured in blue in figure 5.12

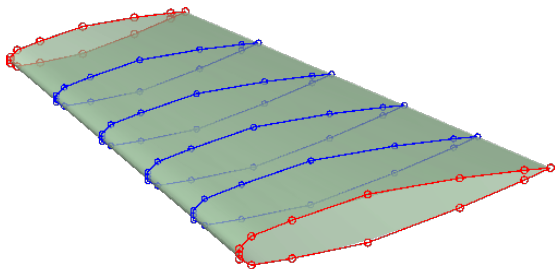


Figure 5.12: The ONERA M6 wing used as the base geometry and parametrised using the control points of the NURBS, denoted by circles. The lines represent the control polygon. The independent variables are coloured in red and the dependent variables in blue.

are linearly displaced and follow the relationship

$$\boldsymbol{x}^*_{i,j} = \boldsymbol{x}_{i,j} + \delta\boldsymbol{x}_{i,1}(1 - \frac{y_{i,j}}{b}) + \delta\boldsymbol{x}_{i,6}\frac{y_{i,j}}{b}, \qquad i = 1, \dots, 8, j = 1, \dots, 6 \tag{5.6}$$

where $\boldsymbol{x}_{i,j}$ denotes the coordinate vector of i-th control points in the j-th aerofoil section, $\boldsymbol{x}^*_{i,j}$ represents the moved coordinate vector and b is the wingspan. The surface triangulation of the base geometry is generated using the advancing front method. The Cartesian coordinates of surface mesh nodes are projected to obtain the corresponding parametric coordinates of the NURBS surface. Given a new set of control points, obtained from design of experiment [251], the same parametric coordinates are evaluated to obtain a new set of Cartesian coordinates for the new geometry. This allows the designer to use the same number of nodes to discretise the geometry in the surface mesh. The coordinates on the planar wingtip surface and the symmetry boundary wall surface are then updated using the Laplacian smoothing technique [259]. A new volume mesh is generated for each new surface mesh using the Delaunay meshing scheme and the nodes are placed according to a specified background mesh and user defined sources [125]. There are 11,283 nodes used to discretise the wing geometry in the surface mesh and around 1.34 million elements in the volume meshes.

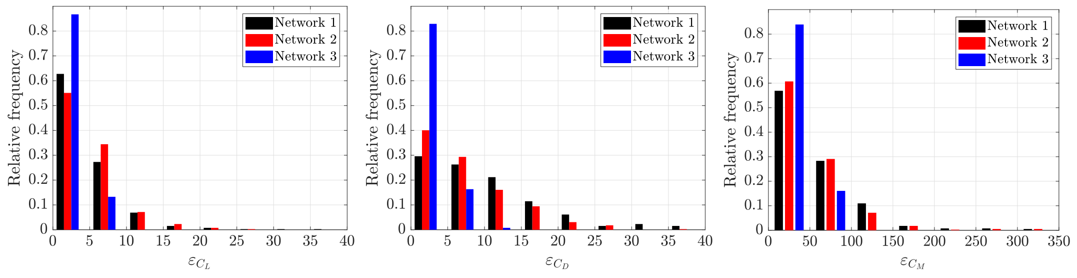
In this example, the training and test cases are of size $\texttt{n}_{\texttt{Tr}} = 1600$ and $\texttt{n}_{\texttt{Te}} = 400$ respectively. Both data sets are obtained by using the latin hypercube sampling. Steady Euler calculations are performed in the transonic regime at a free-stream Mach number, $M_\infty = 0.84$ and an angle of attack, $\alpha = 3.06°$. As before, the trained ROM with the lowest mean value of the error, measured in counts, in the test set is selected by performing a numerical experiment to explore the accuracy of the predicted aerodynamic coefficients as a function of the hyperparameters of the employed ROM. For instance, in the case of NN there is the choice of the number of hidden neurons, $\texttt{n}_{\texttt{N}}$, the number of hidden layers, $\mathtt{n}_\mathtt{L}$ and the overfitting parameters, λ. The NN is trained for each combination of the three listed hyperparameters in a predefined range and, the quadratic cost function is implemented with the regularisation term, as in equation (3.53) is employed to evaluate the errors in the training and to avoid the need to divide the dataset into validation and training cases. The NN is trained 10 times for each combination of hidden layer, hidden neurons and the overfitting parameter. This is performed in an attempt to obtain a trained model that is not stuck in a local minimum. The weights of the trained model with the lowest value of the error measured in the test set is recorded before the next combination of hyperparameters is trained.

Three types of NNs are considered and their performance is compared. The first network considers 50 geometric parameters representing the locations of the control points of the root and the tip aerofoil chord of the wing as inputs and one aerodynamic coefficient, the lift, drag or moment, as output. The second network considers 50 geometric parameters as inputs and the three aerodynamic coefficients as outputs. The third network also considers the 50 geometric parameters as the inputs and the outputs are the pressure defined at the nodes used to discretise the wing surface in the computational mesh. There is a total of 11,283 outputs in this network.

To illustrate the resulting performance of the three NNs employed, figure 5.13 quantifies the accuracies for the three aerodynamic quantities of interest, measured in counts. The histograms represent the relative frequency of the error for all test cases. Figure 5.13(a) shows the results measured in lift counts. The first NN employs a single hidden layer with $\mathtt{n}_\mathtt{N} = 40$ and the over-fitting parameter is selected as $\lambda = 10$. The second NN employs a single hidden layer with 10 hidden neurons and the over-fitting parameter is selected as $\lambda = 1$. Finally the third NN employs one hidden layer with $\mathtt{n}_\mathtt{N} = 290$ and the over-fitting parameter is set as $\lambda = 10$. Table 5.4 summarises the selected NN configurations for the predictions of lift in the test set using the three networks considered in this example. The results show the first and second NN predict only 63% and 56% of the test cases, respectively, with an error below five lift counts. The third NN, however, provides the most accurate predictions with an error below five lift counts for almost 85% of the test cases. This represents a significant difference of about 20%. Moreover, the third NN is observed to achieve a marginally higher performance, with 100% of the test cases predicted with an error below 10 lift counts, compared to almost 80% for the first and second NNs. The maximum error measured

Table 5.4: The selected NN configurations for the prediction of lift using the three networks considered.

	Network		
	1	2	3
Input	$\mathbf{P}^{\pm}$		
Output	C_L	C_L, C_D, C_M	C_p
Hidden layer, $\mathtt{n_L}$	1	1	1
Hidden neuron, $\mathtt{n_N}$	40	10	290
Overfitting, λ	10	1	10



(a) Lift counts (b) Drag counts (c) Moment counts

Figure 5.13: Relative frequency of the error on the test set for the three neural networks employed, measured in counts.

in the test dataset for the three NNs is 36, 26 and 9 lift counts, respectively.

Figure 5.13(b) shows the relative frequency of the error, measured in drag counts. A different set of hyperparameters with the best accuracy is used for first network, while the same configuration is maintained for the second and third NN. Here, the first network employs a single hidden layer with 20 neurons and the over-fitting parameter is set as $\lambda = 10$. The third network produces a prediction with an error below 5 drag counts for almost 83% of the test cases compared to the first and second network with only 30% and 40% of the cases, respectively. This represents a significant difference of about 40% of the cases with better predictions. The maximum drag counts achieved for the three networks are 39, 38 and 11 drag counts, respectively.

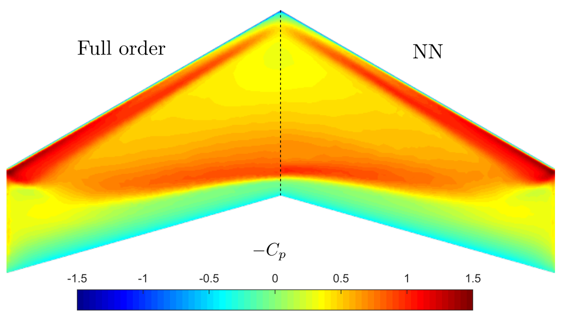
Lastly, the histogram of figure 5.13(c) represents the relative frequency of the error, measured in moment counts. Here, the first two NNs have similar performance with almost 60% of the cases falling below 50 moment counts and about an additional 25-30% of the cases below 100 moment counts. On the other hand, the third network provides a marginally higher accuracy of almost 85% of the cases below 50 moment counts and 100% below 100 moment counts. The maximum accuracies for the three networks employed are, 345, 326 and 96 moment counts. This shows the robustness of the third network for its use as an aerodynamic predictor for highly parametrised geometry.

Figure 5.14 shows a test case to further illustrate the comparison of the performance of the full order model and the output of the proposed NN model. Figure 5.14(a) shows the surface pressure contours of the full order and the corresponding NN prediction. The results show that, visually, there is no significant difference between the CFD data and the NN predictions. Figures 5.14(b), 5.14(c) and 5.14(d) show the aerofoil cross section and their corresponding pressure distributions for the full order and NN predictions at 20%, 50% and 80% in the spanwise direction, respectively. It can be observed how the geometry changes across the cross section, from a comparatively blunt leading edge and an almost symmetric trailing edge at 20% to a more rounded leading edge, a flattened top and a comparatively higher cambered aft section at 80%, some of which are characteristics of supercritical aerofoils [260]. Figure 5.14(b) shows that NN predicts well and captures both shocks at the right position on the upper surface. Minor discrepancies in predicting the position of the second shock are observed in figure 5.14(c). Closer to the wingtip in figure 5.14(d), the reference result shows that there is no shock and the NN model is capable of capturing this behaviour well.

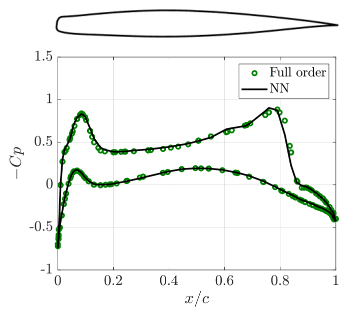
5.3.1 Inverse shape design to maximise lift-to-drag ratio

This section presents another application of the proposed NN to inverse shape design using inviscid compressible data in three dimensions. The goal is to show the potential and reliability of the NN for the fast evaluation of the objective function in an optimisation process in a different setting.

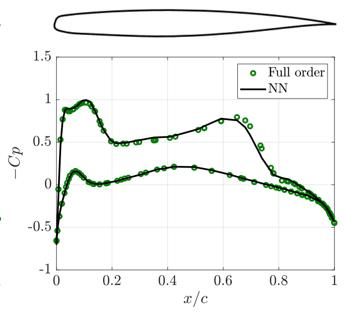
The problem consists of finding the geometric configuration, given by the position of the control points of two NURBS surfaces describing the wing, that leads to the maximum achievable lift to drag ratio within the design space used in the training of the NN. The third network, that is the NN that was built earlier in the previous example, is employed to predict the pressure distribution for a given set of control points. To recall, two NURBS surfaces are considered, where the section is a cubic B-spline with eight control points. A planar surface is considered at the wingtip. The control points for the bottom surface are symmetrically located with respect to the xy-plane and are defined as, $\mathbf{P}^{-}_{i,j} = (x_{i,j}, y_{i,j}, z^{-}_{i,j})$ with $z^{-}_{i,j} = -z^{+}_{i,j}$, for $i = 1, \dots, 8$ and $j = 1, \dots, 6$. The control points on the leading and trailing edges are fixed, i.e.



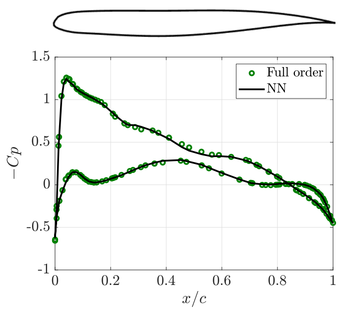
(a) Surface pressure contour



(b) Predictions at 20%



(c) Predictions at 50%



(d) Predictions at 80%

Figure 5.14: Comparison of the pressure coefficients, C_p, obtained with the CFD solver and NN predictions using the third network for a test case at different locations in the spanwise direction of the wing.

$\mathbf{P}^-_{1,j} = \mathbf{P}^+_{1,j}$ and $\mathbf{P}^-_{8,j} = \mathbf{P}^+_{8,j}$. Moreover, to ensure $\mathcal{G}^1$ continuity at the leading edge, it is assumed that $\mathbf{P}^-_{2,j} = \mathbf{P}^+_{1,j} + \mu(\mathbf{P}^+_{1,j} - \mathbf{P}^+_{2,j})$, for $j = 1, \dots, 6$, with μ being a geometric parameter within $[0.5, 1.5]$.

