APPLICATION OF SYMMETRY FILTERS ON A DNS DATABASE TO BUILD EUCLIDEAN INVARIANT DATA-DRIVEN TURBULENCE MODELS: A COMPARISON BETWEEN NEURAL NETWORK AND RANDOM FOREST MACHINE LEARNING TECHNIQUES

Eduardo Ferreira Fonseca

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Mecânica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Mecânica.

Orientador: Roney Leon Thompson

Rio de Janeiro
Dezembro de 2020

APPLICATION OF SYMMETRY FILTERS ON A DNS DATABASE TO BUILD EUCLIDEAN INVARIANT DATA-DRIVEN TURBULENCE MODELS: A COMPARISON BETWEEN NEURAL NETWORK AND RANDOM FOREST MACHINE LEARNING TECHNIQUES

Eduardo Ferreira Fonseca

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA MECÂNICA.

Orientador: Roney Leon Thompson

Aprovada por: Prof. Roney Leon Thompson
Prof. Ricardo Eduardo Musafir
Prof. Tânia Suaiden Klein

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2020

Fonseca, Eduardo Ferreira

Application of symmetry filters on a DNS database to build euclidean invariant data-driven turbulence models: a comparison between neural network and random forest machine learning techniques/Eduardo Ferreira Fonseca. – Rio de Janeiro: UFRJ/COPPE, 2020.

XV, 81 p.: il.; 29, 7cm.

Orientador: Roney Leon Thompson

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Mecânica, 2020.

Referências Bibliográficas: p. 77 – 81.

1. Turbulence. 2. Machine Learning. 3. CFD. I. Thompson, Roney Leon. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Mecânica. III. Título.

We are born into this world
unarmed – our mind is our only
weapon
Ayn Rand, Atlas Shrugged

Agradecimentos

Gostaria de agradecer a todo grupo de Machine Learning por todas as discussões e trocas de conhecimento. Em especial, ao Professor Roney pela orientação durante esses últimos dois anos do mestrado, sempre propondo metodologias inovadoras, ao Matheus Altomare, por partilhar a superação de diversos desafios e ao Bernardo Brener, por estar sempre disponível para me auxiliar em diversos problemas.

À minha familia, em especial, minha mãe e meu irmão, pelo apoio e motivação incondicional que sempre me deram, sempre acreditando em mim. Sem vocês, nada seria possível.

À minha namorada Gabriela, que está sempre ao meu lado, me apoiando em qualquer decisão tomada na minha vida e me confortando sempre que necessário.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

APLICAÇÃO DE FILTROS DE SIMETRIA EM UMA BASE DE DADOS DNS PARA CONSTRUIR MODELOS DE TURBULÊNCIA BASEADOS EM DADOS COM INVARIÂNCIA EUCLIDIANA: UMA COMPARAÇÃO ENTRE REDES NEURAIS E FLORESTAS RANDÔMICAS

Eduardo Ferreira Fonseca

Dezembro/2020

Orientador: Roney Leon Thompson

Programa: Engenharia Mecânica

Em aplicações industriais, grande parte dos escoamentos acontece no regime turbulento. Soluções numéricas de alta acurácia, como Direct Numerical Simulation (DNS), tem custos computacionais proibitivos, portanto, grande parte dos problemas numéricos são resolvidos a partir do uso de modelos Reynolds Average Navier-Stokes (RANS), que têm como característica baixo custo computacional e acurácia muitas vezes não satisfatória. A utilização de técnicas de Machine Learning (ML) para modelos de turbulência, já vem sendo empregada na literatura, utilizando dados DNS como alvo nos modelos de ML, a partir de dados do campo de velocidade, ou do tensor de Reynolds, $\mathbf{R}$. Em estudos recentes foi observado, entretanto, que os dados DNS de $\mathbf{R}$ não apresentam convergência satisfatória, quando comparados aos dados dos campos de velocidade e pressão. Para contornar este problema, foi desenvolvida a metodologia para correção do divergente modificado de $\mathbf{R}$, a partir apenas, do campo de velocidade, denominada de $\mathbf{t}$, apresentando resultados promissores, quando comparados à $\mathbf{R}$. Neste trabalho, o uso de base de dados com invariância Euclideana e a influência da qualidade da base de dados DNS, utilizada em Redes Neurais (NN) e Florestas Randômicas (RF), para a previsão das propriedades turbulentas $\mathbf{R}$ e $\mathbf{t}$, são investigados em um escoamento turbulento em duto quadrado. Um tratamento da base de dados DNS é realizado de modo a emular maiores tempos médios de simulação DNS e, consequentemente, melhor convergência dos mesmos. Os resultados obtidos por todos os modelos construídos mostram que há uma relação direta entre a convergência dos dados DNS, a simetria presente neste padrão de escoamento e o desempenho de modelos de turbulência guiados por dados.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

APPLICATION OF SYMMETRY FILTERS ON A DNS DATABASE TO BUILD EUCLIDEAN INVARIANT DATA-DRIVEN TURBULENCE MODELS: A COMPARISON BETWEEN NEURAL NETWORK AND RANDOM FOREST MACHINE LEARNING TECHNIQUES

Eduardo Ferreira Fonseca

December/2020

Advisor: Roney Leon Thompson

Department: Mechanical Engineering

In industrial applications, great part of the flows occurs in the turbulent regime. Highly accurate numerical solutions, such as Direct Numerical Simulation (DNS), have prohibitive computational costs so most numerical problems are solved using Reynolds Average Navier-Stokes (RANS) models, which feature low computational cost and not often satisfactory accuracy. The use of Machine Learning (ML) techniques for turbulence models has already been used in the literature, setting DNS data as a target in ML models, based on data from the mean velocity field, or the Reynolds stress tensor, $\mathbf{R}$. In recent studies, however, it was observed that the DNS data for $\mathbf{R}$ does not show satisfactory convergence, when compared to data for the mean velocity and pressure fields. In order to get around this problem, a methodology was developed to correct the modified divergent of $\mathbf{R}$, using only data related to the mean velocity field, called $\mathbf{t}$, which presented promising results when compared to $\mathbf{R}$ corrections. In this work, the use of a database with Euclidean invariance and the influence of the quality of the DNS database, used in Neural Networks (NN) and Random Forests (RF), to predict the turbulent properties $\mathbf{R}$ and $\mathbf{t}$, are investigated in a turbulent flow in a square duct. A treatment of the DNS database is carried out in order to emulate longer DNS averaging simulation times and, consequently, better convergence of related fields. The results obtained by all models built, show that there is a direct relation between the convergence of DNS data, the symmetry present in this flow pattern and the performance of data-driven turbulence models.

Contents

List of Figures

List of Tables

List of Symbols

b	Neural network bias, p. 12
$\mathbf{b}$	Normalized Reynolds stress anisotropy tensor, p. 27
C	Cost function, p. 14
$\mathbf{D}$	Mean strain rate tensor, p. 1
D	Square duct hydraulic diameter, p. 39
E_i	Local error, p. 48
$\bar{E}_i$	Global error, p. 48
F	Frame of reference, p. 25
g_t	Gradient of the objective function to the model parameter, p. 16
J	Objective function, p. 16
$\mathbf{L}$	Characteristic length, p. 8
$\mathbf{L}(\mathbf{x}, t)$	Velocity gradient, p. 26
m_t	Adam's first moment of the gradient, p. 18
$\mathbf{P}$	Non-persistence-of-straining tensor, p. 40
$\mathbf{Q}$	Frame-rotation tensor, p. 25
$\mathbf{R}$	Reynolds stress tensor, p. 1
Re_τ	Friction Reynolds number, p. 34
Re	Reynolds number, p. 1
$\mathbf{t}$	Reynolds force vector, p. 30
$\mathbf{U}$	Characteristic velocity, p. 8

v_t	Update vector, p. 17
$\overline{w_i}$	Relative vorticity, p. 41
w	Neural network weight, p. 12
$\mathbf{W}$	Vorticity tensor, p. 26
$\overline{\mathbf{W}}$	Relative-rate-of-rotation tensor, p. 40
γ	Decay term, p. 17
$\boldsymbol{\delta}$	Kronecker delta, p. 27
ε	Rate of dissipation, p. 8
η	Learning rate, p. 16
θ	Model parameters, p. 16
κ	Turbulent kinetic energy, p. 7
λ_i	Eigenvalues of the normalized Reynolds stress anisotropy tensor, p. 27
$\lambda_i^{(D)}$	Mean strain rate tensor eigenvalues, p. 41
μ_{column}	ML parameter mean, p. 51
ν_T	Eddy viscosity, p. 7
σ_{column}	ML parameter standard deviation, p. 51
φ	Neural network activation function, p. 12
ϕ	Sample field, p. 5
ψ	Sample field, p. 6
ω	Specific rate of dissipation, p. 8
Ω	Frame-spin tensor, p. 26
$\mathbf{\Omega^D}$	tensor that gives the rotation of the eigenvectors of $\mathbf{D}$, p. 40

Chapter 1

Introduction

With the exponential power increase of computer hardware, computational fluid dynamics (CFD) has become an even more prevalent tool on fluid flow analysis. Although over recent years Large Eddy Simulations (LES) or Direct Numerical Simulations (DNS) have become more accessible, these methods still remain out of the scope for complex geometries and high Reynolds numbers, Re. Therefore, the use of Reynolds Average Navier Stokes (RANS) models, related to low computational costs and fast results, are needed. Modern machine learning (ML) techniques have opened up a new area of turbulence models, allowing for the tuning of RANS simulations to increase their predictive accuracy. Thus, improving the accuracy of RANS simulations and providing measures of their predictive capability, remains essential for CFD engineering applications.

While several data-driven turbulence models have been proposed, the analysis of high fidelity simulations data, such as LES and DNS, have been neglected. The database treatment of high fidelity simulations is vital, as ML models predictive capability is highly linked with the database utilized and rapidly declines as they are submitted to flow conditions that differ from the training data. Therefore, the treatment of high fidelity databases, used as training target on ML models, is crucial in order to develop a model able to predict a wide range of different conditions of a turbulent flow.

Nowadays, CFD applications on industrial turbulent cases and real world engineering problems rely heavily on RANS simulations. Most of these RANS simulations are two-equation models, which assumes the Boussinesq hypothesis stated by BOUSSINESQ (1877). This hypothesis relate a linear relation between the Reynolds stress tensor ($\mathbf{R}$) and the mean strain rate tensor ($\mathbf{D}$), determined by a turbulent quantity called eddy viscosity.

But as the complexity of the problem increases, with complex geometries or high Reynolds numbers, the capability of RANS simulations to capture the physics of important flows diminishes, imposing a limit associated with these simulations,

as studied by MUCK *et al.* (1985), which states the incapability of two-equation models to provide adequate results on complex curvature flows e.g, secondary flows in ducts. CRAFT *et al.* (1996) declared that these linear relation is unable to deliver acceptable results over a range of engineering relevant flows, involving strong streamline curvature, swirls and pressure gradients.

The application of ML techniques on turbulent flows has been deeply explored over the past few years, in order to obtain better results for the Reynolds stress closure problem. MILANO and KOUMOUTSAKOS (2002) proposed a Neural Network (NN) methodology, developed in order to reconstruct the near wall field in a turbulent flow by exploiting flow fields provided by direct numerical simulations. YARLANKI *et al.* (2012) used ML algorithms to optimize the constants of the $\kappa - \epsilon$ turbulence model for a data center by comparing simulated data with experimentally measured temperature values. SNOEK *et al.* (2012) considered the automatic tuning problem of a NN within the framework of Bayesian optimization, where a learning algorithm's generalization performance is modeled as a sample of a Gaussian process. TRACEY *et al.* (2013) used a kernel regression to correct the eigenvalues of the anisotropic Reynolds stress tensor of a periodic hill flow pattern, from $\kappa - \omega SST$ RANS simulation model, using a related DNS data as target. It was shown a notable correction performed by the kernel regression, correcting the anisotropic field closer to the DNS database, demonstrating the potential of ML techniques in turbulence modeling.

In a latter work, TRACEY *et al.* (2015) suggested a NN with one single hidden layer to model the source terms from the classic Spalart Allmaras RANS simulation model, aiming to correct a specific term of the Reynolds stress tensor. In that work the friction coefficient was corrected by a prediction of the source term of the eddy viscosity of the transport equation of the Spalart Almaras model. In that case, the data were created by the turbulent model, therefore, the result was that a NN could replace the source term with a learned correlation.

LING *et al.* (2016c) used Random Forests (RF) to predict improvements in the Reynolds stress anisotropy in the injection hole, along the wall, and on the lee side of a jet using as target LES results. In Latter work, LING *et al.* (2016a), proposed the use of a set of Galilean invariant data as inputs of a NN in order to predict the Reynolds stress tensor anisotropy. It was shown that embedding the invariance property into the input features yields higher performance at significantly reduced computational training costs. Also, LING *et al.* (2016b) developed a NN architecture with an invariant tensor basis to embed Galilean invariance into the predicted Reynolds stress tensor anisotropy. The results obtained in these work show that deep learning and the proposed architecture are able to provide a substantial performance improvement. XIAO *et al.* (2016), developed a Bayesian framework which quantifies

and reduces uncertainties through the Kalman method, taking into account information of the physics involved, limitations and plausibility of the injected disturbances in RANS data. LING *et al.* (2016b), used the Bayesian optimization process developed by SNOEK *et al.* (2012), to determine the three main hyper-parameters for the NN architecture, where the optimal hyper-parameters were chosen to be those values that yielded the lowest error on the validation data. WU *et al.* (2018) also utilized an invariant database and decomposed the Reynolds stress tensor into an implicit linear and explicit nonlinear parts and predicted them separately. GENEVA and ZABARAS (2019) proposed an invariant Bayesian deep NN, used to predict the anisotropic tensor component of the Reynolds stress tensor. Based on the results obtained by LING *et al.* (2016b), KAANDORP and DWIGHT (2020) proposed the use of a tensor based RF, trained on a database of DNS and LES simulations for varied flows, to predict the Reynolds stress anisotropy tensor and use this tensor as a turbulence model within a custom RANS solver, for multiple flow patterns.

Aiming on the accuracy of available DNS data on the literature, THOMPSON *et al.* (2016) demonstrated the impact of statistical error associated with the Reynolds stress tensor provided by DNS data on RANS modeling that uses this quantity as target. ANDRADE *et al.* (2018) extended the work done by THOMPSON *et al.* (2016) and investigated the convergence rate of the Reynolds stress tensor and mean velocity fields of turbulent plane channel and pipe flow, applying a statistical uncertainty analysis to quantify the associated errors. Motivated by these results, CRUZ *et al.* (2019) proposed the use of a modified Reynolds stress tensor divergent, nominated Reynolds force vector, which can be calculated from first order statistic quantities, usually more converged in DNS simulations than second order statistic quantities as the Reynolds stress tensor.

The motivation of this work is grounded on the promising results using invariant databases for machine learning applications on turbulence obtained by LING *et al.* (2016a), LING *et al.* (2016b), WU *et al.* (2018), GENEVA and ZABARAS (2019) and KAANDORP and DWIGHT (2020), the analysis of statistical error of DNS data studied by THOMPSON *et al.* (2016) and the results obtained by CRUZ *et al.* (2019) using a modified divergent of the Reynolds stress tensor.

This work aims to investigate the influence of the convergence on DNS data on ML based turbulence models, employing the Reynolds stress tensor and the Reynolds force vector as set of outputs for ML models composed of invariant tensors and vectors database, applied on RANS simulations to correct the mean velocity fields, by pre-processing the square duct DNS data made available by PINELLI *et al.* (2010), dividing the data in quadrants, merging and imposing symmetry to them, in order to emulate longer simulation times and hence, obtain better convergence of related quantities.

Chapter 2

Literature Review

2.1 Turbulence Modeling

The governing equations in fluid mechanics are the mass conservation law and the Navier-Stokes equation. For the sake of simplicity, in this work only flows of incompressible Newtonian fluids with constant properties and without body forces will be analyzed.

For an incompressible flow, the conservation of mass is reduced to the called continuity equation

$$\nabla \cdot \mathbf{u} = 0. \tag{2.1}$$

The Navier-Stokes equation for incompressible Newtonian flows can be written as

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j}. \tag{2.2}$$

This equation is particularly interesting and complex as it contains a nonlinear quadratic term in the velocity u_i, present in the last term on the left hand side, which is the advective acceleration term.

This term leads to most of the complex and rich phenomena of fluid mechanics, in which turbulence is a part of, and as stated by POPE (2000), in turbulent flows this term amplifies the perturbations present in the flow. There are no prospects of a simple analytic theory for turbulent flows, the ultimate objective is to obtain a model that can be used to calculate quantities of interest and practical relevance. For that, the use of the constant developing power of digital computers is a hope to achieve the objective of calculating the relevant properties of turbulent flows. With the advent of modern computers, numerical solution of complex fluid phenomena has become viable.

It is important to highlight the particular properties present in turbulent flows that make it difficult to develop an accurate theory or model. The velocity field is three-dimensional, time-dependent and random. The largest turbulent motions are as large as the characteristic width of the flow and therefore, are affected by the boundary geometry. As turbulent flows have multiple length and time scales, present simultaneously, where higher resolutions means solving for smaller scales, the problem lies in having sufficient spatial and temporal resolution to obtain accurate results.

2.1.1 Reynolds Average Navier-Stokes

The most disseminated way to simulate turbulent flows is using the Reynolds Average Navier Stokes (RANS) equations. This framework is the least accurate, when compared with LES and DNS, but is the most computationally viable. RANS simulations can be steady or unsteady, depending on the problem.

Although computer technology has evolved greatly in the last years, it is still impractical to use DNS and a large extent of LES for large scale flows so, despite being more inaccurate, RANS is still the most used simulation in engineering applications. Due to this fact, a large amount of effort is dedicated to increase accuracy of the models based on the RANS equations.

RANS equations are derived by applying the average operator to the Navier Stokes equations (2.2). To the extent of understanding this operation, several rules must be explained. In this approach, a sample field ϕ will be decomposed in a time average, as a mean $\langle\phi\rangle$ and a fluctuating part ϕ', so that

$$\phi = \langle\phi\rangle + \phi' \tag{2.3}$$

WILCOX (2006) asserted that by applying a time average to the sample field, the mean of a time average is the average itself,

$$\langle\langle\phi\rangle\rangle = \langle\phi\rangle. \tag{2.4}$$

Then, it is important to observe that this operator is linear. Considering the linearity of the mean operator and Eq. (2.4), it is observed that the mean of a fluctuation is null

$$\langle\phi'\rangle = 0. \tag{2.5}$$

Continuing to explore the linearity of the average operator, it is necessary to apply it to differentiation. As a result it is possible to affirm that the derivative of the average is the average of the derivative,

$$\frac{\partial\langle\phi\rangle}{\partial x_i} = \left\langle\frac{\partial\phi}{\partial x_i}\right\rangle. \tag{2.6}$$

The last important property concerns the product average of two sample fields,

$$\langle\phi\psi\rangle = \langle\phi\rangle\langle\psi\rangle + \langle\phi'\psi'\rangle\,. \tag{2.7}$$

REYNOLDS (1895) applied this decomposition to the velocity and pressure fields. Applying the Reynolds decomposition, referred in Eq. (2.3), using the relations expressed in Eqs. 2.4, 2.5 and 2.7, into the continuity equation Eq. (2.1) and the Navier-Stokes equation (2.2), the following equations obtained are called the Reynolds Average Navier Stokes (RANS) equations

$$\nabla\cdot\langle\mathbf{u}\rangle = 0 \quad \text{and} \quad \nabla\cdot\mathbf{u}' = 0, \tag{2.8}$$

$$\frac{\partial\langle u_i\rangle}{\partial t} + \langle u_j\rangle\frac{\partial\langle u_i\rangle}{\partial x_j} = -\frac{1}{\rho}\frac{\partial\langle p\rangle}{\partial x_i} + \nu\frac{\partial^2\langle u_i\rangle}{\partial x_j\partial x_j} - \frac{\partial\langle u_i'u_j'\rangle}{\partial x_j}. \tag{2.9}$$

This operation results in the appearance of an extra term $-\langle u_i'u_j'\rangle$. This extra term is often referred to as the Reynolds tensor ($\mathbf{R}$). As observed by WILCOX (2006), this tensor is a symmetric tensor, and thus has six independent components.

The Reynolds stresses results in more unknown parameters than equations, which is known as the closure problem. Thus, the Reynolds tensor has to be modeled. The main idea of RANS equations is to solve for the average field, which theoretically would be the average of DNS, without having access to the time dependent data. There are many different models to estimate the Reynolds stresses, which will be discussed hereafter.

2.1.2 Closure Problem

As mentioned previously, with the appearance of the Reynolds tensor, the system of equations becomes indeterminate, as there are more parameters than equations. Thus 6 more equations are needed in order to close the problem. The initial approach would be to develop a transport equation for the Reynolds stress tensor. However, by applying the average operator on the nonlinear terms, a third-order statistical moment emerges. This means a new third order tensor, representing 27 new variables.

A new transport equation could be deduced for this term, however, there would be a larger order parameter with even more variables to be defined. This suggests that is impossible to infer a closure for the system of equations, as stated by TEN-

NEKES and LUMLEY (1972) that in the customary description of turbulence, there are always more unknowns quantities than equations, which configure the closure problem.

In order to bypass the closure problem, the turbulence stresses must be modeled.

2.1.3 RANS Modeling

There is a extensive amount of RANS turbulence models. This fact shows the nonexistence of an universal model that can solve the closure problem mentioned above.

The most common are based on the Boussinesq hypothesis, BOUSSINESQ (1877), which are also called linear eddy viscosity models. They have this name because the Reynolds tensor, is represented through a linear relation between the anisotropic part of $\langle \mathbf{R} \rangle$ and the mean strain rate, $\langle \mathbf{D} \rangle$, of the flow with an eddy viscosity, ν_T,

$$\mathbf{R} = 2\nu_T \langle \mathbf{D} \rangle - \frac{2}{3}\kappa \mathbf{I} \tag{2.10}$$

where κ is the turbulent kinetic energy, defined as $\kappa \equiv 1/2\,\mathrm{Tr}(\mathbf{R})$.

With this hypothesis, the closure problem has not been solved yet, but has been reduced from six extra variables to only one, being the eddy viscosity. Although being called "viscosity", the eddy viscosity differs from the molecular viscosity, as it is not a property of the fluid, but a characteristic of the flow pattern. In a first analysis, the Boussinesq hypothesis seems reasonable, for it suggests an increase in the effective viscosity of the mean flow. This is in agreement with the characteristics of turbulence of high diffusivity and dissipation.

But the Boussinesq hypothesis has some limitations. As stated by POPE (1975), the explicit description of turbulent stresses as a function of the mean strain rate alone is not correct. Also, POPE (1975) states that even if an explicit relation between $\mathbf{R}$ and $\langle \mathbf{D} \rangle$ could be defined, it would not be a linear one. Even for simple shear flows, it is noticed that the Boussinesq hypothesis is not accurate enough in representation of the Reynolds stress tensor. The linear relation between $\mathbf{R}$ and $\langle \mathbf{D} \rangle$ is concluded to be not enough to capture all the anisotropy of $\mathbf{R}$.

Despite this limitation, according to SCHMITT (2007), RANS simulations using the Boussinesq hypothesis are satisfactory enough, and computationally cheap, for a wide sample of applications.

The hurdle in predicting the Reynolds stress tensor on RANS simulation is modeling eddy viscosity. The mathematical approach to achieve this modeling can be done by a dimensional analysis,

$$[\nu_T] = \left[\frac{L^2}{T}\right] = [L] \times \left[\frac{L}{T}\right] = [L] \times [U] \tag{2.11}$$

Thus, the eddy viscosity is written as a function of a characteristic length [**L**] and a characteristic velocity [**U**]. Models that are based on the Boussinesq hypothesis can be classified depending on the number of transport equations used to model turbulent quantities used to calculate the eddy viscosity.

Algebric or zero-equation models: These models employ an algebraic equation, based on turbulent features, to determine the turbulent viscosity, ν_T;

One-equation models: In these models, one partial differential transport equation is solved for a particular turbulent property. In general, the turbulent property of reference is the turbulent kinetic energy, κ. A second property, commonly a length scale, is then provided using an algebraic expression;

Two-Equation models: These models use two partial differential transport equations of turbulent properties, being defined then, as complete closure models, as stated by POPE (2000). The common turbulent properties solved in this model are the turbulent kinetic energy, κ, the rate of dissipation, ε, and the specific rate of dissipation, ω.

The most widely used models in engineering applications are two-equation models, which involve 2 transport equations. Next, the turbulence model used in this work will be described.

$\kappa - \varepsilon$ Model

POPE (2000) describes the $\kappa - \varepsilon$ model as the most widely used complete turbulence model. The development of the standard $\kappa - \varepsilon$ model is credited to JONES and LAUNDER (1972) and the model was improved by LAUNDER and SHARMA (1974). In addition to the turbulent viscosity hypothesis, the $\kappa - \varepsilon$ model consists of a model transport equation for the turbulent kinetic energy, κ, a model transport equation for the rate of dissipation, ε and the specification of the turbulent viscosity a function of only κ and ε

$$[U] = \kappa^{1/2} \quad \text{and} \quad [L] = \frac{1}{\varepsilon}\kappa^{3/2}. \tag{2.12}$$

Applying these relations to Eq. (2.11), the turbulent viscosity for this model is defined as

$$\nu_t = C_\mu \frac{\kappa^2}{\varepsilon}, \tag{2.13}$$

where C_μ is a constant coefficient, chosen to conform to the observed relationship between shear stress and velocity gradient in a simple boundary layer, (DAVIDSON, 2004).

