THÈSE EN COTUTELLE PRÉSENTÉE
POUR OBTENIR LE GRADE DE

DOCTEUR DE

L'UNIVERSITÉ DE BORDEAUX

ET DU POLITECNICO DI TORINO

École doctorale de Mathématiques et d'informatique
École doctoral d'ingénierie aérospatiale

Mathématiques appliquées et calcul scientifique

Par Thomas PHILIBERT

Modèles basés sur les données pour les écoulements turbulents séparés

Data-driven models for turbulent separated flows

Sous la direction de : Angelo IOLLO
co-directeur : Andrea FERRERO

Soutenue le 20/12/2024

Membres du jury :

M. IOLLO, Angelo Professeur Université de Bordeaux Directeur de thèse
Mme. CINNELLA, Paola Professeure Université de Pierre et Marie Curie rapporteure
M. FRANCK, Emmanuel Chargé de recherche INRIA NANCY GRAND EST rapporteur
Mme. SALVETTI, Maria Vittoria Professeure Université de Pise Examinatrice
M. BERGMANN, Michel Directeur de recherche INRIA Bordeaux Examinateur
M. TADDEI, TOMMASO Chargé de recherche INRIA Bordeaux Examinateur

Membres invités :

M. FERRERO, Andrea Professeur Politecnico di Torino
M. LAROCCA, Francesco Professeur Politecnico di Torino

Institut de Mathématiques de Bordeaux UMR 5251
Université de Bordeaux
351, cours de la Libération - F-33405 TALENCE

Titre : Modèles basés sur les données pour les écoulements turbulents séparés

Résumé : Les modèles de Navier-Stokes moyennés de Reynolds (RANS) sont couramment utilisés dans les applications industrielles en raison de leur efficacité pour simuler des écoulements de fluides complexes, offrant un équilibre pratique entre précision et coût computationnel. Cependant, les modèles RANS présentent des limitations inhérentes pouvant affecter leur précision, en particulier pour les écoulements séparés ou hautement complexes. Un problème important réside dans les écarts observés dans les contraintes de Reynolds par rapport aux données de haute fidélité obtenues via la simulation numérique directe (DNS) ou des mesures expérimentales. Ces divergences peuvent entraîner des imprécisions dans les prédictions des caractéristiques d'écoulement, soulignant le besoin d'approches de modélisation améliorées pour renforcer la fiabilité des résultats des RANS.

Dans ce travail, nous proposons deux approches innovantes pour traiter ces écarts tout en conservant l'efficacité computationnelle du modèle de transport des contraintes de cisaillement de Menter (SST). La première approche implique un modèle algébrique explicite associé à un réseau de neurones (en particulier un perceptron multicouche, ou MLP) pour corriger le temps caractéristique turbulent dans le modèle RANS. La seconde approche vise l'approximation de Boussinesq elle-même, en utilisant soit un MLP soit un réseau antagoniste génératif (GAN) pour corriger directement cette approximation à partir de données guidées.

De plus, pour garantir des prédictions physiquement cohérentes, nous intégrons des contraintes de réalisabilité dans les deux modèles en introduisant des termes de pénalisation. Les conditions de réalisabilité sont essentielles pour les modèles de turbulence, car elles assurent que les tenseurs de contrainte prédits respectent les principes physiques fondamentaux, comme la positivité de l'énergie cinétique turbulente. L'intégration de ces contraintes améliore la stabilité et la fiabilité du modèle tant au cours de l'entraînement que de l'application.

Le modèle de contraintes de Reynolds proposé, complété par la correction du temps caractéristique (via le MLP) et la correction de l'approximation de Boussinesq (via MLP ou GAN), démontre de solides performances prédictives dans des configurations d'écoulement à la fois intra et extrapolées. Testé dans divers scénarios d'écoulement, le modèle capture avec précision les principales caractéristiques des écoulements turbulents tout en maintenant des prédictions physiquement réalistes. Ces résultats indiquent que nos approches améliorent l'adaptabilité et la fiabilité des modèles RANS, permettant des simulations plus précises d'écoulements complexes et pertinents sur le plan industriel, sans les coûts computationnels élevés associés à la DNS.

Mots clés : modélisation numérique, calcul haute performance, apprentissage automatique

Title : Data-driven models for turbulent separated flows

Abstract : Reynolds-averaged Navier-Stokes (RANS) models are commonly used in industrial applications due to their efficiency in simulating complex fluid flows, offering a practical balance between accuracy and computational cost. However, RANS models have inherent limitations that can affect their accuracy, particularly in cases involving separated or highly complex flows. One significant issue is the discrepancy in Reynolds stresses compared to high-fidelity data obtained from Direct Numerical Simulation (DNS) or experimental measurements. These disparities can lead to inaccuracies in flow characteristic predictions, underscoring the need for improved modeling approaches to enhance the reliability of RANS results.

In this work, we propose two innovative approaches aimed at addressing these discrepancies while retaining the computational efficiency of the Menter Shear Stress Transport (SST) model. The first approach involves an explicit algebraic model paired with a neural network (specifically a multilayer perceptron, or MLP) to correct the turbulent characteristic time within the RANS model. The second approach targets the Boussinesq approximation itself, using either an MLP or a Generative Adversarial Network (GAN) to directly correct this approximation based on data-driven insights.

Further, to ensure physically consistent predictions, we integrate realizability constraints within both models by introducing penalization terms. Realizability conditions are essential for turbulence models, as they ensure that the predicted stress tensors align with fundamental physical principles, such as the positivity of turbulent kinetic energy. Incorporating these constraints enhances model stability and reliability during both the training and application phases.

The proposed Reynolds stress model, augmented with characteristic time correction (via the MLP) and Boussinesq approximation correction (via MLP or GAN), demonstrates strong predictive performance in both in-sample and out-of-sample flow configurations. Tested across various flow scenarios, the model accurately captures key turbulent flow characteristics while maintaining physically realistic predictions. These findings indicate that our approaches enhance the adaptability and reliability of RANS models, enabling more accurate simulations of complex, industrially relevant flow conditions without the high computational costs associated with DNS.

Keywords : Numerical modeling, High-performance computing, Machine learning

Contents

1 Introduction **8**

1.1 Background and motivations 8

1.2 Objectives 10

2 Navier-Stokes equations **12**

2.1 Introduction 12

2.2 Historical Context 13

2.3 Continuity Equation (Mass Conservation) 14

2.4 Momentum Equation 16

2.5 Energy Equation 18

3 Reynolds-Averaged Navier-Stokes model **21**

3.1 Introduction 22

3.2 Historical Context 23

3.3 Prerequisites 24

3.3.1 Averaging operator 24

3.3.2 Decomposition of a random variable f 24

3.4 Reynold Averaged Navier Stokes equations 25

3.4.1 Derivation of RANS equations for compressible flows . . . 25
3.5 Boussinesq assumption . . . 27
3.5.1 Corrections to the Boussinesq assumption . . . 29
3.6 RANS SST k-ω model . . . 31
3.7 Limitation of RANS model . . . 33
3.7.1 Fundamental Theoretical Limitations . . . 33
3.7.2 Specific Flow Configuration Limitations . . . 33
3.7.3 Numerical and Implementation Limitations . . . 34

4 Introduction to deep learning **35**
4.1 Introduction . . . 36
4.2 Historical Context . . . 37
4.3 Artificial Neural Networks . . . 38
4.3.1 Artificial Neuron . . . 39
4.4 Multi-Layer Perceptron (MLP) . . . 39
4.4.1 Overview of the Structure . . . 40
4.4.2 Explanation of Feedforward Connections . . . 40
4.4.3 Activation Functions Used in MLPs . . . 41
4.4.4 Mathematical Representation . . . 41
4.4.5 Loss Function . . . 43
4.4.6 Backpropagation . . . 43
4.4.7 Training Process for MLPs . . . 45
4.4.8 Overfitting and Regularization in MLPs . . . 47
4.4.9 Hyperparameter Tuning for MLPs . . . 47
4.5 Generative Adversarial Network . . . 50
4.5.1 Introduction to Generative Adversarial Networks . . . 50
4.5.2 Structure of Generative Adversarial Networks . . . 50
4.5.3 Training Process of Generative Adversarial Networks . . . 52

5 Inputs features **55**
5.1 Guidelines on flow feature selection . . . 56
5.1.1 Galilean invariance . . . 56
5.1.2 Rotational invariance . . . 57
5.1.3 Non dimensionnal flow feature . . . 57
5.2 Explicit Algebraic Reynolds Stress Model . . . 58

6 Correction of the Boussinesq approximation by data-driven modelling 59
6.1 Introduction . . . 60
6.2 Dataset Description: Extended Flows Over Periodic Hills with Parameterized Geometries . . . 61
6.2.1 Introduction . . . 61
6.2.2 Dataset Objectives and Scope . . . 61
6.2.3 Reynolds Number and Bulk Velocity . . . 62
6.2.4 Flow Configurations and Numerical Methods . . . 62
6.2.5 Dataset Content . . . 62
6.2.6 Validation . . . 63
6.3 Realizability condition . . . 63
6.3.1 Diagonal Constraints . . . 63
6.3.2 Cauchy-Schwarz Inequality . . . 64
6.3.3 Eigenvalue Conditions . . . 64
6.3.4 Implications of Realizability Conditions in Turbulence Modeling . . . 64
6.3.5 Realizability Constraints in Neural Network-Assisted Turbulence Models . . . 65
6.4 Artificial neural network . . . 65
6.4.1 Multi layer perceptron . . . 68
6.4.2 Generative adversarial network . . . 78
6.4.3 Results . . . 80
6.5 Constant and linear coefficients . . . 82
6.5.1 Constant and linear correction . . . 83
6.5.2 Characteristic time correction . . . 89
6.5.3 RANS solver . . . 92
6.5.4 Results . . . 92

7 Conclusion 103
7.1 Results and interpretation . . . 103
7.2 Perspectives and future work . . . 104

CHAPTER 1

Introduction

Contents

1.1 Background and motivations 8
1.2 Objectives 10

1.1 Background and motivations

Turbulent separated flows are a common phenomenon in various engineering applications, such as aerospace, automotive, and energy sectors. These flows occur when a fluid, typically air or water, separates from the surface of a body, leading to the formation of complex vortical structures, increased drag, and unsteady aerodynamic forces. Accurate prediction of turbulent separation is critical for improving the design and performance of systems like aircraft wings, turbine blades, and ground vehicles. For instance, delayed separation and reduced drag can enhance the aerodynamic efficiency of aircraft, leading to lower fuel consumption and improved performance.

The accurate modeling of turbulent separated flows is essential not only for optimizing design but also for ensuring operational safety, especially in scenarios where flow-induced vibrations or unsteady forces can cause structural fatigue or failure. This makes turbulence modeling a key aspect of computational fluid dynamics (CFD) across numerous industries. Despite the

importance of predicting turbulent separation, it remains one of the most challenging aspects of fluid dynamics due to the highly complex, nonlinear interactions that characterize turbulent flows.

Traditionally, the simulation of turbulent flows has relied on the Navier-Stokes equations, particularly with the use of Reynolds-Averaged Navier-Stokes (RANS) models. These models incorporate turbulence closure techniques, such as the Boussinesq assumption, to approximate the effects of turbulence through simplified relationships between the mean flow and turbulent stresses. However, while RANS models are widely used due to their computational efficiency [56], they often struggle to accurately predict complex flow phenomena such as separation and reattachment[13] in turbulent boundary layers[54], especially in the presence of adverse pressure gradients [47] or unsteady conditions[57]. The Boussinesq assumption, in particular, introduces limitations by assuming that turbulent eddies behave like a Newtonian fluid with a linear relationship between the mean strain rate and the turbulent stresses, which does not hold in regions of flow separation.

To overcome the limitations of RANS models, more advanced techniques such as Large Eddy Simulation (LES)[22] and Direct Numerical Simulation (DNS) [37] have been developed. LES captures the larger scales of turbulence explicitly while modeling the smaller, subgrid scales, offering improved accuracy over RANS [43], particularly in separated flows. DNS, on the other hand, resolves all scales of turbulence directly, providing the most accurate representation of turbulent flows. However, both LES and DNS come with significantly higher computational costs [7]. DNS, in particular, is impractical for most engineering applications[37] due to the fine spatial and temporal resolution required to capture the intricate dynamics of turbulence, especially at high Reynolds numbers typical of industrial flows.

In recent years, data-driven approaches have emerged as a promising alternative for improving turbulence modeling [12], particularly in the prediction of separated flows. By leveraging machine learning techniques, specifically Artificial Neural Networks (ANNs) , data-driven models can learn from high-fidelity simulation data or experimental results to predict flow behavior more efficiently. These models offer the potential to bypass some of the inherent limitations of empirical closure models used in traditional RANS approaches by learning complex, nonlinear relationships directly from data [31]. This could significantly improve the accuracy of flow predictions in regions where traditional models, such as the Boussinesq approximation, fail to capture the true nature of turbulence.

The motivation for this research stems from the need to push the boundaries of current turbulence modeling methods and explore how data-driven techniques can address the limitations of traditional models. By correcting the Boussinesq approximation using machine learning, particularly with neural networks or GANs, this work seeks to enhance the predictive capabilities of RANS models while maintaining computational efficiency. The ability to accurately model turbulent separated flows has far-reaching implications for a variety of engineering fields, po-

tentially improving the design process, reducing reliance on costly experimental testing, and leading to more efficient and robust engineering systems.

Furthermore, the intersection of fluid mechanics and machine learning represents a novel and exciting direction for turbulence modeling research. This study aims to contribute to this emerging field by developing data-driven models that not only improve the accuracy of turbulence predictions but also demonstrate the practicality of applying advanced computational techniques to real-world fluid dynamics problems.

1.2 Objectives

The primary objective of this research is to develop and implement a data-driven model to improve the accuracy of turbulence predictions in separated flows, specifically by addressing the limitations of the Boussinesq approximation in traditional turbulence models. The Boussinesq assumption, commonly used in Reynolds-Averaged Navier-Stokes (RANS) models, simplifies the relationship between the mean flow and turbulent stresses. However, this approximation often fails to accurately capture complex flow phenomena, such as separation and reattachment, which are critical in many engineering applications. The resulting inaccuracies in RANS models can lead to suboptimal predictions, particularly in cases involving adverse pressure gradients or highly unsteady flows.

To address these challenges, this research aims to leverage machine learning techniques, specifically Artificial Neural Networks (ANNs) and Generative Adversarial Networks (GANs), to develop a data-driven correction for the Boussinesq approximation. By training these models on high-fidelity simulation, the goal is to learn the complex, non-linear interactions present in turbulent separated flows that are missed by traditional empirical models. This data-driven approach has the potential to significantly improve the accuracy of turbulence predictions, while maintaining the computational efficiency of RANS models, making it suitable for practical engineering applications.

The specific objectives of this research are as follows:

- Develop a neural network-based framework, using either Artificial Neural Networks (ANNs) or Generative Adversarial Networks (GANs), to correct the Boussinesq approximation in RANS models for turbulent separated flows. The framework will be trained to capture non-linear relationships between flow variables that are inadequately described by the traditional Boussinesq hypothesis.
- Investigate the performance of different machine learning architectures and training methodologies, focusing on their ability to generalize across a range of flow conditions, particularly in cases of separated flows where conventional turbulence models underperform.
- Validate the proposed models using benchmark cases of turbulent separated flows, such as flowon a periodic hill. The validation process will involve comparing the corrected RANS predictions against high-fidelity simulation methods (DNS)..

By achieving these objectives, this research aims to bridge the gap between traditional turbulence modeling and modern data-driven approaches, offering a more accurate and computationally efficient solution for predicting turbulent separated flows. In doing so, it seeks to advance the field of turbulence modeling by demonstrating the potential of machine learning to improve fundamental approximations such as the Boussinesq hypothesis. This research also aims to contribute to the broader effort of integrating data-driven techniques into engineering applications, ultimately enabling more accurate and reliable predictions in a variety of fluid dynamics problems.

CHAPTER 2

Navier-Stokes equations

Contents

2.1 Introduction **12**
2.2 Historical Context **13**
2.3 Continuity Equation (Mass Conservation) **14**
2.4 Momentum Equation **16**
2.5 Energy Equation **18**

2.1 Introduction

In this chapter, we will explore the history, derivation, and fundamental principles behind the Navier-Stokes equations, which form the cornerstone of fluid dynamics. These equations describe the motion of fluid substances, such as liquids and gases, by representing the balance of forces acting at any point within the fluid. First formulated in the 19th century by Claude-Louis Navier and George Gabriel Stokes, the Navier-Stokes equations have since become an essential tool in understanding a wide range of fluid flow phenomena, from the flow of air over an aircraft wing to the movement of water through pipes.

We begin by tracing the historical development of the Navier-Stokes equations, highlighting key contributions from early fluid mechanics researchers and mathematicians. This historical

perspective provides insight into the evolution of our understanding of fluid motion and the importance of these equations in the broader context of science and engineering.

Next, we introduce the general form of the Navier-Stokes equations, which govern the conservation of mass, momentum, and energy in a fluid. This includes a discussion of the underlying assumptions, such as the continuum hypothesis and the concept of a Newtonian fluid, which are essential for their formulation. We will also introduce the fundamental notations and coordinate systems that will be used throughout this thesis, ensuring consistency in the mathematical expressions of the flow quantities and governing equations.

The Navier-Stokes equations, in their full form, are highly non-linear partial differential equations and are difficult to solve analytically for most real-world applications. Therefore, much of the work in fluid dynamics involves simplifying these equations through various approximations or employing numerical techniques to obtain practical solutions. In particular, we will explore how these equations form the basis for turbulence models such as the Reynolds-Averaged Navier-Stokes (RANS) equations, which will be the focus of subsequent chapters in this thesis.

By establishing a solid understanding of the Navier-Stokes equations, this chapter will provide the foundational framework for the discussions of turbulence modeling and data-driven approaches that follow. The concepts and notations introduced here will serve as a reference point for the more advanced topics covered later, ensuring clarity and consistency in the treatment of fluid flow phenomena throughout this work.

2.2 Historical Context

In fluid mechanics, the Navier-Stokes equations are non-linear partial differential equations that describe the evolution of the motion of fluids over time, including both Newtonian and non-Newtonian fluids. These equations are fundamental in understanding the dynamics of fluid flow, making them essential in a variety of fields, from engineering to meteorology.

The equations are named in honor of two prominent 19th-century scientists who made significant contributions to their development. Henry Navier, a French mathematician and civil engineer, was the first to introduce a fundamental concept into Euler's equations: viscosity. In 1752, D'Alembert noted a paradox indicating that an object immersed in a fluid should move without experiencing friction according to Euler's theory, which contradicted experimental observations at the time. This discrepancy highlighted the inadequacy of the Eulerian framework for describing real fluid behavior. To address this issue, in 1820, Navier added the notion of friction to the equations by introducing a new term called viscosity, which quantifies the internal resistance of a fluid to flow and deformation.

Subsequently, George Gabriel Stokes, a British mathematician and physicist, refined and completed the formulation of the equations. In 1845, he provided the final form of the equation of

conservation of momentum, further elucidating the relationship between viscous effects and fluid motion. Stokes' work was instrumental in establishing a solid theoretical foundation for the Navier-Stokes equations, enabling subsequent researchers to explore their implications in various fluid flow scenarios.

Many other scientists have contributed to the advancement of fluid dynamics, including Augustin Louis Cauchy, who made critical strides in the mathematical formulation of continuum mechanics; Siméon Denis Poisson, who explored the behavior of viscous fluids; and Adhémar Barré de Saint-Venant, who focused on the dynamics of fluid flow in channels. Each of these contributions has enriched the understanding of fluid behavior, allowing for the application of the Navier-Stokes equations to increasingly complex systems.

Despite their significance, the solutions to the Navier-Stokes equations remain elusive, and questions regarding the existence and uniqueness of such solutions have not been definitively resolved. These challenges are part of a Millennium Prize Problem posed by the Clay Mathematics Institute, which offers a substantial reward for anyone who can prove the existence and smoothness of solutions in three-dimensional space. The ongoing research in this area underscores the complexity and importance of the Navier-Stokes equations in mathematical physics.

Thanks to approximate solutions and numerical methods, the Navier-Stokes equations enable us to model a myriad of phenomena, such as ocean currents, blood circulation in the human body, the behavior of wind turbines under varying wind conditions, and even weather patterns. The ability to accurately solve these equations is therefore of paramount importance in both industrial applications and scientific research, influencing fields as diverse as aerodynamics, environmental science, and biomedical engineering.

In the following sections, we will thoroughly examine each constitutive equation of the Navier-Stokes model, providing a detailed analysis of the assumptions, derivations, and implications of these equations. This comprehensive exploration will enable us to introduce the necessary notations and provide a clear explanation of each term, laying the groundwork for understanding their application in the subsequent chapters of this thesis.

2.3 Continuity Equation (Mass Conservation)

The continuity equation expresses the principle of conservation of mass in fluid dynamics. It ensures that the mass of fluid is conserved as it flows through a given volume, meaning that mass cannot be created or destroyed within the flow. This fundamental principle is essential for understanding fluid behavior and is applicable to both incompressible and compressible fluids. By introducing and analyzing the continuity equation, we will establish the necessary framework for understanding fluid motion within the broader context of the Navier-Stokes equations.

The general form of the continuity equation can be expressed as:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{2.1}$$

where ρ is the fluid density and $\mathbf{u}$ is the velocity field. This equation highlights that the rate of change of mass within a control volume must equal the net mass flux across the boundaries of that volume.

Local Rate of Change of Density

The first term of the continuity equation is given by:

$$\frac{\partial \rho}{\partial t} \tag{2.2}$$

- ρ: Fluid density, representing the mass per unit volume of the fluid.
- $\frac{\partial \rho}{\partial t}$: Local rate of change of density with respect to time at a specific point in space.

This term represents the rate at which the density of the fluid changes at a fixed point in space over time. It is particularly important in analyzing compressible flows, where density can vary significantly. For incompressible fluids, where density is considered constant, this term becomes negligible, simplifying the analysis.

Convective Transport of Mass

The second term of the continuity equation is represented as:

$$\nabla \cdot (\rho \mathbf{u}) \tag{2.3}$$

- ρ: Fluid density, which may vary in space and time, particularly in compressible flow scenarios.
- $\mathbf{u}$: Velocity field, indicating the velocity vector of the fluid at each point.
- $\nabla \cdot (\rho \mathbf{u})$: Divergence of the mass flux, calculated as the dot product of the gradient operator with the mass flux vector ($\rho \mathbf{u}$).

This term represents the transport of mass due to the motion of the fluid. Specifically, it accounts for the mass entering and leaving a differential volume element, indicating how the velocity field affects the mass distribution within the fluid. If the divergence is positive, it indicates a net outflow of mass from the control volume, whereas a negative divergence signifies a net inflow.

The continuity equation ensures the conservation of mass in the fluid. It states that any change in density within a given volume must be balanced by the net flux of mass into or out of that volume. This principle is crucial for modeling fluid dynamics as it establishes the relationship

between fluid motion and density changes, forming the basis for analyzing various flow scenarios.

In summary, the continuity equation is a vital component of fluid dynamics that encapsulates the fundamental principle of mass conservation. Its formulation allows engineers and scientists to analyze fluid behavior in various contexts, from simple pipe flow to complex turbulent flows in natural systems. In the following sections, we will discuss how the continuity equation integrates with the Navier-Stokes equations, providing a comprehensive framework for fluid dynamics analysis.

2.4 Momentum Equation

The momentum equation describes the conservation of momentum for fluid flow, accounting for the various forces acting on the fluid, including pressure, viscous stresses, and external forces. This equation provides a comprehensive framework for analyzing how fluid motion evolves over time under various conditions. By applying Newton's second law to a control volume, the momentum equation links the dynamics of fluid flow to the forces influencing that flow.

The general form of the momentum equation can be expressed as:

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \left(\rho \mathbf{u} \cdot \nabla \mathbf{u}\right) = -\nabla p + \nabla \cdot \tau + \rho \mathbf{f} \tag{2.4}$$

where ρ is the fluid density, $\mathbf{u}$ is the velocity field, p is the pressure, τ is the stress tensor, and $\mathbf{f}$ represents external body forces.

Inertial Terms

The left side of the momentum equation contains the inertial terms, given by:

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \left(\rho \mathbf{u} \cdot \nabla \mathbf{u}\right) \tag{2.5}$$

- $\frac{\partial \rho \mathbf{u}}{\partial t}$: Local acceleration, representing the rate of change of velocity with respect to time at a fixed point in space.
- $\left(\rho \mathbf{u} \cdot \nabla \mathbf{u}\right)$: Convective acceleration, which captures the change of velocity due to the motion of the fluid itself.

These terms collectively represent the total acceleration of a fluid particle, combining both local and convective effects. The local acceleration accounts for changes in velocity over time, while

the convective acceleration reflects how the fluid's motion influences itself, such as when fluid flows over surfaces or through varying cross-sections.

Pressure Gradient Force

The next term in the momentum equation is:

$$-\nabla p \tag{2.6}$$

- ∇p: Gradient of the pressure field, which indicates how pressure varies in space.

This term represents the force per unit volume exerted by pressure variations within the fluid. It drives the fluid from regions of high pressure to low pressure, playing a crucial role in initiating flow and influencing fluid motion. Understanding the pressure gradient force is essential for predicting how fluids respond to changes in their environment.

Viscous Stress Tensor

The term associated with viscous effects is represented as:

$$\nabla \cdot \tau \tag{2.7}$$

- τ: Viscous stress tensor, which describes internal forces due to viscosity.
- $\nabla \cdot \tau$: Divergence of the viscous stress tensor, indicating how momentum is diffused throughout the fluid due to viscosity.

This term accounts for the internal friction within the fluid due to its viscosity, representing the diffusion of momentum as a result of viscous effects. The viscous stress tensor τ for a Newtonian fluid is given by:

$$\tau = \mu\left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T\right) + \lambda(\nabla \cdot \mathbf{u})\mathbf{I} \tag{2.8}$$

- μ: Dynamic viscosity, which quantifies the resistance of the fluid to shear stress.
- $\nabla \mathbf{u}$: Velocity gradient tensor, representing how velocity changes with respect to position in the fluid.
- $(\nabla \mathbf{u})^T$: Transpose of the velocity gradient tensor, ensuring the tensor remains symmetric.
- λ: Second viscosity coefficient, which accounts for volumetric effects, often given by $\lambda = -\frac{2}{3}\mu$ in Stokes' hypothesis.
- $\mathbf{I}$: Identity matrix, providing a scalar contribution to the stress tensor.

The term $\mu\left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T\right)$ accounts for shear viscosity, while the term $\lambda(\nabla \cdot \mathbf{u})\mathbf{I}$ accounts for volumetric viscosity. The inclusion of the viscous stress tensor is essential for accurately capturing the behavior of real fluids, especially under conditions where shear and normal stresses play significant roles.

Body Force

The final term in the momentum equation is represented as:

$$\rho \mathbf{f} \tag{2.9}$$

- **f**: External body force per unit volume acting on the fluid, such as gravitational, electromagnetic, or other forces that act throughout the volume of the fluid.

This term represents external forces that influence the fluid's motion. Understanding the role of body forces is crucial for accurately modeling fluid dynamics, as these forces can significantly affect the flow characteristics and behavior of the fluid.

In summary, the momentum equation is a fundamental component of fluid dynamics that links the forces acting on a fluid to its motion. By integrating the concepts of inertial forces, pressure gradients, viscous stresses, and body forces, this equation provides a comprehensive framework for analyzing fluid flow under a variety of conditions. In the subsequent sections, we will explore specific applications of the momentum equation and its implications in the context of turbulent flows and other complex fluid phenomena.

2.5 Energy Equation

In this section, we will explore the energy equation, an essential component of the Navier-Stokes equations. The energy equation represents the conservation of energy in a fluid system, accounting for various forms of energy, such as kinetic, potential, and internal energy. It also includes the effects of heat transfer and work done by viscous forces, providing a comprehensive framework for analyzing energy changes within a fluid.

The general form of the energy equation can be expressed as:

$$\frac{\partial \rho e}{\partial t} + \left(\rho \mathbf{u} \cdot \nabla e\right) = -\nabla \cdot (p\mathbf{u}) + \nabla \cdot (k \nabla T) + \Phi \tag{2.10}$$

where ρ is the fluid density, e is the specific internal energy, p is the pressure, k is the thermal conductivity, T is the temperature, and Φ represents the viscous dissipation function.

Inertial Terms

The left side of the energy equation contains the inertial terms, expressed as:

$$\frac{\partial \rho e}{\partial t} + \left(\rho \mathbf{u} \cdot \nabla e\right) \tag{2.11}$$

- $\frac{\partial \rho e}{\partial t}$: Local rate of change of specific internal energy with respect to time at a fixed point in space.
- $\left(\rho \mathbf{u} \cdot \nabla e\right)$: Convective transport of specific internal energy, representing the change of internal energy due to the movement of the fluid.

These terms represent the total rate of change of the specific internal energy of a fluid particle, combining both local and convective effects. The local term reflects how the internal energy of the fluid changes over time, while the convective term accounts for how the movement of the fluid carries internal energy with it, influencing the energy distribution throughout the fluid.

Work Done by Pressure Forces

The next term in the energy equation is:

$$-\nabla \cdot (p\mathbf{u}) \tag{2.12}$$

- p: Pressure field, indicating the force exerted by the fluid per unit area.
- $\nabla \cdot \mathbf{u}$: Divergence of the velocity field, which reflects the expansion or contraction of the fluid.

This term represents the work done by pressure forces during compression or expansion of the fluid. When the fluid is compressed (i.e., when the divergence is negative), work is done on the fluid, increasing its internal energy. Conversely, during expansion (i.e., when the divergence is positive), work is done by the fluid, potentially decreasing its internal energy.

Thermal Conduction

The term associated with heat transfer due to conduction is represented as:

$$\nabla \cdot (k \nabla T) \tag{2.13}$$

- k: Thermal conductivity, a measure of a material's ability to conduct heat.
- ∇T: Gradient of the temperature field, indicating how temperature varies in space.
- $\nabla \cdot (k \nabla T)$: Divergence of the heat flux due to thermal conduction, representing how heat energy is transported through the fluid.

This term represents the heat transfer due to thermal conduction within the fluid. The heat flux flows from regions of higher temperature to regions of lower temperature, and this term quantifies the rate of heat transfer as a result of temperature gradients. Understanding thermal conduction is crucial for analyzing heat transfer processes in fluid systems, especially in applications involving thermal management.

Viscous Dissipation

The final term in the energy equation is represented as:

$$\Phi = (\tau \cdot \nabla)\mathbf{u} \tag{2.14}$$

- Φ: Viscous dissipation function, quantifying the energy conversion due to viscous effects. This term represents the conversion of kinetic energy into internal energy due to viscous effects. As fluid layers move relative to each other, the viscosity generates heat through friction, which dissipates energy. This dissipation is particularly significant in high-viscosity fluids or in flows with high shear rates, where energy losses can affect the overall efficiency of fluid systems.

In summary, the energy equation is a fundamental component of the Navier-Stokes equations that links the conservation of energy to the various processes occurring within a fluid. By integrating the concepts of inertial effects, pressure work, thermal conduction, and viscous dissipation, this equation provides a comprehensive framework for analyzing energy changes in fluid flows. In the subsequent sections, we will further explore the implications of the energy equation in the context of turbulent flows and heat transfer phenomena, highlighting its relevance in practical engineering applications.

CHAPTER 3

Reynolds-Averaged Navier-Stokes model

Contents

3.1 Introduction 22

3.2 Historical Context 23

3.3 Prerequisites 24

3.3.1 Averaging operator 24

3.3.2 Decomposition of a random variable f 24

3.4 Reynold Averaged Navier Stokes equations 25

3.4.1 Derivation of RANS equations for compressible flows 25

3.5 Boussinesq assumption 27

3.5.1 Corrections to the Boussinesq assumption 29

3.6 RANS SST k-ω model 31

3.7 Limitation of RANS model 33

3.7.1 Fundamental Theoretical Limitations 33

3.7.2 Specific Flow Configuration Limitations 33

3.7.3 Numerical and Implementation Limitations 34

3.1 Introduction

Turbulent flow is a complex and multifaceted phenomenon in fluid dynamics, characterized by chaotic changes in pressure and flow velocity. This unpredictability manifests as fluctuations and vortices, which can significantly affect the performance and stability of various systems. Accurately predicting and analyzing turbulent flows are crucial for a wide range of engineering applications, including aerospace, automotive, chemical processing, and environmental engineering. The ability to model these flows not only enhances the efficiency and safety of designs but also contributes to the development of more sustainable practices in engineering.