The objective function is defined as

$$f(\{\mathbf{P}^{\pm}_{i,j}\}) = -\frac{C^\star_L(\{\mathbf{P}^{\pm}_{i,j}\})}{C^\star_D(\{\mathbf{P}^{\pm}_{i,j}\})}, \tag{5.7}$$

where $C^\star_L$ and $C^\star_D$ denote the lift and drag coefficients predicted by the trained NN model, respectively. The minimisation problem is formally stated as

$$\begin{aligned} \min \quad & f(\{\mathbf{P}^{\pm}_{i,j}\}_{i=1,\dots,8, j=1,\dots,6}) \\ \text{subject to} \quad & \mathbf{P}^-_{1,j} = \mathbf{P}^+_{1,j} \\ & \mathbf{P}^-_{8,j} = \mathbf{P}^+_{8,j} \\ & \mathbf{P}^-_{2,j} = \mathbf{P}^+_{1,j} + \mu(\mathbf{P}^+_{1,j} - \mathbf{P}^+_{2,j}) \\ & \mu \in [0.5, 1.5], \end{aligned} \tag{5.8}$$

for $j = 1, \dots, 6$. In this example, the number of eggs in the first generation and the total

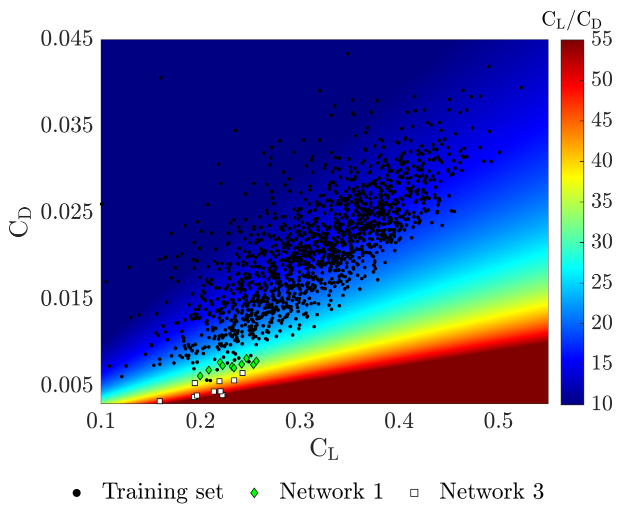
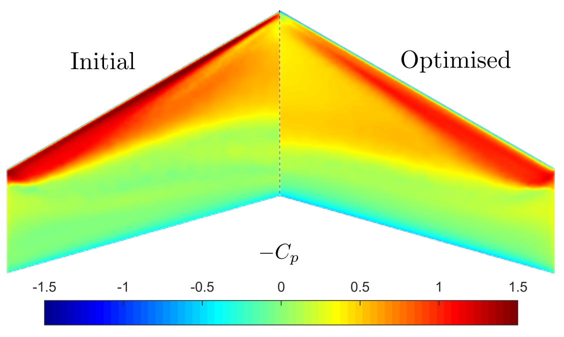


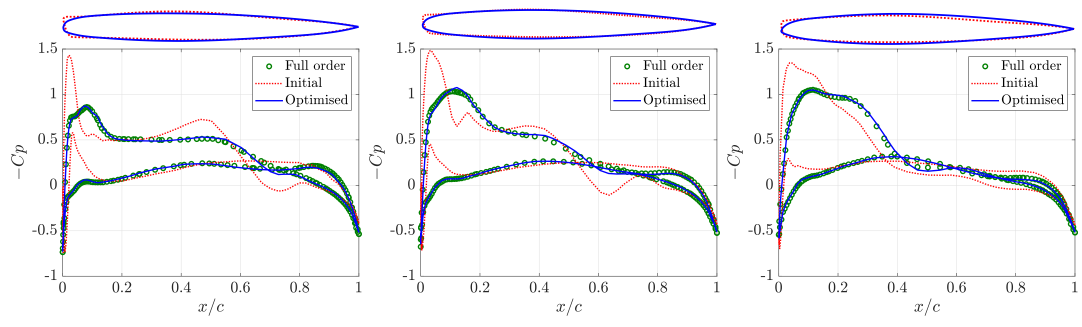
Figure 5.15: A visual comparison of the distribution of lift and drag coefficients in the training dataset (denoted by circles) and, the optimised solutions by the first network (denoted by diamonds) and the third network (denoted by squares) in the minimisation problem.

number of generations are set as $\mathtt{n_{hn}} = 50$ and 100, respectively. The performance of the proposed NN, referred to as the third network in this example, is also compared with existing NN, where only one aerodynamic coefficient, the lift, drag or moment coefficient is predicted. The existing NN is also referred to as the first network in this section. The NN configurations of the first and third network is reported in section 5.3. The MCS is run 10 times in an attempt to obtain 10 geometries with different lift-to-drag ratios in both cases.

Figure 5.15 offers a visual comparison of the optimised geometries in the lift and drag spectrum considered in the training set when $\mathtt{n_{Tr}} = 1,600$ cases are used in the NN. The results show that the MCS employed with the third network is capable of finding geometries with higher lift-to-drag ratios compared to MCS employed with the first network in all the 10 runs. Moreover, the optimised solutions of the MCS with the third network are higher than those observed in the training dataset and above all, within the same design space used in the training of the NN. The maximum lift-to-drag ratio obtained with the third NN is 56 compared to 34 with the first NN, which represents a gain of 21 with the proposed method. The superior performance of the proposed technique is attributed to the fact that the third network attempts to find a geometric configuration that maximises the lift-to-drag ratio in the design space of the pressure distributions observed in the training. Conversely, the first network which models the lift or drag coefficient directly, only sees these values in the training and therefore cannot find a higher maximum of the lift-to-drag ratio.



(a) Surface pressure contour



(b) Prediction at 10% (c) Prediction at 50% (d) Prediction at 90%

Figure 5.16: Comparison of pressure coefficients, C_p, for the initial geometry and the optimised geometry as predicted by NN, and the full order solution of the optimised geometry at various locations in the spanwise direction of the wing.

Figure 5.16 shows the results of the inverse shape design with the maximum lift-to-drag ratio obtained with the third network and corresponds to a lift coefficient of 0.222 and drag coefficient of 0.0040. Figure 5.16(a) offers a visual comparison of the predicted pressure distributions of the initial geometry on the left and that of the optimised geometry on the right. Figures 5.16(b), 5.16(c) and 5.16(d) show the comparison of the aerofoil cross sections and their corresponding pressure distributions for the initial and optimised geometries as predicted by NN, and also the full order solution of the optimised geometry. The results shows a good agreement between the CFD data and the NN predictions for the optimised geometry. The difference in the lift-to-drag ratio between the full order solution and NN predictions of the optimised geometry is less than 6. It is worth mentioning that the MCS takes 41 minutes and NN takes less than 2% of the total time taken in the minimisation problem. Conversely, it took more than 400 hours to obtain the full order solution of the geometries used in the dataset when employed on five cores and with three geometric multi-grids.

5.4 Deforming wings at various flow conditions

The last example considers the computation of the aerodynamic coefficients for a deforming wing at a free stream Mach number, M_∞, and an angle of attack, α, in predefined intervals, $I_M = (0.5, 0.9)$ and $I_\alpha = (0^\circ, 10^\circ)$. The ONERA M6 wing is used as the baseline geometry and it undergoes a geometric twist along the y-axis and a wingtip deflection in the z-axis, in predefined intervals, $I_\beta = (0^\circ, 10^\circ)$ and $I_\kappa = (0\ \text{m}, 1\ \text{m})$. The range of inflow conditions considered leads to the subsonic and transonic transonic regime.

A geometric twist about the y-axis [261] can be written as

$$\boldsymbol{x}_i^\star = \boldsymbol{R}_y(\boldsymbol{x}_i - \boldsymbol{x}_i^{AC}) + \boldsymbol{x}_i^{AC}, \qquad \boldsymbol{R}_y = \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix}, \tag{5.9}$$

for $i = 1, \dots, N$, where $\boldsymbol{x}_i$ represents a Cartesian coordinate on the wing, $\boldsymbol{x}_i^{AC}$ is the reference aerodynamic centre to which the rotation is performed and $\boldsymbol{x}_i^\star$ is the transformed coordinate. $\boldsymbol{R}_y$ denotes the rotation matrix about the y-axis, N is the total number of points used to discretise the geometry and β is the angle of twist, measured in degrees.

The wing is deflected from the xy-plane assuming it acts as a uniformly loaded cantilever beam, with one end fixed, that is the root of the wing, and a free end, here the wingtip. The deflection [262] can be written as

$$\delta z_i = \frac{\kappa}{3}\left[\frac{y_i^4}{b^4} - \frac{4y_i^3}{b^3} + \frac{6y_i^2}{b^2}\right], \tag{5.10}$$

for $i = 1, \dots, N$, where δz_i denotes the amplitude for i-th coordinate in the z direction, y_i is the initial length in the spanwise direction, b is the wingspan and κ is the maximum deflection at the wingtip. Moreover, it is ensured that for every transformed coordinate, the length in the spanwise direction is kept constant. The formulation of the arc length by integration is given as

$$\tilde{b}_i = \int_0^{y_i^\star} \sqrt{1 + \left(\frac{\mathrm{d}z}{\mathrm{d}y_i}\right)^2}\,\mathrm{d}y, \tag{5.11}$$

for $i = 1, \dots, N$, where $\tilde{b}_i$ is the arc length of the i-th coordinate on the wing and is

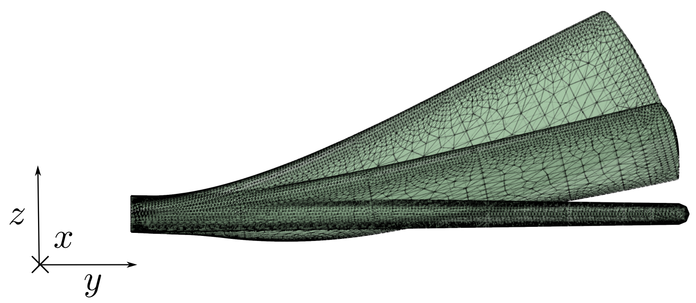
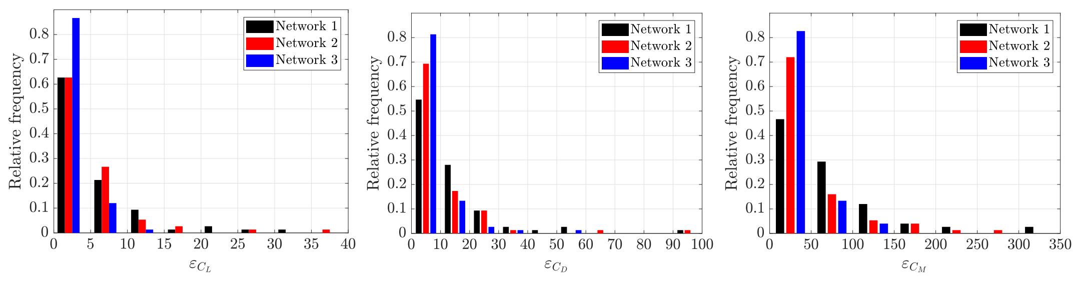


Figure 5.17: Illustration of the deformed ONERA M6 wing in $I_\beta \times I_\kappa$, scaled by an amplifying factor of 10.

equal to y_i in the baseline geometry. $y_i^\star$ represents the new y coordinate for which the length is kept constant.

A reference volume mesh is generated around the ONERA M6 wing and the Delaunay graph method [263] is used to update the nodes of the transformed geometry. This mesh morphing scheme ensures that the primary mesh topology is maintained. Large displacements can be applied at the boundaries in a single step without mesh entanglement that generally affects standard mesh movement procedures [263]. In this example, it was found that there is no negative volume and mesh entanglement moving from the primary mesh in one single step for a geometric twist of up to $\beta = 11^\circ$ and a maximum deflection of $\kappa = 1.1$m at the wingtip. Figure 5.17 illustrates the deformation between the minimum and maximum displacement in $I_\beta \times I_\kappa$, scaled by an amplifying factor of 10, using the ONERA M6 wing as the baseline geometry.

The latin hypercube sampling is used to design the training and test dataset of size $\mathtt{n_{Tr}} = 100$ and $\mathtt{n_{Te}} = 75$, respectively. The reference volume mesh consists of 1,071,457 elements and 16,240 nodes are used to discretise the wing geometry. In this example, the three different types of NN employed consider M_∞, α, β, and κ, as inputs. The first network has one single aerodynamic output, the lift, drag or moment coefficient. The second network has three aerodynamic coefficients as outputs and finally the third network considers the pressure defined at the 16,240 nodes used to discretise the geometry in the volume mesh, as outputs. As in previous examples, numerical experiment is performed to explore the accuracy of the predicted aerodynamics coefficients, measured in counts in the test set, as a function of the number of hidden layers, $\mathtt{n_L}$, the number of hidden neurons, $\mathtt{n_N}$, and various values of the over-fitting parameter, λ for the three types of NN considered.



(a) Lift counts (b) Drag counts (c) Moment counts

Figure 5.18: Relative frequency of the error on the test set for the three neural networks employed, measured in counts.

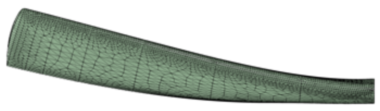
Table 5.5: The selected NN configurations for the prediction of lift using the three networks considered.

	Network		
	1	2	3
Input	$M_\infty, \alpha, \beta, \kappa$		
Output	C_L	C_L, C_D, C_M	C_p
Hidden layer, $\mathtt{n_L}$	3	3	1
Hidden neuron, $\mathtt{n_N}$	140	190	270
Overfitting, λ	0.01	0.01	0.01

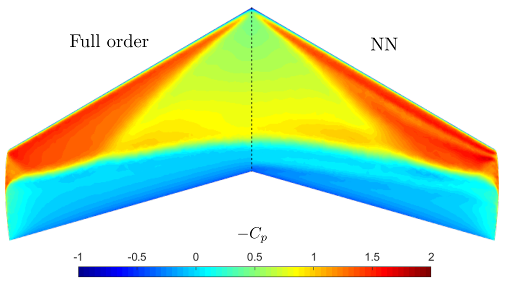
Figure 5.18 shows the comparison of the relative frequency of the error, measured in counts in the test dataset for the three types of NNs employed. Figure 5.18(a) shows the relative frequency of the error measured in the prediction of the lift. Following the numerical experiment carried out in this example, the first network is selected to have three hidden layers and $\mathtt{n_N} = 140$, the second NN has three hidden layers and 190 hidden neurons, and finally the third network employs one hidden layer with $\mathtt{n_N} = 270$. All the selected network configurations employ an over-fitting parameter of $\lambda = 0.01$. The same configuration is maintained in the analysis of the drag and moment for the second and third networks. Table 5.5 summarises the selected NN configurations for the predictions of lift in the test set using the three networks considered in this example. The results show that both the first and second NNs achieve about 60% of the test cases below five lift counts while the third network achieves a marginal 88% of the test cases in the same range. The first noticeable difference is that the third network has almost 100% of the cases with an error less than 10 lift counts in the test set and the maximum error obtained with this NN is 11 lift counts. The second and third network provide a prediction with a maximum error of 33 and 36 lift counts, respectively. Figure 5.18(b) represents the relative frequency of the error, measured in drag counts. The first network provides a much lower accuracy of 55% of the cases below 10 drag

counts, as opposed to 70% and 82% for the second and third networks. The three NNs have a maximum error of 98, 94 and 51 drag counts, all of which corresponds to the same case in the test dataset. Lastly, Figure 5.18(c) describes the relative frequency of the error in the test set, measured in the test set. The first NN achieves 46% of the cases below 50 moment counts, which is comparatively lower to the second and third NN, with 72% and 83% of the cases. The third network has a maximum error of 127 moment counts, while the first and second NNs provide an error of more than double of that, 400 and 270 moment counts respectively.