In order to obtain the transport equation for κ, it is necessary to manipulate the RANS equation, Eq. (2.9), by subtracting it from the Navier-Stokes equation, Eq. (2.2), then multiplying it by u' and applying the time average to all terms of the result equation

$$\begin{aligned}\frac{\partial \kappa}{\partial t} + \langle u_j \rangle \frac{\partial \kappa}{\partial x_j} =& \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_\kappa}\right)\frac{\partial \kappa}{\partial x_j}\right] \\ &+ \left[\nu_t\left(\frac{\partial \langle u_i \rangle}{\partial x_j} + \frac{\partial \langle u_j \rangle}{\partial x_i}\right)\right]\frac{\partial \langle u_i \rangle}{\partial x_j} - \varepsilon.\end{aligned} \tag{2.14}$$

The transport equation for ε, is written in a analogous manner to the transport equation for κ yielding

$$\begin{aligned}\frac{\partial \varepsilon}{\partial t} + \langle u_j \rangle \frac{\partial \varepsilon}{\partial x_j} =& \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_\varepsilon}\right)\frac{\partial \varepsilon}{\partial x_j}\right] \\ &+ C_{\varepsilon 1}\frac{\varepsilon}{\kappa}\left[\nu_t\left(\frac{\partial \langle u_i \rangle}{\partial x_j} + \frac{\partial \langle u_j \rangle}{\partial x_i}\right)\right]\frac{\partial \langle u_i \rangle}{\partial x_j} - C_{\varepsilon 2}\frac{\varepsilon^2}{\kappa}.\end{aligned} \tag{2.15}$$

The coefficients present in Eqs. 2.13, 2.14 and 2.15 are obtained from the correlation of experimental data for multiple turbulent flows and are shown in Table. (2.1).

Table 2.1: $\kappa - \varepsilon$ coefficients, (WILCOX, 2006)

C_μ	$C_{\varepsilon 1}$	$C_{\varepsilon 2}$	σ_κ	σ_ε
0.09	1.44	1.92	1.0	1.3

When evaluating near-walls regions, intensified gradients of the flow properties are observed. An adequate description of the flow properties in these regions requires a refined discretization of the wall-region, whose influence in relation to the required computational effort is obvious. For high Reynolds numbers, it is possible to avoid solving the governing equations in the regions close to the wall by assuming the fully developed turbulent boundary layer hypothesis. In this case, the velocity field in the logarithmic region can be described directly by the classic wall law.

2.2 Neural Networks

Machine learning (ML) is a method of data analysis that automates the construction of analytical models. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with the least human intervention. ML algorithms find natural patterns in data that generate insight and help making better predictions.

There are several ML techniques, as stated by MARSLAND (2014), *e.g.*: Support Vector Machine (SVM), decision trees, random forests, genetic algorithm, k-nearest neighbor, discriminant analysis, logistic regression, and Neural Networks (NN).

In this work NN was chosen for the flexibility of the architecture and promising results obtained by LING *et al.* (2016a) and CRUZ *et al.* (2019).

2.2.1 Artificial Neuron

The idea behind NN is to mimic how the human brain operates and recreate it, because the human brain seems to be one of the most powerful tools on the planet for learning, adapting skills and applying them. If computers could copy that, than it would be possible to construct a powerful predicting mechanism.

Comparing with computer science, the brain could be interpreted as a massive parallel processing performed by approximately 100 billions of processors. These processor are called neurons and its structure is shown in Fig. 2.1.

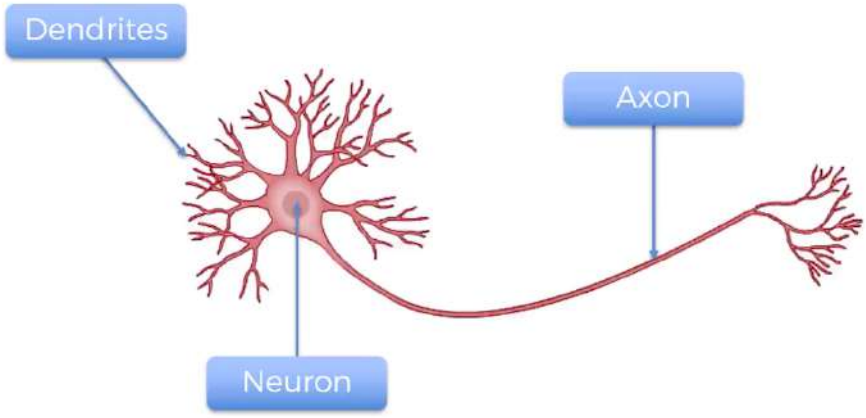


Figure 2.1: Biological neuron structure, (SUPERDATASCIENCE).

The first step to mimic the human brain is to recreate a neuron. The brain neuron is composed of a neuron body, an axon, a dendrite and the synapses

Neuron Body: The most important part of the neuron, where the signals originated on other neurons is processed. As it contains the nucleus, most protein

synthesis occur here. It is also called "Sum" as it sums all the input signals that comes from other neurons.

Axon: Transmits electrical signals generated in the neuron body to other neurons.

Dendrite: Are cellular extensions with many branches that leave the neuron body. This is where the majority of input to the neuron occurs.

Synapse: Is the connection between the neuron axon and other neuron's dendrites.

The interaction between axon and dendrite occurs in all of the 100 billions neurons simultaneously. This massive amount of synaptic connections describe the powerful parallel process capacity of the human brain. This provides the brain with ability to recognize patterns, solve complex problems, learn and make predictions.

Based on the biological neuron structure described above, MCCULLOCH and PITTS (1943) proposed the first mathematical description of an artificial neuron. The purpose of a mathematical model is that it extracts only the essentials required to accurately represent the neuron, removing all of the unnecessary details. The proposed mathematical description has the following structure:

Inputs (in): A simulation of the signals that come from other neurons by the dendrites, receiving the data that come from other neurons connected to it.

Weights (w): Any data received by the artificial neuron is weighted. It correspond to the synapses, where the highest weights are more relevant.

Bias (b): Is an additional set of weights that require no input and correspond to the final output when the neural network has a null input. Without the bias node no layer would be able to reproduce an output that differs from zero, when the feature values are null.

Activation function (φ): Represents the metabolic process that occurs in the neuron body, where a signal is created from the sum of all weighted inputs that arrived at the neuron. It was first formulated by MCCULLOCH and PITTS (1943) as a binary function, but later other mathematical interpretations were suggested, as the bias term. Bias term is a number added to the sum operation of all weighted inputs. These updates improve the performance of the method.

Output (out): Represents the signal transmitted by the neuron axon. It results of the application of activation functions on weighted sums of all inputs of the neuron plus the bias.

The artificial neurons are placed in layers, where neurons can only connect with other neurons from previous or further layers.

Thus, the j^{th} neuron's output, of the m^{th} layer, $(out_j^{(m)})$ is the application of the activation function φ in the weighted sum of signals that outputs from each one of the back neuron's layer n,

$$out_j^{(m)} = \varphi\left(in_j^{(m)}\right) = \varphi\left(\sum_{i=1}^{N} w_i^{(m)}\, \mathrm{out}_i^{(n)} + b_j^{(m)}\right) \tag{2.16}$$

In order to exemplify Eq. (2.16), a graphic model is shown in Fig. 2.2.

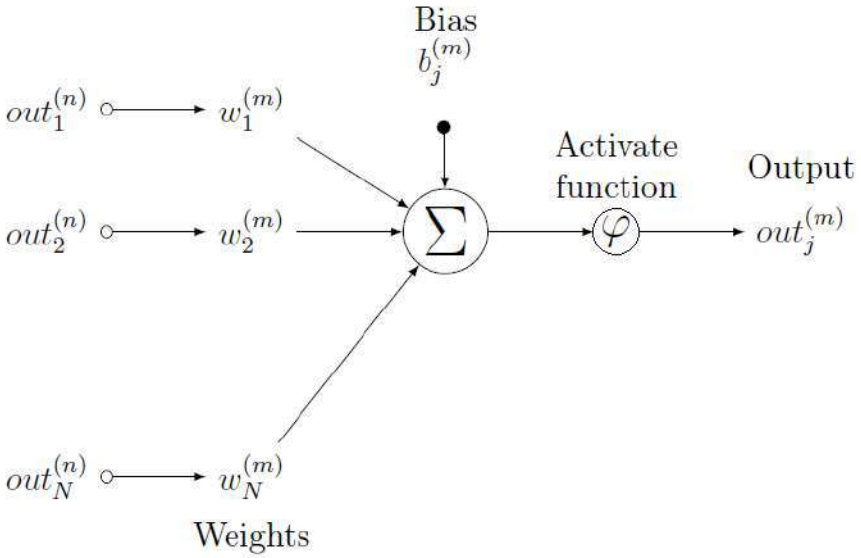


Figure 2.2: j^{th} neuron of the m^{th} layer

The most popular activation functions are:
the binary step function

$$\varphi\left(in_j^{(m)}\right) = \left\{ \begin{array}{l} 1, in_j^{(m)} \geq 0 \\ 0, in_j^{(m)} < 0 \end{array} \right\}, \tag{2.17}$$

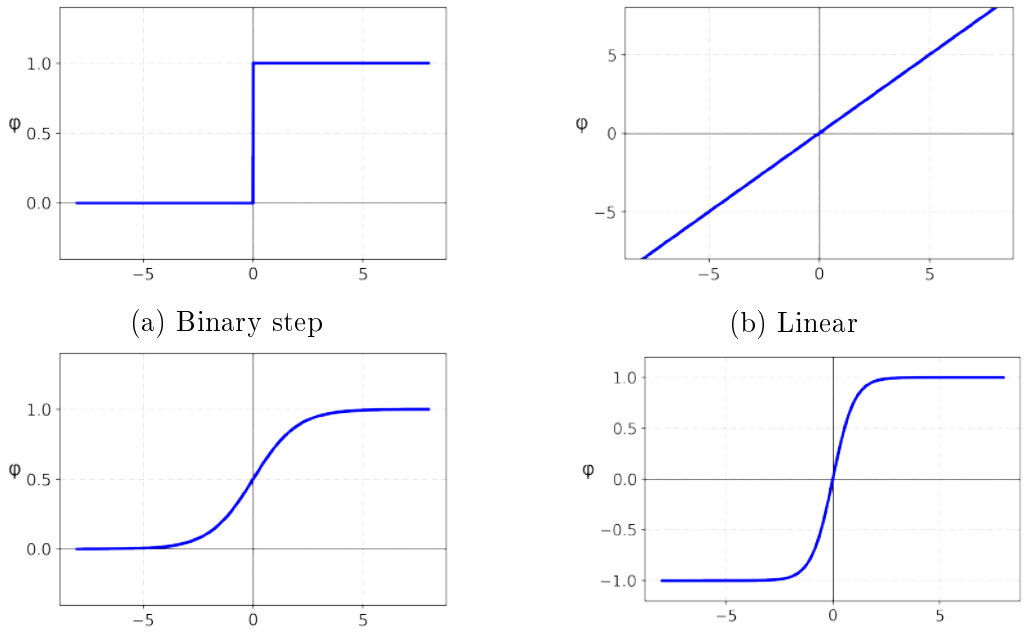
the linear function

$$\varphi\left(in_j^{(m)}\right) = k \cdot in_j^{(m)}, \tag{2.18}$$

the sigmoid function

$$\varphi\left(in_j^{(m)}\right) = \frac{1}{1 + e^{\left(-in_j^{(m)}\right)}}, \tag{2.19}$$

and the hyperbolic tangent function



(a) Binary step

(b) Linear

(c) Sigmoid

(d) Hyperbolic tangent

Figure 2.3: Activation function plots.

$$\varphi\left(in_j^{(m)}\right) = \left(\frac{e^{in_j^{(m)}} - e^{-in_j^{(m)}}}{e^{in_j^{(m)}} + e^{-in_j^{(m)}}}\right) \tag{2.20}$$

Fig. 2.3 shows the plot for the activation functions mentioned above. The non-linear activation functions as the sigmoid and hyperbolic tangent gives the neural network capability to perform non-linear mapping between the inputs **In** and the target **Out**, as shown in Eq. (2.21).

$$\mathbf{Out} = NN(\mathbf{w}, \mathbf{b}; \mathbf{In}) \tag{2.21}$$

CYBENKO (1989) proved the Universal Approximation Theorem, ensuring the existence of a NN capable of mapping **In** in **Out**. In addition, the application of non-linear activation functions helps the NN training process, because they are differentiable and their derivatives are simple to be calculated.

2.2.2 Architecture

With the mathematical model of the neuron established, it is clear that using only one neuron isn't interesting. It can't do very much and given the same set of inputs, the output of the neuron never varies, so it isn't able to learn. So to make the artificial neuron useful, it must be able to learn, and then, a set of neurons can be put together to do something useful.

The first question is how neurons can learn. Looking at the proposed artificial

neuron, the only variables are the weights, bias and activation function. So the learning process do not happen in the neurons at all, but in between the neurons, in the way that they are connected. A collection of neurons together with a set of inputs and weights to fasten the inputs is called a *perceptron*. In a perceptron, the first layer is called the input layer, the last layer is called the output layer and all the layers in between the input and output layers are called hidden layers. Neural networks whose outputs from each layer are not connected with previous layers, *i.e.* are only connected to forward layers, are called feed-forward networks. A feed-forward NN that contains non-linear neurons in their hidden layers is also called a multi layer perceptron (MLP). An example of a MLP is shown in Fig. 2.4.

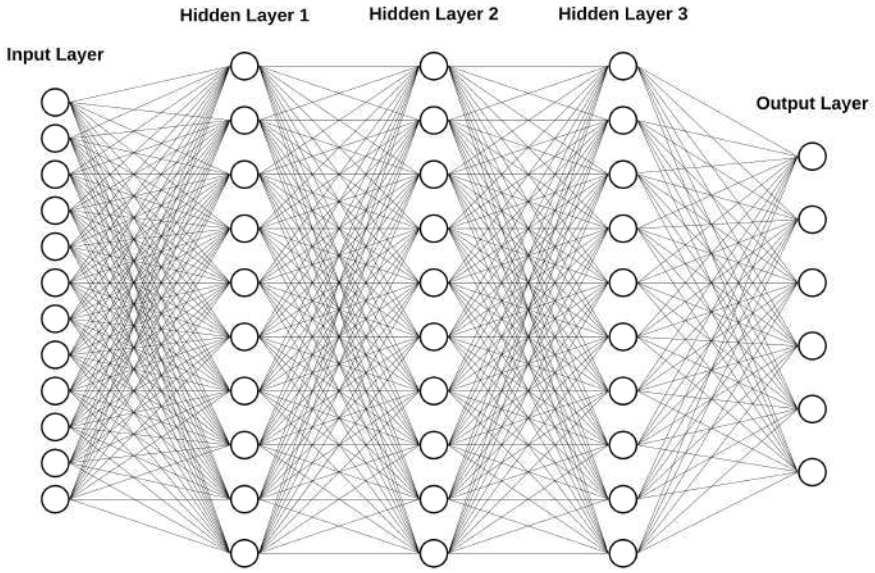


Figure 2.4: Multi layer perceptron example

Therefore, what composes the MLP architecture is the number of neurons per layer, the number of hidden layers and the activation functions of each layer. The amount of neurons in a MLP is critical. An excess of neurons can cause over-fitting, which is the loss of generalization capacity of the NN and a small number of neurons can cause a loss of abstraction capacity, where the NN cannot capture all the patterns present in the data used in the training stage.

2.2.3 Back-Propagation

By default, when initializing a NN, the values for weights and bias are random numbers. Therefore, associated to theses weights and bias there is and error between the regression results obtained by the NN ($\tilde{\mathbf{Out}}$) and the target values (**Out**) available in the database. So this error, called Cost Function (C), is a function that depends only on the values of weight and bias,

$$C = C(\mathbf{w}, \mathbf{b}). \tag{2.22}$$

The training process in a NN algorithm is the search for the set of weights and bias that minimizes this function. This algorithm is usually called back-propagation, in ML applications.

The back-propagation algorithm became important in neural network applications when RUMELHART *et al.* (1986) described several NN where back-propagation showed a faster training process than other earlier approaches to learning, making it possible to use NN to solve problems previously considered unfeasible. The heart of back-propagation is an expression for the partial derivative of the cost function with respect to any weight $\partial C/\partial w$ or bias in the NN. This tells how quickly the cost changes when the weights and bias are modified. The main cost functions used in regression ML applications are:

the mean absolute error

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|\tilde{Out}^{(i)} - Out^{(i)}\right|, \tag{2.23}$$

and the mean square error

$$MSE = \frac{1}{2n}\sum_{i=1}^{n}\left(\tilde{Out}^{(i)} - Out^{(i)}\right)^2, \tag{2.24}$$

where n in Eqs. (2.23) and (2.24) is the number of samples in the database.

So, once each one of the cost derivatives associated with all weights and bias are calculated, techniques called optimization functions are employed in order to minimize the cost function.

2.2.4 Optimization Function

In NN application, optimization functions are algorithms used to update attributes such as weights and bias in order to reduce the error associated with the training process. In regression NN the most used methods are the batch gradient descent, the stochastic gradient descent, RMSprop and Adam.

Batch gradient descent

Gradient descent functions are an iterative optimization algorithms which employ the use of gradients in order to reduce the cost function. The gradient indicates the direction which minimizes the loss. Batch gradient descent, also knows as vanilla gradient descent, is a way to minimize an objective function, $J(\theta)$, parametrized by

a model's parameters, θ, by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$, as asserted by RUDER (2016). In other words, the objective is to follow the direction of the slope of the surface created by the objective function downhill, until a valley is reached, as can be seen in Fig. 2.5.

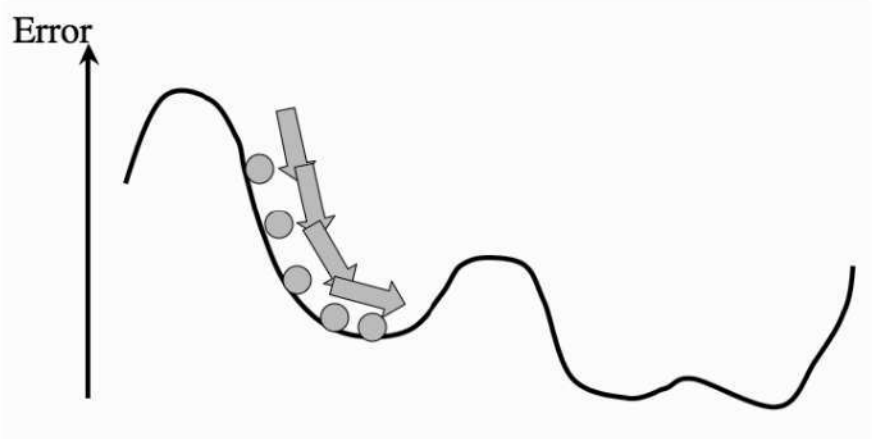


Figure 2.5: Gradient descent updates the weights and bias so that the error goes downhill until it reaches a valley, (MARSLAND, 2014).

This method computes the gradient of the objective function to the model parameter, θ, for the whole training data

$$g_t = \nabla_\theta J(\theta) \tag{2.25}$$

$$\theta = \theta - \eta g_t \tag{2.26}$$

where η is the learning rate, a hyper-parameter that controls how much to change the model in response to the estimated cost function, each time the parameters are updated during training and is usually a number between 0 and 1. As it is necessary to calculate the gradients for the whole dataset every update, this method is considered to be slow, requires large memory and may trap at local minima, as shown in Fig. 2.5. The advantages of this method is the ease of computation, implementation and understanding.

Stochastic gradient descent

Unlike the batch gradient descent, the stochastic gradient descent (SGD), performs a parameter update for each training example, $x^{(i)}$ and label, $y^{(i)}$,

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^{(i)}; y^{(i)}\right). \tag{2.27}$$

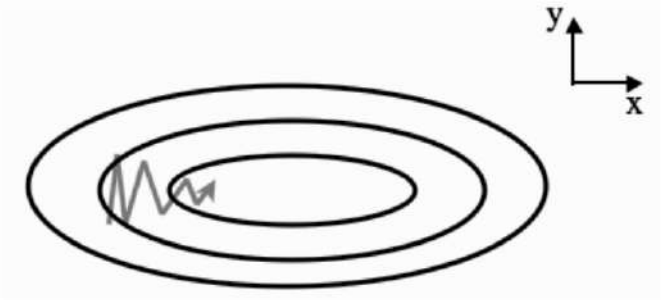


Figure 2.6: Momentum added to SGD, (MARSLAND, 2014).

By performing one update at a time SGD avoids the redundant computations for large databases, present in batch gradient descent, which recomputes gradients for similar examples before each parameter update. As the parameters are frequently updated, it results in a high variance and fluctuations in cost functions at different intensities. SGD also may continue the iterative process even after achieving the global minimal point. RUDER (2016) asserted that to obtain the same convergence as batch gradient descent, SGD needs to slowly reduce the value of learning rate. In order to improve the convergence of SGD, a momentum method is applied, which accelerate SGD in the relevant direction and dampens oscillations, by adding a fraction, γ, of the update vector of the past time step, to the current update vector

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta g_t \\ \theta_{t+1} &= \theta_t - v_t, \end{aligned} \tag{2.28}$$

where the decay term, γ is usually set near to the unit, with values of 0.9 or similar.

RMSprop

RMSprop, is the root mean square propagation, devised by HINTON (2012). It tries to adjust the learning rates by using a moving average of the squared gradient. The values updates are performed as

$$\begin{aligned} v_t &= \gamma v_{t-1} + (1-\gamma)\, g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} g_t, \end{aligned} \tag{2.29}$$

where γ is the decay term, which ranges from 0 to 1 and ϵ is a small constant for numerical stability. In RMSprop, the learning rate gets adjusted automatically and it chooses a different value for each parameter.

Adam

Adam, the adaptive moment estimation, was developed by KINGMA and BA (2015) and is one of the best and most used optimizer for NN applications. It is another algorithm that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients, v_t,like RMSprop, Adam also keeps an exponentially decaying average of past gradient, m_t

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\, g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2, \end{aligned} \tag{2.30}$$

where β_1 and β_2 are decay factors. m_t can be interpreted as the first moment, or mean, of the gradient and v_t, the second moment, or uncentered variance, of the gradient. As these moments are initialized as null vectors, they are biased towards zero, specifically in the initial time steps and when the decay factor is close to 1, as stated by KINGMA and BA (2015). A bias-corrected first and second moment was computed to overcome these biases

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t}. \end{aligned} \tag{2.31}$$

Then Eq. (2.31) is used to update the parameters, just like RMSprop,

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t. \tag{2.32}$$

KINGMA and BA (2015) proposed default values for the coefficients. 0.9 for β_1, 0.999 for β_2 and 10^{-8} for ϵ. Adam algorithm has a higher computational cost, when compared to other methods, but has fast convergence and rectifies the vanishing learning rate and high variance present in other algorithms.

2.2.5 Validation and Test Groups

While the iterative process of updating the weights and bias is active, the data is analyzed through back-propagation methods in order to minimize the associated error. An epoch is the number that represents each iterative time the data is evaluated. During the training process, while the error is minimized, the neural network is applied in a set of data called validation data. So, the NN is applied on training and validation inputs. Both iterations generates an error. The minimizing process acts on the training error, whereas the validation error is used as a training stop criterion. The importance of the validation data is in the fact that NN loses power

of generalization for data that was not part of the training process, which is the so-called over-fitting of the NN.

When the NN converges, after the training process, it is evaluated. The data group used for the evaluation is called test group. Basically, the NN reads an input and predicts an output. This output is compared to the test output, generating an error metric associated with this NN. The test group is a specific group, differing from both training and validation groups.

2.2.6 Cross-Validation

The result of the error metric associated with the NN has a certain randomness. As the initial weights and bias are set stochastically, a second training process with the same training and validation data, could correspond to a different minimization, therefore, impacting the test error. In order to have a statistically significant response, several training processes with random training validation and test groups are performed. In the end, an average error is calculated with a given standard deviation. This average error will validate the NN. An usual division of the database for NN applications consists of allocating 60% of the data for training, 20% for validation and 20% for testing, performing this division through a completely random algorithm.

2.2.7 Over-fitting

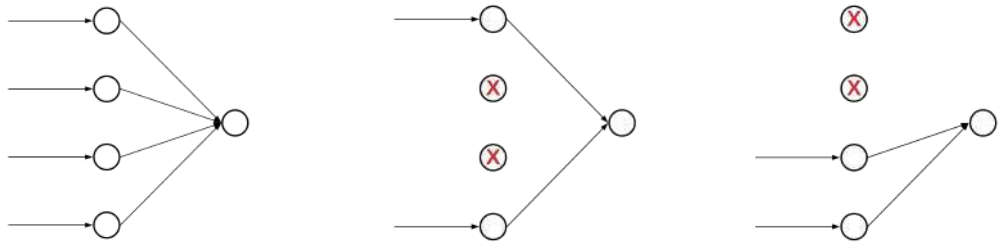
The main objective of a ML model is to have the capability to generalize well, which is the model's ability to give sensitive and accurate outputs to set of inputs it has never seen. Over-fitting happens when the overall error of the training process is small, but the generalization of the model is unreliable. It happens when a model learns too much from the training data, by allowing the training algorithm run for long periods, minimizing the training error but also, catching secondary patterns, such as noises, that may not be needed for the generalization of the model. This results in a NN with high variance.

There are some techniques used in NN applications that reduce the probability of building over-fitted models and consequently, improve the NN model predictive performance. However, implementing these techniques results in more hyperparameters that are needed to be tuned in the model.

Dropout

Dropout is a regularization method that works by randomly setting the outgoing edges of neurons in hidden layers to zero at each update of the training phase. This

method emulates a NN architecture with different configuration at each iteration during the training process, making the layer where the neuron has been dropped out to look-like and be treated-like a layer with a different number of neurons and connectivity to the prior layer, as stated by SRIVASTAVA *et al.* (2014). An illustration of the dropout technique being applied to a NN can be seen in Fig. 2.7. A common way to apply dropout to a NN is to set a dropout ratio, a percentage of neurons in hidden layers which will be dropped out.



(a) Full network (b) Partial network

Figure 2.7: Illustration of dropout regularization.