The Reynolds-Averaged Navier-Stokes (RANS) model provides a practical and widely used approach to addressing the challenges associated with turbulence. By averaging the Navier-Stokes equations over time, the RANS model transforms the governing equations of fluid motion into a more manageable form. This averaging process results in a set of equations that account for the effects of turbulence through the introduction of Reynolds stresses, which represent the turbulent fluctuations in momentum.

This chapter aims to introduce the RANS model comprehensively, starting with a discussion of the Reynolds-Averaged Navier-Stokes equations. We will delve into the mathematical formulation of these equations, highlighting their significance in the context of turbulence modeling. Following this, we will explore the Boussinesq assumption, which relates the Reynolds stresses to the mean rate of strain, and is essential for closing the RANS equations.

While the RANS model has proven invaluable in practical applications, it is not without limitations. We will address these limitations, including issues related to closure problems and the model's inability to capture certain turbulent flow characteristics accurately. Understanding these shortcomings is critical for the ongoing development of improved modeling techniques.

In addition, this chapter will review various correction models designed to enhance the accuracy of the RANS approach. These models aim to mitigate the limitations of the traditional RANS formulation by incorporating additional physics or employing advanced techniques, such as machine learning and turbulence closures.

To establish a solid foundation for the upcoming sections, we will derive the RANS equations by examining their origins and defining the relevant constants and variables. This derivation will include the introduction of key tools such as averaging operators, including the Reynolds and Favre averaging methods. Subsequently, we will discuss the compressible formulation of the RANS equations, which is essential for applications involving compressible flows.

By the end of this chapter, the reader will have a comprehensive understanding of the RANS model, its equations, and its applications, setting the stage for a deeper exploration of turbulence modeling in the following sections.

3.2 Historical Context

The Reynolds-Averaged Navier-Stokes (RANS) model is a significant milestone in the study of turbulent flows within fluid dynamics. Its development is closely linked to the evolution of turbulence theory and the need for effective modeling techniques. The foundation for the RANS model was established in the 19th century with the formulation of the Navier-Stokes equations by Claude-Louis Navier in 1822 and George Gabriel Stokes in 1845, which describe the motion of viscous fluids. However, the complexity of turbulent flows posed a challenge that these equations could not adequately address in their original form.

Osborne Reynolds made pivotal contributions in the late 19th century, particularly with his experiments in 1883 that defined the Reynolds number, a dimensionless quantity characterizing flow regimes. This work revealed the transition from laminar to turbulent flow, underscoring the chaotic nature of turbulence and the necessity for an averaging approach in its study. In 1895, Reynolds introduced the concept of averaging in fluid dynamics, which became the basis for the RANS model. By applying time-averaging to the Navier-Stokes equations, Reynolds derived a set of equations that accounted for the effects of turbulence through the introduction of Reynolds stresses, transforming the complex, unsteady turbulent equations into a more tractable form.

The RANS model gained prominence in the 1970s, driven by advancements in computational fluid dynamics (CFD) and numerical methods. The ability to solve the RANS equations for complex geometries and flow conditions made it an essential tool for engineers and researchers. Its practicality and effectiveness in predicting turbulent flows in various applications, such as aerospace and automotive engineering, solidified its status as a standard approach in turbulence modeling.

Throughout the latter half of the 20th century, the RANS model underwent significant refinement with the introduction of various turbulence closure models, such as the k-epsilon and k-omega models. These developments aimed to enhance the predictive capabilities of the RANS approach, addressing some of the inherent limitations associated with the model. In the 1990s and early 2000s, advancements in numerical methods and increased computational power further expanded the applicability of the RANS model in real-world scenarios, leading to the emergence of hybrid models that combined RANS with large eddy simulations (LES).

Today, the RANS model remains widely used in fluid dynamics, utilized across a broad range of engineering applications. Ongoing research is focused on improving the accuracy of RANS predictions through innovative techniques, such as machine learning and data-driven approaches, to address the model's limitations and enhance its capability to predict complex turbulent flows.

3.3 Prerequisites

3.3.1 Averaging operator

To obtain the averaged equations, it is essential to define an averaging operator. There are three different types of such operators: ensemble average, time average, and spatial average:

- The ensemble average $\overline{f}$ of a random function is defined as the statistical average over the set of independent realizations f^i. So we get:

$$\overline{f} = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} f^i$$

- The time average $\underline{f^i}(t)$ of a random function $f(t)$ is defined as the average over a time T of a particular realization $f^i(t)$. Thus, we have:

$$\underline{f^i}(t) = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} f^i(t)\,dt$$

- The spatial average $\underline{\underline{f^i}}(x)$ of a random function $f(x)$ is defined as the average over a domain Ω of a particular realization $f^i(x)$:

$$\underline{\underline{f^i}}(x) = \lim_{\Omega\to\infty} \frac{1}{|\Omega|} \int_{\Omega} f^i(x)\,dV$$

In the case of a steady and homogeneous flow, we have the following equality:

$$\overline{f}(x,t) = \underline{f^i}(x,t) = \underline{\underline{f^i}}(x,t)$$

This equality comes from the notion of ergodicity. We will assume that we are always in a situation where this notion applies.

3.3.2 Decomposition of a random variable f

Let f be a random variable. By applying the ensemble averaging operator, this function decomposes into a mean part and a fluctuating part:

$$f = \overline{f} + f' \tag{3.1}$$

where $\overline{f}$ is the mean part, and f' is the fluctuating part.

Furthermore, by definition, we have :

$$\overline{f'} = 0$$

In the case of compressible flows, a mass-weighted decomposition, first proposed by Favre[15], is preferred as it simplifies the conservation equations. In this case, we have:

$$f = \tilde{f} + f' \tag{3.2}$$

with :

$$\begin{cases} \tilde{f} = \dfrac{\overline{\rho f}}{\overline{\rho}} \\ \tilde{f'} = 0 \end{cases}$$

The averaging operator satisfies several relations, known as Reynolds relations. Let f and g be two random functions and λ a real number:

$$\overline{f+g} = \overline{f} + \overline{g} \tag{3.3}$$
$$\overline{\lambda f} = \lambda \overline{f} \tag{3.4}$$
$$\overline{fg} = \overline{f}\,\overline{g} + \overline{f'g'} \tag{3.5}$$
$$\overline{\frac{\partial f}{\partial t}} = \frac{\partial \overline{f}}{\partial t} \tag{3.6}$$
$$\overline{\frac{\partial f}{\partial x}} = \frac{\partial \overline{f}}{\partial x} \tag{3.7}$$
$$\overline{\int f\,dt} = \int \overline{f}\,dt \tag{3.8}$$
$$\overline{\int f\,dx} = \int \overline{f}\,dx \tag{3.9}$$
$$\tag{3.10}$$

3.4 Reynold Averaged Navier Stokes equations

To derive the equations used in RANS models, we will apply the averaging operators and their properties to the Navier-Stokes equations. This approach will be applied to compressible flows. For incompressible flows, the method remains the same, with the only difference being that the density is treated as constant.

3.4.1 Derivation of RANS equations for compressible flows

In order to obtain the RANS equations for compressible flows, we will use the Reynolds averaging operator along with the following relation:

$$\overline{\rho f} = \overline{\rho}\,\tilde{f} \tag{3.11}$$

Continuity equation

The continuity equation of the Navier-Stokes equation is :

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{3.12}$$

By applying the averaging operator, we obtain :

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) &= \frac{\partial}{\partial t}(\overline{\rho} + \rho') + \nabla \cdot ((\overline{\rho} + \rho')(\overline{u} + u')) \\ &= \frac{\partial \overline{\rho}}{\partial t} + \frac{\partial \rho'}{\partial t} + \nabla \cdot (\bar{\rho}\,\bar{u} + \overline{\rho} u' + \rho' \bar{u} + \bar{\rho'}\bar{u'}) \end{aligned}$$

By time averaging the previous equation and using the Reynolds averaging rules, we arrive to the Reynolds-averaged continuity equation for compressible flows,

$$\frac{\partial \overline{\rho}}{\partial t} + \nabla \cdot (\bar{\rho}\,\bar{u} + \overline{\rho' u'}) = 0 \tag{3.13}$$

If we expand the left hand side of equation 3.11 (that is, we substitute the Reynolds decomposition and use the Reynolds averaging rules), we obtain,

$$\overline{\rho}\,\tilde{u} = \bar{\rho}\,\bar{u} + \overline{\rho' u'}$$

Then, we can substitute in the previous equation :

$$\frac{\partial \overline{\rho}}{\partial t} + \nabla \cdot \overline{\rho}\,\tilde{u} = 0 \tag{3.14}$$

It is observed that the averaged mass conservation equation retains the same structure as the original equation.

Momentum equation

The momentum equation of the Navier-Stokes equation is :

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \left(\rho \mathbf{u} \cdot \nabla \mathbf{u}\right) = -\nabla p + \nabla \cdot \tau + \rho \mathbf{f}$$

By applying the average operator on the left hand side of this equation and then time averaging:

$$\begin{aligned}\frac{\partial \rho \mathbf{u}}{\partial t} &= \frac{\partial}{\partial t}(\overline{\rho}+\rho')(\overline{u}+u') \\ &= \frac{\partial}{\partial t}(\bar{\rho}\bar{u}+\bar{\rho}u'+\rho'\bar{u}+\bar{\rho'}\bar{u'}) \\ &= \frac{\partial \bar{\rho}\tilde{u}}{\partial t}\end{aligned}$$

$$\begin{aligned}\left(\rho \mathbf{u}\cdot\nabla\mathbf{u}\right) &= (\overline{\rho}+\rho')(\overline{u}+u')\cdot\nabla(\bar{u}+u') \\ &= (\bar{\rho}\bar{u}+\bar{\rho}u'+\rho'\bar{u}+\rho'u')\cdot\nabla(\bar{u}+u') \\ &= \bar{\rho}\tilde{u}\cdot\nabla\tilde{u}\end{aligned}$$

We proceed similarly with the right-hand side, leading to:

$$\frac{\partial \bar{\rho}\tilde{u}}{\partial t}+\bar{\rho}\tilde{u}\cdot\nabla\tilde{u} = -\nabla\overline{p}+\nabla\cdot\bar{\tau}+\bar{\mathbf{f}} \tag{3.15}$$

With : $\bar{\tau} = -\overline{\rho}\widetilde{u'u'}$ This variable is the turbulent stress tensor known as the Reynolds stress tensor.

In the RANS approach, the turbulent stress term can be addressed in different ways. An option is to solve the transport equations for each component of the stress tensor τ. This introduces third-order terms, which require additional closure assumptions, leading to what are known as second-order models. To avoid solving the transport equations for the turbulent stress tensor components, one can directly model the term τ using an assumption. In this case, the models are referred to as zero-order models. The most commonly used closure assumption is the Boussinesq hypothesis.

3.5 Boussinesq assumption

The Boussinesq assumption is a widely used approximation in turbulence modeling, particularly within the context of RANS models. It postulates that the turbulent stresses, which arise due to velocity fluctuations, can be modeled in a manner analogous to molecular viscosity. Specifically, it assumes a linear relationship between the turbulent stresses and the mean rate of strain, similar to how viscous stresses are related to velocity gradients in laminar flow. This allows the turbulent stress tensor to be expressed as a function of the viscosity of the eddy, a property that characterizes the intensity of the turbulence. The Boussinesq assumption greatly simplifies the closure problem in turbulence modeling by reducing the complexity of the turbulent stress terms. However, while it works well for many practical flows, it has limitations,

particularly in cases with strong anisotropy, non-equilibrium turbulence, or complex flow geometries, where the assumption of isotropic turbulence may no longer hold. Despite these limitations, the Boussinesq hypothesis remains a cornerstone of many zero-equation models, due to its simplicity and effectiveness in a wide range of engineering applications.

By analogy with the viscous stress tensor for a Newtonian fluid, the Reynolds stress tensor is assumed to be expressed as follows:

Incompressible flows :

$$\tau_{ij} = \rho\overline{u_i' v_j'} = \mu_t S_{ij} - \frac{2}{3}k\delta_{ij} \tag{3.16}$$

With :

- μ_t : turbulent viscosity
- $S_{ij} = \frac{1}{2}\left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right)$: strain rate tensor
- δ_{ij} : the kronecker delta
- $k = \frac{1}{2}\overline{u_i u_i}$: turbulent kinetic energy
- $\tau = \frac{k}{\epsilon}$: caracteristic time

Compressible flows :

$$\tau_{ij} = \rho\widetilde{u_i' v_j'} = \mu_t\left(S_{ij} - \frac{2}{3}\frac{\partial \tilde{u}_n}{\partial x_n}\delta_{ij}\right) - \frac{2}{3}\overline{\rho}k\delta_{ij} \tag{3.17}$$

With :

- $S_{ij} = \frac{1}{2}\left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i}\right)$: strain rate tensor
- $k = \frac{1}{2}\widetilde{u_i u_i}$: turbulent kinetic energy

The Boussinesq assumption simplifies the turbulent stress tensor by replacing its six unknowns with a single scalar: the turbulent viscosity μ_t. This assumption is based on several key principles:

- The turbulence is isotropic at small scales
- The ratio of production P_k to ϵ is close to 1
- The Reynolds stress tensor is proportional to the strain rate tensor
- The turbulent flow immediately responds to changes in the mean flow without memory effects
- A diffusive nature (suited to small scales) is assigned to a large-scale phenomenon, arising from the non-linearity of the Navier-Stokes equations. As a result, the Reynolds stresses tend to stabilize unsteady advective mechanisms, which somewhat conflicts with their theoretical origin.

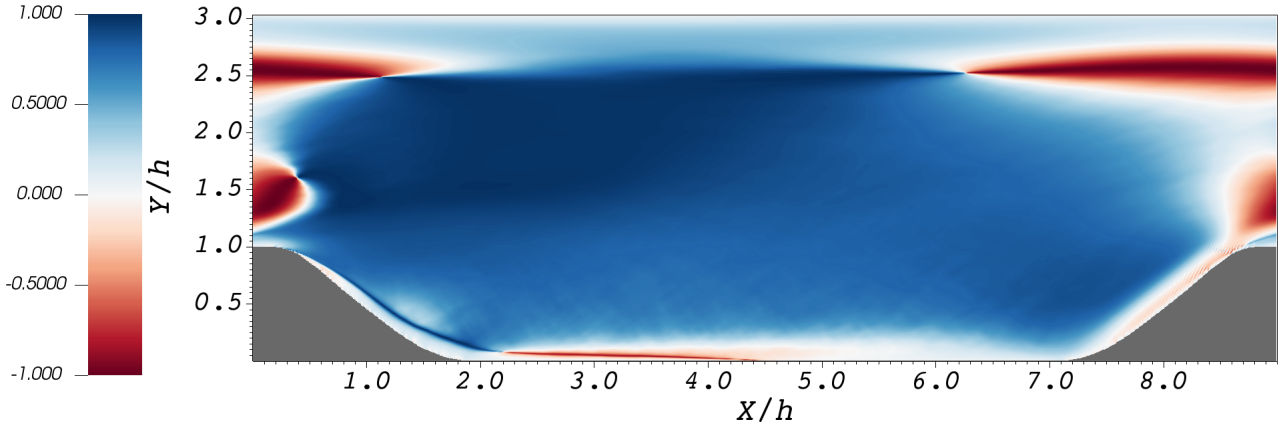


Figure 3.1: Alignment of anisotropic part of Reynolds stress tensor with the strain rate tensor using DNS data of a flow over a periodic hill at $Re = 5600$ [68]

3.5.1 Corrections to the Boussinesq assumption

Despite these key principles, the Boussinesq assumption is an approximation of the Reynold stress tensor. It is therefore possible to assess the quality of this approximation by using data from high-fidelity models such as DNS and by applying the following formula:

$$\gamma = \frac{a_{ij} S^*_{ij}}{\sqrt{a_{mn} a_{nm} S^*_{pq} S^*_{qp}}}, \qquad -1 \leq \gamma \leq 1 \tag{3.18}$$

Where :

$$S^*_{ij} = \tau^{DNS} S = \frac{\tau^{DNS}}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \tag{3.19}$$

$$a_{ij} = \frac{\tau^{DNS}_{ij}}{\rho^{DNS} k^{DNS}} + \frac{2}{3} \delta_{ij} \tag{3.20}$$

This expression is simply a normalized dot product between the strain rate tensor and the anisotropic part of the Reynolds stress tensor. It thus represents the angle between these two quantities. If the Reynolds stress were, as shown by the Boussinesq approximation, linearly dependent on the strain rate tensor, then this quantity would be equal to 1 everywhere. However, if we apply this formula to a flow over a periodic hill 3.1 (that will be described later), we notice that this is not the case everywhere.

To address this issue, much research has been conducted. Initially, in 1975, Pope[44] introduced a nonlinear correction to account for the difference between the Reynolds stress tensor obtained from DNS and the one obtained through this approximation. This correction improves

the linear Boussinesq approximation by adding additional terms, creating a nonlinear constitutive stress-strain relationship. This model was based on the weak equilibrium hypothesis for the anisotropic tensor a_{ij}. The correction is expressed by means of a tensor basis V^i and scalar invariants I_k.

$$\tau_{ij} = 2\mu_t S_{ij} - \frac{2}{3}\rho k \delta_{ij} - \rho k \sum_{k=1}^{10} \alpha_k V_{ij}^k \tag{3.21}$$

Where V_k are basis functions formed from polynomials of non-dimensional strain rate $S_{ij}^* = \tau S_{ij}$ and rotation rate $R_{ij}^* = \tau R_{ij}$, where S_{ij} is the previously defined strain rate tensor and R_{ij} is the rotation rate tensor:

$$R_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right) \tag{3.22}$$

$V^1 = S^*$	$V^6 = R^{*2}S + S^*R^{*2} - \frac{2}{3}I\mathrm{Tr}(S^*R^{*2})$
$V^2 = S^*R^* - R^*S^*$	$V^7 = R^*S^*R^{*2} - R^{*2}S^*R^*$
$V^3 = S^{*2} - \frac{1}{3}I\mathrm{Tr}(S^{*2})$	$V^8 = S^*R^*S^{*2} - S^{*2}R^*S^*$
$V^4 = R^{*2} - \frac{1}{3}I\mathrm{Tr}(R^{*2})$	$V^9 = R^{*2}S^{*2} + S^{*2}R^{*2} - \frac{2}{3}I\mathrm{Tr}(S^{*2}R^{*2})$
$V^5 = R^*S^{*2} - S^{*2}R^*$	$V^{10} = R^*S^{*2}R^{*2} - R^{*2}S^{*2}R^*$

The coefficients α_k are functions of the non-dimensional invariants I_k formed by R^* and S^*.

$I_1 = \mathrm{Tr}(S^{*2})$	$I_4 = \mathrm{Tr}(R^{*2}S^*)$
$I_2 = \mathrm{Tr}(R^{*2})$	$I_5 = \mathrm{Tr}(R^{*2}S^{*2})$
$I_3 = \mathrm{Tr}(S^{*3})$	

τ is the turbulent timescale. In its original work, Pope suggested two possible ways to evaluate it:

$$\tau_\omega = \frac{k}{\epsilon} = \frac{1}{\omega} \qquad \text{or} \qquad \tau_\upsilon = \left(\frac{\partial u_i}{\partial x_j}\frac{\partial u_i}{\partial x_j}\right)^{-1/2}$$

In this work, the first approach $\tau = \tau_\omega$ is adopted, following Pope's recommendation [44]. This correction model is called the Explicit Algebraic Reynolds Stress Model (EARSM).

Since then, extensive research has been conducted based on this nonlinear correction. Initially, these studies aimed to identify algebraic identity to compute α_k. However, since the 2000s, with the rise of artificial intelligence and the growing availability of high-fidelity data, these coefficients have gradually transformed into functions. These functions are determined using various machine learning algorithms, such as:

- ML technics[60] : to directly correct the Reynolds stress tensor based on mean flow features, leading to improved accuracy even on different test cases
- evolutionary algorithm[52] : to obtain the coefficients within the framework of the EARSM model

- random forest model[66] : to estimate the confidence of RANS solutions before prediction

3.6 RANS SST k-ω model

There are many different two-equation transport models, and this variety comes from the choice of the second transported variable used to determine the turbulence length scale ℓ^*. While all models use turbulent kinetic energy k to define the velocity scale u^*, the length scale can be derived from several options:

- **Turbulent dissipation** ϵ, which, combined with k, gives the length scale $\ell^* = \frac{k^{3/2}}{\epsilon}$,
- **Specific dissipation** $\omega = \frac{\epsilon}{k}$, which provides the length scale $\ell^* = \frac{k^{1/2}}{\omega}$,
- A **direct turbulence length scale** ℓ.

Based on these approaches, we get the $k-\epsilon$, $k-\omega$, and $k-\ell$ turbulence models, along with their variants.

In this work, we focused on Menter's SST (Shear Stress Transport) k-ω model [34].

The classical k-ω model by Wilcox [63] is highly sensitive to the boundary conditions imposed on the specific dissipation rate ω at the edges of boundary layers and wakes. To mitigate this sensitivity, Menter proposed a two-layer model that combines the k-ϵ and k-ω models. The core idea is to use the k-ω model in the inner region of the boundary layer and, through a blending function, transition to the k-ϵ model outside the boundary layer. This approach makes the model less sensitive to the boundary conditions imposed on ω outside the boundary layer.

Additionally, based on Bradshaw's observations of two-dimensional boundary layers with adverse pressure gradients, the ratio of shear stress $\Sigma = \widetilde{u_i''u_j''}$ to turbulent kinetic energy k is approximately:

$$\frac{\Sigma}{k} \approx \sqrt{C_\mu} \approx 0.3$$

where $C_\mu = 0.09$. However, in a k-ϵ or k-ω turbulence model, this ratio is given by:

$$\frac{\Sigma}{\overline{\rho} k} = \sqrt{C_\mu \frac{P}{\epsilon}}$$

In situations where the ratio $\frac{P}{\epsilon}$ exceeds 1, as is the case in boundary layer flows with adverse pressure gradients, the shear stresses tend to be overestimated. To address this issue, Menter introduced a limiter on the turbulent viscosity in the outer region of boundary layers.

The resulting k-ω SST model is expressed as:

$$\begin{cases} \dfrac{\partial(\rho k)}{\partial t} + \dfrac{\partial(\rho u_j k)}{\partial x_j} = P - \beta^* \rho \omega k + \dfrac{\partial}{\partial x_j}\left[(\mu + \sigma_k \mu_t)\dfrac{\partial k}{\partial x_j}\right] \\ \dfrac{\partial(\rho \omega)}{\partial t} + \dfrac{\partial(\rho u_j \omega)}{\partial x_j} = \dfrac{\gamma}{\mu_t} P - \beta \rho \omega^2 + \dfrac{\partial}{\partial x_j}\left[(\mu + \sigma_\omega \mu_t)\dfrac{\partial \omega}{\partial x_j}\right] + 2(1-F_1)\dfrac{\rho \sigma_{\omega 2}}{\omega}\dfrac{\partial k}{\partial x_j}\dfrac{\partial \omega}{\partial x_j} \end{cases}$$

The turbulent viscosity is expressed with the introduction of the limiter as:

$$\mu_t = \frac{\rho a_1 k}{\max(a_1\omega, \Omega F_2)} \tag{3.23}$$

With :

$$\Omega = \sqrt{2\Omega_{ij}\Omega_{ij}} \quad \text{and} \quad \Omega_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right) \tag{3.24}$$

The function F_1 facilitates the transition from the k-ω model near the wall to the k-ϵ model outside the boundary layer. Meanwhile, the function F_2 serves as a limiter for the turbulent viscosity. These functions are defined by the following expression :

$$F_1 = \tanh(arg_1^4) \quad \text{with} \quad arg_1 = min\left[max\left(\frac{\sqrt{k}}{0.09\omega y}, \frac{500\nu}{y^2\omega}\right), \frac{4\rho\sigma_{\omega 2}k}{D_\omega y^2}\right] \tag{3.25}$$

$$\text{and} \quad D_\omega = max\left(\frac{\rho\sigma_{\omega 2}}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial \omega}{\partial x_i}, 10^{-20}\right) \tag{3.26}$$

$$F_2 = \tanh(arg_2^2) \quad \text{with} \quad arg_2 = max\left(2\frac{\sqrt{k}}{0.09\omega y}, \frac{500\nu}{y^2\omega}\right) \tag{3.27}$$

with the wall distance y.

The constants in this model are derived from the constants of each individual model (with index 1 for the Wilcox model and index 2 for the Launder and Sharma model). For a given constant ϕ, the expression is:

$$\phi = F_1\phi_1 + (1 - F_1)\phi_2 \tag{3.28}$$

with :

$$\sigma_1^* = 0.5 \quad ; \quad \sigma_1 = 0.5 \quad ; \quad \beta_1 = 0.075 \quad ; \quad \sigma_{\omega 1} = 0$$
$$\sigma_2^* = 0.85 \quad ; \quad \sigma_2 = 0.856 \quad ; \quad \beta_2 = 0.0828 \quad ; \quad \sigma_{\omega 2} = 0.856$$
$$\kappa = 0.41 \quad ; \quad a_1 = \sqrt{\beta^*} = 0.3 \quad ; \quad \gamma_i = \frac{\beta_i}{\beta} - \sigma_i\frac{\kappa^2}{\sqrt{\beta^*}} \quad \text{for i = 1, 2}$$

3.7 Limitation of RANS model

While Reynolds-Averaged Navier-Stokes (RANS) models are widely used for simulating turbulent flows due to their computational efficiency and ability to provide time-averaged quantities, they come with several limitations:

3.7.1 Fundamental Theoretical Limitations

3.7.1.1 Closure Problem and Turbulence Modeling

The fundamental challenge in RANS modeling stems from the Reynolds averaging process itself. The nonlinear terms in the Navier-Stokes equations give rise to the closure problem, where the Reynolds stress tensor must be modeled rather than directly computed [64]. This leads to several inherent limitations:

- Loss of information about turbulent fluctuations during the averaging process
- Inability to capture non-local effects in turbulent transport
- Challenges in representing pressure-strain correlations accurately

[58] demonstrates that these limitations become particularly significant in flows where the turbulent structure plays a crucial role in the overall flow dynamics.

3.7.1.2 Isotropic Turbulence Assumptions

Many RANS models, particularly two-equation models like k-ε and k-ω, rely on the Boussinesq hypothesis, which assumes isotropic turbulence [14]. This assumption leads to significant limitations:

- Poor prediction of secondary flows in non-circular ducts [59]
- Inaccurate representation of flow separation and reattachment
- Inadequate modeling of rotating and curved flows

3.7.2 Specific Flow Configuration Limitations

3.7.2.1 Separated Flows

RANS models show significant weaknesses in predicting separated flows [30]:

- Difficulty in accurately predicting separation points
- Poor representation of recirculation regions
- Inaccurate prediction of reattachment lengths

Experimental studies by [11] have shown discrepancies between RANS predictions and measured data in backward-facing step flows, particularly in:

- Reattachment length prediction
- Recovery region characteristics
- Turbulence intensity levels

3.7.2.2 Transitional Flows

The prediction of laminar-to-turbulent transition remains a significant challenge [35]:

- Limited ability to capture bypass transition
- Poor prediction of transition location under varying pressure gradients
- Inadequate representation of intermittency effects

3.7.2.3 Complex Geometries and Three-Dimensional Effects

RANS models face particular challenges in complex geometric configurations [9]:

- Poor prediction of corner flows and junction flows
- Limitations in capturing secondary flows
- Inaccurate representation of three-dimensional separation

3.7.3 Numerical and Implementation Limitations

3.7.3.1 Grid Sensitivity

RANS models exhibit significant sensitivity to grid resolution and quality :

- Wall treatment and near-wall grid requirements
- Solution dependency on grid structure
- Convergence issues in complex geometries

3.7.3.2 Boundary Condition Sensitivity

The accuracy of RANS predictions heavily depends on boundary condition specification [19]:

- Inlet turbulence specification effects
- Outflow boundary condition influences
- Wall roughness treatment limitations

CHAPTER 4

Introduction to deep learning

Contents

4.1 Introduction 36
4.2 Historical Context 37
4.3 Artificial Neural Networks 38
4.3.1 Artificial Neuron 39
4.4 Multi-Layer Perceptron (MLP) 39
4.4.1 Overview of the Structure 40
4.4.2 Explanation of Feedforward Connections 40
4.4.3 Activation Functions Used in MLPs 41
4.4.4 Mathematical Representation 41
4.4.5 Loss Function 43
4.4.6 Backpropagation 43
4.4.7 Training Process for MLPs 45
4.4.8 Overfitting and Regularization in MLPs 47
4.4.9 Hyperparameter Tuning for MLPs 47
4.5 Generative Adversarial Network 50
4.5.1 Introduction to Generative Adversarial Networks 50

4.5.2 Structure of Generative Adversarial Networks 50

4.5.3 Training Process of Generative Adversarial Networks 52

4.1 Introduction

This section examines the topic of neural networks, a crucial aspect of artificial intelligence that has transformed the way we approach complex data analysis. Neural networks have emerged as a critical technology in modern artificial Intelligence (AI), providing robust frameworks for modeling intricate patterns and relationships inherent in wide datasets. Their remarkable capacity to learn from extensive volumes of information and generate predictive insights has had a profound impact across various sectors, including healthcare, finance, transportation, and technology. This transformation has not only enhanced decision-making processes but has also led to innovative solutions that redefine existing paradigms.

To fully appreciate the intricacies of neural networks, we will explore their fundamental architecture and operational mechanisms. Understanding these foundational components is essential for grasping the complexities involved in the functioning of neural networks. Key concepts such as neurons, which serve as the building blocks of the network; layers, which define the structure and depth of the model; activation functions, which introduce non-linearity and enable the network to capture complex patterns; and loss functions, which guide the learning process by quantifying errors, will be discussed in detail. These elements work in concert to enhance a network's ability to learn from data and progressively improve its predictive accuracy.

As we progress through this section, particular attention will be given to two prominent neural network architectures: Multilayer Perceptrons (MLPs) and Generative Adversarial Networks (GANs). MLPs represent a fundamental model that encapsulates the basic principles of neural computation, showcasing the layered approach to learning. Their straightforward architecture allows for a clear demonstration of how inputs are processed through interconnected layers, leading to meaningful outputs. In contrast, GANs present a more sophisticated framework that pushes the boundaries of neural network capabilities by employing adversarial training techniques. This innovative architecture facilitates the generation of new data, mimicking the statistical properties of a given dataset, and showcases the versatility of neural networks in generative tasks.

Both MLPs and GANs illustrate the diverse applications of neural networks, reflecting their significance in contemporary artificial intelligence research. By examining these architectures, we gain valuable insights into the mechanics of neural networks and their profound implications for advancing AI technology.

4.2 Historical Context

The history of neural networks dates back to the early 1940s, when the foundations for what would become a pivotal technology in artificial intelligence were first laid. The concept of an artificial neuron was introduced by Warren McCulloch, a neuroscientist, and Walter Pitts, a logician, in 1943. They developed a mathematical model of a neuron [33], known as the McCulloch-Pitts neuron. This model was designed to mimic the logical functions of the human brain, representing the first attempt to create a system capable of performing computations similar to those of biological neural networks. Their work established the groundwork for future developments in the field of neural computation, inspiring subsequent research into artificial intelligence.

During the 1950s and 1960s, significant advancements in neural network research were driven by pioneers such as Frank Rosenblatt. In 1958, Rosenblatt introduced the Perceptron [48], an algorithm aimed at modeling the neural processes associated with human vision. The Perceptron demonstrated the ability to learn from input data and recognize patterns, marking a significant leap forward in the field. However, its limitations became evident shortly thereafter. The influential 1969 book "Perceptrons" [36] by Marvin Minsky and Seymour Papert highlighted that single-layer Perceptrons were unable to solve certain classes of problems, such as those involving non-linearly separable data. This revelation led to a temporary decline in neural network research, as interest shifted toward alternative approaches in artificial intelligence.

Interest in neural networks experienced a resurgence in the 1980s, largely due to the development of multi-layered networks and the backpropagation algorithm. This revival was fueled by the efforts of researchers like Geoffrey Hinton, David Rumelhart, and Ronald Williams, who demonstrated that backpropagation could effectively train multi-layered networks. This advancement allowed neural networks to tackle more complex problems and achieve improved performance. As a result, researchers and practitioners began to recognize the potential of neural networks for various applications, rekindling enthusiasm in the field and leading to broader adoption in areas such as speech recognition, image processing, and natural language processing.

