POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

SCIENCES
SORBONNE
UNIVERSITÉ

DE LA RECHERCHE À L'INDUSTRIE
cea

Code integration and validation of a machine learning based RANS model

Tesi di Laurea Magistrale in
Mathematical Engineering - Ingegneria Matematica

Author: **Davide Repetto**

Student ID: 990543
Politecnico di Milano Advisor: Prof. Marco Verani
Sorbonne Université Advisor: Prof. Corrado Maurini
CEA/CNRS Advisors: Pierre-Emmanuel Angeli, Didier Lucor

Academic Year: 2022-23

Abstract

Turbulence presents a complex modeling challenge in fluid dynamics simulations. Traditional approaches rely on Reynolds-averaged Navier-Stokes (RANS) equations paired with closure models, but often lack accuracy and universality. This thesis investigates integrating machine learning into RANS frameworks to enhance turbulence modeling.

Both high-Reynolds and low-Reynolds neuronal models are proposed and validated on the canonical turbulent channel flow configuration. Preliminary tests injecting explicit Reynolds stresses from direct numerical simulations (DNS) into RANS equations reveal ill-conditioning concerns. The high-Reynolds neuronal $k - \varepsilon$ model demonstrates comparable performance to the standard $k - \varepsilon$ model, while requiring over twice the computational expense. However, the low-Reynolds neuronal models exhibit promising capabilities, with neural network outputs significantly outperforming analytical closures in predicting Reynolds stress anisotropy.

This research clarifies ambiguities in prior approaches and validates a generalized tensor formulation to address limitations. Code integration in the TRUST/TrioCFD solver enables practical usage for academic and industrial simulations. Overall, the physics-informed machine learning techniques presented strong potential to enhance turbulence modeling. Although limitations exist, the improved accuracy and reliability constitute valuable contributions towards advancing RANS capabilities.

Keywords: CFD, RANS, Turbulence, Machine Learning, $k - \varepsilon$ models, Low-Reynolds models.

Abstract in lingua italiana

La turbolenza rappresenta una sfida complessa nella modellistica delle simulazioni di dinamica dei fluidi. Gli approcci tradizionali si basano sulle equazioni Reynolds-Averaged Navier-Stokes (RANS) abbinati a modelli di chiusura del tensore di Reynolds, ma spesso presentano problemi di precisione e universalità. Questa tesi investiga l'integrazione del machine learning nei modelli di chiusura delle RANS per migliorare la modellazione della turbolenza.

Vengono proposti e validati modelli neurali sia ad alto che basso numero di Reynolds sulla configurazione di flusso turbolento in un canale piano. I test preliminari in cui il tensore Reynolds è trattato esplicitamente mostrano problemi dovuti al cattivo condizionamento delle equazioni RANS. Il modello neuronale $k-\varepsilon$ ad alto numero di Reynolds dimostra una performance comparabile al modello standard $k-\varepsilon$, richiedendo però oltre il doppio del costo computazionale. D'altra parte, i modelli neurali a basso numero di Reynolds mostrano capacità promettenti. In particolare gli output della rete neurale sono significativamente migliori dei valori relativi alle stesse grandezze ottenuti dai modelli tradizionali.

L'integrazione del codice nel solver TRUST/TrioCFD consente un utilizzo del modello neuronale a basso Reynolds per simulazioni accademiche e industriali. Nel complesso, le tecniche di apprendimento automatico presentano un forte potenziale per migliorare la modellazione della turbolenza. Nonostante le limitazioni, l'aumentata precisione e affidabilità costituiscono contributi preziosi per l'avanzamento delle capacità RANS.

Parole chiave: Fluidodinamica Computazionale, RANS, Turbolenza, Machine Learning, Modelli $k-\varepsilon$, Modelli a basso numero di Reynolds.

Resumé en français

La turbulence représente un défi complexe dans la modélisation des simulations de dynamique des fluides. Les approches traditionnelles sont basées sur les équations de Navier-Stokes moyennées dans le temps (RANS) associées à des modèles de fermeture du tenseur de Reynolds, mais elles présentent souvent des problèmes de précision et d'universalité. Cette thèse explore l'intégration de l'apprentissage automatique dans les modèles de fermeture RANS afin d'améliorer la modélisation de la turbulence.

Des modèles neuronaux sont proposés et validés pour des nombres de Reynolds haut et bas, sur la configuration d'écoulement turbulent dans un canal plan. Les tests préliminaires dans lesquels le tenseur de Reynolds est traité explicitement révèlent des problèmes liés au mauvais conditionement des équations RANS. Le modèle neuronal $k-\varepsilon$ à haut nombre de Reynolds montre des performances comparables au modèle standard $k-\varepsilon$, mais nécessite plus du double du coût de calcul. En revanche, les modèles neuronaux à bas nombre de Reynolds montrent des capacités prometteuses. En particulier, les sorties du réseau neuronal sont nettement meilleures que les valeurs correspondantes obtenues à partir des modèles traditionnels.

L'intégration du code dans le solveur TRUST/TrioCFD permet l'utilisation du modèle neuronal à bas Reynolds pour des simulations académiques et industrielles. Dans l'ensemble, les techniques d'apprentissage automatique présentent un fort potentiel pour améliorer la modélisation de la turbulence. Malgré les limitations, l'augmentation de la précision et de la fiabilité apporte des contributions précieuses à l'avancement des capacités RANS.

Mots clés: CFD, RANS, Turbulence, Machine Learning, Modèles $k-\varepsilon$, Modèles bas Reynolds.

Contents

Introduction

Turbulence constitutes one of the most complex unsolved problems in classical physics. The chaotic and stochastic nature of turbulent flows poses immense challenges for computational modeling and simulation. Precisely predicting turbulence remains a goal difficult to reach despite decades of research.

In the field of computational fluid dynamics (CFD) simulation, the use of machine learning has experienced significant growth in recent years, partially driven by the increase in available computational resources. Machine learning allows for the improvement of turbulence models that are often ineffective for complex flow situations by leveraging simulated data or experimental measurements. This involves utilizing functional structures such as neural networks, which are flexible and adaptable, to learn models from reference numerical data obtained through Direct Numerical Simulations (DNS). However, such data-driven learning presents a major drawback: it can produce models that do not conform to the laws of physics, resulting in predictions that are not guaranteed beyond the training domain. In fluid mechanics, a turbulence model must adhere to several invariances, notably Galilean and rotational invariances. Therefore, the functional structure must ensure these invariances independently of the data to advance towards a trusted form of Artificial Intelligence (AI). Undoubtedly, this represents one of the challenges in applying machine learning to CFD.

This document presents the continuation of the work on machine learning of the Reynolds tensor for Reynolds-Averaged Navier-Stokes (RANS) calculations, building upon the work of Cai *et al.* [3]. The focus of this work is on the closure of the Reynolds tensor, particularly the $k-\varepsilon$ model. The RANS equation, obtained by applying a statistical averaging operator, introduces an unknown object known as the Reynolds stress tensor. The challenge lies in establishing how the Reynolds tensor depends on the deformations of the mean velocity field. Historically, this dependence is expressed through explicit mathematical functions of varying complexity. However, these models are not sufficiently generic to guarantee their applicability to all flow cases. The limitations become particularly pronounced for flows involving curvature, separation, rotation, and swirl [4]. Machine learning now offers an alternative to these closure laws based on physical expertise.

This master's thesis, conducted in the LMSF laboratory of the French Atomic Energy Commission (CEA), aims to perform an *a posteriori* validation of the Reynolds tensor predicted through the neural networks trained by Cai *et al.* [3]. The advantage of this approach is to benefit from a functional structure proposed by Pope [17], which ensures all invariances on the Reynolds tensor, and to leverage the learning capabilities of a neural

network. The combination of the tensor basis and a neural network is referred to as TBNN (Tensor Basis Neural Network). The training of TBNN was conducted on databases derived from DNS calculations on turbulent flows in plane channels. The neural models obtained were then integrated into the TrioCFD computation code. This significant step allowed for the first RANS simulations with TrioCFD using machine learning-based models.

This document provides a detailed description of the project's progress and its various stages. It is structured as follows. Chapter 1 describes the physical modeling of fluid mechanics and analyzes the TBNN model in the case of the flat channel to evaluate its validity. Chapter 2 is dedicated to the machine learning method for the Reynolds tensor and their training. Chapter 3 presents the machine learning based RANS closure models object of validation in this work. Chapter 4 provides details on the integration of the low-Reynolds neuronal model in the TRUST/TrioCFD solver. Chapter 5 is dedicated to the *a posteriori* validation of the neural models through simulations with TrioCFD. Finally, Chapter 6 concludes the note, discussing the encountered challenges and presenting numerous prospects for the future.

CEA: The French Alternative Energies and Atomic Energy Commission

The French Atomic Energy Commission (CEA) was created in 1945 by Charles de Gaulle, headed by Frédéric Joliot-Curie (High Commissioner for Atomic Energy) and Raoul Dautry (General Administrator). The organization's purpose is to pursue scientific and technical research into the use of nuclear energy in science (particularly medical applications), industry (electricity) and national defense. With more than 20,000 employees - technicians, engineers, researchers, doctoral and post-doctoral students, and research support staff - the CEA is involved in numerous collaborative projects alongside its academic and industrial partners. The four main departments are: Direction of Energies (DES), Direction of Militar Applications (DAM), Direction of technological research (DRT/CEA tech) and Division of Fundamental Research (DRF). CEA's main research centers are in Saclay (Essonne), Fontenay-aux-Roses (Hauts de Seine), Marcoule (Gard), Cadarache (Bouches du Rhône), Grenoble (Isère) and Le Ripault (Indre-et-Loire).

The present work has been developed in the Direction of Energies and more precisely in the Service de Thermohydraulique et de Mécanique des Fluides (STMF). The goal of this service is to develop thermal-hydraulics and fluid mechanics simulation software applied to low-carbon energy technologies, mainly for nuclear reactors and facilities.

1 | Context of Study

In this chapter the theoretical framework of this master thesis is discussed. Section 1.1 presents the Navier-Stokes equations for incompressible fluids, Section 1.2 delves into their computational modeling, Section 1.3 exhibits various turbulence models based on the Boussinesq assumption, Section 1.4 is dedicated to the discussion of the Pope's model as an alternative to the Boussinesq assumption, Section 1.5 analyzes the simplifications of the Pope's model in the plane channel problem, Section 1.6 presents the generalized $\mathbf{T}^{*(0)}$ tensor proposed by Cai *et al.* within the framework of the Pope's model.

1.1. Navier-Stokes equations for incompressible fluids

The Navier-Stokes equations constitute a fundamental framework for characterizing the dynamic behavior of fluids, playing a pivotal role in the study of fluid mechanics. Specifically, when considering a Newtonian fluid that is both incompressible and isothermal, such as water for the purposes of this investigation, the Navier-Stokes equations take the following form:

$$\begin{cases} \nabla \cdot \mathbf{u} = 0 & \text{(1.1a)} \\ \dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = -\dfrac{1}{\rho}\nabla p + \nu \Delta \mathbf{u} & \text{(1.1b)} \end{cases}$$

Here, the vector field $\mathbf{u}$ represents the velocity field in the Cartesian coordinate system, while p stands for the pressure, ρ denotes the fluid density, and ν the kinematic viscosity of the fluid.

Equation 1.1a expresses the mass equation, ensuring that the rate of mass flow into any given region is equal to the rate of outgoing mass flow. This conservation of mass principle is crucial in understanding the behavior of fluid flows.

Equation 1.1b captures the essence of fluid motion. It accounts for the temporal evolution

of the velocity field $\mathbf{u}$, incorporating the effects of advection (the convection of fluid properties by the velocity field), pressure gradients, and viscosity. The left-hand side of the equation represents the acceleration of the fluid particles, while the right-hand side encompasses the influence of pressure and viscosity. The presence of the Laplacian operator Δ on the velocity field signifies the diffusion of momentum due to viscosity.

The Navier-Stokes equations hold a critical place in many scientific and engineering applications. From understanding natural phenomena like ocean currents and atmospheric flows to designing complex systems such as aircraft aerodynamics and fluid transport networks, these equations provide a foundational framework for analyzing and predicting fluid behavior.

In the subsequent chapters, we will delve into the mathematical properties, analytical solutions, and numerical methods associated with the Navier-Stokes equation.

1.2. Computational modeling

The Computational Fluid Dynamics (CFD) is the study of the flows through numerical methods. Three main methods are employed: Direct Numerical Simulation (DNS), Large Eddy Simulation (LES) and Reynolds-Averaged Navier-Stokes (RANS). In this section, the DNS and RANS methodologies are exhibited, as they will subsequently be employed throughout the course of this thesis.

1.2.1. DNS modeling

The Direct Numerical Simulation (DNS) method stands as a powerful approach within fluid dynamics for studying complex flow phenomena in remarkable detail. In DNS, the Navier-Stokes equations, as introduced Section 1.1, are solved directly, without resorting to any simplifying approximations or turbulence model. This enables a comprehensive examination of fluid flow at the smallest scales, providing insights into intricate behaviors that might be missed when relying on coarser approximations.

DNS aims to capture the complete range of spatio-temporal scales present in a fluid flow, from the large-scale structures down to the smallest turbulent eddies. By numerically solving the governing equations on a discretized grid, DNS reveals the evolution of the velocity and pressure fields at each point in space and time. This meticulous level of detail allows for the investigation of phenomena such as vortex shedding, boundary layer interactions, and turbulence dynamics with high precision.

However, it's important to acknowledge that DNS comes at a significant computational

cost. Due to the vast number of grid points required to accurately capture the smallest turbulent scales, DNS simulations demand substantial computational resources and time. The computations involve solving differential equations over a three-dimensional grid, often with hundreds of millions to billions of grid points, making DNS one of the most computationally intensive methods in fluid dynamics. This substantial computational cost originates from the need to resolve a wide range of scales, which pushes the limits of available computational technology.

1.2.2. RANS modeling

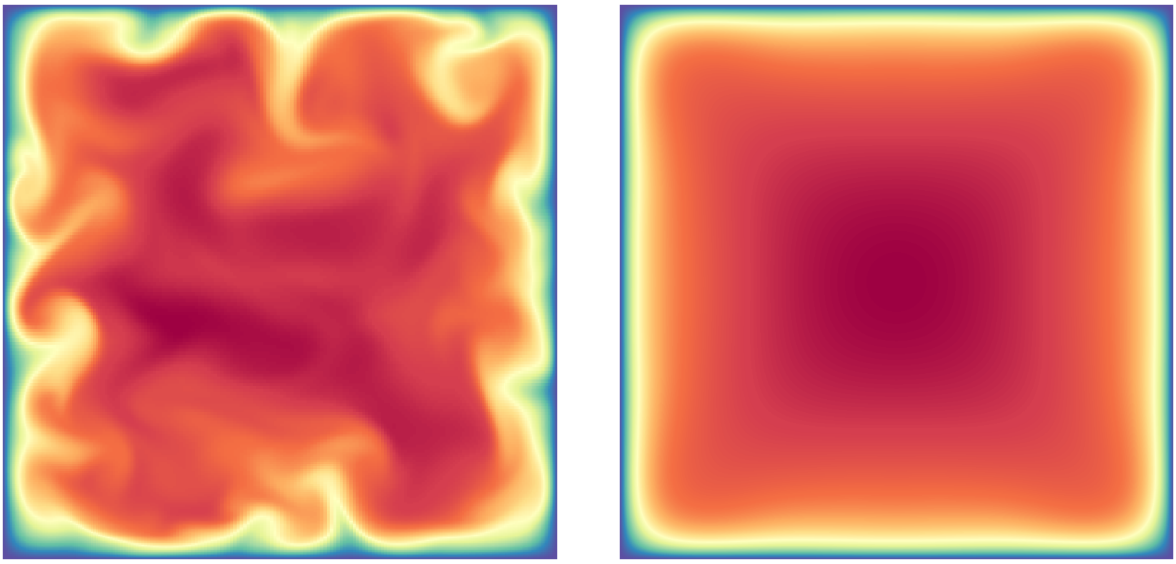
The Reynolds-Averaged Navier-Stokes (RANS) modeling approach presents a framework that provides a computationally affordable approach to capture turbulent effects while serving as a bridge between the computational cost of Direct Numerical Simulation (DNS) and the practical demands of computational resources. The fundamental concept underlying RANS modeling involves a decomposition of the flow field $\mathbf{u}$ into a time-averaged component $\overline{\mathbf{u}}$ and smaller, local fluctuations $\mathbf{u}'$. This approach allows us to separate the mean behavior of the flow from the turbulent fluctuations, leading to a simplified representation of the governing equations. With the RANS modeling the aim becomes to determine the averaged field $\overline{\mathbf{u}}$, therefore the expected output for time-dependent problems, such has the ones involving turbulence, differs from the one obtained from a DNS. This difference is displayed in Figure 1.1.

Upon applying the statistical mean operator to the Navier-Stokes equations (Equations 1.1), and introducing the decomposition $\mathbf{u} = \overline{\mathbf{u}} + \mathbf{u}'$, the resulting equations, called Reynolds-Average Navier-Stokes (RANS) equations, take the form:

$$\begin{cases} \dfrac{\partial \bar{u}_i}{\partial x_i} = 0 & \text{(1.2a)} \\ \dfrac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \dfrac{\partial \bar{u}_i}{\partial x_j} = -\dfrac{1}{\rho}\dfrac{\partial \bar{p}}{\partial x_i} + \dfrac{\partial}{\partial x_j}\left[\nu\left(\dfrac{\partial \bar{u}_i}{\partial x_j} + \dfrac{\partial \bar{u}_j}{\partial x_i}\right) - \overline{u_i' u_j'}\right] & \text{(1.2b)} \end{cases}$$

In these equations, an additional component emerges: the Reynolds tensor, denoted as $\mathscr{R}_{ij} = \overline{u_i' u_j'}$. This tensor represents turbulent fluctuations in the flow. This variable assumes a null value in laminar flow conditions. A challenge that must be addressed is the necessity to close the Equations 1.2 by representing the Reynolds tensor using average field values. To overcome this, the turbulent viscosity concept is introduced.

Analogous to the Newtonian hypothesis relating viscous stresses to strain rates, Boussinesq proposed a simple assumption. Turbulent motions are assimilated to viscous effects



(a) DNS. (b) RANS.

Figure 1.1: Comparison between DNS and RANS simulation outputs in the turbulent square channel [1].

but with a turbulent viscosity coefficient ν_t. This hypothesis, known as the Linear Eddy Viscosity Model (LEVM) or Boussinesq assumption, provides an avenue to tackle the RANS closure problem [2]. It is represented as:

$$\mathscr{R}_{ij} = -2\nu_t S_{ij} + \frac{2}{3}k\delta_{ij} \tag{1.3}$$

Where S_{ij} is defined as:

$$S_{ij} = \frac{1}{2}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) \tag{1.4}$$

and k is the turbulent kinetic energy, $k = \frac{1}{2}\overline{u_i' u_i'}$.

In scientific literature, it is conventional to utilize the dimensionless anisotropy tensor b_{ij}, as a replacement for the Reynolds tensor. It is defined as:

$$b_{ij} = \frac{\mathscr{R}_{ij}}{2k} - \frac{1}{3}\delta_{ij} \tag{1.5}$$

By combining the equations 1.3 and 1.5, the Boussinesq hypothesis becomes:

$$b_{ij} = -\frac{\nu_t}{k} S_{ij} \tag{1.6}$$

1.3. Turbulence Models based on LEVM

In this section, the concept of turbulence and the most relevant turbulence models based on the Boussinesq Hypothesis are discussed. Turbulence is a complex phenomenon observed in fluid flows where small-scale eddies emerge as the flow rate increases. These eddies introduce spatial and temporal oscillations, making it computationally infeasible to directly solve, by means of DNS, the Navier-Stokes equations for turbulent flows. To address this challenge, the RANS formulation is employed.

One approach to turbulence modeling involves using supplementary transport equations to describe turbulence-related variables. In one- and two-equations models, such additional equations are incorporated to estimate turbulence intensity. By employing these models, a more manageable set of equations can be used to represent turbulence.

Alternatively, algebraic models adopt a different approach by introducing algebraic equations that depend on the velocity field to characterize turbulence intensity. This is achieved by calculating an eddy viscosity, which augments the molecular viscosity of the fluid. The turbulent viscosity accounts for the momentum that would be carried by small-scale eddies, effectively attributing it to a viscous transport process. Throughout most of the flow domain, turbulence dissipation largely dominates over viscous dissipation, except in the vicinity of solid walls. In these cases, turbulence models must adapt by continually attenuating the turbulence level, as observed in low-Reynolds number models, or by calculating new boundary conditions through the application of wall functions.

Figure 1.2 presents the concept of turbulence through the example of a uniform flux over a flat plate. As the distance from the leading edge increases, the Reynolds number grows, leading to a transition from laminar to turbulent flow.

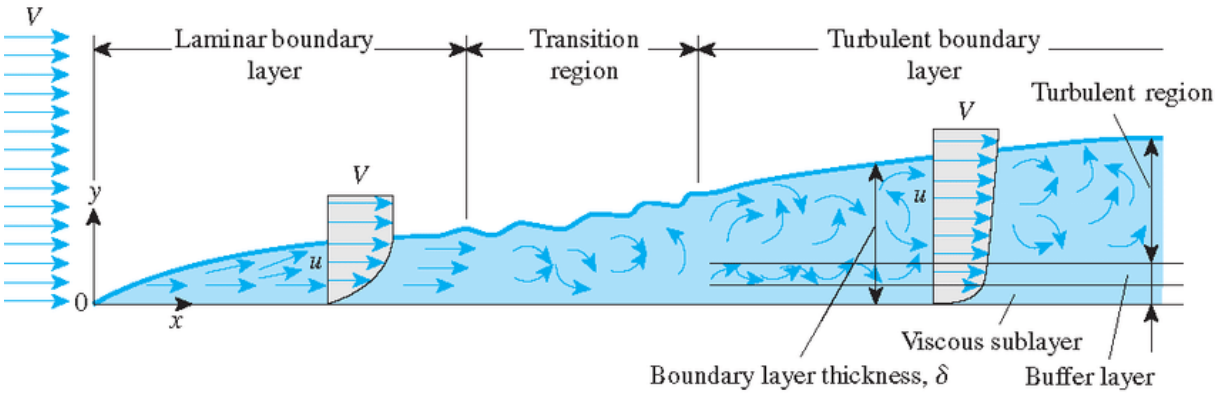


Figure 1.2: Development of the boundary layer for flow over a flat plate.
Courtesy of University of Delaware.

To estimate the value of turbulent viscosity (ν_t) more accurately, various turbulence models have been developed over the years. These models are classified based on the number of additional transport equations that need to be solved alongside the RANS flow equations. In the following subsections, the most widely used turbulence models based on the Boussinesq Hypothesis are discussed in detail, as a summary of the ones presented in the work of Versteeg and Malasekara [23].

1.3.1. Mixing length model

The mixing length model proposed by Ludwig Prandtl [19] belongs to the category of the zero equation models, that are the models that add zero transport equations to close the RANS problem. Indeed, ν_t is directly computed by means of a dimensional analysis. Since ν_t has dimensions $\left[\frac{m^2}{s}\right]$ it can be expressed as the product of a velocity $\left[\frac{m}{s}\right]$ and a length $[m]$. Thus, one can write the expression of ν_t as function of a velocity ϑ and a length ℓ as:

$$\nu_t = c\vartheta\ell \tag{1.7}$$

where c represents a dimensionless constant. By assuming that there is a strong connection between the mean flow and the behavior of the largest eddies, one can attempt to link the characteristic velocity scale of the eddies with the mean flow properties. Therefore, by further assuming that the only significant mean velocity gradient is $\frac{\partial U}{\partial y}$, one has:

$$\vartheta = C\ell \left|\frac{\partial U}{\partial y}\right| \tag{1.8}$$

with ℓ that represents the characteristic length of the larger eddies.
By combining Equations 1.7 and 1.8 the equation of the mixing length model is obtained and reads:

$$\nu_t = \ell_m^2 \left| \frac{\partial U}{\partial y} \right| \tag{1.9}$$

where the mixed length, ℓ_m, is obtained from ℓ and the two constants c and C. It has to be noticed that when the turbulence changes, this model has to adapt by changing ℓ_m, but when the structure of the flow is simple, ℓ_m is obtained from algebraic formulas.

1.3.2. Standard $k-\varepsilon$ model

The $k-\varepsilon$ model introduces two additional transport equations to represent the turbulent properties of the flow. These are the turbulent kinetic energy, k, and its rate of dissipation, ε.