Weight Constraints

On complex NN models, weights with large values can be employed which brings instability to the model and can result in over-fitting. Smaller weights and bias on a NN model can result in a more stable an less likely to over-fit model. Weight constraint is a trigger that checks the size or magnitude of the weights and scale them so that the values are bellow a predefined threshold. This constraints are enforced on each neuron within a hidden layer, making all the neurons on the hidden layer, subjects to the same constraint. As stated by SRIVASTAVA *et al.* (2014), the use of weight constraints is associated with improvement in the back-propagation algorithm. There are several methods to implement a weight constraint in a NN model, such as

Maximum norm: forces the weights to have a magnitude at or below a given limit.

Non-negative norm: forces the weights to have a positive value.

Unit norm: forces the weights to have a magnitude of 1.0.

Min-Max norm: forces the weights to have a value between a given range.

Early-Stopping

One of the challenges of building a NN model is to set how long the training process will run. Too little time will result in models that did not learn enough from

the database to acquire the capability of generalize. Too much time can result in a over-fit model which means no capability of generalization and a poor performance on the test data. There is a technique called Early Stopping which stops the training process when the chosen performance measure stops improving. Usually this is set on the validation data, where the training process is stopped when the validation error ceases to decrease. Early stopping monitors the performance of the model at each epoch and interrupt the training process after a defined number of epochs where the validation error did not decrease.

Learning Rate Reduction

The learning rate, as explained in Subsection. 2.2.3, controls how much to change the model's weights and bias in response to the estimated error at each epoch. In order to avoid a possible over-fit on long training periods, there is a technique called learning rate reduction which, as the name implies, reduces the value of the learning rate by a given factor, once the learning stagnates. This technique monitors a quantity and if, after a defined number of epochs, there is no improvement, the learning rate is reduced. The quantity often monitored in the learning rate reduction is the validation error.

2.3 Decision Trees

As asserted by MARSLAND (2014), the idea behind decision trees is that a classification process can be broken down into a set of choices about each feature involved in turns, starting at the root, or base, of the tree and progressing down to the leaves, where the output of the classification decision is received. The concept of decision trees is easy to understand and can even be defined as a set of if-then rules.

To better understand the concept of a decision tree, an example is described. Suppose a database composed of ones and zeros as shown in Fig. 2.8. In this case there are three 1's and six 0's, which are the classes of the data, and a separation of the classes using their features is desired. In this case, the features are the color, red or blue, and whether the observation is underlined, striked or not. Therefore, through a set of if-then rules, or questions, it is possible to separate the data. Each if-then rule is called a node and the first node in the example is based on the feature color. After the node, the tree is split in two branches, the Yes branch and the No branch. The first No branch consist only of non underlined blue zeros, so it is not necessary to split the branch further. However, the Yes branch can still be split further. So, another feature can be used to split this branch, splitting the data into

underlined or not. This yields in another two Yes and No branches and to end this classification process one last if-then rule must be made. Then the decision tree is done, being necessary three features in order to split up the database.

This simple example shows the logic of a decision tree. At each node it infers what feature will allow the tree to split the observations in a way that the resulting groups are as distinct from each other as possible.

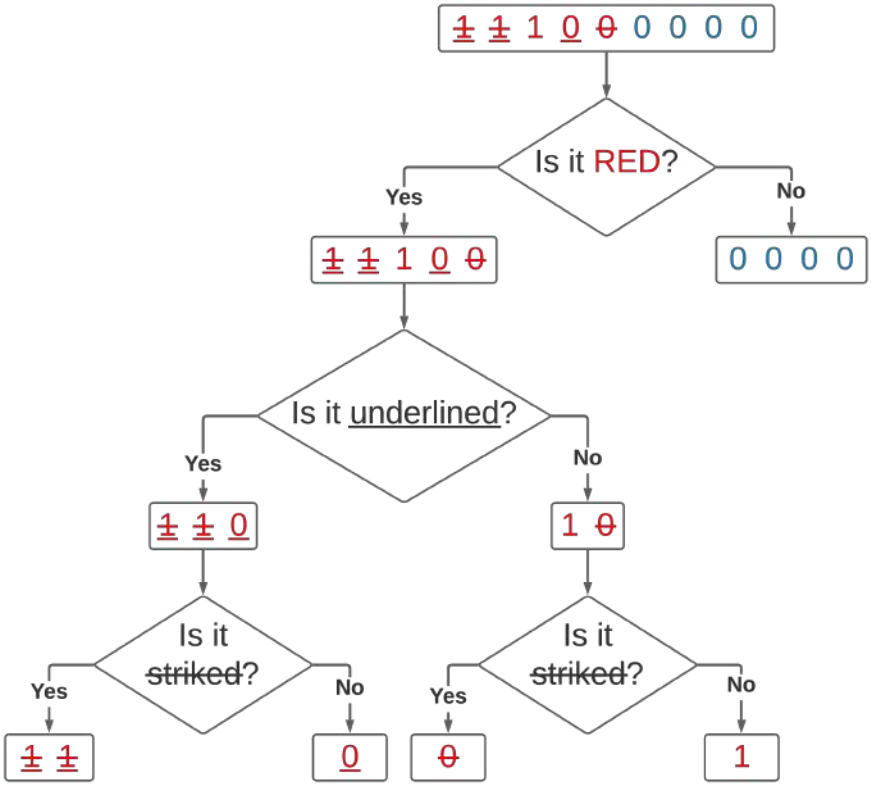


Figure 2.8: Decision tree schematic

In order to build a decision tree for regression, only a simple modification in the model is necessary. In regression models, the inputs and outputs are continuous, therefore, to evaluate the choice of which feature to use next, is also necessary to find the value at which to split the database according to that feature. For this procedure, the MSE from Eq. (2.24), is used. The output is a value at each leaf that, in general, is just a constant value computed as the mean average of all the data-point that are situated in that leaf. This value is the optimal choice in order to minimize the MSE, but also means that the split point can be chosen quickly for a given feature, by minimizing the MSE. This is how a decision tree for regression differs from one for classification.

As stated by BREIMAN (2017), the computational cost of making a decision tree is fairly low and the cost of using it for predicting values is even lower. The computational cost is directly related to the number of nodes in the decision tree.

Also, implementing a decision tree algorithm is considerably easy, when compared to NN and other ML models. However, decision tree algorithms tend to find local optima in the training due to the fact that once the split has been done, there is no going back. Also, decision trees are sensitive to the data they are trained and if the data is changed the prediction results can be quite different, implying a high variance to this model. These reasons make decision trees prone to over-fitting. To avoid these weakness, the random forest algorithm was developed.

2.4 Random Forests

Random forest (RF) is a ML method of supervised learning. The 'forest', refers to an ensemble of decision trees. RF operates by building a multitude of decision trees at the training process and having as the output, the mode of the classes in a classification RF or the mean prediction in regression RF, of each individual decision tree. RF are based on the simplest method of combining classifiers, called bagging, which stands for bootstrap aggregating. A bootstrap sample is a sample taken from the original database with replacement, so that some data can be selected several times and other none at all. The idea behind bagging is that combining models yields a better prediction result. Applying this method implies in a reduced variance for the model and since decision trees have high variance, this is advantageous. Bagging makes each model run independently and after the training, aggregates the outputs, without giving preference to any model. So a RF can be considered a meta-estimator, as it combines the result of multiple predictions from many decision trees, as can be seen in Fig. 2.9. As each tree inside the forest runs independently, they can be ran in parallel, decreasing the computational cost.

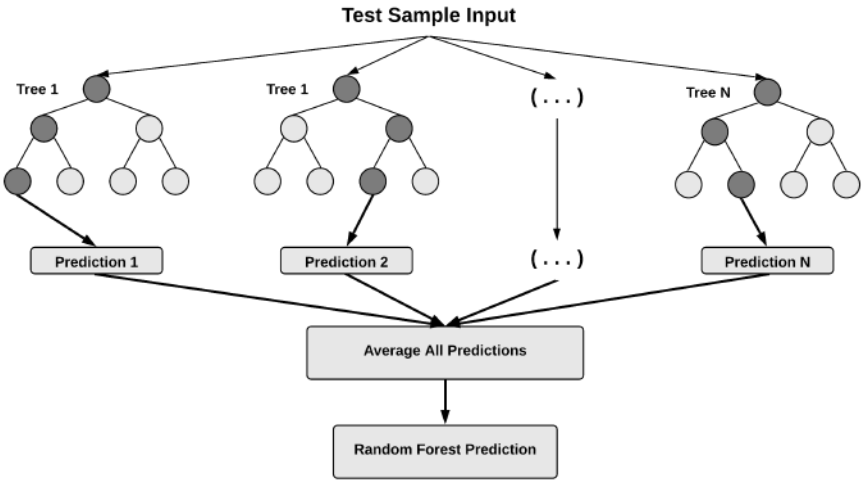


Figure 2.9: Random forest structure.

RF is considered an easier ML model to implement then NN and having a low variance implies that even without the best set of hyper-parameters, a RF can produce great results. However, there are some important hyper-parameters to tune when building a RF, namely:

Number of estimators: Is the number of decision trees inside the forest. In general, a higher number of trees increases the performance of the model and makes the predictions more stable, less prone to variance, but it also increases the computational time

Max features: Is the maximum number of features the RF considers when splitting a node into two branches.

Max depth: Is the maximum depth of each tree, defining the number of nodes at each tree.

Min sample split: Is the minimum number of samples required to split a node.

Min sample leaf: Is the minimum number of leafs required to split a tree node.

Understanding the hyper-parameters on a RF is pretty straightforward and there are not as many when compared to NN. Over-fitting, a recurrent problem in ML model can be easily avoided if there are enough trees in the forest.

2.5 Euclidean Invariance

As stated by LING *et al.* (2016a), the Navier-Stokes equations obey invariance, meaning that the laws of motion do not change for inertial transformations of the frame of reference. Therefore, applying invariant quantities to ML models for turbulence is desired and has shown improvements in previous works where invariance was applied.

As presented by GURTIN (2010), in continuum mechanics, the basic property of a body is that it may occupy regions of the Euclidean point space $\mathcal{E}$, where it is possible to identify the body within region B of $\mathcal{E}$ as it occupies some fixed configuration, called reference configuration, whose choice is arbitrary. A motion of B is a smooth function χ that assigns to each material point $\mathbf{X}$ and time t, a point

$$\mathbf{x} = \chi(\mathbf{X}, t), \tag{2.33}$$

where $\mathbf{x}$ is referred to as the spatial point occupied by $\mathbf{X}$ at time t.

On a frame of reference F, a change of frame is, at each time, a rotation and translation of the observed space. Therefore, a change of frame $F \to F^*$ can be defined, at each time t by a rotation $\mathbf{Q}(t)$ and a spatial point $\mathbf{y}(t)$ and transforms the spatial points $\mathbf{x}$ to spatial points

$$\mathbf{x}^* = \mathbf{y}(t) + \mathbf{Q}(t)(\mathbf{x} - \mathbf{o}), \tag{2.34}$$

where $\mathbf{o}$ is a fixed spatial origin. The tensor $\mathbf{Q}$ is usually referred to as the frame-rotation. The frame-rotation tensor is an orthogonal tensor, where the transpose is equal to the inverse, as

$$\mathbf{Q}^T = \mathbf{Q}^{-1}. \tag{2.35}$$

While a change of frame affects the observed space, it does not affect the reference space, thus, scalar fields such as the density ρ and the viscosity μ are Euclidean invariant.

$$\rho^* = \rho, \quad \mu^* = \mu. \tag{2.36}$$

For a vector field $\mathbf{g}$, it is said to be invariant if it simply rotates with the frame rotation

$$\mathbf{g}^* = \mathbf{Q}\mathbf{g}. \tag{2.37}$$

For a tensor field $\mathbf{G}$, it is said to be invariant if, given invariant vector fields $\mathbf{g}$ and $\mathbf{h}$, at any change in frame

$$\mathbf{h} = \mathbf{G}\mathbf{g} \quad \text{implies that} \quad \mathbf{h}^* = \mathbf{G}^*\mathbf{g}^*. \tag{2.38}$$

applying Eq. (2.37) and (2.38):

$$\begin{aligned} \mathbf{h}^* &= \mathbf{Q}\mathbf{h} \\ &= \mathbf{Q}\mathbf{G}\mathbf{g} \\ &= \mathbf{Q}\mathbf{G}\mathbf{Q}^\top\mathbf{g}^*. \end{aligned} \tag{2.39}$$

As defined in Eq. (2.38), $\mathbf{h}^* = \mathbf{G}^*\mathbf{g}^*$ must be satisfied, therefore,

$$\begin{aligned} \mathbf{G}^*\mathbf{g}^* &= \mathbf{Q}\mathbf{G}\mathbf{Q}^\top\mathbf{g}^* \\ \mathbf{G}^* &= \mathbf{Q}\mathbf{G}\mathbf{Q}^\top. \end{aligned} \tag{2.40}$$

Applying the Euclidean invariance to turbulence, it is possible to evaluate quantities of interest such as the velocity field. Considering the spatial description of the

velocity field $\mathbf{u}(\mathbf{x}, t)$, to express the velocity field in an invariant frame, it transforms to

$$\mathbf{u}^*(\mathbf{x}^*, t) = \mathbf{Q}(t)\mathbf{u}(\mathbf{x}, t) + \dot{\mathbf{y}}(t) + \dot{\mathbf{Q}}(t)(\mathbf{x} - \mathbf{o}). \tag{2.41}$$

It is also possible to evaluate whether quantities such as the velocity gradient, $\mathbf{L}(\mathbf{x}, t) = \nabla\mathbf{u}(\mathbf{x}, t)$ have Euclidean invariance.

$$\mathbf{L}^*(\mathbf{x}^*, t) = \nabla^*\mathbf{u}^*(\mathbf{x}^*, t). \tag{2.42}$$

Using the chain-rule to differentiate Eq. (2.41) in respect to $\mathbf{x}$, while using the relation between $\mathbf{x}$ and $\mathbf{x}^*$ stated in Eq. (2.34), it yields

$$\begin{aligned} (\nabla^*\mathbf{u}^*)\,\mathbf{Q} &= \mathbf{Q}(\nabla\mathbf{u}) + \dot{\mathbf{Q}} \\ &= \left(\mathbf{Q}(\nabla\mathbf{u})\mathbf{Q}^\top + \boldsymbol{\Omega}\right)\mathbf{Q}, \end{aligned} \tag{2.43}$$

where $\Omega = \dot{\mathbf{Q}}\mathbf{Q}^\top$ is the frame-spin tensor, which is an skew-symmetric tensor. So, the velocity gradient transforms according to

$$\mathbf{L}^*(\mathbf{x}^*, t) = \mathbf{Q}(t)\mathbf{L}(\mathbf{x}, t)\mathbf{Q}^\top(t) + \dot{\mathbf{Q}}(t)\mathbf{Q}^\top(t). \tag{2.44}$$

The velocity gradient tensor can be split in two parts, one symmetric and one skew-symmetric, the mean strain rate $\mathbf{D}$ and the vorticity tensor $\mathbf{W}$, as

$$\mathbf{L} = \mathbf{D} + \mathbf{W}. \tag{2.45}$$

Therefore, since the frame-spin Ω is skew-symmetric, the transformed velocity gradient yields in

$$\mathbf{D}^* = \operatorname{sym}\mathbf{L}^* \quad \text{and} \quad \mathbf{W}^* = \operatorname{asym}\mathbf{L}^* \tag{2.46}$$

and the two tensors can be written as

$$\begin{aligned} \mathbf{D}^*(\mathbf{x}^*, t) &= \mathbf{Q}(t)\mathbf{D}(\mathbf{x}, t)\mathbf{Q}^\top(t) \\ \mathbf{W}^*(\mathbf{x}^*, t) &= \mathbf{Q}(t)\mathbf{W}(\mathbf{x}, t)\mathbf{Q}^\top(t) + \dot{\mathbf{Q}}(t)\mathbf{Q}^\top(t). \end{aligned} \tag{2.47}$$

It is possible to observe that only the symmetric part of the velocity gradient, the mean strain rate $\mathbf{D}$, has Euclidean invariance.

Chapter 3

Machine Learning and Turbulence

3.1 Recent Work

In recent years, machine learning applications on computational fluid dynamics simulations of turbulent flows such as RANS closure problem, suggests a new perspective on modeling. The main objective is to correct shortcomings of turbulence models on describing phenomena such as secondary flows and flow separation at boundary layers. It is interesting to use non-linear regression process for describing some characteristics of turbulence modeling, such as the Reynolds stress tensor. For these applications, non-linear regression is created from data coming from high fidelity sources, such as DNS and LES, that provide data targets for machine learning techniques used in the literature. In short, the goal is to extract some inputs from RANS simulations, perform a non-linear regression, and predict some turbulence aspect, such as a new Reynolds stress field, close to a DNS or LES established as target.

Due to limitations in RANS models, especially when applied to situations that differ from those in which the models were calibrated, TRACEY *et al.* (2013) proposed a methodology aimed at improving low-fidelity models of turbulence and combustion, through a ML method named kernel regression. The ML algorithm aims to predict the anisotropy in the Reynolds tensor, using a DNS database as target. This starts with the computation of the eigenvalues λ_i of the normalized Reynolds stress anisotropy tensor

$$\mathbf{b} = \frac{1}{2\kappa}\mathbf{R} - \frac{1}{3}\boldsymbol{\delta}. \tag{3.1}$$

To visualize the anisotropy of the turbulence field, TRACEY *et al.* (2013) proposed the use of the Barycentric map proposed by BANERJEE *et al.* (2008). This mapping represents the Reynolds stress tensor as a point within an equilateral triangle, where the corners correspond to limiting states of anisotropy and the interior

represents a realizable Reynolds stress. These eigenvalues can be used to construct the quantities

$$\begin{aligned} C_{1c} &= \lambda_1 - \lambda_2 \\ C_{2c} &= 2\left(\lambda_2 - \lambda_3\right) \\ C_{3c} &= 3\lambda_3 + 1, \end{aligned} \tag{3.2}$$

where the subscripts represent the one, two and three component limits of the turbulence. Once these quantities are determined, this map can be plotted via the relation

$$\begin{aligned} x_{\text{bary}} &= C_{1c}x_{1c} + C_{2c}x_{2c} + C_{3c}x_{3c} \\ y_{\text{bary}} &= C_{1c}y_{1c} + C_{2c}y_{2c} + C_{3c}y_{3c} \end{aligned} \tag{3.3}$$

where the x_{1c}, y_{1c} and others are the locations of the corners of the Barycentric triangle, as shown in Fig. 3.1.

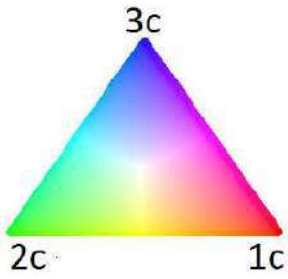
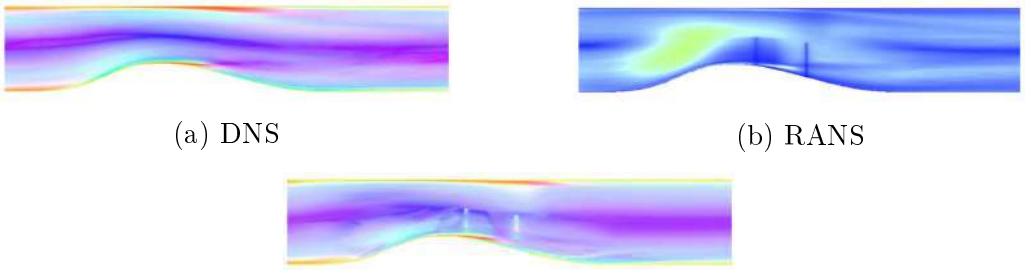


Figure 3.1: Colored map representing the location in the barycentric map, (TRACEY *et al.*, 2013).

The authors applied this methodology to predict a corrected Reynolds stress anisotropy tensor for a periodic hill turbulent flow, a classic pattern flow that presents detachment and reattachment of the boundary layer, modeled using a $\kappa-\omega$ SST turbulence model. The results, shown in Fig. 3.2, revealed the capability of the ML algorithm to correct the eigenvalues of the anisotropic part of $\mathbf{R}$.

Using another ML technique, RF, LING *et al.* (2016c) proposed an improvement of RANS $\kappa - \varepsilon$ capability of accurately represent the Reynolds stress anisotropy, shown in Eq. (3.1), based on LES data for a jet-in-crossflow. Results, shown in Fig. 3.3, revealed significantly improved anisotropy when compared to the default RANS.

LING *et al.* (2016a) also propose a comparison between two approaches to insert invariant properties on the training of a machine learning algorithm. In the first, a basis of invariant inputs is constructed and the ML is trained based on these inputs. In the second, the ML is trained on multiple transformation of the raw



(a) DNS (b) RANS

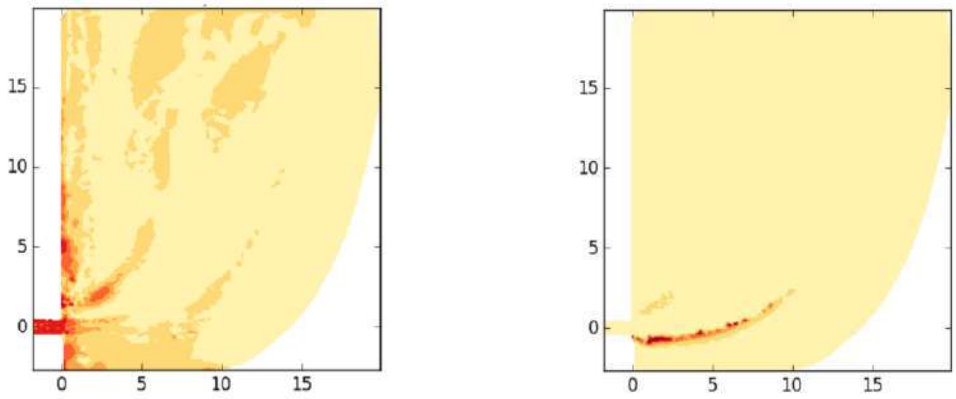
(c) Correction

Figure 3.2: Anisotropy in periodic hill flows, (TRACEY *et al.*, 2013).

input data, until the model learns invariance to that transformation. The authors used two different regression ML algorithm, NN and RF, to validate the approach. The appeal of this method is to give the capability to a ML model to predict fields for similar flows regardless of their physical orientation. The results obtained revealed that models trained on an invariant database showed better prediction capability, as shown in Fig. 3.4, and used less computational effort to achieve these results, which exalts the importance of using invariant inputs in ML models.

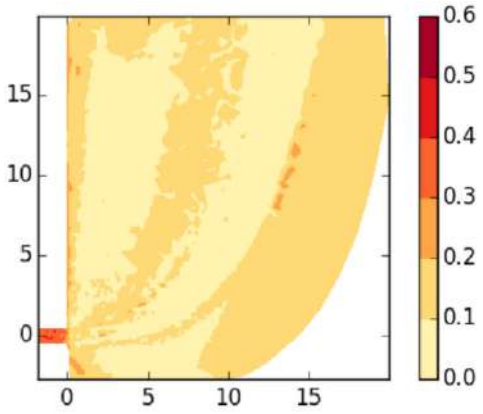
In latter work, LING *et al.* (2016b) presented a method of using deep NN to learn a model for the Reynolds stress anisotropy tensor, which uses a multiplicative layer with invariant tensor basis to embed Galilean invariance into the prediction. The authors compared the results obtained with a non invariant MLP and two RANS models. The model with embedded invariance was called tensor basis neural network (TBNN). The authors used nine different flow databases to train validate and test the ML models. Six were used in training, a duct flow, a channel flow, a perpendicular jet-in-cross-flow, an inclined jet-in-cross-flow around a square cylinder and flow through a converging-diverging channel. One case was used for validation, which was a wall-mounted cube in cross-flow. To test the performance of the ML models, two cases were used, a duct flow and a flow over a wavy wall, which are flows where the Reynolds stress anisotropy plays a key role. The results obtained were compared with two RANS models, one linear eddy viscosity model (LEVM) and one quadratic eddy viscosity model (QEVM), a MLP and the DNS data and can be seen in Fig. 3.5, which revealed that the TBNN was shown to have significantly more accurate predictions than a generic MLP. This corroborates the importance of using an invariant database when applying ML techniques to turbulence modeling.

Inspired by these results, KAANDORP and DWIGHT (2020) proposed a similar approach using, instead of TBNN, a tensor basis random forest (TBRF), which, when compared to the TBNN, is relatively easier to implement and train, due to the fact that it is not necessary to use an optimization algorithm to tune the ML model. This tensor based model was also trained with several flow databases and it was tested with different flow patterns. The results for different flow patterns revealed



(a) LES

(b) RANS



(c) Correction

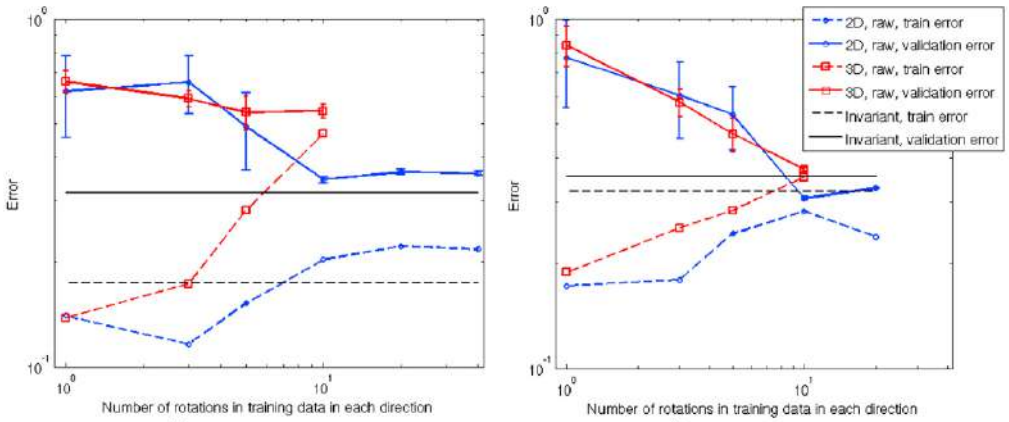
Figure 3.3: Contours of the second anisotropy invariant in a plane, (LING *et al.*, 2016c).

that the TBRF performed in a similar way to the TBNN, which is satisfactory, due to the fact that it is easier to implement. Fig. 3.6, shows one of the flows studied with different ML training data groups, named $C3$ and $C4$.