The turn of the 21st century marked the beginning of a new era for neural networks, driven by rapid advancements in computational power and the availability of large datasets. The rise of deep learning, a subfield of machine learning that involves neural networks with many layers, began to gain prominence. Breakthroughs in deep learning were propelled by the development of innovative architectures, such as Convolutional Neural Networks (CNNs), which excel in image recognition tasks, and Recurrent Neural Networks (RNNs), which are particularly effective for sequential data and time series analysis. The implementation of Graphics Processing Units (GPUs) for parallel processing significantly enhanced the capabilities of neural networks, allowing for the training of larger and more complex models within feasible timeframes.

Today, neural networks are at the forefront of artificial intelligence research and applications, impacting numerous fields, including computer vision, natural language processing, healthcare, and robotics. The continual evolution of neural network architectures and training techniques promises to drive further advancements, pushing the boundaries of what these systems can achieve. As we look to the future, the historical journey of neural networks reflects a dynamic interplay of theoretical insights and technological progress. This journey has not only shaped the development of intelligent systems but also revolutionized our understanding of cognition, perception, and decision-making processes.

4.3 Artificial Neural Networks

Artificial Neural Networks (ANNs) have become an integral part of modern artificial intelligence (AI) and machine learning (ML) due to their remarkable ability to learn patterns and representations from data. Inspired by the structure and function of the human brain, ANNs are designed to model complex relationships between inputs and outputs through interconnected layers of artificial neurons. Over the past few decades, these networks have been applied to a wide range of tasks, from image and speech recognition to natural language processing and decision-making systems, making them a cornerstone of contemporary AI research and applications.

The concept of artificial neural networks finds its roots in the study of biological neurons, particularly how they communicate through synapses in the human brain. Early work by Warren McCulloch and Walter Pitts [33] in 1943 proposed a simple computational model of a neuron, laying the foundation for future advancements in neural network research. These artificial neurons, often referred to as nodes or units, mimic the behavior of biological neurons by receiving inputs, processing them, and generating outputs. However, unlike biological systems, artificial neural networks are purely mathematical constructs designed to solve computational problems.

One of the earliest models of artificial neural networks was the perceptron, introduced by Frank Rosenblatt [48] in 1958. The perceptron was capable of performing binary classification tasks, making decisions based on a weighted sum of inputs passed through an activation function. While this was a significant step forward, perceptrons were limited in their ability to solve non-linearly separable problems, such as the XOR problem, as noted by Marvin Minsky and Seymour Papert [36] in 1969. This limitation stalled progress in neural network research for many years, a period often referred to as the "AI Winter."

However, the revival of interest in neural networks came in the 1980s with the introduction of the backpropagation algorithm, which enabled the training of multilayer perceptrons (MLPs)—a class of neural networks with multiple hidden layers capable of learning non-linear relationships. This advancement, coupled with increases in computational power and availability of large datasets, paved the way for the development of deep neural networks (DNNs) in the

2000s, where networks with many layers could be trained to solve highly complex tasks. The rise of deep learning has revolutionized fields such as computer vision, speech recognition, and robotics.

4.3.1 Artificial Neuron

The fundamental building block of an Artificial Neural Network (ANN) is the *artificial neuron*, a computational unit that serves as the core mechanism for processing input data. An artificial neuron applies a function f_w to a d-dimensional input vector $\mathbf{x} \in \mathbb{R}^d$ to produce an output. This neuron is characterized by several key components, including a weight vector $\mathbf{w} \in \mathbb{R}^d$, a bias term $b \in \mathbb{R}$, and an activation function ϕ.

The weight vector $\mathbf{w}$ determines the importance of each input feature, allowing the neuron to prioritize certain inputs over others. Each weight is adjusted during the training process, enabling the neuron to learn from the data and improve its performance over time. The bias term b acts as a threshold, shifting the activation function to control the output independently of the input, thereby enhancing the neuron's flexibility in modeling complex relationships.

The activation function ϕ introduces non-linearity into the neuron's output, which is crucial for enabling the network to capture intricate patterns and interactions within the data. The choice of activation function can significantly impact the network's performance and learning capacity. Commonly used activation functions include Sigmoid function, Rectified Linear Unit (ReLU), hyperbolic tangent

The resulting output $y \in \mathbb{R}$ of the artificial neuron is defined by the equation:

$$y = \phi(\mathbf{w}^T \mathbf{x} + b)$$

This formulation allows the neuron to effectively map inputs to outputs in a manner that can learn and model complex patterns within the data. The linear combination of inputs, represented by $\mathbf{w}^T \mathbf{x} + b$, forms the basis of the neuron's decision-making process. By applying the activation function ϕ, the neuron introduces non-linearity, enabling it to capture more intricate relationships in the input space.

4.4 Multi-Layer Perceptron (MLP)

A Multi-Layer Perceptron (MLP) is a foundational type of artificial neural network composed of multiple layers of nodes, or neurons. The architecture includes an input layer, one or more hidden layers, and an output layer. Each neuron in a given layer is connected to every neuron in the subsequent layer, resulting in a fully connected network. MLPs utilize a feedforward architecture, which means that information flows in a single direction—from the input layer

through the hidden layers to the output layer—without any cycles or loops. This design allows the MLP to perform complex computations and learn intricate patterns in the data.

In MLPs, each neuron applies an activation function to the weighted sum of its inputs, introducing non-linearity into the model. This capability enables the network to capture and approximate complex, non-linear functions. MLPs are typically trained using backpropagation and gradient descent algorithms, which iteratively adjust the weights and biases to minimize the error in predictions.

4.4.1 Overview of the Structure

- **Input Layer:** The input layer is composed of neurons that receive the initial data and forward it to the subsequent layers. Each neuron in this layer corresponds to one feature of the input data.
  - This design enables the network to directly ingest and process raw data, ensuring that each feature is independently fed into the network for further transformation.
- **Hidden Layers:** The hidden layers consist of neurons that process inputs from the previous layer using weights and biases. MLPs can have one or more hidden layers, each performing non-linear transformations of the inputs.
  - The presence of hidden layers allows the network to learn and model complex relationships within the data. Each hidden layer can be viewed as extracting increasingly abstract features from the input data, enabling the network to develop a more sophisticated understanding of the underlying patterns.
- **Output Layer:** The output layer comprises neurons that provide the final predictions of the network. The number of neurons in this layer depends on the specific task at hand, such as classification or regression.
  - This layer translates the abstract features identified by the hidden layers into a form suitable for the task, whether predicting a class label or a continuous value.

4.4.2 Explanation of Feedforward Connections

- **Feedforward Connections:** Feedforward connections denote the one-way connections between neurons across different layers. In an MLP, each neuron in one layer connects to every neuron in the following layer.
  - This configuration guarantees that information flows unidirectionally, from input to output, without any feedback loops. Such an arrangement simplifies the training process and enhances the network's stability.
- The flow of information traverses from the input layer to the hidden layers and ultimately to the output layer. This characteristic unidirectional flow of information is a defining feature of feedforward neural networks.
  - Consequently, each layer progressively refines and transforms the data, leading to a more accurate and reliable final output.

4.4.3 Activation Functions Used in MLPs

- **Role of Activation Functions:** Activation functions introduce non-linearity into the network, allowing it to learn and represent complex patterns effectively.
  - In the absence of activation functions, the entire network would behave as a linear model, regardless of the number of layers. Non-linear activation functions enable the network to approximate complex, non-linear functions, which is crucial for effective learning.
- **Common Activation Functions:**
  - **Sigmoid:**
$$\sigma(x) = \frac{1}{1+e^{-x}}$$
    * The sigmoid function maps any real-valued number into the range (0, 1), making it useful for probabilistic interpretations, particularly in binary classification tasks.
  - **Hyperbolic Tangent (tanh):**
$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$
    * The tanh function maps input values to the range (-1, 1), centering the data and often leading to better convergence in practice compared to the sigmoid function.
  - **Rectified Linear Unit (ReLU):**
$$\text{ReLU}(x) = \max(0, x)$$
    * ReLU introduces non-linearity while remaining computationally efficient. It helps mitigate the vanishing gradient problem, enabling faster training and improved performance in deep networks.
  - **Leaky ReLU:**
$$\text{Leaky ReLU}(x) = \max(0.01x, x)$$
    * Leaky ReLU addresses the "dying ReLU" problem by allowing a small, non-zero gradient when the unit is not active, ensuring that all neurons can continue to learn.

4.4.4 Mathematical Representation

The simplest and historically oldest network architecture is the multilayer perceptron (MLP), where neurons are arranged in layers. In this architecture, the output of a neuron in layer n is connected only to the inputs of neurons in layer $n+1$. When every neuron in a layer is connected to all neurons in the preceding layer, that layer is referred to as *fully connected*. The neurons in the initial layer receive the network inputs, while the outputs from the final layer serve as the network outputs.

Given that a layer is a stack of multiple neurons, we can express the output $y_k \in \mathbb{R}^{d_k}$ of layer k in a n-layers fully connected MLP as follows:

$$y_k = f_{W_k}(x_{k-1}) = \phi_k(W_k^T x_{k-1} + b_k)$$

Where:

- $x \in \mathbb{R}^{d_{k-1}}$: the vector of outputs from layer $k-1$ and the input to layer k.
- $W_k \in \mathbb{R}^{(k-1)\times k}$: the weight matrix for layer k.
- $b_k \in \mathbb{R}^k$: the bias for layer k.
- ϕ_k: activation function for layer k.

With these inputs and outputs defined for any layer k, we can recursively derive the formula for a fully connected MLP:

$$\begin{aligned} y &= \phi_n\left(W_n x_{n-1} + b_n\right) && (4.1)\\ &= \phi_n\left(W_n \phi_{n-1}\left(W_{n-1} x_{n-2} + b_{n-1}\right) + b_n\right) && (4.2)\\ &= \phi_n\left(W_n \phi_{n-1}\left(W_{n-1} \phi_{n-2}\left(W_{n-2} x_{n-3} + b_{n-2}\right) + b_{n-1}\right) + b_n\right) && (4.3) \end{aligned}$$

Where $\forall k \in \{1, \ldots, n\}$:

- $x_k \in \mathbb{R}^{d_k}$: the vector of outputs from layer k.
- $W_k \in \mathbb{R}^{(k-1)\times k}$: the weight matrix for layer k.
- $b_k \in \mathbb{R}^k$: the bias for layer k.
- ϕ_k: activation function for layer k.
- y: output of the neural network.

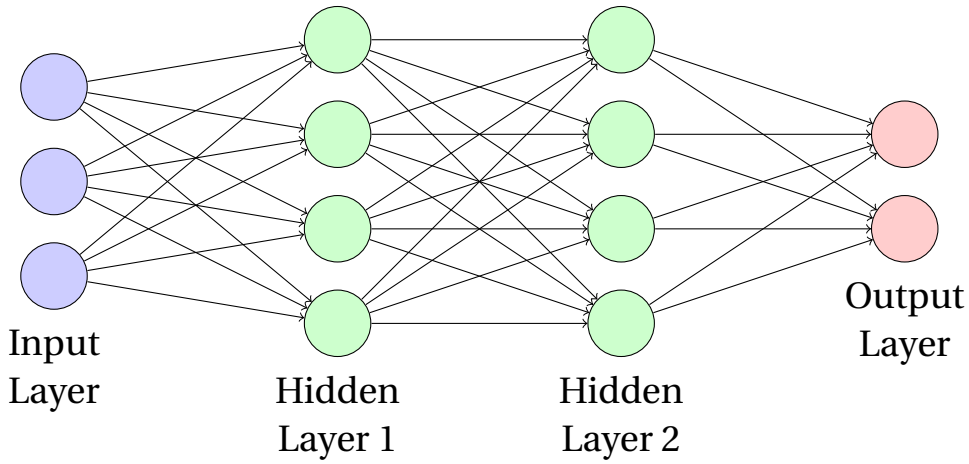


Figure 4.1: Illustration of a Multi-Layer Perceptron (MLP) architecture. The MLP consists of an input layer, two hidden layers, and an output layer. Each neuron in one layer is connected to every neuron in the next layer (feedforward connections). Activation functions (not shown) are applied to the outputs of each neuron to introduce non-linearity.

4.4.5 Loss Function

The training of Multi-Layer Perceptrons (MLPs) involves optimizing a loss function to minimize the discrepancy between the predicted outputs of the network and the ground truth labels.

The loss function measures the discrepancy between the predicted outputs of the network and the actual targets. It quantifies the error made by the network in its predictions and serves as a guide for adjusting the weights during training. The choice of loss function depends on the nature of the task, such as regression or classification, and the desired properties of the network outputs. For regression tasks, where the goal is to predict continuous values, common loss functions include the Mean Squared Error (MSE) or Mean Absolute Error (MAE). In contrast, for classification tasks, where the goal is to assign input data to discrete classes, cross-entropy loss is typically used to measure how well the predicted probability distribution aligns with the true class labels.

In some cases, regularization terms are added to the loss function to prevent overfitting. Regularization techniques, such as L2 regularization (also known as weight decay), penalize large weights in the network, encouraging the model to generalize better to unseen data. Additionally, for specific tasks, custom loss functions may be designed to incorporate domain-specific knowledge, ensuring that the model's training is tailored to the problem at hand. For example, physical laws or properties can be incorporated into the loss function as additional penalty terms, encouraging the network to adhere to these laws. Although adding such penalties does not guarantee that the laws or properties will always be respected, it biases the model towards satisfying them as often as possible, by penalizing violations during training.

4.4.6 Backpropagation

Backpropagation [62] is the algorithm used to efficiently compute the gradients of the loss function with respect to the weights of the network, enabling the optimization process.

Explanation of Backpropagation Algorithm:
Backpropagation is a fundamental algorithm in neural networks, designed to optimize the weights of the network by efficiently computing the gradients of the loss function with respect to each parameter. It works by propagating the error backwards through the network, starting from the output layer and moving towards the input layer. At each layer, the error is distributed and transformed according to the network's architecture. This backward pass utilizes the chain rule of calculus to decompose the gradients into manageable components, making it possible to calculate how much each weight contributed to the overall error.

In practice, backpropagation can be broken down into two key phases: the **forward pass** and the **backward pass**. During the forward pass, the network computes the output of the input data. The loss function then measures the difference between the predicted output and the true target values. In the backward pass, the gradients of the loss function with respect to the

activations and weights are computed layer by layer, starting from the output layer and progressing backwards. This backward flow of gradients allows for efficient weight updates across all layers of the network.

Derivation of Gradients and Updating of Weights:
The gradients of the loss function with respect to the weights are derived using the chain rule of calculus, where the gradient is decomposed into a sequence of simpler derivatives. The core idea is to trace how the error at the output layer propagates back through each layer, affecting the network's weights. Mathematically, this involves calculating the partial derivatives of the loss with respect to each weight, often referred to as $\frac{\partial L}{\partial w}$, where L is the loss function and w represents the weight parameters.

At each layer l, the gradients of the loss function are given by:

$$\delta^l = \frac{\partial L}{\partial z^l} \cdot \sigma'(z^l)$$

where δ^l is the gradient of the loss with respect to the weighted sum z^l, and $\sigma'(z^l)$ is the derivative of the activation function. These gradients are then used to compute the gradients with respect to the weights and biases, such as:

$$\frac{\partial L}{\partial W^l} = \delta^l \cdot a^{l-1}$$

where a^{l-1} represents the activations from the previous layer.

Once these gradients are computed, the weights are updated using an optimization algorithm, such as stochastic gradient descent (SGD) or one of its variants (e.g., Adam). The general weight update rule in SGD is given by:

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial L}{\partial w}$$

where η is the learning rate, a hyperparameter that controls the size of the weight updates. Choosing the appropriate learning rate is critical, as too large a value can lead to unstable training, while too small a value may result in slow convergence.

Regularization and Weight Updates:
To prevent overfitting, regularization terms such as L_2 regularization can be added to the loss

function, penalizing large weights and encouraging smoother, more generalizable models. This leads to modified weight update rules, such as:

$$w_{\text{new}} = w_{\text{old}} - \eta \left(\frac{\partial L}{\partial w} + \lambda w \right)$$

where λ is the regularization coefficient. Such techniques help ensure that the network learns meaningful patterns from the data without overfitting to noise or outliers.

4.4.7 Training Process for MLPs

Training Multi-Layer Perceptrons (MLPs) involves several phases aimed at updating the network's weights to minimize the loss function. This section provides a detailed overview of the training process, discusses various gradient descent methods, and introduces commonly used optimization algorithms.

4.4.7.1 Overview of Training Phases

The training process for MLPs unfolds in three primary phases:

1. **Forward Pass:** During the forward pass, input data propagates through the network, generating predictions. These predictions are compared with the ground truth labels to compute the loss. The loss function provides a measure of the discrepancy between the predicted outputs and the actual targets.

2. **Backward Pass:** The backward pass involves computing the gradients of the loss function with respect to the network's parameters (weights and biases) using backpropagation. These gradients indicate how much each parameter contributed to the error and serve as the basis for adjusting the weights. The chain rule of calculus is applied layer by layer to propagate the gradients back through the network.

3. **Weight Updates:** Based on the computed gradients, the network's parameters are updated using an optimization algorithm. The goal of this step is to minimize the loss function by adjusting weights and biases in a direction that reduces the error. Over successive updates, the network's performance improves as the parameters converge toward an optimal solution.

4.4.7.2 Discussion on Gradient Descent Methods

Various gradient descent methods are utilized for parameter updates during MLP training. Each method differs in how it computes the gradients and updates the parameters, offering different trade-offs between computational efficiency and accuracy:

- **Batch Gradient Descent:** This method [49] computes the gradients using the entire training dataset in each iteration. While it provides the most accurate parameter updates since it considers all training data, it can be computationally expensive and slow, especially for large datasets, as the full dataset needs to be processed during each epoch.
- **Mini-Batch Gradient Descent:** Mini-batch gradient descent[49] divides the training dataset into smaller batches, allowing for gradient computation and parameter updates after processing each batch. This method strikes a balance between the computational efficiency of stochastic gradient descent and the stability of batch gradient descent. By adjusting the batch size, it offers flexibility and is widely used in practice, particularly for large datasets.
- **Stochastic Gradient Descent (SGD):** In SGD[49], gradients are computed and parameters are updated for each individual training example. This method is computationally very efficient, but the updates can be noisy and less stable compared to batch or mini-batch gradient descent. The noise introduced by SGD can help escape local minima, but it may also lead to slower convergence and higher variability in the optimization process.

4.4.7.3 Introduction to Optimization Algorithms

In addition to the basic gradient descent methods, various optimization algorithms have been developed to improve the training efficiency of MLPs. These algorithms incorporate advanced techniques such as momentum and adaptive learning rates, which help accelerate convergence and improve the network's ability to generalize.

- **Adam (Adaptive Moment Estimation):** Adam [29] is an optimization algorithm that combines the benefits of two popular methods, momentum and RMSprop. It uses adaptive learning rates for each parameter, based on estimates of the first and second moments (mean and variance) of the gradients. Adam has become one of the most commonly used optimizers due to its efficiency and robustness in practice. The update rule for Adam is given by:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

  where g_t is the gradient at time step t, m_t and v_t are the first and second moment estimates, and $\beta_1, \beta_2, \eta, \epsilon$ are hyperparameters.
- **RMSprop (Root Mean Square Propagation):** RMSprop [21] is another popular optimization algorithm that addresses the issue of rapidly decaying learning rates in SGD. It adapts the learning rate for each parameter by dividing the gradient by a running average of the magnitudes of recent gradients. The update rule for RMSprop is:

$$v_t = \beta v_{t-1} + (1-\beta) g_t^2$$

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{v_t + \epsilon}} g_t$$

where g_t is the gradient at time step t, v_t is the exponentially weighted moving average of the squared gradients, and β and η are hyperparameters.

These optimization algorithms significantly impact the efficiency and convergence of the training process. By dynamically adjusting learning rates and leveraging momentum, they help MLPs achieve better performance and faster convergence toward optimal solutions.

4.4.8 Overfitting and Regularization in MLPs

Definition and Causes of Overfitting in MLPs: Overfitting occurs when a model learns the training data too well, capturing noise or irrelevant patterns that do not generalize to unseen data. This happens due to various reasons. Firstly, a complex model architecture with a large number of parameters relative to the size of the training dataset can lead to overfitting. Additionally, insufficient training data makes it easier for the model to memorize the training examples rather than learn generalizable patterns. Moreover, noisy or irrelevant features in the input data can also contribute to overfitting as the model may mistakenly learn to exploit them.

Explanation of Regularization Techniques: Regularization techniques help prevent overfitting by penalizing overly complex models. L1 and L2 regularization add a regularization term to the loss function, penalizing large weights in the model. L1 regularization adds the sum of the absolute values of the weights to the loss, while L2 regularization adds the sum of the squares of the weights. Dropout is another technique where randomly selected neurons are ignored during training with a certain probability, preventing co-adaptation of neurons and encouraging the network to learn more robust features. Early stopping halts the training process when the performance on a validation dataset starts to degrade, preventing the model from continuing to learn from noise in the training data.

Role of Regularization in Preventing Overfitting and Improving Generalization in MLPs: Regularization techniques play a crucial role in preventing overfitting and improving the generalization ability of MLPs. By penalizing overly complex models, dropout and regularization techniques encourage the network to learn simpler, more robust representations of the data. This helps prevent the model from fitting noise in the training data and allows it to generalize better to unseen examples, leading to improved performance on real-world tasks.

4.4.9 Hyperparameter Tuning for MLPs

The optimization of hyperparameters stands as a pivotal stage in the training of Multi-Layer Perceptrons (MLPs), directly influencing the efficacy and adaptability of the model. This section delves into a meticulous examination of hyperparameters, the methodologies employed for their refinement, and their indispensable role in enhancing MLP performance.

Understanding Hyperparameters: In the realm of MLPs, hyperparameters are the architectural and operational parameters that remain static throughout training. Profoundly impacting the model's learning dynamics, they encompass critical elements such as:

- **Network Architecture:** Comprising the network's depth (number of layers) and breadth (number of neurons per layer), the architecture profoundly shapes the model's capacity to extract intricate patterns from data.
- **Learning Rate:** A fundamental parameter governing the pace of parameter updates during training, the learning rate necessitates careful calibration to ensure effective convergence while averting divergence.
- **Activation Functions:** Choices encompassing ReLU, sigmoid, or tanh dictate the nonlinear transformations at each neuron, fundamentally influencing the network's expressive prowess.
- **Regularization Techniques:** Parameters such as L1 and L2 regularization strengths and dropout rates serve as bulwarks against overfitting, amplifying the model's generalization capabilities.

Strategies for Hyperparameter Tuning: Various methodological paradigms are employed for hyperparameter optimization in MLPs, each offering unique advantages:

- **Grid Search:** An exhaustive approach, grid search systematically traverses predefined hyperparameter grids, meticulously evaluating each combination to discern the optimal configuration.
- **Random Search:** Embracing stochasticity, random search selectively samples hyperparameter values from predefined ranges, adeptly exploring the parameter space to unearth promising configurations.
- **Bayesian Optimization:** Grounded in probabilistic modeling, Bayesian optimization harnesses past observations to intelligently predict the performance of unexplored hyperparameter configurations, thereby efficaciously navigating the optimization landscape.

Significance of Hyperparameter Tuning: The meticulous calibration of hyperparameters stands as a cornerstone in the quest for optimal MLP performance. By judiciously selecting hyperparameters, researchers can unlock the model's latent potential, fostering superior learning dynamics and bolstering generalization capabilities. A meticulously tuned model not only yields superlative performance metrics but also exhibits robustness and reliability in real-world scenarios.

In essence, hyperparameter tuning emerges as an indispensable facet of MLP training, serving as a conduit for unleashing the model's full prowess and realizing exemplary outcomes across diverse domains.

4.4.9.1 Evaluation and Testing of MLPs

This section thoroughly explores the multifaceted processes involved in evaluating and testing Multi-Layer Perceptrons (MLPs), critical for discerning their performance and generalization capabilities.

Overview of Evaluation Metrics: Evaluation metrics serve as quantitative measures to gauge the performance of MLPs across various tasks. Among the metrics specific to MLPs, accuracy stands as a fundamental measure, quantifying the proportion of correctly classified instances. Precision delves deeper, emphasizing the model's ability to minimize false positives by measuring the ratio of true positive predictions to the total number of positive predictions. Conversely, recall, also known as sensitivity, assesses the model's capacity to capture all positive instances by measuring the proportion of true positive predictions to the total number of actual positive instances. These metrics alone offer valuable insights into the model's performance, but the F1-score, a harmonic mean of precision and recall, provides a balanced assessment, particularly useful when dealing with imbalanced datasets or scenarios where both precision and recall are of equal importance.

Explanation of Cross-Validation: Cross-validation emerges as a robust technique for evaluating MLPs, offering insights into their stability and generalization capabilities. This technique involves partitioning the dataset into multiple subsets, typically through k-fold cross-validation or leave-one-out cross-validation. In k-fold cross-validation, the dataset is divided into k subsets, with each subset serving as the validation set once while the remaining subsets are used for training. This process is repeated k times, ensuring that each subset serves as the validation set exactly once. Leave-one-out cross-validation, on the other hand, involves training the model on all but one data point and evaluating it on the omitted data point. This process is repeated for each data point in the dataset. By assessing the model's performance across multiple subsets of the data, cross-validation offers a robust estimate of its generalization capabilities, mitigating the risk of overfitting.

Importance of Testing on Unseen Data: Testing on unseen data serves as the ultimate litmus test for MLPs, validating their generalization performance. By evaluating the model on data it has not encountered during training, researchers can ascertain its ability to make accurate predictions in real-world scenarios. This process helps mitigate the risk of overfitting, where the model memorizes the training data instead of learning underlying patterns, thereby ensuring that the model can effectively generalize to unseen instances. Testing on unseen data is paramount for validating the model's efficacy and reliability in practical applications, providing confidence in its performance beyond the confines of the training dataset.

In essence, evaluation and testing represent essential stages in the development and assessment of MLPs. Leveraging a diverse array of evaluation metrics, cross-validation techniques, and rigorous testing on unseen data enables researchers to gain profound insights into the model's performance and generalization capabilities, driving advancements in machine learning research and application.

4.5 Generative Adversarial Network

4.5.1 Introduction to Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a category of machine learning models designed to generate data samples that look like a given training dataset. Introduced by Ian Goodfellow and his colleagues in 2014 [24], Generative Adversarial Networks consist of two neural networks: the generator and the discriminator. These networks are trained concurrently through a process of adversarial competition. This adversarial interaction allows both networks to improve over time, leading to a generator that can produce realistic data samples. The concept behind Generative Adversarial Networks is based on game theory, where the generator seeks to increase the discriminator's error, and the discriminator tries to minimize it. This interaction results in the generation of data distributions that can be applied to tasks such as image generation, data augmentation, and creative applications like art synthesis.

The concept of Generative Adversarial Networks was first introduced in Ian Goodfellow's 2014 paper, "Generative Adversarial Nets." Before the advent of this architecture, generative models mainly relied on approaches such as Variational Autoencoders (VAEs) and Restricted Boltzmann Machines (RBMs), which had some limitations in producing sharp and realistic samples. The introduction of Generative Adversarial Networks brought a new approach to generative modeling, offering improved results in terms of generating realistic images.

Since their introduction, Generative Adversarial Networks have undergone various developments. Some important advancements include the Deep Convolutional GAN (DCGAN) in 2015, which improved the quality of generated images by using convolutional layers, and the Conditional GAN (cGAN), which allowed for more controlled generation by conditioning on additional inputs. The Wasserstein GAN (WGAN), introduced in 2017, addressed issues with training stability, which had been a challenge in earlier GAN models. Generative Adversarial Networks have since been used in a variety of applications, including image-to-image translation (CycleGAN), super-resolution (SRGAN), and text-to-image synthesis (StyleGAN).

Although Generative Adversarial Networks have seen significant success and diverse applications, their architecture can be challenging to understand. To better understand the relationship between the generator and discriminator, as well as the training process, we will explore their structure and operation in the following sections.

4.5.2 Structure of Generative Adversarial Networks

Generative Adversarial Networks consist of two neural networks, the *generator* and the *discriminator*, that are trained simultaneously through an adversarial process. Each network has a distinct role in the learning process, and their structures are designed to complement their tasks.

4.5.2.1 Generator

The generator is responsible for creating fake data samples that closely resemble real data. It achieves this by transforming a random noise vector, typically sampled from a simple distribution, such as a Gaussian or uniform distribution, into a data sample that ideally should be indistinguishable from real data. The structure of the generator typically comprises multiple layers of neural networks, including fully connected layers, convolutional layers (for image data), or other types of layers suited for the specific application (e.g., deconvolutional layers in image generation tasks).

- **Noise Input:** The generator takes a random noise vector $z \in \mathbb{R}^n$, where z is drawn from a prior distribution $p_z(z)$ (commonly a Gaussian distribution). This input is passed through the network, which transforms the noise into a data sample $G(z)$ in the desired data space.
- **Network Structure:** The generator network is typically composed of a series of fully connected or convolutional layers, with activation functions such as ReLU or Leaky ReLU applied after each layer. The final output layer often uses a non-linear activation function, such as a sigmoid (for binary data) or a tanh function (for image data), to map the output to the desired range.
- **Objective:** The generator's goal is to generate data samples that can *fool* the discriminator into classifying them as real. During training, the generator minimizes the loss function by adjusting its weights to produce data that better mimics the real data distribution. The generator loss function can be expressed as:

  $$\mathscr{L}_G = \mathbb{E}_{z \sim p_z(z)} \left[\log(1 - D(G(z))) \right]$$

  where $D(G(z))$ represents the probability assigned by the discriminator to the fake sample $G(z)$, and the generator aims to minimize this value by maximizing the discriminator's error.

4.5.2.2 Discriminator

The discriminator acts as the adversary to the generator. It receives either real data samples (from the training set) or fake data samples (generated by the generator) and is tasked with distinguishing between them. The structure of the discriminator typically mirrors that of a conventional binary classifier, processing the input through several layers and outputting a probability score indicating whether the input is real or fake.

- **Input:** The discriminator takes as input either real data $x \sim p_{data}(x)$ or fake data $G(z)$ generated by the generator. It processes this data through a neural network and outputs a scalar value $D(x)$ representing the probability that the input data is real.
- **Network Structure:** Like the generator, the discriminator typically consists of several layers, often convolutional in the case of image data. Each layer applies an activation function such as Leaky ReLU, and dropout may be used to prevent overfitting. The final layer outputs a probability value using a sigmoid activation function.
- **Objective:** The discriminator's goal is to maximize its ability to correctly distinguish between real and fake data. During training, it aims to correctly classify real samples as real

and fake samples as fake. The discriminator minimizes its own loss function, which is given by:

$$\mathscr{L}_D = -\left(\mathbb{E}_{x\sim p_{data}(x)}\left[\log D(x)\right] + \mathbb{E}_{z\sim p_z(z)}\left[\log(1-D(G(z)))\right]\right)$$

where $D(x)$ is the discriminator's probability that the input x is real, and $D(G(z))$ is the probability assigned to a generated sample.

4.5.2.3 Adversarial Process

The training of GANs is a competitive process, wherein the generator and the discriminator are trained simultaneously. The generator seeks to minimize its loss by producing more realistic data, while the discriminator seeks to maximize its accuracy by correctly distinguishing between real and fake data. The two networks are engaged in a minimax game with the following value function $V(G,D)$:

$$\min_G \max_D V(G,D) = \mathbb{E}_{x\sim p_{data}(x)}\left[\log D(x)\right] + \mathbb{E}_{z\sim p_z(z)}\left[\log(1-D(G(z)))\right]$$

This adversarial process forces the generator to continuously improve its outputs, as it tries to create data that can deceive the increasingly proficient discriminator. Conversely, the discriminator learns to identify the subtle differences between real and fake data, thereby pushing the generator towards producing more convincing samples over time.

4.5.2.4 Convergence and Stability Issues

The training of GANs is notoriously unstable due to the adversarial nature of the process. Convergence is difficult to guarantee, and several techniques, such as feature matching, batch normalization, and the use of Wasserstein loss (in Wasserstein GANs), have been introduced to improve training stability and ensure better convergence properties.

4.5.3 Training Process of Generative Adversarial Networks

The training process of Generative Adversarial Networks involves simultaneously training two neural networks—the generator and the discriminator—through an adversarial framework. This section outlines the key phases of GAN training, the loss functions for both networks, and techniques to improve the stability of the training process.