The equations for the turbulent kinetic energy, k, and for the rate of dissipation, ε, are given by:

$$\begin{cases} \dfrac{\partial k}{\partial t} + \bar{u}_i \dfrac{\partial k}{\partial x_i} = \dfrac{\partial}{\partial x_i}\left[\left(\nu + \dfrac{\nu_t}{\sigma_k}\right)\dfrac{\partial k}{\partial x_i}\right] + \mathscr{P} - \varepsilon & \text{(1.10a)} \\ \dfrac{\partial \varepsilon}{\partial t} + \bar{u}_i \dfrac{\partial \varepsilon}{\partial x_i} = \dfrac{\partial}{\partial x_i}\left[\left(\nu + \dfrac{\nu_t}{\sigma_\varepsilon}\right)\dfrac{\partial \varepsilon}{\partial x_i} + \dfrac{\varepsilon}{k}\left(C_{\varepsilon_1}\mathscr{P} - C_{\varepsilon_2}\varepsilon\right)\right] & \text{(1.10b)} \end{cases}$$

where μ_t is the turbulent viscosity, and ρ is the fluid density, while σ_k, σ_ε $C_{1\varepsilon}$ and $C_{2\varepsilon}$ are constants.

The two quantities introduced by the model, k and ε, are then used to define the velocity and length scale by means of a dimensional analysis, respectively ϑ and ℓ:

$$\vartheta = k^{\frac{1}{2}},\ \ell = \frac{k^{\frac{3}{2}}}{\varepsilon} \tag{1.11}$$

By following the same procedure of section 1.3.1 one obtains that the turbulent viscosity, μ_t, is computed from k and ε as:

$$\mu_t = \rho\, C \vartheta \ell = \rho\, C_\mu \frac{k^2}{\varepsilon} \tag{1.12}$$

where C_μ is a model constant. Therefore, since $\mu_t = \rho\nu_t$, ν_t is given by:

$$\nu_t = C_\mu \frac{k^2}{\varepsilon} \tag{1.13}$$

The standard $k-\varepsilon$ model is a semi-empirical model, with constants derived from experimental data. The commonly used values are $\sigma_k = 1.00$, $\sigma_\varepsilon = 1.30$, $C_\mu = 0.09$, $C_{1\varepsilon} = 1.44$, and $C_{2\varepsilon} = 1.92$.

Despite its simplicity and robustness, the standard $k-\varepsilon$ model has limitations. It assumes isotropic turbulence, which is not accurate for flows with strong streamline curvature or swirl, near-wall flows, and flows with rapid strain rates [4]. Nevertheless, it remains a popular choice for initial studies and engineering applications due to its ease of implementation and computational efficiency.

Since the standard $k-\varepsilon$ model does not provide a good estimation of k and ε in the near wall region, this model has to be used always paired with a wall function. Wall functions are employed to approximate the flow characteristics in the buffer region and analytically determine a non-zero fluid velocity in proximity to the wall. Adopting a wall function approach presumes an analytical solution for the flow within the viscous layer, leading to substantially reduced computational demands for the resultant models.

1.3.3. Low-Reynolds Number $k-\varepsilon$ Models

The term "Low-Reynolds number model" may appear paradoxical, as turbulent flows typically occur at high Reynolds numbers. However, this denomination does not pertain to the flow on a global scale; rather, it pertains to the near-wall region where viscous effects dominate, specifically the viscous sublayer as depicted in the figure above. A low-Reynolds number model is one that accurately captures the limiting behaviors of various flow parameters as the distance to the wall approaches zero. The accurate representation of these limiting behaviors enables the turbulence model to simulate the entire boundary layer, encompassing both the viscous sublayer and the buffer layer.

Low-Reynolds number models often provide remarkably precise descriptions of the boundary layer. Nonetheless, the sharp gradients near walls necessitate extremely high mesh resolutions, resulting in a substantial computational cost associated with achieving this high accuracy.

The Low-Reynolds number $k-\varepsilon$ model, an extension of the conventional $k-\varepsilon$ model, shares many advantageous characteristics while eliminating the need for wall functions, as it can adequately resolve the flow throughout the entire domain. Consequently, a finer mesh, and therefore a higher computational cost, is typically required in proximity to

walls.

The equations for the turbulent kinetic energy, k, and for the rate of dissipation, ε, are given by:

$$\begin{cases} \dfrac{\partial k}{\partial t} + \bar{u}_i \dfrac{\partial k}{\partial x_i} = \dfrac{\partial}{\partial x_i}\left[\left(\nu + \dfrac{\nu_t}{\sigma_k}\right)\dfrac{\partial k}{\partial x_i}\right] + \mathscr{P} - \varepsilon - D & (1.14a) \\ \dfrac{\partial \varepsilon}{\partial t} + \bar{u}_i \dfrac{\partial \varepsilon}{\partial x_i} = \dfrac{\partial}{\partial x_i}\left[\left(\nu + \dfrac{\nu_t}{\sigma_\varepsilon}\right)\dfrac{\partial \varepsilon}{\partial x_i} + \dfrac{\varepsilon}{k}\left(C_{\varepsilon_1} f_1 \mathscr{P} - C_{\varepsilon_2} f_2 \varepsilon\right)\right] + E & (1.14b) \\ \nu_t = C_\mu f_\mu \dfrac{k^2}{\varepsilon} & (1.14c) \end{cases}$$

In red are highlighted all the terms introduced in the Low-Reynolds number $k-\varepsilon$ model that were not there in the Standard one (Equation 1.10). Due to the empirical nature of the parameters involved in this formulation, Table 1.1 presents an overview of their values in two popular models.

	Jones-Launder [9]	**Launder-Sharma** [11]
C_μ	0.09	0.09
σ_k	1.0	1.0
σ_ε	1.3	1.3
D	$2\,\nu \left(\dfrac{\partial \sqrt{k}}{\partial y}\right)^2$	$2\,\nu \left(\dfrac{\partial \sqrt{k}}{\partial y}\right)^2$
E	$2\,\nu\,\nu_t \left(\dfrac{\partial^2 u}{\partial y^2}\right)^2$	$2\,\nu\,\nu_t \left(\dfrac{\partial^2 u}{\partial y^2}\right)^2$
C_{ε_1}	1.45	1.44
C_{ε_2}	2.0	1.92
f_μ	$\exp \dfrac{-2.5}{(1+Re_t/50)}$	$\exp \dfrac{-3.4}{(1+Re_t/50)^2}$
f_1	1.0	1.0
f_2	$1-0.3\exp\left(-Re_t^2\right)$	$1-0.3\exp\left(-Re_t^2\right)$
Re_t	$k^2/\nu\varepsilon$	$k^2/\nu\varepsilon$

Table 1.1: Low-Reynolds Models Parameter Selection

The utilization of the Low-Reynolds number $k-\varepsilon$ model enables more accurate computations of lift and drag forces, as well as improved predictions of heat fluxes compared to the standard $k-\varepsilon$ model. Additionally, the model demonstrates commendable per-

formance in predicting separation and reattachment phenomena across various scenarios. These valuable attributes render it a powerful tool for investigating fluid dynamics problems involving Low-Reynolds number flows, where traditional turbulence models may lack precision and reliability.

1.4. General eddy viscosity model

It has been shown that for complex flows, involving curvature, impingement and separation, the linear relationship proposed by Boussinesq (Equation 1.3) between the Reynolds stress tensor and the mean velocity gradient turns out to be inaccurate [4]. Moreover, *Tracey et al.* demonstrated that the LEVM does not capture the correct Reynolds stress anisotropy in many flows, including simple shear flows [21].

For this reason, a variety of Non-Linear Eddy Viscosity Models (NLEVM) have been developed to ensure the capture of these more complex effects. For the sake of example, the Quadratic Eddy Viscosity Model introduced by *Craft et al.* [4] can be cited.

Pope proposed one of the most widely used Nonlinear Eddy Viscosity Models (NLEVM) to extend the applicability of Reynolds-Averaged Navier-Stokes (RANS) closure models [17]. Pope's approach is based on the Reynolds stress anisotropy tensor $\mathbf{b}$ and postulates that it can be expressed as a function of normalized strain-rate $\mathbf{S}^*$ and rotation-rate $\mathbf{R}^*$ tensors for homogeneous flows:

$$\mathbf{b} = \mathbf{b}\left(\mathbf{S}^*, \mathbf{R}^*\right) \tag{1.15}$$

Where the tensors $\mathbf{S}^*$ and $\mathbf{R}^*$ are normalized using a turbulent timescale formed with the turbulent kinetic energy and dissipation rate, and they read as follows:

$$S_{ij}^* = \frac{1}{2}\frac{k}{\varepsilon}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right)$$

$$R_{ij}^* = \frac{1}{2}\frac{k}{\varepsilon}\left(\frac{\partial \bar{u}_i}{\partial x_j} - \frac{\partial \bar{u}_j}{\partial x_i}\right)$$

Assuming a polynomial form for the function 1.15 and applying the Cayley-Hamilton theorem, Pope obtained a closure model for $\mathbf{b}$ in the form of n a series of finite tensor polynomials:

$$\mathbf{b}\left(\mathbf{S}^*, \mathbf{R}^*\right) = \sum_n g^{(n)}(\lambda_1^*, \lambda_2^*, ...)\mathbf{T}^{*(n)} \tag{1.16}$$

Here, $g^{(n)}$ are coefficient functions dependent on physical independent invariants λ_i^* , while $\mathbf{T}^{*(n)}$ are basis tensors dependent on $\mathbf{S}^*$ and $\mathbf{R}^*$. In the general case, there are five invariants and ten tensors defined as:

$$\lambda_1^* = \mathrm{tr}\left(\mathbf{S}^{*2}\right), \quad \lambda_2^* = \mathrm{tr}\left(\mathbf{R}^{*2}\right), \quad \lambda_3^* = \mathrm{tr}\left(\mathbf{S}^{*2}\right), \quad \lambda_4 = \mathrm{tr}\left(\mathbf{R}^{*2}(\mathbf{S}\right), \quad \lambda_5^* = \mathrm{tr}\left(\mathbf{R}^{*2}\mathbf{S}^{*2}\right) \tag{1.17}$$

$$\begin{cases} \mathbf{T}^{*(1)} = \mathbf{S}^* & \mathbf{T}^{*(2)} = \mathbf{S}^*\mathbf{R}^* - \mathbf{R}^*\mathbf{S}^* \\ \mathbf{T}^{*(3)} = \mathbf{S}^{*2} - \dfrac{\lambda_1^*}{3}\mathbf{I}_3 & \mathbf{T}^{*(4)} = \mathbf{R}^{*2} - \dfrac{\lambda_2^*}{3}\mathbf{I}_3 \\ \mathbf{T}^{*(5)} = \mathbf{R}^*\mathbf{S}^{*2} - \mathbf{S}^{*2}\mathbf{R}^* & \mathbf{T}^{*(6)} = \mathbf{R}^{*2}\mathbf{S}^* + \mathbf{S}^*\mathbf{R}^{*2} - \dfrac{2\lambda_4^*}{3}\mathbf{I}_3 \\ \mathbf{T}^{*(7)} = \mathbf{R}^*\mathbf{S}^*\mathbf{R}^{*2} - \mathbf{R}^{*2}\mathbf{S}^*\mathbf{R}^* & \mathbf{T}^{*(8)} = \mathbf{S}^*\mathbf{R}^*\mathbf{S}^{*2} - \mathbf{S}^{*2}\mathbf{R}^*\mathbf{S}^* \\ \mathbf{T}^{*(9)} = \mathbf{R}^{*2}\mathbf{S}^{*2} + \mathbf{S}^{*2}\mathbf{R}^{*2} - \dfrac{2\lambda_5^*}{3}\mathbf{I}_3 & \mathbf{T}^{*(10)} = \mathbf{R}\mathbf{S}^{*2}\mathbf{R}^{*2} - \mathbf{R}^{*2}\mathbf{S}^{*2}\mathbf{R} \end{cases} \tag{1.18}$$

It is worth noting that Pope's model can be seen as a generalized form of the Linear Eddy Viscosity Model (LEVM) and the Quadratic Eddy Viscosity Model (QEVM) in specific approximations. For example, in the first-order approximation, the coefficient function $g^{(1)}$ can be identified with $-C_\mu$.

1.5. Turbulent Plane Channel Analysis

From the Pope's model it has been demonstrated that for flows where the mean velocity and variation of mean quantities in a specific direction are zero, only two invariants and a basis of three tensors are sufficient ($0 \leq n \leq 2$). The choice of the tensors $\mathbf{T}^{*(n)}$ depends on the flow's characteristics, such as the direction of invariance. For instance, if the flow exhibits invariance and zero mean velocity along the x_3 direction, the identity tensor $\mathbf{I}_2$ is set to be equal to `diag(1,1,0)`. Therefore, now $\mathbf{b}$ reads:

$$\mathbf{b} = g^{(0)}(\lambda_1^*, \lambda_2^*)\mathbf{T}^{*(0)} + g^{(1)}(\lambda_1^*, \lambda_2^*)\mathbf{T}^{*(1)} + g^{(2)}(\lambda_1^*, \lambda_2^*)\mathbf{T}^{*(2)} \tag{1.19}$$

with

$$\begin{cases} \mathbf{T}^{*(0)} = \frac{1}{2}\mathbf{I}_2 - \frac{1}{3}\mathbf{I}_3 \\ \mathbf{T}^{*(1)} = \mathbf{S}^* \\ \mathbf{T}^{*(2)} = \mathbf{S}^*\mathbf{R}^* - \mathbf{R}^*\mathbf{S}^* \end{cases} \tag{1.20}$$

It should be observed that the invariants and basis tensors in this simplified case are the same as those in the general model presented in 1.17 and 1.18, except for the choice of $\mathbf{T}^{*(0)}$ instead of $\mathbf{T}^{*(3)}$. It can be easily demonstrated that $\mathbf{T}^{*(3)} = -\lambda_1^*\mathbf{T}^{*(0)}$ under the restricted conditions of this simplified case.

Like the Navier-Stokes equations, the RANS equations and other closure models, Pope's model, presented in Equation 1.16, satisfies Galilean and rotational invariances. This means that the model remains invariant under translation and rigid-body rotation, which are fundamental properties of the fluid. However, it is important to consider that Pope's model may have limitations, especially in regions such as the near-wall region where additional parameters are needed to represent $\mathbf{b}$ [18]. This is often referred to as a multivalue problem [14] and will be addressed in subsequent discussions.

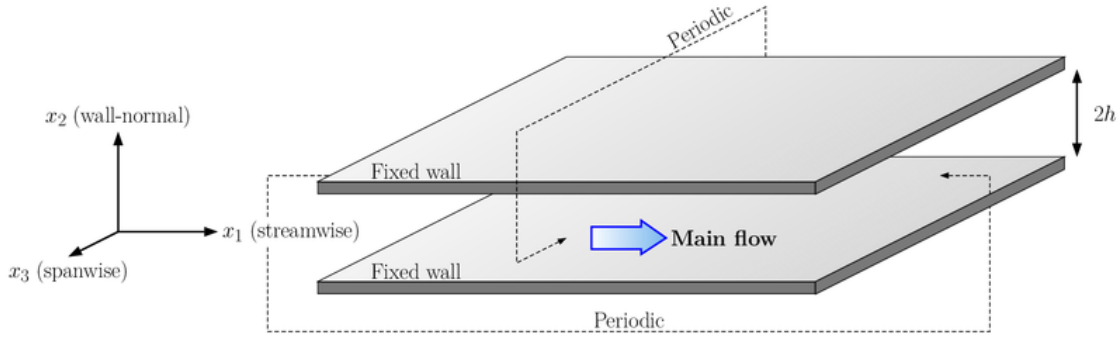


Figure 1.3: Sketch of the turbulent channel flow configuration.

The turbulent channel flow under consideration in this study refers to the flow between two parallel plates separated at a distance of $2h$. The streamwise direction is denoted as x_1, while the wall-normal and spanwise directions are denoted as x_2 and x_3, respectively. Figure 1.3 provides a sketch of the flow configuration. Previous investigations in the literature have extensively examined this configuration, and high fidelity simulation data can be found [7, 10, 16]. Researchers commonly use this flow as an academic case to validate newly developed models, including machine learning closure models proposed in recent years [5, 20]. Hence, a thorough investigation of this flow configuration is undertaken in the present work. Due to geometric invariance along the streamwise and

spanwise directions, the velocity statistics of this flow can be considered independent of x_1 and x_3 and therefore solely dependent on x_2. Consequently, their derivatives with respect to x_1 and x_3 are all zero. The mean continuity equation, as expressed in Equation 1.2a, can be simplified to:

$$\frac{\partial \bar{u}_2}{\partial x_2} = 0$$

This simplification implies that $\bar{u}_2 = 0$, as $\bar{u}_2(x_2 = 0,\, x_2 = 2h) = 0$ at the walls. Considering the physical invariance in the x_3 direction, one has that $\bar{u}_3 = 0$. Assuming that the system has evolved sufficiently for the flow to reach statistical stationarity, independent of time t, the momentum equations in 1.2b can be further simplified as :

$$\begin{cases} x_1\text{-direction: } 0 = -\dfrac{1}{\rho}\dfrac{\partial \bar{p}}{\partial x_1} + \nu\dfrac{\partial^2 \overline{u_1}}{\partial x_2^2} - \dfrac{\partial \overline{u_1' u_2'}}{\partial x_2} & \text{(1.21a)} \\ x_2\text{-direction: } 0 = -\dfrac{1}{\rho}\dfrac{\partial \bar{p}}{\partial x_2} - \dfrac{\partial \overline{u_2' u_2'}}{\partial x_2} & \text{(1.21b)} \\ x_3\text{-direction: } 0 = -\dfrac{\partial \overline{u_2' u_3'}}{\partial x_2} & \text{(1.21c)} \end{cases}$$

It is worth noting that the mean pressure gradient $\dfrac{\partial p}{\partial x_1}$ is non-zero, unlike the velocity statistics, as it serves as the driving force of the flow. From the above equations, it can be observed that only closure is required to determine the streamwise velocity profile $\bar{u}_1(x_2)$. This explains the focus of most researchers on predicting the primary component of the Reynolds stress anisotropy tensor in this flow configuration [5, 20, 25]. However, in contrast to their approach, the present study aims to fully predict the Reynolds stress anisotropy tensor. The inclusion of all non-zero statistics of the Reynolds stress would be highly beneficial in practical Reynolds-Averaged Navier-Stokes (RANS) simulations, particularly in the near-wall region and at the beginning of the calculation in terms of convergence. Furthermore, as presented in Equation 1.21, the b_{22} component is necessary to determine the pressure profile $\bar{p}(x_1, x_2)$.

To achieve this objective, the Pope's model will be applied to the turbulent channel flow. It should be noted that the characteristics in the x_3 direction of this flow satisfy the conditions of Pope's simplified model presented in Equations 1.19 and 1.20. To express the Reynolds stress anisotropy tensor, the normalized mean strain-rate, rotation-rate, and Reynolds stress anisotropy tensors are provided as follows:

$$\mathbf{S}^* = \frac{1}{2}\begin{bmatrix} 0 & \alpha & 0 \\ \alpha & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{R}^* = \frac{1}{2}\begin{bmatrix} 0 & \alpha & 0 \\ -\alpha & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} b_{11} & b_{12} & 0 \\ b_{12} & b_{22} & 0 \\ 0 & 0 & b_{13} \end{bmatrix} \tag{1.22}$$

where $\alpha = \frac{k}{\varepsilon}\frac{\partial \bar{u}_1}{\partial x_2}$ is a normalized mean velocity gradient, also the only nonzero mean velocity statistics.

By substituting 1.22 into 1.17 and 1.20 one obtains:

$$\lambda_1^* = \text{tr}\left(\mathbf{S}^{*2}\right) = \frac{\alpha^2}{2}, \quad \lambda_2^* = \text{tr}\left(\mathbf{R}^{*2}\right) = -\frac{\alpha^2}{2} \tag{1.23}$$

$$\begin{cases} \mathbf{T}^{*(0)} = \frac{1}{2}\mathbf{I}_2 - \frac{1}{3}\mathbf{I}_3 & = \begin{bmatrix} \frac{1}{6} & 0 & 0 \\ 0 & \frac{1}{6} & 0 \\ 0 & 0 & -\frac{1}{3} \end{bmatrix} \\ \mathbf{T}^{*(1)} = \mathbf{S}^* & = \begin{bmatrix} 0 & \frac{\alpha^2}{2} & 0 \\ \frac{\alpha^2}{2} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \mathbf{T}^{*(2)} = \mathbf{S}^*\mathbf{R}^* - \mathbf{R}^*\mathbf{S}^* & = \begin{bmatrix} \frac{\alpha^2}{2} & 0 & 0 \\ 0 & -\frac{\alpha^2}{2} & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{cases} \tag{1.24}$$

Since it is important to notice that $\lambda_1^* = -\lambda_2^*$, it is enough to keep only λ_1^* in the following. From now on $\mathbf{T}^{*(0)}$ will be denoted as $\mathbf{T}^{*(03)}$ to keep track of the location of the zero in the $\mathbf{I}_2$ tensor. Therefore, the expression of the Reynolds stress anisotropy tensor $\mathbf{b}$ presented in Equation 1.19 can be rewritten as:

$$\begin{aligned} \mathbf{b} &= g^{(0)}(\lambda_1^*)\mathbf{T}^{*(0)} + g^{(1)}(\lambda_1^*)\mathbf{T}^{*(1)} + g^{(2)}(\lambda_1^*)\mathbf{T}^{*(2)} = \\ &= g^{(0)}(\alpha)\mathbf{T}^{*(0)} + g^{(1)}(\alpha)\mathbf{T}^{*(1)} + g^{(2)}(\alpha)\mathbf{T}^{*(2)} \end{aligned} \tag{1.25}$$

The first persistent ambiguity surrounding the application of Pope's approach to the

turbulent channel flow has been addressed in the study of *Cai et al.* [3]. It has been clarified that only one invariant and three tensors are necessary, and they solely depend on parameter α. Specifically, within the domain of interest, it has been observed that researchers attempted to apply a general model, as presented in equation 1.16, involving five invariants and ten tensors, for predicting the Reynolds stress anisotropy tensor in the turbulent channel flow using deep neural networks [5].

1.6. Generalized $\mathbf{T}^{*(0)}$

In this section it is presented a summary of the work of Cai *et al.* [3] on the generalized $\mathbf{T}^{*(0)}$ tensor, that is the theoretical basis of the model validation performed in this work.

Addressing another concern linked to the selection of the constant tensor $\mathbf{T}^{*(0)}$, an identification has been made. It has been discovered that two alternative permutations of $\mathbf{I}_2$ can yield $\mathbf{T}^{*(0)}$ configurations that also establish an integrity basis alongside $\mathbf{T}^{*(1)}$ and $\mathbf{T}^{*(2)}$. This assertion is evidenced by the following relations:

$$\mathbf{T}^{*(01)} = \mathrm{diag}(-1/3, 1/6, 1/6) = -\frac{1}{2}\mathbf{T}^{*(03)} - \frac{1}{4\lambda_1^*}\mathbf{T}^{*(2)}$$

and

$$\mathbf{T}^{*(02)} = \mathrm{diag}(1/6, -1/3, 1/6) = -\frac{1}{2}\mathbf{T}^{*(03)} + \frac{1}{4\lambda_1^*}\mathbf{T}^{*(2)}$$

Substituting Equation 1.24 into Equation 1.25, we obtain the expression of the Reynolds stress anisotropy tensor components as follows, the three systems of equations using $\mathbf{T}^{*(01)}$, $\mathbf{T}^{*(02)}$ and $\mathbf{T}^{*(03)}$, respectively:

$$\begin{cases} b_{11} = -\dfrac{1}{3}g^{(0)} - \dfrac{\alpha^2}{2}g^{(2)} \\ b_{12} = \dfrac{\alpha}{2}g^{(1)} \\ b_{22} = \dfrac{1}{6}g^{(0)} + \dfrac{\alpha^2}{2}g^{(2)} \\ b_{33} = \dfrac{1}{6}g^{(0)} \end{cases} \tag{1.26}$$

or

$$\begin{cases} b_{11} = \dfrac{1}{6}g^{(0)} - \dfrac{\alpha^2}{2}g^{(2)} \\ b_{12} = \dfrac{\alpha}{2}g^{(1)} \\ b_{22} = -\dfrac{1}{3}g^{(0)} + \dfrac{\alpha^2}{2}g^{(2)} \\ b_{33} = \dfrac{1}{6}g^{(0)} \end{cases} \tag{1.27}$$

or

$$\begin{cases} b_{11} = \dfrac{1}{6}g^{(0)} - \dfrac{\alpha^2}{2}g^{(2)} \\ b_{12} = \dfrac{\alpha}{2}g^{(1)} \\ b_{22} = \dfrac{1}{6}g^{(0)} + \dfrac{\alpha^2}{2}g^{(2)} \\ b_{33} = -\dfrac{1}{3}g^{(0)} \end{cases} \tag{1.28}$$

An arising question then concerns an optimal choice of $\mathbf{T}^{*(0)}$, among these three alternatives, in the context of statistical learning. One may observe from the b_{ij} values obtained from the DNS data [7, 10, 16] that $b_{12} \approx 0$ and $b_{22} \approx b_{33} \approx -b_{11}/2$ at the channel center. This holds for various $\mathrm{Re}_\tau = \dfrac{u_\tau h}{\nu}$ values, where $u_\tau = \sqrt{\nu \dfrac{d\bar{u}_1}{dx_2}}$. It turns out that only $\mathbf{T}^{*(01)}$ is proportional to the Reynolds stress anisotropy tensor at the channel center, and physically makes sense by including it in the basis tensors. To this end, a new generalized $\mathbf{T}^{*(0)}$ has been proposed by Cai *et al.*.