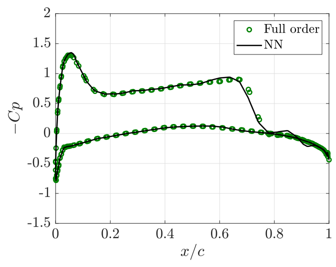
Figure 5.19 illustrates the comparison between the performance of the full order model and the output of the third network. Figure 5.19(a) shows the deformed geometry of the test case, with an amplification factor of 10. The CFD simulation corresponds to a free-stream Mach number, $M_\infty = 0.83$ and an angle of attack, $\alpha = 5.4°$, which represents a transonic flow. Figure 5.19(b) shows the surface pressure contours of the full order on the left and the corresponding NN predictions on the right. The surface contour plots compare well with each other visually. Figures 5.19(c), 5.19(d) and 5.19(e) show the comparison of the predicted surface pressure distributions by the network with the full order solution at 20%, 50% and 80% of the wing in the spanwise direction, respectively. Figure 5.19(c) shows that the NN predicts well and captures both shocks at the correct position on the upper surface. Minor discrepancies are observed over the top surface, at $x/c = 0.9c$. More oscillations are observed in the prediction over the top surface in both the 50% and 80% sectional pressure plots. The NN model captures the shocks relatively well and shows good agreement with the reference CFD results.



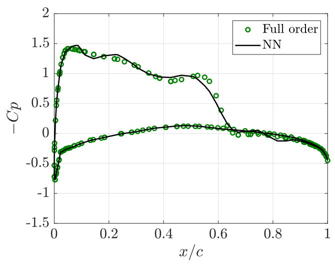
(a) $\beta = 3.0°$, $\kappa = 0.86$m



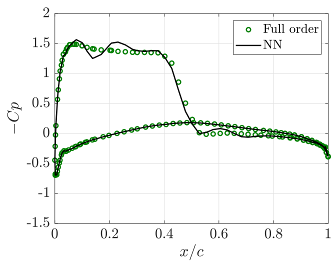
(b) Surface pressure contour



(c) Predictions at 20%



(d) Predictions at 50%



(e) Predictions at 80%

Figure 5.19: Comparison of the pressure coefficients, C_p, obtained with the CFD solver and NN predictions using the third network for a test case with inflow conditions, $M_\infty = 0.83$ and $\alpha = 5.4°$, at different locations in the spanwise direction of the wing.

Chapter 6

Concluding remarks

The present chapter concludes the thesis with a recollection of the proposed method and the outcomes achieved during the course of the current work. A brief discussion is also provided on the outlook of the used technologies for the progression and further development of ideas proposed in this thesis.

6.1 Conclusions

The primary aim of the work in this thesis was to establish a fast and reliable non-intrusive ROM that is capable of providing accurate solutions for aerodynamic applications. This would enable the routine use of the aerodynamic system during the conceptual design stage. A new multi-output NN methodology has been presented for the prediction of aerodynamic coefficients of aerofoils in two dimensions and wings in three dimensions. The proposed multi-output NN predicts the pressure or stress distributions on the aerodynamic surface. The dataset used to train the network is obtained using inviscid and viscous compressible flow data from a vertex-centred CFD solver. The aerodynamic coefficients are computed after the pressure and stress distributions are predicted.

The accuracy of the predictions using the proposed multi-output NN has been shown to be higher when compared to a NN where the output consists of the three aerodynamic coefficients. The results also show the superiority when compared to a NN where only one aerodynamic coefficient is predicted. The superior performance has been demonstrated for both two and three dimensional examples involving free stream Mach number and the angle of attack as the inputs of the NNs. Moreover, the performance of the proposed NN has been compared against NNs which predict pressure at various regions in the computational domain to defend the choice of the number of outputs in the NN. It was deduced that the NNs predict with similar accuracies. However, the NNs which predict at nodes other than the surface nodes are more expensive to train as more weights in the network need to be optimised.

In addition, the proposed NN has also been compared to the POD and the results show the better performance of the NN, especially when predicting flows that involve shocks. The superior accuracy of the proposed NN has been shown for problems in which the parameters involve flow conditions and for more challenging scenarios in which the parameters are the control points of the NURBS that describe the geometry of the aerodynamic shape. An extensive set of numerical studies has been presented to show the effect of the NN hyperparameters and numerical parameters of the POD on the accuracy of the predictions. Furthermore, the effect of increasing the number of cases in the training set on the accuracy of both the NNs and the POD has also been studied.

A NN was also trained to predict the viscous flow solutions using a database of inviscid flow solutions. The network considers the pressure obtained by solving the Euler equations and defined at the mesh nodes discretising the aerofoil as the inputs. The output consists of pressure and stresses obtained using the Favre-averaged NS solver and also defined at the mesh nodes discretising the aerofoil surface. The trained NN model can be employed to reduce the computational effort of solving the NS equations as only the Euler equation need to be resolved, leading to accurate viscous predictions.

The potential of the proposed NN has also been demonstrated for three dimensional examples involving flow conditions and geometric parameters. It is worth noting that examples involving up to 50 parameters have been considered in three dimensions. The influence of the accuracy of the CFD data on the accuracy of the NN predictions has been studied in two and three dimensions. It was found that the higher the accuracy in the CFD data, the more challenging it is for the NNs to perform accurate predictions. This is attributed to the stronger shocks present in finer meshes due to the increased resolution of the flow field. The accuracy of the predictions in two and three dimensions has also been compared. It has been shown that for a swept wing the NN offers a greater accuracy when compared to a two dimensional aerofoil. This is explained by the weaker shocks in three dimensions and the delayed appearance of shocks in swept wings, leading to a greater set of training cases containing smooth solutions.

The proposed NN enables the fast prediction of aerodynamic quantities of interest and can be applied to speed up the design and optimisation of aerodynamic shapes. This work has also presented the use of the new multi-output NN in the context of aerodynamic shape optimisation for aerofoils in two dimensions and wings in three dimensions. The potential and reliability of the proposed NN for the fast evaluation of the objective function is shown using a gradient-free optimisation algorithm, the MCS.

In the first inverse design example, the predictions given by the novel NN made it possible to perform an inverse identification of the aerofoil geometry for a given pressure distribution in less than one minute for both inviscid and viscous flows. Another example shows the application of the multi-output NN in an optimisation design context. The problem consisted of maximising the lift-to-drag ratio using the predictions of the novel NN. The MCS was capable of finding geometries with higher lift-to-drag ratio than observed in the training set and within the same design space. Additionally, the superior performance of the proposed optimisation framework was shown against an existing framework where the more traditional NN is employed. The proposed framework obtained a maximum lift-to-drag ratio of 56 while with the existing framework, a maximum of 34 was obtained.

6.2 Recommendations for future work

There are various directions in which the proposed method or NNs in general, could be pursued based on the work presented here. Some examples are listed below.

- **Complexity of the problem.** The performance of the proposed NNs has been assessed in numerical examples involving flow or geometric parameters. However, an interesting scenario would be the inclusion of the proposed NN to predict the aerodynamic coefficients in numerical examples involving both flow conditions and parametrised geometries. In addition, it would be compelling to find out the number of training cases that are required to match the accuracies obtained in the examples of this thesis where flow and geometric parameters were considered separately. There are only a few examples in the literatures where both flow and geometric parameters are considered. In such cases, the classic non-intrusive ROMs, such as the POD [31] or traditional NNs where the aerodynamic quantities of interest are predicted directly [31], have been considered. There are also works in which the aerofoil shapes and flow conditions are synthesised in the form of images and are considered to be the inputs to the NN, and the aerodynamic coefficients are predicted as the outputs [61].

Another scenario is the extension of the use of the proposed NN to predict aerodynamic coefficient using unsteady compressible CFD data. This opens the possibility of performing a range of different analyses. One example is the prediction of the dynamic responses of aerodynamic components due to gust [264, 17]. Traditional techniques involves the use of the POD [12, 23, 26]. More recently, machine learning techniques [265, 266], as well as NNs [267, 268], have also been employed to provide fast solutions in unsteady problems. It is therefore reasonable to use the proposed technique to predict unsteady CFD solutions to compare its performance with traditional methods.

- **Comparison.** During the course of the work presented in this thesis, the author has noticed a large volume of research focused on the use of intrusive and non-intrusive ROMs for aerodynamic applications. However, there is a lack of fair comparison between the performance of the two types of ROMs in general to show their strengths and weaknesses. Moreover, with the recent advances in physics-informed NNs [269, 270] as a non-intrusive ROM, it will be fascinating to see the comparison of the performance of physics-informed NNs against the proposed non-intrusive data-driven ROMs in this thesis.

  Additionally, the current work employs the multilayer perceptron and its extended variant, the deep NN. There are several other choices of NNs which are more complex. It is therefore of interest to explore the varieties of NNs that are available in the literature. Two popular choices are the convolutional NN [61, 62, 47] and the long-short term memory NN [271]. The performance of these more complex NN architectures when predicting the surface pressure as formulated in this thesis, can be compared against the proposed NN.

- **Accurate shock prediction.** Throughout the course of this work, the author has noticed the difficulty of non-intrusive or data-driven ROMs to accurately predict sharp gradients such as shocks in the flow. The authors in [7, 8] employ the POD and domain decomposition techniques to accurately reconstruct shocks in the

the pressure field. Therefore, it is relevant to employ the domain decomposition with the proposed NN and compare the performance with similar techniques in the literature.

- **Smart geometric filtering.** In an attempt to reduce the number of dimensions used to parametrise aerofoils and wings, researchers have previously employed dimension reduction techniques such as the POD [272] or coupled two parametrisation techniques [55]. More recently, there have been new applications of NNs in geometry parametrisation, such as the use of auto-encoders [273] or generative-adversarial networks [56]. According to the author's knowledge, the use of generative NNs coupled with NURBS has not been implemented before. Additionally, this will greatly help to reduce the computational effort to train a non-intrusive ROM, for instance the NN in general [53].

Appendix A

Supporting materials

The appendix provides additional supporting materials that have been employed in this thesis and consists of three sections. The first section provides a brief description of the non-uniform rational B-splines (NURBS). The second section describes the Delaunay graph mapping that was employed to deform a reference mesh in numerical examples involving geometric perturbations. The third section presents an overview of the cuckoo search algorithm with remarks on the key implementation of the modified cuckoo search algorithm.

A.1 Non-Uniform Rational B-splines (NURBS)

NURBS are industry standards employed to represent and exchange geometric information, on which many international standards such as IGES and STEP files are based. In addition to the capability of producing analytical shapes such as conic and quadric surfaces, NURBS are known for representation of free-form geometries using a unified mathematical basis [274].

A NURBS curve is a vector-valued piecewise rational function of degree r defined in parametric form as

$$\boldsymbol{C}(u) = \frac{\sum_{i=0}^{\mathrm{n_{cp}}} w_i \boldsymbol{P}_i N_i^r(u)}{\sum_{i=0}^{\mathrm{n_{cp}}} w_i N_i^r(u)} \tag{A.1}$$

where $\boldsymbol{P}_i$ are the coordinates of the $n_{cp}+1$ control points in the control polygon, w_i represents the weights at each control point, and $N_i^r(u)$ is the normalised B-spline

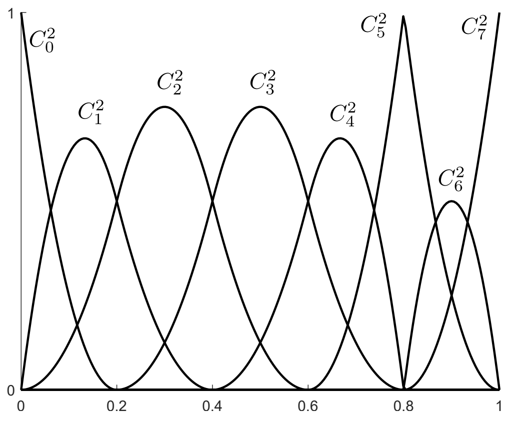


Figure A.1: B-spline basis functions using the knot vector in .

basis function of r-th degree defined recursively as

$$N_i^0(u) = \begin{cases} 1, & u \in [u_i, u_{i+1}) \\ 0, & \text{otherwise} \end{cases}, \tag{A.2}$$

$$N_i^k(u) = \frac{u - u_i}{u_{i+k} - u_i} N_i^{k-1}(u) + \frac{u_{i+k+1} - u}{u_{i+k+1} - u_{i+1}} N_{i+1}^{k-1}(u),$$

for $k = 1, 2, ..., r$ and the knots, u_i forming the knot vector, $\Lambda_u = [u_0, u_1, ..u_{n_k}]$, where n_k is defined as $n_k = \mathtt{n_{cp}} + r + 1$. Therefore for non-uniform and non-periodic B-splines, the knot vector takes the form

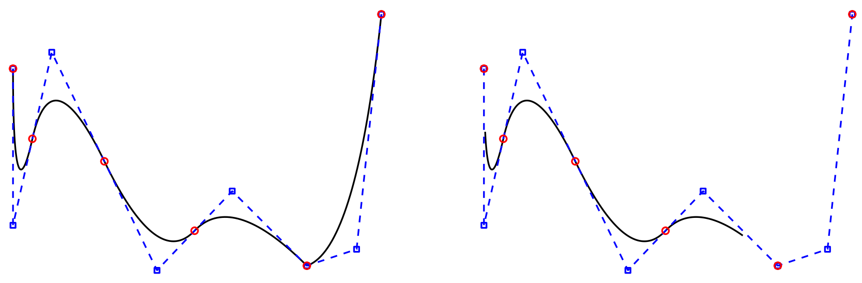
$$\Lambda_u = [0, 0, .., 0, u_{r+1}, ..., u_{m-r-1}, 1, 1, ..., 1] \tag{A.3}$$

where the end knots are repeated with multiplicity, $r + 1$. The basis functions defined in (A.2) is used over the entire curve with focus in the interval $[0, 1]$. Using the knot vector of equation (A.3) in (A.1) defines a Bezier-like curve which interpolates the endpoints and is tangential to the first and last legs of the control polygon. Figure A.1 shows the basis functions for the knot vector

$$\Lambda_u = \{0, 0, 0, 0.2, 0.4, 0.6, 0.8, 0.8, 1, 1, 1\}. \tag{A.4}$$

An example of a NURBS curve is illustrated in figure A.2. The knots, denoted by circles, emphasise on the discontinuous definition of the parametrisation. Figure A.2(a) shows an untrimmed NURBS curve and figure A.2(b) depicts the corresponding trimmed curve in the interval $u \in [0.1, 0.8]$.