Influenced by the analysis of statistical error of DNS data studied by THOMPSON *et al.* (2016), where it was shown an intrinsic uncertainty associated with second order statistics such as $\mathbf{R}$, which is not well converged as the mean velocity and pressure fields in DNS, CRUZ *et al.* (2019) proposed a NN that, instead of aiming at quantities derived from $\mathbf{R}$, aimed at the correction of a modified divergent of $\mathbf{R}$, called by the authors, Reynolds force vector, $\mathbf{t}$.

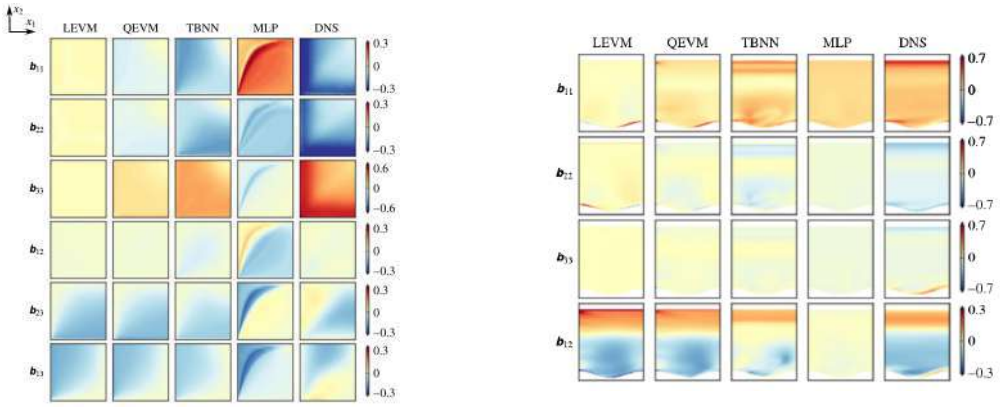
The divergent of $\mathbf{R}$ can be obtained by applying the operator directly to $\mathbf{R}$, but it would propagate and amplify the associated uncertainties. To bypass this problem, the modified divergent of $\mathbf{R}$ was implemented, which computes the quantity indirectly, associating it with the well converged DNS fields in the mean linear momentum equation,

$$\boldsymbol{\nabla} \cdot \mathbf{R} = -(\langle \mathbf{u} \rangle \cdot \boldsymbol{\nabla}) \langle \mathbf{u} \rangle - \frac{1}{\rho} \boldsymbol{\nabla} \langle p \rangle + \nu \boldsymbol{\nabla}^2 \langle \mathbf{u} \rangle. \tag{3.4}$$



(a) Random Forest (b) Neural Network

Figure 3.4: ML model errors as a function of number of training rotations, (LING *et al.*, 2016a).



(a) Duct (b) Wavy wall

Figure 3.5: Predictions of Reynolds stress anisotropy, here stated as $\mathbf{b_{ij}}$, (LING *et al.*, 2016b).

The square duct DNS database used by the authors did not provide a mean pressure field. Therefore, $\mathbf{t}$ was defined as,

$$\mathbf{t} \equiv \boldsymbol{\nabla} \cdot \mathbf{R} + \frac{1}{\rho}\boldsymbol{\nabla}\langle p \rangle, \tag{3.5}$$

and can be calculated only as function of the velocity field, a first order statistics, as,

$$\mathbf{t} = -(\langle \mathbf{u} \rangle \cdot \boldsymbol{\nabla})\langle \mathbf{u} \rangle + \nu \boldsymbol{\nabla}^2 \langle \mathbf{u} \rangle. \tag{3.6}$$

The results obtained by injecting $\mathbf{t}$ in a RANS solver showed to be a considerable improvement over the injection of $\mathbf{R}$, especially in the principal direction of the mean velocity field U_x, as can be seen in Fig. 3.7.

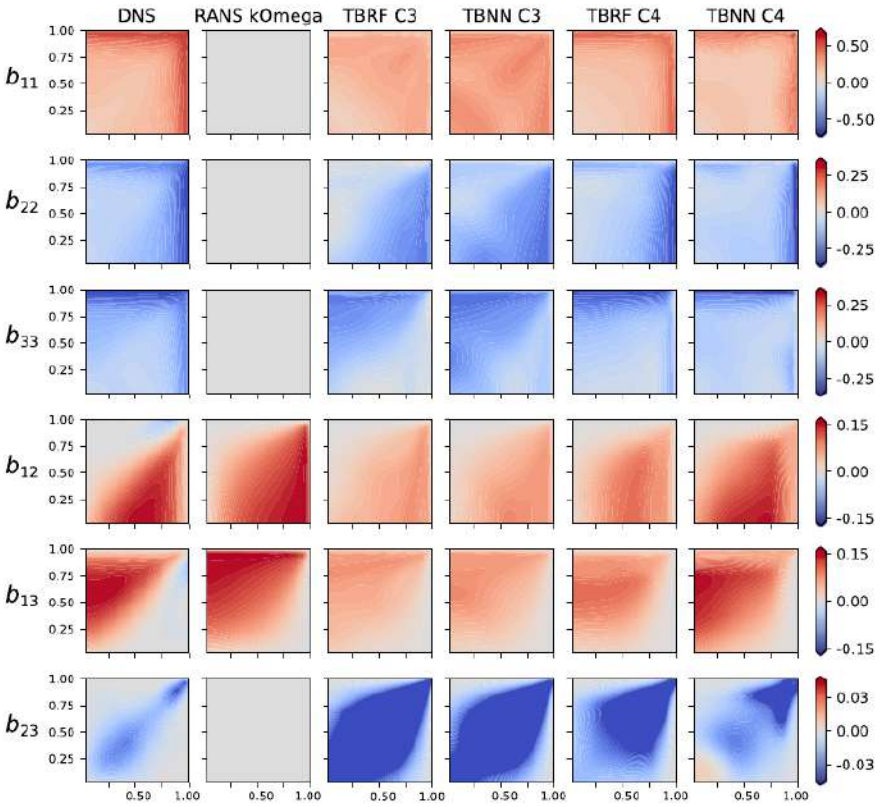


Figure 3.6: Components of the Reynolds stress anisotropy tensor, here stated as $\mathbf{b_{ij}}$, for a square duct flow, (KAANDORP and DWIGHT, 2020).

3.2 DNS Convergence

The effects of imbalance in the momentum equation in DNS simulations, where the residuals of the velocity profile affects turbulence modeling, especially in a RANS approach, where the Reynolds stress tensor is a target, was stated by THOMPSON *et al.* (2016). The authors revealed that some DNS database have an inherent uncertainty associated to second order statistics, such as the Reynolds stress tensor, which is less converged when compared to first order statistics, such as the mean velocity and pressure fields. In order to improve DNS data, the authors proposed a new methodology for statistical error evaluation, which would provide a more accurate second order statistics and, by consequence, more reliable targets for modeling purposes.

As stated by THOMPSON *et al.* (2016), the two most obvious statistics provided in a channel flow DNS database are the mean velocity field and the Reynolds stress tensor associated with the fully developed steady-state flow. Both theses quantities are determined by a hybrid averaging process, where averaging is performed spatially in wall-parallel slabs and in time, using many flows instantaneous snapshots as available, being both subject to discretization and sampling errors. For this analysis, variables with a superscript $+$ are expressed in wall units, the $\hat{()}$ symbol was em-

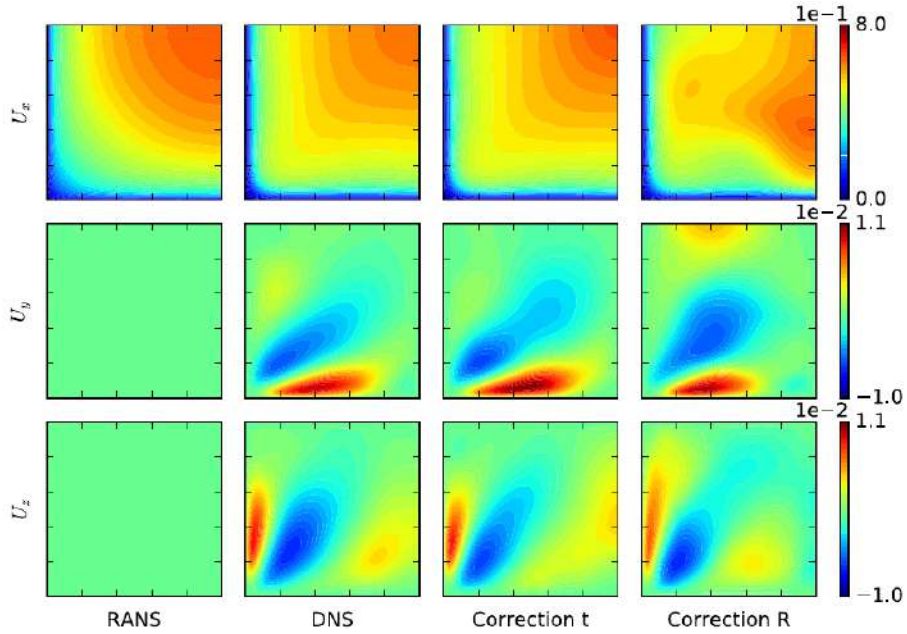


Figure 3.7: Mean velocity field of a square duct flow for RANS, DNS and propagated Reynolds force vector and Reynolds stress tensor, (CRUZ *et al.*, 2019).

ployed to label any variable that is provided by the DNS hybrid space-time average and the $\widetilde{()}$ symbol was used to label a variable that is computed from a conservative equation. Numerical experience in DNS shows that the shear component xy of the Reynolds stress tensor is the most difficult component to converge. Based on that, the authors wrote the following two equations

$$\widetilde{R}_{yx}^{+}\left(y^{+}\right) = 1 - \frac{y^{+}}{Re_{\tau}} - \frac{d\hat{U}^{+}}{dy^{+}}, \tag{3.7}$$

$$\widetilde{U}^{+}\left(y^{+}\right) = y^{+} - \frac{y^{+2}}{2Re_{\tau}} - \int_{0}^{y^{+}} \hat{R}_{yx}^{+}\left(y'\right) dy', \tag{3.8}$$

where $Re_{\tau} \equiv u_{\tau}h/v$ is the usual friction Reynolds number. Eq. (3.7) expresses the yx-component of the Reynolds stress tensor that balances the momentum equation in the x-direction using the mean velocity gradient provided by DNS data as input. Eq. (3.8) expresses the mean velocity profile that would balance the moment equation using a component of the Reynolds stress tensor from DNS data as input. Eq. (3.7) and (3.8) can be used to define two residuals

$$E_{R}\left(y^{+}\right) = \widetilde{R}_{yx}^{+}\left(y^{+}\right) - \widehat{R}_{yx}^{+}\left(y^{+}\right), \tag{3.9}$$

$$E_{U}\left(y^{+}\right) = \widetilde{U}^{+}\left(y^{+}\right) - \widehat{U}^{+}\left(y^{+}\right). \tag{3.10}$$

From Eqs. (3.7) and (3.9),

$$E_R\left(y^+\right) = 1 - \frac{y^+}{Re_\tau} - \frac{d\hat{U}^+}{dy^+} - \hat{R}^+_{yx}\left(y^+\right), \tag{3.11}$$

which can be interpreted as the residual of the momentum balance with respect to steady state fully developed channel flow. To ensure a steady state fully developed flow, $E_R\left(y^+\right)$ computed with DNS data, ought to be as small as possible across the channel width.

From Eqs. (3.8) and (3.11), Eq. (3.10) can be written as

$$E_U\left(y^+\right) = \int_0^{y^+} E_R\left(y'\right) dy', \tag{3.12}$$

which means that $E_U\left(y^+\right)$ is originated from the cumulative error, associated with the residuals from the momentum equation. The impact of the residual of the momentum equation on the velocity profile is cumulative on y^+ and can be significant far from the wall in high Reynolds number simulations, which affects turbulence modeling, such as a RANS approach that uses the Reynolds stress tensor as a target. This proved that the convergence criteria can be formulated based on the velocity residual $E_U\left(y^+\right)$ instead of stress residual $E_R\left(y^+\right)$. This methodology provided a new path for statistical error evaluation for future DNS of plane channel yielding more reliable targets for modeling purposes.

Based on this methodology, ANDRADE *et al.* (2018) proposed an uncertainty analysis for DNS turbulent flow on plane channel and pipe. Fig. 3.8 shows the comparison of residual values, obtained from Eqs. (3.11) and (3.12), $E_R\left(y^+\right)$ (left column) and $E_U\left(y^+\right)$ (right column), for several Re_τ numbers, divided in four regions: (I) the viscous sub-layer, (II) buffer layer, (III) log-law region and (IV) outer layer.

Fig. 3.9 shows the comparison of the same residual errors for various DNS simulation averaging times, for two Re_τ numbers. From this figure, it is easy to notice a direct improvement in accuracy on DNS as the averaging simulation time is increased, as expected. For both residuals, the residual error has a tendency to decrease as the averaging times increases. The results obtained correlates the intrinsic uncertainties present in some DNS database to the insufficient DNS simulation averaging time.

Inspired by the results obtained by CRUZ *et al.* (2019) and the uncertainties associated with the Reynolds stress tensor, RANGEL (2019) proposed splitting the DNS data from a turbulent square duct into four quadrants. These four quadrants were combined resulting in an increased emulated averaging DNS simulation time.

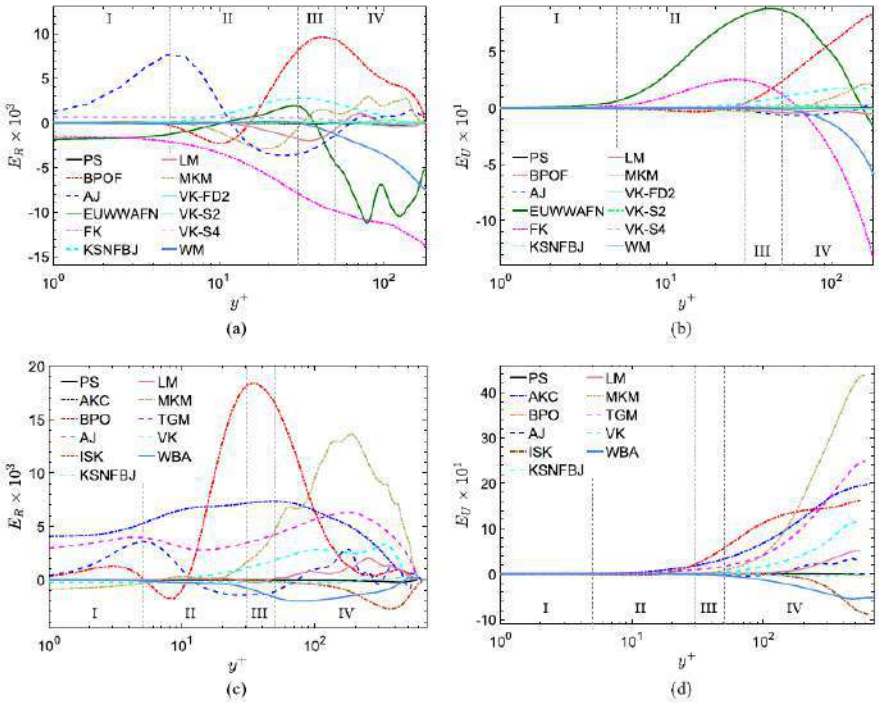


Figure 3.8: Comparison of residual values between several DNS database across a plane channel half-width or radius for pipe flow, (ANDRADE *et al.*, 2018).

The constructed DNS database for different emulated times was used as a ML output, using the same parameters and architecture as CRUZ *et al.* (2019). Figs. 3.10 and 3.11 show the results obtained by the proposed methodology, which confirms the influence of the convergence of DNS database in the correction of the velocity field using a RANS approach, as the lowest errors are associated with the highest emulated averaging times.

The previous works shown in this chapter served as a template for the methodology used in this work, with application of ML techniques in turbulence modeling, use invariant quantities and analysis of the convergence of the DNS database used.

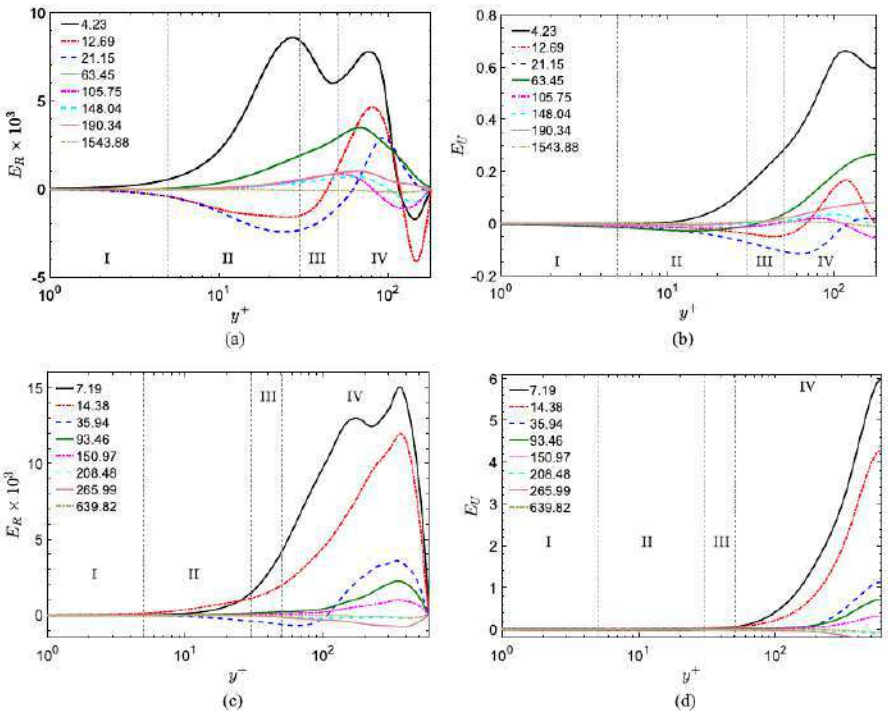


Figure 3.9: Temporal development of residual profiles at several dimensionless averaging times, (ANDRADE *et al.*, 2018).

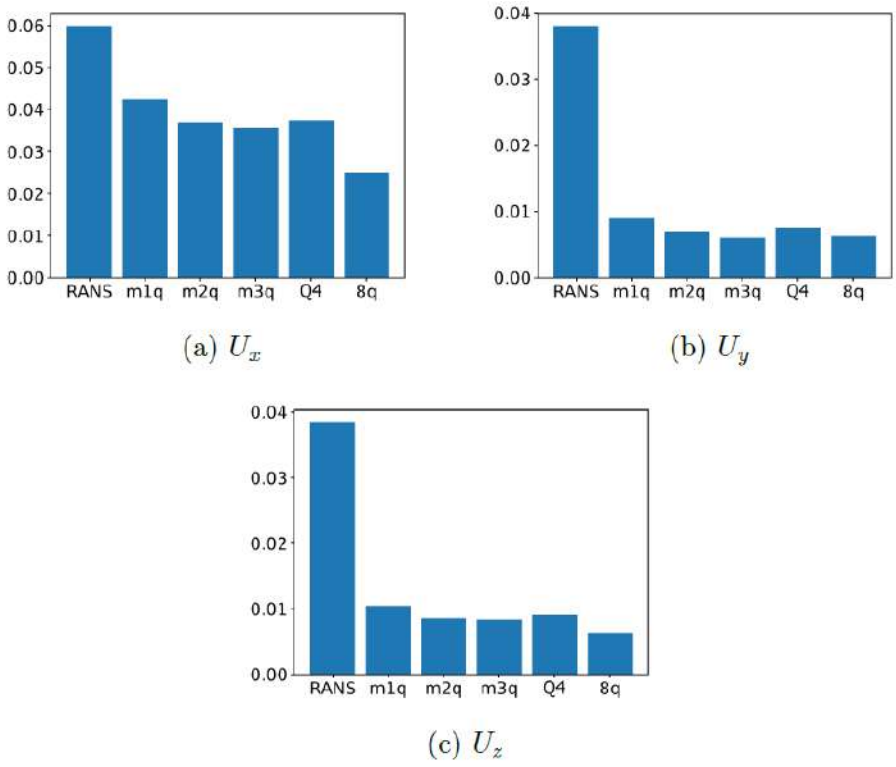
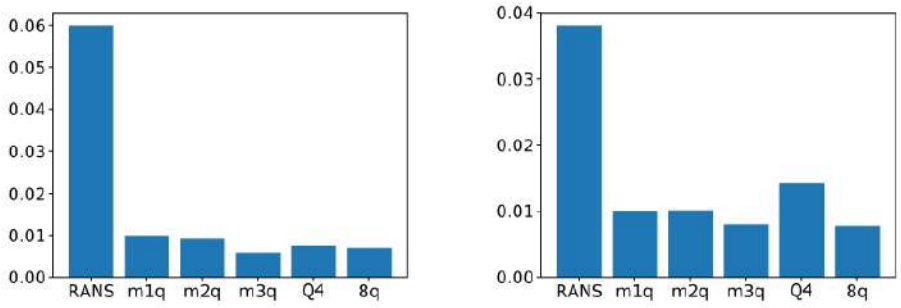
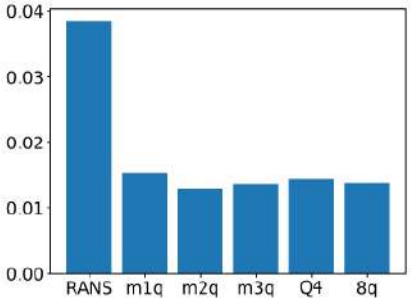


Figure 3.10: Global error of the prediction of $\mathbf{R}$ for $Re = 3200$, (RANGEL, 2019).



(a) U_x (b) U_y



(c) U_z

Figure 3.11: Global error of the prediction of $\mathbf{t}$ for $Re = 3200$, (RANGEL, 2019).

Chapter 4

Methodology

4.1 Square Duct Flow

The square duct flow is one of the most studied flow patterns in turbulence modeling. The reason for this choice is due to the existence of a secondary flow located in the cross-plane. For this flow pattern, two equation RANS models are known to have difficulty capturing this recirculation due to be based on the linear eddy viscosity hypothesis, as stated by MUCK *et al.* (1985). In the square duct, the Reynolds number is calculated by the hydraulic diameter D and the bulk velocity. Fig. 4.1, shows the schematic of the square duct flow. It is possible to observe a symmetry pattern on each quarter of the geometry, as shown in Fig. 4.1(b). Due to this pattern, it is only necessary to model one quarter of the duct by applying symmetry boundary conditions. The boundary conditions applied to the model are the non-slip condition on solid faces, symmetry in the cut-off regions of the quarter duct, cyclic conditions in the inlet and outlet and constant average velocity in the direction of the main flow. The chosen RANS model was the $\kappa - \varepsilon$, implemented in the open source program OpenFOAM-v7 (OF). A finite volume OF solver, based on the Semi-Implicit Method for Pressure Linked Equations (SIMPLE) algorithm for pressure and velocity decoupling and used in incompressible and steady flows for Newtonian fluids, called simpleFoam was used. The discretization of divergent operator from the transport equations chosen for the $\kappa-\varepsilon$ simulations was the Gauss upwind. The gradient operators were discretized using the Gauss linear algorithm. For the Laplacian operators, a corrected Gauss linear was implemented. The point-to-point interpolation scheme used in the simulations was linear. For the discrete pressure equation, the Preconditioned Conjugate Gradient (PCG) solver was used, coupled with the Geometric agglomerated Algebraic MultiGrid (GAMG) preconditioner and Diagonal-based Incomplete Cholesky (DIC) smoother. In the equations for the velocity, κ and ε, the Preconditioned bi-conjugate gradient (PBiCGStab),

coupled with the Diagonal-based Incomplete LU (DILU) smoother. The tolerances used for these four variables were $1e-07$, with relaxation factor of 0.3 for the velocity equation and 0.4 for the pressure and turbulent fields.

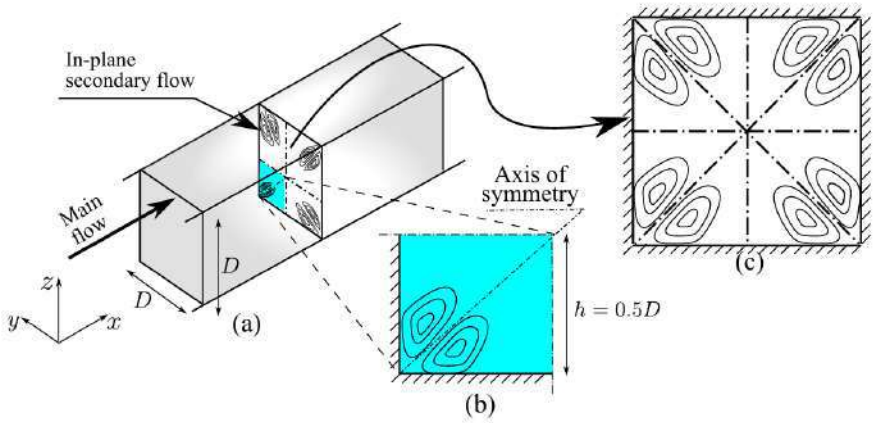


Figure 4.1: Illustration of the flow in a square duct, showing (a) the geometry configuration, (b) the region modeled in the CFD solver and (c) the illustration of the in-plane secondary flow, (MUCK *et al.*, 1985).

For all simulations, the used bulk velocity was 0.4819 $\mathrm{m\,s^{-1}}$.The mesh used in simulations has a 125×125 nodes, being more refined near the walls, as shown in Fig. 4.2, to ensure $y^+ < 0.6$. In the main direction of the flow, the mesh has only one node, resulting in a total of 15.625 centroids.