Indeed, to circumvent an arbitrary selection of one among the three potential $\mathbf{T}^{*(0)}$ configurations, a novel formulation for $\mathbf{T}^{*(0)}$ is proposed. This new expression generalizes $\mathbf{T}^{*(0)}$ as a linear combination of each of the alternative forms, denoted as $\mathbf{T}^{*(0)}_{\mathrm{gen}}$:

$$\mathbf{T}^{*(0)}_{\mathrm{gen}} = g_{01}\mathbf{T}^{*(01)} + g_{02}\mathbf{T}^{*(02)} + g_{03}\mathbf{T}^{*(03)} \tag{1.29}$$

where g_{01}, g_{02} and g_{03} are coefficient functions depending on α, instead of some fixed constants, in order to make the generalization as broad as possible under Pope's framework. A more concise formulation of Equation 1.29 can be obtained as:

$$\mathbf{T}^{*(0)}_{\text{gen}} = \begin{bmatrix} -\frac{1}{3}g_{01} + \frac{1}{6}g_{02} + \frac{1}{6}g_{03} & 0 & 0 \\ 0 & \frac{1}{6}g_{01} - \frac{1}{3}g_{02} + \frac{1}{6}g_{03} & 0 \\ 0 & 0 & \frac{1}{6}g_{01} + \frac{1}{6}g_{02} - \frac{1}{3}g_{03} \end{bmatrix} \tag{1.30}$$
$$= \begin{bmatrix} f_{01} & 0 & 0 \\ 0 & f_{02} & 0 \\ 0 & 0 & f_{03} \end{bmatrix}$$

where f_{01}, f_{02} and f_{03} are functions of α as g_{01}, g_{02} and g_{03} are, with $f_{01} + f_{02} + f_{03} = 0$ to preserve the zero-trace of the Reynolds stress anisotropy tensor.

It's interesting to mention that the information about α in $\mathbf{T}^{*(2)}$ is actually contained in $\mathbf{T}^{*(0)}_{\text{gen}}$. Because of this, one can get rid of $\mathbf{T}^{*(2)}$, and only needs to consider $\mathbf{T}^{*(0)}_{\text{gen}}$ and $\mathbf{T}^{*(1)}$ for the tensor basis. In the scenario of turbulent channel flow, the modified Equation 1.25 becomes:

$$\mathbf{b} = \mathbf{T}^{*(0)}_{\text{gen}}(\alpha) + g^{(1)}(\alpha)\mathbf{T}^{*(1)} \tag{1.31}$$

which can be developed into the following system of equations, giving the expression of $\mathbf{T}^{*(0)}_{\text{gen}}$ shown in Equation 1.30:

$$\begin{cases} b_{11} = f_{01} \\ b_{12} = \dfrac{\alpha}{2} g^{(1)} \\ b_{22} = f_{02} \\ b_{33} = -\left(f_{01} + f_{02}\right) \end{cases} \tag{1.32}$$

Consequently, four distinct representations of the Reynolds stress anisotropy tensor have been established: Equations 1.26, 1.27, 1.28, and 1.32. These formulations utilize either one of the three constant $\mathbf{T}^{*(0)}$ tensors or the newly introduced $\mathbf{T}^{*(0)}_{\text{gen}}$. Subsequently, the intention is to study each of these representations of Pope's model, examining the potential advantages of $\mathbf{T}^{*(0)}_{\text{gen}}$ in the model validation.

2 | Neural Networks in Turbulence Modeling

2.1. Introduction to Machine Learning approaches for turbulence modeling

In recent years, there has been a growing interest in the integration of machine learning techniques to enhance the development of Reynolds stress closures in fluid dynamics. Notably, Tracey *et al.* [21] employed kernel regression to construct models for the eigenvalues of Reynolds stress anisotropy. However, this method exhibited limitations in terms of generalization to new flow scenarios and scalability with extensive training data. Subsequently, Tracey, Duraisamy & Alonso [22] utilized neural networks featuring a solitary hidden layer to represent source terms originating from the Spalart-Allmaras RANS model. This application of neural networks demonstrated their proficiency in accurately reconstructing these source terms, thereby showing their potential utility in turbulence modeling. Another significant contribution was made by Zhang & Duraisamy [26], who employed neural networks to predict correction factors associated with turbulent production terms, influencing the magnitude, albeit not the anisotropy, of projected Reynolds stress tensors.

While certain attempts have been made to incorporate machine learning approaches in turbulence modeling, deep learning, specifically employing deep neural networks, emerges as an appealing alternative for Reynolds stress modeling. Deep neural networks constitute a subset of machine learning techniques wherein input features undergo transformation through multiple layers of nonlinear interactions. Despite their success in intricate problem domains, the integration of deep learning methodologies into turbulence modeling has been relatively limited. Previous efforts, such as those by Tracey *et al.* [21] and Zhang & Duraisamy [26], employed neural networks featuring only one or two hidden layers. Moreover, Milano & Koumoutsakos [15] leveraged neural networks with multiple hidden layers to replicate near-wall channel flows, although these neural networks were not em-

ployed as forward models for predicting turbulent flows. This study aims to establish that leveraging deep neural networks can yield enhanced Reynolds stress closures.

A salient advantage of neural networks lies in their architectural flexibility. Unlike random forests, which face challenges in preserving Galilean invariance for tensor quantities, neural networks can be easily adapted to enforce such invariance. Ling, Jones & Templeton [12] demonstrated the potential of neural networks in predicting Reynolds stress anisotropy eigenvalues by incorporating rotationally invariant input features. Their findings underscored the paramount significance of embedding invariance properties into the machine learning framework to achieve heightened predictive performance.

In the following subsections, the architecture of the most widely used Neural Networks in the domain of Turbulence modeling are presented. Subsequently, the neural networks employed for the validation subjects within this study will be introduced. Specifically, the neural networks for the low-Reynolds model will be elaborated upon in Section 2.2.

2.1.1. Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) represents the fundamental building block of the neural networks. Specifically, it embodies a densely interconnected architecture designed for feed-forward operations. The term "feed-forward" signifies the absence of feedback loops among the nodes, ensuring a unidirectional flow of information. Furthermore, the label "densely connected" implies that every node within a given layer maintains direct links with all nodes in the subsequent layer. As illustrated in Figure 2.1, an MLP assumes this structural configuration.

For instance, Fang *et al.* [5] employed the Fully-Connected Feed-Forward (FCFF) neural network, also known as Multi-Layer Perceptron (MLP), with five hidden layers and 50 nodes per layer. They used the normalized velocity gradient as a basis input to predict b_{ij} and attempted to incorporate more physical information into the model. They achieved the best performance by embedding a constraint function depending on y^+ in the model to enforce the no-slip boundary condition at the channel wall, and the Re_τ in the input feature, which also outperformed the TBNN.

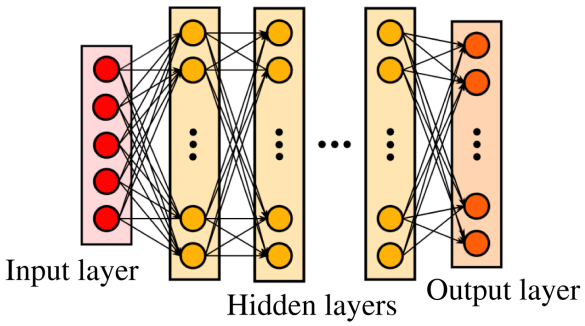


Figure 2.1: MLP architecture [13].

2.1.2. Convolutional Neural Network

Sáez de Ocáriz Borde *et al.* [20] designed a Convolutional Neural Network (CNN) architecture with five one-dimensional convolution layers to better capture non-local effects for the prediction of b_{ij}. Unlike earlier models which gave one-point predictions, the newly proposed CNN takes an array containing all the normalized velocity gradients as inputs and predicts an array of b_{ij} at all vertical locations. The same boundary condition enforcement and friction Reynolds number incorporation techniques proposed by Fang *et al.* [5] were used. This model yielded even better results when evaluating the R^2 score.

2.1.3. Tensor Basis Neural Networks

The Tensor Basis Neural Network (TBNN), designed by Ling *et al.* [13], is based on Pope's general model. This innovative architecture, as illustrated in Figure 2.2, incorporates two input layers: one for the invariants λ_i, λ_j and another one for the tensors $\mathbf{T}^{*(n)}$ for $n = 1, ..., 10$. The network consists of eight hidden layers, each with 30 nodes, which are used to learn the ten coefficient functions g^n in the final hidden layer. The output of this layer is then combined with the basis tensors input layer through element-wise multiplications to predict the Reynolds stress anisotropy tensor. The TBNN architecture ensures Galilean invariance and rotational invariance, as Pope's model does.

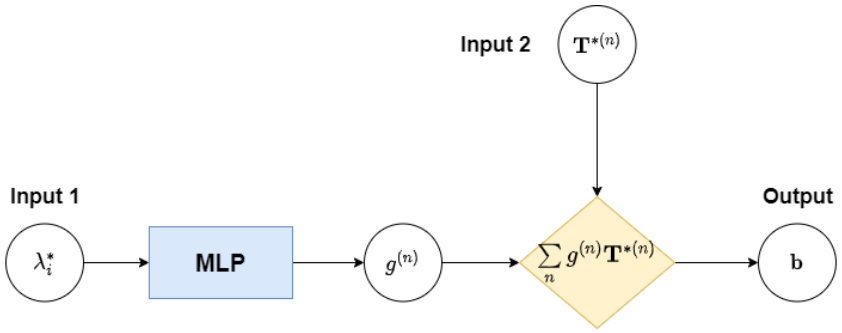


Figure 2.2: TBNN architecture.

Ling *et al.* [13] trained, validated, and tested the TBNN on a high-fidelity database comprising nine diverse flow cases. These ranged from duct flows and channel flows to a jet in a cross-flow for training, and from a wall-mounted cube in a cross-flow for validation to flow over a wavy wall for testing. Despite the diversity of the flow cases, the TBNN outperformed traditional RANS models and a generic neural network that did not incorporate invariance properties, both in terms of a priori predictions on the Reynolds stress anisotropy tensor and a posteriori results on mean velocity.

Inspired by the work of Ling *et al.*, several studies have been conducted to further explore and improve upon the TBNN. Zhang *et al.* [25] simplified the TBNN for the case of turbulent channel flow, focusing on predicting only one component of the Reynolds stress anisotropy tensor, b_{12}. They used a smaller network structure with four hidden layers and 20 nodes per layer to avoid overfitting. They introduced regularization and focused on variable selection, embedding the dimensionless wall distance $y^+ = \dfrac{yu_\tau}{\nu}$ in the input for better predictions in the near-wall region, and using only two invariants and three tensors considering the 2D nature of turbulent channel flow in a RANS modeling framework. These models were trained with Moser *et al.*'s DNS database [16] for channel flows at four turbulent Reynolds numbers, with the data set at $Re_\tau = 2000$ reserved for testing. These simplified and adapted TBNN models outperformed the original model in predicting b_{ij}.

2.2. Training of low-Reynolds number model Neural Network

In this section are presented the data and architecture of the neural networks trained by Cai *et al.* [3], which are the object of *a posteriori* validation in Section 5.3.

Despite these successes in predicting the Reynolds stress anisotropy tensor in the case of turbulent channel flow, there is still a need for a more profound physical explanation

regarding the selection of input features. In the current era of physics-based machine learning, it is crucial to understand not only why some models fail, but also why they succeed. To this end, Cai *et al.* extended previous studies on simpler MLP architectures with different combinations of input features used in the past, to understand the role of each feature in the neural networks.

Having previously clarified some persisting ambiguities concerning the application of Pope's model for turbulent channel flow in Section 1.6, Cai *et al.* examine the performance of the TBNN model for this specific case and propose the augmented TBNN architectures with additional input features other than Pope's representation.

Unlike previous studies on turbulent channel flow focusing only on b_{12} component, the aim is to provide predictions for all terms with nonzero statistics, that is b_{11}, b_{12}, b_{22} and b_{33}. Furthermore, models in previous studies were only evaluated at one turbulent Reynolds number Re_τ at a time, either in an interpolation case, which means that the tested Re_τ is within the range of the learning database, or on the contrary in an extrapolation case. In the present study, thanks to newly available DNS databases [7, 10], Cai *et al.* assess both interpolating and extrapolating predictability of the neural networks simultaneously. This step is challenging but critical since the prediction model should be accurate in both scenarios for practical use.

2.2.1. Data Set

The dataset utilized in their study comprises DNS data obtained at seven distinct frictional Reynolds numbers, namely Re_τ = [550, 1,000, 2,000, 4,000, 5,200, 8,000, 10,000], pertaining to turbulent channel flow [7, 10, 16]. Among these data corresponding to Re_τ = [550, 10,000] were exclusively reserved for the test set, while the data for Re_τ = 5,200 were randomly partitioned into 80% for test data and 20% for validation data. The remaining dataset was then randomly divided, allocating 80% for training data and reserving 20% for validation data. The test set was used to assess how well our neural network models can make predictions. This assessment covered scenarios where the models had to estimate values within known ranges (interpolation) and also predict values outside those ranges (extrapolation). An overview of the dataset sizes is provided in Table 2.1.

	550	1000	2000	4000	5200	8000	10000
Data size	191	255	383	1023	767	2047	1050
Reference	[16]	[16]	[16]	[10]	[16]	[10]	[7]

Table 2.1: Data size at each friction Reynolds number (Re_τ).

A visual representation of the data splitting process is depicted in Figure 2.3. The Neural Network is designed with three specific input features: firstly, the normalized parameter α, which according to Pope's model applied to turbulent channel flow, serves as the singular parameter on which b_{ij} relies. Additionally, the dimensionless wall distance y^+ and the turbulent Reynolds number Re_τ constitute the other two input parameters. The reason behind the selection of these features is discussed in Subsection 2.2.3. The targets for the learning process encompass all non-zero elements of the Reynolds stress anisotropy tensor $\mathbf{b}$, namely b_{11}, b_{12}, b_{22}, and b_{33}.

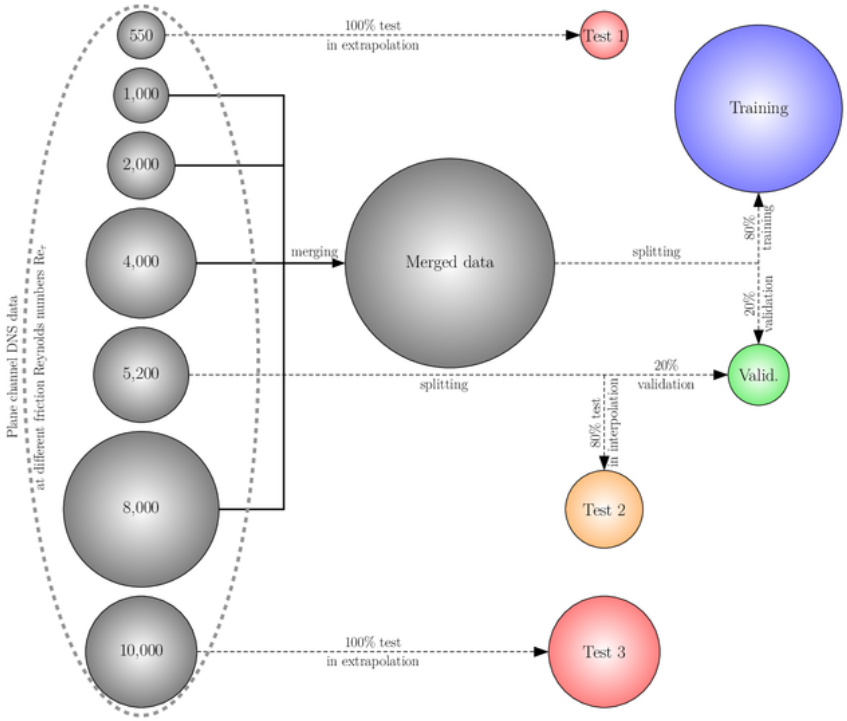


Figure 2.3: Diagram of data split and test process performed by Cai *et al.* [3]. The size of each bubble is proportional to the data size.

2.2.2. Pre-processing

Data quality significantly influences the performance of deep learning frameworks. During training, neural networks tend to assign greater importance to inputs with larger value ranges, particularly when significant differences in scales exist. This situation is less than ideal, as smaller inputs might also hold importance for predictions. As depicted in Figure 2.4a, two scale-related issues become apparent: firstly, the range of α is notably smaller compared to that of y^+ and Reτ; secondly, the distribution and range of y^+ varies across different Reτ values. This variation stems from the definition of y^+, which is given by:

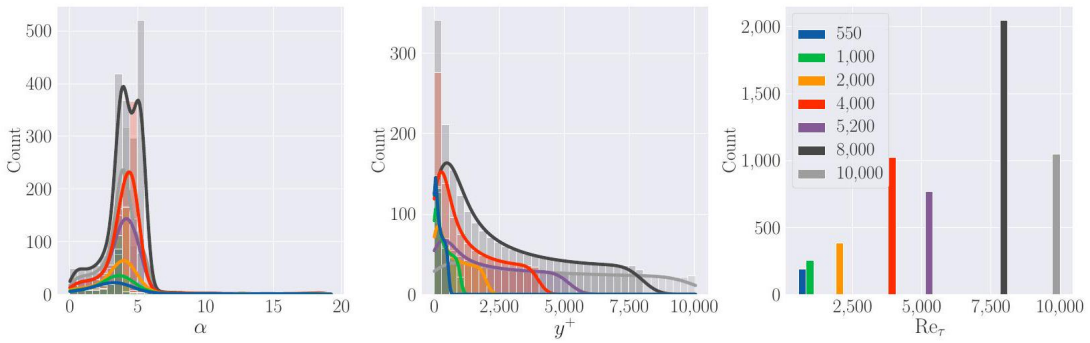
$$\mathrm{Re}_\tau = \max(y^+)$$

In order to tackle these concerns, it is necessary to pre-process the input data before providing it to the model. This increases the robustness of the neural network training. A widely used normalization method known as "max normalization" is utilized for this purpose. This technique involves dividing a feature by its maximum value. While this normalization is directly applied to α and Reτ, a log-transformation is initially applied to y^+ to mitigate the impact of the long tail distribution. Consequently, the pre-processed input features, denoted as $\widetilde{\alpha}$, $\widetilde{y^+}$, and $\widetilde{\mathrm{Re}\tau}$, are expressed as follows:

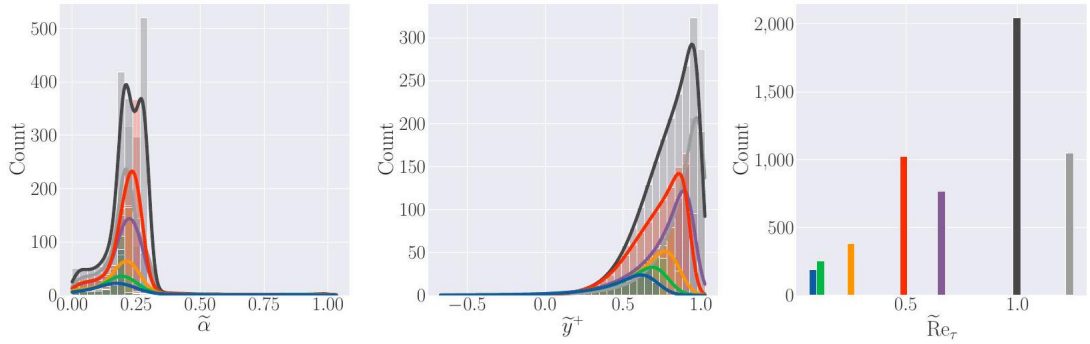
$$\begin{aligned}
\widetilde{\alpha} &= \frac{\alpha}{\max(\alpha)} \\
\widetilde{y^+} &= \frac{\log\left(y^+\right)}{\max\left(\log\left(y^+\right)\right)} \\
\widetilde{Re}_\tau &= \frac{\mathrm{Re}_\tau}{\max\left(\mathrm{Re}_\tau\right)}
\end{aligned}$$

where $\max(x)$ refers to the maximum value of x in the training set.

The distributions of the pre-processed input features are presented in Figure 2.4b, showcasing effective resolution of the earlier issues, especially since the three input features are rescaled to comparable ranges.



(a) Before pre-processing



(b) After pre-processing

Figure 2.4: Distribution of the input features before and after preprocessing, for various DNS experiments with different Re_τ [3].

On the contrary, it is generally preferable to normalize each regression target of a neural network separately, in order to prevent the dominance of any single target within the loss function. However, Cai *et al.* have chosen not to adopt this approach, with the intention of preserving the zero trace of the Reynolds stress tensor. Instead, they perform a global reduction based on the Frobenius norm of b_{ij}:

$$\sigma_{\mathbf{b}} = \sqrt{\frac{1}{m}\left[\sum_{k=1}^{3} b_{kk}^2 + b_{12}^2\right]}$$

$$\widetilde{\mathbf{b}} = \frac{\mathbf{b}}{\sigma_{\mathbf{b}}} \tag{2.1}$$

where m is the number of training samples.

2.2.3. Input parameters choice

The anisotropy components b_{ij}, obtained from DNS, are shown plotted against individual features post pre-processing in Figure 2.5. This representation reveals the limitations of Pope's model since it asserts that, in the context of turbulent channel flow, b_{ij} solely depends on α. However, a closer examination of the four sub-figures in the first row of Figure 2.5 reveals the presence of a multi-value phenomenon: b_{ij} does not exhibit a singular dependence on α. Instead, for a given α value, multiple b_{ij} values can arise. Furthermore, the four sub-figures in the second row of Figure 2.5 illustrate that b_{ij} exhibits some sensitivity to the turbulent Reynolds number, particularly when y^+ approaches its upper threshold, where this nuance, not accounted for in Pope's model, becomes evident.

As a result, to overcome these difficulties, it is necessary to include other representative input features into the deep neural networks in order to forecast b_{ij} accurately. Cai *et al.* rely on y^+ and Re_τ. Firstly, one can see from Figure 2.5 that b_{ij} are functions of y^+ at one given Re_τ; secondly, Re_τ was included as a classifier of data originating from flows with different turbulent levels.