Similarly, the NURBS curve can be extended to describe a surface. A NURBS surface



(a) Untrimmed (b) Trimmed

Figure A.2: Illustration of an untrimmed and trimmed NURBS curve. The solid black line represents the curve, formed using knots which are denoted by ∘. The NURBS curve is parametrised using control points, denoted by □ and found within the control polygon denoted by the blue dashed line.

of degree r in u and degree q in v, is a piecewise rational function defined in parametric form as

$$\boldsymbol{S}(u,v) = \frac{\sum_{i=0}^{\mathtt{n}^u_{\mathtt{cp}}} \sum_{i=0}^{\mathtt{n}^v_{\mathtt{cp}}} w_{ij} \boldsymbol{P}_{ij} S^{r,q}_{i,j}(u,v)}{\sum_{i=0}^{\mathtt{n}^u_{\mathtt{cp}}} \sum_{i=0}^{\mathtt{n}^v_{\mathtt{cp}}} w_{ij} S^{r,q}_{i,j}(u,v)}, \quad 0 \leq u,v \leq 1, \tag{A.5}$$

where $\boldsymbol{P}_{ij}$ are the coordinates of the $(\mathtt{n}^u_{\mathtt{cp}}+1)(\mathtt{n}^v_{\mathtt{cp}}+1)$ control points defining the control net, w_{ij} are the corresponding weights and $S^{r,q}_{i,j}(u,v)$ are the two dimensional B-spline basis functions of degree r in u and q in v. The basis function in two dimensions is defined as a tensor product of one dimensional basis functions, written as

$$S^{r,q}_{i,j}(u,v) := C^r_i(u) C^q_j(v). \tag{A.6}$$

A.2 Delaunay graph method

The *Delaunay graph mapping* is an interpolation-based mesh morphing scheme [263]. It maps meshes of any topology to a Delaunay graph, which can subsequently be moved according to the geometric perturbation. The Delaunay graph method is comparable to the spring analogy method in terms of robustness and surpasses it in efficiency [275]. Moreover, it is non-interative and uses significantly less memory. Despite these advantages, the Delaunay graph has one major drawback. The quality of the mesh near the boundaries may deteriorate when large deformations are performed. To address this problem, an intermediate mesh is usually generated first to ensure a valid mesh before obtaining the mesh with the final perturbed geometry. However, this approach still encounters issues when the deformation involves large rotations. The Delaunay graph may lose its topology and becomes invalid. Researchers have combined the Delaunay graph with RBF [276] to improve the robustness of the original method when large rotational deformations are performed. It is reported that this approach preserves the near-wall mesh quality while maintaining the efficiency of the original Delaunay graph method [276]. As the work in this thesis involves only translational perturbations in the geometry, the original graph method, as formulated in [263], is implemented.

The Delaunay graph is generated based on the original mesh and it is used as an intermediate step where only the boundary nodes are involved. Performing mesh deformation using this method involves four main steps. The first step generates the Delaunay graph on the original mesh. The second step locates the mesh nodes in the graph by means of the barycentric coordinate system. The third step involves moving the Delaunay graph according to the boundary change due to the perturbation in the geometry. The fourth step transforms the mesh from the barycentric coordinates calculated in the second step, into the physical space using the new location of the Delaunay graph.

Consider a generic tetrahedral element, defined by the vertices A, B, C and D, containing an internal node P, as shown in figure A.3. The node P can be uniquely defined by the four relative volume coefficients e_i defined by the facets ABCD and the point P as

$$e_i = \frac{|\Omega_i|}{|\Omega|}, \tag{A.7}$$

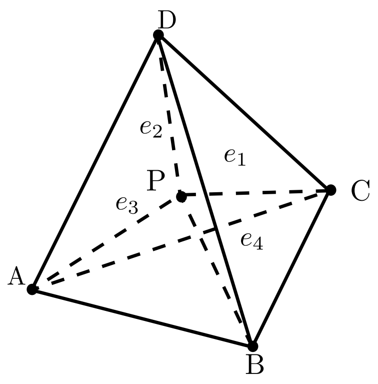


Figure A.3: A tetrahedral element with the annotated relative volume coefficients around node P.

for $i = 1, 2, 3, 4$. In the above expression, $|\Omega_i|$ denotes the individual volumes belonging to each quadrant and $|\Omega|$ is the total volume of the element. Given the calculated relative volume coefficients, the new internal points of coordinates $\boldsymbol{x}_P^\star$ in the volume mesh are updated using the coordinates of the deformed Delaunay graph, $\boldsymbol{x}^\star$, as

$$\boldsymbol{x}_P^\star = \boldsymbol{X}^\star \boldsymbol{e}^T \tag{A.8}$$

where $\boldsymbol{X}^\star = [\boldsymbol{x}_A^\star, \boldsymbol{x}_B^\star, \boldsymbol{x}_C^\star, \boldsymbol{x}_D^\star]$. A walk-through algorithm is needed to identify the Delaunay element in which each mesh node lies and the associated relative volume coefficients with that node. This has to be performed only once and represents the largest computational cost of this mesh morphing scheme. Additionally, it is important to note that the transformation from barycentric to the physical coordinates will remain valid as long as the boundary movement does not cause inversion or flipping, resulting in negative volumes. When this occurs, the procedure needs to be performed at an intermediate mesh movement step to ensure a valid mesh.

A.3 The cuckoo search

The cuckoo search (CS) method [277] is a technique that mimics a behaviour observed in nature. More precisely, the CS mimics the aggressive reproduction strategy of cuckoos. The strategy consists of laying the eggs at a nest of other birds, called hosts. When a host bird discovers that the eggs are not its own, it either disposes of the alien eggs or abandons the nest.

To translate this behaviour into an optimisation algorithm with an objective function $f(\boldsymbol{x})$, the method starts by generating a population of $\mathtt{n}_{\mathtt{egg}}$ eggs, namely $\{\boldsymbol{x}^i\}_{i=1,\ldots,\mathtt{n}_{\mathtt{egg}}}$, and placing them in a set of $\mathtt{n}_{\mathtt{hn}}$ host nests. Each solution is represented by an egg in a host nest and each new solution represents a cuckoo egg. For simplicity, it is assumed that each nest contains only one egg, so that $\mathtt{n}_{\mathtt{egg}} = \mathtt{n}_{\mathtt{hn}}$.

The eggs are ordered, such that $i < j$ if $F^i < F^j$, where $F^k = f(\boldsymbol{x}^k)$ is the fitness of the egg $\boldsymbol{x}^k$. To simulate the discovery of alien eggs, the fraction $\mathtt{p}_{\mathtt{a}}$ of the $\mathtt{n}_{\mathtt{egg}}$ eggs with lowest fitness are replaced by new eggs. The generation of new solutions (i.e., eggs) is performed by taking random walks [278] from a randomly selected nest. When the fitness of the new eggs is higher than the fitness of another randomly selected solution, the egg is moved to the new nest.

The original CS method [277] uses a random walk with a step length that is proportional to a Lévy distribution, namely

$$\boldsymbol{x}^{i,n+1} = \boldsymbol{x}^{i,n} + \boldsymbol{\alpha} \circ \Delta \boldsymbol{x}^i, \tag{A.9}$$

where α_k is a scaling factor, usually taken as a constant during the iterations, for the k-th component of $\boldsymbol{x}^i$, $\Delta \boldsymbol{x}$ is the step length of the random walk and $\circ$ denotes the Hadamard product. The components of the step length are computed using Mantenga's algorithm [278] as $x_r^i = u_r / |v_r|^{-\beta}$, where $\beta \in [1,2]$ and u_r and v_r are drawn from normal distributions with zero mean and standard deviations

$$\sigma_u = \left\{ \frac{\sin(\beta\pi/2)\Gamma[1+\beta]}{2^{(\beta-1)/2}\beta\Gamma[(\beta+1)/2]} \right\}^{-\beta} \tag{A.10}$$

and $\sigma_v = 1$ respectively. Γ denotes the Gamma function and the value $\beta = 1.5$ is used in this work.

A.3.1 Modified cuckoo search

In this work, the modified cuckoo search (MCS) method [75] is employed. The MCS has demonstrated a better robustness when compared to other optimisation algorithms, particularly in the context of high dimensional problems [75]. This technique introduces two modifications to the original CS algorithm that increase the convergence rate of the algorithm.

The first modification involves the definition of the scaling factor $\boldsymbol{\alpha}$. In the MCS it is taken as $\alpha_k = 1/n^{\delta}$, where n is the generation number and δ is a parameter to adjust the localisation of the search. This modification, inspired by the particle swarm optimisation, ensures a localised search as the generation number increases.

The second modification adds a new feature to the original CS method, which involves the exchange of information between solutions. The strategy consists of selecting a set of elite eggs, with the best fitness, as a percentage $\mathsf{p_b}$ of the total number of eggs. For each egg within the elite, another egg, is selected randomly in the same set. Denoting by $\boldsymbol{x}^i$ the egg with best fitness and the second egg by $\boldsymbol{x}^j$, a new solution is then generated as

$$\boldsymbol{x}^k = \boldsymbol{x}^i + \varphi^{-1} \frac{\boldsymbol{x}^j - \boldsymbol{x}^i}{\|\boldsymbol{x}^j - \boldsymbol{x}^i\|_2}, \tag{A.11}$$

where $\varphi = (1 + \sqrt{5})/2$ is the golden ratio. When the two eggs selected coincide, a local random walk with step size $1/n^2$ is used to create a nest for the second egg. In addition, when the fitness of both eggs selected is the same, the new solution is computed as the mid-point between $\boldsymbol{x}^i$ and $\boldsymbol{x}^j$.

The parameters for the MCS algorithm are taken from [75], namely $\mathsf{p_a} = 0.75$, $\mathsf{p_b} = 1 - \mathsf{p_a}$ and $\delta = 0.5$. In addition, to increase the efficiency of the method, a minimum number of sets is defined, equal to 10, and the number of nests is decreased by one in every generation.

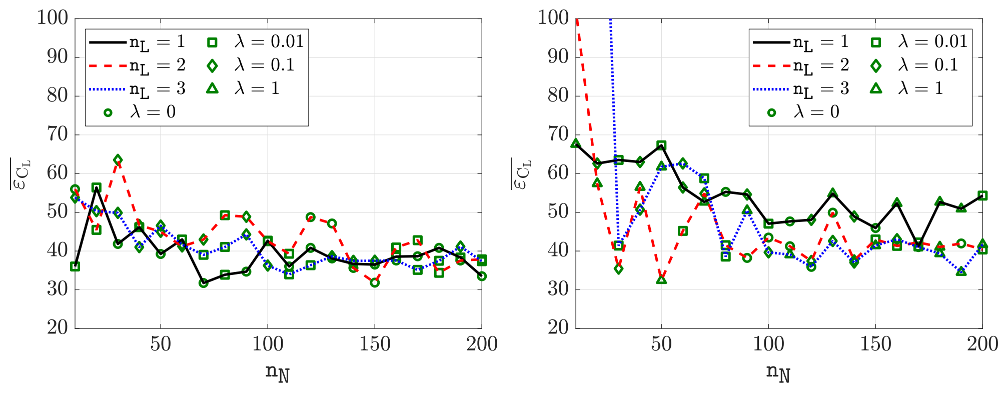
Appendix B

Comparison of two NN architectures

In order to defend the choice of NN architecture employed in this work, the performance of two types of NN architectures is considered and compared using the example described in section 4.1. The performance of the proposed NN, which employs the extended variant of a multi-layer perceptron, termed as MLP in this example, is compared against a more complex NN architecture, the long short term memory network, termed as LSTM [279]. To recall, the example considered the computation of the aerodynamic coefficients of a NACA0012 aerofoil at a free stream Mach number, M_∞, and an angle of attack, α, in predefined intervals $I_M = (0.3, 0.9)$ and $I_\alpha = (-5^\circ, 11^\circ)$, respectively. Here, only the performance in the predictions of the lift coefficient are compared.