4.5.3.1 Adversarial Training Framework

In GANs, the generator and the discriminator are trained together in a competitive process, where the generator tries to produce data that mimics the real data distribution, while the discriminator attempts to distinguish between real and fake data. The training is a minimax game between the two networks, with the following objective function:

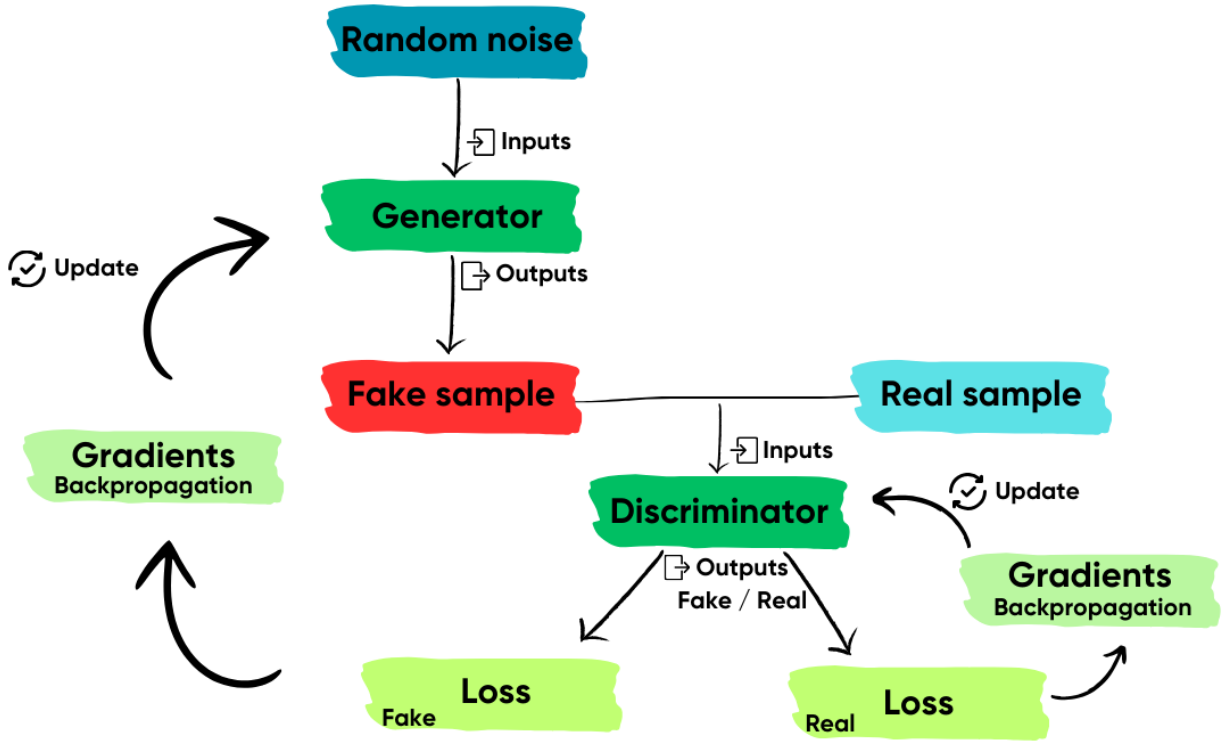


Figure 4.2: Illustration of a Generative Adversarial network

$$\min_G \max_D V(G,D) = \mathbb{E}_{x\sim p_{data}(x)}\left[\log D(x)\right] + \mathbb{E}_{z\sim p_z(z)}\left[\log(1-D(G(z)))\right]$$

Here, $D(x)$ is the probability that the discriminator assigns to the real data sample x, and $G(z)$ is the fake data generated from noise vector z by the generator. The generator aims to minimize this objective function by fooling the discriminator, while the discriminator aims to maximize its ability to correctly classify real and fake data.

4.5.3.2 Training Phases

The GAN training process consists of alternating updates to the generator and discriminator. Typically, the training cycle involves the following steps:

1. **Sample Real and Fake Data:** A batch of real data samples $x \sim p_{data}(x)$ is drawn from the training dataset, and a batch of noise vectors $z \sim p_z(z)$ is drawn from a random distribution (e.g., Gaussian or uniform).

2. **Update the Discriminator:** The discriminator is trained by maximizing the probability of correctly identifying real and fake samples. This is done by minimizing the discriminator's

loss function:

$$\mathscr{L}_D = -\mathbb{E}_{x\sim p_{data}(x)}\left[\log D(x)\right] - \mathbb{E}_{z\sim p_z(z)}\left[\log(1-D(G(z)))\right]$$

The discriminator adjusts its weights based on these samples to better distinguish between real and fake data in future iterations.

3. **Update the Generator:** The generator's objective is to fool the discriminator into classifying its outputs as real data. To achieve this, the generator minimizes its loss function:

$$\mathscr{L}_G = \mathbb{E}_{z\sim p_z(z)}\left[\log(1-D(G(z)))\right]$$

However, in practice, it is common to minimize the alternative loss:

$$\mathscr{L}_G = -\mathbb{E}_{z\sim p_z(z)}\left[\log D(G(z))\right]$$

This formulation helps improve gradient behavior during training. The generator updates its weights to reduce this loss, improving its ability to produce realistic data.

4. **Alternating Training:** The generator and discriminator are trained in alternating steps. Typically, after one update of the discriminator, one or more updates to the generator are performed to ensure that the generator keeps pace with the discriminator's learning.

4.5.3.3 Challenges and Stabilization Techniques

Training GANs can be challenging due to the adversarial nature of the process, which often leads to instability, vanishing gradients, or mode collapse. Several techniques have been proposed to improve the stability of GAN training:

- **Wasserstein GAN (WGAN):** The WGAN [1] introduces a new loss function based on the Earth-Mover (Wasserstein) distance, which improves gradient flow and stabilizes training by providing meaningful gradients even when the discriminator is optimal.
- **Batch Normalization:** Applying batch normalization [28] to both the generator and the discriminator helps mitigate issues related to internal covariate shifts, leading to more stable training.

4.5.3.4 Convergence and Evaluation

GANs do not have a well-defined convergence criterion, and training is typically stopped when the generator produces data samples that are visually indistinguishable from real data. Several evaluation metrics, such as the Inception Score (IS) and the Fréchet Inception Distance (FID), are commonly used to quantitatively assess the quality of the generated data.

In summary, the training process of GANs involves iterative and adversarial updates to both the generator and discriminator, with several techniques available to stabilize and improve the quality of the generated data. The competitive interaction between the two networks drives the generator towards producing realistic data, while the discriminator becomes more proficient at distinguishing real from fake, ultimately refining the performance of the GAN.

CHAPTER 5

Inputs features

Contents

5.1 Guidelines on flow feature selection 56

5.1.1 Galilean invariance 56

5.1.2 Rotational invariance 57

5.1.3 Non dimensionnal flow feature 57

5.2 Explicit Algebraic Reynolds Stress Model 58

The primary objective of this work is to enhance the accuracy of the Boussinesq approximation through the use of neural networks. However, the effectiveness of a neural network is not only contingent on its architecture or training process but is fundamentally influenced by the quality and relevance of the inputs it receives. Therefore, careful consideration and in-depth research were required to determine the most suitable inputs for the model.

This chapter aims to provide a comprehensive explanation of the decisions made regarding the selection of these inputs. First, we will discuss the nature of the data utilized in this work, including a detailed account of how the data was gathered and processed. The accuracy of any data-driven model relies heavily on the quality of the underlying dataset, so it is important to understand the origin and structure of the data being used.

Next, we will explore the rationale behind the selection of the inputs for the neural network. This process involved a detailed investigation into the various features of the flow that could be relevant to improving the Boussinesq approximation. By analyzing different flow variables and their impact on the network's predictive capabilities, we sought to identify the most informative features that would lead to the best possible model performance. Finally, we will explain the reasoning behind the final set of inputs chosen and how they contribute to enhancing the neural network's ability to correct the Boussinesq approximation.

5.1 Guidelines on flow feature selection

The application of machine learning techniques to turbulence modeling, particularly for Reynolds stress closure, requires careful consideration of fundamental physical principles. Among these, invariance properties stand as essential requirements that ensure the physical consistency of predictions regardless of the observer's frame of reference or coordinate system choice. This section presents a comprehensive analysis of these invariance requirements.

The Reynolds-averaged Navier-Stokes (RANS) equations inherit fundamental invariance properties from the original Navier-Stokes equations. These properties must be preserved in any machine learning framework to ensure physical consistency. The Reynolds stress tensor, τ_{ij}, being a second-order tensor, must transform appropriately under coordinate transformations.

5.1.1 Galilean invariance

Definition 5.1. *Galilean invariance ensures that the physics remains unchanged under a uniform translation of the reference frame. For a Galilean transformation:*

$$x_i' = x_i + V_i \tag{5.1}$$

$$u_i' = u_i + V_i \tag{5.2}$$

To ensure Galilean invariance [45], input features for machine learning models must be constructed from:

1. Velocity gradients rather than absolute velocities:

$$\frac{\partial u_i}{\partial x_j} \tag{5.3}$$

2. Strain rate tensor:

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{5.4}$$

3. Rotation rate tensor:

$$\Omega_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right) \tag{5.5}$$

5.1.2 Rotational invariance

Rotational invariance[45] represents a fundamental physical requirement in turbulence modeling, ensuring that the predicted physics remains unchanged under coordinate system rotations.

Definition 5.2. *Under a coordinate rotation, a physical quantity must transform according to specific rules to maintain consistency. For a rotational transformation defined by an orthogonal matrix Q_{ij}:*

$$x'_i = Q_{ij} x_j \tag{5.6}$$

The velocity components transform as:

$$u'_i = Q_{ij} u_j \tag{5.7}$$

where Q satisfies the orthogonality condition:

$$Q^t Q = I_d \tag{5.8}$$

5.1.3 Non dimensionnal flow feature

Ensuring that flow features are non-dimensional[12] enhances the model's adaptability and applicability across diverse flow conditions and configurations. Non-dimensionality plays a critical role for the following reasons:

- Scale Independence: Non-dimensional features abstract away from specific physical units and magnitudes, making the correction model applicable across different flow scales. This property is crucial, as turbulence models are often deployed in scenarios that vary significantly in size and velocity—such as flow around an aircraft wing versus a turbine blade. By relying on scale-independent parameters, the correction model can generalize more effectively, maintaining accuracy across these diverse cases.
- Enhanced Flow Similarity and Comparability: Non-dimensional features provide a basis for comparing and capturing the underlying similarities across varied flow regimes. Using key non-dimensional numbers, such as Reynolds and Mach numbers, helps encapsulate the essential behavior of a flow. This enables the correction model to respond consistently across flows that may have different absolute properties but share similar non-dimensional characteristics.

5.2 Explicit Algebraic Reynolds Stress Model

For the remainder of this thesis, we will focus on the EARSM framework, which we will now review.

Pope [44] introduced a nonlinear correction to compute the Reynold stress tensor. This correction improves upon the linear Boussinesq approximation by incorporating additional terms, resulting in a nonlinear constitutive relationship between stress and strain. The model is based on the weak equilibrium hypothesis concerning the anisotropic tensor a_{ij}. The correction is formulated using a tensor basis V^i and scalar invariants I_k, expressed as follows:

$$\tau_{ij} = 2\mu_t S_{ij} - \frac{2}{3}\rho k \delta_{ij} - \rho k \sum_{k=1}^{10} \alpha_k V_{ij}^k \tag{5.9}$$

Here, the basis functions V_k are constructed from polynomials of the non-dimensional strain rate $S^*_{ij} = \tau S_{ij}$ and the rotation rate $R^*_{ij} = \tau R_{ij}$, where S_{ij} and R_{ij} are the strain rate and rotation rate define previously.

$V^1 = S^*$	$V^6 = R^{*2}S + S^*R^{*2} - \frac{2}{3}I\mathrm{Tr}(S^*R^{*2})$
$V^2 = S^*R^* - R^*S^*$	$V^7 = R^*S^*R^{*2} - R^{*2}S^*R^*$
$V^3 = S^{*2} - \frac{1}{3}I\mathrm{Tr}(S^{*2})$	$V^8 = S^*R^*S^{*2} - S^{*2}R^*S^*$
$V^4 = R^{*2} - \frac{1}{3}I\mathrm{Tr}(R^{*2})$	$V^9 = R^{*2}S^{*2} + S^{*2}R^{*2} - \frac{2}{3}I\mathrm{Tr}(S^{*2}R^{*2})$
$V^5 = R^*S^{*2} - S^{*2}R^*$	$V^{10} = R^*S^{*2}R^{*2} - R^{*2}S^{*2}R^*$

The coefficients α_k are functions of the non-dimensional invariants I_k, which are formed from R^* and S^*.

$I_1 = \mathrm{Tr}(S^{*2})$	$I_4 = \mathrm{Tr}(R^{*2}S^*)$
$I_2 = \mathrm{Tr}(R^{*2})$	$I_5 = \mathrm{Tr}(R^{*2}S^{*2})$
$I_3 = \mathrm{Tr}(S^{*3})$	

In this context, our goal is to construct the coefficients α_k using neural networks.

Following Pope's recommendation, we employed invariants as input features for the neural networks. It is important to emphasize the significant role played by these invariants. The basis described by Pope in this context is distinguished by its Galilean invariance and rotational invariance, meaning that it remains consistent under any change of inertial reference frame. Consequently, the scalar quantities $I_1, \ldots, I_5$, derived from this basis, also possess these crucial properties. Theses invariances are essentials because they ensure that the basis, the scalar invariants, and hence the coefficients α_k will be applicable and consistent across any inertial reference frame.

CHAPTER 6

Correction of the Boussinesq approximation by data-driven modelling

Contents

6.1 Introduction 60

6.2 Dataset Description: Extended Flows Over Periodic Hills with Parameterized Geometries 61

6.2.1 Introduction 61

6.2.2 Dataset Objectives and Scope 61

6.2.3 Reynolds Number and Bulk Velocity 62

6.2.4 Flow Configurations and Numerical Methods 62

6.2.5 Dataset Content 62

6.2.6 Validation 63

6.3 Realizability condition 63

6.3.1 Diagonal Constraints 63

6.3.2 Cauchy-Schwarz Inequality 64

6.3.3 Eigenvalue Conditions 64

6.3.4 Implications of Realizability Conditions in Turbulence Modeling 64

6.3.5 Realizability Constraints in Neural Network-Assisted Turbulence Models . 65

6.4 Artificial neural network 65
6.4.1 Multi layer perceptron 68
6.4.2 Generative adversarial network 78
6.4.3 Results 80
6.5 Constant and linear coefficients 82
6.5.1 Constant and linear correction 83
6.5.2 Characteristic time correction 89
6.5.3 RANS solver 92
6.5.4 Results 92

6.1 Introduction

One of the fundamental features—and limitations—of RANS models lies in their dependence on the Boussinesq approximation, a closure assumption that simplifies calculations by assuming a linear relationship between the Reynolds stress tensor and the strain rate tensor. However, high-fidelity data show that this approximation rarely holds true in practice. To address this, Pope [44] proposed a refined approach, leading to the development of explicit algebraic Reynolds stress models (EARSMs). This family of non-linear closure models employs a tensor basis to enforce Galilean invariance, a critical property that ensures model consistency across different coordinate systems and observer perspectives.

To further refine turbulence models, various machine learning methods have been introduced that either improve or bypass Pope's assumption. Data-augmented models, for example, use high-fidelity data to mitigate RANS model inaccuracies [67]. Ling and Templeton [31] utilized DNS and LES data to develop classifiers that identify regions where conventional models fall short, while Wang et al. [60] applied machine learning directly to correct the Reynolds stress tensor based on mean flow features, achieving robust improvements across multiple test cases. Similarly, [52, 61] employed evolutionary algorithms to optimize the coefficients of EARSM models.

Other approaches focus on assessing the reliability of RANS models prior to application. For instance, Wu et al. [66] applied a random forest model to estimate confidence in RANS solutions, while Ling et al. [32] proposed a neural network approach with built-in invariance properties to correct the Reynolds stress tensor. This concept was later extended to a more comprehensive, albeit computationally expensive, framework by Wu et al. [65]. Although these methods have proven effective, they often rely heavily on high-fidelity data, which can introduce structural inconsistencies within RANS models.

To address such issues and maintain model integrity, data assimilation techniques based on inverse methods [20] have been used to refine model parameters. Parish and Duraisamy [40] pi-

oneered a field inversion approach using Bayesian inference and machine learning to enhance RANS predictions, showing significant improvement in predicting flow separation around airfoils [55]. This methodology has also been adapted for transition modeling in turbomachinery applications [17]. In this work, we aim to advance these methods by integrating realizability constraints to bridge the gap between RANS models and DNS data, ultimately improving model predictions based on Pope's framework.

In this chapter, we will first describe the DNS data used to train our various neural networks. We will then discuss the realizability conditions, which are extremely important. Finally, we will outline each of our methods and present the results we obtained.

6.2 Dataset Description: Extended Flows Over Periodic Hills with Parameterized Geometries

6.2.1 Introduction

The development and validation of turbulence models, particularly those driven by data, require high-fidelity datasets that span a wide range of flow conditions. Existing datasets, however, often lack sufficient systematic variations in key flow parameters, such as geometry and Reynolds number. To address this limitation, the dataset *"Flows Over Periodic Hills of Parameterized Geometries"* was introduced by Xiao et al. [68]. This dataset was specifically designed to advance data-driven turbulence models, particularly for scenarios involving flow separation, by providing detailed information derived from direct numerical simulations (DNS). Since its initial release, the database has been significantly extended, now offering a comprehensive resource with systematic variations in hill shape, domain height, and streamwise extent, thereby enabling the study of turbulence across a wider range of separated flow scenarios.

6.2.2 Dataset Objectives and Scope

The extended dataset was developed to address limitations in the original version, introducing additional configurations to enable more robust training and validation of data-driven turbulence models. The parameterization now includes:

- **Hill Shape**: Scaling factor $\alpha = 0.5, \quad 0.75, \quad 1.0$ (baseline), $\quad 1.25, \quad 1.5$.
- **Domain Height**: $L_y/h = 2.024, \quad 3.036, \quad 4.048$.
- **Streamwise Extent**: Three lengths for the flat section, leading to total domain lengths $L_x/h = 3.858\alpha + 2.142, \quad 3.858\alpha + 5.142, \quad 3.858\alpha + 8.142$.

This expanded dataset includes 27 configurations.

6.2.3 Reynolds Number and Bulk Velocity

The Reynolds number (Re) is defined as:

$$Re_b = \frac{u_b h}{\nu} = 5600, \tag{6.1}$$

where:

- u_b is the bulk velocity, defined as the spatially-averaged streamwise velocity at the hill crest,
- h is the hill height (fixed across all configurations),
- ν is the kinematic viscosity of the fluid.

The bulk velocity U_b is computed as:

$$u_b = \frac{1}{A}\int_A u(x,y)\, dA, \tag{6.2}$$

where A represents the cross-sectional area at the hill crest, and $u(x,y)$ is the streamwise velocity at the given cross-section. For all cases in the dataset, $Re_b = 5,600$ is maintained, ensuring that the effects of geometric variations are isolated from changes in flow regime.

6.2.4 Flow Configurations and Numerical Methods

The dataset features a broad set of configurations, enabling detailed exploration of diverse flow regimes:

- **Incipient Separation**: Observed for flatter geometries.
- **Mild Separation**: Present in configurations with intermediate steepness.
- **Massive Separation**: Characteristic of steeper geometries.

The DNS simulations were performed using the *Incompact3d* solver, a high-order flow solver that employs:

- Sixth-order compact finite-difference schemes for spatial discretization,
- Third-order Adams-Bashforth scheme for time integration,
- Immersed boundary method for modeling the hill geometry with a direct forcing approach.

The solver ensures incompressibility using a fractional step method, solving the Poisson equation in the spectral space, and maintains divergence-free velocity fields to machine precision. Periodic boundary conditions are applied in the streamwise and spanwise directions, with no-slip boundary conditions at the top and bottom walls.

6.2.5 Dataset Content

The dataset includes a rich set of output files for each simulation:

- Time-averaged velocity, pressure, dissipation, and vorticity fields.
- Velocity and pressure fields.
- Reynolds stress components.

Data were collected over a sufficiently long averaging period ($T = 750h/u_b$), after the flow reached a fully developed state.

6.2.6 Validation

The dataset was rigorously validated by comparison with existing benchmark data, demonstrating excellent agreement in mean velocity profiles and Reynolds stress components across all configurations. Mesh resolution and timestep sensitivity studies were conducted to ensure numerical accuracy.

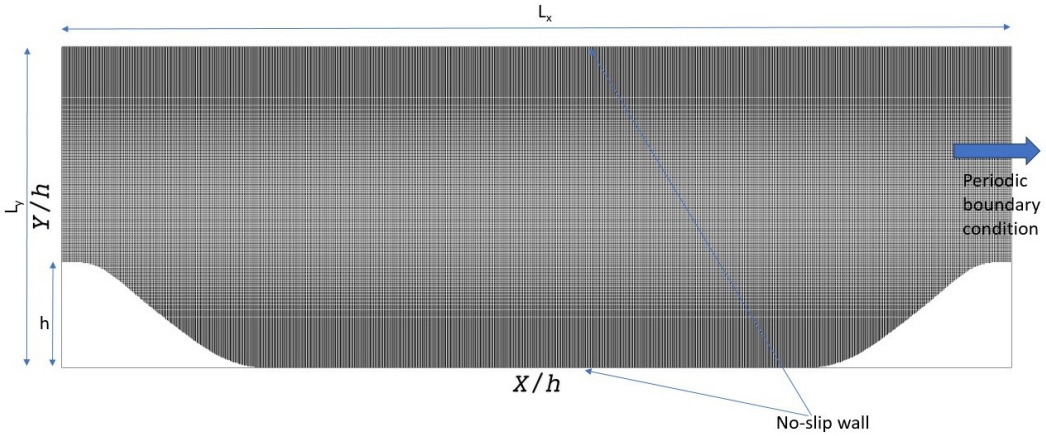


Figure 6.1: Computational domain for the baseline geometry

6.3 Realizability condition

In turbulence modeling, the Reynolds stress tensor plays a critical role in capturing the effects of turbulent fluctuations on the mean flow. However, for the Reynolds stress tensor to remain physically consistent, it must satisfy a set of mathematical and physical constraints known as *realizability conditions* [53]. These conditions ensure that the modeled stresses are consistent with the underlying physics of turbulence, preventing the model from generating unphysical results, such as negative energy or invalid stress states.

The most important realizability conditions include the following:

6.3.1 Diagonal Constraints

The diagonal elements of the Reynolds stress tensor correspond to the normal stresses in the flow, which represent the intensity of turbulence in each spatial direction. For a physically valid model, these normal stresses must be non-positive:

$$-\overline{\rho u_i'^2} \leq 0 \tag{6.3}$$

This condition reflects the fact that the turbulent kinetic energy (TKE) is always non-negative, meaning that the square of velocity fluctuations in each direction cannot be negative. Physically, this implies that turbulence cannot add energy to the system in a way that violates conservation principles. If this condition is violated, the model may predict an increase in turbulence that is not physically possible, leading to erroneous flow behavior.

6.3.2 Cauchy-Schwarz Inequality

The off-diagonal elements of the Reynolds stress tensor correspond to the covariances between different velocity components, which indicate the correlation between velocity fluctuations in different directions. The Cauchy-Schwarz inequality ensures that these covariances remain physically realistic:

$$\left(\overline{\rho u_i' u_j'}\right)^2 \leq \overline{\rho u_i'^2} \, \overline{\rho u_j'^2} \tag{6.4}$$

This inequality guarantees that the covariance between two velocity components does not exceed the product of their respective variances. In physical terms, this means that the cross-correlations between different components of turbulent velocity fluctuations must be bounded by the energy in each component. A violation of this condition would imply an unrealistic coupling between different components of turbulence, potentially leading to non-physical stress states.

6.3.3 Eigenvalue Conditions

Another important way to assess the realizability of the Reynolds stress tensor is to examine its eigenvalues. Since the Reynolds stress tensor is symmetric, its eigenvalues must all be real and non-positive. This condition ensures that the tensor remains negative semi-definite, meaning that the stress distribution resulting from turbulence does not introduce unphysical energy into the system:

$$\lambda_i \leq 0 \quad \forall i \tag{6.5}$$

where λ_i are the eigenvalues of the Reynolds stress tensor. If any eigenvalue is positive, it would suggest that the Reynolds stresses are contributing positive energy, which contradicts the fundamental principles of turbulence, where turbulence primarily dissipates energy rather than generating it.

6.3.4 Implications of Realizability Conditions in Turbulence Modeling

In practical applications, realizability conditions play a crucial role in maintaining the stability and accuracy of turbulence models, particularly in Reynolds-Averaged Navier-Stokes (RANS) simulations. Violations of these conditions can lead to non-physical results, such as negative

turbulence kinetic energy or unbounded stresses, which destabilize the numerical solver and result in incorrect flow predictions.

Turbulence models that do not explicitly enforce realizability conditions are prone to such issues, especially in complex flow regimes like those involving strong separations, reattachment, or adverse pressure gradients. As a result, many advanced turbulence models, including those enhanced by machine learning techniques, incorporate these constraints directly into their formulation or training process to prevent such violations.

6.3.5 Realizability Constraints in Neural Network-Assisted Turbulence Models

In the context of this work, realizability conditions are enforced through penalization terms introduced during the training process of the neural network used to correct the Boussinesq approximation. The Boussinesq approximation assumes a linear relationship between the anisotropic part of the Reynolds stress tensor and the strain rate tensor, which can lead to inaccuracies, especially in separated flows. To improve this approximation, non-linear corrections are applied using machine learning techniques. However, these corrections can potentially violate realizability conditions if not properly constrained.

To ensure that the Reynolds stress tensor remains physically valid throughout the flow domain, penalization terms are included in the training objective function. These terms penalize violations of the diagonal constraints, the Cauchy-Schwarz inequality, and the eigenvalue conditions. Specifically, a penalty function is added for each instance where an eigenvalue of the Reynolds stress tensor becomes positive, ensuring that the model generates only negative semi-definite stress tensors.

By integrating these realizability constraints directly into the model, the neural network is guided to produce corrections that improve the accuracy of the RANS simulation while maintaining the physical integrity of the stress tensor. This approach is particularly important in regions of complex flow, such as recirculation zones or boundary layers, where standard turbulence models may fail to capture the correct behavior.

6.4 Artificial neural network

In this section, we will focus on the correction of the Reynolds stress tensor within the framework of the EARSM model using a neural network. The EARSM model is as follow :

$$\tau_{ij} = 2\mu_t S_{ij} - \frac{2}{3}\rho k \delta_{ij} - \rho k \sum_{k=1}^{10} \alpha_k V_{ij}^k \tag{6.6}$$

Where V_k are basis functions formed from polynomials of non-dimensional strain rate $S^*_{ij} = \tau S_{ij}$ and rotation rate $R^*_{ij} = \tau R_{ij}$

$V^1 = S^*$	$V^6 = R^{*2}S + S^*R^{*2} - \frac{2}{3}I\mathrm{Tr}(S^*R^{*2})$
$V^2 = S^*R^* - R^*S^*$	$V^7 = R^*S^*R^{*2} - R^{*2}S^*R^*$
$V^3 = S^{*2} - \frac{1}{3}I\mathrm{Tr}(S^{*2})$	$V^8 = S^*R^*S^{*2} - S^{*2}R^*S^*$
$V^4 = R^{*2} - \frac{1}{3}I\mathrm{Tr}(R^{*2})$	$V^9 = R^{*2}S^{*2} + S^{*2}R^{*2} - \frac{2}{3}I\mathrm{Tr}(S^{*2}R^{*2})$
$V^5 = R^*S^{*2} - S^{*2}R^*$	$V^{10} = R^*S^{*2}R^{*2} - R^{*2}S^{*2}R^*$

The coefficients α_k are functions of the non-dimensionnal invariants I_k formed by R^* and S^*.

$I_1 = \mathrm{Tr}(S^{*2})$	$I_4 = \mathrm{Tr}(R^{*2}S^*)$
$I_2 = \mathrm{Tr}(R^{*2})$	$I_5 = \mathrm{Tr}(R^{*2}S^{*2})$
$I_3 = \mathrm{Tr}(S^{*3})$	

Here, the coefficients α_k will be determined by MLPs. To focus on statistically 2D flows, as is the case with flow over a periodic hill, only two invariants and three tensors basis are sufficient to have good results in a two-dimensional flow and the time for the training is greatly reduced.

Therefore, we have :

$$\tau_{ij} = 2\mu_t S_{ij} - \frac{2}{3}\rho k\delta_{ij} - \rho k\sum_{k=1}^{3} f_k(I_1, I_2)V^k_{ij} \tag{6.7}$$

Where :

$$\alpha_k = f(I_1, I_2) \quad \forall k \in \{1,2,3\} \tag{6.8}$$

Within the EARSM framework, and using DNS data from the flow over a periodic hill, calculating the optimal coefficients at each mesh point is fairly straightforward. Indeed, using DNS data, we can determine what the Boussinesq hypothesis should produce and compare it to the Reynolds stress derived from the DNS. Therefore, we should have :

$$\sum_{k=1}^{3} f_k(I_1, I_2)V_k \approx \frac{\tau^{DNS}_{ij}}{\rho k} + \frac{2}{3}\delta_{ij} - \frac{2\mu_t}{\rho k}S_{ij} \tag{6.9}$$

Finally, we determined the values of f_k at each mesh point using a least squares method. However, when plotting f_k against the invariants I_1 and I_2, we observe that some values can become quite large. For example, when examining f_1:

To avoid instabillities, we will need to smooth them by ensuring that the corrections applied between two neighboring points remain relatively consistent. Which lead to an optimisation problem of this form:

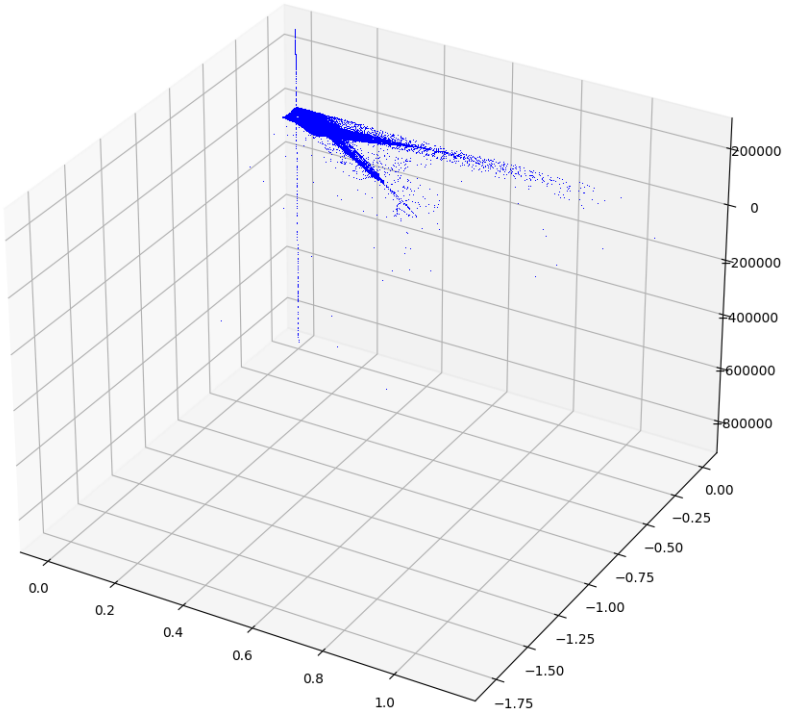


Figure 6.2: $f_1(I_1, I_2)$

$$argmin \|Ax - b\|^2 + \alpha \sum_{i=0}^{n} \|A_i x - b_i\|^2$$
$$\Rightarrow x = (A^T A + \alpha \sum_{i=0}^{n} A_i^T A_i)^{-1} (A^T b + \alpha \sum_{i=0}^{n} A_i^T b_i)$$

Where :

- A : aggregated matrix of V_k (each V_k is a line of A)
- x : unknown vector corresponding of $f_k(I_1, I_2)$
- b : aggregated vector of the right hand side of equation 6.9
- α : penalization factor
- A_i : Same as A but on the neighboring points
- b_i : Same as b but on the neighboring points

We then arrive at a more reasonable outcome :

At this point, we have established the inputs and outputs that the various neural networks are expected to generate.