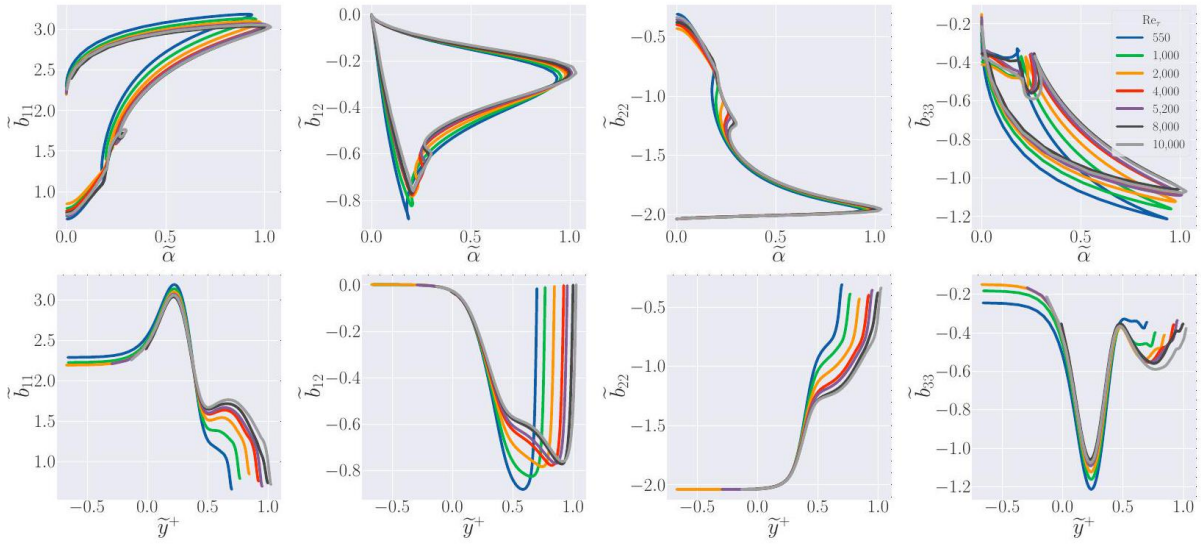


Figure 2.5: Visualization of $\widetilde{b}_{ij}$ as a function of $\widetilde{\alpha}$ and $\widetilde{y}^+$, for various DNS experiments with different Re_τ [3].

2.2.4. Neural networks

Cai *et al.* propose two different architectures of neural networks to fully predict the Reynolds stress anisotropy tensor for turbulent channel flow. In this work it is presented only one of these two, since the other one is not object of validation.

It is an augmented TBNN model specially designed for the channel flow configuration with only one invariant and three tensors, as previously clarified in Section 1.6. Two slightly different models are proposed: the first one using the three alternative constant $\mathbf{T}^{*(0)}$ and the other using the newly proposed generalized $\mathbf{T}^{*(0)}_{\text{gen}}$. In particular the Neural Networks called `Case 5`, `Case 6` and `Case 7` refer respectively to the choice of $\mathbf{T}^{*(01)}$, $\mathbf{T}^{*(02)}$, $\mathbf{T}^{*(03)}$ as $\mathbf{T}^{*(0)}$ tensor, while for the `Case 8` Neural Network, $\mathbf{T}^{*(0)}_{\text{gen}}$ is selected. As two new features are included into the augmented TBNN model apart from α, the expression of the Reynolds stress anisotropy tensor shown in Equations 1.25 and 1.31 can be reformulated as:

$$\mathbf{b} = g^{(0)}\left(\alpha, y^+, \mathrm{Re}_\tau\right)\mathbf{T}^{*(0)} + g^{(1)}\left(\alpha, y^+, \mathrm{Re}_\tau\right)\mathbf{T}^{*(1)} + g^{(2)}\left(\alpha, y^+, \mathrm{Re}_\tau\right)\mathbf{T}^{*(2)}$$

for the first model, and

$$\mathbf{b} = \mathbf{T}^{*(0)}_{\text{gen}}\left(\alpha, y^+, \mathrm{Re}_\tau\right) + g^{(1)}\left(\alpha, y^+, \mathrm{Re}_\tau\right)\mathbf{T}^{*(1)}$$

for the generalized one.

To form the Neural Network, Cai *et al.* use three hidden layers with 10 nodes per layer, which are activated by hyperbolic tangent function (tanh). The output layer contains three nodes for the three corresponding coefficient functions. Except for the output node of $g^{(1)}$ which is activated by the Softplus Linear Unit (SLU) to assure that the predicted $g^{(1)}$ is negative (as explained in Section 1.4), the others are linearly activated by default. Illustrations of these two augmented TBNN models can be seen in Figures 2.6a and 2.6b, respectively.

The model is implemented by calling an open-source library, named TensorFlow, in the language of Python. The loss function is defined as the Mean Squared Error (MSE) based on the Reynolds stress anisotropy tensor components:

$$\mathrm{MSE} = \frac{1}{4m}\sum_{i=1}^{m}\left[\sum_{k=1}^{3}\left(b_{kk} - \widehat{b}_{kk}\right)^2 + \left(b_{12} - \widehat{b}_{12}\right)^2\right]$$

where the predicted outputs are denoted with a hat.

As evaluation metrics, the coefficient of determination R^2 is used:

$$R^2 = 1 - \frac{\sum \left(y_i - \widehat{y}_i\right)^2}{\sum \left(y_i - \bar{Y}\right)^2}$$

where $\widehat{y}_i$ is the predicted i^{th} value, $\bar{y}_i$ is the actual i^{th} value and $\bar{Y}$ is the mean of the true values.

A weighted R^2 error is defined to evaluate the global predictive performance:

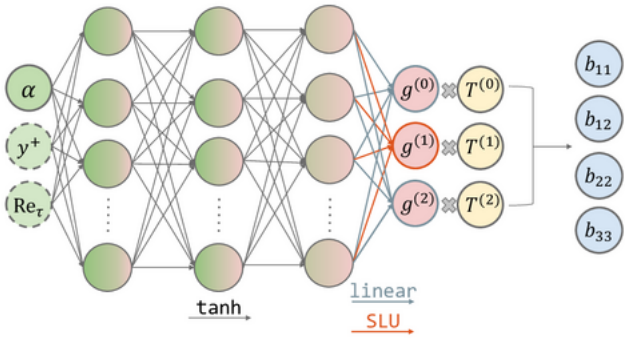
$$R^2_{\text{test}} = \frac{\sum m_{\text{test},i} \times R^2_{\text{test},i}}{\sum m_{\text{test},i}}$$

where $m_{\text{test},i}$ and $R^2_{\text{test},i}$ are respectively the size and the R^2 error of the i^{th} test set.

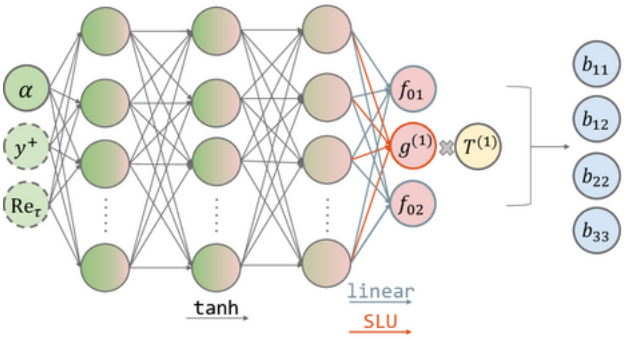
A summary of hyperparameter setting is shown in Table 2.2.

Hyperparameter	Value
Number of hidden layers	3
Number of nodes per hidden layer	10
Loss function	MSE
Optimization algorithm	Adam
Maximum learning rate	0.0001
Batch size	Truncated normal distribution
Weight initialization function	64

Table 2.2: Hyperparameter setting.



(a) Cases 5,6,7



(b) Case 8

Figure 2.6: Diagrams of the neural network architectures used in the work [3].

3 | Turbulence Models

This brief chapter aims to present from a theoretical point of view the three turbulence models object of the RANS simulations showcased in Chapter 5. In particular, the explicit treatment of the Reynolds tensor will be discussed in Section 3.1, while the two machine-learning based turbulence models are presented in Section 3.2: the High-Reynolds number neuronal model in Subsection 3.2.1 and the Low-Reynolds number neuronal model in Subsection 3.2.2. It has to be remarked that the two machine-learning based models work with an implicit treatment of the Reynolds tensor.

3.1. Explicit Treatment of the Reynolds tensor

The first model object of validation consists in the explicit treatment of the Reyolds tensor. The explicit treatment does not provide any model to estimate the value of the Reynolds tensor $\mathscr{R}_{ij} = \overline{u_i' u_j'}$. Instead, the values of $\overline{u_i' u_j'}$ are obtained directly from the DNS simulations [7, 10, 16]. Here are reported the equations resolved in this problem:

$$\begin{cases} \dfrac{\partial \bar{u}_i}{\partial x_i} = 0 & \text{(3.1a)} \\ \dfrac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \dfrac{\partial \bar{u}_i}{\partial x_j} = -\dfrac{1}{\rho}\dfrac{\partial \bar{p}}{\partial x_i} + \dfrac{\partial}{\partial x_j}\left[\nu\left(\dfrac{\partial \bar{u}_i}{\partial x_j} + \dfrac{\partial \bar{u}_j}{\partial x_i}\right)\right] + S & \text{(3.1b)} \end{cases}$$

One can observe that the Reynolds tensor is substituted by a source term, denoted as S. In order to solve the RANS problem, Equation 1.2, it has to be ensured that $S = -\dfrac{\partial}{\partial x_j}\overline{u_i' u_j'}$.

To do so, the terms of $\overline{u_i' u_j'}$ are firstly derived and summed on the j component as provided for by the Equation 3.1b. Afterwards, the three-components vector obtained in the previous step is injected in the solver as source term. One can observe that Equation 3.1b is in fact three distinct equation since i is intended to be equal to x, y and z. Therefore, one has that the first component of the source vector is added to the Equation 3.1b relative to the x coordinate, the second one to the equation relative to the y coordinate and the third one to the z one.

3.2. Implicit Treatment of the Reynolds tensor

The implicit treatment of the Reynolds tensor is based on the following definition of the Reynolds stress anisotropy tensor:

$$\mathbf{b} = -\frac{\nu_t}{k}\mathbf{S} + \mathbf{b}^{NL}$$

where ν_t is the turbulent viscosity and $\mathbf{S}$ the strain-rate tensor. One can observe that the non-linear term $\mathbf{b}^{NL}$ appears which was not there in the models based on the Boussinesq assumption, which is indeed a linear approximation. In the two models presented in this section, this term is computed by means of a neural network.

3.2.1. A High-Reynolds number neuronal model

The first implicit model investigated in this section is the Neuronal $k-\varepsilon$ model proposed by Angeli *et al.* [1]. This model is based on the Standard $k-\varepsilon$ model discussed in Subsection 1.3.2. Compared to the standard model, certain modifications are set up as a result of incorporating the Neural Network integration. In particular, this modifications concern not only the momentum equation (Equation 1.2b), but also the transport equations for turbulent kinetic energy k, Equation 1.10a and dissipation rate of turbulence ε, Equation 1.10b, due to the presence of turbulent viscosity and turbulent kinetic energy production. The Neuronal $k-\varepsilon$ model read as follows:

$$\begin{cases} \dfrac{\partial \bar{u}_i}{\partial x_i} = 0 & (3.2a) \\ \dfrac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \dfrac{\partial \bar{u}_i}{\partial x_j} = -\dfrac{1}{\rho}\dfrac{\partial \bar{p}}{\partial x_i} + \dfrac{\partial}{\partial x_j}\left[(\nu + \widetilde{\nu_t})\left(\dfrac{\partial \bar{u}_i}{\partial x_j} + \dfrac{\partial \bar{u}_j}{\partial x_i}\right) - 2k b_{ij}^{NL}\right] & (3.2b) \\ \dfrac{\partial k}{\partial t} + \bar{u}_i \dfrac{\partial k}{\partial x_i} = \dfrac{\partial}{\partial x_i}\left[\left(\nu + \dfrac{\widetilde{\nu_t}}{\sigma_k}\right)\dfrac{\partial k}{\partial x_i}\right] + \widetilde{\mathscr{P}} - \varepsilon & (3.2c) \\ \dfrac{\partial \varepsilon}{\partial t} + \bar{u}_i \dfrac{\partial \varepsilon}{\partial x_i} = \dfrac{\partial}{\partial x_i}\left[\left(\nu + \dfrac{\widetilde{\nu_t}}{\sigma_\varepsilon}\right)\dfrac{\partial \varepsilon}{\partial x_i} + \dfrac{\varepsilon}{k}\left(C_{\varepsilon_1}\widetilde{\mathscr{P}} - C_{\varepsilon_2}\varepsilon\right)\right]. & (3.2d) \\ \widetilde{\nu_t} = \widetilde{C_\mu}\dfrac{k^2}{\varepsilon} & (3.2e) \end{cases}$$

The terms $\widetilde{C_\mu}$, $\widetilde{\nu_t}$ and $\widetilde{\mathscr{P}}$ are expressed with the tilde to highlight the difference with respect to the standard model, while b_{ij}^{NL} is not expressed with a tilde since it is not found in the standard model. In Table 3.1 these modifications are displayed:

	b_{ij}^{NL}	C_μ	ν_t	$\mathscr{P}$
Standard $k-\varepsilon$	0	0.09	$0.09k^2/\varepsilon$	$2\nu_t S_{ij}S_{ij}$
Neuronal $k-\varepsilon$	$\sum_{k\neq 1} g_k T_{ij}^{*(k)}$	$-g_1$	$-g_1 k^2/\varepsilon$	$-2kb_{ij}S_{ij}$

Table 3.1: $k-\varepsilon$ Models: Differences between Standard and Neuronal.

The values of the empirical constants of the standard $k-\varepsilon$ model are maintained, they are indeed: $\sigma_k = 1.00$, $\sigma_\varepsilon = 1.30$, $C_{\varepsilon 1} = 1.44$ and $C_{\varepsilon 2} = 1.92$

3.2.2. A low-Reynolds number neuronal model

In this thesis, it is proposed a low-Reynolds number $k-\varepsilon$ model. The requirement for a low Reynolds number neural model arises from the intention to fully leverage the neural network trained by Cai *et al.*, presented in Section2.2, which is capable of predicting the Reynolds stress anisotropy tensor, $\mathbf{b}$, across the entire domain, including the near wall region. The rationale behind the definition of the model is to adapt the neuronal $k-\varepsilon$ model introduced by Angeli *et al.*, introduced in Subsection 3.2.1, to the Launder-Sharma low-Reynolds $k-\varepsilon$ model, discussed in Subsection 1.3.3.

$$
\begin{cases}
\dfrac{\partial \bar{u}_i}{\partial x_i} = 0 & \text{(3.3a)} \\
\dfrac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \dfrac{\partial \bar{u}_i}{\partial x_j} = -\dfrac{1}{\rho}\dfrac{\partial \bar{p}}{\partial x_i} + \dfrac{\partial}{\partial x_j}\left[(\nu + \widetilde{\nu}_t)\left(\dfrac{\partial \bar{u}_i}{\partial x_j} + \dfrac{\partial \bar{u}_j}{\partial x_i}\right) - 2k b_{ij}^{NL}\right] & \text{(3.3b)} \\
\dfrac{\partial k}{\partial t} + \bar{u}_i \dfrac{\partial k}{\partial x_i} = \dfrac{\partial}{\partial x_i}\left[\left(\nu + \dfrac{\widetilde{\nu}_t}{\sigma_k}\right)\dfrac{\partial k}{\partial x_i}\right] + \widetilde{\mathscr{P}} - \varepsilon - D & \text{(3.3c)} \\
\dfrac{\partial \varepsilon}{\partial t} + \bar{u}_i \dfrac{\partial \varepsilon}{\partial x_i} = \dfrac{\partial}{\partial x_i}\left[\left(\nu + \dfrac{\widetilde{\nu}_t}{\sigma_\varepsilon}\right)\dfrac{\partial \varepsilon}{\partial x_i} + \dfrac{\varepsilon}{k}\left(C_{\varepsilon_1} f_1 \widetilde{\mathscr{P}} - C_{\varepsilon_2} f_2 \varepsilon\right)\right] + \widetilde{E} & \text{(3.3d)} \\
\widetilde{\nu}_t = \widetilde{C_\mu}\dfrac{k^2}{\varepsilon} & \text{(3.3e)}
\end{cases}
$$

As in the previous neuronal model, the terms $\widetilde{C_\mu}$, $\widetilde{\nu}_t$ and $\widetilde{\mathscr{P}}$ are expressed with the tilde to highlight the difference with respect to the standard model, while b_{ij}^{NL} is not expressed with a tilde since it is not found in the standard model. Table 3.2 summarizes the differences between the classical low-Reynolds models and the model proposed above.

	b_{ij}^{NL}	C_μ	ν_t	$\mathscr{P}$
Low-Reynolds Classical Models	0	$0.09 f_\mu$	$0.09 f_\mu k^2/\varepsilon$	$2\nu_t S_{ij} S_{ij}$
Low-Reynolds Neuronal Model	$\sum_{k\neq 1} g_k T_{ij}^{*(k)}$	$-g_1$	$-g_1 k^2/\varepsilon$	$-2k b_{ij} S_{ij}$

Table 3.2: Low-Reynolds $k-\varepsilon$ Models: Differences between Classical and Neuronal.

Among the possible coefficients sets of the Low-Reynolds $k-\varepsilon$ models, the one of Jones and Launder has been chosen to perform this validation. The parameters σ_k, σ_ε, D, $C_{\varepsilon 1}$, $C_{\varepsilon 2}$, f_1, f_2 and Re_t are taken as in the Launder-Sharma model; E and ν_t keep the same formulas of the Launder-Sharma model but with their terms affected by the neural network; C_μ and $\mathbf{b}^{NL}$ are directly obtained from the neural network while f_μ is excluded from the model since the adaptation of the value of C_μ close to the wall is no more needed thanks to the neural network. The Equations 3.3 and the Table 3.3 present the Low-Reynolds number neuronal $k-\varepsilon$ model used to perform the validation.

$\widetilde{C_\mu}$	-g_1
σ_k	1.0
σ_ε	1.3
D	$2\,\nu \left(\frac{\partial \sqrt{k}}{\partial y}\right)^2$
$\widetilde{E}$	$2\,\nu\,\widetilde{\nu_t} \left(\frac{\partial^2 u}{\partial y^2}\right)^2$
C_{ε_1}	1.44
C_{ε_2}	1.92
f_μ	-
f_1	1.0
f_2	$1 - 0.3\exp\left(-Re_t^2\right)$
Re_t	$k^2/\nu\varepsilon$

Table 3.3: Low-Reynolds number Neuronal $k-\varepsilon$ Model Parameters Selection

In summary, the distinctions between this model and the High-Reynolds neuronal $k-\varepsilon$ model discussed in Subsection 3.2.1 can be synthetically described as follows. The present model incorporates additional terms to adjust values in the vicinity of the wall, specifically:

f_1, which is generally set to 1 in most widely-used models and thus does not significantly influence the results; f_2, D and $\widetilde{E}$. However, it is important to note that f_μ has no bearing in the low-Reynolds neuronal model, as C_μ is determined through the utilization of the Neural Network.

This model employs two distinct neural networks. To perform the subsequent validation using the low-Reynolds model, the neural networks presenting the most promising results in the *a priori* validation by Cai *et al.* [3] were selected. These networks are denoted as follows, maintaining the notation from Cai *et al.* discussed in Section 2.2: `Case5` which corresponds to the selection $\mathbf{T}^{*(0)} = \mathbf{T}^{*(01)}$, and `Case8` corresponding to $\mathbf{T}^{*(0)} = \mathbf{T}^{*(0)}_{\text{gen}}$.

4 | TRUST/TrioCFD code integration

4.1. TRUST/TrioCFD solver introduction

TRUST is an open-source software developed by the Thermohydraulics and Fluid Mechanics Department (STMF) of the Nuclear Energy Division at CEA. It is specifically designed for conducting fluid dynamics simulations on Linux environment. The genesis of this project dates back to 1994 when Trio_U was launched, the software kept this name until 2015 when it was split in TRUST e TrioCFD. Over time, TRUST has undergone extensive development, incorporating diverse numerical schemes, a parallel version, and turbulence models. It serves as the foundational framework for TrioCFD. The latter is used to conduct simulations for incompressible or quasi-compressible single-phase flows, as well as local-scale two-phase flows. Primarily tailored for application in the nuclear industry, TRUST plays a vital role in simulating flows within nuclear power plant cores.

Implemented in C++, TRUST follows an object-oriented paradigm that ensures inheritance and polymorphism relationships. The code consists of about 1600 classes, organized based on the nature of the problems considered or the type of discretization employed. Additionally, TRUST supports parallel processing on distributed memory systems through the use of MPI.

TRUST offers multiple spatial discretization methods, namely:

- Finite Differences Volumes (VDF) method
- Finite Elements Volumes (VEF) method
- PolyMAC method (arbitrary polygonal mesh)

The VDF method employs finite differences to approximate partial derivatives and assumes that the mesh conforms to a Cartesian grid. On the other hand, the VEF method relies on a variational approximation of the problem.

For temporal discretization, various schemes are available, including explicit, semi-implicit, and implicit schemes. The code also accommodates different boundary conditions, such as periodic boundary conditions and Neumann conditions. To solve the differential systems resulting from the discretization of the Navier-Stokes equations, TRUST provides iterative linear solvers (Conjugate Gradient, GMRES, etc.) as well as integrated solvers from the PETSc library.

To initiate computations, the user must create a data file defining the mesh, spatial discretization, temporal discretization, the nature of the problem and the post-processing requirements.

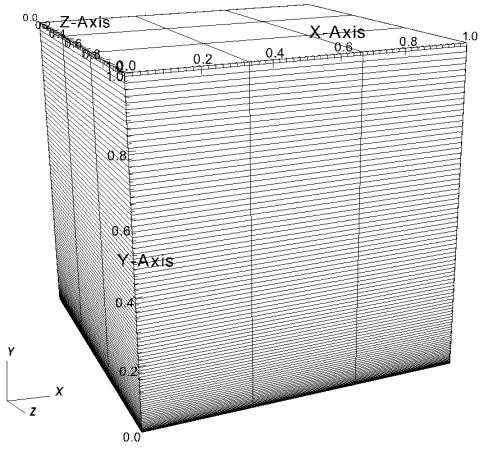
4.2. Plane Channel Problem

In this section it is discussed the problem setting of the plane channel from the point of view of the domain, its discretization and the RANS model employed to solve the problem.

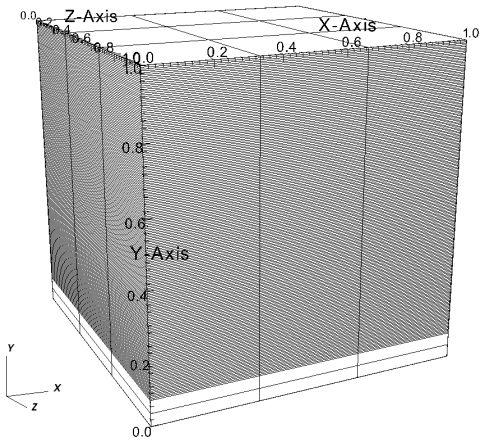
4.2.1. Domain Discretization and Boundary Conditions

The numerical domain used to represent the plane channel in this work is the cube. Indeed, by properly assign to every boundary a condition of periodicity, symmetry or wall one can obtain a numerical approximation of the infinite plane channel as represented in Figure 1.3. In particular, the conditions imposed to every face of the cube have to be the followings:

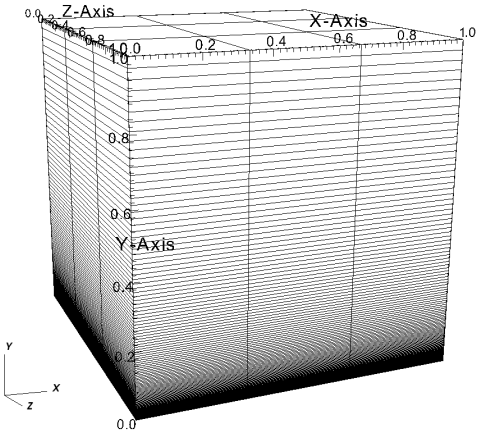
- Faces perpendicular to the x-axis: Periodicity, which means that the out-going flux on one face is imposed as in-going flux on the other face, in order to guarantee the infinite length of the channel along the x direction.
- Faces perpendicular to the y-axis: Wall conditions for the bottom face in order to impose the contact with the bottom wall of the channel; Symmetry conditions for the upper face in order to avoid replicating the domain in the upper half of the channel since it is identical to the lower one.
- Faces perpendicular to the z-axis: Periodicity in order to guarantee the infinite length of the channel along the z direction



(a) DNS mesh.



(b) High-Reynolds model mesh.



(c) Low-Reynolds model mesh.

Figure 4.1: Meshes of the simulations performed in this work for $Re_\tau = 180$, displayed thanks to the software VISIT.

In the case of explicit treatment of the Reynolds tensor, the domain refinement used corresponds to one used in the DNS simulations [7, 10, 16]. In Table 4.1 are presented the mesh refinements given by the DNS simulations for every problem studied in this work, i.e. every Re_τ. The choice to keep the same refinements is due to the fact that the values of the Reynolds tensor obtained from the DNS are given according to the DNS mesh presented in Figure 4.1a. Therefore, the same mesh has been kept, in order to avoid a possible source of error given by the interpolation of the values.