The MLP considers M_∞ and α as inputs and the output is the pressure at a user defined set of points on the aerofoil. The set of points considered corresponds to the 300 mesh nodes used to discretise the aerofoil. The LSTM also considers M_∞ and α as inputs and the pressure defined at the 300 mesh nodes as outputs. Both NNs are trained using a dataset of $\mathtt{n_{Tr}} = 40$ simulations and tested using $\mathtt{n_{Te}} = 119$ simulations. To ensure a fair comparison in the training of the networks, the numerical parameters in the momentum gradient descent, which includes the learning rate, the momentum coefficient, and the cost function tolerance, are identical.

Figure B.1 shows the mean value of the error, measured in lift counts, as a function of the number of hidden neurons, $\mathtt{n_N}$, and the number of hidden layers, $\mathtt{n_L}$. Figure B.1(a) corresponds to the third network in section 4.1 of this thesis. The results show that



(a) MLP (b) LSTM

Figure B.1: Mean value of the error measured in lift counts, $\overline{\varepsilon_{C_L}}$, as a function of the number of hidden neurons, $\mathtt{n_N}$, and the number of hidden layers, $\mathtt{n_L}$.

the MLP is weakly dependent on the number of neurons in the hidden layers. The mean error has a tendency to decrease as the number of neurons is increased. The best accuracy obtained with this network is 31 lift counts when one hidden layer and 70 hidden neurons is employed. The LSTM network in figure B.1(b) is also capable of achieving a similar accuracy, however it can be observed to be more dependent on the number of hidden neurons and hidden layers. The LSTM is observed to provide more accurate results with two or three hidden layers using multiple combinations of hidden neurons compared to when one hidden layer is employed. The highest accuracy obtained with the LSTM is 32 lift counts when two LSTM layers and 50 hidden neurons are used.

Although LSTM is capable of predicting with similar accuracies to the MLP, this comes at the cost of optimising the additional weights in the complex LSTM model. For instance, the MLP obtains the highest accuracy when one hidden layer and $\mathtt{n_N} = 70$ hidden neurons are employed. This amounts to a total of 25, 131 weights in the network. The MLP achieves this accuracy in 115, 602 iterations. Conversely, the LSTM network optimises 48, 701 weights when two LSTM hidden layers are employed and $\mathtt{n_N} = 50$ neurons are employed. The network converges to the set tolerance in 513, 414 iterations. We can therefore deduce that LSTM can achieve the same accuracy when almost double the number of weights to the MLP are used. Additionally, it requires more than four times the number of iterations to converge to the set tolerance. As a result of these setbacks, the MLP is selected for the examples considered in this thesis.

Bibliography

[1] J. Jupp. “Wing aerodynamics and the science of compromise”. In: *The Aeronautical Journal* 105.1053 (2001), pp. 633–641.

[2] A. Krein and G. Williams. “Flightpath 2050: Europes vision for aeronautics”. In: *Innovation for Sustainable Aviation in a Global Environment: Proceedings of the Sixth European Aeronautics Days, Madrid* 30 (2012).

[3] B. K. *Perspectives for CFD*. DGLR-2002-013, DGLR Jahrbuch 2002, Band III, Germany, 2003.

[4] S. L. Brunton, J. N. Kutz, K. Manohar, A. Y. Aravkin, K. Morgansen, J. Klemisch, N. Goebel, J. Buttrick, J. Poskin, A. Blom-Schieber, T. Hogan, and D. McDonald. *Data-Driven Aerospace Engineering: Reframing the Industry with Machine Learning*. 2020. arXiv: `2008.10740 [cs.LG]`.

[5] A. Quarteroni and G. Rozza. *Reduced order methods for modeling and computational reduction*. Vol. 9. Springer, 2014.

[6] M. Fossati. “Evaluation of aerodynamic loads via reduced-order methodology”. In: *AIAA Journal* 53.8 (2015), pp. 2389–2405.

[7] D. J. Lucia, P. I. King, and P. S. Beran. “Reduced order modeling of a two-dimensional flow with moving shocks”. In: *Computers & Fluids* 32.7 (2003), pp. 917–938.

[8] P. Constantine and G Iaccarino. “Reduced order models for parameterized hyperbolic conservations laws with shock reconstruction”. In: *Center for Turbulence Research Annual Brief* (2012).

[9] C. Wales, A. Gaitonde, and D. Jones. “Reduced-order modeling of gust responses”. In: *Journal of Aircraft* 54.4 (2017), pp. 1350–1363.

[10] G. Berkooz, P. Holmes, and J. L. Lumley. “The proper orthogonal decomposition in the analysis of turbulent flows”. In: *Annual review of fluid mechanics* 25.1 (1993), pp. 539–575.

[11] T. Lieu, C. Farhat, and M. Lesoinne. “POD-based aeroelastic analysis of a complete F-16 configuration: ROM adaptation and demonstration”. In: *46th AIAA Structures, Structural Dynamics and Materials Conference*. 2005, p. 2295.

[12] S. Walton, O. Hassan, and K. Morgan. “Reduced order modelling for unsteady fluid flow using proper orthogonal decomposition and radial basis functions”. In: *Applied Mathematical Modelling* 37.20-21 (2013), pp. 8930–8945.

[13] F. Ballarin, A. D’Amario, S. Perotto, and G. Rozza. “A POD-selective inverse distance weighting method for fast parametrized shape morphing”. In: *International Journal for Numerical Methods in Engineering* 117.8 (2019), pp. 860–884.

[14] G. Rozza, D. B. P. Huynh, and A. T. Patera. “Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations”. In: *Archives of Computational Methods in Engineering* 15.3 (2008), pp. 229–275.

[15] J. S. Hesthaven, G. Rozza, and B. Stamm. *Certified reduced basis methods for parametrized partial differential equations*. Vol. 590. Springer, 2016.

[16] A. L. Gaitonde and D. Jones. “Study of linear response identification techniques and reduced-order model generation for a 2D CFD scheme”. In: *International journal for numerical methods in fluids* 52.12 (2006), pp. 1361–1402.

[17] A. K. Bagheri, D. P. Jones, and A. L. Gaitonde. “Linear Reduced-Order Model of Airfoil Gust Response”. In: *Journal of Aircraft* 56.3 (2019), pp. 1264–1271.

[18] K. Elsayed and C. Lacor. “CFD modeling and multi-objective optimization of cyclone geometry using desirability function, artificial neural networks and genetic algorithms”. In: *Applied Mathematical Modelling* 37.8 (2013), pp. 5680–5704.

[19] R. Franke. “Scattered data interpolation: tests of some methods”. In: *Mathematics of computation* 38.157 (1982), pp. 181–200.

[20] D. J. Linse and R. F. Stengel. “Identification of aerodynamic coefficients using computational neural networks”. In: *Journal of Guidance, Control, and Dynamics* 16.6 (1993), pp. 1018–1025.

[21] V. Tsiolakis, M. Giacomini, R. Sevilla, C. Othmer, and A. Huerta. “Nonintrusive proper generalised decomposition for parametrised incompressible flow problems in OpenFOAM”. In: *Computer physics communications* 249 (2020), p. 107013.

[22] L. Sirovich. “Turbulence and the dynamics of coherent structures. I. Coherent structures”. In: *Quarterly of applied mathematics* 45.3 (1987), pp. 561–571.

[23] E. Dowell, K. Hall, J. Thomas, R. Florea, B. Epureanu, and J. Heeg. "Reduced order models in unsteady aerodynamics". In: *40th Structures, Structural Dynamics, and Materials Conference and Exhibit*. 1999, p. 1261.

[24] K. Hall, J. Thomas, and E. Dowell. "Reduced-order modelling of unsteady small-disturbance flows using a frequency-domain proper orthogonal decomposition technique". In: *37th Aerospace Sciences Meeting and Exhibit*. 1999, p. 655.

[25] M. Romanowski. "Reduced order unsteady aerodynamic and aeroelastic models using Karhunen-Loeve eigenmodes". In: *6th Symposium on Multidisciplinary Analysis and Optimization*. 1996, p. 3981.

[26] D. J. Lucia, P. S. Beran, and W. A. Silva. "Reduced-order modeling: new approaches for computational physics". In: *Progress in aerospace sciences* 40.1-2 (2004), pp. 51–117.

[27] R. Yondo, E. Andrés, and E. Valero. "A review on design of experiments and surrogate models in aircraft real-time and many-query aerodynamic analyses". In: *Progress in aerospace sciences* 96 (2018), pp. 23–61.

[28] P. LeGresley and J. Alonso. "Airfoil design optimization using reduced order models based on proper orthogonal decomposition". In: *Fluids 2000 conference and exhibit*. 2000, p. 2545.

[29] L. Tang, P.-C. Chen, D. Liu, X.-W. Gao, W. Shyy, Y. Utturkar, and B.-N. Zhang. "Proper orthogonal decomposition and response surface method for tps/rlv structural design and optimization: X-34 case study". In: *43rd AIAA Aerospace Sciences Meeting and Exhibit*. 2005, p. 839.

[30] S. Walton, O. Hassan, and K. Morgan. "Selected engineering applications of gradient free optimisation using cuckoo search and proper orthogonal decomposition". In: *Archives of Computational Methods in Engineering* 20.2 (2013), pp. 123–154.

[31] M. Mifsud, S. Shaw, and D. MacManus. "A high-fidelity low-cost aerodynamic model using proper orthogonal decomposition". In: *International journal for numerical methods in fluids* 63.4 (2010), pp. 468–494.

[32] T Bui-Thanh, M. Damodaran, and K. Willcox. "Proper orthogonal decomposition extensions for parametric applications in compressible aerodynamics". In: *21st AIAA Applied Aerodynamics Conference*. 2003, p. 4213.

[33] V. Dolci and R. Arina. “Proper orthogonal decomposition as surrogate model for aerodynamic optimization”. In: *International Journal of Aerospace Engineering* 2016 (2016).

[34] E. Iuliano and D. Quagliarella. “Proper orthogonal decomposition, surrogate modelling and evolutionary optimization in aerodynamic design”. In: *Computers & Fluids* 84 (2013), pp. 327–350.

[35] T. Bui-Thanh, M. Damodaran, and K. E. Willcox. “Aerodynamic data reconstruction and inverse design using proper orthogonal decomposition”. In: *AIAA journal* 42.8 (2004), pp. 1505–1516.

[36] W. Chen, J. S. Hesthaven, B. Junqiang, Y. Qiu, Z. Yang, and Y. Tihao. “Greedy nonintrusive reduced order model for fluid dynamics”. In: *AIAA Journal* 56.12 (2018), pp. 4927–4943.

[37] T. Braconnier, M. Ferrier, J.-C. Jouhaud, M. Montagnac, and P. Sagaut. “Towards an adaptive POD/SVD surrogate model for aeronautic design”. In: *Computers & Fluids* 40.1 (2011), pp. 195–209.

[38] D. J. Lucia and P. S. Beran. “Projection methods for reduced order models of compressible flows”. In: *Journal of Computational Physics* 188.1 (2003), pp. 252–280.

[39] K. Willcox and J. Peraire. “Balanced model reduction via the proper orthogonal decomposition”. In: *AIAA journal* 40.11 (2002), pp. 2323–2330.

[40] J. Degroote, J. Vierendeels, and K. Willcox. “Interpolation among reduced-order matrices to obtain parameterized models for design, optimization and probabilistic analysis”. In: *International Journal for Numerical Methods in Fluids* 63.2 (2010), pp. 207–230.

[41] J. Laurenceau and P Sagaut. “Building efficient response surfaces of aerodynamic functions with kriging and cokriging”. In: *AIAA journal* 46.2 (2008), pp. 498–507.

[42] B. Rosenbaum and V. Schulz. *Comparing sampling strategies for aerodynamic Kriging surrogate models*. 2012.

[43] Y. Ju, C. Zhang, and L. Ma. “Artificial intelligence metamodel comparison and application to wind turbine airfoil uncertainty analysis”. In: *Advances in Mechanical Engineering* 8.5 (2016), p. 1687814016647317.

[44] J. S. Anttonen, P. I. King, and P. S. Beran. “Applications of multi-POD to a pitching and plunging airfoil”. In: *Mathematical and Computer Modelling* 42.3-4 (2005), pp. 245–259.

[45] M. Frank, D. Drikakis, and V. Charissis. “Machine-learning methods for computational science and engineering”. In: *Computation* 8.1 (2020), p. 15.

[46] G. Sun and S. Wang. “A review of the artificial neural network surrogate modeling in aerodynamic design”. In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 233.16 (2019), pp. 5863–5872.

[47] H. Chen, L. He, W. Qian, and S. Wang. “Multiple aerodynamic coefficient prediction of airfoils using a convolutional neural network”. In: *Symmetry* 12.4 (2020), p. 544.

[48] H. Jihong, H Su, and X Zhao. “Aerodynamic coefficient prediction of airfoil using BP neural network”. In: *Advances in Aeronautical Science and Engineering* 1.1 (2010), pp. 36–39.

[49] S Huang, L Miller, and J Steck. “An exploratory application of neural networks to airfoil design”. In: *32nd Aerospace Sciences Meeting and Exhibit*. 1994, p. 501.

[50] M. Santos, B. Mattos, and R. Girardi. “Aerodynamic coefficient prediction of airfoils using neural networks”. In: *46th AIAA aerospace sciences meeting and exhibit*. 2008, p. 887.

[51] M. Khurana, H. Winarto, and A. Sinha. “Application of swarm approach and artificial neural networks for airfoil shape optimization”. In: *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. 2008, p. 5954.

[52] G. Sun, Y. Sun, and S. Wang. “Artificial neural network based inverse design: Airfoils and wings”. In: *Aerospace Science and Technology* 42 (2015), pp. 415–428.

[53] A. Kharal and A. Saleem. “Neural networks based airfoil generation for a given Cp using Bezier–PARSEC parameterization”. In: *Aerospace science and Technology* 23.1 (2012), pp. 330–344.