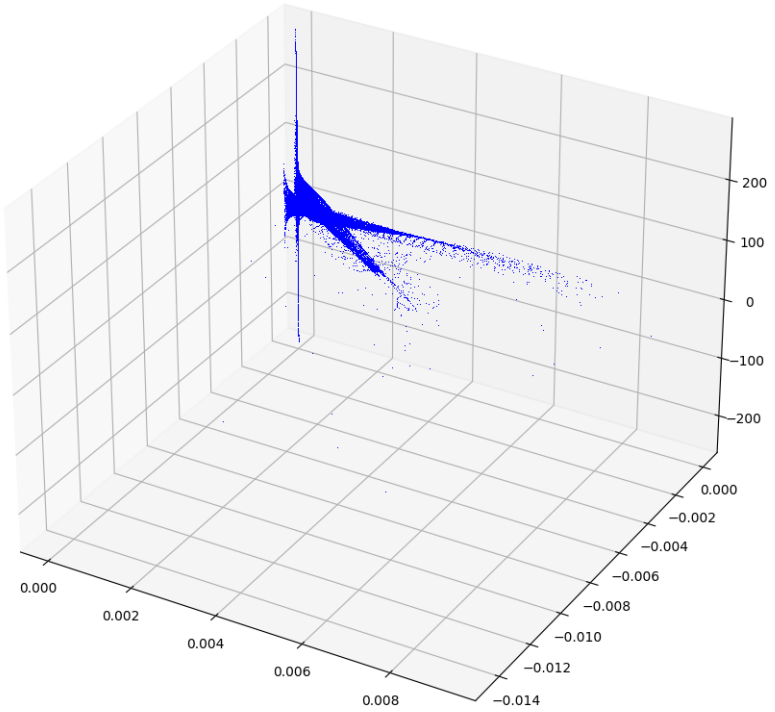


Figure 6.3: $f_1(I_1, I_2)$

6.4.1 Multi layer perceptron

6.4.1.1 First architecture

As detailed in the previous section, our objective is to determine the function $f_k(I_1, I_2)$ from equation 6.5. To achieve this, we will employ three multilayer perceptrons. Each neural network will take the two invariants as input and produce the value of one of the functions f_k as output 6.4. The mean squared error will serve as the loss function and a penalization term is added to take into account the realizability condition.

All three neural networks share the same architecture, consisting of 6 layers with 256 neurons in each layer. This architecture was selected through Bayesian hyperparameter optimization. Additionally, the networks will implement connection skipping to mitigate the vanishing gradient problem and enhance training stability and convergence.

To train this network, we will utilize 22 differentes meshes from the DNS database as the training set, reserving the remaining 5 for validation and 2 for the testing set. The neural networks were trained for 500 epochs using the ADAM optimizer, with the loss calculated as follows:

$$\left\| \tau^{DNS} - \tau^{Approx} \right\|^2 + \eta g$$

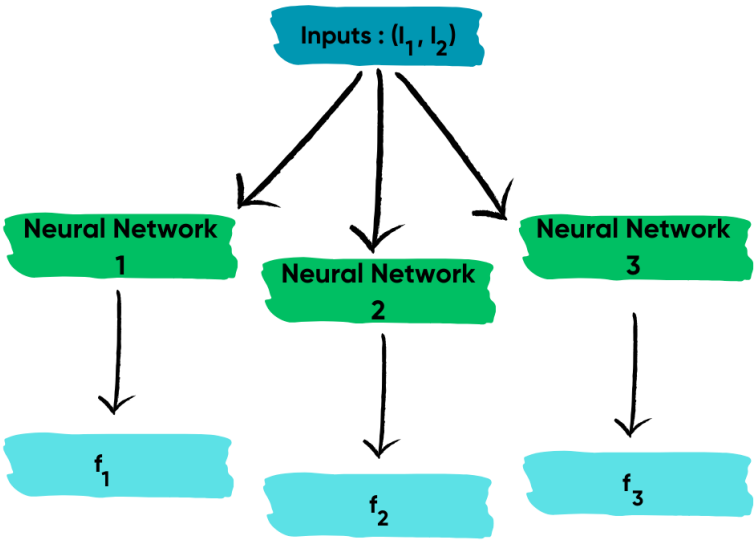


Figure 6.4: First architecture : The network consists of three identical neural networks, each tasked with providing one of the three different coefficients as output.

Where :

$$\begin{cases} \tau_{ij}^{DNS} = -\rho\overline{u_i' u_j'} \\ \tau_{ij}^{Approx} = 2\mu_t S_{ij} - \frac{2}{3}\rho k\delta_{ij} - \rho k \sum_{k=1}^{3} f_k(I_1, I_2) V_{ij}^k \\ g = \sum_{i=1}^{2} \max(\lambda_i, 0)^2, \quad \lambda = \text{eigen value of } \tau^{Approx} \\ \eta = 10 \end{cases}$$

The norm used to calculate the loss is the Frobenius norm. Additionally, g represents the constraint penalization. In the section 6.3.3, we defined that the Reynolds stress must have negative eigenvalues; therefore, we constrain the neural network by penalizing the loss with the sum of the positive eigenvalues. The square is added to ensure a differentiable penalization.

6.4.1.2 Results

In this section, we present the results obtained using the previously explained method. These results illustrate the training of the neural network using DNS data. At the end of this section, we will explain why we do not have results from using the different networks in a RANS model. Each graph will display the error produced by this method across various meshes (on the right). We first present results for two meshes from the training data, followed by two from the validation data, and finally two from the test data. For comparison, we will include the error introduced by the Boussinesq approximation for each of these meshes (on the left).

The error is calculated using the Frobenius norm:

$$\varepsilon^{Boussinesq} = \left\| \tau^{DNS} - \tau^{Boussinesq} \right\|^2$$
$$\varepsilon^{Approx} = \left\| \tau^{DNS} - \tau^{Approx} \right\|^2$$

Where :

$$\begin{cases} \tau^{DNS} = -\overline{\rho u_i' u_j'} \\ \tau^{Boussinesq} = 2\mu_t S_{ij} - \frac{2}{3}\rho k \delta_{ij} \\ \tau_{ij}^{Approx} = 2\mu_t S_{ij} - \frac{2}{3}\rho k \delta_{ij} - \rho k \sum_{k=1}^{3} f_k(I_1, I_2) V_{ij}^k \end{cases}$$

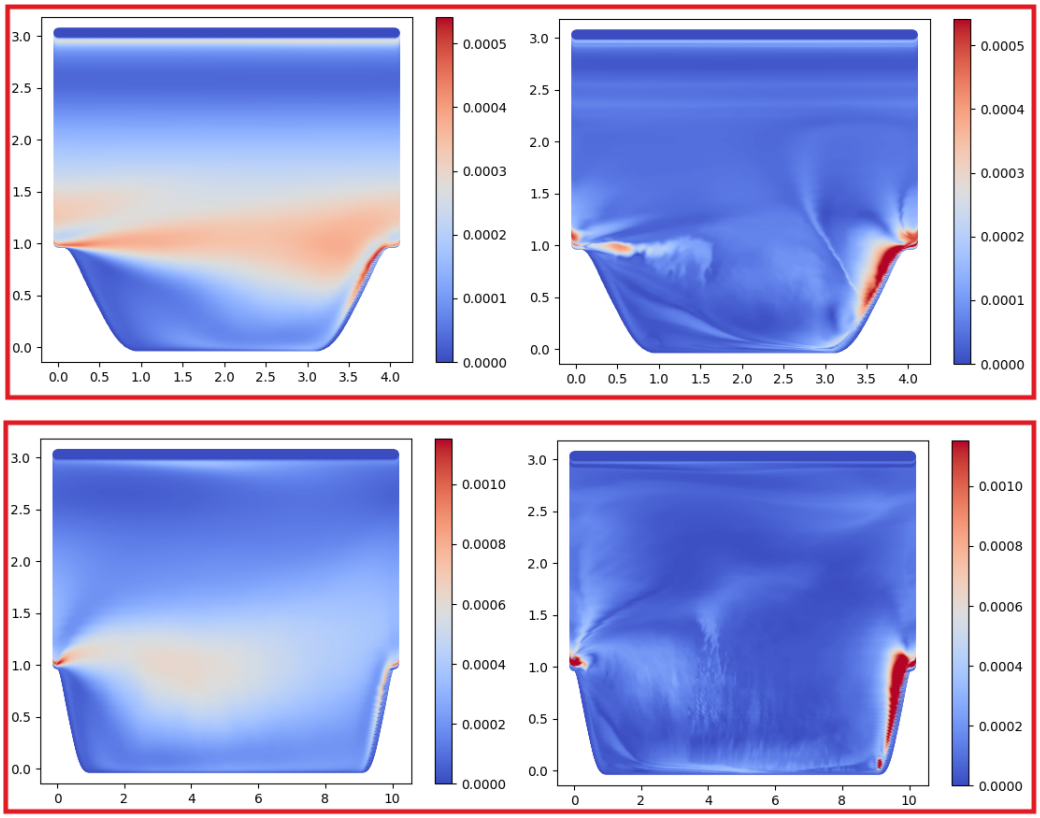


Figure 6.5: Frobenius Error of the Boussinesq assumption (left) and the corrected Boussinesq assumption (right) on two mesh used for training

As seen in the graphs 6.5 6.6 and 6.7, the error is significantly lower than with the Boussinesq model, except in two regions of the domain. For each of these results, whether for the training,

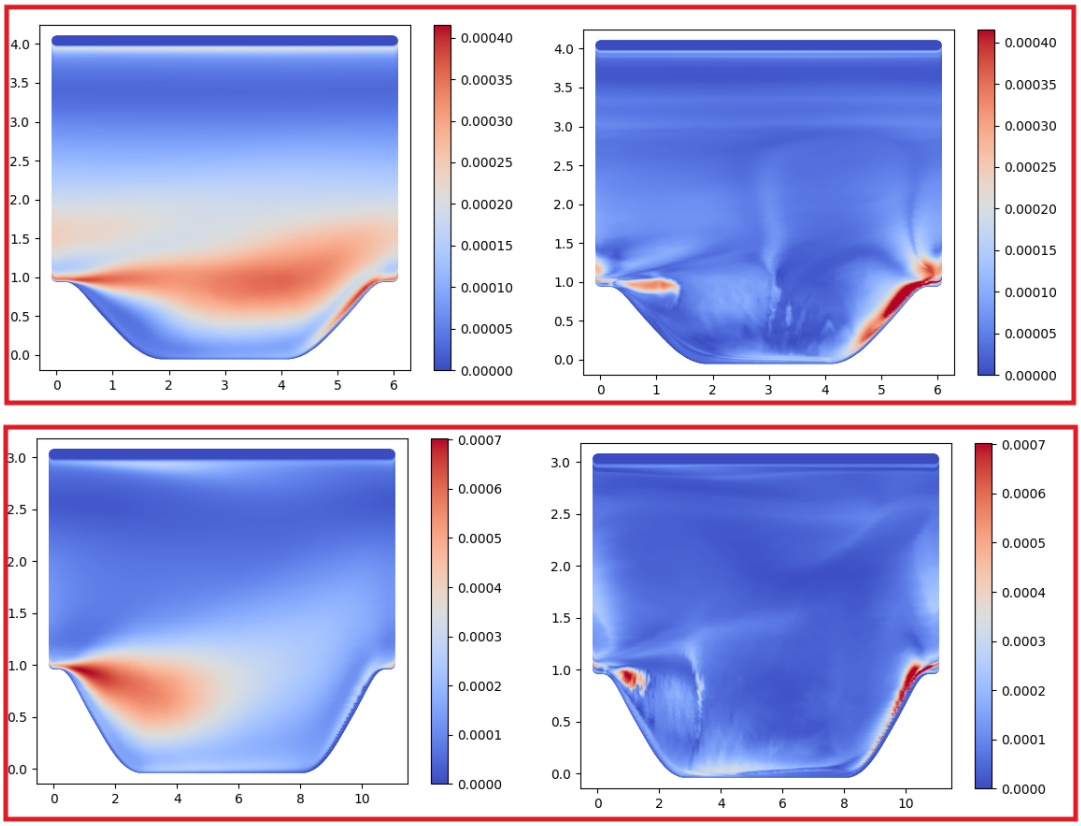


Figure 6.6: Frobenius Error of the Boussinesq assumption (left) and the corrected Boussinesq assumption (right) on two mesh used for validation

validation, or test datasets, we observe a significantly higher error in the fluid inlet region as well as in the area where the fluid flows back along the wall.

To address this, we will introduce a new neural network architecture.

6.4.1.3 Second architecture

In the previous method, we identified correction issues at two specific locations within the geometries. To resolve this, we will introduce a new neural network architecture designed to locate one of these areas using the same inputs as the correction networks. The choosen area is the one where the fluid flows back. This will lead to a less rigid correction for both regions. Initially, we will aim to identify regions within the inputs that contribute to inaccurate corrections. For this, we will use a clustering algorithm known as Gaussian Mixture to isolate these regions. As shown in figure 6.8 and figure 6.9, this algorithm successfully identifies the regions in certain geometries. To extend this across all geometries, we will train another clustering algorithm, the Support Vector Machine [8], on geometries where correct labels were obtained using Gaussian Mixture [10]. This approach will then be applied to geometries where the labels were initially

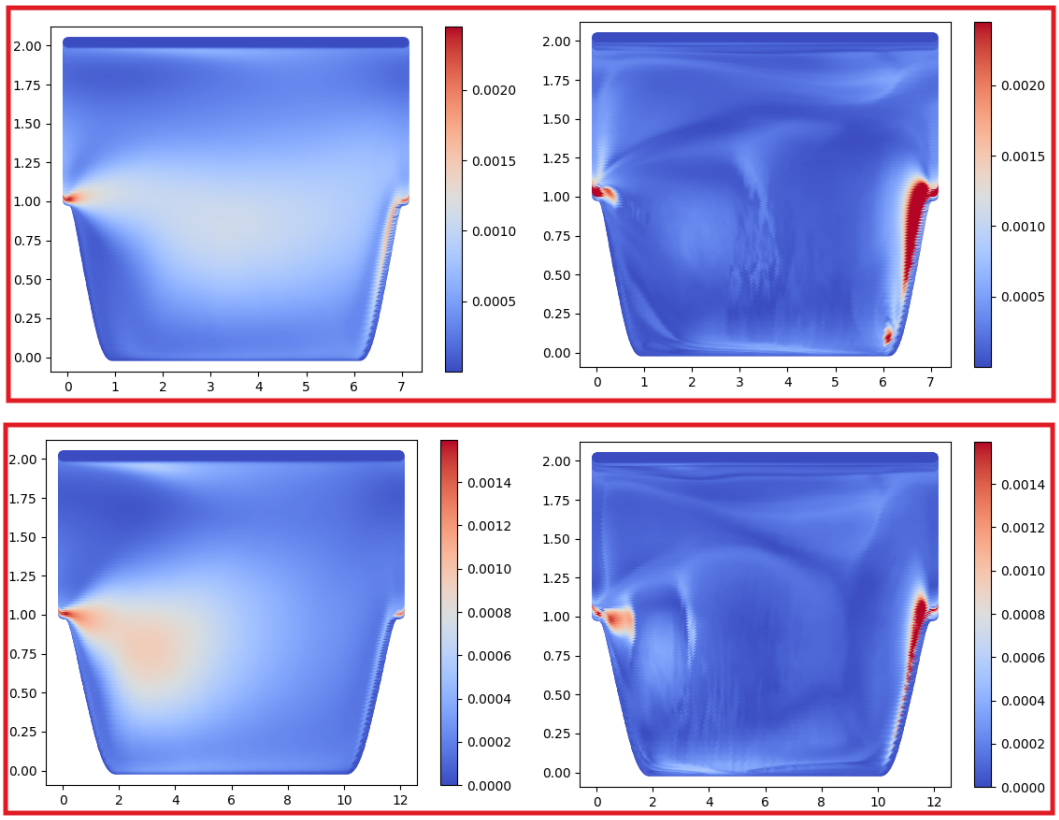


Figure 6.7: Frobenius Error of the Boussinesq assumption (left) and the corrected Boussinesq assumption (right) on two mesh used for testing

inaccurate 6.9.

As previously mentioned, the regions where the correction is inadequate can be identified through the correction inputs. We will leverage this new information to achieve an improved correction. This new architecture will consist of three components: two parts that generate distinct sets of coefficients and a third component that incorporates a blending function. Each component of this new architecture6.10 will be designed as a neural network structured as follows:

- The two components responsible for generating the coefficients (referred to as the regression network) will each be designed with the same architecture as the previous method. Therefore, each of these components will consist of three distinct MLPs, each featuring six layers with 256 neurons per layer, and employing the LeakyReLU activation function.
- The blending function will be implemented as a separate neural network with the following specifications: three layers, each consisting of 64 neurons, utilizing the LeakyReLU activation function, while the final layer will employ a Sigmoid activation function.

As before, to train this architecture, we utilized 22 different meshes from the DNS database as the training set, reserving the remaining 5 for validation and 2 for testing. The networks were

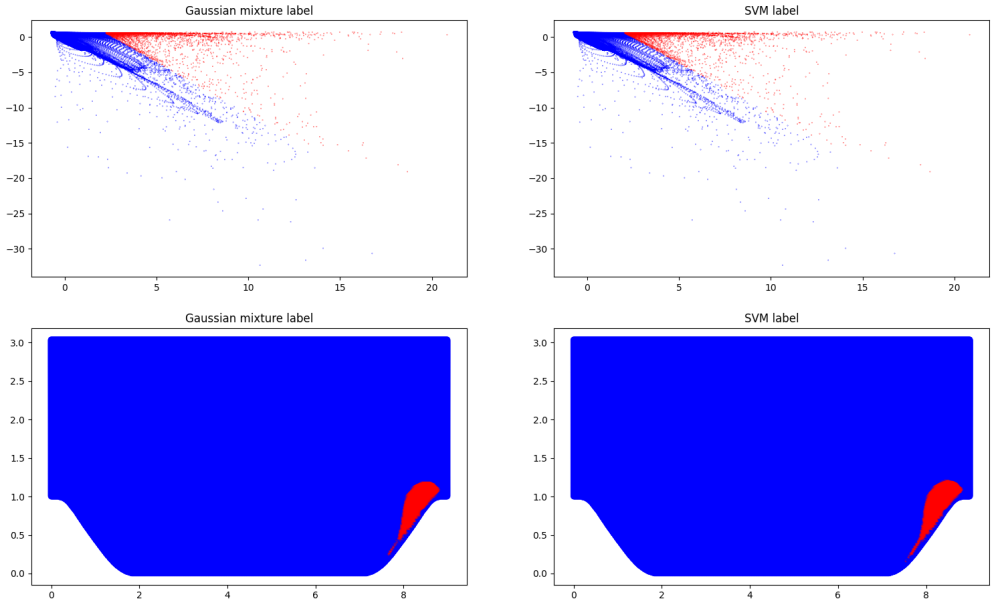


Figure 6.8: Clustering on a geometry where the Gaussian Mixture model was effective.

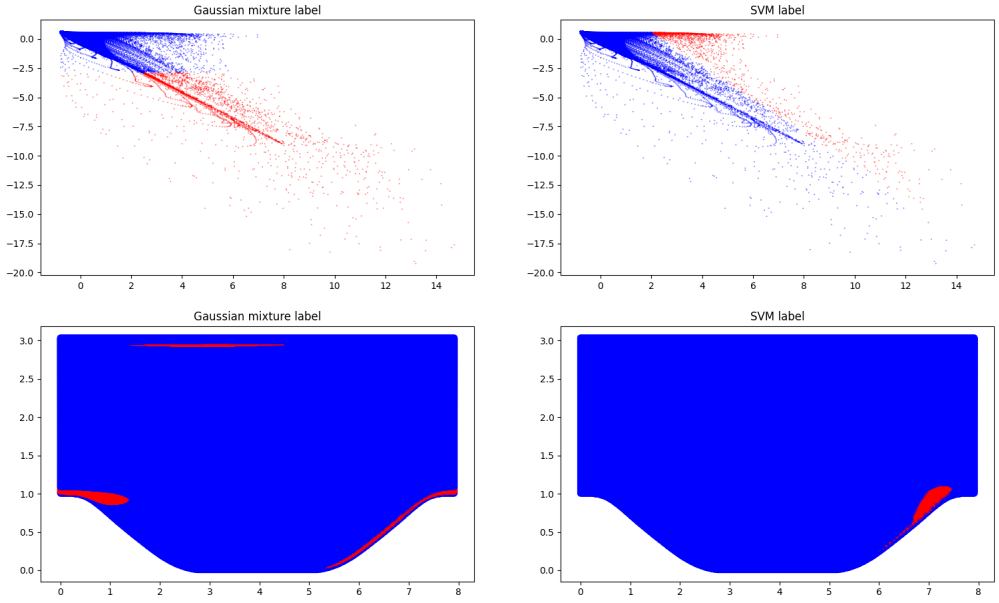


Figure 6.9: Clustering on a geometry where the Gaussian Mixture model was ineffective.

trained for 500 epochs using the ADAM optimizer, with the loss calculated as follows:

$$\left\| \tau^{DNS} - \tau^{Approx} \right\|^2 + \eta g$$

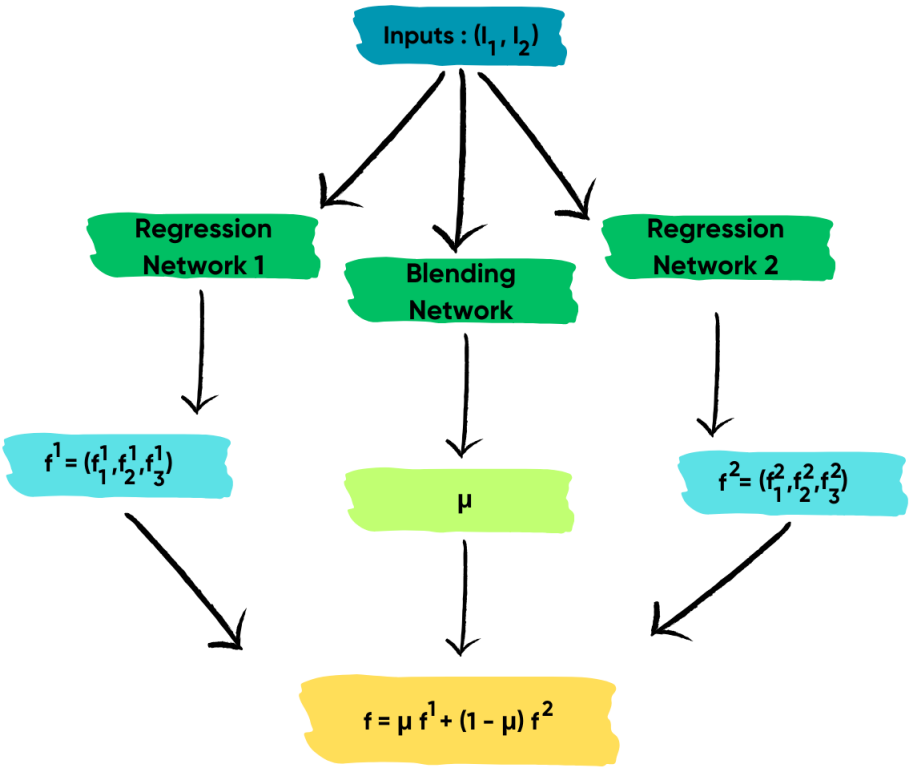


Figure 6.10: Second architecture: The network is composed of two identical regression networks, each responsible for predicting three different coefficients, along with a blending network designed to identify specific regions of the domain

where:

$$\begin{cases} \tau_{ij}^{DNS} = -\rho\overline{u_i' u_j'} \\ \tau_{ij}^{Approx} = 2\mu_t S_{ij} - \frac{2}{3}\rho k\delta_{ij} - \rho k \sum_{k=1}^{3} f_k(I_1, I_2) V_{ij}^k \\ g = \sum_{i=1}^{2} \max(\lambda_i, 0)^2, \quad \lambda = \text{eigenvalue of } \tau^{Approx} \\ \eta = 10 \end{cases}$$

The norm used to calculate the loss is the Frobenius norm, and g represents the constraint penalization.

6.4.1.4 Results

In this section, we present the results obtained. As before, the results shows the training of the neural network using only DNS data (offline). Each graph will display the error produced by the boussinesq assumption and this correction method accross various meshes. We will show at first the results from one geometry of the training data set then one from the validation data set and one from the testing data set. As previously, we will compute the error using the Frobenius norm :

$$\varepsilon^{Boussinesq} = \left\| \tau^{DNS} - \tau^{Boussinesq} \right\|^2$$
$$\varepsilon^{Approx} = \left\| \tau^{DNS} - \tau^{Approx} \right\|^2$$

Where :

$$\begin{cases} \tau^{DNS} = -\overline{\rho u_i' u_j'} \\ \tau^{Boussinesq} = 2\mu_t S_{ij} - \frac{2}{3}\rho k \delta_{ij} \\ \tau_{ij}^{Approx} = 2\mu_t S_{ij} - \frac{2}{3}\rho k \delta_{ij} - \rho k \sum_{k=1}^{3} f_k(I_1, I_2) V_{ij}^k \end{cases}$$

Additionally, we will include a graph illustrating the results of the neural network performing the blending.

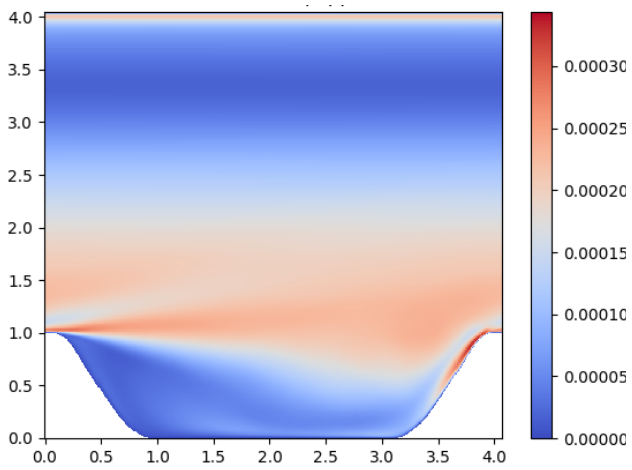


Figure 6.11: Error of the Boussinesq assumption on a mesh of the training set

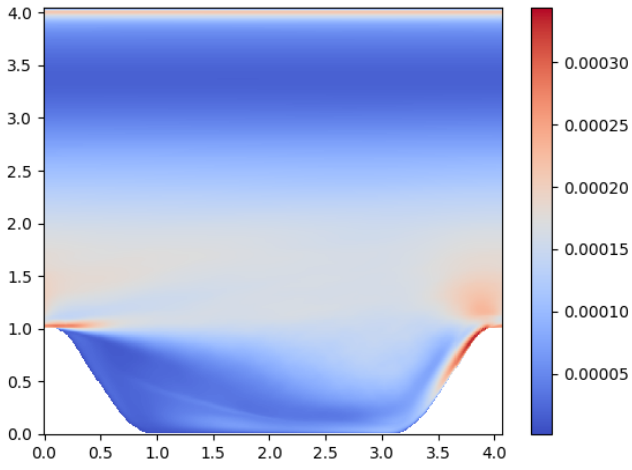


Figure 6.12: Error of the corrected Boussinesq assumption on a mesh of the training set

By employing this method, we obtained results 6.12, 6.15, 6.18 across all data sets that surpass those offered by the Boussinesq approximation. Furthermore, when compared to the first method presented, we observe that the issues at the two specific locations have been resolved.

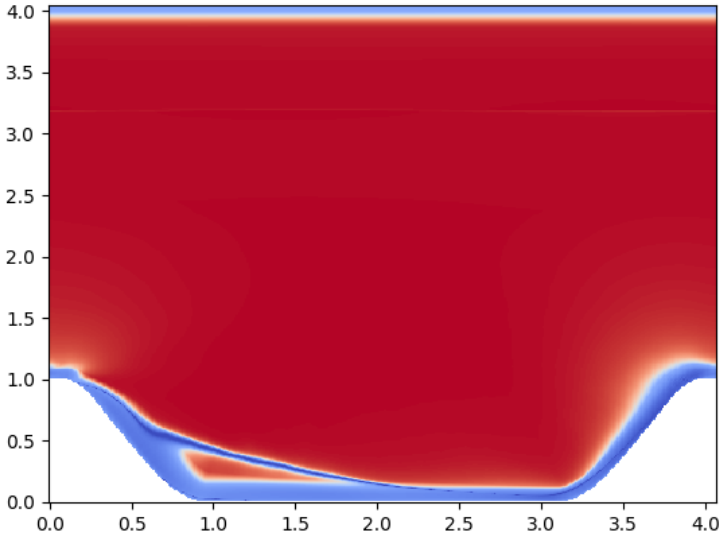


Figure 6.13: Results of the Blending network on a mesh of the training set

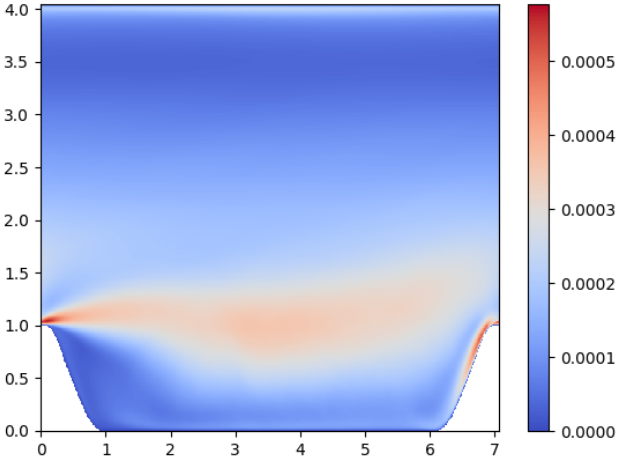


Figure 6.14: Error of the Boussinesq assumption on a mesh of the validation set

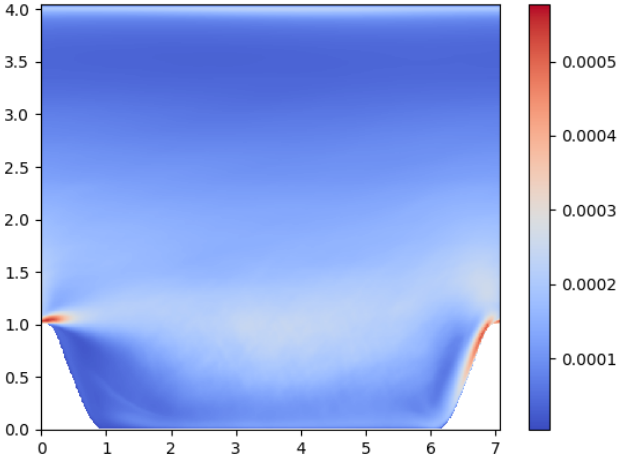


Figure 6.15: Error of the corrected Boussinesq assumption on a mesh of the validation set

Although the correction is slightly less accurate overall, we have significantly mitigated the substantial errors that were previously present. It is noteworthy that the network responsible for blending successfully identifies the problematic regions 6.13, 6.16, 6.19, even though we did not provide it with the corresponding labels.

6.4.1.5 Limitations

In this study, we presented a correction to the Boussinesq approximation using two distinct methods. Both methods were trained on DNS data, which provided a solid foundation for our

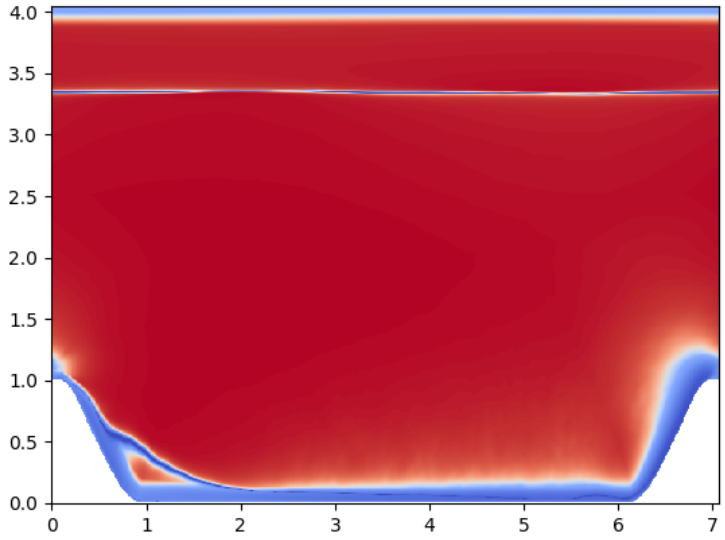


Figure 6.16: Results of the Blending network on a mesh of the validation set

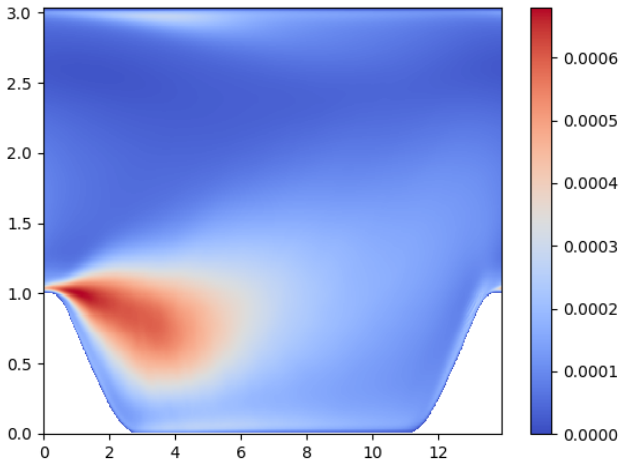


Figure 6.17: Error of the Boussinesq assumption on a mesh of the test set

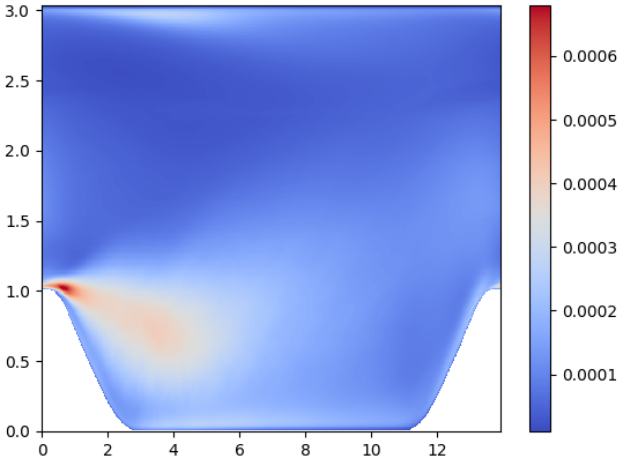


Figure 6.18: Error of the corrected Boussinesq assumption on a mesh of the test set

approach.