Re_τ	$N_x \times N_y \times N_z$
180	$4 \times 97 \times 4$
550	$4 \times 194 \times 4$
1000	$4 \times 257 \times 4$
2000	$4 \times 385 \times 4$
4000	$4 \times 1024 \times 4$
5200	$4 \times 767 \times 4$
8000	$4 \times 2047 \times 4$
10000	$4 \times 1051 \times 4$

Table 4.1: DNS mesh: Number of nodes along the three directions for every Re_τ.

For what concerns the high-Reynolds models, that are standard $k-\varepsilon$ model and neuronal $k-\varepsilon$ model, the mesh refinement has to be coherent with the presence of a wall function and therefore must respect the condition $y^+ \geq 30$ in the first cell to ensure that the first element includes the viscous sublayer as well as the buffer layer, as explained in Section 1.3. The mesh used for the high-Reynolds models is the one presented in Figure 4.1b. On the other hand, the low-Reynolds models, that is Launder-Sharma low-Reynolds $k-\varepsilon$ model and low-Reynolds neuronal $k-\varepsilon$ model, are forced to have a highly refined mesh, especially in the vicinity of the wall so that the condition $y^+ \leq 1$ in the first element is respected. For this reason this mesh looks similar to the one of the DNS simulations but in most of the treated cases the number of elements is however significantly smaller, contributing to a drop of the computational cost of the simulations. The mesh used for the low-Reynolds models is the one presented in Figure 4.1c.

4.2.2. Problem Definition

In the case of the explicit treatment of the Reynolds tensor, the data file of the problem is presented in Figure 4.2. In this file, the problem is defined with the keyword `Navier_Stokes_Standard` and not `Navier_Stokes_Turbulent`. This means that the equation solved in this problem are the RANS equations, 1.2, without the contribution of the Reynolds tensor $\mathscr{R}_{ij} = \overline{u_i' u_j'}$, because its value is equal to zero in case of absence of turbulence. Since every problem that has been studied presents turbulence, the contribution of this element of the equation is substituted by the source term `source_DNS_180.dat` that contains the information of the Reynolds tensor as explained in Section 3.1.

```
Navier_Stokes_standard
        {
                Projection_initiale       0
                Solveur_pression        PETSC Cholesky { }
                Convection                { quick }
                Diffusion                { }
                Conditions_initiales        { vitesse champ_fonc_xyz dom 3 3/2*y*(2-y) 0 0 }
                Conditions_limites        {
                                          periox        periodique
                                          perioz        periodique
                                          wall        paroi_fixe
                                          sym        symetrie
                                          }
                Sources                         {
                                          canal_perio { bord periox debit_impose 1 } ,
                                          source_qdm champ_som_lu_VDF dom 3 1e-6 source_DNS_180.dat
                                          }
        }
```

Figure 4.2: Explicit treatment of the Reynolds tensor: Data file of the problem definition.

In Figure 4.3 the data file of the Standard $k-\varepsilon$ model is shown. One can notice that the keyword of this problem is `"Navier_Stokes_turbulent"`, this means that the contribution of the Reynolds tensor is taken into account. For this purpose a turbulence model is required; in particular the $k-\varepsilon$ one is used.

In Figure 4.4 it is presented the data file of the low-Reynolds neuronal $k-\varepsilon$ model. One can remark two differences with respect to the data file of the Standard $k-\varepsilon$ model, Figure 4.3: on one side the model has to include the neural network; on the other side some features have to be adapted to the low-Reynolds model.

The changes due to the neural network are highlighted with the green color. In particular, the two keywords `tenseur_Reynolds_externe` have two different purposes:

- In the diffusion term the keyword is needed to compute the terms b_{ij}^{NL}, C_μ, ν_t and $\mathscr{P}$ as in Table 3.1, and not as the Standard case.
- In the "sources" term it aims to provide the name of the neural network used in the problem, in this case it is `Cas8`.

The modifications due to the Low-Reynolds model are highlighted with the red color. In particular:

- `with_nu` is set equal to "yes" in order to consider the contribution of the viscosity ν that plays a pivotal role in the near wall region. The default setting for this value is "no" because when the distance from the wall is significant, the value of ν becomes negligible compared to ν_t. Consequently, in cases where a wall function is applied, the contribution of ν is not significant.

```
Navier_Stokes_turbulent
{
    Solveur_pression        PETSC Cholesky { }
    Convection               { quick }
    Diffusion               { }
    Conditions_initiales       { vitesse champ_fonc_xyz dom 3 3/2*y*(2-y) 0 0 }
    Conditions_limites       {
                periox       periodique
                perioz       periodique
                wall        paroi_fixe
                sym        symetrie
                }
    Modele_turbulence k_epsilon
                {
                transport_k_epsilon
                {
                    convection               { amont }
                    diffusion               { }
                    conditions_initiales         { k_eps champ_uniforme 2 0.1 0.1 }

                    conditions_limites       {
                                periox       periodique
                                perioz       periodique
                                wall        paroi
                                sym        symetrie
                                }
                    sources                        { source_transport_k_eps { C1_eps 1.44 C2_eps 1.92 } }
                    }

                turbulence_paroi loi_standard_hydr
                }
    Sources                       { canal_perio { bord periox debit_impose 1 } ,
                                    canal_perio { bord perioz debit_impose 0 } }
}
```

Figure 4.3: Standard $k-\varepsilon$ model: Data file of the problem definition.

- `Modele_Fonc_Bas_Reynolds` is the keyword to set a low-Reynolds number model for the resolution of the problem, in this case the one proposed by Launder and Sharma is chosen, as discussed in Subsection 1.3.3.
- The keyword `turbulence_paroi` is responsible for the choice of the wall function. Since in the low-Reynolds number models there is no wall function, this keyword is set to be "negligible".

In the case of a neuronal model, that is Neuronal $k-\varepsilon$ model and low-Reynolds Neuronal $k-\varepsilon$ model, the keyword `"tenseur_Reynolds_externe"` it is added as one can see in Figure 4.4. Thanks to this keyword, the values of $\mathbf{b}^{NL}$, C_μ and $\mathscr{P}$ as in Table 3.1 and not as the Standard case. One can The name of the neural network used in the model is `CP3` for the Neuronal $k-\varepsilon$ model while varies from `Cas5` to `Cas8` included for the low-Reynolds Neuronal $k-\varepsilon$ model.

```
Navier_Stokes_turbulent
{
    Solveur_pression        PETSC Cholesky { }
    Convection                { quick }
    Diffusion                { tenseur_Reynolds_externe }
    Conditions_initiales        { vitesse champ_fonc_xyz dom 3 3/2*y*(2-y) 0 0 }
    Conditions_limites        {
                periox        periodique
                perioz        periodique
                wall        paroi_fixe
                sym        symetrie
                }
    Modele_turbulence k_epsilon {
                transport_k_epsilon
                {
                    with_nu                yes
                    convection                { quick }
                    diffusion                {  }
                    conditions_initiales        { k_eps champ_fonc_xyz dom 2 0.1 0.1 }

                    conditions_limites        {
                                periox        periodique
                                perioz        periodique
                                wall        paroi_fixe
                                sym        symetrie
                                }
                    sources{ source_transport_k_eps { C1_eps 1.44 C2_eps 1.92 }
                                                    }
                }
                Modele_Fonc_Bas_Reynolds Launder_Sharma { }
                k_min 1e-40
                turbulence_paroi negligeable
                }
    Sources                        { canal_perio { bord periox debit_impose 1 } ,
                      canal_perio { bord perioz debit_impose 0 } ,
                      tenseur_Reynolds_externe { nom_fichier Cas8 }  }
```

Figure 4.4: Low-Reynolds Neuronal $k-\varepsilon$ model: Data file of the problem definition.

It has to be noticed that while for the `CP3` neural network the code architecture was already fully implemented in the solver, for the neural networks associated to the low-Reynolds Neuronal $k-\varepsilon$ model various functions had to be adapted. This topic will be extensively discussed in Section 4.3.

4.3. Code Integration

The methodology of the neural networks training underwent significant modifications in transitioning from the `CP3` framework to the neural networks associated with the low-Reynolds neural $k-\varepsilon$ model. The work presented in this thesis fits into the context of this transition. For this reason, the main aim of this project is to integrate the TRUST/TrioCFD code so that it can treat low-Reynolds models based on neural networks. These alterations cover two principal branches. Firstly, due to the shift from a high-Reynolds regime to a low-Reynolds one, two extra variables, that are y^+ and Re_τ necessitated in-

corporation into the neural network architecture. Consequently, their computation had to be integrated in the class treating the neural network. Secondly, adaptations were introduced to the code to facilitate the execution of the neural network. This transition involved the adoption of a new library, `frugally-deep` [6], designed to streamline the use of the neural network within a C++ environment.

The code implementation consists of the modification of three classes in the TRUST/TrioCFD solver. They are the following:

- `PrePostNN` has the role of reading from an external file which variables have to be pre-treated and post-treated, and to get the relative pre- and post-treatment values.
- `TBNN` has the role of pre-treating the inputs, applying the neural network and post-treating the outputs.
- `Tenseur_Reynolds_Externe_VDF_Face` has the role of computing the tensor $\mathbf{b}^{NL}$ from the Post-treated neural network outputs and injecting $\mathbf{b}^{NL}$ into the rest of the code.

Every aspect of the code implementation is treated in detail in the following subsections.

Neural network inputs

As explained in Section 1.5, a low-Reynolds model requires more information than a high-Reynolds model. This is due to the impact that the quantities that depend on the distance from the wall have on the whole model. In particular, for the neural network used for the low-Reynolds model presented in this work, whose training is presented in Section 2.2, the quantities $y^+ = \dfrac{u_\tau y}{\nu}$ and $Re_\tau = \dfrac{u_\tau h}{\nu}$ are added as inputs of the neural network; where $u_\tau = \sqrt{\nu \left.\dfrac{d\bar{u}_1}{dx_2}\right|_{y=0}}$.

The framework of the class `Tenseur_Reynolds_Externe_VDF_Face` before the beginning of this work was thought only for high-Reynolds models, therefore the only quantity computed was the λ vector. In order to provide the values of the input quantities of the neural network presented in Section 2.2, α, y^+ and Re_τ have to be computed for every cell of the domain inside the class `Tenseur_Reynolds_Externe_VDF_Face`. Since the λ vector contained the information of α, it was effortless to adapt to the new model. For what concerns y^+ and Re_τ, they are computed as follows. As first, a `for` cycle is performed in order to compute $\left.\dfrac{d\bar{u}_1}{dx_2}\right|_{y=0}$ for each of the nine mesh columns, see Figure 4.1, and save the x and z values of the mesh element to keep track of the column. The code reads as

follows:

```
for ( int num_face=ndeb; num_face<nfin; num_face++)
            {
              elem_paroi = le_dom_VDF->face_voisins(num_face,1);

              xpx[num_face-ndeb] =le_dom_VDF->xp(elem_paroi, 0) ; // x=0
              xpz[num_face-ndeb] =le_dom_VDF->xp(elem_paroi, 2) ; // z=2


              y_maille_paroi =le_dom_VDF->xp(elem_paroi, 1) ;
              dudy_max = gij(elem_paroi,0,1,0);
              u_t = sqrt(nu* dudy_max);
              y_plus_wall[num_face-ndeb] = y_maille_paroi * u_t / nu;
            }
```

Listing 4.1: Wall Derivative

Successively, a `for` cycle on every element of the mesh is performed. For every element the value `y_plus_wall` refers to the element next to the wall of the same column, this is done by comparing the x and z values of the element with the ones of the nine elements next to the wall. The values of y^+ and Re_τ are therefore obtained from the value of y^+ on the first cell thanks to the linearity on y of the y^+ function and the definition of Re_τ: $Re_\tau = max(y^+) = y^+(y = h)$. In the code, `h_maille_paroi` represents the distance of the center of the first mesh element from the wall, while `h_elem` is the distance of the center of the current element from the wall. It has to be noticed that $h = 1.0$ according to the domain settings.

```
double Tenseur_Reynolds_Externe_VDF_Face::compute_y_plus( double y_plus_wall, double
    h_maille_paroi, double h_elem)
{
  double y_plus;
  y_plus = y_plus_wall / h_maille_paroi * h_elem;

  return y_plus;
}
```

Listing 4.2: Compute y^+.

```
double Tenseur_Reynolds_Externe_VDF_Face::compute_Re_t(double y_plus_wall, double
    h_maille_paroi)
{
  double Re_t;
  Re_t = y_plus_wall / h_maille_paroi * 1.0;

  return Re_t;
}
```

Listing 4.3: Compute Re_τ.

Pre- and post-processing values file

As a consequence of the changes in terms of input and output quantities of the neural network used to perform the validation of the low-Reynolds model, the pre- and post-processed quantities present the same modifications. The file where the pre- and post-processing values and functions are stored, in `.ppp` extension, keeps the same structure for the neural networks of high- and low-Reynolds models, therefore the only modification in the `PrePostNN` class consists of adapting the keywords of the `C++` function to read and store the new values. As an example, here it is presented the function implemented to read the pre-processing function of y^+ which is indeed the maximum of the logarithm of the value.

```
enum pp_y_plus PrePostNN::ReadPPYPlusFromLine(string buffer,string tag,size_t npos)
{
  string tmp;
  size_t ltag;
  enum pp_y_plus ret = INDEFY_PLUS;

  ltag = tag.length();
  tmp = buffer.substr(npos+ltag,buffer.length()-ltag);
  tmp.erase(remove(tmp.begin(), tmp.end(), ' '), tmp.end());
  if( tmp.compare("MAXLOG") == 0 ) ret = MAXLOG;

  return(ret);
}
```

Listing 4.4: Read the pre-treating function of y^+.

For what concerns the effective pre- and post-treatment of the values, some modifications had to be done in the `TBNN` class. In particular, the pre-treatment of y^+ is presented as an example.

```
void TBNN::process_y_plus(double y_plus)
{
  switch(_ppNN->get_ppy_plus())
  {
  case MAXLOG:

    if( _ppNN->get_y_plus_max_log() > 0 )
      _pp_y_plus = log10(y_plus) / _ppNN->get_y_plus_max_log();
    else
      _pp_y_plus = log10(y_plus);
    break;

  default:
    cerr << "Mauvaise methode de pre traitement des y_plus" << endl;
    break;
  }
}
```

Listing 4.5: Pre-treatment of the value of y^+.

Neural network upload

The neural network upload has radically changed due to the introduction of the library `frugally-deep`. Indeed, while the old uploading procedure relied on the upload of every feature of the neural network, with this new library the upload can be done in one command. The new uploading procedure is implemented in the `TBNN` class. It has to be observed that the library `frugally-deep` can not be included due to a conflict of class inclusions in another class of the solver. To upload the neural network two remaining options were therefore available: upload it in the `.cpp` every time that a prediction is needed, that is for every mesh element; otherwise the neural network could have been defined in the header file through a pointer to a class not defined in the header file but only in the implementation file. The second option have been considered more computationally costly since it allows uploading the neural network only once every time step and for every element for each time step. The upload and the prediction read as follows:

```
_model_uploaded = std::make_unique<fdeep::model>(fdeep::load_model( _model_file ));
```

Listing 4.6: neural network Upload.

```
const auto result = _model_uploaded->predict({ fdeep::tensor(fdeep::tensor_shape(
    static_cast<std::size_t>(3)), vector<float>{static_cast<float>(_pp_alpha),
    static_cast<float>(_pp_y_plus), static_cast<float>(_pp_Re_t)}) });

_g.resize(result[0].to_vector().size());
for (unsigned int i =0; i < result[0].to_vector().size(); i++)
    _g[i] = result[0].to_vector()[i];
_g[1] *= -1;
```

Listing 4.7: neural network Prediction.

One can observe that the value of g_1 is forced to change sign. This is due to the fact that at the time of the training of the neural network, in order to force g_1 not to change sign (it must be always negative since $g_1 = -C_\mu$), the SLU activation function was chosen for this node to guarantee the positivity of the output. In this way, the negative sign of g_1 is ensured by changing its sign.

Reynolds stress anisotropy tensor $\mathbf{b}^{NL}$

The structure of $\mathbf{b}^{NL}$ has deeply changed due to the introduction of the different versions of the tensor $\mathbf{T}^{*(0)}$ in the low-Reynolds model. The formula to compute the components of the $\mathbf{b}^{NL}$ tensor depends on the neural network used. For sake of simplicity, in the code the values assigned to the components of the tensor depend on the name of the neural network used, i.e. `Cas5`, `Cas6`, *etc.* .

As an example, the definition of the $\mathbf{b}^{NL}$ tensor in the `Cas5` neural network reads as follows in the `TBNN` class:

```
_pb[1] = 0.0;
_pb[2] = 0.0;
_pb[4] = 0.0;

if (_model_file.find("Cas5") != string::npos) {

  _pb[0] = -1.0 / 3.0 *_g[0] - 0.5* _pp_alpha*_pp_alpha*_g[2];
  _pb[3] = 1.0 / 6.0 *_g[0] + 0.5* _pp_alpha*_pp_alpha*_g[2];
  _pb[5] = 1.0 / 6.0 *_g[0];
}

for(unsigned int i=0;i<nbb;i++)
  _b[i] = _ppNN->get_bsigma() * _pb[i];
```

Listing 4.8: $\mathbf{b}^{NL}$ computation with `Cas5` neural network.

One can notice that the post-treatment of $\mathbf{b}^{NL}$ is the one presented in Equation 2.1. The `b` vector computed in `TBNN` is assigned to the `Tenseur_Reynolds_Externe_VDF_Face` table containing the values of the `b` matrix for every element as it follows:

```
    b = tbnn->predict(alpha, y_plus, Re_t);

    resu(elem,0,0) = b[0];
    resu(elem,0,1) = b[1];
    resu(elem,0,2) = b[2];
    resu(elem,1,0) = b[1];
    resu(elem,1,1) = b[3];
    resu(elem,1,2) = b[4];
    resu(elem,2,0) = b[2];
    resu(elem,2,1) = b[4];
    resu(elem,2,2) = b[5];
```

Listing 4.9: $\mathbf{b}^{NL}$ assignement.

C_μ computation

In order to compute the value of C_μ, the output of the neural network g_1 must be post-treated. The most effective way to do it, since no post-treating parameters are available for g_1, is to use the definition of b_{12}. Indeed, by post-treating b_{12} as in Equation 2.1 and using the value of α, one can invert the formula of b_{12} in the Equations 1.26 and followings in order to obtain the value of g_1. This computation is performed in a dedicated function in the `TBNN` class, and reads as follows:

```
double TBNN::get_g1(double b12, double alpha)
{
  return 2 * b12/ alpha; //-c_mu
}
```

Listing 4.10: g_1 computation.

Production term

The production term is computed in the `Calcul_Production_K_VDF` class. As explained in Table 3.1, the old computation of the production term, $\mathscr{P} = 2\nu_t S_{ij} S_{ij}$ is subject to a change. For this reason, a new function called `calculer_terme_production_K_neuronal` in the `Calcul_Production_K_VDF` class has been created. Thanks to this function, the production term can now be computed as $\mathscr{P} = -2kb_{ij}S_{ij}$.

```
DoubleVect& Calcul_Production_K_VDF::calculer_terme_production_K_neuronal(const
    Domaine_VDF& domaine_VDF, const Domaine_Cl_VDF& domaine_Cl_VDF , DoubleVect& S,
    const DoubleTab& K_eps, const DoubleTab& vitesse,const Champ_Face_VDF& vit,  const
    DoubleTab& visco_turb, const DoubleTab& bij )  const
{
  int nb_elem = domaine_VDF.domaine().nb_elem();
  int elem;
  S = 0. ;
  DoubleTab gij(nb_elem,Objet_U::dimension,Objet_U::dimension, vitesse.line_size());
  vit.calcul_duidxj( vitesse,gij,domaine_Cl_VDF );
  for ( elem=0; elem<nb_elem; elem++)
    {
      //P= -2*k*bij*Sij
      for (int i=0; i<Objet_U::dimension; i++)
        for (int j=0; j<Objet_U::dimension; j++)

          S(elem) -= K_eps(elem,0)*bij(elem,i,j)*( gij(elem,i,j,0) + gij(elem,j,i,0));


      file_ecrire_prod << domaine_VDF.xp(elem, 1) << " " << S(elem) << std::endl;
    }
  return S ;
}
```

Listing 4.11: Production term computation.

5 | Results validation

This chapter presents the outcomes of the validation process, where various case studies were employed to assess the accuracy and reliability of the proposed models. Section 5.1 describes the preliminary validation procedure conducted by directly injecting the Reynolds Tensor obtained from the DNS simulation into the RANS equations. Subsequently, Section 5.2 presents the results of the analysis of the high-Reynolds neuronal $k-\varepsilon$ model. Finally, in Section 5.3, the validation of the low-Reynolds turbulence model is showcased, considering the two most promising selections of the tensor $\mathbf{T}^{*(0)}$, that is $\mathbf{T}^{*(01)}$ and $\mathbf{T}^{*(0)}_{\text{gen}}$. For the matter of readability, for every model studied in this work the plots of the velocity profiles are reported only for four problems out of eight studied. The four most representative problems selected to be displayed are the ones corresponding to the values of the friction Reynolds number, such that $Re_\tau = [550, 2000, 4000, 5200]$. This choice comes from the will to display two cases in interpolation, that is $Re_\tau = [2000, 4000]$, and two in extrapolation, that is $Re_\tau = [2000, 4000]$, with respect to the training of the neural networks used for the low-Reynolds neuronal model (see Section 2.2). The velocity profile of every problem studied are displayed in Appendix A.

Eight problems

Eight case studies are performed in this work. The geometry and the settings of the problems are described in Section 1.5 and 4.2. The only parameter that differs from one case to the other is the viscosity, ν, this yields to a difference also in the friction Reynolds number since it depends on ν. In Table 5.1 the viscosity values relative to the different Re_τ are showcased. The variety of cases presented in this work allows drawing more precise conclusion based on more data.

Re_τ	ν
180	$3.5 \cdot 10^{-4}$
550	$1.0 \cdot 10^{-4}$
1000	$5.0 \cdot 10^{-5}$
2000	$2.3 \cdot 10^{-5}$
4000	$1.06945 \cdot 10^{-5}$
5200	$8.0 \cdot 10^{-6}$
8000	$5.00814 \cdot 10^{-6}$
10000	$3.88613 \cdot 10^{-6}$

Table 5.1: Viscosity values, ν, relative to every Re_τ.

Error definition

To evaluate the accuracy of the results, two different metrics are used in this work. The error is always intended as the difference between the velocity profile obtained from the RANS simulation and the one obtained from the DNS. For the plane channel problem, the only non-zero component of the velocity is along the x-axis and depends only on y, thus it is noted as $U(y)$. The first metric, called E_q, computes the squared relative error on the entire domain and reads as follows:

$$E_q = \frac{1}{h}\sqrt{\sum_{i=1}^{N_y-1}\left(\frac{U^{DNS}(y_i) - U^{RANS}(y_i)}{U^{DNS}(y_i)}\right)^2 (y_{i+1} - y_i)} \tag{5.1}$$

where h is half of the height of the canal, y_i is the y coordinate of the node i, N_y is equal to the number of nodes of the mesh along the y direction. The other metric aims to evaluate the maximal relative error and reads as follows:

$$E_{max} = max\left(\frac{\|U^{DNS}(y_i) - U^{RANS}(y_i)\|}{\|U^{DNS}(y_i)\|}\right) \tag{5.2}$$

5.1. Explicit treatment of the Reynolds tensor

As first, it is conducted a RANS simulation utilizing the Reynolds tensor acquired from DNS [7, 10, 16]. The results presented in this section refer to the model setting elaborated in Section 3.1. This preliminary result has been performed in order to demonstrate the

inadequacy of the explicit treatment of the Reynolds tensor to predict the velocity profile. Indeed, Wu *et al.* [24] proved that due to the ill-conditioning of the RANS equations, even a minor error less than 0.5% in the Reynolds stress tensor can lead to variations up to 35% in the velocity profile.