[54] W. Song and A. Keane. “A study of shape parameterisation methods for airfoil optimisation”. In: *10th AIAA/ISSMO multidisciplinary analysis and optimization conference*. 2004, p. 4482.

[55] D. A. Masters, N. J. Taylor, T Rendall, C. B. Allen, and D. J. Poole. “Review of aerofoil parameterisation methods for aerodynamic shape optimisation”. In: *53rd AIAA Aerospace Sciences Meeting*. 2015, p. 0761.

[56] W. Chen, K. Chiu, and M. D. Fuge. “Airfoil Design Parameterization and Optimization Using Bézier Generative Adversarial Networks”. In: *AIAA Journal* 58.11 (2020), pp. 4723–4735.

[57] M Rai. “Three-dimensional aerodynamic design using artificial neural networks”. In: *40th AIAA Aerospace Sciences Meeting & Exhibit*. 2002, p. 987.

[58] F. Mazhar, A. M. Khan, I. A. Chaudhry, and M. Ahsan. “On using neural networks in UAV structural design for CFD data fitting and classification”. In: *Aerospace Science and Technology* 30.1 (2013), pp. 210–225.

[59] V. Sekar, Q. Jiang, C. Shu, and B. C. Khoo. “Fast flow field prediction over airfoils using deep learning approach”. In: *Physics of Fluids* 31.5 (2019), p. 057103.

[60] J. Yu and J. S. Hesthaven. “Flowfield reconstruction method using artificial neural network”. In: *Aiaa Journal* 57.2 (2019), pp. 482–498.

[61] Y. Zhang, W. J. Sung, and D. N. Mavris. “Application of convolutional neural network to predict airfoil lift coefficient”. In: *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. 2018, p. 1903.

[62] E. Yilmaz and B. German. “A Convolutional Neural Network Approach to Training Predictors for Airfoil Performance”. In: *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. American Institute of Aeronautics and Astronautics, 2017.

[63] S. Albawi, T. A. Mohammed, and S. Al-Zawi. “Understanding of a convolutional neural network”. In: *2017 International Conference on Engineering and Technology (ICET)*. Ieee. 2017, pp. 1–6.

[64] W. Rawat and Z. Wang. “Deep convolutional neural networks for image classification: A comprehensive review”. In: *Neural computation* 29.9 (2017), pp. 2352–2449.

[65] A. Kamilaris and F. X. Prenafeta-Boldú. “A review of the use of convolutional neural networks in agriculture”. In: *The Journal of Agricultural Science* 156.3 (2018), pp. 312–322.

[66] E. Ulu, R. Zhang, and L. B. Kara. “A data-driven investigation and estimation of optimal topologies under variable loading configurations”. In: *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization* 4.2 (2016), pp. 61–72.

[67] J. S. Hesthaven and S. Ubbiali. “Non-intrusive reduced order modeling of nonlinear problems using neural networks”. In: *Journal of Computational Physics* 363 (2018), pp. 55–78.

[68] R. Swischuk, L. Mainini, B. Peherstorfer, and K. Willcox. “Projection-based model reduction: Formulations for physics-based machine learning”. In: *Computers & Fluids* 179 (2019), pp. 704–717.

[69] Z. Han, K. Zhang, W. Song, and J. Liu. “Surrogate-based aerodynamic shape optimization with application to wind turbine airfoils”. In: *51st AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*. 2013, p. 1108.

[70] R. M. Hicks and P. A. Henne. “Wing design by numerical optimization”. In: *Journal of Aircraft* 15.7 (1978), pp. 407–412.

[71] A. Jameson. “Aerodynamic design via control theory”. In: *Journal of scientific computing* 3.3 (1988), pp. 233–260.

[72] D. W. Zingg, M. Nemec, and T. H. Pulliam. “A comparative evaluation of genetic and gradient-based algorithms applied to aerodynamic optimization”. In: *European Journal of Computational Mechanics/Revue Européenne de Mécanique Numérique* 17.1-2 (2008), pp. 103–126.

[73] J. McCall. “Genetic algorithms for modelling and optimisation”. In: *Journal of Computational and Applied Mathematics* 184.1 (2005), pp. 205–222.

[74] J. Kennedy and R. Eberhart. “Particle swarm optimization”. In: *Proceedings of IEEE International Conference on Neural Networks*. 1995, pp. 1942–1948.

[75] S Walton, O Hassan, K Morgan, and M. Brown. “Modified cuckoo search: a new gradient free optimisation algorithm”. In: *Chaos, Solitons & Fractals* 44.9 (2011), pp. 710–718.

[76] E. J. Whitney, M. Sefrioui, K. Srinivas, and J. Périaux. “Advances in hierarchical, parallel evolutionary algorithms for aerodynamic shape optimisation”. In: *JSME International Journal Series B Fluids and Thermal Engineering* 45.1 (2002), pp. 23–28.

[77] S. Mirjalili, J. S. Dong, A. Lewis, and A. S. Sadiq. “Particle swarm optimization: theory, literature review, and application in airfoil design”. In: *Nature-inspired optimizers* (2020), pp. 167–184.

[78] D. Naumann, B Evans, S Walton, and O Hassan. “A novel implementation of computational aerodynamic shape optimisation using modified cuckoo search”. In: *Applied Mathematical Modelling* 40.7-8 (2016), pp. 4543–4559.

[79] E. Iuliano and D. Quagliarella. “Aerodynamic design with physics-based surrogates”. In: *Springer Handbook of Computational Intelligence*. Springer, 2015, pp. 1185–1209.

[80] H. Kwon and S. Choi. “A trended Kriging model with R2 indicator and application to design optimization”. In: *Aerospace Science and Technology* 43 (2015), pp. 111–125.

[81] J. Mao, D. Hu, D. Li, R. Wang, and J. Song. “Novel adaptive surrogate model based on LRPIM for probabilistic analysis of turbine disc”. In: *Aerospace Science and Technology* 70 (2017), pp. 76–87.

[82] V. Sekar, M. Zhang, C. Shu, and B. C. Khoo. “Inverse Design of Airfoil Using a Deep Convolutional Neural Network”. In: *AIAA Journal* 57.3 (2019), pp. 993–1003.

[83] M. M. Rai and N. K. Madavan. “Aerodynamic design using neural networks”. In: *AIAA journal* 38.1 (2000), pp. 173–182.

[84] L. Hu, J. Zhang, Y. Xiang, and W. Wang. “Neural networks-based aerodynamic data modeling: A comprehensive review”. In: *IEEE Access* 8 (2020), pp. 90805–90823.

[85] J. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, and D. Mavriplis. *CFD vision 2030 study: a path to revolutionary computational aerosciences*. 2014.

[86] S. Suresh, S. Omkar, V. Mani, and T. G. Prakash. “Lift coefficient prediction at high angle of attack using recurrent neural network”. In: *Aerospace Science and Technology* 7.8 (2003), pp. 595–602.

[87] N. R. Secco and B. S. Mattos. “Artificial Neural Networks Applied to Airplane Design”. In: *53rd AIAA Aerospace Sciences Meeting*. 2015, p. 1013.

[88] O. Hassan, K. Morgan, and N. Weatherill. *Flite system version 4 Theoretical Manual*. Swansea University. 2009.

[89] A. Thom. “The flow past circular cylinders at low speeds”. In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 141.845 (1933), pp. 651–669.

[90] J. M. McDonough. "Introductory lectures on turbulence: physics, mathematics and modeling". In: (2007).

[91] K. V. Belyaev, A. V. Garbaruk, M. L. Shur, M. K. Strelets, and P. R. Spalart. "Experience of direct numerical simulation of turbulence on supercomputers". In: *Russian Supercomputing Days*. Springer. 2016, pp. 67–77.

[92] J. F. Wendt. *Computational fluid dynamics: an introduction*. Springer Science & Business Media, 2008.

[93] K. Morgan, J. Peraire, J. Peiro, and O. Hassan. "The computation of three-dimensional flows using unstructured grids". In: *Computer Methods in Applied Mechanics and Engineering* 87.2-3 (1991), pp. 335–352.

[94] K. A. Sørensen, O. Hassan, K. Morgan, and N. P. Weatherill. "A multigrid accelerated hybrid unstructured mesh method for 3D compressible turbulent flow". In: *Computational Mechanics* 31.1-2 (2003), pp. 101–114.

[95] R. Sevilla and S. Fernández-Méndez. "Numerical integration over 2D NURBS-shaped domains with applications to NURBS-enhanced FEM". In: *Finite Elements in Analysis and Design* 47.10 (2011), pp. 1209–1220.

[96] R. Sevilla, S. Fernández-Méndez, and A. Huerta. "Comparison of high-order curved finite elements". In: *International Journal for Numerical Methods in Engineering* 87.8 (2011), pp. 719–734.

[97] R. Sevilla and A. Huerta. "HDG-NEFEM with degree adaptivity for Stokes flows". In: *Journal of Scientific Computing* 77.3 (2018), pp. 1953–1980.

[98] D. C. Wilcox. *Turbulence modeling for CFD*. Vol. 2. DCW industries La Canada, CA, 1998.

[99] M. Wolfshtein. "Some comments on turbulence modelling". In: *International journal of heat and mass transfer* 52.17-18 (2009), pp. 4103–4107.

[100] P. R. Spalart. "Strategies for turbulence modelling and simulations". In: *International journal of heat and fluid flow* 21.3 (2000), pp. 252–263.

[101] P. Spalart, W. Jou, M Strelets, and S. Allmaras. "Comments on the feasibility of LES for wings, and on a hybrid RANS/LES approach." In: *Conf. on DNS/LES*. 1997.

[102] S. Krajnović. "Large eddy simulation of flows around ground vehicles and other bluff bodies". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* (2009).

[103] A. Hutton. “The emerging role of large eddy simulation in industrial practice: challenges and opportunities”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* (2009).

[104] O. Reynolds. “On the dynamical theory of incompressible viscous fluids andthe determination of the criterion”. In: *Proceedings of the Royal Society of London* 56.336-339 (1894), pp. 40–45.

[105] A. Favre. *The equations of compressible turbulent gases*. Tech. rep. Aix-Marseille Univ (France) Inst de Mecanique Statisque de la Turbulence, 1965.

[106] M. V. Morkovin. “Effects of compressibility on turbulent flows”. In: *Mécanique de la Turbulence* (1962).

[107] J. Blazek. *Computational fluid dynamics: principles and applications*. Butterworth-Heinemann, 2015.

[108] P Bradshaw. “Possible origin of Prandt's mixing-length theory”. In: *Nature* 249.5453 (1974), pp. 135–136.

[109] A. Smith and T. Cebeci. *Numerical solution of the turbulent-boundary-layer equations*. Tech. rep. Douglas Aircraft Co Long Beach CA, 1967.

[110] B. Baldwin and H. Lomax. “Thin-layer approximation and algebraic model for separated turbulentflows”. In: *16th aerospace sciences meeting*. 1978, p. 257.

[111] B. Baldwin and T. Barth. “A one-equation turbulence transport model for high Reynolds number wall-bounded flows”. In: *29th Aerospace Sciences Meeting*. 1991, p. 610.

[112] P. Spalart and S. Allmaras. “A One-Equation Turbulence Model for Aerodynamic Flows”. In: *AIAA* 439 (Jan. 1992).

[113] J. E. Bardina, P. G. Huang, and T. J. Coakley. “Turbulence modeling validation, testing, and development”. In: (1997).

[114] V. A. Sai and F. M. Lutfy. “Analysis of the Baldwin-Barth and Spalart-Allmaras one-equation turbulence model”. In: *AIAA journal* 33.10 (1995), pp. 1971–1974.

[115] B. E. Launder and B. Sharma. “Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc”. In: *Letters in heat and mass transfer* 1.2 (1974), pp. 131–137.

[116] C. G. Speziale, R. Abid, and E. C. Anderson. “Critical evaluation of two-equation models for near-wall turbulence”. In: *AIAA journal* 30.2 (1992), pp. 324–331.

[117] F. R. Menter. “Two-equation eddy-viscosity turbulence models for engineering applications”. In: *AIAA journal* 32.8 (1994), pp. 1598–1605.

[118] F. Menter and C. Rumsey. “Assessment of two-equation turbulence models for transonic flows”. In: *Fluid Dynamics Conference*. 1994, p. 2343.

[119] P. Spalart and S. Allmaras. “A one-equation turbulence model for aerodynamic flows”. In: *30th aerospace sciences meeting and exhibit*. 1992, p. 439.

[120] Y. Tamura and K. Fujii. “Conservation law for moving and transformed grids”. In: *11th Computational Fluid Dynamics Conference*. 1993, p. 3365.

[121] B. Nkonga and H. Guillard. “Godunov type method on non-structured meshes for three-dimensional moving boundary problems”. In: *Computer methods in applied mechanics and engineering* 113.1-2 (1994), pp. 183–204.

[122] V. D. Liseikin. *Grid generation methods*. Vol. 1. Springer, 1999.

[123] N. Weatherill. “A method for generating irregular computational grids in multiply connected planar domains”. In: *International Journal for Numerical Methods in Fluids* 8.2 (1988), pp. 181–197.

[124] J. Peraire, M. Vahdati, K. Morgan, and O. C. Zienkiewicz. “Adaptive remeshing for compressible flow computations”. In: *Journal of computational physics* 72.2 (1987), pp. 449–466.

[125] N. P. Weatherill and O. Hassan. “Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints”. In: *International Journal for Numerical Methods in Engineering* 37.12 (1994), pp. 2005–2039.

[126] D. Smith, M. Lowenberg, D. Jones, and M. Friswell. “Computational and experimental validation of the active morphing wing”. In: *Journal of Aircraft* 51.3 (2014), pp. 925–937.

[127] G. L. Dirichlet. “Über die Reduction der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen.” In: *Journal für die reine und angewandte Mathematik* 1850.40 (1850), pp. 209–227.