During our tests in a RANS framework, we encountered significant challenges, particularly regarding stability issues when incorporating the correction. These problems can be attributed to several key factors:

- **Dissipation Conditions**: In the standard SST model, it is essential that the dissipation associated with the anisotropic part of the Reynolds stress remains positive. Unfortunately, this condition was not consistently met with the correction applied. In certain regions of the tested meshes, the correction resulted in negative values for the Reynolds stress,

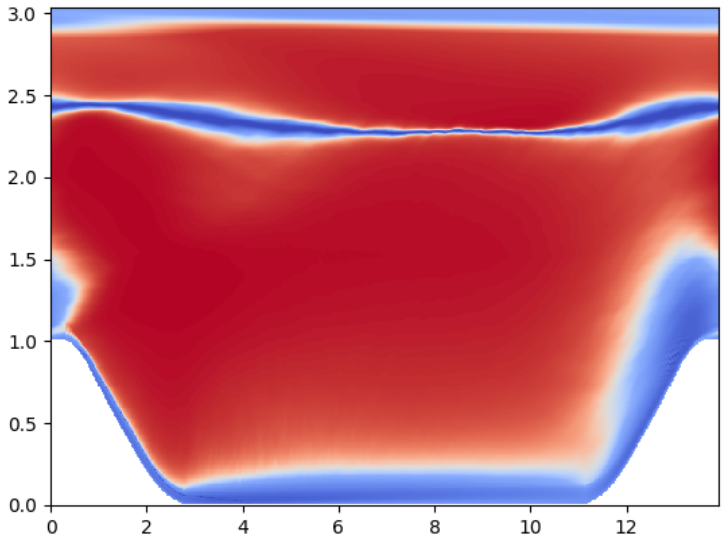


Figure 6.19: Results of the Blending network on a mesh of the test set

leading to strong oscillations in the numerical solution. This instability undermines the reliability of the simulation results and highlights the need for careful handling of the correction terms.

- **Characteristic Time Differences**: Another critical factor is the difference in characteristic time scales between the DNS data used for training our networks and the characteristic time in a RANS simulation.

6.4.2 Generative adversarial network

In this section, we will build upon our previous efforts to correct the Boussinesq approximation by leveraging DNS data while employing a more innovative approach using Generative Adversarial Networks (GANs). GANs are a unique class of neural networks that consist of two competing models: the generator, which creates synthetic data, and the discriminator, which evaluates the authenticity of the generated data against real data.

The use of GANs in our study allows us to explore a more sophisticated correction method that can potentially capture complex relationships within the data. By training the generator to produce accurate representations of the Reynolds stresses and having the discriminator assess the quality of these representations, we aim to enhance the performance of our correction strategy beyond what traditional methods can achieve.

It is important to note that the generator of a GAN typically takes random noise as input. However, in our case, we will draw inputs from the probability distribution of the available DNS data. This approach enables us to create more relevant and representative synthetic data that aligns closely with the characteristics of the underlying flow phenomena we are studying. By utiliz-

ing inputs derived from the DNS data distribution, we aim to enhance the generator's ability to produce accurate and meaningful corrections to the Boussinesq approximation.

To achieve this, we will employ an algorithm called kernel density estimation [42] on each geometries of the training dataset. This method will enable us to reconstruct the Reynolds stress from our training set, which consists of 22 different geometries. By utilizing kernel density estimation, we can effectively capture the underlying distribution of the Reynolds stress, providing a robust foundation for generating inputs that are representative of the flow characteristics within our data. This approach ensures that the generated synthetic data closely aligns with the true statistical properties of the Reynolds stresses across the various geometries in our training set. We will apply this algorithm to the data that allows us to reconstruct a Reynolds stress tensor. For this, we need:

- The velocity gradient: to reconstruct S, R, and therefore μ
- The turbulent kinetic energy k: to reconstruct μ
- The dissipation rate of turbulent kinetic energyϵ: to reconstruct μ

From these quantities, we can then reconstruct a Reynolds stress tensor using the Boussinesq approximation.

We will then sample data from these distributions to feed into our generator, which will be tasked with producing the correction. This generator will have the same architecture as the one used in our first example in the previous section 6.4.1.1. To recap, this architecture consists of three different MLPs, each corresponding to a set of coefficients. Each of these MLPs will be composed of six layers with 256 neurons, and they will use LeakyReLU as the activation function.

Using these inputs, the generator will create a Reynolds stress tensor based on the corrected Boussinesq approximation. These newly generated tensors, along with the inputs used to produce them, will then be fed into the discriminator, alongside data from the DNS. The Boussinesq approximation from the DNS data will be calculated using the coefficients derived through the least squares method presented at the beginning of this Section.

The discriminator will then assign a score indicating how likely it believes each example is real (from the DNS data) or synthetic (generated by the network). This score is then used to alternately train the discriminator and the generator, one after the other.

Figure 6.20: Visual representation of the procedure

6.4.3 Results

In this section, we present the results obtained using the GAN approach. As in previous sections, each graph will display the error produced by the Boussinesq approximation as well as the error produced by the GAN generator when applying the correction. Once the GAN is fully trained, we will rely exclusively on the generator to apply the correction.

As a reminder, the error is computed as follows :

$$\varepsilon^{Boussinesq} = \left\| \tau^{DNS} - \tau^{Boussinesq} \right\|^2$$
$$\varepsilon^{Approx} = \left\| \tau^{DNS} - \tau^{Approx} \right\|^2$$

Where :

$$\begin{cases} \tau^{DNS} = -\overline{\rho u_i' u_j'} \\ \tau^{Boussinesq} = 2\mu_t S_{ij} - \frac{2}{3}\rho k \delta_{ij} \\ \tau_{ij}^{Approx} = 2\mu_t S_{ij} - \frac{2}{3}\rho k \delta_{ij} - \rho k \sum_{k=1}^{3} f_k(I_1, I_2) V_{ij}^k \end{cases}$$

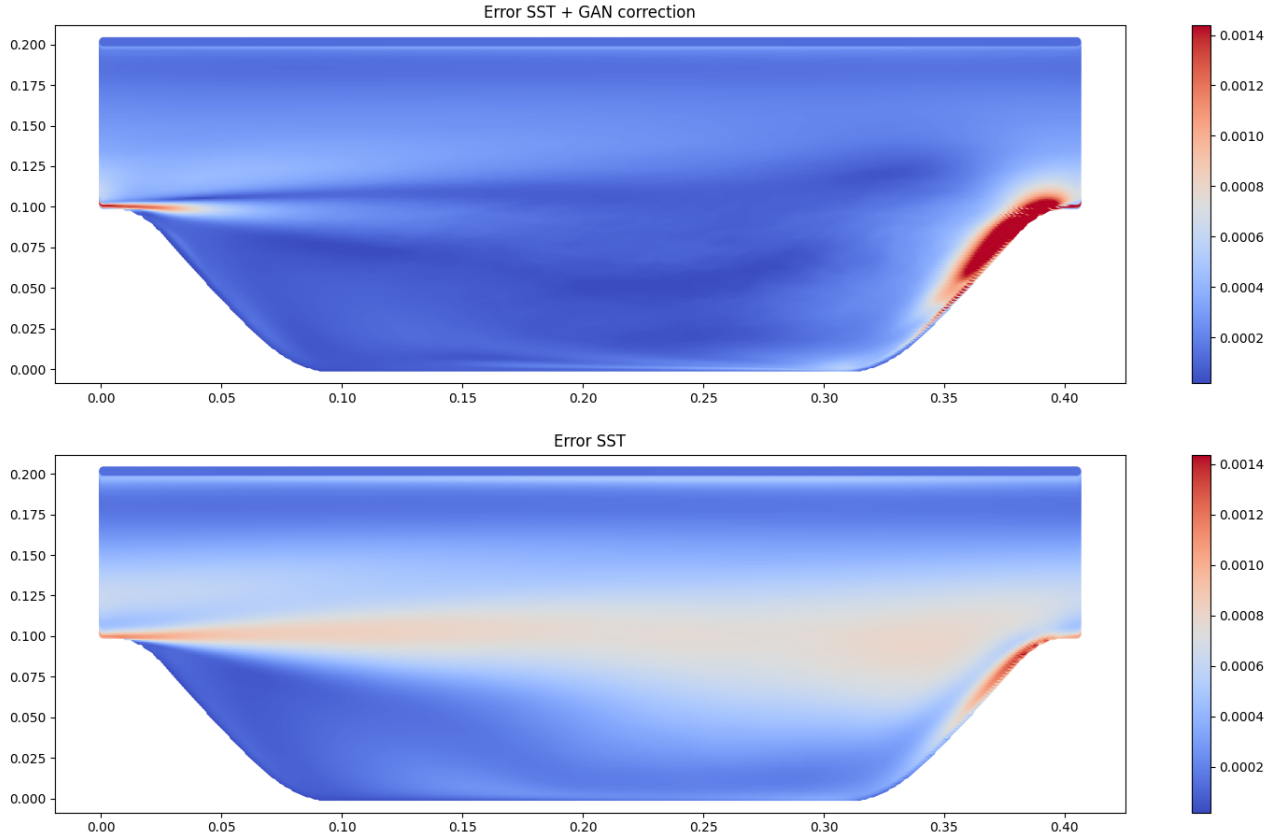


Figure 6.21: Error of the Reynold stress tensor on the training set

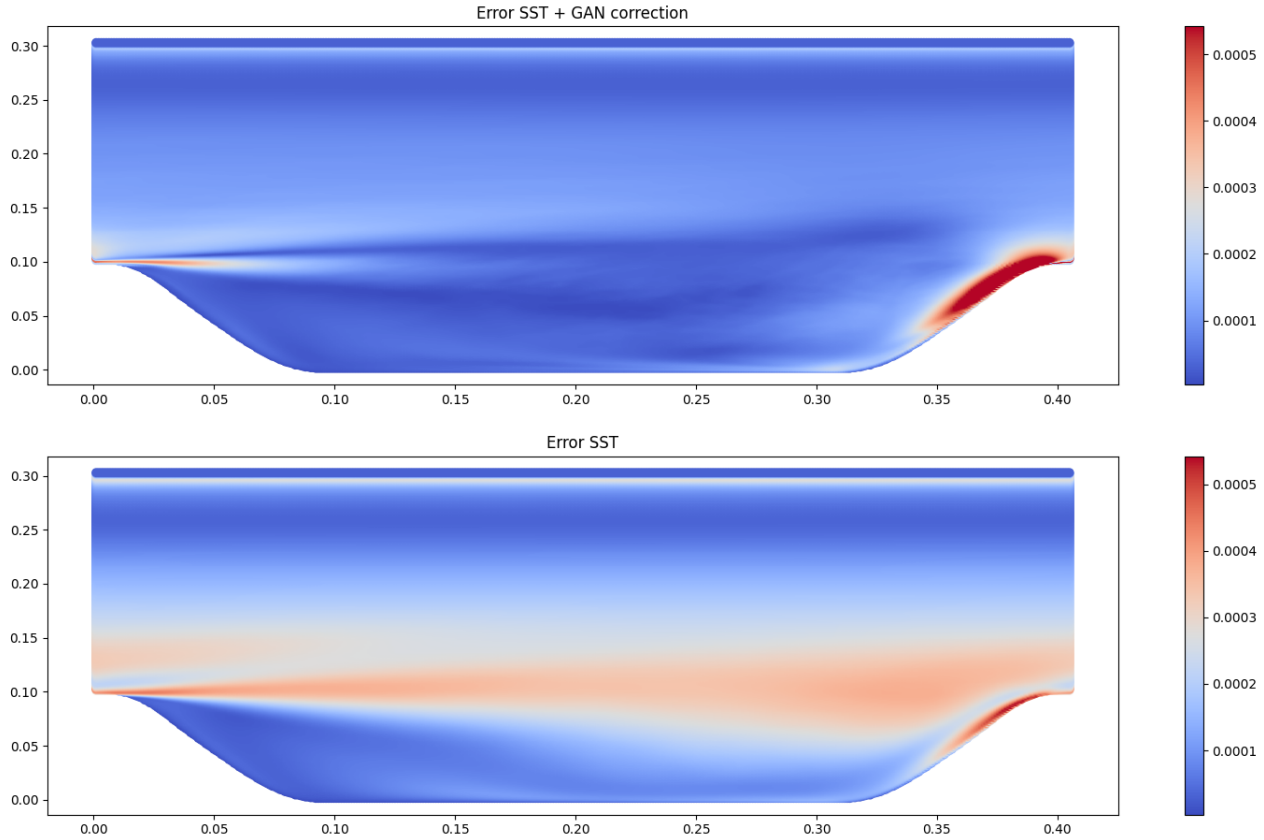


Figure 6.22: Error of the Reynold stress tensor on the test set

It is interesting to note that the results are similar to those obtained with the first architecture in the section on multilayer perceptrons despite a completely different training process. Unfortunately, we did not have the time to further explore these results with different architectures for the generator. Additionally, since the results were very similar to those obtained with the first method presented in this chapter, we did not attempt to introduce this correction into a RANS model, which would likely have led to the same complications.

6.5 Constant and linear coefficients

The initial step of the proposed methodology centers on the comprehensive analysis of high-fidelity data derived from Direct Numerical Simulations (DNS). This analysis aims to identify the coefficients of the non-linear correction outlined in Eq. 6.5:

$$\tau_{ij} = 2\mu_t S_{ij} - \frac{2}{3}\rho k \delta_{ij} - \rho k \sum_{k=1}^{10} \alpha_k V_{ij}^k$$

The ideal scenario involves expressing the coefficients α_k as functions of the flow invariants,

which are quantities that remain unchanged under certain transformations of the flow field. Sandberg and Michelassi [52] made significant strides in establishing this relationship through the application of genetic programming techniques. They proposed a truncated expansion, focusing solely on the leading terms of the function to simplify the analysis, while considering only the first two invariants of the flow.

In a preliminary phase of this research, the non-linear models developed by Zhao et al. [69] were evaluated in the context of the periodic hill test case [68]. The results from these evaluations demonstrated that the coefficients α_k exhibited minimal variation across the domain, remaining almost constant, with only slight adjustments induced by changes in the invariants. This finding prompted the current investigation into the practicality of employing constant coefficients α_k in the model. Utilizing constant values for α_k is advantageous as it significantly enhances the robustness of subsequent Reynolds-Averaged Navier-Stokes (RANS) simulations. By doing so, the methodology effectively mitigates the risk of generating non-physical values during transient phases of predictive simulations, ensuring a more stable and reliable modeling process.

6.5.1 Constant and linear correction

This section explores two methods for correcting the Reynolds stress tensor: one based on constant coefficients, and the other utilizing linear functions. Both approaches leverage high-fidelity data to derive the necessary coefficients and functions, ensuring that the corrections accurately reflect the complex behavior of turbulent flows. By incorporating these advanced data-driven techniques, the models aim to improve the predictive capabilities of turbulence simulations while maintaining physical consistency and stability.

6.5.1.1 Constant coefficients correction

The optimization of the coefficients α_k is a crucial step in the constant coefficient method. By selecting constant values for these coefficients, we simplify the computational model while maintaining accuracy in turbulence predictions. The optimization is performed by minimizing the L_2 norm of the error between the Reynolds stress tensor predicted by the non-linear model and the high-fidelity Reynolds stress tensor obtained from Direct Numerical Simulation (DNS) data. This error term ensures that the non-linear model accurately represents the turbulent flow features as captured by DNS.

In addition to minimizing the error, a penalization term is included in the objective function to enforce realizability conditions on the Reynolds stress tensor. These conditions are critical to ensure that the computed Reynolds stresses remain physically valid. Specifically, the Reynolds stress tensor must be symmetric and negative semi-definite, which corresponds to physically meaningful turbulence quantities such as ensuring that the turbulent kinetic energy is non-negative. The penalization term, g, measures the violation of these conditions, and the constant

η controls the balance between minimizing the error and enforcing the realizability constraints. The complete objective function for the optimization is given by:

$$f(\alpha_1, \alpha_2, \alpha_3) = ||\tau^{DNS} - \tau^{Approx}||_2 + \eta g, \tag{6.10}$$

where :

$$\begin{cases} \tau_{ij}^{DNS} = -\rho\overline{u_i' u_j'} \\ \tau_{ij}^{Approx} = 2\mu_t S_{ij} - \frac{2}{3}\rho k \delta_{ij} - \rho k \sum_{k=1}^{3} \alpha_k V_{ij}^k \\ g = \sum_{i=1}^{2} \max(\lambda_i, 0)^2, \quad \lambda = \text{eigen value of } \tau^{Approx} \\ \eta = 1000 \end{cases}$$

where τ^{DNS} represents the Reynolds stress tensor from DNS, τ^{Approx} is the corresponding tensor from the Boussinesq assumption computed from the DNS data and corrected with the constant coefficient method. g is the realizability penalty function. The constant η is determined through a trial-and-error procedure to ensure that the magnitude of the error term and the penalization term are comparable. This ensures that the optimization does not excessively prioritize error minimization at the cost of violating physical constraints, or vice versa.

The advantage of using constant coefficients α_k lies in the robustness and simplicity it introduces correction model. In previous approaches, where the coefficients were allowed to vary across the flow domain, issues such as non-physical oscillations and numerical instabilities were common, especially during transients in predictive simulations. These instabilities often led to divergence in the solution or unrealistic flow features, rendering the model unsuitable for practical simulations. By keeping the coefficients constant, we avoid such oscillations, leading to a more stable and reliable computational model. Moreover, constant coefficients reduce the computational cost compared to more complex models that require local variations in the coefficients, making this approach highly suitable for large-scale industrial applications.

Despite the simplicity of the constant coefficient method, it has been shown to yield significant improvements in the accuracy of Reynolds stress predictions, particularly in flows with strong anisotropy or separation. This is especially true when applied to challenging flow configurations such as the periodic hill or the NASA hump test case, both of which exhibit complex separation and recirculation zones. In these cases, the standard Shear Stress Transport (SST) model, without non-linear corrections, tends to overestimate the size of the recirculation region and underpredict velocity gradients in the near-wall region. The introduction of non-linear correc-

tions with constant coefficients not only reduces the overall error in the Reynolds stress tensor but also improves the predictive accuracy of flow features such as the separation point, reattachment length, and turbulence intensity.

Furthermore, the constant coefficient method has proven to be more robust in out-of-sample tests, where the model is applied to flow configurations not included in the DNS training dataset. Since the coefficients are constant, the model retains its stability and accuracy across a variety of flow regimes without the need for re-tuning or recalibration. This robustness is critical for practical engineering applications, where the flow conditions can vary significantly between different geometries and operating points.

From this method, we extracted two sets of coefficients. These were calculated using a subset of the DNS data from Xiao et al [68]. The first set was obtained by minimizing the problem without penalization, while the second set included penalization, allowing us to assess the utility of the constraint. The coefficients obtained without the constraint were calculated using a standard BFGS [6] optimization algorithm. For the second set, the minimization problem appeared to have local minima, so we adopted a genetic algorithm [27] approach, which provided better results than classical optimization algorithms (BFGS [6], Nelder-Mead [38], Powell[46]).

For the first method, where penalization ($\eta = 0$) is not active, we obtained:

$$\alpha_1 = -8.484, \qquad \alpha_2 = -5.956, \qquad \alpha_3 = 0.021 \tag{6.11}$$

When the penalization term is active ($\lambda = 1000$) the following coefficients are obtained:

$$\alpha_1 = -5.456, \qquad \alpha_2 = -19.999, \qquad \alpha_3 = 19.999 \tag{6.12}$$

6.5.1.2 Linear correction

In an effort to further improve the accuracy of Reynolds stress predictions in turbulent flow simulations, a linear coefficient correction method has been investigated. This approach differs from the constant coefficient method by allowing the correction coefficients to vary as linear functions of the invariants of the strain rate tensor. These linear coefficients are aimed at capturing more detailed variations in the flow that cannot be represented with constant values.

In this method, the correction coefficients α_k are expressed as linear functions of the first two invariants, I_1 and I_2, of the strain rate tensor. This results in the following expression for the corrected Reynolds stress tensor:

$$\tau_{ij}^{RANS} = 2\mu_t S_{ij} - \frac{2}{3}\rho k \delta_{ij} - \rho k \sum_{k=1}^{3} (\alpha_k^0 + \alpha_k^1 I_1 + \alpha_k^2 I_2) V_{ij}^k, \tag{6.13}$$

where V_{ij}^k are the basis tensors, and the coefficients α_k^0, α_k^1, and α_k^2 are determined by optimizing the objective function:

$$f(\alpha_1, \alpha_2, \alpha_3) = ||\tau^{DNS} - \tau^{Approx}||_2 + \eta g, \tag{6.14}$$

Where :

$$\begin{cases} \alpha_i = (\alpha_i^0, \alpha_i^1, \alpha_i^2) \\ \tau_{ij}^{DNS} = -\rho\overline{u_i' u_j'} \\ \tau_{ij}^{Approx} = 2\mu_t S_{ij} - \dfrac{2}{3}\rho k \delta_{ij} - \rho k \displaystyle\sum_{k=1}^{3} (\alpha_k^0 + \alpha_k^1 I_1 + \alpha_k^2 I_2) V_{ij}^k \\ g = \displaystyle\sum_{i=1}^{2} \max(\lambda_i, 0)^2, \quad \lambda = \text{eigen value of } \tau^{Approx} \\ \eta = 1000 \end{cases}$$

with g representing a realizability penalty to ensure that the Reynolds stresses remain physically meaningful. The linear functions allow the coefficients α_k to adapt locally to the flow conditions by incorporating the influence of the flow invariants, thereby improving the model's capacity to capture complex flow features such as anisotropy and separation.

To find the optimal values of α_k^0, α_k^1, and α_k^2, a genetic algorithm is employed to avoid local minima that might otherwise be encountered with more traditional optimization methods such as BFGS. This approach proves beneficial as it ensures that the correction terms maintain a smooth variation across the flow domain and prevent numerical instabilities. The use of linear functions for the coefficients offers a more flexible and accurate correction compared to the constant coefficient approach, especially for flows with more pronounced non-linearities.

The following coefficients were obtained :

$$\begin{cases} \alpha_1 = (-6.568, -18.128, -93.892) \\ \alpha_2 = (-34.643, 15.834, 49.487) \\ \alpha_3 = (53.297, 67.308, 8.101) \end{cases} \tag{6.15}$$

These data were derived from the DNS data, enabling us to calculate the error associated with this correction by comparing the different Reynolds stress tensors with those obtained from the DNS. From these graphs 6.23, we can see that when the nonlinear correction is applied, the

average error is reduced compared to the initial Boussinesq approximation. However, there is a limited region near the outlet where the correction results in a larger error than the Boussinesq approximation without correction.

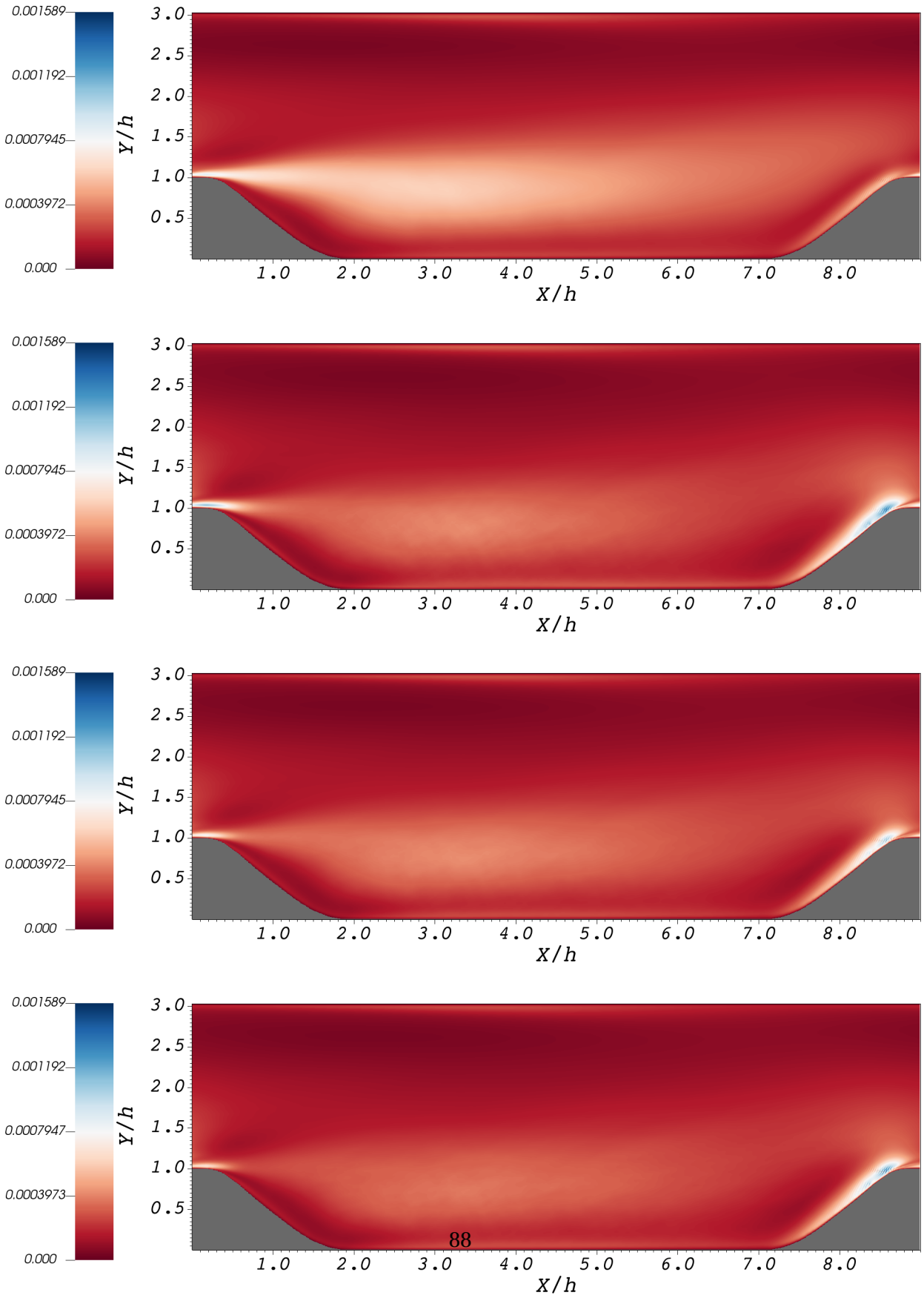


Figure 6.23: Error on the Reynolds stress tensor. From top to bottom: Boussinesq assumption, Boussinesq assumption corrected with constant α_k found without the penalization term, Boussinesq assumption corrected with constant α_k found with the penalization term, Boussinesq assumption with linear $\alpha_k(I_1, I_2)$ found with the penalization term

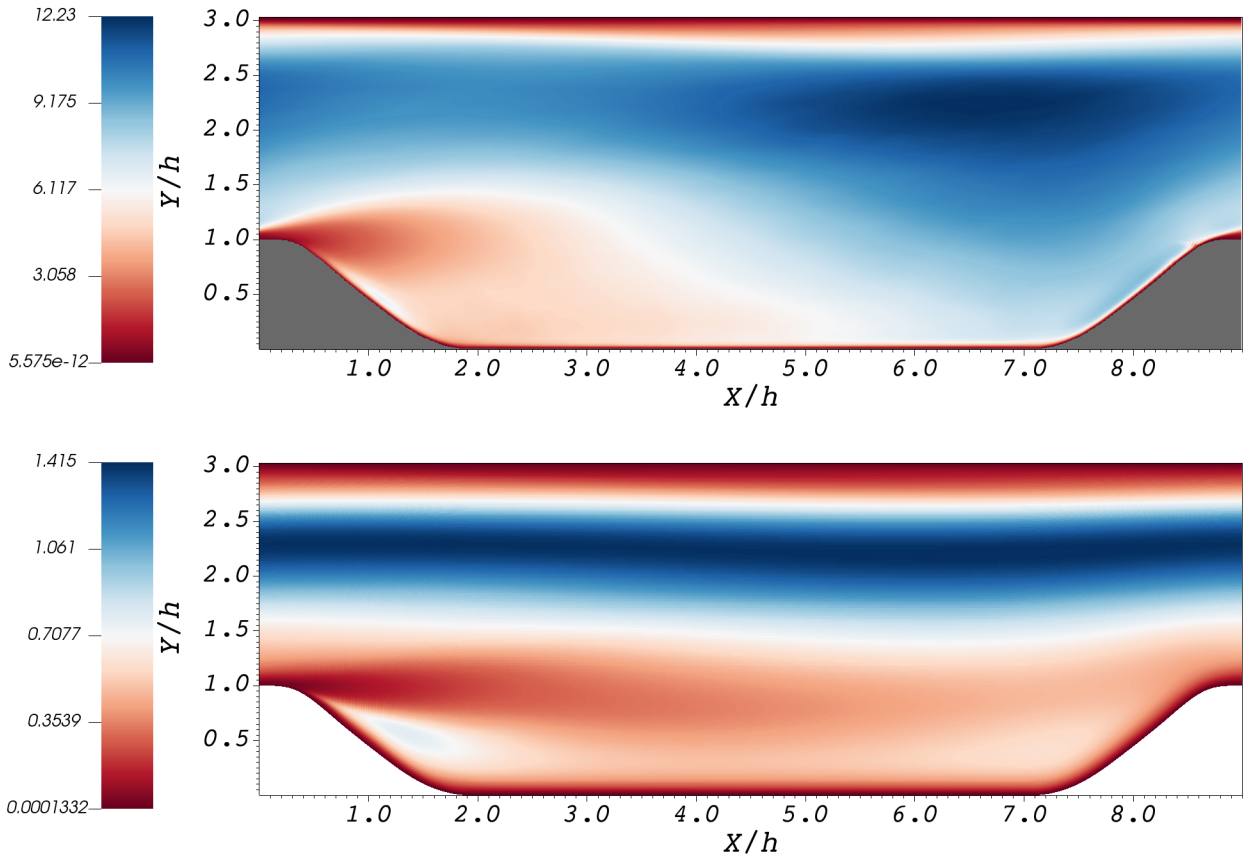


Figure 6.24: Characteristic time in DNS database (top) and SST model (bottom)

6.5.2 Characteristic time correction

The non-linear model, trained on DNS data, can be integrated and tested within the RANS solver. Initial tests conducted on the same geometries used during the training phase revealed a significant impact of the non-linear terms on the RANS results. Notably, these non-linear terms introduced an excessive correction, which will be further discussed in the following section in the context of the periodic hill test case. This over-correction can be attributed to the fact that the non-linear model was trained using averaged DNS fields, which are not perfectly replicated in the RANS simulations. A major source of discrepancy lies in the characteristic time scale, $\tau = 1/\omega$, which is used to normalize the tensor basis. This uncertainty stems from the definition of specific dissipation ω, as it relates to turbulence dissipation ϵ through dimensional analysis. However, dimensional analysis does not provide a concrete scaling factor, leading to differences between the characteristic time derived from DNS and that estimated from the ω field in RANS.