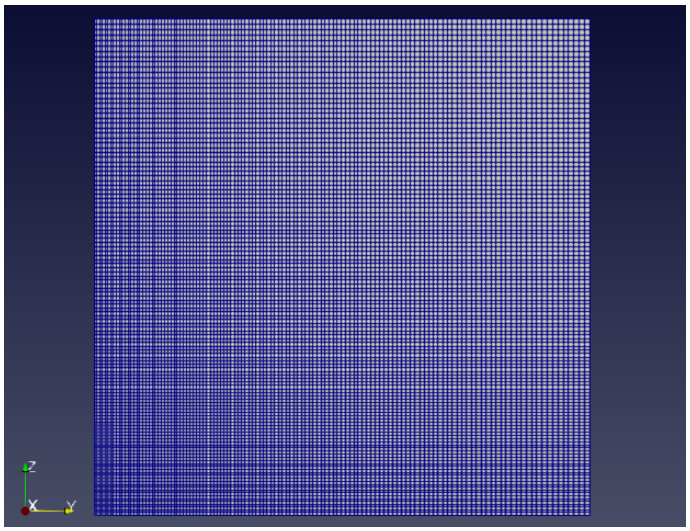


Figure 4.2: Square duct mesh.

4.2 Machine Learning Database

In order to build a ML model a high fidelity simulation database of square duct simulations is necessary. In this work, DNS data made available by PINELLI

et al. (2010) were used. For a square duct flow the authors provided DNS data for 6 different Reynolds numbers: 2200, 2400, 2600, 2900, 3200 and 3500. In order to build the input database for the ML model, for each simulation in the DNS database, a correspondent RANS simulation is implemented. In this case, as the bulk velocity and duct diameter are constants, the Reynolds numbers are differentiated by their kinematic viscosity. Table 4.1, shows the kinematic viscosity used to estimate the Reynolds number in each DNS and RANS simulation.

Table 4.1: Square duct simulations viscosity, (PINELLI *et al.*, 2010).

Re	$\nu\,[\mathrm{m^2 s^{-1}}] \times 10^{-4}$
2200	2.18580
2400	2.00816
2600	1.85368
2900	1.66190
3200	1.50611
3500	1.37700

The correspondent RANS simulations will provide quantities associated with mean kinematics and turbulence and will be used as inputs for the selected ML model. A list of symmetrical tensors and vectors that correspond to the divergence operator applied to these tensors were used as input data. This set of inputs was proposed by CRUZ *et al.* (2019) and showed promising results in ML applications. As shown in Table 4.2, in total, there are eight symmetrical tensors and eight vectors that are related to the mean strain rate, $\mathbf{D}$, the non-persistence-of-straining tensor, $\mathbf{P}$, and the Reynolds stress tensor, $\mathbf{R}$, which represent a number of 72 input features that will be applied to the NN model.

The tensor $\mathbf{P}$, idealized by THOMPSON and MENDES (2005), is the non-persistence-of-straining tensor

$$\mathbf{P} = \mathbf{D} \cdot \overline{\mathbf{W}} - \overline{\mathbf{W}} \cdot \mathbf{D}, \tag{4.1}$$

where $\overline{\mathbf{W}}$ is the relative-rate-of-rotation tensor, defined as the difference between the vorticity tensor, $\mathbf{W}$, and the tensor that gives the rotation of the eigenvectors of $\mathbf{D}$, $\mathbf{\Omega^D}$,

$$\overline{\mathbf{W}} = \mathbf{W} - \mathbf{\Omega^D}. \tag{4.2}$$

Both $\mathbf{D}$ and $\overline{\mathbf{W}}$ are objective tensors, hence, $\mathbf{P}$ is also objective. The non-persistence-of-straining tensor is a deformation local measurement. If $\mathbf{P} = 0$, the flow is considered extensional and has total persistence of straining, being in a

configuration of maximum deformation. This can be observed if $\mathbf{P}$ is written in the basis of the eigenvectors of $\mathbf{D}$

$$[\mathbf{P}]^{[\mathbf{D}]} = \begin{bmatrix} 0 & \left(\lambda_2^{(D)} - \lambda_1^{(D)}\right)\overline{w_3} & \left(\lambda_1^{(D)} - \lambda_3^{(D)}\right)\overline{w_2} \\ \left(\lambda_2^{(D)} - \lambda_1^{(D)}\right)\overline{w_3} & 0 & \left(\lambda_3^{(D)} - \lambda_2^{(D)}\right)\overline{w_1} \\ \left(\lambda_1^{(D)} - \lambda_3^{(D)}\right)\overline{w_2} & \left(\lambda_3^{(D)} - \lambda_2^{(D)}\right)\overline{w_1} & 0 \end{bmatrix},$$

where $\mathbf{P}$ can only be null if all eigenvalues of $\mathbf{D}$ are equal, which means that any direction is a direction of maximum stretching, or if the relative vorticity, $\overline{w_i}$, is null, indicating that the filament rotates along with the eigenvectors of $\mathbf{D}$ and it always stays at maximum stretching direction.

Table 4.2: ML model inputs.

Tensors ($\mathbf{T}_i$)	Vectors ($\nabla \cdot \mathbf{T}_i$)
$\mathbf{D}$	$\nabla \cdot \mathbf{D}$
$\mathbf{P}$	$\nabla \cdot \mathbf{P}$
$\mathbf{D}^2$	$\nabla \cdot \mathbf{D}^2$
$\mathbf{P}^2$	$\nabla \cdot \mathbf{P}^2$
$\mathbf{D} \cdot \mathbf{P} + \mathbf{P} \cdot \mathbf{D}$	$\nabla \cdot (\mathbf{D} \cdot \mathbf{P} + \mathbf{P} \cdot \mathbf{D})$
$\mathbf{D}^2 \cdot \mathbf{P} + \mathbf{P} \cdot \mathbf{D}^2$	$\nabla \cdot (\mathbf{D}^2 \cdot \mathbf{P} + \mathbf{P} \cdot \mathbf{D}^2)$
$\mathbf{P}^2 \cdot \mathbf{D} + \mathbf{D} \cdot \mathbf{P}^2$	$\nabla \cdot (\mathbf{P}^2 \cdot \mathbf{D} + \mathbf{D} \cdot \mathbf{P}^2)$
$\mathbf{R}$	$\nabla \cdot \mathbf{R}$

The ML model inputs, then, are the 72 components of tensors and vectors of Table 4.2, produced from the RANS simulations for the Reynolds numbers from Table 4.1. In order to build the outputs of the ML models, two different set of parameters will be used. In the first ML model, the components of $\mathbf{R}$, from the DNS database will be used. In the second model, the components of $\mathbf{t}$, the modified Reynolds force vector will be used, calculated from the velocity fields made available from de DNS database, using Eq. (3.6).

Each RANS simulation was performed in a mesh with 15.625 centroids. So for the entire set of six simulations a total of 93.750 points will be used in the ML models. In this work, the NN model simulations were divided into: four simulations for training the NN, one simulation to validate the NN training and one simulation to test and evaluate the model. For the RF models, five simulations were used in training and one simulation to test and evaluate the RF model.

Therefore, for the NN models, the input matrix for training, X_{train}, has a dimension of 62.500×72. The output matrix Y_{train} has the dimension of 62.500×6 in the NN model built to predict $\mathbf{R}$ and of 62.500×3 in the NN built to predict $\mathbf{t}$. The input matrix for validation and test, X_{val} and X_{test}, respectively, have the

same dimension of 15.625×72. The output matrices for validation and test, Y_{val} and Y_{test}, respectively, have the same dimension of 15.625×6 in the NN model built to predict $\mathbf{R}$ and the dimension of 15.625×3 in the NN built to predict $\mathbf{t}$.

For the RF models, the validation data is not necessary, hence, the input matrix for training, X_{train}, has the dimension of 78.125×72. The output matrix Y_{train} has the dimension of 78.125×6 in the model built to predict $\mathbf{R}$ and the dimension of 78.125×3 in the built to predict $\mathbf{t}$. The input matrix for test, X_{test}, has the dimension of 15.625×72. The output matrix for test, Y_{test}, has the same dimension of 15.625×6 in the model built to predict $\mathbf{R}$ and the dimension of 15.625×3 in the model built to predict $\mathbf{t}$.

4.2.1 Invariant Database

As recent work presented advantages when using an invariant database in ML applications in turbulence modeling, the database used in this work also aims to be Euclidean invariant. As stated by GURTIN (2010), in order to a field to be invariant, it must obey the relations shown in Eqs. (2.37) and (2.40). Applying these relations to all components in Table 4.2, it is observed that not all fields have Euclidean invariance.

In order to build an Euclidean invariant database, all tensor and vector fields are written in the basis of the eigenvectors of the mean strain tensor, $\mathbf{D}$, which was shown in Eq. (2.47), to have Euclidean invariance. To perform this operation, the matrix of frame-rotation, $\mathbf{Q}$ must be written as a function of the components of the eigenvectors of $\mathbf{D}$

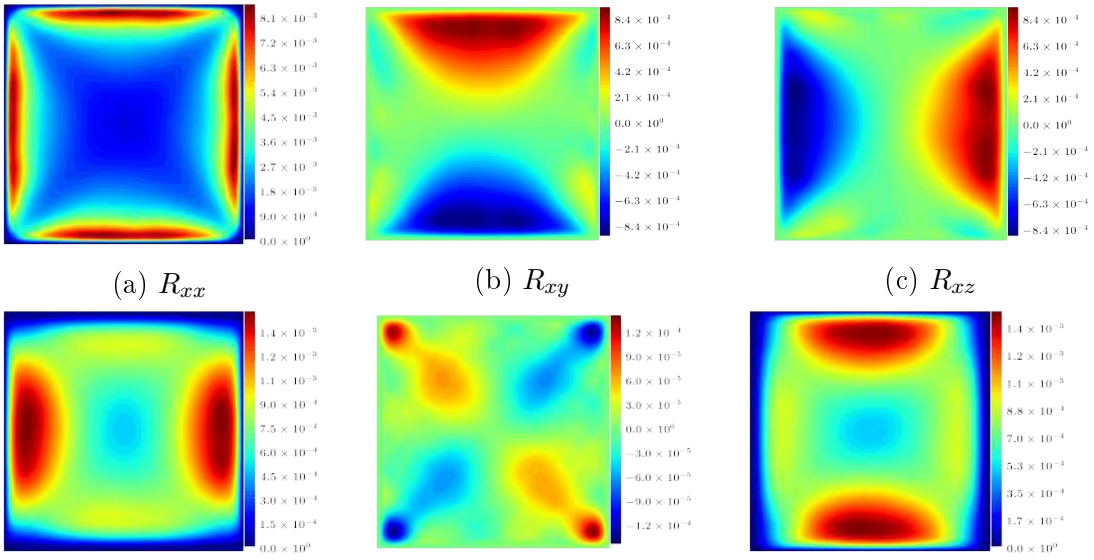
$$\mathbf{Q}_D = \begin{bmatrix} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} \end{bmatrix},$$

where $\mathbf{Q}_D$ columns, $\mathbf{v_1}$, $\mathbf{v_2}$ and $\mathbf{v_3}$ are the three linearly independent eigenvectors of $\mathbf{D}$. The frame-rotation matrix, $\mathbf{Q}_D$, can also be written as

$$\mathbf{Q}_D = \begin{bmatrix} v_{(1,1)} & v_{(2,1)} & v_{(3,1)} \\ v_{(1,2)} & v_{(2,2)} & v_{(3,2)} \\ v_{(1,3)} & v_{(2,3)} & v_{(3,3)} \end{bmatrix} \quad \text{and} \quad \mathbf{Q}_D^\top = \begin{bmatrix} v_{(1,1)} & v_{(1,2)} & v_{(1,3)} \\ v_{(2,1)} & v_{(2,2)} & v_{(2,3)} \\ v_{(3,1)} & v_{(3,2)} & v_{(3,3)} \end{bmatrix} \tag{4.3}$$

Applying Eq. (4.3) to Eqs. (2.37) and (2.40), The invariant input database becomes the fields shown in Table 4.3. The outputs of the NN models should also have Euclidean invariance. Therefore, $\mathbf{R}$ and $\mathbf{t}$ can be written as

$$\mathbf{R}^D = \mathbf{Q}_D\left(\mathbf{R}\right)\mathbf{Q}_D^\top \quad \text{and} \quad \mathbf{t}^D = \mathbf{Q}_D\left(\mathbf{t}\right). \tag{4.4}$$



(a) R_{xx} (b) R_{xy} (c) R_{xz}

(d) R_{yy} (e) R_{yz} (f) R_{zz}

Figure 4.3: DNS Reynolds stress tensor components for $Re = 2900$, (PINELLI *et al.*, 2010).

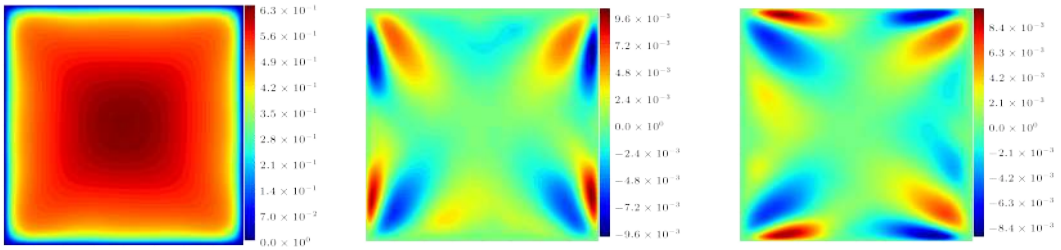
Table 4.3: ML model invariant inputs.

Tensors $\left(\mathbf{Q}_D \mathbf{T}_i \mathbf{Q}_D^\top\right)$	Vectors $\left(\mathbf{Q}_D \nabla \cdot \mathbf{T}_i\right)$
$\mathbf{Q}_D\left(\mathbf{D}\right)\mathbf{Q}_D^\top$	$\mathbf{Q}_D\left(\nabla \cdot \mathbf{D}\right)$
$\mathbf{Q}_D\left(\mathbf{P}\right)\mathbf{Q}_D^\top$	$\mathbf{Q}_D\left(\nabla \cdot \mathbf{P}\right)$
$\mathbf{Q}_D\left(\mathbf{D}^2\right)\mathbf{Q}_D^\top$	$\mathbf{Q}_D\left(\nabla \cdot \mathbf{D}^2\right)$
$\mathbf{Q}_D\left(\mathbf{P}^2\right)\mathbf{Q}_D^\top$	$\mathbf{Q}_D\left(\nabla \cdot \mathbf{P}^2\right)$
$\mathbf{Q}_D\left(\mathbf{D}\cdot\mathbf{P}+\mathbf{P}\cdot\mathbf{D}\right)\mathbf{Q}_D^\top$	$\mathbf{Q}_D\left(\nabla \cdot \left(\mathbf{D}\cdot\mathbf{P}+\mathbf{P}\cdot\mathbf{D}\right)\right)$
$\mathbf{Q}_D\left(\mathbf{D}^2\cdot\mathbf{P}+\mathbf{P}\cdot\mathbf{D}^2\right)\mathbf{Q}_D^\top$	$\mathbf{Q}_D\left(\nabla \cdot \left(\mathbf{D}^2\cdot\mathbf{P}+\mathbf{P}\cdot\mathbf{D}^2\right)\right)$
$\mathbf{Q}_D\left(\mathbf{P}^2\cdot\mathbf{D}+\mathbf{D}\cdot\mathbf{P}^2\right)\mathbf{Q}_D^\top$	$\mathbf{Q}_D\left(\nabla \cdot \left(\mathbf{P}^2\cdot\mathbf{D}+\mathbf{D}\cdot\mathbf{P}^2\right)\right)$
$\mathbf{Q}_D\left(\mathbf{R}\right)\mathbf{Q}_D^\top$	$\mathbf{Q}_D\left(\nabla \cdot \mathbf{R}\right)$

4.3 Database Treatment

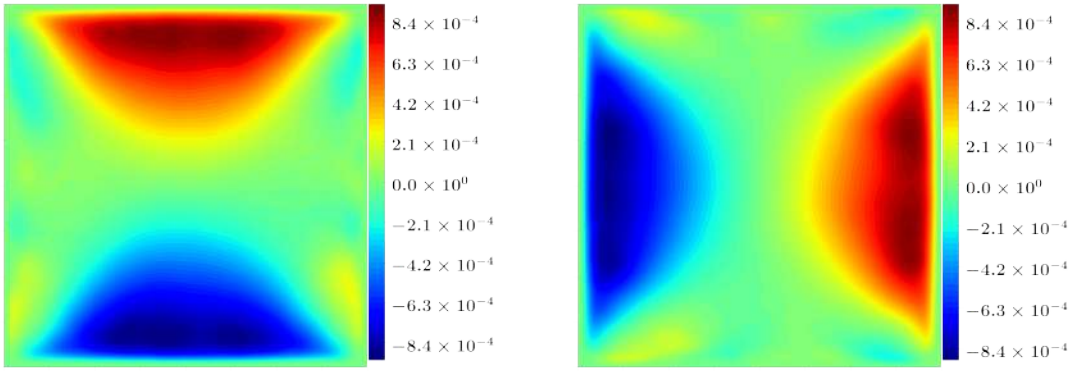
The output fields are provided by DNS simulations performed by PINELLI *et al.* (2010). For each Reynolds number, nine components are made available by the authors, the six components of the symmetrical $\mathbf{R}$, R_{xx}, R_{xy}, R_{xz}, R_{yy}, R_{yz} and R_{zz} and the three components of the mean velocity field, U_x, U_y and U_z. The data contain information for the whole square duct. Fig. 4.3 and 4.4 shows these DNS fields.

To show that the DNS simulation does not present the symmetry relations expected in a square duct pattern, Fig. 4.5, shows the shear components of $\mathbf{R}$, R_{xy}, R_{xz}, for $Re = 2900$. These fields should present symmetry along the vertical and horizontal midsections and also present symmetry between themselves as the shear



(a) U_x (b) U_y (c) U_z

Figure 4.4: DNS mean velocity components for $Re = 2900$, (PINELLI *et al.*, 2010).



(a) R_{xy} (b) R_{xz}

Figure 4.5: DNS Reynolds shear stress tensor components for $Re = 2900$, (PINELLI *et al.*, 2010).

components are differentiated only by a simple rotation.

As stated by MUCK *et al.* (1985), the domain of the square duct presents, when converged, a mean velocity field with symmetric aspects in relation with the vertical and horizontal midsections as well as in relation with the diagonals of the domain. Then, the square duct domain can be divided in four quadrants, as shown in Fig. 4.6, and RANS simulations can be performed only in one quadrant, in this case, the quadrant $3q$.

Another example showing that DNS does not present the expected symmetry, Fig. 4.7, reveals the difference in the main direction of the mean velocity, U_x, for $Re = 2900$, calculated by subtracting, from the quadrant $3q$, the three quadrants, $1q$, $2q$ and $4q$. This shows that the results obtained by the DNS simulation does not present the symmetry in relation to the vertical and horizontal axis, which goes against the logic of flow pattern.

Assuming that the DNS database does not represent the whole symmetry pattern expected in a square duct flow, the use of different quadrants in the NN model, would yield different predictions, influencing the results obtained and the model evaluation. Based on the work done by RANGEL (2019), each quadrant and combinations between quadrants will be analyzed by using them as the outputs of the NN in this

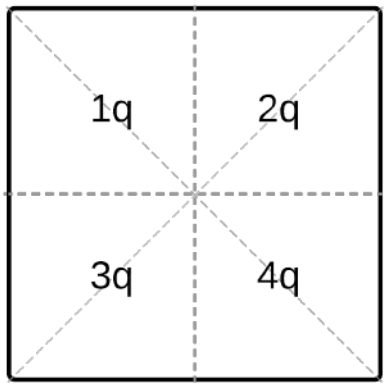
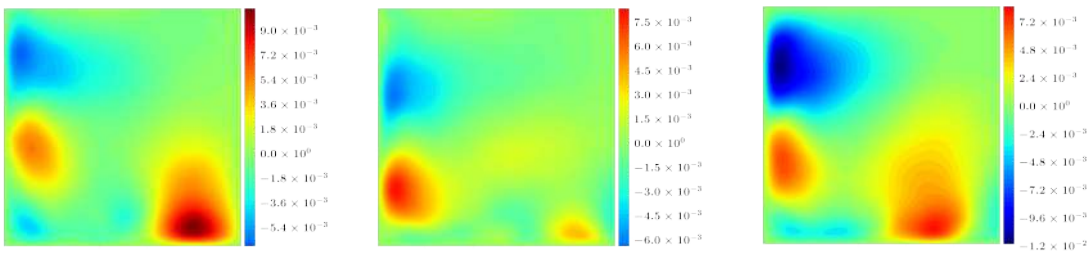


Figure 4.6: Square duct quadrants.



(a) $3q - 1q$ (b) $3q - 2q$ (c) $3q - 4q$

Figure 4.7: Discrepancy in DNS mean velocity in the main direction, U_x, for $Re = 2900$.

work.

4.3.1 Improving DNS Convergence

The DNS database used in this work was converged in a simulation time τ. As stated by ANDRADE *et al.* (2018), longer simulation times yields better converged fields. However, the available DNS already presents prohibitive simulation times and extending this times is infeasible.

Based on the methodology applied by RANGEL (2019), when splitting the square duct domain in four quadrants, as shown in Fig. 4.1, the fields of each quadrant were combined, aiming to emulate DNS results for longer simulation times. So, when combining n fields converged for a simulation time τ, it yields an emulated time of $n\tau$. This operation is done by adding n fields and dividing the resultant field by n.

This means that combining the quadrants in groups of two, generate a combination of six new DNS field sets with an emulated simulation time of 2τ. Combining the quadrants in groups of three, generates a combination of four new DNS field sets with an emulated simulation time of 3τ. Similarly, combining all quadrant in a group, yields in a new DNS field set with emulated simulation time of 4τ and symmetry along the vertical and horizontal midsections. In order to obtain symmetry in relation with the diagonals of the domain and between fields that are similar but rotated, as shown in Fig. 4.5, in the set of fields where all quadrants are combined the diagonal symmetry was applied as well as similar rotated fields, resulting in a full symmetrical domain, with a combined of eight times the emulated simulation time, 8τ. All this quadrant combination sets were divided in five groups: $Q1$ for sets of only one quadrant, $Q2$ for combinations with two-by-two quadrants, $Q3$ for combinations with three-by-three quadrants, $Q4$ for combinations with four-by-four quadrants and $Q8$ for the full symmetrical domain of the square duct fields. These groups are listed in Table 4.4.

The DNS fields of $Q8$ represents the maximum symmetry of the domain, resulting in the most accurate DNS data available for the square duct. Fig. 4.8, represent the evolution of the R_{yz} component for $Re = 2900$, which is as component that presents symmetry along the diagonal line of the domain.

So, to build the full developed database used in this work, data from $\mathbf{R}$ and $\mathbf{t}$ were extracted from each quadrant set, resulting in a set of 32 NN models and 32 RF models evaluated in this work.

Table 4.4: Quadrant combination sets.