[128] J. F. Thompson, B. K. Soni, and N. P. Weatherill. *Handbook of grid generation*. CRC press, 1998.

[129] P. J. Green and R. Sibson. “Computing Dirichlet tessellations in the plane”. In: *The computer journal* 21.2 (1978), pp. 168–173.

[130] J. A. George. *Computer implementation of the finite element method*. Tech. rep. Department of Computer Science Stanford University, 1971.

[131] J. Peraire, J. Peiro, L. Formaggia, K. Morgan, and O. C. Zienkiewicz. “Finite element Euler computations in three dimensions”. In: *International Journal for Numerical Methods in Engineering* 26.10 (1988), pp. 2135–2159.

[132] O. Hassan, E. Probert, K Morgan, and J Peraire. “Mesh generation and adaptivity for the solution of compressible viscous high speed flows”. In: *International journal for numerical methods in engineering* 38.7 (1995), pp. 1123–1148.

[133] O. Hassan, K Morgan, E. Probert, and J Peraire. “Unstructured tetrahedral mesh generation for three-dimensional viscous flows”. In: *International Journal for Numerical Methods in Engineering* 39.4 (1996), pp. 549–567.

[134] R. Eymard, T. Gallouët, and R. Herbin. “Finite volume methods”. In: *Handbook of numerical analysis* 7 (2000), pp. 713–1018.

[135] K Morgan and J Peraire. “Unstructured grid finite-element methods for fluid mechanics”. In: *Reports on Progress in Physics* 61.6 (1998), p. 569.

[136] N. Kroll and J. K. Fassbender, eds. *MEGAFLOW - Numerical Flow Simulation for Aircraft Design*. Springer Berlin Heidelberg, 2005.

[137] P. I. Crumpton, P. Moinier, and M. B. Giles. “An unstructured algorithm for high Reynolds number flows on highly stretched grids”. In: *Proceedings of the 10th International Conference on Numerical Methods for Laminar and Turbulent Flows*. Vol. 21. 1997, p. 25.

[138] T. J. Barth. “Aspects of unstructured grids and finite-volume solvers for the Euler and Navier-Stokes equations”. In: *AGARD* (1992).

[139] R. Sevilla, L. M. Vieira, M. Giacomini, and A. Huerta. “A second-order face-centred finite volume method for elliptic problems”. In: *Computer Methods in Applied Mechanics and Engineering* 358 (2020), p. 112655.

[140] B. Diskin, J. L. Thomas, E. J. Nielsen, H. Nishikawa, and J. A. White. “Comparison of node-centered and cell-centered unstructured finite-volume discretizations: viscous fluxes”. In: *AIAA journal* 48.7 (2010), pp. 1326–1338.

[141] K. Sørensen, O Hassan, K Morgan, and N. Weatherill. “A multigrid accelerated time-accurate inviscid compressible fluid flow solution algorithm employing mesh movement and local remeshing”. In: *International journal for numerical methods in fluids* 43.5 (2003), pp. 517–536.

[142] R. Chima, E. Turkel, and S. Schaffer. “Comparison of three explicit multigrid methods for the Euler and Navier-Stokes equations”. In: *25th AIAA Aerospace Sciences Meeting*. 1987, p. 602.

[143] A. Jameson, W. Schmidt, and E. Turkel. “Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes”. In: *14th fluid and plasma dynamics conference*. 1981, p. 1259.

[144] R. Seanson and E. Turkel. *Multistage schemes with multigrid for Euler and Navier-Stokes equations*. NASA, Langley Research Center, 1997.

[145] F. Chinesta, E. Cueto, and A. Huerta. “PGD for solving multidimensional and parametric models”. In: *Separated representations and PGD-based model reduction*. Vol. 554. CISM Courses and Lectures. Springer, Vienna, 2014, pp. 27–89.

[146] F. Chinesta, R. Keunings, and A. Leygue. *The proper generalized decomposition for advanced numerical simulations. A primer*. Springer Briefs in Applied Sciences and Technology. Springer, Cham, 2014, p. 117.

[147] R. Sevilla, S. Zlotnik, and A. Huerta. “Solution of geometrically parametrised problems within a CAD environment via model order reduction”. In: *Computer Methods in Applied Mechanics and Engineering* 358 (2020), p. 112631.

[148] J.-N. Juang and R. S. Pappa. “An eigensystem realization algorithm for modal parameter identification and model reduction”. In: *Journal of guidance, control, and dynamics* 8.5 (1985), pp. 620–627.

[149] F. Chinesta, A. Huerta, G. Rozza, and K. Willcox. “Model order reduction”. In: *Encyclopedia of Computational Mechanics* (2016).

[150] X. Zou, M. Conti, P. Díez, and F. Auricchio. “A nonintrusive proper generalized decomposition scheme with application in biomechanics”. In: *International Journal for Numerical Methods in Engineering* 113.2 (2018), pp. 230–251.

[151] A. Leon, S. Mueller, P. de Luca, R. Said, J.-L. Duval, and F. Chinesta. “Non-intrusive proper generalized decomposition involving space and parameters: application to the mechanical modeling of 3D woven fabrics”. In: *Advanced Modeling and Simulation in Engineering Sciences* 6.1 (2019), p. 13.

[152] P.-E. Allier, L. Chamoin, and P. Ladevèze. “Proper generalized decomposition computational methods on a benchmark problem: introducing a new strategy based on constitutive relation error minimization”. In: *Advanced Modeling and Simulation in Engineering Sciences* (2015).

[153] M Amabili and C. Touzé. “Reduced-order models for nonlinear vibrations of fluid-filled circular cylindrical shells: comparison of POD and asymptotic non-linear normal modes methods”. In: *Journal of fluids and structures* 23.6 (2007), pp. 885–903.

[154] J. Burkardt, M. Gunzburger, and H.-C. Lee. “POD and CVT-based reduced-order modeling of Navier–Stokes flows”. In: *Computer methods in applied mechanics and engineering* 196.1-3 (2006), pp. 337–355.

[155] M. I. Jordan and T. M. Mitchell. “Machine learning: Trends, perspectives, and prospects”. In: *Science* 349.6245 (2015), pp. 255–260.

[156] M. J. Khoury and J. P. Ioannidis. “Big data meets public health”. In: *Science* 346.6213 (2014), pp. 1054–1055.

[157] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben. “Machine learning in manufacturing: advantages, challenges, and applications”. In: *Production & Manufacturing Research* 4.1 (2016), pp. 23–45.

[158] R. Baker. “Data mining for education”. In: *International encyclopedia of education* 7.3 (2010), pp. 112–118.

[159] L. Einav and J. Levin. “Economics in the age of big data”. In: *Science* 346.6210 (2014).

[160] G. Cui, M. L. Wong, and H.-K. Lui. “Machine learning for direct marketing response models: Bayesian networks with evolutionary programming”. In: *Management Science* 52.4 (2006), pp. 597–612.

[161] J. E. Van Engelen and H. H. Hoos. “A survey on semi-supervised learning”. In: *Machine Learning* 109.2 (2020), pp. 373–440.

[162] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar. “Integrating physics-based modeling with machine learning: A survey”. In: *arXiv preprint arXiv:2003.04919* (2020).

[163] L. Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[164] S. L. Brunton and J. N. Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2019.

[165] X. Wu. “Top 10 algorithms in data mining”. In: *Knowledge and Information Systems* (2008).

[166] S. Sharma, J. Agrawal, S. Agarwal, and S. Sharma. “Machine learning techniques for data mining: A survey”. In: *2013 IEEE International Conference on Computational Intelligence and Computing Research*. IEEE. 2013, pp. 1–6.

[167] R. Gholami and N. Fakhari. “Support vector machine: principles, parameters, and applications”. In: *Handbook of Neural Computation*. Elsevier, 2017, pp. 515–535.

[168] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*. Vol. 1. MIT press Cambridge, 2016.

[169] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators." In: *Neural networks* 2.5 (1989), pp. 359–366.

[170] L. Torrey and J. Shavlik. "Transfer learning". In: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264.

[171] J. Flynn and C. Giannetti. "Using Convolutional Neural Networks to Map Houses Suitable for Electric Vehicle Home Charging". In: *AI* 2.1 (2021), pp. 135–149.

[172] T. Kohonen. "The self-organizing map". In: *Proceedings of the IEEE* 78.9 (1990), pp. 1464–1480.

[173] L. Cao, K. S. Chua, W. Chong, H. Lee, and Q. Gu. "A comparison of PCA, KPCA and ICA for dimensionality reduction in support vector machine". In: *Neurocomputing* 55.1-2 (2003), pp. 321–336.

[174] R. Rosipal, M. Girolami, L. J. Trejo, and A. Cichocki. "Kernel PCA for feature extraction and de-noising in nonlinear regression". In: *Neural Computing & Applications* 10.3 (2001), pp. 231–243.

[175] B. Schölkopf, S. Mika, A. Smola, G. Rätsch, and K.-R. Müller. "Kernel PCA pattern reconstruction via approximate pre-images". In: *International Conference on Artificial Neural Networks*. Springer. 1998, pp. 147–152.

[176] M. A. Kramer. "Nonlinear principal component analysis using autoassociative neural networks". In: *AIChE journal* 37.2 (1991), pp. 233–243.

[177] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1096–1103.

[178] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[179] D. Berthelot, N. Carlini, I. Goodfellow, N. Papernot, A. Oliver, and C. Raffel. "Mixmatch: A holistic approach to semi-supervised learning". In: *arXiv preprint arXiv:1905.02249* (2019).

[180] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. "Improved techniques for training gans". In: *Advances in neural information processing systems* 29 (2016), pp. 2234–2242.

[181] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. “Semi-supervised learning with deep generative models”. In: *Advances in neural information processing systems*. 2014, pp. 3581–3589.

[182] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[183] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[184] S. L. Brunton, B. R. Noack, and P. Koumoutsakos. “Machine Learning for Fluid Mechanics”. In: *Annual Review of Fluid Mechanics* 52.1 (2020), pp. 477–508.

[185] S. Verma, G. Novati, and P. Koumoutsakos. “Efficient collective swimming by harnessing vortices through deep reinforcement learning”. In: *Proceedings of the National Academy of Sciences* 115.23 (2018), pp. 5849–5854.

[186] G. Kerschen, J.-c. Golinval, A. F. Vakakis, and L. A. Bergman. “The method of proper orthogonal decomposition for dynamical characterization and order reduction of mechanical systems: an overview”. In: *Nonlinear dynamics* 41.1-3 (2005), pp. 147–169.

[187] M. Bergmann and L. Cordier. “Optimal control of the cylinder wake in the laminar regime by trust-region methods and POD reduced-order models”. In: *Journal of Computational Physics* 227.16 (2008), pp. 7813–7840.

[188] P. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley. *Turbulence, coherent structures, dynamical systems and symmetry*. Cambridge university press, 2012.

[189] M. V. Tabib and J. B. Joshi. “Analysis of dominant flow structures and their flow dynamics in chemical process equipment using snapshot proper orthogonal decomposition technique”. In: *Chemical Engineering Science* 63.14 (2008), pp. 3695–3715.

[190] F. Fang, C. Pain, I. Navon, G. Gorman, M. Piggott, P. Allison, P. Farrell, and A. Goddard. “A POD reduced order unstructured mesh ocean modelling method for moderate Reynolds number flows”. In: *Ocean Modelling* 28.1-3 (2009), pp. 127–136.

[191] J. Rambo and Y. Joshi. “Reduced-order modeling of turbulent forced convection with parametric conditions”. In: *International journal of heat and mass transfer* 50.3-4 (2007), pp. 539–551.

[192] Y. Cao, J. Zhu, Z. Luo, and I. M. Navon. “Reduced-order modeling of the upper tropical pacific ocean model using proper orthogonal decomposition”. In: *Computers & mathematics with Applications* 52.8-9 (2006), pp. 1373–1386.

[193] A. Iollo, S. Lanteri, and J.-A. Désidéri. “Stability properties of POD–Galerkin approximations for the compressible Navier–Stokes equations”. In: *Theoretical and Computational Fluid Dynamics* 13.6 (2000), pp. 377–396.

[194] C. Huang, W. E. Anderson, and C. Merkle. “Exploration of POD-Galerkin Techniques for Developing Reduced Order Models of the Euler Equations”. In: *AIAA Modeling and Simulation Technologies Conference*. 2016, p. 1917.

[195] D Xiao, F Fang, J Du, C. Pain, I. Navon, A. Buchan, A. H. Elsheikh, and G Hu. “Non-linear Petrov–Galerkin methods for reduced order modelling of the Navier–Stokes equations using a mixed finite element pair”. In: *Computer Methods In Applied Mechanics and Engineering* 255 (2013), pp. 147–157.

[196] Y. Wang, B. Yu, Z. Cao, W. Zou, and G. Yu. “A comparative study of POD interpolation and POD projection methods for fast and accurate prediction of heat transfer problems”. In: *International Journal of Heat and Mass Transfer* 55.17-18 (2012), pp. 4827–4836.

[197] Y.-G. Wang, Z.-N. Li, B. Gong, and Q.-S. Li. “Reconstruction & prediction of wind pressure on heliostat”. In: *Kongqi Donglixue Xuebao/Acta Aerodynamica Sinica* 27.5 (2009), pp. 586–591.

[198] D Xiao, F Fang, C Pain, and G Hu. “Non-intrusive reduced-order modelling of the Navier–Stokes equations based on RBF interpolation”. In: *International Journal for Numerical Methods in Fluids* 79.11 (2015), pp. 580–595.

[199] M. D. Buhmann. *Radial basis functions: theory and implementations*. Vol. 12. Cambridge university press, 2003.

[200] M. Sharan, E. Kansa, and S. Gupta. “Application of the multiquadric method for numerical solution of elliptic partial differential equations”. In: (1997).