To address this issue, Parneix [41] proposed integrating the RANS ω equation up to a steady state while freezing all other flow variables to their averaged DNS values. The resulting ω field could

then be used to estimate a turbulence characteristic time consistent with the RANS model's non-linear terms. However, in this work, an alternative approach is adopted. Specifically, the turbulence characteristic time, $\tau = 1/\omega$, is corrected by introducing a factor calculated via an Artificial Neural Network (ANN), denoted K_{ANN}:

$$\tilde{\tau} = \tau K_{ANN}(x_1, x_2) \tag{6.16}$$

The inputs to this neural network are defined as follows:

$$x_1 = \min(2, \rho\sqrt{k}d/\mu), \qquad x_2 = \frac{\tau_v}{\tau_v + \tau_\omega + 1e-10} \tag{6.17}$$

The first input, x_1, represents the Reynolds number based on the wall distance and is capped at two, following [60]. The second input, x_2, measures the discrepancy between two different definitions of the characteristic time. These inputs are restricted to a predefined range to ensure smooth behavior.

After preliminary investigations, a small neural network architecture was selected, consisting of two inputs, a hidden layer with two neurons, and a single output providing the corrected characteristic time. The ANN's weights and biases are determined by solving an optimization problem that incorporates the RANS solver within the optimization loop. The optimization goal is to minimize the following objective function:

$$f = \int_{\mathrm{CS}} (u_{RANS} - u_{DNS})^2 dy + \beta Y \tag{6.18}$$

Here, u_{RANS} and u_{DNS} represent the velocity profiles at various control stations (CS) within the computational domain. The penalization term Y accounts for realizability constraints and is calculated as the integral of the realizability error E_{real} over the entire domain Ω:

$$Y = \int_{\Omega} E_{\mathrm{real}} d\Omega \tag{6.19}$$

The realizability error, E_{real}, is given by:

$$E_{\mathrm{real}} = \frac{|\tau - \tilde{\tau}|}{\rho_b u_b^2} \tag{6.20}$$

where τ is the Reynolds stress tensor from the RANS model, and $\tilde{\tau}$ is its modified version that satisfies the realizability conditions. The Frobenius norm is used to compute the difference between the two tensors, and the result is normalized by a reference pressure based on bulk density ρ_b and bulk velocity u_b. The realizable tensor $\tilde{\tau}$ is computed as follows:

$$\tilde{\tau}_{ij} = \min(0, \tau_{ij}) \quad \text{if} \quad i = j \tag{6.21}$$

$$\tilde{\tau}_{ij} = \begin{cases} sign(\tau_{ij})\sqrt{\tilde{\tau}_{ij}\tilde{\tau}_{ij}} & \text{if} \quad \tau_{ij}^2 > \tilde{\tau}_{ii}\tilde{\tau}_{jj} \\ \tau_{ij} & \text{if} \quad \tau_{ij}^2 \leq \tilde{\tau}_{ii}\tilde{\tau}_{jj} \end{cases} \qquad \text{if} \quad i \neq j \tag{6.22}$$

The penalization constant β is chosen such that the magnitude of the error in the velocity profile and the penalization term Y are of the same order. Initially, this is done by evaluating these terms at the beginning of the optimization process and setting β as:

$$\beta_0 = \frac{\left[\int_{\mathrm{CS}} (u_{RANS} - u_{DNS})^2, dy\right]_0}{Y_0} \tag{6.23}$$

The neural network consists of a single hidden layer with two neurons. The ReLU function is used as the activation function for the hidden layer, while the sigmoid function is employed for the output layer.

The optimization problem is solved using a conjugate gradient descent method [26], and iterations are halted when the gradient becomes negligible. The training is performed using a Fortran 90 code integrated into the CFD code, which is also written in Fortran 90. Once the neural network was trained, using this method resulted in a negligible performance decrease due to the small size of the network (< 10%).

Including the RANS solver in the optimization loop helps to improve the robustness of the model by identifying any numerical instabilities or stiffness in the non-linear correction. Indeed, a preliminary study was conducted by training the same network offline using DNS data and RANS simulation data. However, when this model was integrated into the CFD code, instabilities emerged in the solution.

This optimization approach allows the ANN to correct for various error sources, not just those related to the turbulence characteristic time.

Finally, by directly optimizing the ANN coefficients, we obtain a data-driven model that can be immediately employed in RANS simulations. Since the RANS solver is integrated into the opti-

mization loop, the resulting corrections are inherently compatible with the model. This strategy aligns with the approach outlined in [18], where ANN weights were optimized to identify a correlation for a closure model.

6.5.3 RANS solver

In the next section, we will present and discuss results obtained from the RANS equations solved with various turbulence closure models. But before that, let's discuss the RANS solver used to obtain these results.

The governing equations are discretized using a finite volume approach, implemented within a research code designed for compressible turbulent flows. For this study, the flow field configurations include an incompressible case (periodic hill test case) and a case with minimal compressibility (NASA hump test case). To simulate incompressibility, the Mach number is set to 0.1, sufficiently low to minimize compressibility effects without requiring preconditioning methods specific to low Mach number flows.

The computational domain is discretized using a structured mesh generated with the open-source tool Gmsh [23]. Parallelization is handled through the DMPlex class from the PETSc library [2, 4, 3]. Spatial discretization achieves second-order accuracy and applies the Barth-Jespersen limiter [5] to control oscillations. Convective fluxes are computed using a hybrid approach [16] that blends the local Lax-Friedrichs (Rusanov) flux [51] with an upwind flux based on an approximate Riemann solver [39].

To compute diffusive fluxes, gradients are evaluated through a weighted least squares method. Temporal integration of the governing equations is achieved using a linearized implicit Euler scheme, where the resulting linear system is solved with the GMRES algorithm and preconditioned by an Additive-Schwarz method.

6.5.4 Results

In this section, we will present the results of various tests that were conducted. Each test corresponds to the addition of a specific feature of the method, allowing us to clearly identify the contribution of each element. We will perform tests with and without the different penalizations, with and without the correction of the characteristic time, as well as using the two different approaches for calculating the coefficients α_k.

For the sake of clarity, we will describe the methods in the following summary table:

- Model A: The characteristic time was not adjusted, and no penalization was applied in determining the constant coefficients α_k.
- Model B: The characteristic time was adjusted by the ANN without the penalization, and no penalization was included in the determination of the constant coefficients.

Model	$\tilde{\tau}/\tau$	η	β	Coefficients α_k
A	1	0	-	Constant (see eq: 6.11)
B	$K_{ANN}(x_1, x_2)$	0	β_0	Constant (see eq: 6.11)
C	$K_{ANN}(x_1, x_2)$	1000	β_0	Constant (see eq: 6.12)
D	$K_{ANN}(x_1, x_2)$	1000	β_0	$\alpha_k = \alpha_k^0 + \alpha_k^1 I_1 + \alpha_k^2 I_2$ (see eq: 6.15)

Table 6.1: Properties of investigated non-linear corrections

- Model C: The characteristic time was corrected by the ANN with penalization and the constant coefficients were found with the penalization.
- Model D: The characteristic time was corrected by the ANN with penalization, and a linear correction was applied in order to find the coefficient.

6.5.4.1 Results on periodic hill

The axial velocity field (normalized by the bulk velocity) obtained from the DNS, the original SST model, and model D is presented in Figure 6.25. It is evident that the original SST model tends to overpredict the size of the recirculation region. Additionally, the turbulent kinetic energy, normalized by the square of the bulk velocity, is illustrated in Figure 6.26 for the same models.

A more quantitative comparison can be conducted by plotting the axial velocity profiles at various control sections. In particular, the axial velocity profiles at $x/h = 2$ and $x/h = 4$ are presented in Figures 6.27 and 6.28, respectively.

It can be observed that model A, which was trained on the DNS data but does not incorporate the correction for the characteristic time, tends to underpredict the velocity profile. In contrast, models B, C, and D, which include the RANS characteristic time correction, yield similar results that align more closely with the DNS data.

Table 6.2 displays the velocity errors for the different models at $x/h = 2$ and $x/h = 4$. This measure of error is computed as:

$$\epsilon = \int_{CS} (U_{DNS} - U_{model})^2 dy$$

Finally, a measure of the error regarding the realizability conditions is presented in Figure 6.29. The error is computed according to Eq. 6.20.

It is important to note that the original SST model introduces negligible violations of the realizability conditions in this test case; these violations are confined to a very thin layer near the wall, which is not visible in the figure. However, when a non-linear correction is applied to the Boussinesq assumption without any control over the realizability conditions, significant

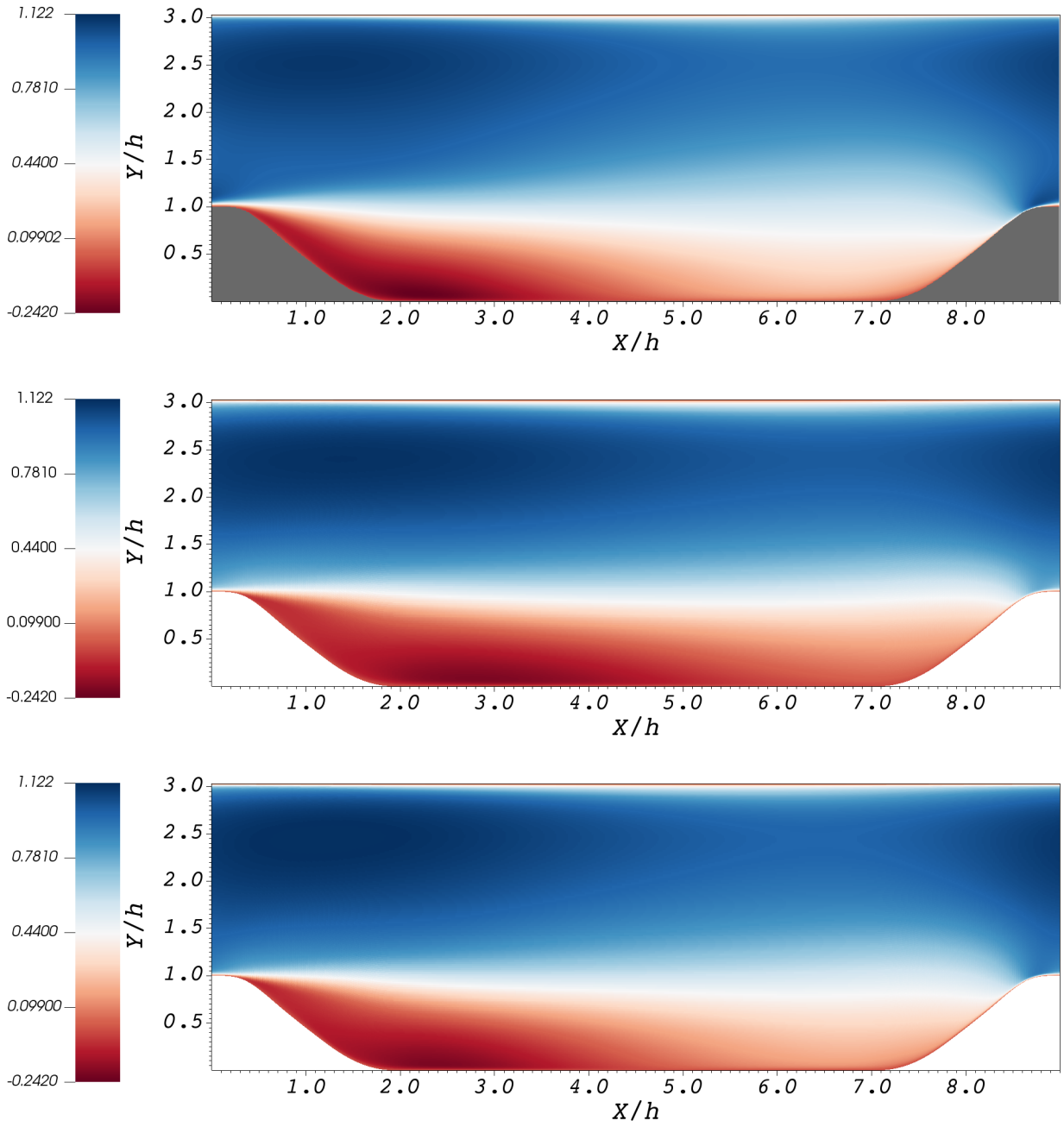


Figure 6.25: Axial velocity field. From top to bottom DNS data, RANS SST model, RANS SST+model D

violations can occur, as evidenced by the results from model A.

For this reason, it is crucial to incorporate a penalization term addressing the realizability con-

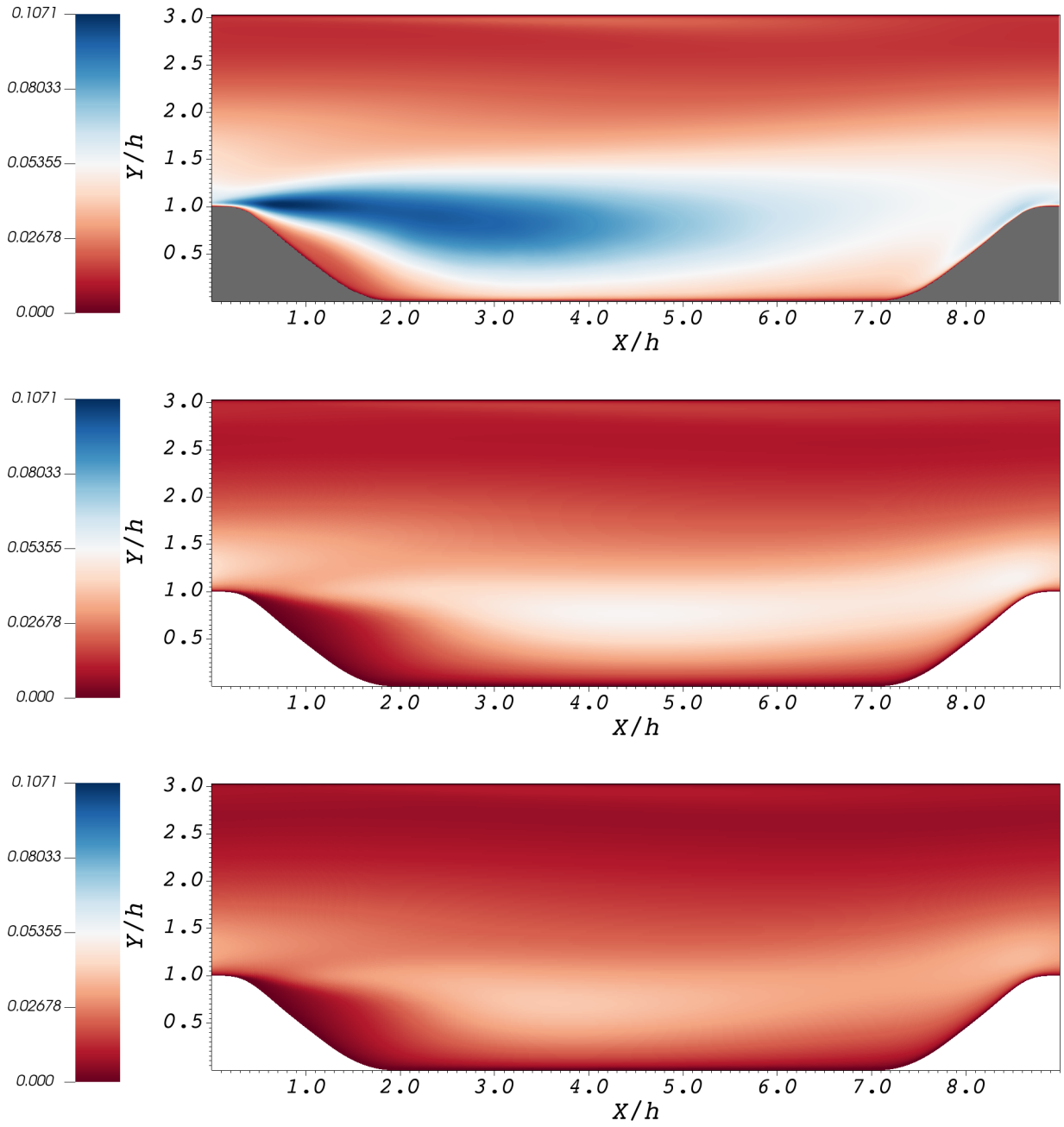


Figure 6.26: Turbulent kinetic energy normalized with the square of the bulk velocity. From top to bottom DNS data, RANS SST model, RANS SST + model D.

ditions. The results from models C and D demonstrate this approach, characterized by very small violations of the realizability conditions, while also providing improved predictions of the velocity profile compared to the original SST model.

Model	ϵ at $x/h = 2$	ϵ at $x/h = 4$
SST	$9.8007e^{-3}$	$2.9351e^{-2}$
A	$5.4322e^{-3}$	$1.2499e^{-2}$
B	$2.0684e^{-3}$	$3.1875e^{-3}$
C	$2.0367e^{-3}$	$2.7958e^{-3}$
D	$2.5217e^{-3}$	$3.9705e^{-3}$

Table 6.2: Velocity error on the 2 profiles for each model

The results regarding the axial velocity distribution at $x/h = 2$ and $x/h = 4$ are presented in Figures 6.27 and 6.28. The plots include the DNS data from the aforementioned database [68]. It is noteworthy that the non-linear model D, trained on the DNS data and utilizing the characteristic time correction provided by the ANN, demonstrates commendable performance by predicting a velocity profile that aligns more closely with the DNS data compared to the original SST model.

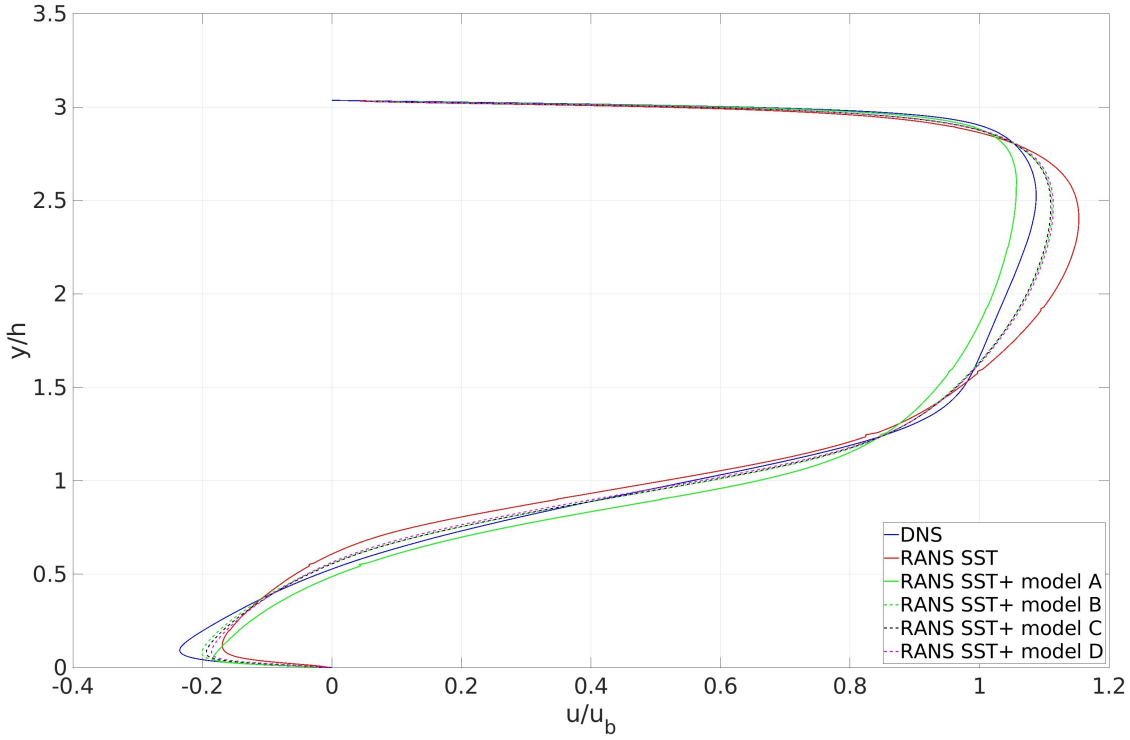


Figure 6.27: Axial velocity profile at $x/h = 2$ for baseline geometry

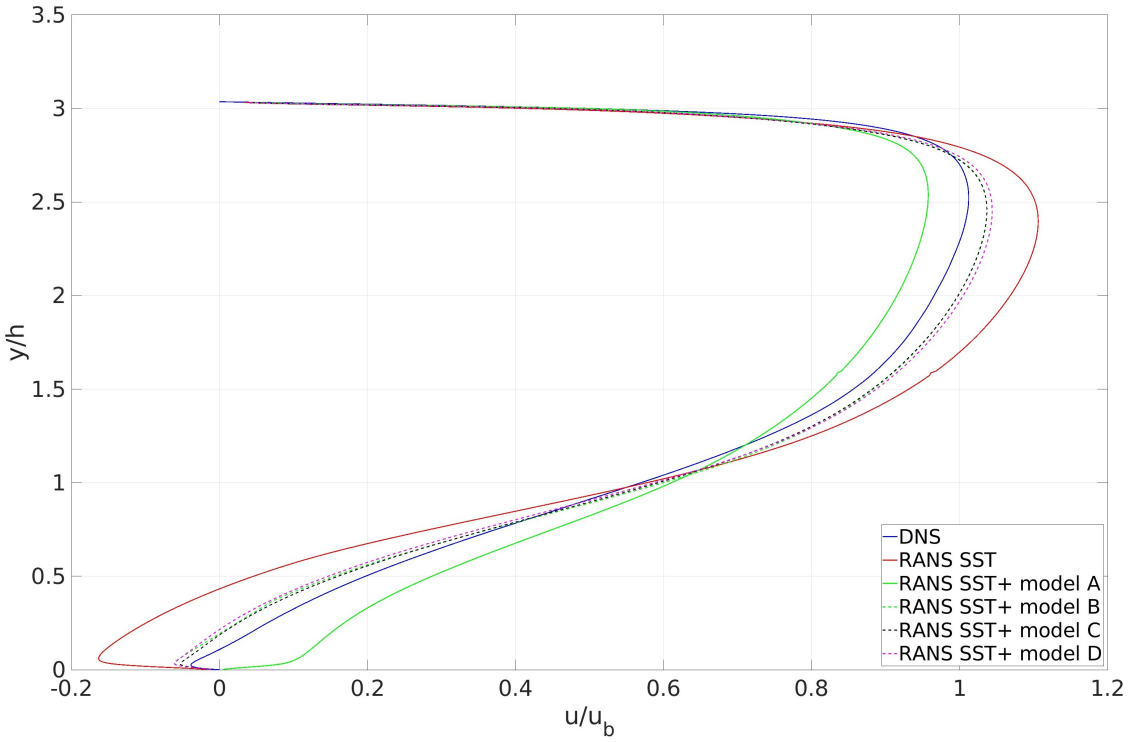
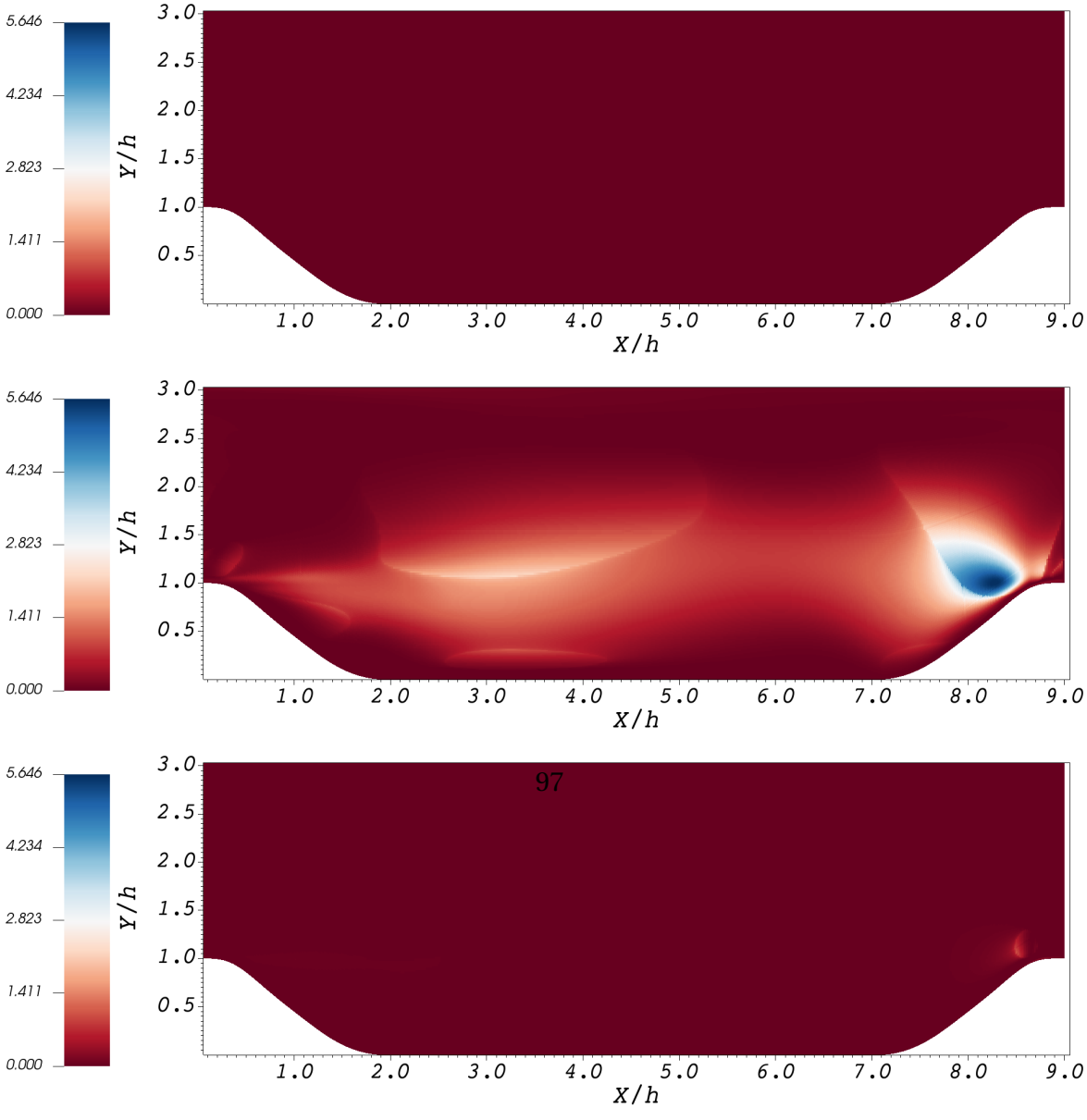


Figure 6.28: Axial velocity profile at $x/h = 4$ for baseline geometry



The neural network employed to correct the characteristic time was trained on the baseline geometry. In the following part of this section, we will present predictive results for other hill geometries. As in previous analyses, we will compare the DNS, RANS SST, and model D by examining the axial velocity profile at two control stations, $x/h = 2$ and $x/h = 4$.

The first test is conducted on the geometry characterized by the values $\alpha = 1.5$, $\beta = 5.142$, and $L_y/h = 2.024$. The velocity profiles generated by the various models are displayed in Figure 6.30 for the control point $x/h = 2$ and in Figure 6.31 for the control point $x/h = 4$. These figures clearly illustrate that model D represents a significant improvement over the RANS model.

To facilitate a more quantitative comparison, the errors calculated for this test case are summarized in Table 6.3.

Model	ϵ at $x/h = 2$	ϵ at $x/h = 4$
SST	$2.3793e^{-2}$	$8.0659e^{-2}$
D	$3.9749e^{-2}$	$1.6983e^{-2}$

Table 6.3: Velocity error on two control stations for SST model and SST+model D.

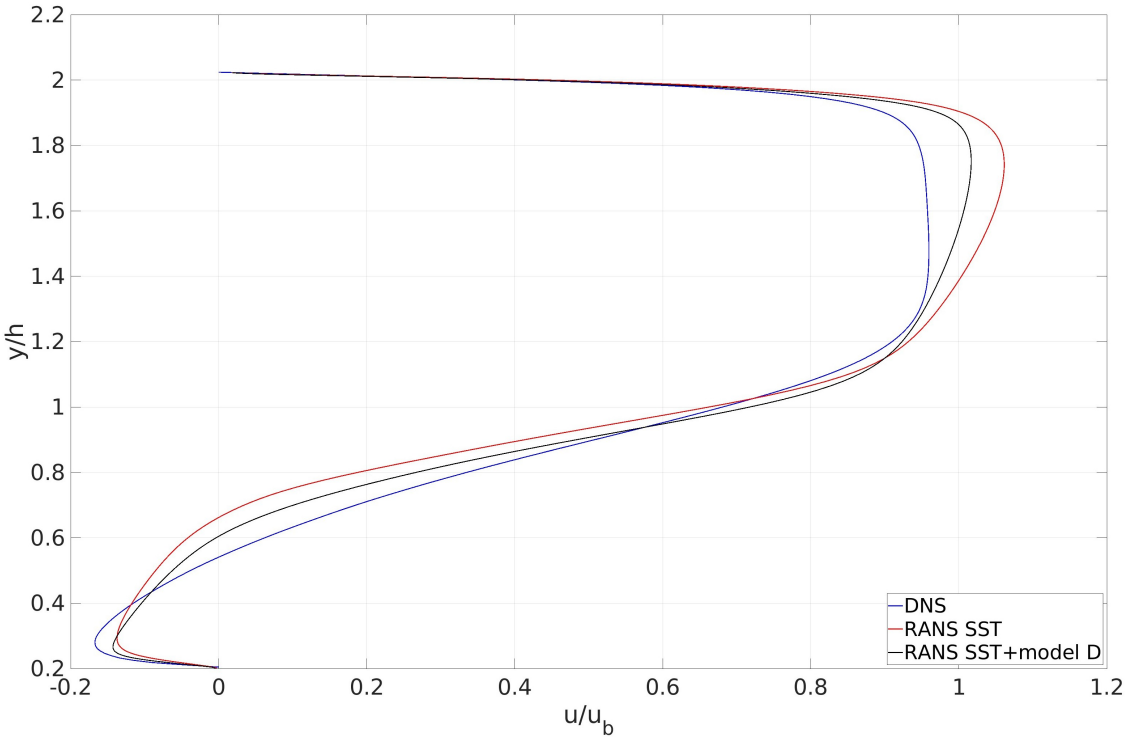


Figure 6.30: Axial velocity profile at $x/h = 2$ for Test 1 geometry

The second test is conducted on a geometry characterized by the values $\alpha = 1.5$, $\beta = 8.142$, and $L_y/h = 3.036$. The velocity profiles for the three models applied to this geometry are presented in Figure 6.32 for the control point $x/h = 2$ and in Figure 6.33 for the control point $x/h = 4$.

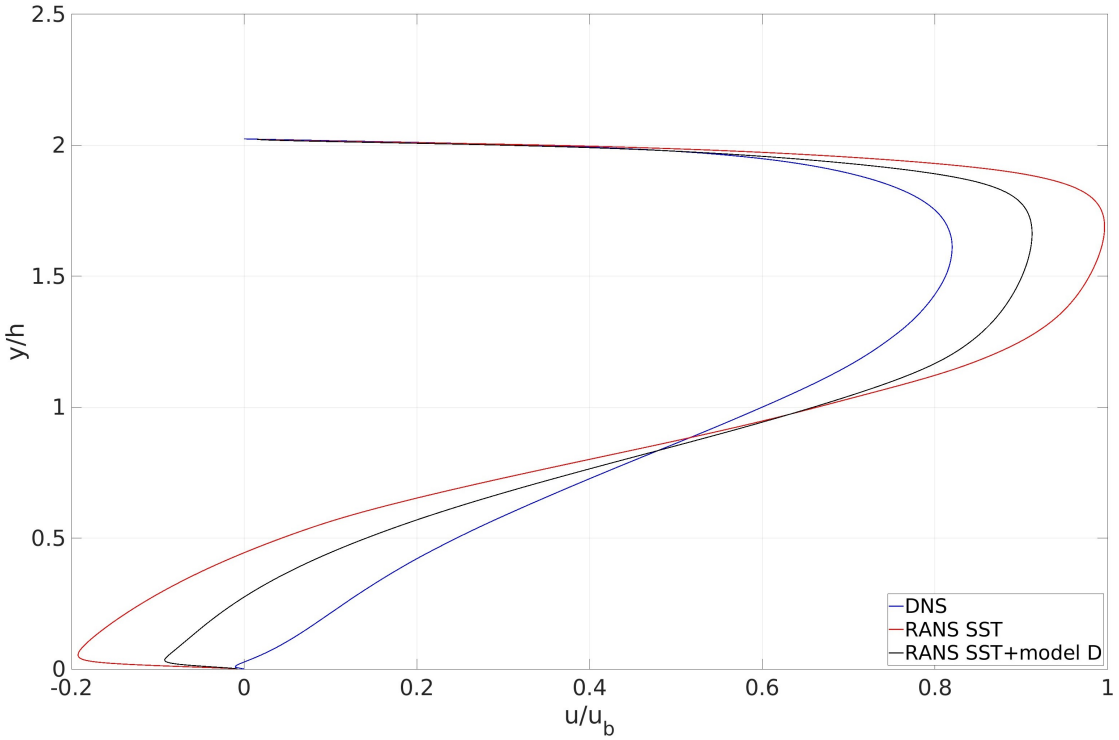


Figure 6.31: Axial velocity profile at $x/h = 4$ for Test 1 geometry

Both graphs indicate an improvement with the use of model D. However, this enhancement is less pronounced compared to the baseline geometry and the results from test 1.