Combination Groups	Quadrant Sets
$Q1$ (Combinations 1 by 1)	$1q$ $2q$ $3q$ $4q$
$Q2$ (Combinations 2 by 2)	$12q$ $13q$ $14q$ $23q$ $24q$ $34q$
$Q3$ (Combinations 3 by 3)	$123q$ $124q$ $134q$ $234q$
$Q4$ (Combinations 4 by 4)	$1234q$
$Q8$ (Full symmetry set)	$8q$

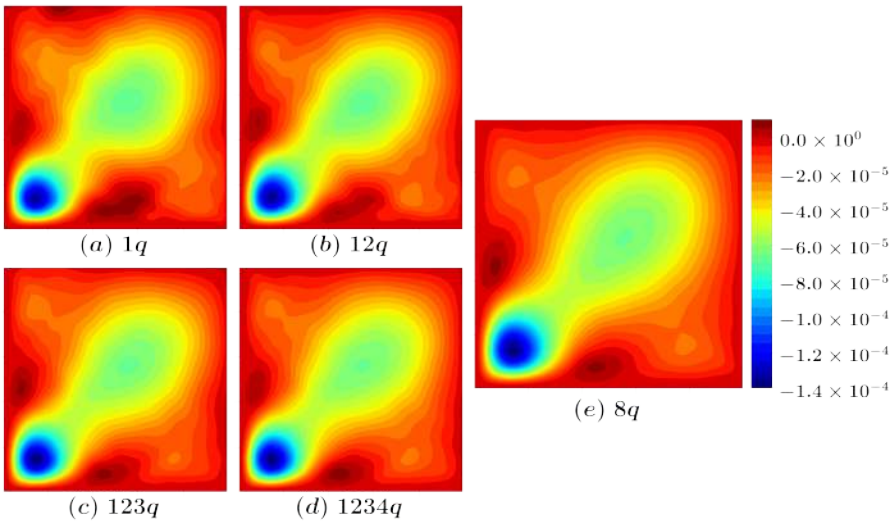


Figure 4.8: Evolution of DNS component R_{yz} for different quadrant sets

4.4 Global Error

In order to evaluate the propagation of the DNS quantities into the velocity field, the mean momentum balance equations, coupled with the continuity equation for mean pressure and velocity fields, are solved, injecting the respective quantity of interest, $\mathbf{R}$ or $\mathbf{t}$, from the DNS database associated with each case. In this work the RANS equation with a source term, $\nabla \cdot \mathbf{R}_\theta$, is numerically solved in OF, where, when solving for $\mathbf{R}$, the divergence operator is applied and when solving for $\mathbf{t}$, the equation is written as

$$(\langle \mathbf{u} \rangle \cdot \nabla)\langle \mathbf{u} \rangle + \mathbf{t}_\theta = \nu \nabla^2 \langle \mathrm{u} \rangle \tag{4.5}$$

After running OF for all sets of $\mathbf{R}$ and $\mathbf{t}$, the local metric of error implemented by CRUZ *et al.* (2019) is used, consisting in a relative $L_2 - norm$ of the error corresponding to the propagation error of the velocity field and can be calculated by

$$E_i = \frac{\sqrt{\left(\langle \mathbf{u} \rangle_{\theta i} - \langle \mathbf{u} \rangle_{R_{\theta i}}\right)^2}}{U_{bulk}}, \quad i = x, y, z, \tag{4.6}$$

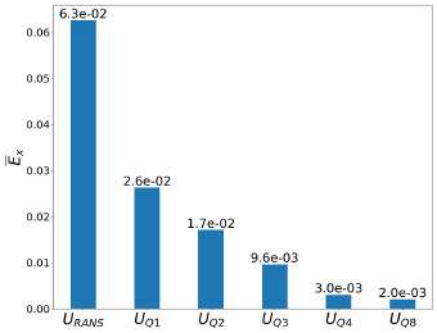
where i represents each component of the velocity field, U_{bulk} is constant for all cases and $\langle \mathbf{u} \rangle_{\theta i}$ and $\langle \mathbf{u} \rangle_{R_{\theta i}}$ are the DNS mean velocity field and the reconstructed DNS mean velocity field, respectively. In order to calculate a global error to attribute a singular value for the error in the cross section area domain in each field, the local error in Eq. (4.6) is integrated across the area of the domain

$$\bar{E}_i = \frac{1}{A} \int_A E_i dA \quad i = x, y, z. \tag{4.7}$$

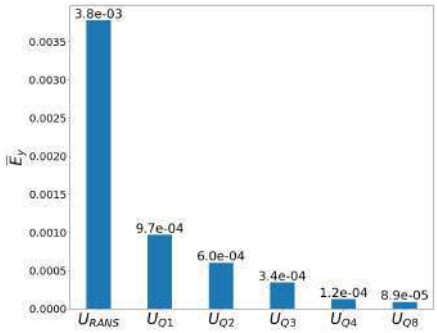
The global error is then calculated for all DNS data obtained for each quadrant set, for both $\mathbf{R}$ and $\mathbf{t}$. In Figs. 4.9 and 4.10, the global error value for each combination group is the arithmetic mean for the global error for each quadrant set, for example for the global error of the combination group $Q3$

$$\bar{E}_i\left(U_{Q3}\right) = \frac{\bar{E}_i\left(U_{123q}\right) + \bar{E}_i\left(U_{124q}\right) + \bar{E}_i\left(U_{134q}\right) + \bar{E}_i\left(U_{234q}\right)}{4} \quad i = x, y, z. \tag{4.8}$$

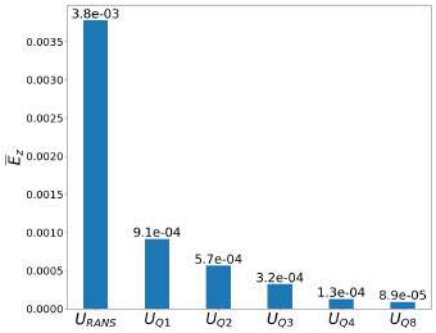
Figs. 4.9 and 4.10 show that, as expected, as the emulated simulation time increases, the global error associated with the propagation of the DNS quantities decreases, with the global error associated with $\mathbf{t}$, being smaller than the error associated with $\mathbf{R}$, which is in agreement with the results obtained by CRUZ *et al.* (2019). This shows an advantage of using data from emulated DNS simulation time



(a) U_X

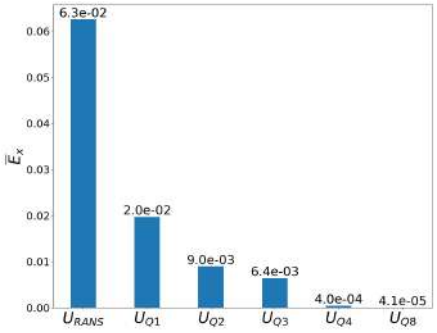


(b) U_Y

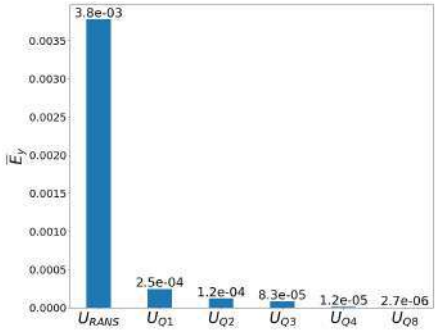


(c) U_Z

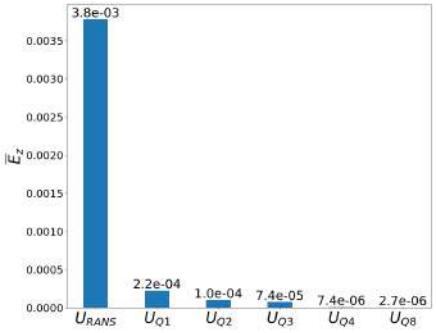
Figure 4.9: Global error of the velocity field for DNS **R** injected in OF.



(a) U_X



(b) U_Y



(c) U_Z

Figure 4.10: Global error of the velocity field for DNS **t** injected in OF.

as outputs of as ML model, where is expected that better converged data yields better predictions.

4.5 Post-Processing

Based on the evolution of the global error when applying symmetry to the square duct, as shown in Figs. 4.9 and 4.10, it seems to be an advantage to impose symmetry in every field from the ML models, $\mathbf{R}$ or $\mathbf{t}$. For the predictions of the ML models, the same conditions of $Q8$ were imposed: for fields that have symmetry along the diagonal of the domain, as shown in Fig. 4.11, such as R_{xx}, R_{yz} and t_x, it was imposed symmetry in relation to the axis of symmetry, from the vertex of the walls to the center of the flow. For other fields, a rotation has been applied and an average has been taken between the components such as,

$$R_{yy} = \frac{R_{yy} + R_{zz}^{rotated}}{2}, \tag{4.9}$$

which can be visualized in Fig. 4.12.

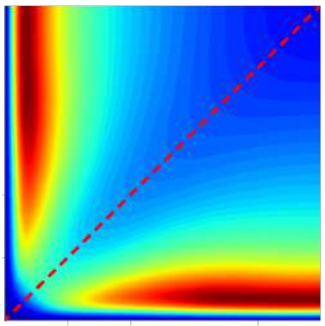
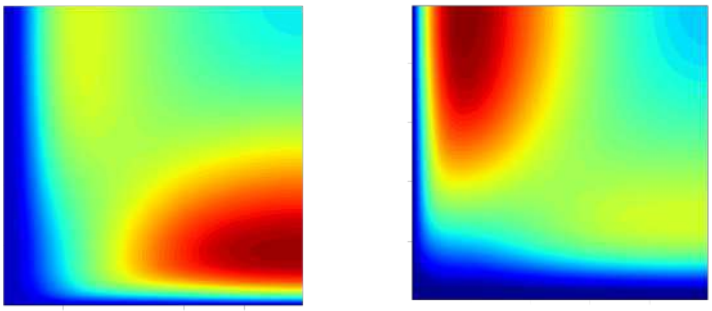


Figure 4.11: Diagonal symmetry in R_{xx}.



(a) R_{yy} (b) R_{zz}

Figure 4.12: Rotation associated with R_{yy} and R_{zz} components

4.6 Training

All codes were developed in the Python programming language. After defining the inputs, Table 4.3, and outputs, Eq. (4.4), used for all 64 ML models, a common pre-processing method in ML application is applied. As different components in the inputs and output matrices has different orders of magnitude, this can induce the NN to set higher values of weights and bias to components with higher magnitude lowering the learning capability of the NN and yielding an inaccurate ML model. In order to work around this issue, a normalization is applied to each column of the input and output matrix. A standard normalization is applied, scaling the columns in relation to their standard deviation, σ_{column} and their mean, μ_{column} , as

$$\text{normalized column } = \frac{\text{column} - \mu_{\text{column}}}{\sigma_{\text{column}}}. \tag{4.10}$$

This normalization is applied to each column of the input matrices, X_{train} X_{val} and X_{test}, and output matrices, Y_{train} Y_{val} and Y_{test}. The resulting matrix, here called Y_{pred}, the output result of the NN, is also normalized and needs to be re-scaled using the inverse process

$$\text{column } = \sigma_{\text{column}} \text{ (normalized column)} + \mu_{\text{column}} . \tag{4.11}$$

For this matter, the open source python library, SCIKIT-LEARN, was used through a function called Standard Scale, which makes the standard normalization for all matrices.

The same NN architecture, based on the results obtained by CRUZ *et al.* (2019) , was implemented for all 32 NN models built, with the objective to evaluate the impact of improving DNS convergence under the same conditions for every NN model. To build the NN model, the open source library KERAS, based on Google TensorFlow with an easier implementation, was used. The chosen architecture is composed of two hidden layers with 100 neuron in each. The activation function for the first two layers was the hyperbolic tangent, and for the last layer a linear one, as usual in regression ML applications. The employed optimization function was the ADAM algorithm with the mean squared error, MSE, for cost function, with a starting learning rate of 1×10^{-3}, with a reduction rate of 0.6 after 5 epochs. Weights and bias were constrained using a maximum norm of 2.0. No dropout was used in this architecture as it showed no improvement for this number of neurons. To stop the training process, the early-stopping method was used, set to interrupt the training if the validation error does not decrease for 20 consecutive epochs. Table 4.5, summarizes the hyper-parameter used in the NN models.

As wells as for NN models, the RF developed in this work has the same architec-

Table 4.5: Summary of the NN hyper-parameters

Number of layers	2
Neurons per layer	100
Activation function	*tanh*
Batch-size	32
Weight and Bias constraint	2.0
Optimization function	ADAM
Starting learning rate, η	$10-3$
η decay rate	0.6
Number of epochs to reduce η	5
Number of epochs to stop the training	20

ture for all 32 models. To build the RF, the SCIKIT-LEARN library was used. The chosen architecture is composed of two hundred decision trees, with a max feature of 24 features, a max depth of 50 and the default values for minimum samples split and minimum samples leaf, of two and one, respectively. Table 4.6, summarizes the hyper-parameter used in the RF models.

Table 4.6: Summary of the RF hyper-parameters

Number of trees	200
Max features	24
Max depth	50
Min samples split	2
Min samples leaf	1

After training the ML models, they are able to predict new sets of **R** or **t**, called Y_{pred}, using the input matrices, X_{test}. To evaluate the performance of each ML model, a metric called coefficient of determination, R^2 score, was used. R^2 score indicates the percentage of the variance in the dependent variable that the independent variables can explain collectively. It measures the strength between the predicted values, Y_{pred}, and the target values, Y_{test},on a convenient percentage scale. Expressing the mean of the predicted data, $\overline{Y_{pred}}$, as

$$\overline{Y_{pred}} = \frac{1}{n}\sum_{i=1}^{n} Y_i^{pred}, \tag{4.12}$$

where Y_i^{pred} is the predicted value of the i-th sample of Y_{pred}. Expressing Y_i^{test} as the target value of the i-th sample of Y_{test}, the R^2 score can be calculated as

$$R^2\left(Y_{test}, Y_{pred}\right) = 1 - \frac{\sum_{i=1}^{n}\left(Y_i^{test} - Y_i^{pred}\right)^2}{\sum_{i=1}^{n}\left(Y_i^{test} - \overline{Y_{pred}}\right)^2}. \tag{4.13}$$

The six simulations available, corresponding to the Reynolds number in Table

4.1, will be divided in four for training, one for validation and one for testing the NN models. In order to choose which will be used as training, validation and testing, a cross-validation process was done and the division of groups can be seen on Tab, 4.7. Therefore, as $Re = 2900$ is used as testing group, all the results will be based on simulations for this Reynolds number. As NN have a random starting point for weights and bias, even with the same set of hyper-parameters, the accuracy of a NN can vary. A set of 20 NN's was built for each model and the one with highest R^2 score was used to predict the desired fields. Each of these 20 NN is called a fold, so for each model 20 folds will be made and the evaluation of the model will be expressed as the mean of the R^2 score for each fold. As well as for NN models, a set of 20 RF were built for each model and the one with the highest R^2 score was used to predict the desired fields.

Table 4.7: Division of NN groups.

NN Groups	**Reynolds number**
Training Group	2200
	2600
	3200
	3500
Validation Group	2400
Test Group	2900

The ML training process and the correction of velocity fields using $\mathbf{R}$ or $\mathbf{t}$ predicted by the ML model can be described in a step-by-step as

1. **ML model training:**
    - Select the respective DNS database;
    - Run RANS simulations for the same set of conditions;
    - Extract the inputs from Table 4.2 from all RANS simulations;
    - Apply Eqs. (2.37) and (2.40), for the respective vectors and tensors from Table 4.2, creating the input matrices;
    - Extract $\mathbf{R}$ and calculate $\mathbf{t}$ from DNS database;
    - Apply Eq. (4.4), to create the output matrices;
    - Split the available data into training and test groups, for RF and training, validation and test groups, for NN;
    - Run the Python code in order to build the desired ML model on training and validation groups, over 20 folds.

2. **Correcting the velocity fields:**

   - Use the input test group on the ML model created;
   - Compare the predicted output with the output test group to evaluate the ML model;
   - Post-process the predicted fields imposing symmetry;
   - Replace the predicted fields on OF;
   - With the predicted fields, rerun the used solver in order to obtain the mean corrected velocity;
   - Evaluate the global error, given by Eq. (4.7), associated with the propagation of the predicted fields.

This procedure was applied to all 32 NN and 32 RF models developed in this work and the results are exhibited in the next chapter.

Chapter 5

Results

5.1 Machine Learning Predicted Results

In Tables 5.1 and 5.2, it is possible to observe the R^2 score for all NN and RF models developed respectively, sixteen predicting the Reynolds stress tensor, $\mathbf{R}$, and sixteen predicting the modified Reynolds force vector, $\mathbf{t}$. All OF simulations were ran for $Re = 2900$. In order to compare the results, one set of each group will be used. These sets will be $1q$ from $Q1$, $12q$ from $Q2$, $123q$ from $Q3$, $1234q$ from $Q4$ and $8q$ from $Q8$. As the combination groups $Q4$ and $Q8$ have only one set, they will be designated in the subsequent figures as $Q4$ and $Q8$, respectively. All predicted values are compared to the base $\kappa - \varepsilon$ RANS fields and the most accurate DNS fields, with full symmetry applied, referred to as Target.

Table 5.1: R^2 Score of NN quadrant sets.

		R^2 **Score** $(mean \pm std)$	
Combination Group	**Quadrant set**	**R**	**t**
$Q1$	$1q$	0.9817 ± 0.0015	0.9576 ± 0.0003
	$2q$	0.9846 ± 0.0008	0.9333 ± 0.0021
	$3q$	0.9713 ± 0.0012	0.8893 ± 0.0024
	$4q$	0.9708 ± 0.0007	0.9125 ± 0.0005
$Q2$	$12q$	0.9937 ± 0.0003	0.9669 ± 0.0004
	$13q$	0.9928 ± 0.0004	0.9604 ± 0.0006
	$14q$	0.9852 ± 0.0005	0.9427 ± 0.0003
	$23q$	0.9879 ± 0.0004	0.9159 ± 0.0022
	$24q$	0.9896 ± 0.0004	0.9557 ± 0.0006
	$34q$	0.9897 ± 0.0003	0.9595 ± 0.0006
$Q3$	$123q$	0.9951 ± 0.0002	0.9553 ± 0.0010
	$124q$	0.9923 ± 0.0002	0.9621 ± 0.0002
	$134q$	0.9938 ± 0.0004	0.9669 ± 0.0003
	$234q$	0.9935 ± 0.0002	0.9557 ± 0.0009
$Q4$	$1234q$	0.9956 ± 0.0002	0.9656 ± 0.0005
$Q8$	$8q$	0.9969 ± 0.0002	0.9949 ± 0.0001

Table 5.2: R^2 Score of RF quadrant sets.

		R^2 **Score** $(mean \pm std)$	
Combination Group	**Quadrant set**	**R**	**t**
$Q1$	$1q$	0.98300 ± 0.00009	0.93554 ± 0.00049
	$2q$	0.98966 ± 0.00005	0.89779 ± 0.00104
	$3q$	0.96645 ± 0.00009	0.84966 ± 0.00126
	$4q$	0.96734 ± 0.00013	0.89541 ± 0.00058
$Q2$	$12q$	0.99391 ± 0.00004	0.94055 ± 0.00062
	$13q$	0.99229 ± 0.00003	0.93459 ± 0.00068
	$14q$	0.98325 ± 0.00012	0.92393 ± 0.00047
	$23q$	0.98715 ± 0.00005	0.87999 ± 0.00087
	$24q$	0.98845 ± 0.00006	0.93275 ± 0.00058
	$34q$	0.98779 ± 0.00005	0.93301 ± 0.00066
$Q3$	$123q$	0.99503 ± 0.00004	0.92691 ± 0.00082
	$124q$	0.99125 ± 0.00005	0.93943 ± 0.00057
	$134q$	0.99272 ± 0.00004	0.94473 ± 0.00064
	$234q$	0.99247 ± 0.00005	0.92773 ± 0.00072
$Q4$	$1234q$	0.99489 ± 0.00004	0.93995 ± 0.00056
$Q8$	$8q$	0.99665 ± 0.00003	0.99399 ± 0.00007

5.1.1 Reynolds Stress Tensor Prediction

It is possible to observe from Figs. 5.1 to 5.6, that all NN and RF models are capable of generalizing the R_{xx}, R_{xy} and R_{yy}components of $\mathbf{R}$ with satisfying accuracy, the prediction of NN being smoother than the RF, and how RANS results differ from them. As mentioned in section 4.5, symmetry along the diagonal and between components were applied in the predicted fields, therefore, components R_{xz} and R_{zz} are equivalent to R_{xy} and R_{yy}, respectively, and are not shown.

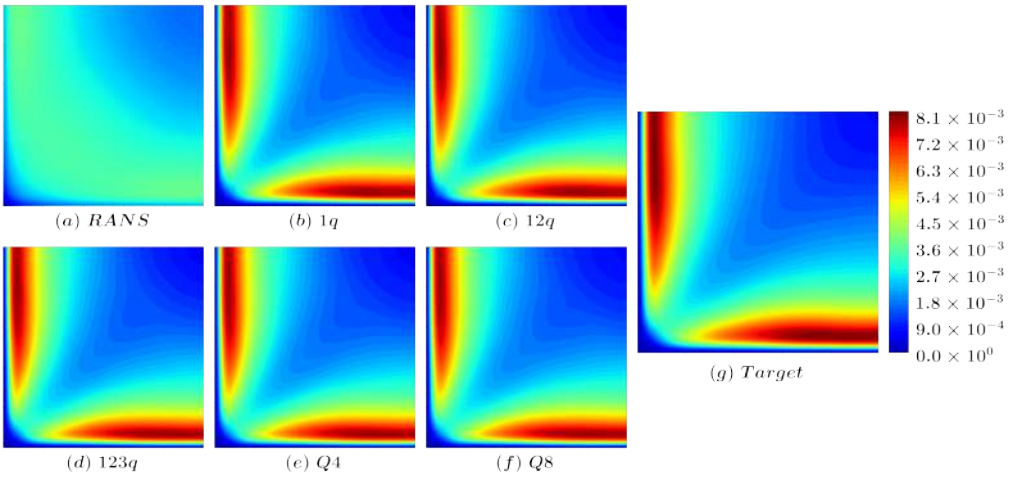


Figure 5.1: Prediction of component R_{xx} of $\mathbf{R}$ by the NN model.

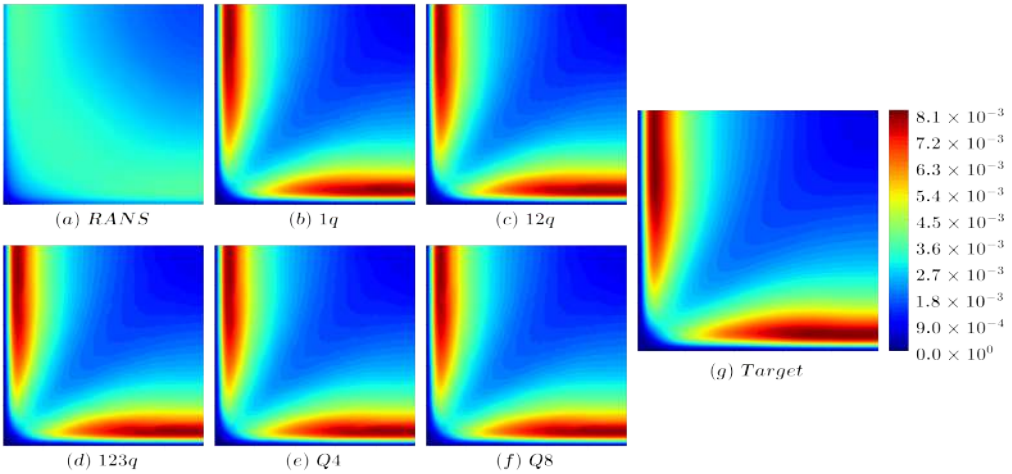


Figure 5.2: Prediction of component R_{xx} of $\mathbf{R}$ by the RF model.

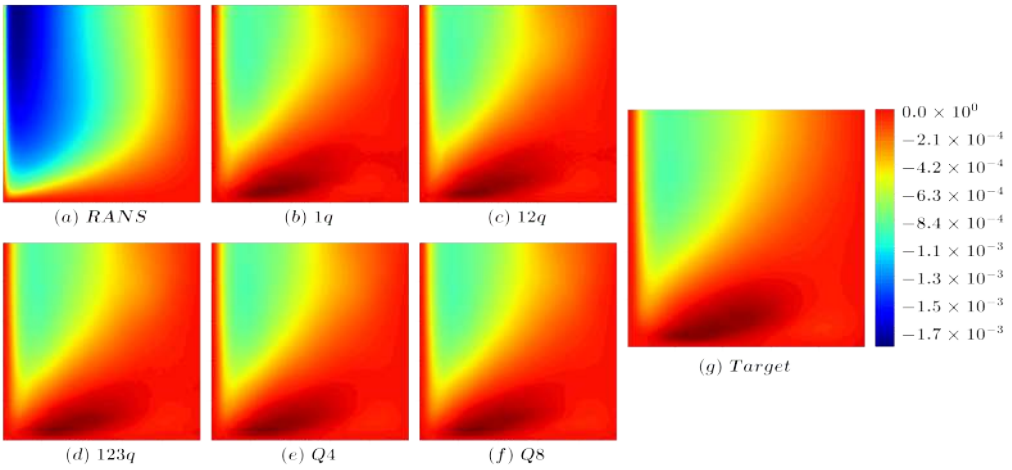


Figure 5.3: Prediction of component R_{xy} of $\mathbf{R}$ by the NN model.

R_{yz} is the most difficult component of $\mathbf{R}$ to predict. Figs. 5.7 and 5.8 show the evolution of the predictions for different quadrant sets. It is possible to observe that adding quadrant sets improve the prediction of both the NN and RF, and imposing symmetry along the diagonal for the $Q8$ group yields the best predictions for this component. It can be observed that RF predictions for this component show better results for all quadrant groups along the symmetry line, when compared to the NN predictions which are, again, smoother.

Overall, the 32 ML models were able to learn from the data available and replicate satisfactory fields for $\mathbf{R}$.

5.1.2 Reynolds Force Vector Prediction

The prediction of $\mathbf{t}$ showed that all ML models were able to generalize the available data and predict satisfactory fields. Because the post-processing symmetry was applied, the field t_z is equivalent to t_x and it is not shown.

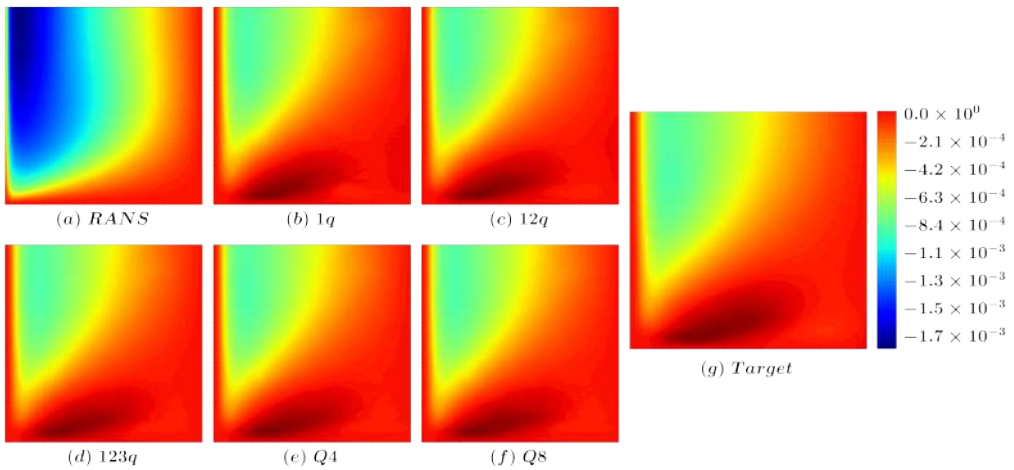


Figure 5.4: Prediction of component R_{xy} of $\mathbf{R}$ by the RF model.

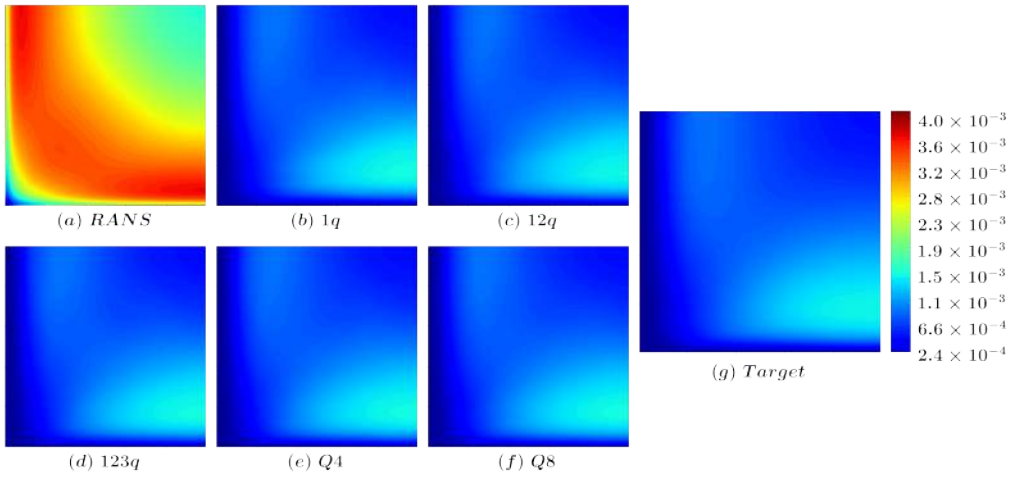


Figure 5.5: Prediction of component R_{yy} of $\mathbf{R}$ by the NN model.