[201] W. Costin and C. Allen. “Numerical study of radial basis function interpolation for data transfer across discontinuous mesh interfaces”. In: *International Journal for Numerical Methods in Fluids* 72.10 (2013), pp. 1076–1095.

[202] A. K. Michler. “Aircraft control surface deflection using RBF-based mesh deformation”. In: *International Journal for Numerical Methods in Engineering* 88.10 (2011), pp. 986–1007.

[203] Z. Majdisova and V. Skala. “Radial basis function approximations: comparison and applications”. In: *Applied Mathematical Modelling* 51 (2017), pp. 728–743.

[204] B. Fornberg and N. Flyer. “Accuracy of radial basis function interpolation and derivative approximations on 1-D infinite grids”. In: *Advances in Computational Mathematics* 23.1-2 (2005), pp. 5–20.

[205] V. Shcherbakov and E. Larsson. “Radial basis function partition of unity methods for pricing vanilla basket options”. In: *Computers & Mathematics with Applications* 71.1 (2016), pp. 185–200.

[206] M. D. Buhmann. “Radial functions on compact support”. In: *Proceedings of the Edinburgh Mathematical Society* 41.1 (1998), pp. 33–46.

[207] M. Buhmann. “A new class of radial basis functions with compact support”. In: *Mathematics of Computation* 70.233 (2001), pp. 307–318.

[208] F. C. Menandro, C. F. L. Neto, H. J. Meira, and R. P. Bastos. “Compactly supported radial basis functions for function interpolation: Comparative behaviour”. In: *Mecánica Computacional* 29.47 (2010), pp. 4733–4752.

[209] D. Broomhead and D Lowe. “Multivariable functional interpolation and adaptive networks, complex systems, vol. 2”. In: (1988).

[210] T. Krishnamurthy. “Comparison of response surface construction methods for derivative estimation using moving least squares, kriging and radial basis functions”. In: *46th AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference*. 2005, p. 1821.

[211] S. Rippa. “An algorithm for selecting a good value for the parameter c in radial basis function interpolation”. In: *Advances in Computational Mathematics* 11.2-3 (1999), pp. 193–210.

[212] R. Sadiq, M. J. Rodriguez, and H. R. Mian. “Empirical models to predict disinfection by-products (DBPs) in drinking water: an updated review”. In: (2019).

[213] M. M. Nelson and W. T. Illingworth. *A practical guide to neural nets*. Reading, MA (USA); Addison-Wesley Publishing Co., Inc., 1991.

[214] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak. “Convolutional, long short-term memory, fully connected deep neural networks”. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2015, pp. 4580–4584.

[215] M. Hagan, H. Demuth, M. Beale, and O. DeJesus. *Neural network design, 2nd edition*. Martin Hagan, 2014.

[216] A. R. Zamir, T.-L. Wu, L. Sun, W. B. Shen, B. E. Shi, J. Malik, and S. Savarese. "Feedback networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1308–1317.

[217] S. Trenn. "Multilayer perceptrons: Approximation order and necessary number of hidden units". In: *IEEE transactions on neural networks* 19.5 (2008), pp. 836–844.

[218] P. Mazzatorta, E. Benfenati, D. Neagu, and G. Gini. "The importance of scaling in data mining for toxicity prediction". In: *Journal of chemical information and computer sciences* 42.5 (2002), pp. 1250–1255.

[219] B. Karlik and A. V. Olgac. "Performance analysis of various activation functions in generalized MLP architectures of neural networks". In: *International Journal of Artificial Intelligence and Expert Systems* 1.4 (2011), pp. 111–122.

[220] S. Wang, T. Zhou, and J. Bilmes. "Bias also matters: Bias attribution for deep neural network explanation". In: *International Conference on Machine Learning*. 2019, pp. 6659–6667.

[221] T. Amaral, L. M. Silva, L. A. Alexandre, C. Kandaswamy, J. M. Santos, and J. M. de Sá. "Using different cost functions to train stacked auto-encoders". In: *2013 12th Mexican international conference on artificial intelligence*. IEEE. 2013, pp. 114–120.

[222] E. Abounoori and M. Ali Heydari. "Comparison of Kullback-Leibler, Hellinger and LINEX with Quadratic Loss Function in Bayesian Dynamic Linear Models: Forecasting of Real Price of Oil". In: *International Journal of Business and Development Studies* 12.1 (2020), pp. 21–39.

[223] Y. Liu, H. Zhang, X. Zhang, and Y. Cao. "Investigation of Cost Function for Supervised Monaural Speech Separation." In: *INTERSPEECH*. 2019, pp. 3178–3182.

[224] D. Pham and D. Karaboga. *Intelligent optimisation techniques: genetic algorithms, tabu search, simulated annealing and neural networks*. Springer Science & Business Media, 2012.

[225] W. Brogan. *Modern control theory*. 3rd Ed. Englewood cliffs,NJ: Prentice-Hall, 1991.

[226] C. A. Floudas and P. M. Pardalos. *Encyclopedia of optimization*. Springer Science & Business Media, 2008.

[227] L. Scales. *Introduction to non-linear optimization*. Macmillan Education UK, 1985.

[228] M. Hestenes and E. Stiefel. “Methods of conjugate gradients for solving linear systems”. In: *Journal of Research of the National Bureau of Standards* 49.6 (1952).

[229] I. Sutskever, J. Martens, G. Dahl, and E. Geoffrey. “On the importance of initialisation and momentum in deep learning”. In: *Proceedings of the 30th international conference on machine learning (ICML-13)*. Vol. 28. Atlanta, 2013, pp. 1139–1147.

[230] J. L. Holi and J. Hwang. “Finite precision error analysis of neural network hardware implementations”. In: *IEEE Transactions on Computers* 42.3 (1993), pp. 281–290.

[231] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[232] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *arXiv preprint arXiv:1207.0580* (2012).

[233] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. “Large-scale video classification with convolutional neural networks”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2014, pp. 1725–1732.

[234] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[235] Y. Liu, J. A. Starzyk, and Z. Zhu. “Optimized approximation algorithm in neural networks without overfitting”. In: *IEEE transactions on neural networks* 19.6 (2008), pp. 983–995.

[236] S. Haykin. *Neural networks: a comprehensive foundation*. Prentice-Hall, Inc., 2007.

[237] R. M. Zur, Y. Jiang, L. L. Pesce, and K. Drukker. “Noise injection for training artificial neural networks: A comparison with weight decay and early stopping”. In: *Medical physics* 36.10 (2009), pp. 4810–4818.

[238] A. F. Murray and P. J. Edwards. “Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training”. In: *IEEE Transactions on neural networks* 5.5 (1994), pp. 792–802.

[239] J. Sum and K. Ho. “SNIWD: Simultaneous weight noise injection with weight decay for MLP training”. In: *International Conference on Neural Information Processing*. Springer. 2009, pp. 494–501.

[240] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens. "Adding gradient noise improves learning for very deep networks". In: *arXiv preprint arXiv:1511.06807* (2015).

[241] S. Ognawala and J. Bayer. "Regularizing Recurrent Networks-On Injected Noise and Norm-based Methods". In: *arXiv preprint arXiv:1410.5684* (2014).

[242] A. Krogh and J. A. Hertz. "A simple weight decay can improve generalization". In: *Advances in neural information processing systems*. 1992, pp. 950–957.

[243] H. Zhao, Y.-H. H. Tsai, R. R. Salakhutdinov, and G. J. Gordon. "Learning Neural Networks with Adaptive Regularization". In: *Advances in Neural Information Processing Systems*. 2019, pp. 11389–11400.

[244] J. Chorowski and J. M. Zurada. "Learning understandable neural networks with nonnegative weight constraints". In: *IEEE transactions on neural networks and learning systems* 26.1 (2014), pp. 62–69.

[245] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536.

[246] A. Y. Ng. "Feature selection, L 1 vs. L 2 regularization, and rotational invariance". In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 78.

[247] D. J. MacKay. "Bayesian interpolation". In: *Neural computation* 4.3 (1992), pp. 415–447.

[248] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Y. W. Teh and M. Titterington. Vol. 9. Proceedings of Machine Learning Research. 2010, pp. 249–256.

[249] T. J. Mitchell. "An algorithm for the construction of D-optimal experimental designs". In: *Technometrics* 42.1 (2000), pp. 48–54.

[250] G. Taguchi and Y. Yokoyama. *Taguchi methods: design of experiments*. Vol. 4. Amer Supplier Inst, 1993.

[251] M. D. Mckay, R. J. Beckman, and W. J. Conover. "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code". In: *Technometrics* 42.1 (2000), pp. 55–61.

[252] M. D. Shields and J. Zhang. "The generalization of Latin hypercube sampling". In: *Reliability Engineering & System Safety* 148 (2016), pp. 96–108.

[253] G. G. Wang. "Adaptive response surface method using inherited latin hypercube design points". In: *J. Mech. Des.* 125.2 (2003), pp. 210–220.

[254] R. Unal, R. Lepsch, and M. McMillin. "Response surface model building and multidisciplinary optimization using D-optimal designs". In: *7th AIAA Symposium on Multidisciplinary Analysis and Optimization*. 1998, p. 4759.

[255] G. G. Wang and S. Shan. "Review of Metamodeling Techniques in Support of Engineering Design Optimization". In: *Journal of Mechanical Design* 129.4 (2006), pp. 370–380.

[256] D. E. Bossert, S. L. Morris, W. F. Hallgren, and T. R. Yechout. *Introduction to aircraft flight mechanics: Performance, static stability, dynamic stability, and classical feedback control*. American Institute of Aeronautics and Astronautics, 2003.

[257] A. Chatterjee. "An introduction to the proper orthogonal decomposition". In: *Current science* (2000), pp. 808–817.

[258] J. Mayeur, A. Dumont, V. Gleize, and D. Destarac. "RANS simulations on TMR 3D test cases with the Onera elsA flow solver". In: *54th AIAA Aerospace Sciences Meeting*. 2016, p. 1357.

[259] L. R. Herrmann. "Laplacian-isoparametric grid generation scheme". In: *Journal of the Engineering Mechanics Division* 102.5 (1976), pp. 749–907.

[260] C. Harris. *NASA Supercritical Airfoils: A Matrix of Family-related Airfoils*. NASA technical paper. National Aeronautics, Space Administration, Office of Management, Scientific, and Technical Information Division, 1990.

[261] G. G. Slabaugh. "Computing Euler angles from a rotation matrix". In: *Retrieved on August* 6.2000 (1999), pp. 39–63.

[262] R. C. Hibbeler and S. Fan. *Statics and mechanics of materials*. Vol. 2. Prentice Hall Upper Saddle River, 2004.

[263] X. Liu, N. Qin, and H. Xia. "Fast dynamic grid deformation based on Delaunay graph mapping". In: *Journal of Computational Physics* 211.2 (2006), pp. 405–423.

[264] S. Komala-Sheshachala, R. Sevilla, and O. Hassan. "A coupled HDG-FV scheme for the simulation of transient inviscid compressible flows". In: *Computers & Fluids* 202 (2020), p. 104495.

[265] P. J. Sallis, W. Claster, and S. Hernández. "A machine-learning algorithm for wind gust prediction". In: *Computers & geosciences* 37.9 (2011), pp. 1337–1344.

[266] W. Hou, D. Darakananda, and J. D. Eldredge. “Machine-learning-based detection of aerodynamic disturbances using surface pressure measurements”. In: *AIAA Journal* 57.12 (2019), pp. 5079–5093.

[267] A. T. Nguyen, J.-H. Han, and A. T. Nguyen. “Application of artificial neural networks to predict dynamic responses of wing structures due to atmospheric turbulence”. In: *International Journal of Aeronautical and Space Sciences* 18.3 (2017), pp. 474–484.

[268] R Halder, M Damodaran, and B. Khoo. “Deep learning based reduced order model for airfoil-gust and aeroelastic interaction”. In: *AIAA Journal* 58.10 (2020), pp. 4304–4321.

[269] Z. Mao, A. D. Jagtap, and G. E. Karniadakis. “Physics-informed neural networks for high-speed flows”. In: *Computer Methods in Applied Mechanics and Engineering* 360 (2020), p. 112789.

[270] M. Raissi, P. Perdikaris, and G. E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[271] W. Li, X. Gao, and H. Liu. “Efficient prediction of transonic flutter boundaries for varying Mach number and angle of attack via LSTM network”. In: *Aerospace Science and Technology* 110 (2021), p. 106451.

[272] D. J. Toal, N. W. Bressloff, A. J. Keane, and C. M. Holden. “Geometric filtration using proper orthogonal decomposition for aerodynamic design optimization”. In: *AIAA journal* 48.5 (2010), pp. 916–928.

[273] W. Jing, L. Runze, H. Cheng, C. Haixin, R. CHENG, Z. Chen, and M. ZHANG. “An inverse design method for supercritical airfoil based on conditional generative models”. In: *Chinese Journal of Aeronautics* (2021).

[274] L. Piegl and W. Tiller. *The NURBS Book*. Springer Berlin Heidelberg, 1995.

[275] M. Selim and R. Koomullil. “Mesh deformation approaches–a survey”. In: *Journal of Physical Mathematics* 7.2 (2016).

[276] Y. Wang, N. Qin, and N. Zhao. “Delaunay graph and radial basis function for fast quality mesh deformation”. In: *Journal of Computational Physics* 294 (2015), pp. 149–172.

[277] X.-S. Yang and S. Deb. “Engineering optimisation by cuckoo search”. In: *International Journal of Mathematical Modelling and Numerical Optimisation* 1.4 (2010), pp. 330–343.

[278] X.-S. Yang. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[279] A. Graves. “Long short-term memory”. In: *Supervised sequence labelling with recurrent neural networks*. Springer, 2012, pp. 37–45.