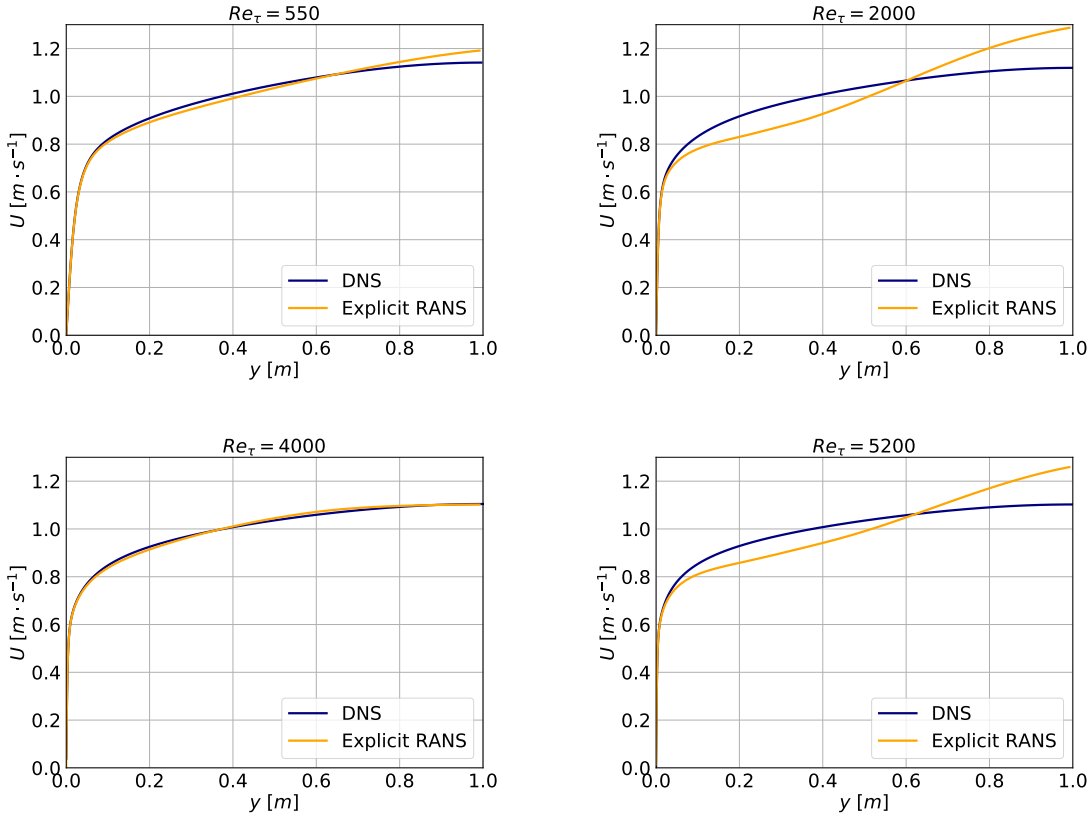


Figure 5.1: Superposition of Velocity Profiles for $Re_\tau = [550, 2000, 4000, 5200]$: RANS with Explicit Treatment of the Reynolds Tensor vs. DNS.

The plots exhibited in Figure 5.1 align with Wu *et al.*'s findings for two reasons. As first, it is evident that while the Reynolds tensor, acquired through DNS, is reliable and precise, there exists a notable discrepancy in the velocity profile. Moreover, it can be noticed in Table 5.2 that the velocity profiles of the explicit treatment of the Reynolds tensor tend to be less accurate while Re_τ grows, notably for what concerns E_q. This result represents the second result in accordance to Wu *et al.* that state that the RANS equations become the more and more ill-conditioned with growing values of Re_τ. The case $Re_\tau = 4000$ represents a case against the trend, since its velocity profile turns out to be very accurate. A possible explanation can be drawn by observing that a more refined mesh grid can lead to a more accurate result of the Reynolds tensor. Indeed, considering that the refinement

required has to grow with Re_τ, for cases $Re_\tau = [4000, 8000]$ Moser *et al.* [16] used a mesh grid that was more refined than for the other cases (see Table 4.1).

Error Analysis

	180	**550**	**1000**	**2000**	**4000**	**5200**	**8000**	**10000**
E_q	0.012	0.0185	0.0197	0.0801	0.0082	0.0687	0.0297	0.0722
E_{max}	0.4157	0.1153	0.0684	0.1481	0.0856	0.141	0.0688	0.2311

Table 5.2: Explicit treatment: E_q and E_{max} for every Re_τ.

To conclude, it has been proven that the explicit treatment of the Reynolds tensor does not represent a reliable method to obtain accurate velocity profiles. Therefore, in order to provide a model able to produce solid results in terms of velocity profile, the implicit treatment of the Reynolds tensor has been employed in this work. The validation of the two implicit models studied is presented in the following sections.

5.2. Validation of the high-Reynolds neuronal $k - \varepsilon$ model

In this section, it is presented the validation of the high-Reynolds neuronal $k - \varepsilon$ model, presented in Subsection 3.2.1. In particular, the results of this neuronal model are compared with the results obtained from the standard $k-\varepsilon$ model, both from the point of view of the accuracy of the results and the computational cost. The analysis of this model was thought as a preliminary result of this thesis, since its validation was already performed in [8]. However, while performing this study, an error in the code TRUST/TrioCFD has been found and solved, therefore a new validation has been performed.

Grid refinement

This neuronal model is based on the neural networks able to predict the flow only outside the near-wall region. For this reason, the training of the neural network has been performed only in a region far from the wall. In particular, since the lowest Re_τ that has been used to perform the training was $Re_\tau = 1000$, the region taken into account to train the neural network was $0.07996 \leq y \leq 1$. One can indeed observe that for $Re_\tau = 1000$, the value of y^+ at $y = 0.07996$ is greater than 30, in particular it is 42.9 . The condition $y^+ \geq 30$ is therefore automatically verified for every $Re_\tau \geq 1000$ in the region

$0.07996 \leq y \leq 1$, but it had to be proven *a posteriori* for the cases $Re_\tau = [180, 550]$. In Table 5.3 it is displayed the result of this study. One can observe that the region of the wall function, that is the first cell, it has to be expanded in the first two cases in order to verify the necessary condition of a high-Reynolds model: $y^+ \geq 30$.

Re_τ	y	y^+
180	0.35	35.0
550	0.15	35.2
1000	0.07996	42.9
2000	0.07996	79.7
4000	0.07996	154.4
5200	0.07996	191.6
8000	0.07996	301.8
10000	0.07996	338.1

Table 5.3: High-Reynolds models: first element height, y, and dimensionless height, y^+, relative to every Re_τ.

Velocity profiles

The velocity profiles of the RANS simulation performed with the high-Reynolds neuronal model are displayed in Figure 5.2. These profiles are compared with the ones of the DNS and of the standard $k-\varepsilon$ model. It can be observed that for $Re_\tau = [2000, 4000, 5200]$ the velocity profiles of the two $k-\varepsilon$ are comparable with the one of the DNS. In particular, the standard model is more able to capture the velocity profile of the DNS in the region $0.07996 \leq y \leq 0.5$ while the neuronal one performs better in the region $0.5 \leq y \leq 1$. For what concerns the case $Re_\tau = 550$, which is in extrapolation with respect to the neural network training, it can be noticed that while the neuronal model captures the shape of the profile of the DNS but never reaches its values, the standard model provides a profile which is more adherent to the one of the DNS, especially for $y \geq 0.3$.

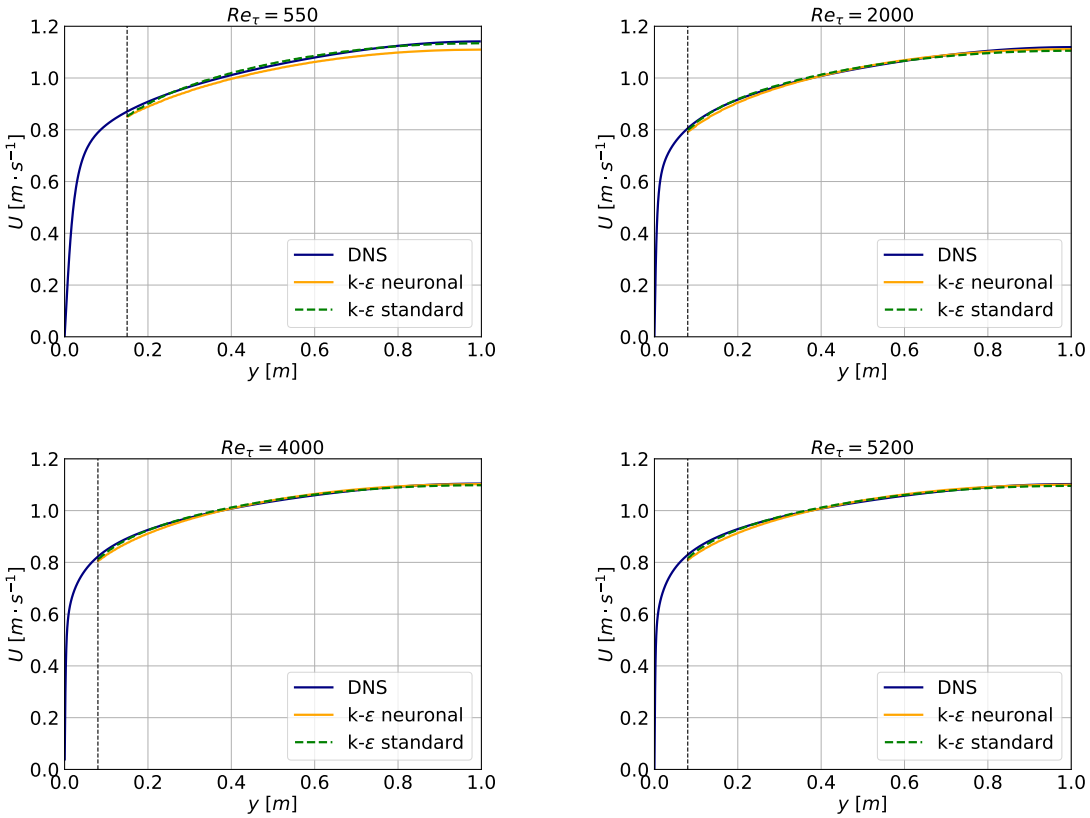
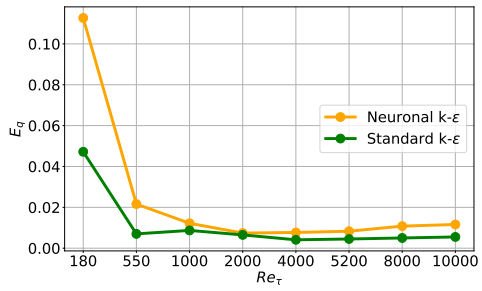


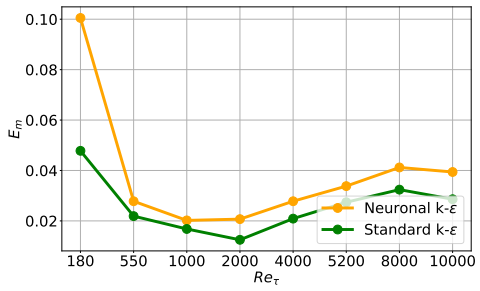
Figure 5.2: Superposition of Velocity Profiles for every Re_τ: RANS with high-Reynolds $k-\varepsilon$ models vs. DNS.

Error analysis

In Figure 5.3 are presented the results of the error comparison between the standard $k-\varepsilon$ model and the neuronal $k-\varepsilon$ model. One can observe that the neuronal $k-\varepsilon$ model does not outperform the standard $k-\varepsilon$ model in any of the two metrics. For $Re_\tau = 180$ any of the two models is able to provide a good estimate of the velocity profile; for the other cases presented in this work the neuronal model approaches the standard one in terms of quadratic error and maximal error.



(a) E_q .



(b) E_{max} .

Figure 5.3: Standard and neuronal $k-\varepsilon$ models: E_q and E_{max} for every Re_τ.

Computational cost

In Table 5.4 the time needed to reach the convergence is showcased. One can observe that also for what concerns the computational cost, the standard model outperforms the neuronal one. Namely, the time required from the simulation is more than double for every Re_τ studied.

Convergence Time

	180	**550**	**1000**	**2000**	**4000**	**5200**	**8000**	**10000**
Standard $k-\varepsilon$	24	128	38	123	476	1020	1857	1858
Neuronal $k-\varepsilon$	81	322	84	245	1328	2215	3806	3910

Table 5.4: Convergence time: Comparison between different high-Reynolds models for every Re_τ. Times are expressed in seconds.

5.3. Validation of the low-Reynolds neuronal $k-\varepsilon$ model

In this section, it is presented the validation of the low-Reynolds neuronal model proposed in Subsection 3.2.2. This model is based on the neural networks proposed by Cai *et al.* [3] and discussed in Section 2.2. In their work, a validation of the anisotropic Reynolds tensor **b** was already performed. The aim of the present work is to exhibit the results of the *a posteriori* validation, this is after the RANS simulation, of the low-Reynolds neuronal model.

5.3.1. Grid Independence

The first step in validating the model involved performing a grid independence study. This study aims to identify the least refined mesh that ensures that the results remain unaffected by its excessive coarseness, in order to minimize the computational cost.

To determine the optimal mesh refinement, various simulations were conducted for each Re_τ value. Each simulation differed solely in the number of mesh elements along the y direction of the domain, ranging from 75 to 250 elements, with an increment of 25 elements for each test.

The grid independence analysis includes both qualitative and quantitative assessments. For the quantitative aspect, the quadratic error, Equation 5.1, was plotted against the mesh grid resolution on the x-axis. The choice of the quadratic error metric was motivated by its ability to facilitate a quantitative comparison between similar velocity profiles. To consider the qualitative aspect and avoid relying solely on quantitative value congruence for similarity, it was crucial to evaluate how closely the velocity profiles resembled each other across different mesh refinements in identical simulations.

By combining the qualitative and quantitative analyses, the mesh refinement that ensures independence for each Re_τ value was identified and marked with a magenta dot on each graph in Figure A.3. For matter of readability, the values of the mesh refinement are also reported in Table 5.5. Together with the selected number of nodes for every Re_τ it is reported also the scale factor chosen for the mesh refinement. This value, if different from 1, allows changing the size of every element while keeping the number of elements fixed. In particular, a scale factor greater than 1 yields to a more refined mesh in the near-wall region and a coarser one in the middle of the channel, while a scale factor smaller than 1 refines the mesh more in the middle of the channel. For the low-Reynolds models the main concern is to refine the mesh in the near-wall region, for this reason a scale factor greater than one has been selected. The default choice of 1.03 has been relaxed to 1.02 for the case $Re_\tau = 180$ since the gradients in the near-wall region are not as high as in the other cases, therefore a high refinement was not needed. The condition $y^+ \leq 1$ has been verified *a posteriori* for every case.

Re_τ	$N_x \times N_y \times N_z$	Scale Factor
180	$4 \times 126 \times 4$	1.02
550	$4 \times 151 \times 4$	1.03
1000	$4 \times 176 \times 4$	1.03
2000	$4 \times 201 \times 4$	1.03
4000	$4 \times 226 \times 4$	1.03
5200	$4 \times 226 \times 4$	1.03
8000	$4 \times 226 \times 4$	1.03
10000	$4 \times 226 \times 4$	1.03

Table 5.5: Low-Reynolds models mesh: Number of nodes along the three directions for every Re_τ.

5.3.2. Results

The results of the validation are separated in two parts: the analysis of the error of the velocity profile compared to the one of the DNS and computational time required to reach the convergence of the simulation. The aim of this study is to understand whether the model proposed outperforms the existing model in terms of exactness of the RANS solution and computational cost. As in the previous sections, the graphs reported concern the problems with $Re_\tau = [550, 2000, 4000, 5200]$; the graphs of the other problems can be seen in Appendix A. Among the four neural network proposed in the work of Cai *et al.* that could have been used for the low-Reynolds neural model validated in this section, only the neural network `Case5` and `Case8` are the object of study. This choice is due to the fact that in the validation of the anisotropy Reynolds tensor performed in the work of Cai *et al.* these two neural networks had the most promising results.

Velocity Profile

The analysis of the validation begins with the neural network `Case 5`, that is with the tensor $\mathbf{T}^{*(0)} = \mathbf{T}^{*(01)}$. The velocity profile of this RANS simulation is presented in Figure 5.4. One can observe that as the friction Reynolds number Re_τ increases, the Launder-Sharma model provides a more accurate velocity profile. This is due to the nature of the model, which is design specifically for the highly turbulent plane channel case. The better performance of the Launder-Sharma model goes together with a better accuracy of the low-Reynolds neuronal model performed with the `Case 5` neural network, as it can be

seen for cases $Re_\tau = [550, 2000, 4000, 5200]$ in Figure 5.4. The velocity profile for every Re_τ performed with this neural network are showcased in Figure A.4.

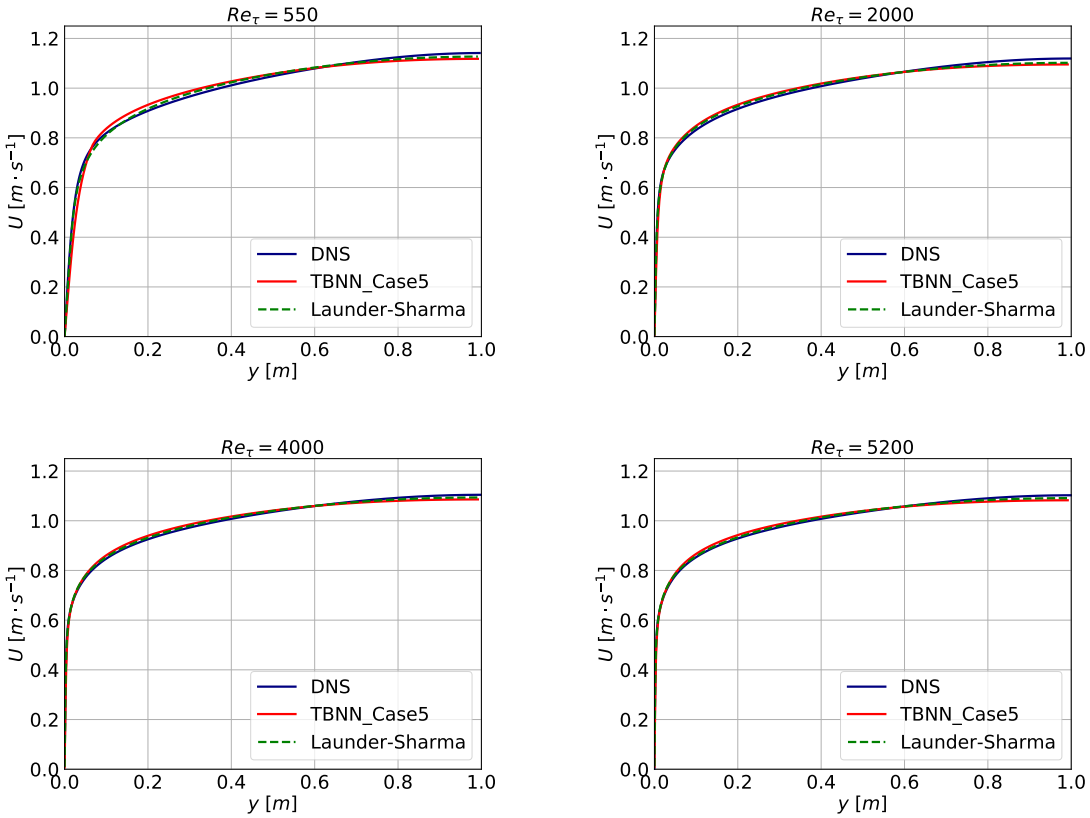


Figure 5.4: Superposition of Velocity Profiles for $Re_\tau = [550, 2000, 4000, 5200]$: RANS with case 5 neural network vs. DNS Simulations.

In Figure 5.5 the velocity profiles of the low-Reynolds neuronal model based on the `Case 8` neural network is compared to the ones coming from the DNS and the Launder-Sharma model. One can observe that this neuronal model provides a velocity profile similar to the one based on the `Case 5` neural network. Indeed, in both cases one can observe that while the Launder-Sharma model provides a slightly greater velocity in the region $0.2 \leq y \leq 0.5$ and a slightly lower one in the region $0.5 \leq y \leq 1.0$, the neural method, which is based on the Launder-Sharma one, emphasizes this same error. This behavior of the Launder-Sharma model is in accordance to the observation regarding the better performance with highly turbulent problems done previously, since for high values of Re_τ the velocity profile tends to be higher in the region $0.2 \leq y \leq 0.5$ and lower in $0.5 \leq y \leq 1.0$.

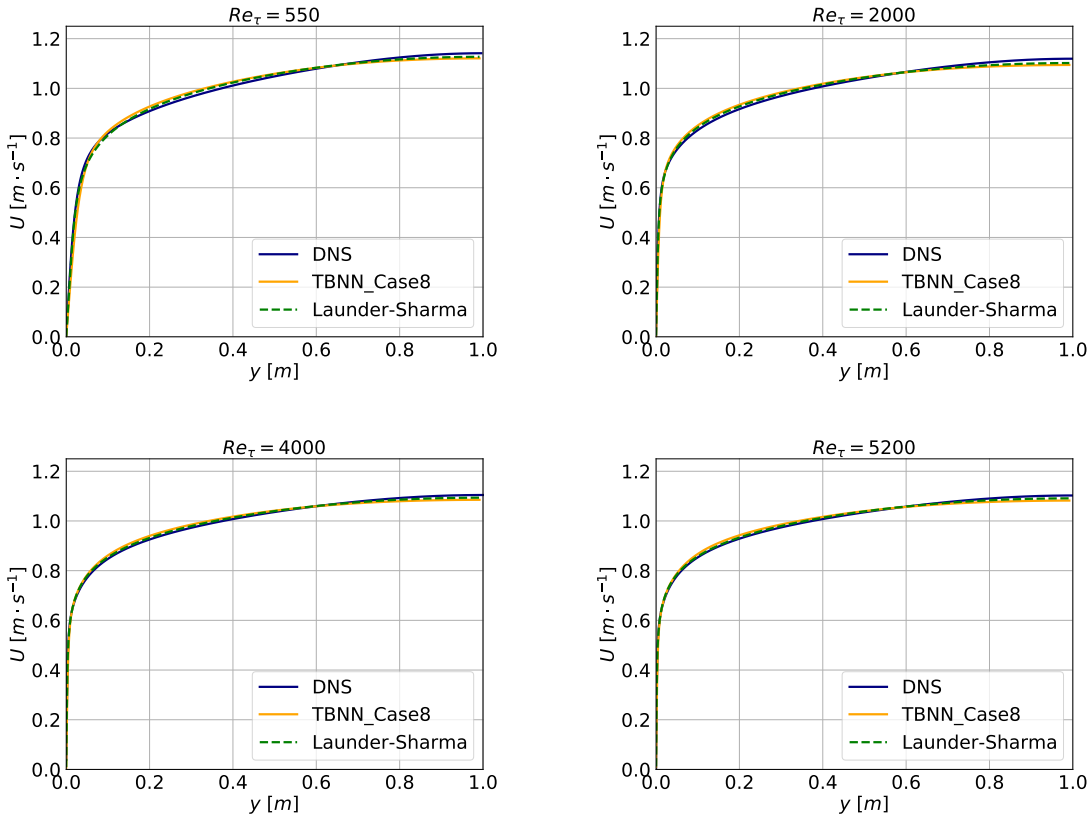


Figure 5.5: Superposition of Velocity Profiles for $Re_\tau = [550, 2000, 4000, 5200]$: RANS with case 8 neural network vs. DNS Simulations.

The analysis of the velocity profiles regarding the low-Reynolds models ends with the study of the near-wall region. This study is performed, in the scientific literature, with a plot of the dimensionless velocity, U^+, against the dimensionless distance from the wall, y^+, in a logarithmic scale on the x-axis. The dimensionless variables are defined as follows:

$$U^+ = \frac{U(y)}{u_\tau} \qquad y^+ = \frac{y u_\tau}{\nu}$$

where $u_\tau = \sqrt{\nu \left.\frac{d\bar{u}_1}{dx_2}\right|_{y=0}}$.

In Figure 5.6 the results of this analysis are presented. Both the Launder-Sharma and neuronal models are able to provide a profile of $U^+(y^+)$ that follows the one of the DNS on the region $0 \leq y^+ \leq 10$ for every Re_τ. In the region $y^+ \geq 10$ the Launder-Sharma model is able to capture the dimensionless velocity profile of the DNS, while the values provided by the neuronal models are higher than the ones of the DNS. This is due to

the fact that the values of u_τ provided by the Launder-Sharma model and the DNS are comparable, while the ones of the neuronal models are smaller. Between the two neural networks one can observe that for $Re_\tau > 500$ the differences are undetectable, while for the cases $Re_\tau = [180, 550]$ the `Case8` neural network is able to provide slightly more accurate values of the velocity in the region $y^+ \geq 10$.