It is possible to observe in Figs. 5.9 and 5.10, that the evolution of the quadrant fields yields better predictions along the symmetry diagonal of the domain for the t_x component. For all components of $\mathbf{t}$, the NN and RF predictions are accurate and present more similarity with the target field as the quadrant group is increased.

5.2 Corrected Velocity Field

Due to the imposition of symmetry in the predicted fields, the corrected mean velocity fields also present symmetry in the secondary components, U_y and U_z, making them equivalent. Therefore, only U_y is shown.

5.2.1 Reynolds Stress Tensor Correction

As shown by CRUZ *et al.* (2019), it is expected that injecting in OF the predicted field of $\mathbf{R}$, does not yield satisfactory results for the main direction of the velocity

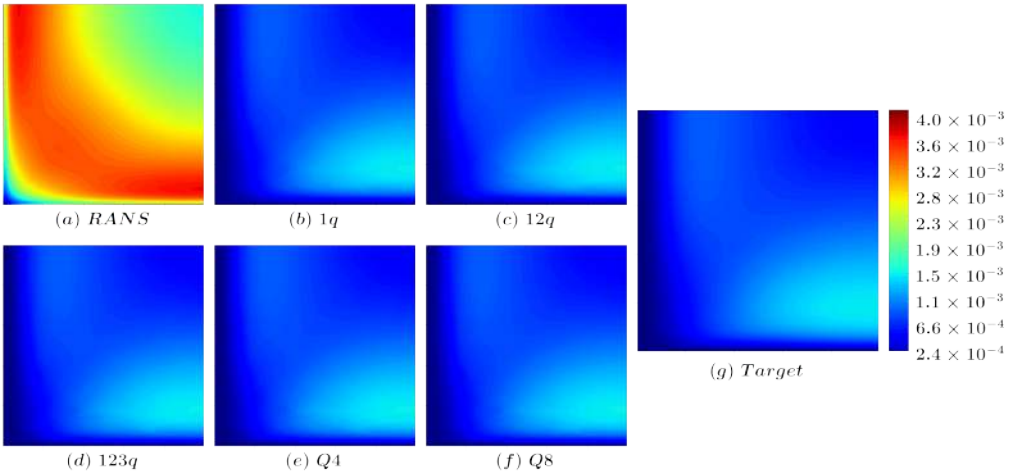


Figure 5.6: Prediction of component R_{yy} of $\mathbf{R}$ by the RF model.

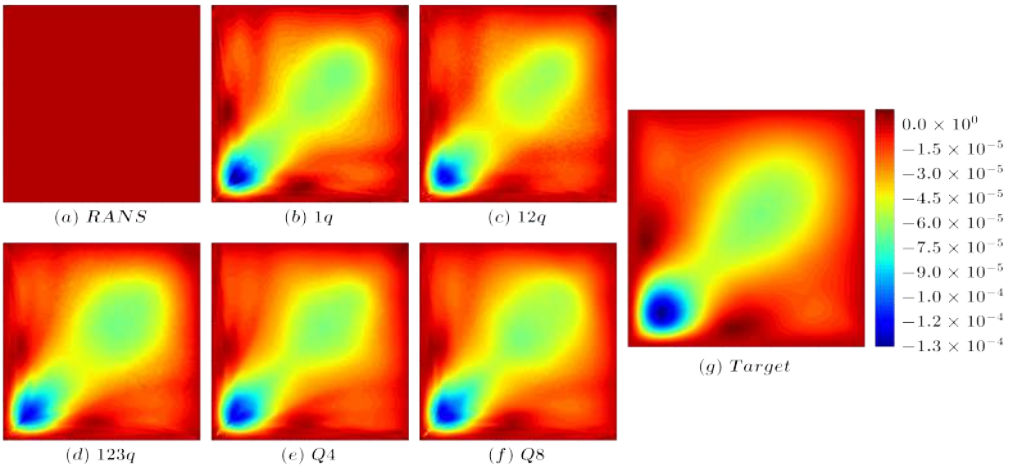


Figure 5.7: Prediction of component R_{yz} of $\mathbf{R}$ by the NN model.

field, U_x. It is possible to observe in Figs. 5.13 and 5.14, that imposing symmetry and improving the convergence of the DNS data, resulted in better predicted fields for U_x. Comparing the results obtained by the NN and the RF models, it can be observed that, overall, the NN models presented a better result near the center of the square duct, which is the upper right corner of the domain, when compared to the RF models, in a region where the maximum values are observed.

Fig. 5.15 shows the resultant corrected velocity component, U_y, obtained from predicted $\mathbf{R}$ fields from the NN. It can be observed that all NN models are capable of reproducing the recirculation expected in this flow, but with slightly different values in the center of the square duct, which in this case is the upper right corner of the domain, presenting negative values. But with the evolution of the convergence of the quadrant groups, this difference is minimized. Fig. 5.16 shows the same results for the U_y, obtained from predicted $\mathbf{R}$ fields from the RF. For all quadrants shown, the resultant fields from the RF present better recirculation when compared to the

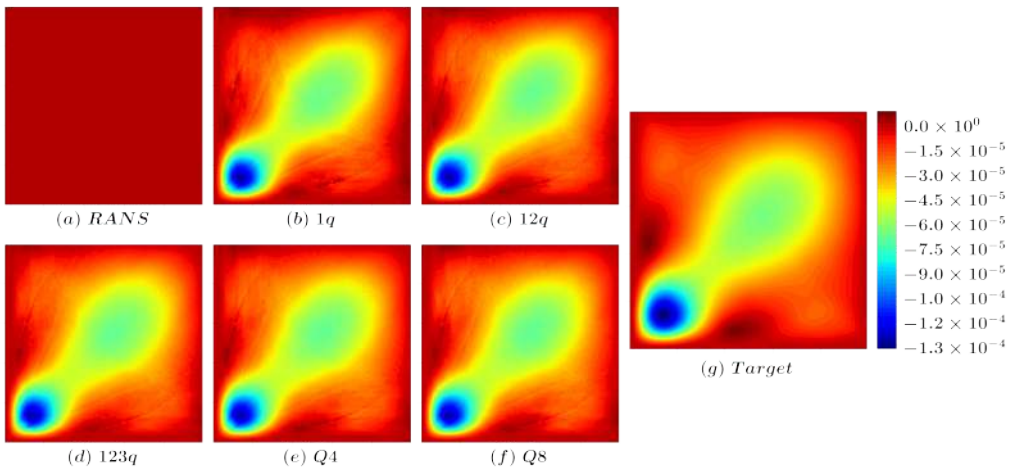


Figure 5.8: Prediction of component R_{yz} of $\mathbf{R}$ by the RF model.

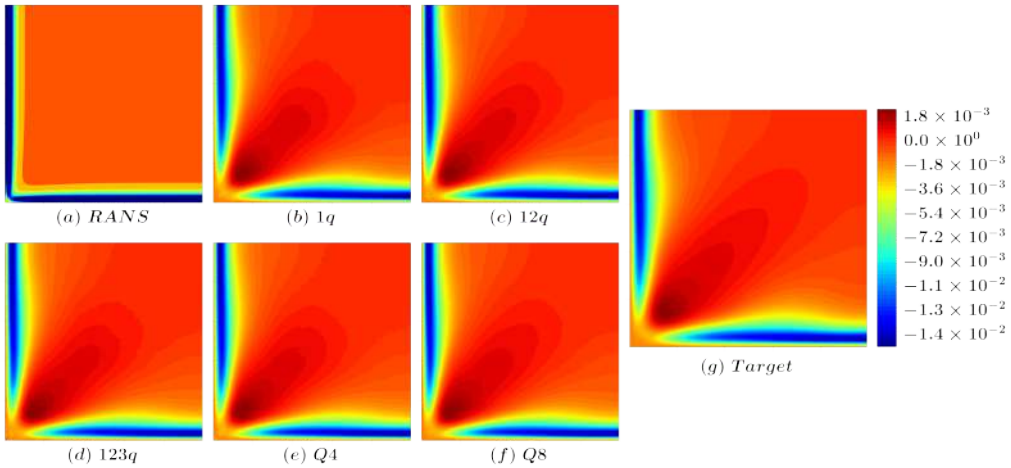


Figure 5.9: Prediction of component t_x of $\mathbf{t}$ by the NN model.

NN models, but present values higher than the target in the upper region of the domain.

5.2.2 Reynolds Force Vector Correction

As expected from the results obtained by CRUZ *et al.* (2019), the velocity fields obtained from the predicted $\mathbf{t}$ are more accurate when compared with $\mathbf{R}$, especially in the main direction. Figs. 5.17 and 5.18 show that for all models, the resulting velocity field component, U_x, is accurate and very similar with the DNS target data. It is possible to observe that in the center of the square duct, the more converged quadrant groups shows better results, being more similar to the target.

In the secondary direction, y, it can be observed that all models can accurately reproduce the recirculation expected in the flow pattern and it is not possible to distinguish the fields, proving the accuracy expected from the injection of $\mathbf{t}$ in OF.

The corrected velocity field obtained with the predicted $\mathbf{t}$ were shown to be very

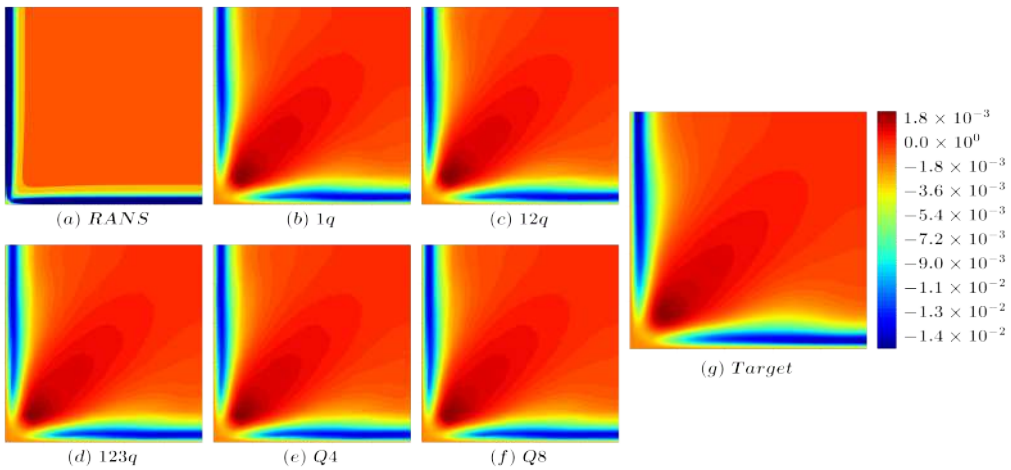


Figure 5.10: Prediction of component t_x of $\mathbf{t}$ by the RF model.

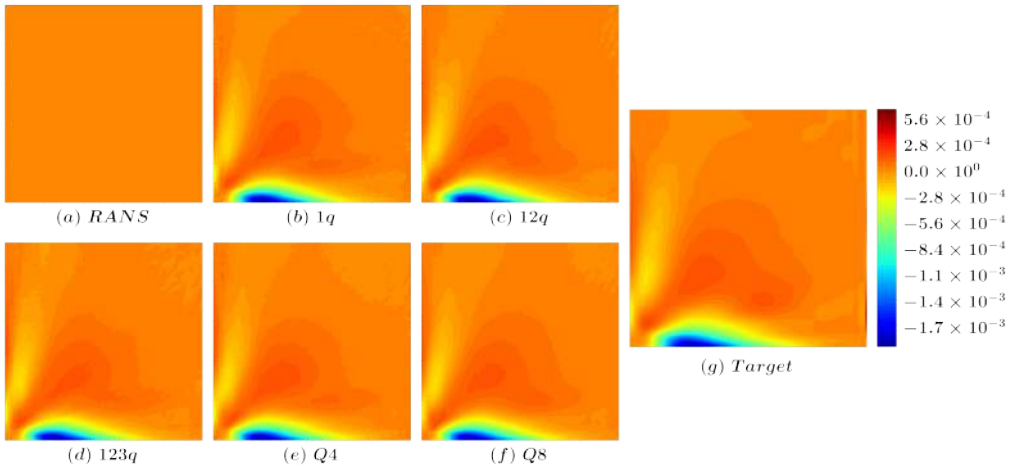


Figure 5.11: Prediction of component t_y of $\mathbf{t}$ by the NN model.

accurate, highlighting advantages of imposing the expected symmetry and the use of the Reynolds force vector high fidelity quantity from the DNS database.

5.3 Global Errors

In order to obtain one single value for the error of each corrected velocity field, the global error metric was calculated for all 64 models corrected from the predicted $\mathbf{R}$ and $\mathbf{t}$, for both NN and RF models. All the results presented were calculated for $Re = 2900$. The baseline RANS global error and the optimal $Q8$ results were inserted in all graphs to serve as comparative reference. As the imposed symmetry was applied, the global error in y and z directions are equal and therefore, only the result for y is shown. Results are analyzed for quadrant groups as follows.

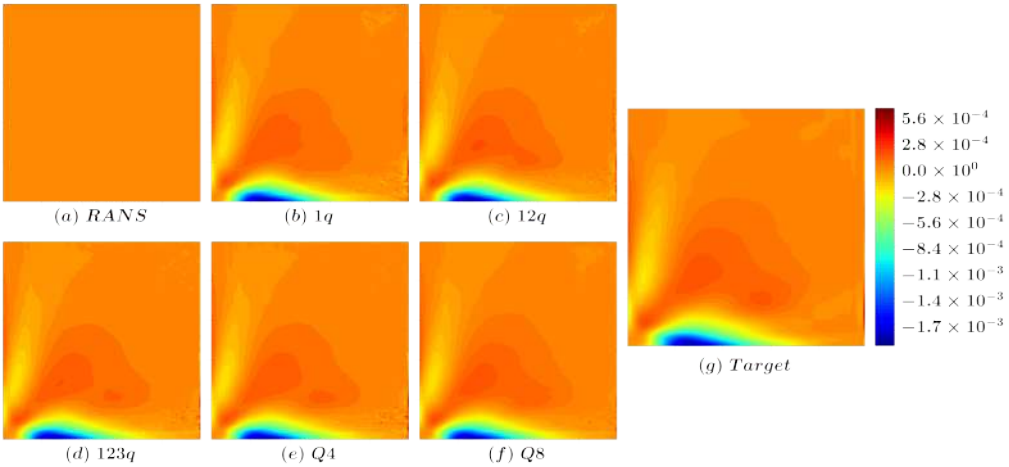


Figure 5.12: Prediction of component t_y of $\mathbf{t}$ by the RF model.

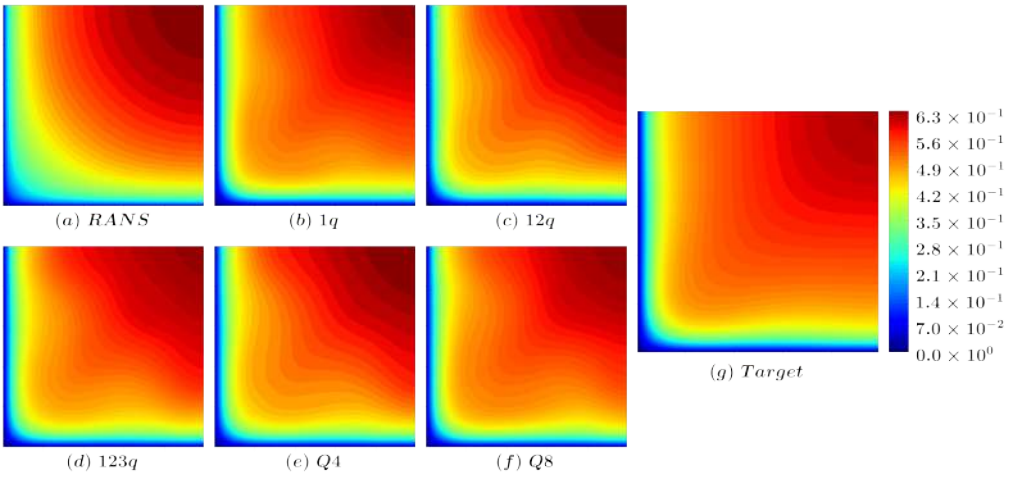


Figure 5.13: Corrected velocity field component U_x, by injecting in OF values of $\mathbf{R}$ predicted with the NN model.

5.3.1 Q1

Figs. 5.21 to 5.24 show the global error of the four quadrants in the DNS database for the predicted values based on NN and RF, $\mathbf{R}$ and $\mathbf{t}$, respectively. It is possible to observe that compared to the baseline RANS global error, applying the ML based $\mathbf{R}$ in OF, yields an accuracy improvement to all quadrants, yet the quadrants does not present similar results, showing that choosing different quadrants from the high fidelity database is important when constructing a ML based model.

In the main direction of the flow, U_x, where the baseline RANS can provide part of the physics involved in the problem, it is possible to observe, in Figs. 5.21a and 5.22a, that all $\mathbf{R}$ based velocity fields shows an improvement when compared to the baseline RANS, with the field corrected from the quadrant $3q$ presenting the worst result for the NN models and the field corrected from the quadrant $1q$ the worst for the RF models. Even with the intrinsic incapability of $\mathbf{R}$ based models to provide

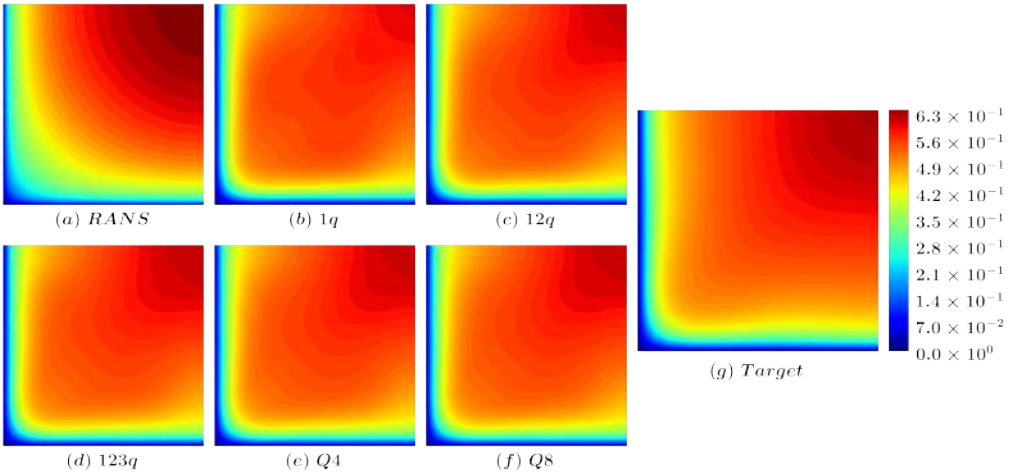


Figure 5.14: Corrected velocity field component U_x, by injecting in OF values of $\mathbf{R}$ predicted with the RF model.

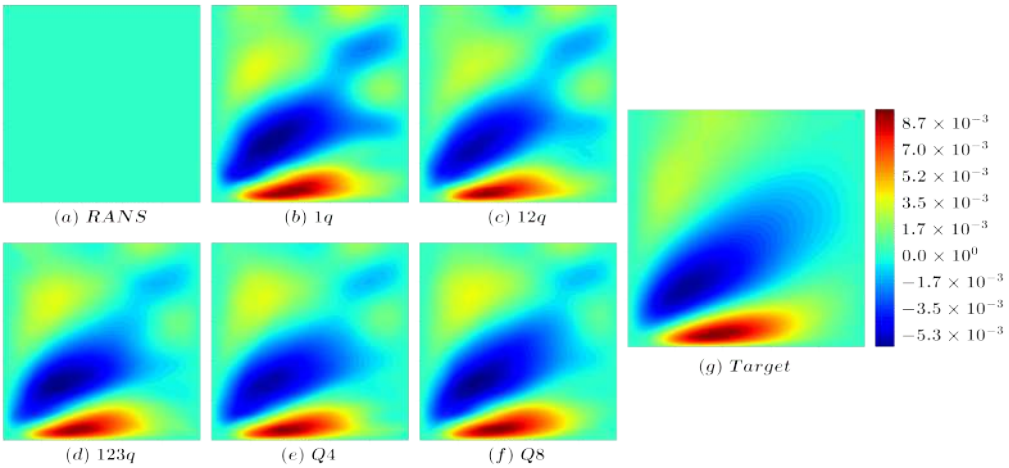


Figure 5.15: Corrected velocity field component U_y, by injecting in OF values of $\mathbf{R}$ predicted with the NN model.

good results in the main direction of the flow, an improvement from baseline RANS is evident.

In the secondary components, represented by U_y in Figs. 5.21b and 5.22b, where the baseline RANS model is incapable of capturing the physics involved, it is possible to observe a considerable improvement by implementing ML based turbulence models, with the RF based models yielding in better results.

It is also possible to observe that the DNS data from the quadrant $3q$ resulted in the worst errors for both main and secondary directions for the models based on $\mathbf{R}$ predicted by the NN while $1q$ resulted in the worst errors for the RF.

When comparing the results obtained by injecting $\mathbf{t}$ in OF, it is easy to observe a substantial improvement when compared with the baseline RANS model, as can be seen in Figs. 5.23 and 5.24. Also there is a significant improvement when compared to the results obtained by the same set of quadrants using the predicted $\mathbf{R}$ in Figs.

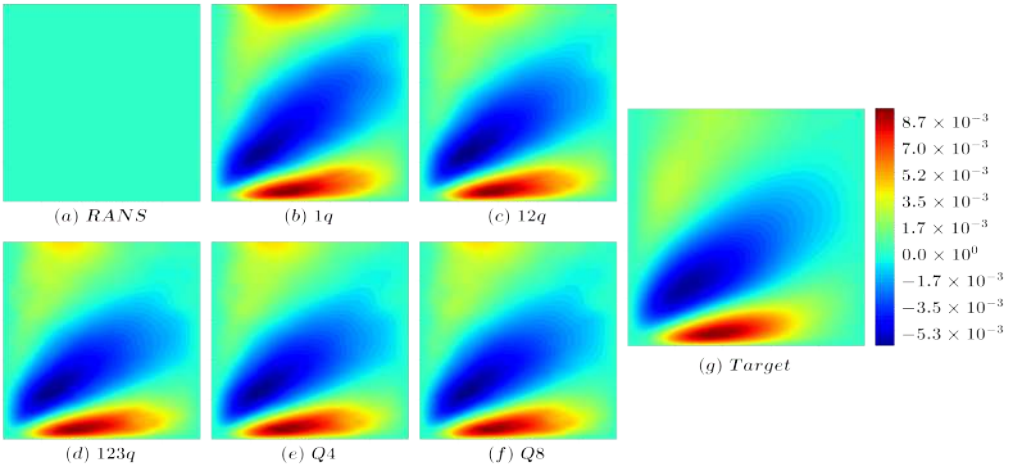


Figure 5.16: Corrected velocity field component U_y, by injecting in OF values of **R** predicted with the RF model.

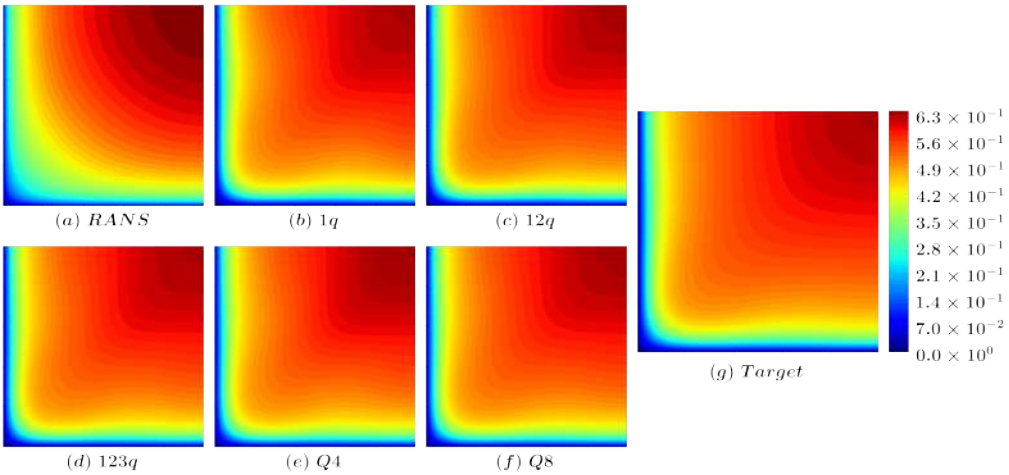


Figure 5.17: Corrected velocity field component U_x, by injecting in OF values of **t** predicted with the NN model.

5.21 and 5.22, as expected when using **t**-based models.

It can be observed in Figs. 5.23a and 5.24a that the RF based models presented better results in the main direction of the flow, but in the secondary direction, Figs. 5.23b and 5.24b show that the NN based models presented better results.

5.3.2 Q2

The results for the quadrant group $Q2$ aims to show how the ML model would perform if the DNS data used as output had an emulated simulation time two times greater than $Q1$. Figs. 5.25 and 5.26 show the global error of the corrected mean velocity field obtained by injecting **R** in OF, for the NN and RF, respectively. Comparing the results, it is possible to observe that the results of the RF based models are, in general, better than the NN based models.