Model	ϵ at $x/h = 2$	ϵ at $x/h = 4$
SST	$2.2205e^{-1}$	$2.9798e^{-2}$
SST+Model D	$1.7921e^{-1}$	$1.8650e^{-1}$

Table 6.4: Velocity error on the 2 profiles for each model (first test)

6.5.4.2 Results on NASA hump

The original SST model and model D are evaluated using the 2D NASA wall-mounted hump separated flow validation case [25]. This test case was designed to assess the capability of turbulence models in predicting 2D separation from a smooth body and the subsequent reattachment. The working conditions and geometry can be found on the NASA Turbulence Modeling Resource website [50]. The far-field Mach number is set at 0.1, while the Reynolds number based on the chord of the hump is 936,000.

The initial experimental setup included a plenum for flow control; however, the present study focuses on the scenario without flow control, thus excluding the plenum from the computational domain. Simulations are conducted on a structured mesh consisting of 409x109 cells,

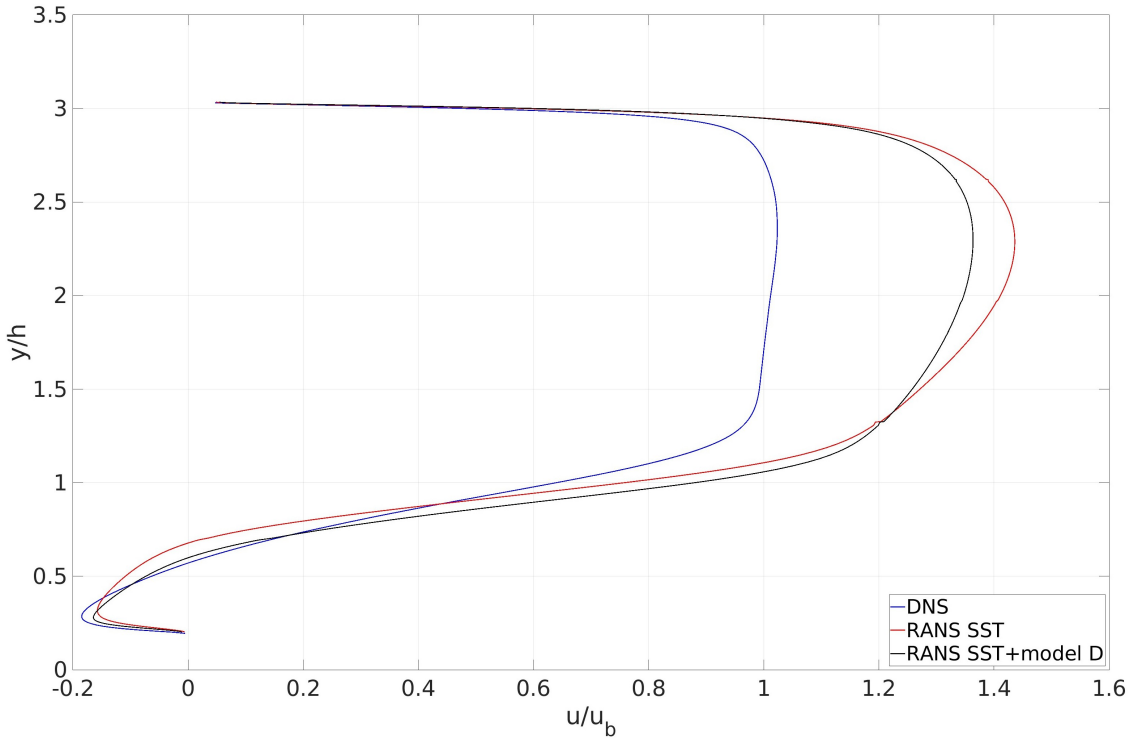


Figure 6.32: Axial velocity profile at $x/h = 2$ for Test 2 geometry

sourced from the NASA Turbulence Modeling Resource website [50]. The Mach field produced by the SST model is illustrated in Figure 6.34, clearly depicting the large separation and reattachment phenomena. While the Mach field provides a qualitative assessment, it is challenging to evaluate the prediction quality based solely on this visualization.

A more objective comparison between the SST model and SST+model D is achieved by analyzing the wall pressure coefficient from both models against the experimental results, as shown in Figure 6.35. This comparison reveals that model D outperforms the original SST model. The improvement is further substantiated by the prediction of the reattachment position, detailed in Table 6.5. The reattachment position, identified by observing a sign change in the wall friction coefficient, demonstrates significant enhancement with the introduction of model D.

Source	Axial position for reattachmet (x/c)
Experiment	1.10
RANS SST	1.26
RANS SST + model D	1.10

Table 6.5: Position of reattachment point

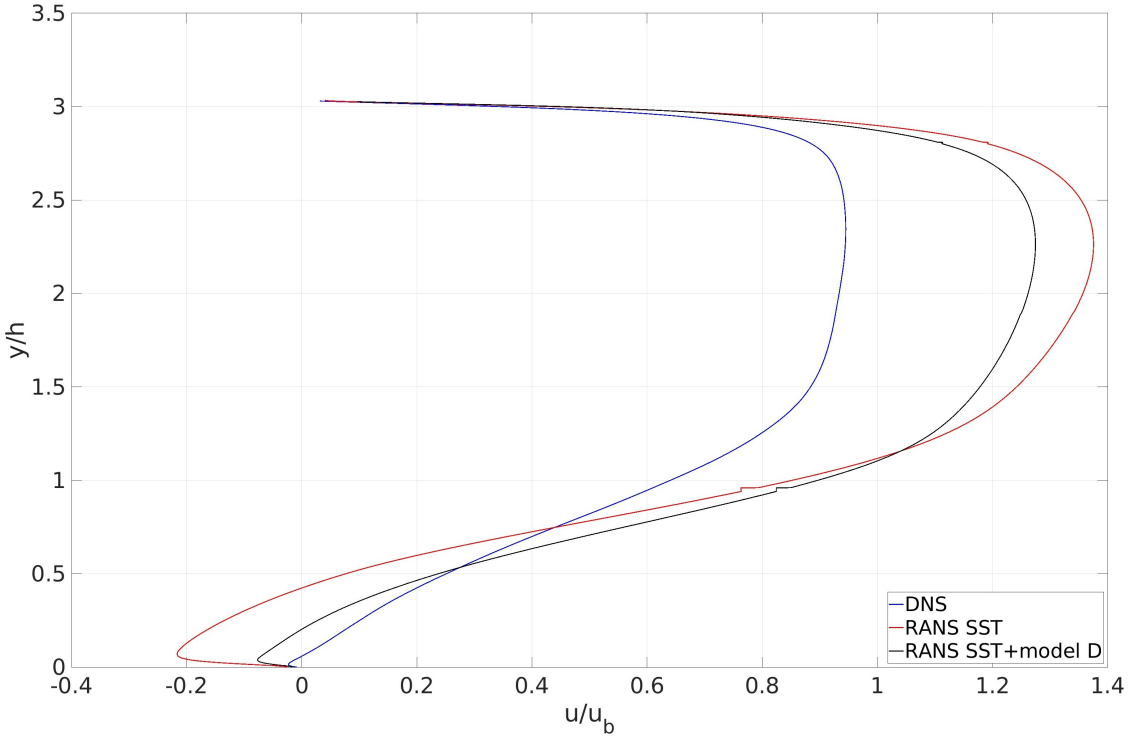


Figure 6.33: Axial velocity profile at $x/h = 4$ for Test 2 geometry

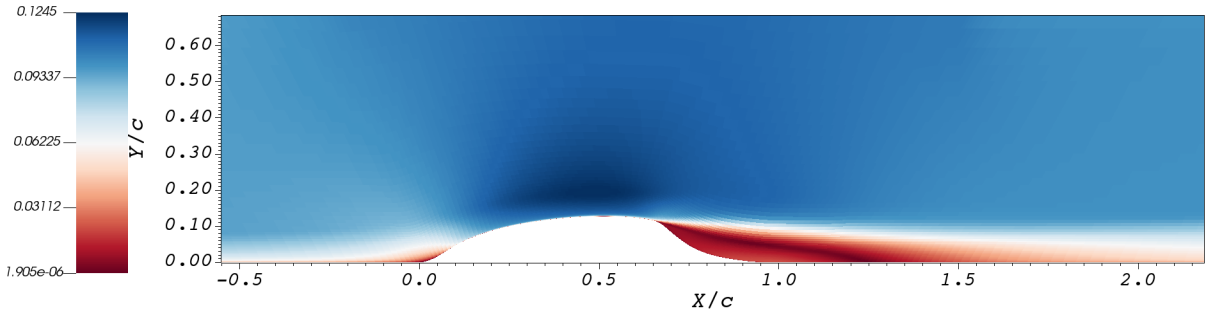


Figure 6.34: Detail of the Mach number field for the NASA hump case with SST model.

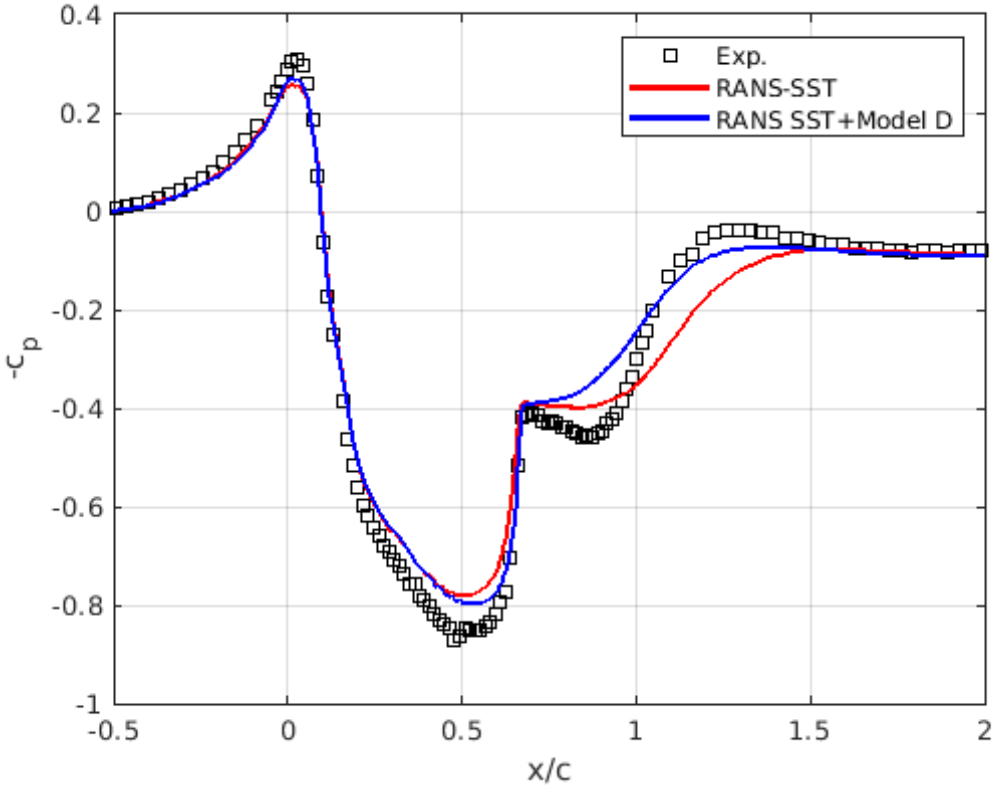


Figure 6.35: Wall pressure coefficient on the NASA hump case.

CHAPTER 7

Conclusion

Contents

7.1 Results and interpretation 103

7.2 Perspectives and future work 104

7.1 Results and interpretation

In this study, we investigated methods for correcting the Reynolds stress tensor using Pope's framework and DNS data. The first two approaches employed neural networks—specifically, a simple Multi-Layer Perceptron (MLP) and a Generative Adversarial Network (GAN). Both of these methods followed a two-step process: (1) training the network coefficients based on DNS data, and (2) integrating the trained networks into a RANS solver. While the training phase produced promising results, the second step presented significant challenges. Instabilities arose during the implementation of the networks within the RANS model, primarily due to discrepancies in the characteristic time scales between DNS and RANS simulations. These issues prevented the development of a more robust RANS model, as the instabilities undermined the accuracy and consistency of the simulations.

To address these challenges, we introduced a third approach, which involved two stages: (i) data assimilation was used to derive a non-linear correction to the Boussinesq approxima-

tion, and (ii) a neural network was trained to adjust the characteristic time directly within the RANS framework to minimize prediction errors. This refined method, combining a non-linear Reynolds stress correction with a neural network-based adjustment for the characteristic time, was then applied to out-of-sample simulations for validation.

All these methods were tested across various periodic hill configurations at a constant Reynolds number, as well as the NASA hump test case. Both scenarios exhibit pronounced flow separation, making them ideal benchmarks for evaluating the performance of the enhanced turbulence model in simulating fully developed turbulent flows.

Results from these out-of-sample simulations revealed a notable improvement in velocity profiles when compared to the standard SST model, highlighting the effectiveness of the new correction. A critical aspect of this study involved the inclusion of realizability conditions during the training process of the data-augmented model. Extensive testing showed that neglecting these constraints could result in severe violations during the prediction phase. To prevent such issues, we incorporated realizability constraints through penalization in both the data assimilation and neural network training steps.

By integrating these constraints, the proposed approach ensures that the predictions remain valid and physically meaningful, leading to more reliable and consistent outcomes in RANS simulations.

Despite these encouraging results, some challenges persist, particularly in terms of numerical stability. The coupling between the Reynolds stress corrections and the numerical integration scheme requires careful consideration to avoid instabilities and to ensure that the simulations maintain physical realism. Utilizing RANS simulations within the optimization loop to adjust the characteristic time played a crucial role in mitigating these difficulties, but further improvements are needed.

7.2 Perspectives and future work

The primary objective of this thesis was to employ machine learning methods to enhance the Boussinesq assumption used in RANS techniques. The research presented herein raises additional questions and opens up numerous avenues for future exploration.

First, as discussed in this presentation, we faced stability issues when integrating the correction model for the Boussinesq approximation into the RANS framework. This raises important questions about how we can avoid such challenges in future implementations. Stability is a critical aspect of computational fluid dynamics simulations, as instabilities can lead to unreliable results and hinder the practical application of turbulence models.

The literature suggests that some approaches freeze RANS models on specific flow features us-

ing DNS data to ensure consistent characteristic times. This method can provide a controlled environment for model training, but it may limit the model's adaptability to varying geometries and flow conditions. By fixing the RANS model to certain flow features, researchers attempt to maintain stability and ensure that the characteristic times align with those observed in DNS data. However, this approach may not always be feasible or effective, particularly in complex and dynamic flow scenarios.

In light of this, we must consider whether our method for correcting the characteristic time offers a more flexible and robust alternative. Our neural network training demonstrated the potential to achieve good results without relying on DNS data for geometries different from those used during the training phase. This suggests that our approach may provide a pathway to enhance the adaptability of RANS models across a wider range of flow conditions.

Secondly, it is worth exploring the possibility of integrating a data-driven Reynolds stress correction model with a characteristic time correction model, as highlighted in this presentation. Such a combination could offer a more robust and comprehensive approach to turbulence modeling, enhancing the overall performance of RANS simulations.

The data-driven Reynolds stress correction model aims to refine the estimation of turbulent stresses by utilizing information derived from Direct Numerical Simulation (DNS) data. This model allows for a more accurate representation of the complex interactions inherent in turbulent flows. On the other hand, the characteristic time correction model focuses on addressing stability issues associated with RANS simulations by ensuring that the time scales used in the model are appropriate for the flow conditions being simulated.

By merging these two methodologies, we could capitalize on the strengths of each while mitigating their respective limitations. The data-driven Reynolds stress correction could provide a nuanced understanding of turbulence dynamics, capturing intricate flow features that traditional models might overlook. Meanwhile, the characteristic time correction could stabilize the integration of these corrections within the RANS framework, ensuring that the model operates effectively across a variety of flow regimes.

We have trained and tested these methods on flows characterized by relatively low Reynolds numbers. However, an important question arises: how do these methods perform in flows with significantly higher Reynolds numbers?

High Reynolds number flows are often associated with more complex turbulence dynamics, including increased inertial effects and larger-scale turbulent structures. These flows pose unique challenges for turbulence modeling, as the assumptions underlying RANS models may become less applicable. As a result, the performance of correction methods, such as those developed in this study, may differ when applied to these more turbulent regimes.

It is essential to investigate the adaptability and robustness of our data-driven approaches in the context of high Reynolds number flows. One key consideration is whether the training datasets, primarily composed of lower Reynolds number flow characteristics, adequately represent the behaviors observed at higher Reynolds numbers. This mismatch could potentially limit the effectiveness of the correction models, leading to reduced predictive accuracy.

To address this concern, future research should focus on expanding the training datasets to include a diverse range of flow conditions, particularly those relevant to high Reynolds number scenarios. This could involve the use of DNS data from high Reynolds number flows, enabling the models to learn from a more comprehensive array of turbulence behaviors.

In conclusion, while our current methods have shown promise in low Reynolds number flows, the prospect of combining them with additional corrections and adapting them for high Reynolds number scenarios represents an exciting avenue for future research. By addressing these challenges, we can contribute to the development of more robust turbulence models that effectively handle the complexities of turbulent flows across a wide range of applications.

Bibliography

[1] Martín Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein GAN". In: *ArXiv* abs/1701.07875 (2017). URL: https://api.semanticscholar.org/CorpusID:13943041.

[2] Satish Balay et al. "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries". In: *Modern Software Tools in Scientific Computing*. Ed. by E. Arge, A. M. Bruaset, and H. P. Langtangen. Birkhäuser Press, 1997, pp. 163–202.

[3] Satish Balay et al. *PETSc Web page*. https://petsc.org/. 2023. URL: https://petsc.org/.

[4] Satish Balay et al. *PETSc/TAO Users Manual*. Tech. rep. ANL-21/39 - Revision 3.19. Argonne National Laboratory, 2023. DOI: 10.2172/1968587.

[5] Timothy J. Barth and Dennis C. Jespersen. "The design and application of upwind schemes on unstructured meshes". In: 1989. URL: https://api.semanticscholar.org/CorpusID:122107708.

[6] C.G. Broyden. "A new double-rank minimization algorithm". In: *Notices American Mathematical Society* 16 (1969), p. 670.

[7] Haecheon Choi and Parviz Moin. "Grid-point requirements for large eddy simulation: Chapman's estimates revisited". In: *Physics of Fluids* 24 (2012), p. 011702. URL: https://api.semanticscholar.org/CorpusID:122728957.

[8] Corinna Cortes and Vladimir Naumovich Vapnik. "Support-Vector Networks". In: *Machine Learning* 20 (1995), pp. 273–297. URL: https://api.semanticscholar.org/CorpusID:52874011.

[9] Timothy J. Craft, B. E. Launder, and Kazuhiko Suga. "Development and application of a cubic eddy-viscosity model of turbulence". In: *International Journal of Heat and Fluid*

Flow 17 (1996), pp. 108–115. URL: https://api.semanticscholar.org/CorpusID:119918040.

[10] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. "Maximum likelihood from incomplete data via the EM - algorithm plus discussions on the paper". In: 1977. URL: https://api.semanticscholar.org/CorpusID:4193919.

[11] David M. Driver and H. Lee Seegmiller. "Features of a reattaching turbulent shear layer in divergent channel flow". In: *AIAA Journal* 23 (1985), pp. 163–171. URL: https://api.semanticscholar.org/CorpusID:123693266.

[12] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. "Turbulence Modeling in the Age of Data". In: *Annual Review of Fluid Mechanics* (2018). URL: https://api.semanticscholar.org/CorpusID:4683538.

[13] Paul A. Durbin. "Some Recent Developments in Turbulence Closure Modeling". In: *Annual Review of Fluid Mechanics* 50 (2018), pp. 77–103. URL: https://api.semanticscholar.org/CorpusID:126221994.

[14] Paul A. Durbin and Bjørn Anders Pettersson Reif. "Statistical Theory and Modeling for Turbulent Flows". In: 2001. URL: https://api.semanticscholar.org/CorpusID:116956047.

[15] Alexandre Favre. "La Turbulence en mécanique des fluides : bases théoriques et expérimentales, méthodes statistiques". In: 1976. URL: https://api.semanticscholar.org/CorpusID:117976560.

[16] Andrea Ferrero and Domenic D'Ambrosio. "An hybrid numerical flux for supersonic flows with application to rocket nozzles". In: 2020. URL: https://api.semanticscholar.org/CorpusID:229636932.

[17] Andrea Ferrero, Angelo Iollo, and Francesco Larocca. "Field inversion for data-augmented RANS modelling in turbomachinery flows". In: *Computers & Fluids* 201 (2020), p. 104474. URL: https://api.semanticscholar.org/CorpusID:214018817.

[18] Andrea Ferrero, Angelo Iollo, and Francesco Larocca. "RANS CLOSURE APPROXIMATION BY ARTIFICIAL NEURAL NETWORKS". In: *ETC 2019-13th European Turbomachinery Conference on Turbomachinery Fluid Dynamics and Thermodynamics*. 2019. URL: https://api.semanticscholar.org/CorpusID:203636166.

[19] Joel H. Ferziger and Milovan Peric. "Computational methods for fluid dynamics". In: 1996. URL: https://api.semanticscholar.org/CorpusID:63211967.

[20] D.P.G. Foures et al. "A data-assimilation method for reynolds-averaged navier-stokes-driven mean flow reconstruction." In: *Journal of Fluid Mechanics* 759 (2014).

[21] N Srivastava G Hinton and K Swersky. "Coursera: Neural networks for machine learning: Lecture 6 (a)–overview of mini-batch gradient descent". 2014.

[22] M. Germano. "Turbulence: the filtering approach". In: *Journal of Fluid Mechanics* 238 (1992), pp. 325–336. URL: https://api.semanticscholar.org/CorpusID:122744487.

[23] Christophe Geuzaine and Jean-François Remacle. "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities". In: *International Journal for Nu-*

merical Methods in Engineering 79 (2009). URL: https://api.semanticscholar.org/CorpusID:110330097.

[24] Ian J. Goodfellow et al. "Generative adversarial networks". In: *Communications of the ACM* 63 (2014), pp. 139–144. URL: https://api.semanticscholar.org/CorpusID:1033682.

[25] David Greenblatt et al. "Experimental Investigation of Separation Control Part 1: Baseline and Steady Suction". In: *AIAA Journal* 44 (2006), pp. 2820–2830. URL: https://api.semanticscholar.org/CorpusID:120514020.

[26] Magnus R. Hestenes and Eduard Stiefel. "Methods of conjugate gradients for solving linear systems". In: *Journal of research of the National Bureau of Standards* 49 (1952), pp. 409–435. URL: https://api.semanticscholar.org/CorpusID:2207234.

[27] John H. Holland. "Genetic Algorithms and Adaptation". In: 1984. URL: https://api.semanticscholar.org/CorpusID:59661044.

[28] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *ArXiv* abs/1502.03167 (2015). URL: https://api.semanticscholar.org/CorpusID:5808102.

[29] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). URL: https://api.semanticscholar.org/CorpusID:6628106.

[30] Michael A. Leschziner. "Statistical Turbulence Modelling For Fluid Dynamics - Demystified: An Introductory Text For Graduate Engineering Students". In: 2015. URL: https://api.semanticscholar.org/CorpusID:57296262.

[31] J. Ling and J. Templeton. "Evaluation of machine learning algorithms for prediction of regions of high reynolds averaged Navier Stokes uncertainty". In: *Physics of Fluids* 27 (2015).

[32] Julia Ling, Andrew Kurzawski, and Jeremy A. Templeton. "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance". In: *Journal of Fluid Mechanics* 807 (2016), pp. 155–166. URL: https://api.semanticscholar.org/CorpusID:125898515.

[33] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *Bulletin of Mathematical Biology* 52 (1990), pp. 99–115. URL: https://api.semanticscholar.org/CorpusID:15619658.

[34] Florian R. Menter. "Two-equation eddy-viscosity turbulence models for engineering applications". In: *AIAA Journal* 32 (1994), pp. 1598–1605. URL: https://api.semanticscholar.org/CorpusID:120712103.

[35] Florian R. Menter et al. "A Correlation-Based Transition Model Using Local Variables—Part I: Model Formulation". In: *Journal of Turbomachinery-transactions of The Asme* 128 (2006), pp. 413–422. URL: https://api.semanticscholar.org/CorpusID:122103330.

[36] Marvin Minsky and Seymour Papert. "Perceptrons: An Introduction to Computational Geometry". In: 1969. URL: https://api.semanticscholar.org/CorpusID:267833493.

[37] Parviz Moin and Krishnan Mahesh. “DIRECT NUMERICAL SIMULATION: A Tool in Turbulence Research”. In: *Annual Review of Fluid Mechanics* 30 (1998), pp. 539–578. URL: https://api.semanticscholar.org/CorpusID:14986531.

[38] John A. Nelder and Roger Mead. “A Simplex Method for Function Minimization”. In: *Comput. J.* 7 (1965), pp. 308–313. URL: https://api.semanticscholar.org/CorpusID:2208295.

[39] Maurizio Pandolfi. “A Contribution to the Numerical Prediction of Unsteady Flows”. In: *AIAA Journal* 22 (1983), pp. 602–610. URL: https://api.semanticscholar.org/CorpusID:121337372.

[40] Eric J. Parish and Karthik Duraisamy. “A paradigm for data-driven predictive modeling using field inversion and machine learning”. In: *J. Comput. Phys.* 305 (2016), pp. 758–774. URL: https://api.semanticscholar.org/CorpusID:7190361.

[41] Sacha Parneix, Dominique R. Laurence, and Paul A. Durbin. “A Procedure for Using DNS Databases”. In: *Journal of Fluids Engineering-transactions of The Asme* 120 (1998), pp. 40–47. URL: https://api.semanticscholar.org/CorpusID:119898690.

[42] Emanuel Parzen. “On Estimation of a Probability Density Function and Mode”. In: *Annals of Mathematical Statistics* 33 (1962), pp. 1065–1076. URL: https://api.semanticscholar.org/CorpusID:122932724.

[43] Ugo Piomelli. “Large-eddy simulation: achievements and challenges”. In: *Progress in Aerospace Sciences* 35 (1999), pp. 335–362. URL: https://api.semanticscholar.org/CorpusID:14957210.

[44] Stephen B. Pope. “A more general effective-viscosity hypothesis”. In: *Journal of Fluid Mechanics* 72 (1975), pp. 331–340. URL: https://api.semanticscholar.org/CorpusID:56429958.

[45] Stephen B. Pope. “Turbulent Flows”. In: *Measurement Science and Technology* 12 (2000), pp. 2020–2021. URL: https://api.semanticscholar.org/CorpusID:221565114.

[46] M. J. D. Powell. “An efficient method for finding the minimum of a function of several variables without calculating derivatives”. In: *Comput. J.* 7 (1964), pp. 155–162. URL: https://api.semanticscholar.org/CorpusID:62756844.

[47] Wolfgang Rodi and Georg Scheuerer. “Scrutinizing the k-ϵ Turbulence Model Under Adverse Pressure Gradient Conditions”. In: *Journal of Fluids Engineering-transactions of The Asme* 108 (1986), pp. 174–179. URL: https://api.semanticscholar.org/CorpusID:120562892.

[48] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65 6 (1958), pp. 386–408. URL: https://api.semanticscholar.org/CorpusID:12781225.

[49] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *ArXiv* abs/1609.04747 (2016). URL: https://api.semanticscholar.org/CorpusID:17485266.

[50] Christopher Lockwood Rumsey, Brian R. Smith, and George P. Huang. “Description of a Website Resource for Turbulence Modeling Verification and Validation”. In: 2010. URL: https://api.semanticscholar.org/CorpusID:62132754.

[51] Ventzislav Rusanov. “The calculation of the interaction of non-stationary shock waves and obstacles”. In: *Ussr Computational Mathematics and Mathematical Physics* 1 (1962), pp. 304–320. URL: https://api.semanticscholar.org/CorpusID:121629888.

[52] Richard D. Sandberg and Vittorio Michelassi. “The Current State of High-Fidelity Simulations for Main Gas Path Turbomachinery Components and Their Industrial Impact”. In: *Flow, Turbulence and Combustion* 102 (2019), pp. 797–848. URL: https://api.semanticscholar.org/CorpusID:127878295.

[53] Ulrich Schumann. “Realizability of Reynolds-Stress Turbulence Models”. In: *Physics of Fluids* 20 (1977), pp. 721–725. URL: https://api.semanticscholar.org/CorpusID:121481036.

[54] Roger L. Simpson. “Turbulent Boundary-Layer Separation”. In: *Annual Review of Fluid Mechanics* 21 (1989), pp. 205–234. URL: https://api.semanticscholar.org/CorpusID:31818144.

[55] Anand Pratap Singh, Shivaji Medida, and Karthik Duraisamy. “Machine Learning-augmented Predictive Modeling of Turbulent Separated Flows over Airfoils”. In: *ArXiv* abs/1608.03990 (2016). URL: https://api.semanticscholar.org/CorpusID:1789115.

[56] Philippe R. Spalart. “Comments on the feasibility of LES for wings, and on a hybrid RANS/LES approach”. In: 1997. URL: https://api.semanticscholar.org/CorpusID:116820008.

[57] Philippe R. Spalart. “Detached-Eddy Simulation”. In: *Annual Review of Fluid Mechanics* 41 (2009), pp. 181–202. URL: https://api.semanticscholar.org/CorpusID:119889966.

[58] Philippe R. Spalart. “Philosophies and fallacies in turbulence modeling”. In: *Progress in Aerospace Sciences* 74 (2015), pp. 1–15. URL: https://api.semanticscholar.org/CorpusID:122501561.

[59] Charles G. Speziale. “ANALYTICAL METHODS FOR THE DEVELOPMENT OF REYNOLDS-STRESS CLOSURES IN TURBULENCE”. In: 1990. URL: https://api.semanticscholar.org/CorpusID:54690219.

[60] Jian-Xun Wang, Jin-Long Wu, and Heng Xiao. “Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data”. In: *arXiv: Fluid Dynamics* 2 (2016), p. 034603. URL: https://api.semanticscholar.org/CorpusID:73687397.

[61] Jack Weatheritt and Richard D. Sandberg. “A novel evolutionary algorithm applied to algebraic modifications of the RANS stress-strain relationship”. In: *J. Comput. Phys.* 325 (2016), pp. 22–37. URL: https://api.semanticscholar.org/CorpusID:30351646.

[62] Paul J. Werbos. “Generalization of backpropagation with application to a recurrent gas market model”. In: *Neural Networks* 1 (1988), pp. 339–356. URL: https://api.semanticscholar.org/CorpusID:205118721.

[63] David C. Wilcox. “Formulation of the k-w Turbulence Model Revisited”. In: *AIAA Journal* 46 (2008), pp. 2823–2838. URL: https://api.semanticscholar.org/CorpusID:120424514.

[64] David C. Wilcox. *Turbulence modeling for CFD*. DCW Industries, 1993. URL: https://api.semanticscholar.org/CorpusID:117365446.

[65] Jin-Long Wu, Heng Xiao, and Eric Paterson. "Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework". In: *Physical Review Fluids* (2017). URL: https://api.semanticscholar.org/CorpusID:119085443.

[66] Jin-Long Wu et al. "A Priori Assessment of Prediction Confidence for Data-Driven Turbulence Modeling". In: *Flow, Turbulence and Combustion* 99 (2016), pp. 25–46. URL: https://api.semanticscholar.org/CorpusID:119239370.

[67] Heng Xiao and Paola Cinnella. "Quantification of model uncertainty in RANS simulations: A review". In: *Progress in Aerospace Sciences* (2018). URL: https://api.semanticscholar.org/CorpusID:53479580.

[68] Heng Xiao et al. "Flows over periodic hills of parameterized geometries: A dataset for data-driven turbulence modeling from direct simulations". In: *Computers & Fluids* (2019). URL: https://api.semanticscholar.org/CorpusID:203642119.

[69] Yaomin Zhao et al. "Turbulence Model Development using CFD-Driven Machine Learning". In: *arXiv: Fluid Dynamics* (2019). URL: https://api.semanticscholar.org/CorpusID:119361386.