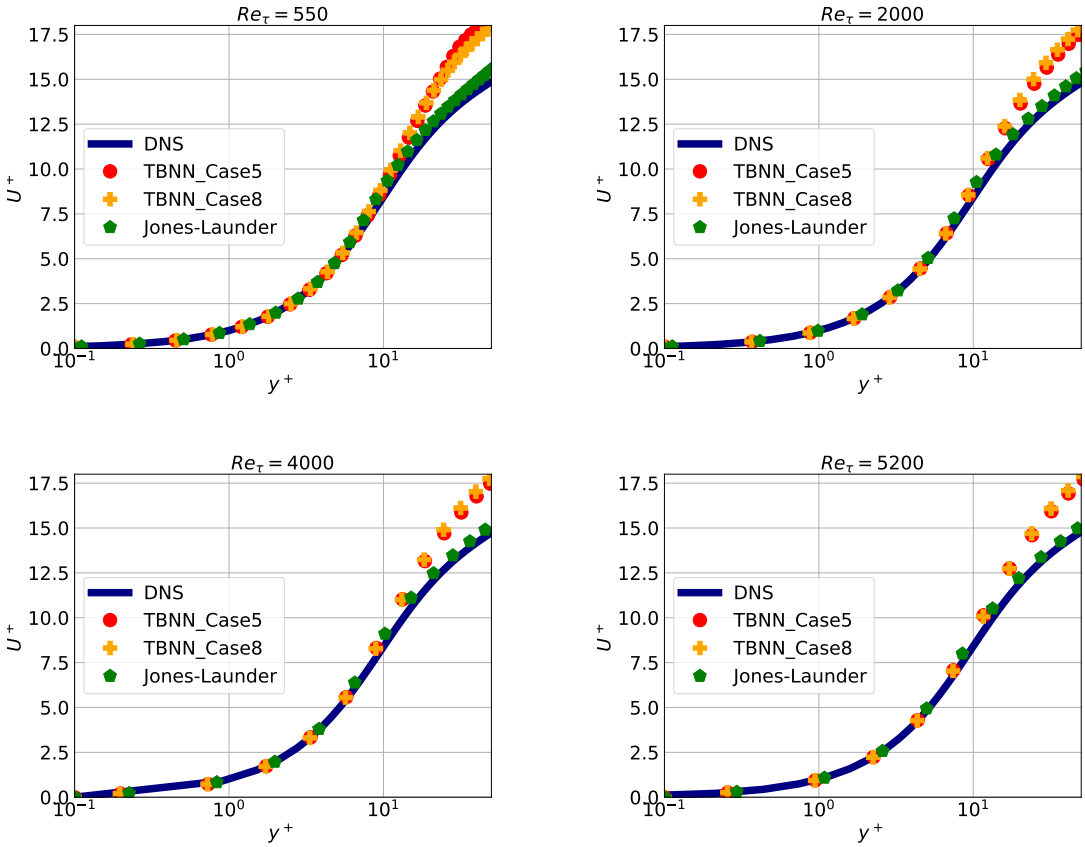
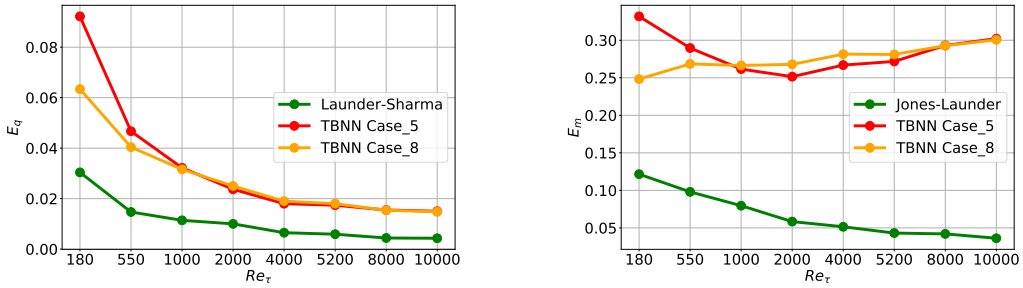


Figure 5.6: Superposition of Velocity Profiles for $Re_\tau = [550, 2000, 4000, 5200]$: RANS with case 8 neural network vs. DNS Simulations.

Error analysis

In Figure 5.7 is reported the errors E_q, Equation 5.1, and E_{max}, Equation 5.2, of the simulations performed with the neural networks 5 and 8 compared to the error obtained from the simulations performed with the Launder-Sharma model. One can observe that for low values of the friction Reynolds number, $Re_\tau \leq 1000$, the two errors are significantly higher for the models based on the neural networks. This can be explained by recalling that the problems with $Re_\tau = [180, 550]$ are in extrapolation with respect to the training of the two neural networks. On the other side it can be observed that, for values of

Re_τ greater than 2000, the two models based on machine learning provide similar velocity profiles and their E_q error follows the same trend of the one of the Launder-Sharma model. This might suggest that since the machine learning based model does not provide every coefficient by means of the neural network but utilizes some coefficient of the model.



(a) E_q . (b) E_{max} .

Figure 5.7: Low-Reynolds models: E_q and E_{max} for every Re_τ.

Computational Cost

In Table 5.6 are presented the results of the analysis on the time required by every simulation to reach convergence. The Launder-Sharma model significantly outperforms the neuronal models proposed in this work in terms of computational cost. Indeed while around 100 seconds are required to reach convergence in the Launder-Sharma model, the time required for the neuronal models was approximately 5 to 10 times higher. Likely, this occurred because the architecture of the solver necessitated the neural network to be uploaded at each time step, leading to a substantial increase in computational time.

Convergence Time

	180	**550**	**1000**	**2000**	**4000**	**5200**	**8000**	**10000**
Launder-Sharma	103	51	71	39	63	90	146	103
TBNN Case 5	1126	570	503	353	583	758	811	454
TBNN Case 8	627	277	712	337	500	744	889	475

Table 5.6: Convergence time: Comparison between different low-Reynolds models for every Re_τ. Times are expressed in seconds.

Model Evaluation and Comparative Analysis

This section examines the reasons contributing to the less precise outcomes of the machine learning-based model in comparison to those of the Launder-Sharma model. Additionally, a potential explanation for this situation is proposed.

From the previous results, one can deduce that the low-Reynolds neuronal $k-\varepsilon$ model does not outperform the existing Launder-Sharma model neither in the accuracy of the velocity profile nor in the computational time required to reach the converge point. Despite this statement is true, it does not necessarily mean that the machine learning based models perform worse than the standard ones in estimating the Reynolds tensor. In Figure 5.8 one can observe the Reynolds tensor computed by the neural network at the convergence point of the simulation performed with the low-Reynolds neuronal model. It has to be recalled that, since the Launder-Sharma model is based on the linearity assumption of Boussinesq the only non-zero component of the tensor is b_{12}. Therefore it is evident that the prediction of $\mathbf{b}$ performed by the neuronal model outperforms the one of the Launder-Sharma model for $Re_\tau = 5200$.

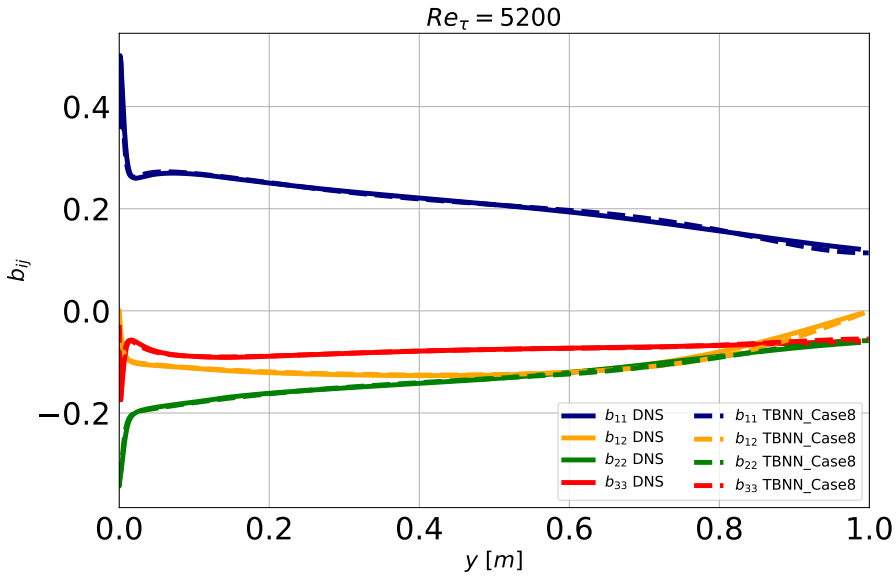
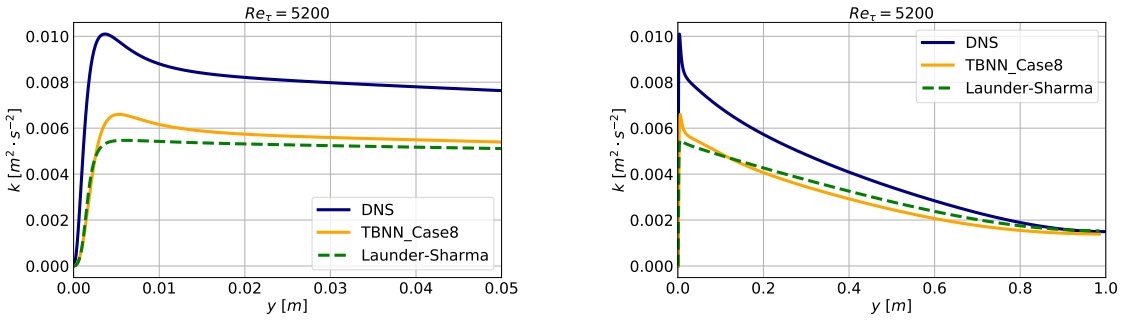


Figure 5.8: Anisotropic Reynolds tensor components, b_{ij}, for $Re_\tau = 5200$: `Case 8` low-Reynolds neuronal model vs. DNS .

It is a strong belief of the author that the main weakness of the low-Reynolds neuronal $k-\varepsilon$ model proposed in this work is to be based on a classical low-Reynolds $k-\varepsilon$ model. As shown in Table 3.3, most of the parameters are either kept identical to the Launder-Sharma model, either slightly changed by substituting the old ν_t with the $\widetilde{\nu_t}$ computed

from the neural network. These changes are not sufficient because the rationale behind the choice of the parameters of the classical models does not come from the physics of the problem, but it is led by the necessity to reach a velocity profile as close as possible to the real one. The aim of the model is indeed to provide a good estimate of ν_t, which is the only quantity involved in the RANS equation for the linear models, thus based on the Boussinesq assumption. For the non-linear models it is crucial to provide also a good estimate for the value of k since it is involved in the computation of the factor $\mathbf{b}^{NL}$.

The inadequacy of the existing $k-\varepsilon$ models to provide a good estimate for the turbulent kinetic energy, k, is clearly shown in Figure 5.9. One can observe that the neuronal model outperforms the Launder-Sharma model regarding the shape of k with respect to y both in the near-wall region and in the entire domain. On the other side, the neuronal model pays its dependency from the classical model for what concerns the magnitude of the values that are similar to those of the latter model.

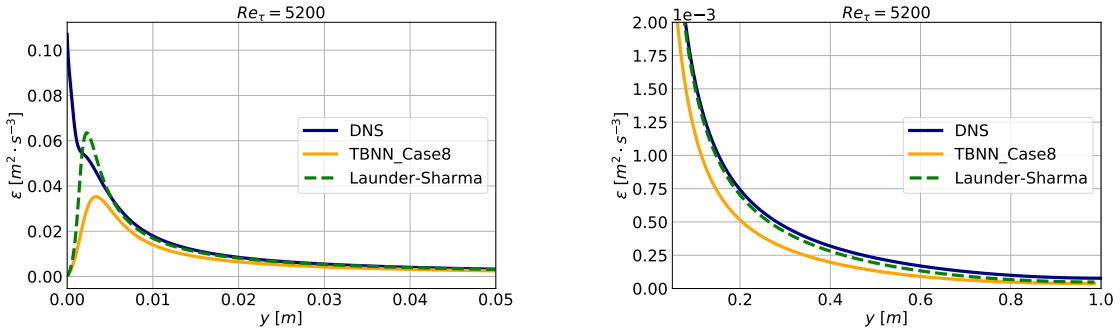


(a) Near-wall region. (b) Entire domain.

Figure 5.9: Low-Reynolds models: k comparison for $Re_\tau = 5200$. General behavior and focus on the near-wall region. k is expressed in $[m^2 \cdot s^{-2}]$ and y in $[m]$.

The rate of dissipation of turbulent kinetic energy, ε, is strongly linked to k in the framework of the $k-\varepsilon$ models. Its values are displayed in Figure 5.10. One can observe that apart from the small region $0 \leq y \leq 0.05$, where ε is forced by the model to reach 0 at $y = 0$, the values obtained from the Launder-Sharma model coincide with the ones obtained from the DNS. The values of ε obtained from the neuronal model are not dissimilar to the good values obtained from the Launder-Sharma model. The only difference, particularly visible in the region $0.05 \leq y \leq 0.1$ is a slightly lower value probably due to a higher value of k in the corresponding area. Following this consideration one can think that the parameters of the equation regarding ε, Equation 3.3d, can be left unchanged; it has to be kept in mind that this differential equation is dependent from the value of k, therefore some adjustments have to be made in order to maintain the values of ε close to

the ones of the DNS while fixing the values of k.

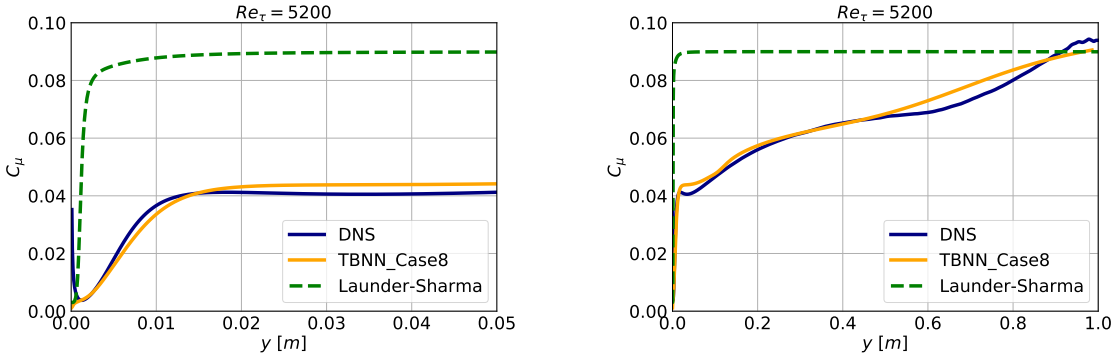


(a) Near-wall region.

(b) Complementary region of the domain.

Figure 5.10: Low-Reynolds models: ε comparison for $Re_\tau = 5200$.

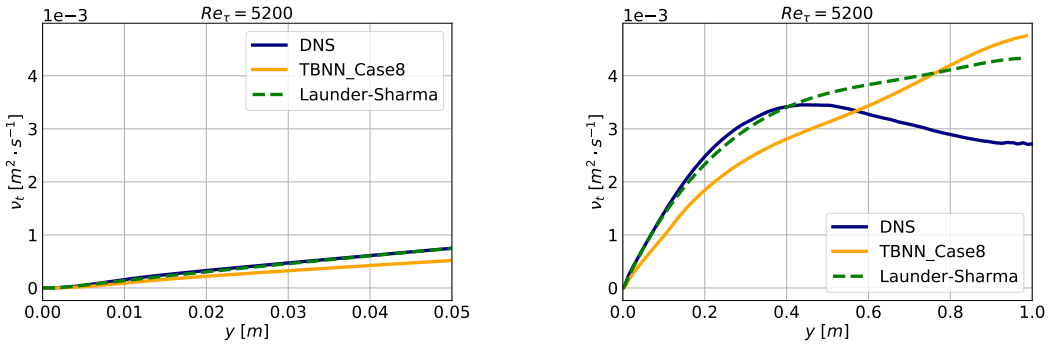
The analysis proceeds with the assessment of the values of C_μ in the two low-Reynolds models, classic and neuronal, compared to the ones of the Direct Numerical Simulation (DNS) presented in Figure 5.11. It is self-evident to observe that the value of C_μ directly obtained from the neural network - in the neuronal model $C_\mu = -g_1$ - significantly outperforms the analytical model proposed by Launder and Sharma where C_μ is set to be equal to 0.09 times a function, called f_μ, that aims to adjust its value in the near-wall region. From the analysis of C_μ it is clear that while the classical model only aims to provide a C_μ such that the model can give a good final result in terms of velocity profile, the neuronal model aims to keep the model linked to the physics and therefore is able to provide a C_μ similar to the one obtained from the DNS. It has to be observed that the values proposed for the neuronal model do not correspond to the best values that this model can achieve but to the ones related to the convergence point reached with the low-Reynolds neuronal model, the problems of which have already been discussed.



(a) Near-wall region. (b) Entire domain.

Figure 5.11: Low-Reynolds models: C_μ comparison for $Re_\tau = 5200$.

The analysis ends with the study of the behavior of the turbulent viscosity $\nu_t = C_\mu \frac{k^2}{\varepsilon}$. One can remark that even if the neuronal model is able to provide a more accurate profile of C_μ than the one given by the Launder-Sharma model, the profile of ν_t presents the opposite behavior. This is probably due to the fact that, given an analytical formula for C_μ, the parameters of the equations of k and ε(σ_k, σ_ε, D, E, $C_{\varepsilon 1}$, $C_{\varepsilon 2}$, f_1, f_2 and Re_t), are adapted to obtain the k and ε profiles that provide a good value of ν_t. For what concerns the neuronal model, the values of ν_t are far from the ones of the DNS because of the wrong estimate of k and ε given by the equations of the model. In particular, in the region $0 \leq y \leq 0.6$ ν_t is more influenced by the low values of k while in the region $0.6 \leq y \leq 1$ k approaches the DNS values and therefore the lower values of ε influence the final result.



(a) Near-wall region. (b) Entire domain.

Figure 5.12: Low-Reynolds models: ν_t comparison for $Re_\tau = 5200$

6 | Conclusions and future developments

The central focus of this thesis was placed on assessing machine learning techniques to enhance Reynolds stress closures. Both high-Reynolds and low-Reynolds neuronal models were evaluated on the turbulent channel flow configuration.

The preliminary explicit treatment of the Reynolds stress tensor obtained from DNS highlighted concerns regarding the ill-conditioning of the explicit treatment of the Reynolds tensor in the RANS equations. Subsequently, the proposed high-Reynolds neuronal $k-\varepsilon$ model demonstrated comparable performance to the standard $k-\varepsilon$ model, while requiring over twice the computational expense. The low-Reynolds neuronal model, assessed thanks to the TRUST/TrioCFD code integration, exhibited very promising results. Indeed, the values of the quantities directly predicted by the neural network, that is C_μ and b_{ij}, strongly outperform the existing $k-\varepsilon$ model. Even if this result already represents an achievement itself, the perspectives of application of this neural network to new models, able to take advantage of these results to provide better estimates of the other quantities involved in the flow, are impressive.

A primary research objective was assessing whether machine learning techniques could enhance turbulence closures. The analysis revealed that specifically tailored neural network architectures possess distinctive advantages. Additionally, this work clarified ambiguities surrounding the application of Pope's model for the turbulent channel flow configuration. The generalized tensor formulation addressed limitations of prior approaches.

Despite the encouraging findings, some limitations remain. The full intricacies of turbulence continue to pose modeling challenges. Furthermore, mapping complex fluid behaviors through neural networks involves persistent difficulties in ensuring generalizability across diverse scenarios. Significant data requirements persist, necessitating reliance on DNS databases. The high computational cost also remains an issue, but it can be reduced by adapting the code to upload only once the neural network for the entire simulation.

The present work can be developed by performing further analysis with the low-Reynolds

model proposed. The validation of the model on more complex geometries, such as the square channel, is a crucial next step to understand whether the model is able to confirm or even meliorate its results with respect to the existing models. This work opens the way to various scenarios in the field of machine learning based turbulence models. In particular, the main work is to provide a model entirely adapted to the neural networks. This model can be obtained starting from the low-Reynolds neuronal $k-\varepsilon$ model proposed in this work by tailoring the empirical constants of the old models to the machine learning based one. Another valuable possibility can be to directly estimate the values of k and ε by means of a neural network.

In summary, this thesis has undertaken an extensive investigation into machine learning augmented turbulence modeling. The merits of data-driven closures have been systematically assessed. While limitations exist, the outlook remains highly promising.

Bibliography

[1] P.-E. Angeli, N. Leterrier, J.-M. Martinez, and B. Secher. Modélisation et intégration d'un modèle du tenseur de Reynolds par réseaux de neurones dans TrioCFD. Technical report, CEA, 2020.

[2] J. Boussinesq. Théorie de l'Écoulement tourbillonnant et tumultueux des liquides dans les lits rectilignes à grande section. *Gauthier-Villars et fils (Paris)*, 2:64–76, 1897.

[3] J. Cai, P.-E. Angeli, J.-M. Martinez, G. Damblin, and D. Lucor. Reynolds stress anisotropy tensor predictions for turbulent channel flow using neural networks. 2023.

[4] T. Craft, B. Launder, and K. Suga. Development and application of a cubic eddy-viscosity model of turbulence. *International Journal of Heat and Fluid Flow*, 17(2): 108–115, 1996.

[5] R. Fang, D. Sondak, P. Protopapas, and S. Succi. Neural network models for the anisotropic Reynolds stress tensor in turbulent channel flow. *Journal of Turbulence*, 21:1–19, 2019.

[6] T. Hermann. frugally-deep, 2016. URL `https://github.com/Dobiasd/frugally-deep`. MIT Licence.

[7] S. Hoyas, M. Oberlack, F. Alcántara-Ávila, S. V. Kraheberger, and J. Laux. Wall turbulence at high friction Reynolds numbers. *Phys. Rev. Fluids*, 7, 2022.

[8] S. Janati. Intégration d'un modèle neuronal du tenseur de Reynolds dans le logiciel de simulation en mécanique des fluides TRUST/TrioCFD. Master's thesis, INP Bordeaux, 2022.

[9] W. Jones and B. Launder. The calculation of low-Reynolds-number phenomena with a two-equation model of turbulence. *International Journal of Heat and Mass Transfer*, 16(6):1119–1130, 1973.

[10] Y. Kaneda and Y. Yamamoto. Velocity gradient statistics in turbulent shear flow:

an extension of Kolmogorov's local equilibrium theory. *Journal of Fluid Mechanics*, 929:A13, 2021.

[11] B. Launder and B. Sharma. Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc. *Letters in Heat and Mass Transfer*, 1 (2):131–137, 1974.

[12] J. Ling, R. Jones, and J. Templeton. Machine learning strategies for systems with invariance properties. *Journal of Computational Physics*, 318:22–35, 2016.

[13] J. Ling, A. Kurzawski, and J. Templeton. Reynolds average turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

[14] W. Liu, J. Fang, S. Rolfo, C. Moulinec, and D. R. Emerson. An iterative machine-learning framework for RANS turbulence modeling. *International Journal of Heat and Fluid Flow*, 90, 2021.

[15] M. Milano and P. Koumoutsakos. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics*, 182(1):1–26, 2002.

[16] R. D. Moser, J. Kim, and N. N. Mansour. Direct numerical simulation of turbulent channel flow up to $Re_\tau = 590$. *Physics of Fluids*, 11(4):943–945, 1999.

[17] S. B. Pope. A more general effective-viscosity hypothesis. *Journal of Fluid Mechanics*, 72(2):331–340, 1975.

[18] S. B. Pope. *Turbulent Flows*. Cambridge University Press, 2000.

[19] L. Prandtl. Bericht uber untersuchungen zur ausgebildeten turbulenz. *Journal of Applied Mathematics and Mechanics*, 5(2):136–139, 1925.

[20] H. Sáez de Ocáriz Borde, D. Sondak, and P. Protopapas. Convolutional neural network models and interpretability for the anisotropic Reynolds stress tensor in turbulent one-dimensional flows. *Journal of Turbulence*, 23(1-2):1–28, 2021.

[21] B. Tracey, K. Duraisamy, and J. Alonso. *Application of Supervised Learning to Quantify Uncertainties in Turbulence and Combustion Modeling*. AIAA Aerospace Sciences Meeting, 2013.

[22] B. D. Tracey, K. Duraisamy, and J. J. Alonso. *A Machine Learning Strategy to Assist Turbulence Model Development*.

[23] H. K. Versteeg and W. Malalasekera. *An introduction to computational fluid dynamics : the finite volume method*. Pearson Education Ltd., second edition, 2007.

[24] J. Wu, H. Xiao, R. Sun, and Q. Wang. Reynolds-Averaged Navier–Stokes equations with explicit data-driven Reynolds stress closure can be ill-conditioned. *Journal of Fluid Mechanics*, 869:553–586, 2019.

[25] Z. Zhang, X. Song, S. Ye, Y. Wang, C. Huang, Y. An, and Y. Chen. Application of deep learning method to Reynolds stress models of channel flow based on reduced-order modeling of DNS data. *Journal of Hydrodynamics*, 31:58–65, 2019.

[26] Z. J. Zhang and K. Duraisamy. *Machine Learning Methods for Data-Driven Turbulence Modeling.*

A | Validation Plots

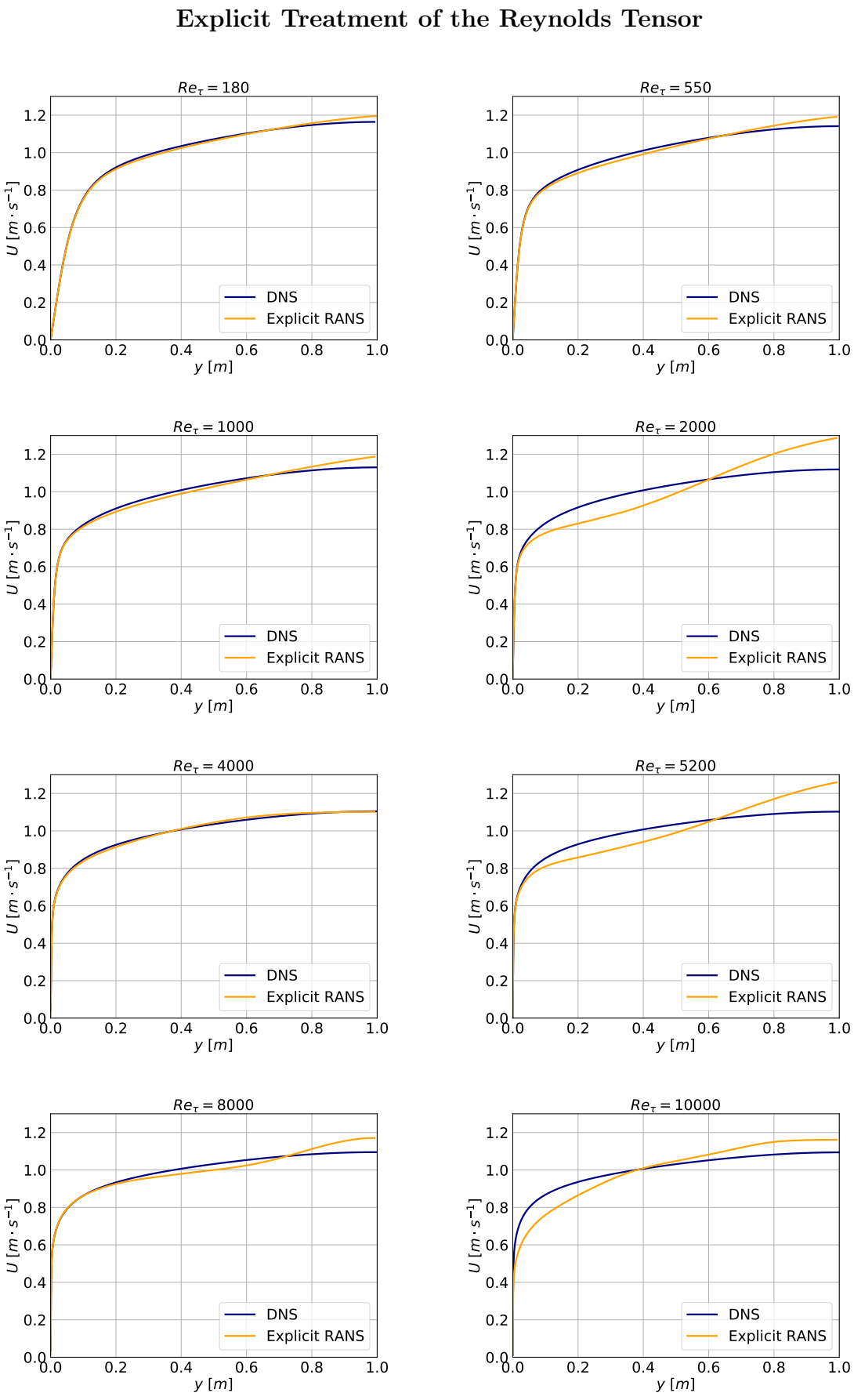


Figure A.1: Superposition of Velocity Profiles: RANS with Explicit Treatment of the Reynolds Tensor vs. DNS Simulations.

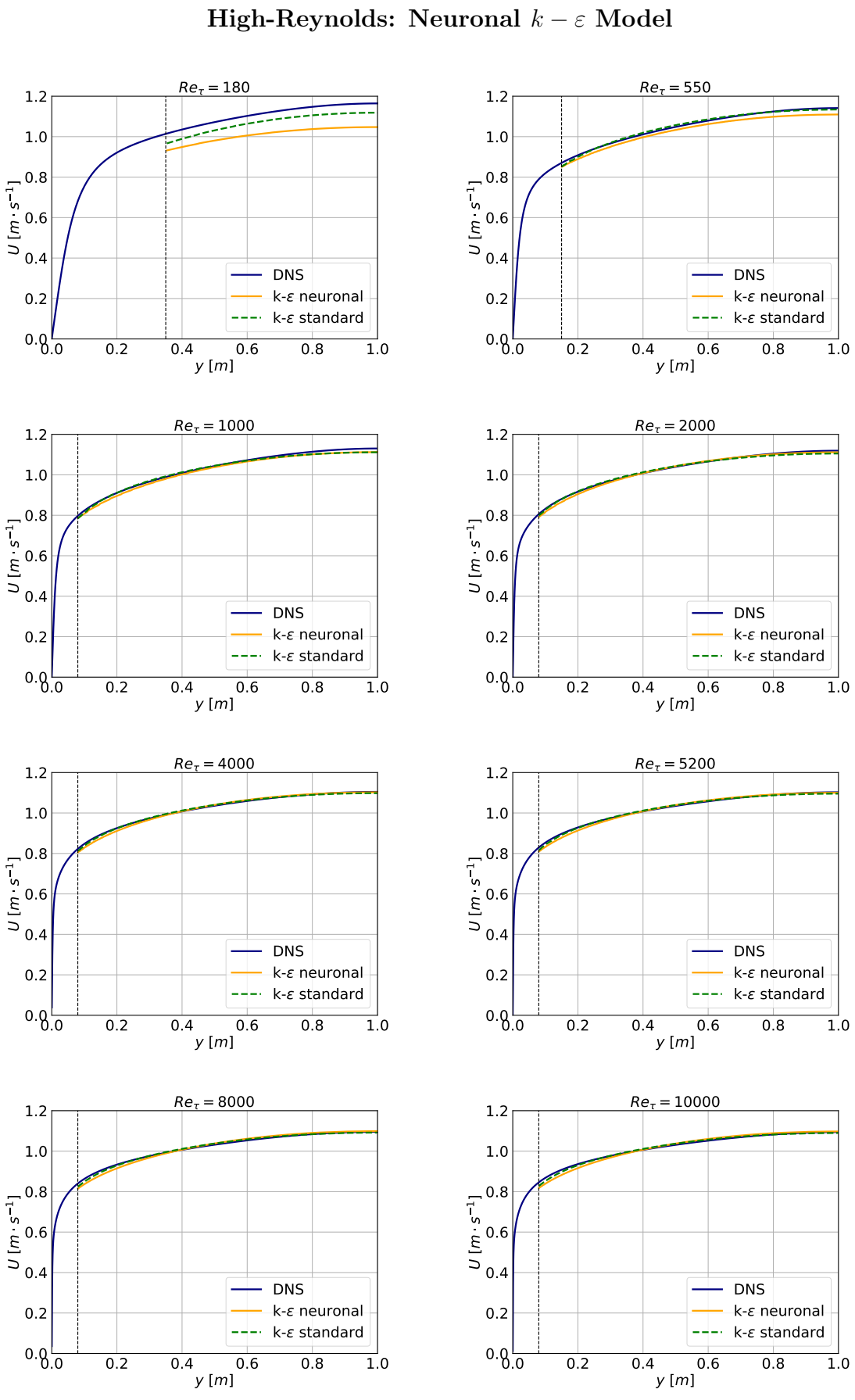


Figure A.2: Superposition of Velocity Profiles: RANS with neuronal $k-\varepsilon$ model vs. DNS Simulations.

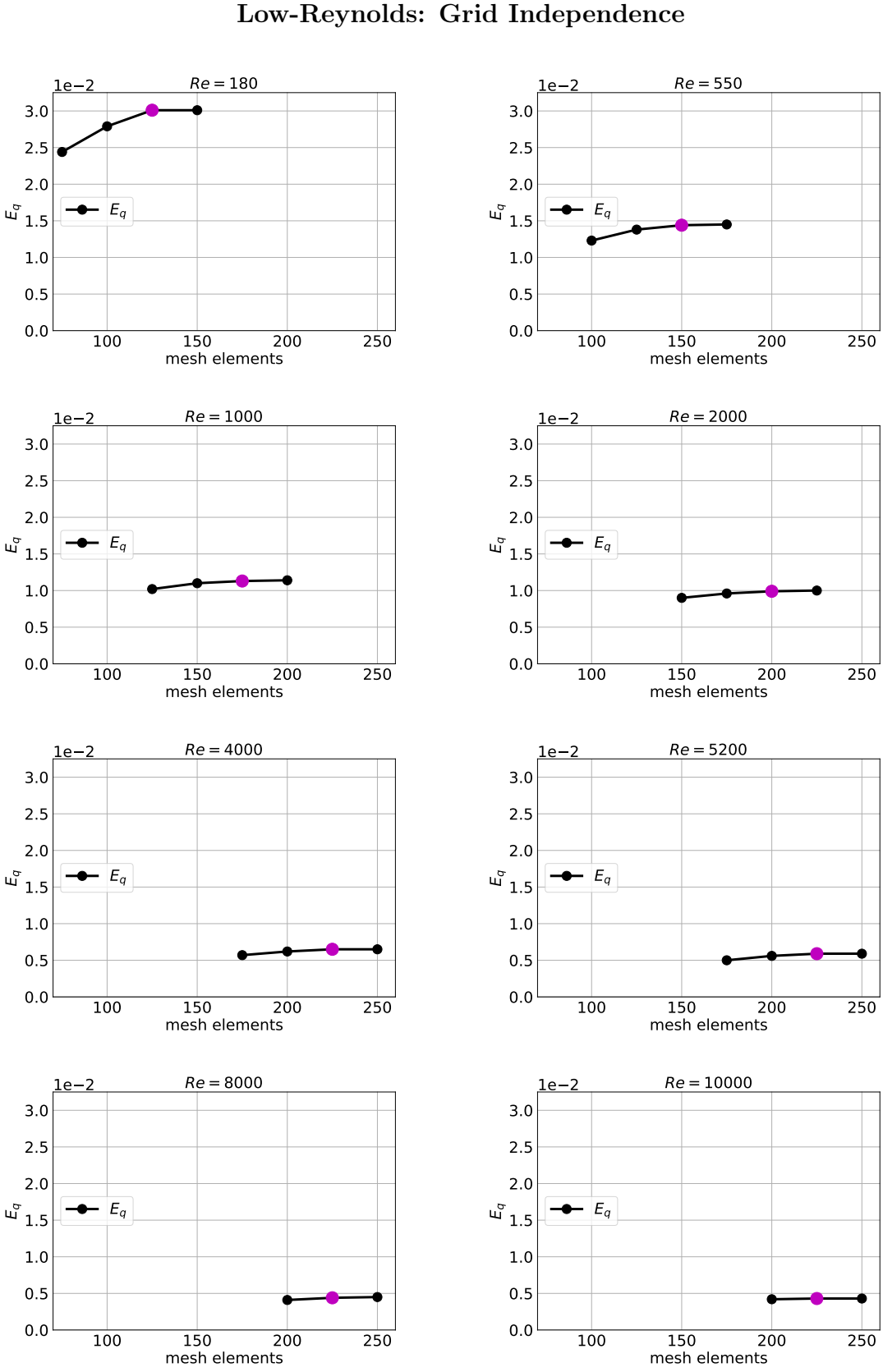


Figure A.3: Grid independence: quantitative results.

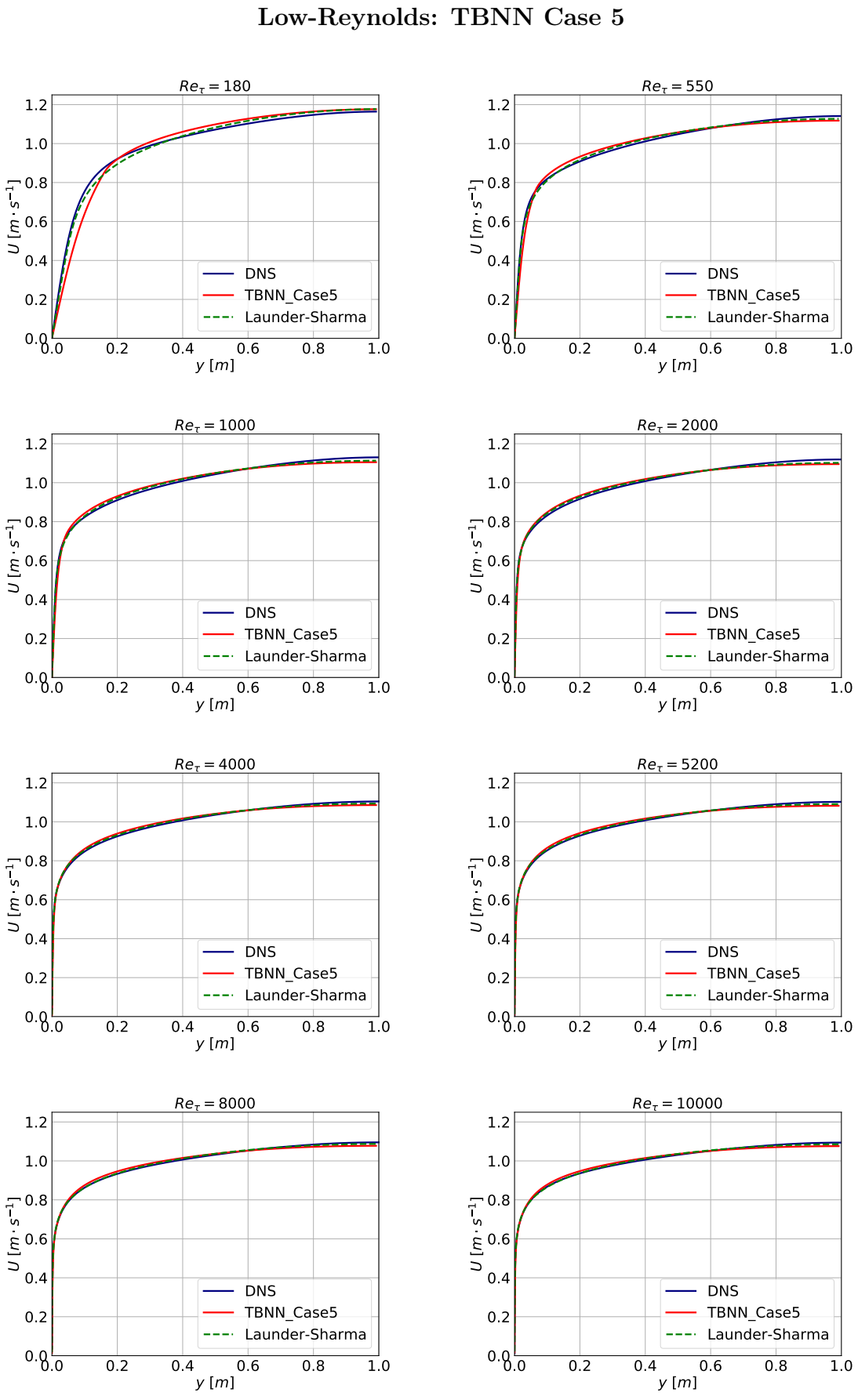


Figure A.4: Superposition of Velocity Profiles: RANS with case 5 Neural Network vs. DNS Simulations.

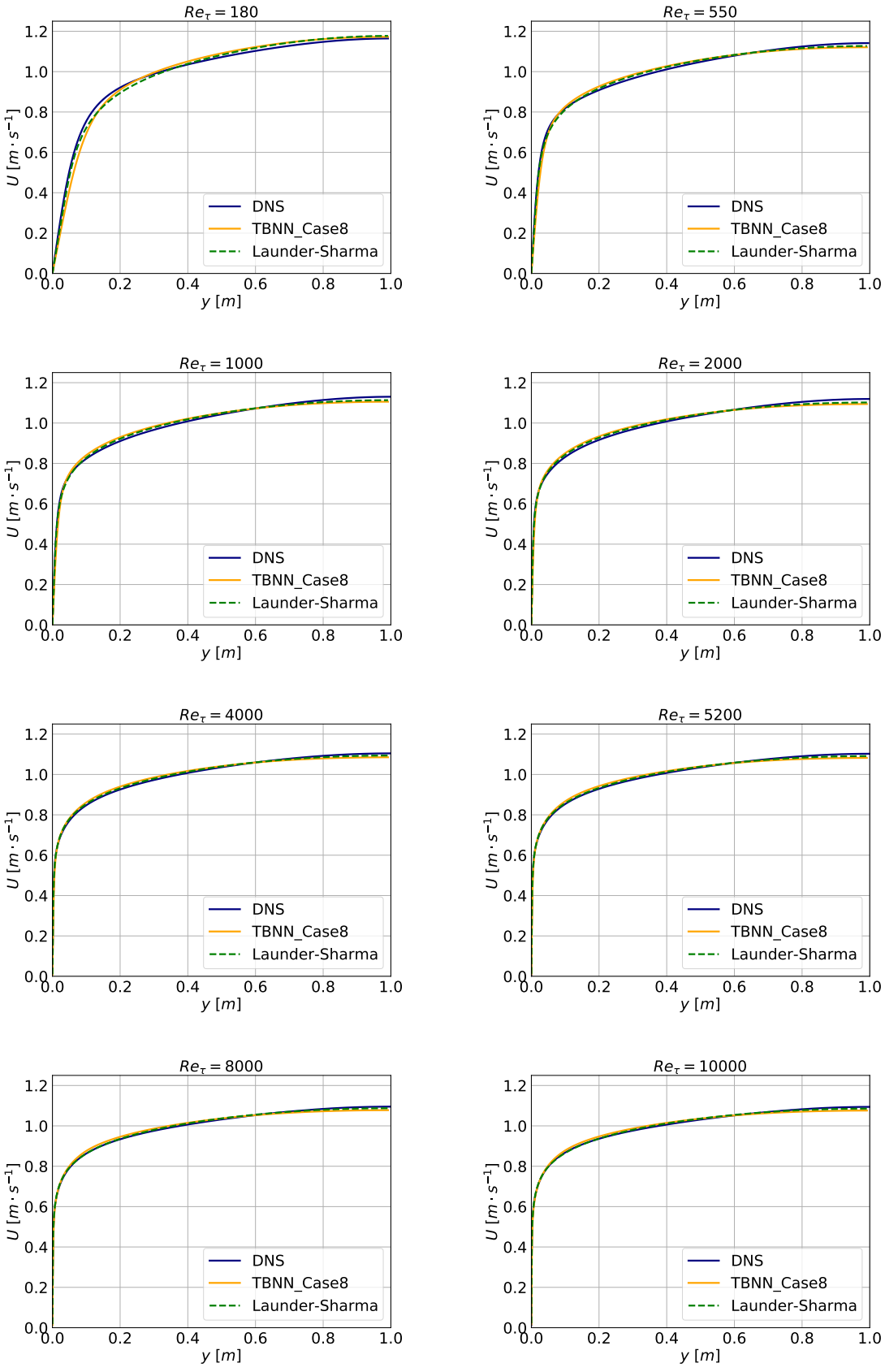


Figure A.5: Superposition of Velocity Profiles: RANS with case 8 Neural Network vs. DNS Simulations.

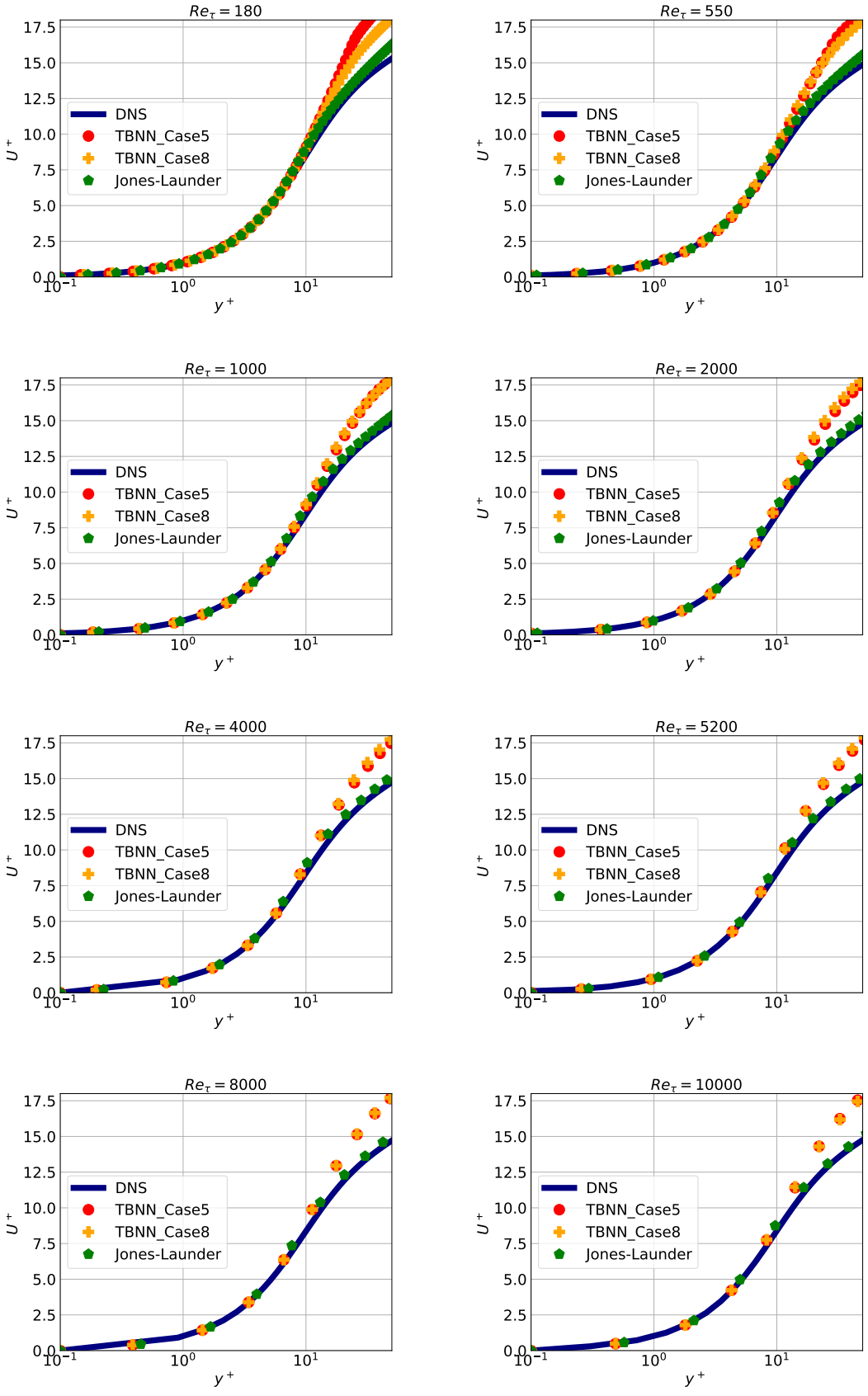


Figure A.6: Superposition of Velocity Profiles: Dimensionless velocity U^+ against dimensionless wall distance y^+ for every Re_τ .

List of Figures

List of Tables

Listings

Acknowledgements

Here you might want to acknowledge someone.