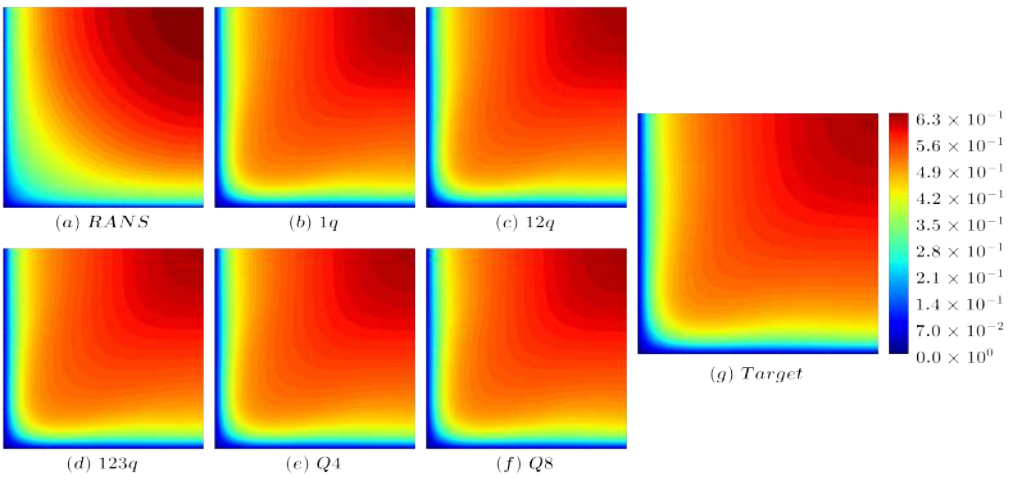


Figure 5.18: Corrected velocity field component U_x, by injecting in OF values of $\mathbf{t}$ predicted with the RF model.

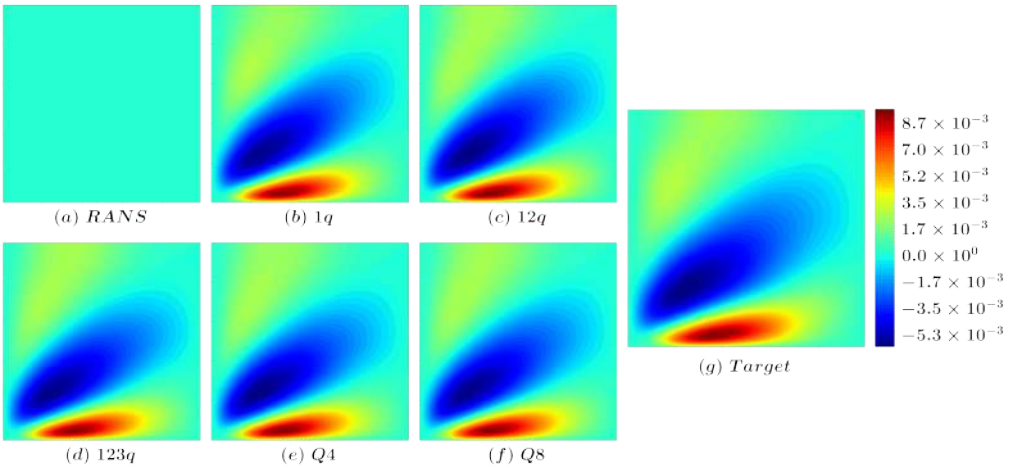


Figure 5.19: Corrected velocity field component U_y, by injecting in OF values of $\mathbf{t}$ predicted with the NN model.

It can be observed in Fig. 5.25a that, overall, for the NN based models, the global error has increased in the main direction of the flow, but the variance between different sets has decreased, when compared with $Q1$ results shown in Fig. 5.21a. This shows the inconsistency in the main direction of the mean velocity field obtained from ML models using second order statistic quantities as $\mathbf{R}$. In the secondary components represented by Fig. 5.25b and 5.26b, it is possible to observe an improvement when compared with the global error of $Q1$, shown in Figs. 5.21b and 5.22b.

Figs. 5.27 and 5.28 show that for both components of the flow, U_x and U_y, when injecting predicted values of $\mathbf{t}$ with greater emulated simulation time than the one of $Q1$, better results are obtained. As well as in $Q1$ results, RF based models present better results in the main direction while NN based models perform better in the secondary direction.

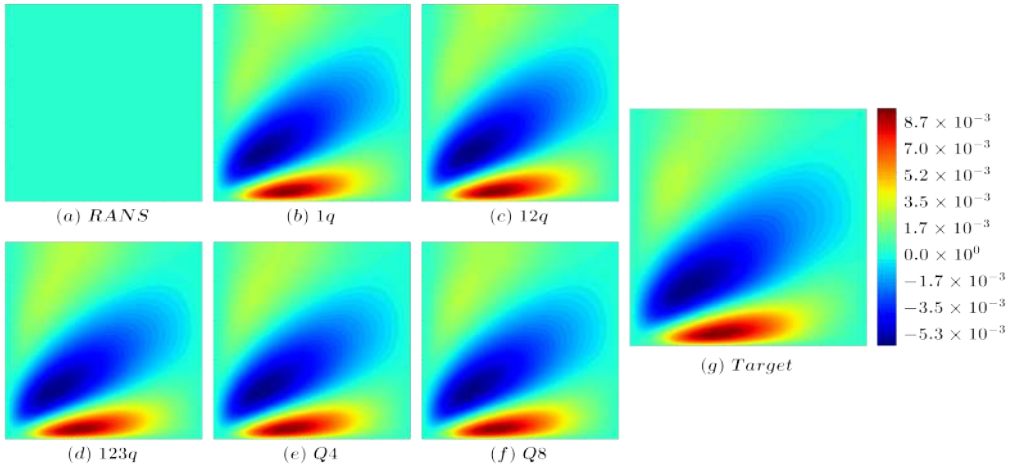
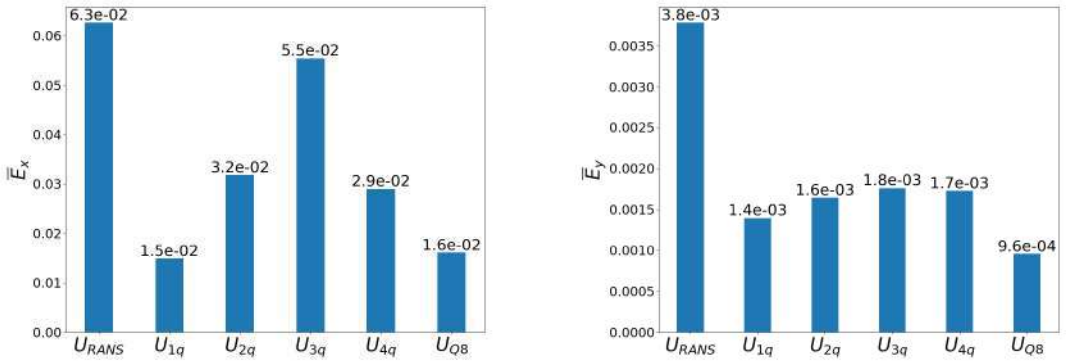


Figure 5.20: Corrected velocity field component U_y, by injecting in OF values of $\mathbf{t}$ predicted with the RF model.



(a) U_X (b) U_Y

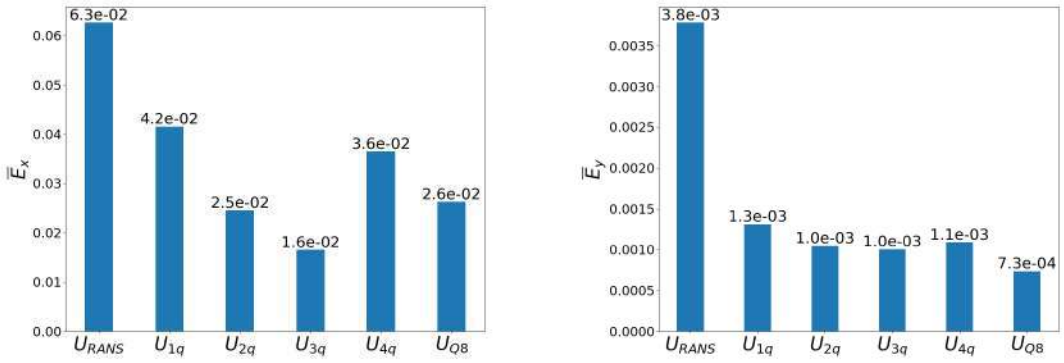
Figure 5.21: Global error of the velocity field for $Q1$ group injecting $\mathbf{R}$ predicted by the NN model.

5.3.3 Q3

Following the same methodology, $Q3$ shows how the ML model would perform if the DNS data used as output had an emulated simulation time three times greater than than the one of $Q1$. Figs. 5.29 and 5.30 show the global errors for models based on $\mathbf{R}$ predicted by NN and RF, respectively.

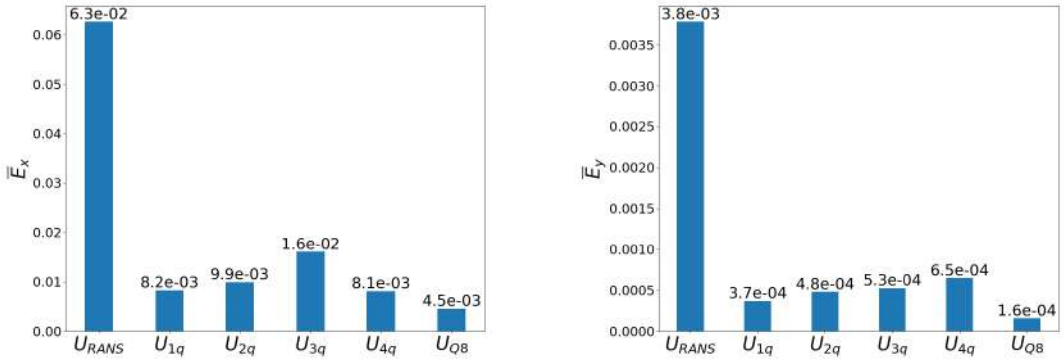
It can be observed that, overall, a decrease of the global error can be observed in Figs. 5.29a and 5.30a, for the main direction U_x in $Q3$, when compared to $Q1$ and $Q2$, for both NN and RF. Figs. 5.29b and 5.30b shows, as well, an overall decrease of the global error for the secondary component, U_y, which corroborates the capability of the ML model to learn from better converged DNS data.

For models based on $\mathbf{t}$, the global errors for $Q3$ can be observed in Figs. 5.31 and 5.32. Its is possible to ascertain that, overall, the global error for both directions decreases when compared to $Q1$ and $Q2$. Again, it can be observed, in general,



(a) U_X (b) U_Y

Figure 5.22: Global error of the velocity field for $Q1$ group injecting $\mathbf{R}$ predicted by the RF model.



(a) U_X (b) U_Y

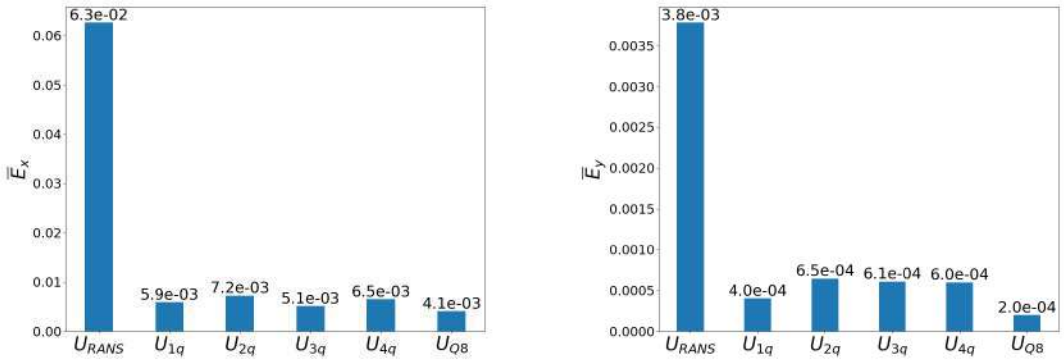
Figure 5.23: Global error of the velocity field for $Q1$ group injecting $\mathbf{t}$ predicted by the NN model.

a better result of the RF based models in the main direction while the NN based models perform better in the secondary direction.

5.3.4 Q4 & Q8

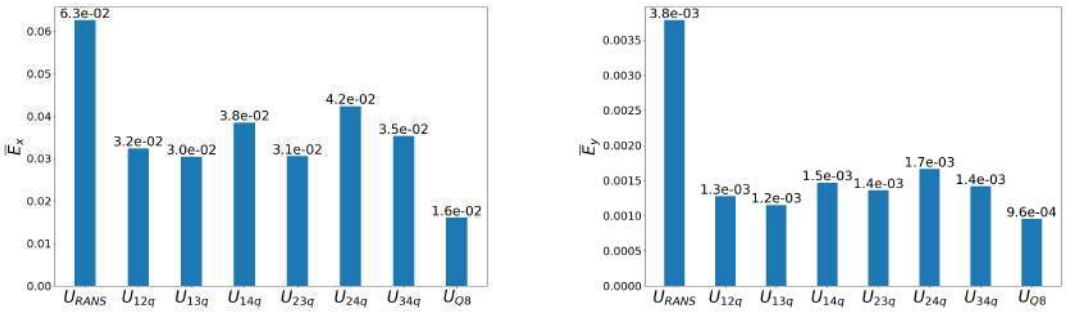
In order to present the results for all quadrant groups, the global error value for each combination group is the arithmetic mean for the global error for each quadrant set, as stated in Eq. (4.8). The overall global error for all groups based on $\mathbf{R}$ can be seen in Figs. 5.33 and 5.34.

It is possible to observe that in the main direction, U_x for NN based models, shown in Fig. 5.33a, increasing the emulated simulation time by adding quadrants did not result in a better corrected velocity field, as the global error did not decrease as expected. However, imposing symmetry, represented in $Q8$, yield the best global error in the main direction for all combination groups. This corroborates the diffi-



(a) U_X (b) U_Y

Figure 5.24: Global error of the velocity field for $Q1$ group injecting $\mathbf{t}$ predicted by the RF model.



(a) U_X (b) U_Y

Figure 5.25: Global error of the velocity field for $Q2$ group injecting $\mathbf{R}$ predicted by the NN model.

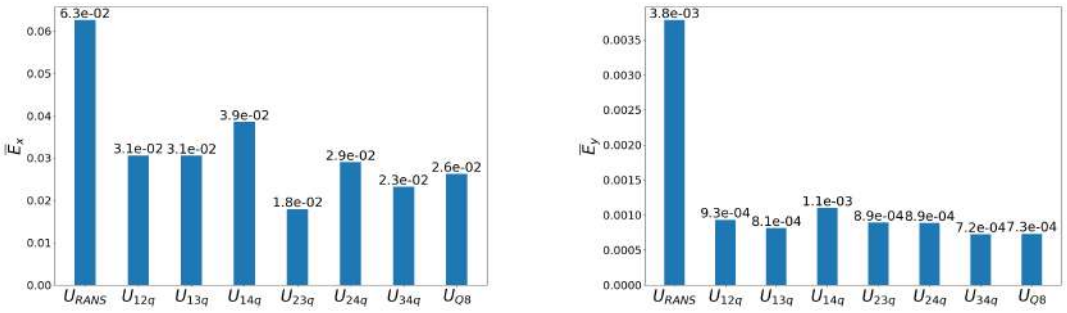
culty to replicate the main flow with ML models based on second order statistics, such as $\mathbf{R}$.

However, Fig. 5.34a shows the global errors in the main direction, for RF based models for all quadrant groups, where it is possible to observe that increasing the emulated simulation time by adding quadrants yields better corrected velocity fields in the main direction of the flow.

In the secondary direction, shown in Figs. 5.33b and 5.34b, it is possible to observe that emulating higher DNS simulation times results in better corrected velocity fields, with the best result being based on the most accurate DNS data, with full symmetry, $Q8$, with overall better results for the RF based models.

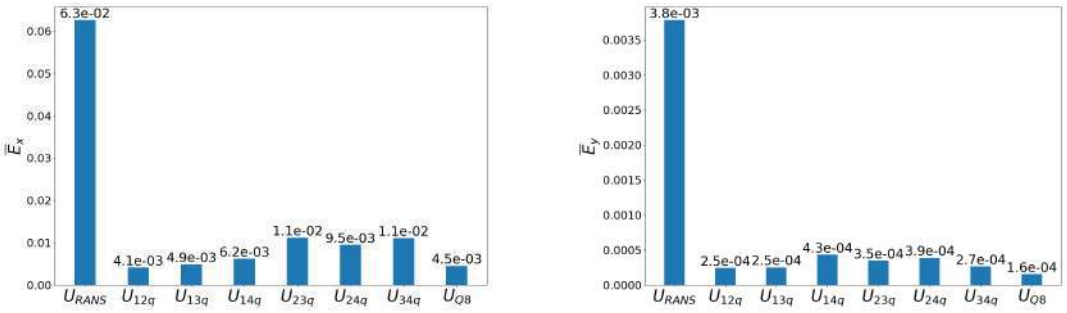
It can be stated that the RF predictions of $\mathbf{R}$ yielded better results when injected in OF, when compared with NN predictions.

Results based on $\mathbf{t}$ can be seen in Figs. 5.35 and 5.36. It can be observed, in Figs. 5.35a and 5.36a, that for the main direction of the flow, the evolution of the emulated simulation time results in does not necessary reflect in improvement in



(a) U_X (b) U_Y

Figure 5.26: Global error of the velocity field for $Q2$ group injecting $\mathbf{R}$ predicted by the RF model.



(a) U_X (b) U_Y

Figure 5.27: Global error of the velocity field for $Q2$ group injecting $\mathbf{t}$ predicted by the NN model.

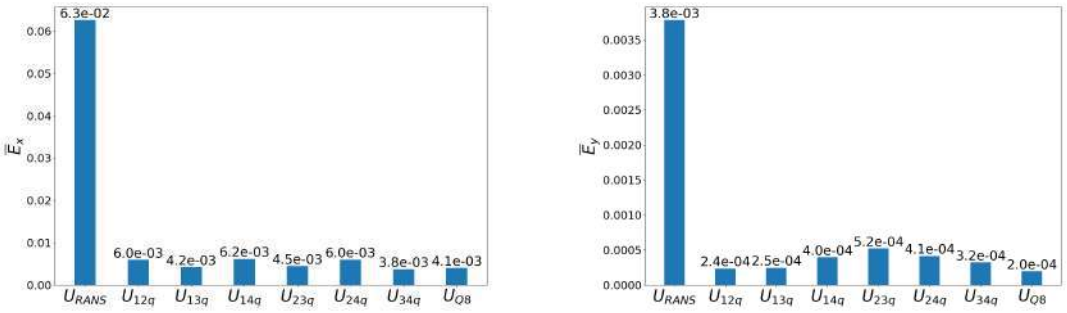
the global error. This can be observed since $Q4$ results have a higher error than $Q3$ and $Q2$ in NN based models and $Q2$ has a lower error than $Q3$ and $Q4$ for RF based models. However, when imposing full symmetry, on $Q8$, the best results are observed for both models.

In the secondary components, represented by U_y in Figs. 5.35b and 5.36b, it can be established a direct relation between emulating DNS simulation times and improving the corrected velocity field, with the best result associated with the fully symmetric case, $Q8$, for both NN and RF as well.

It is possible to observe that, in general, for predicted $\mathbf{t}$ injected in OF, RF based models performed better in the main direction of the flow, while NN based model performed better in the secondary direction.

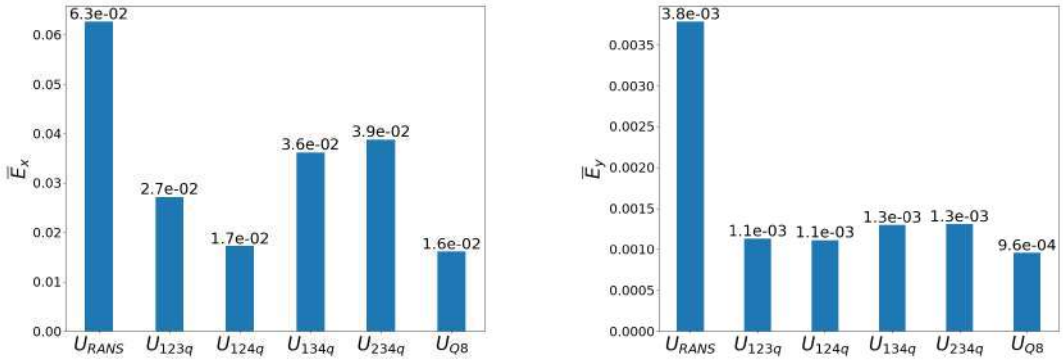
5.4 Imposed symmetry on ML models

In order to investigate the capability of ML models when trained with symmetric fields as outputs, the same symmetry conditions of Section 4.5 and the quadrant



(a) U_X (b) U_Y

Figure 5.28: Global error of the velocity field for $Q2$ group injecting $\mathbf{t}$ predicted by the RF model.



(a) U_X (b) U_Y

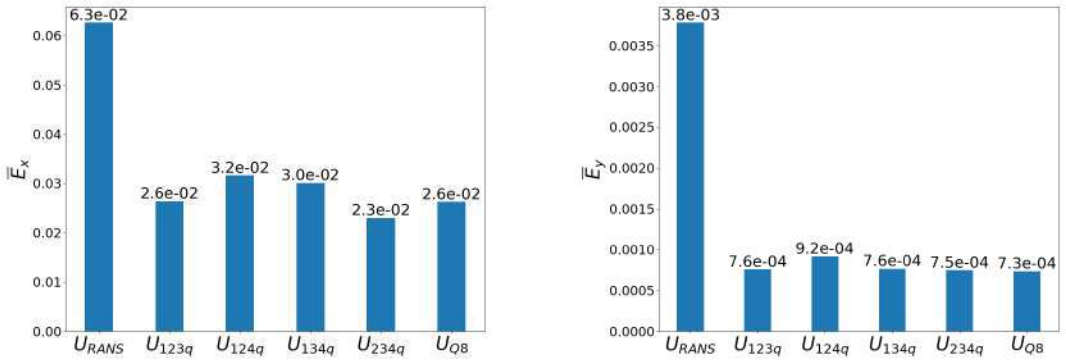
Figure 5.29: Global error of the velocity field for $Q3$ group injecting $\mathbf{R}$ predicted by the NN model.

group $Q8$ were applied to the set of $\mathbf{R}$ and $\mathbf{t}$ of the quadrant group $Q3$, which involves the quadrant sets $123q$, $124q$, $134q$, $234q$. For this new set of four $\mathbf{R}$ and four $\mathbf{t}$, eight NN and eight RF were built in order to predict the desired quantities.

When comparing the results shown in Fig. 5.37 to the original $Q3$ results shown in Fig. 5.29, for NN based predicted $\mathbf{R}$, it is possible to observe that for the main direction, imposing symmetry to the DNS data resulted in better results, overall, for the $Q3$ quadrant group. In the secondary direction, a slight improvement can be observed when imposing the symmetry.

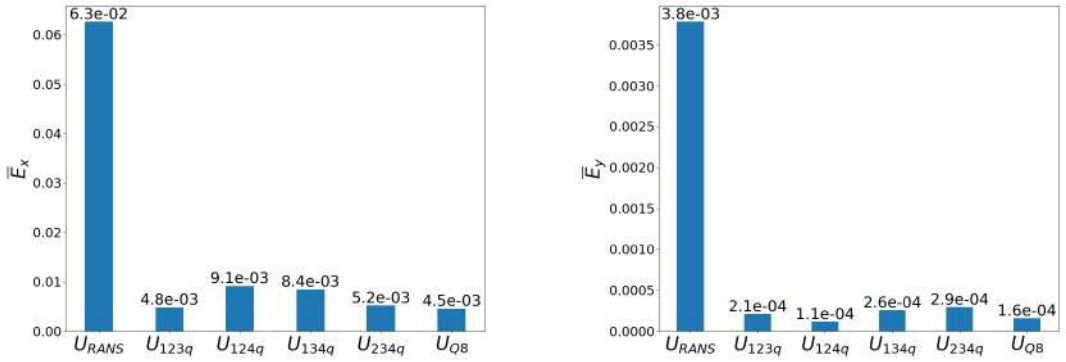
Fig 5.38 shows the results for imposing symmetry to the DNS data on the RF model. By comparing with the results shown in Fig. 5.30, it can be observe that, in general, for the RF predicted $\mathbf{R}$ no significant improvement was observed in the main direction of the flow, and a subtle improvement was observed in most of the quadrant sets for the secondary direction.

For the $\mathbf{t}$ predicted by the NN, Fig. 5.39 shows, when compared to Fig. 5.31, that for the main direction of the flow, it is not possible to ascertain that impos-



(a) U_X (b) U_Y

Figure 5.30: Global error of the velocity field for $Q3$ group injecting $\mathbf{R}$ predicted by the RF model.



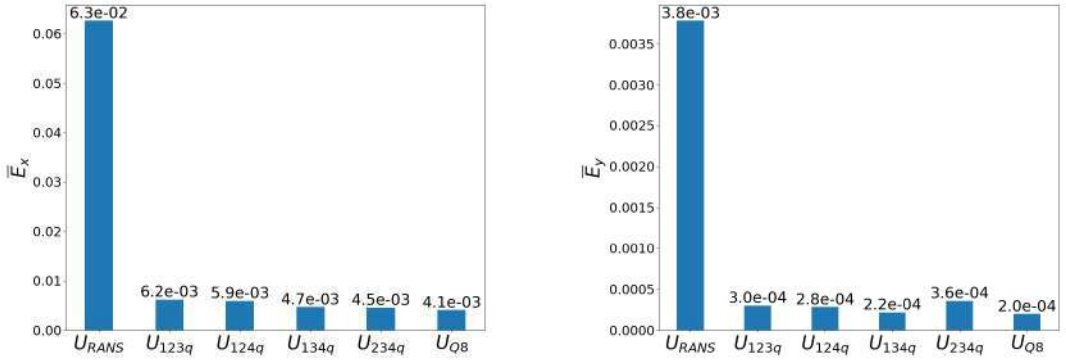
(a) U_X (b) U_Y

Figure 5.31: Global error of the velocity field for $Q3$ group injecting $\mathbf{t}$ predicted by the NN model.

ing symmetry resulted in better results. However, when analyzing the secondary direction, it is possible to observe that imposing symmetry culminated in better results.

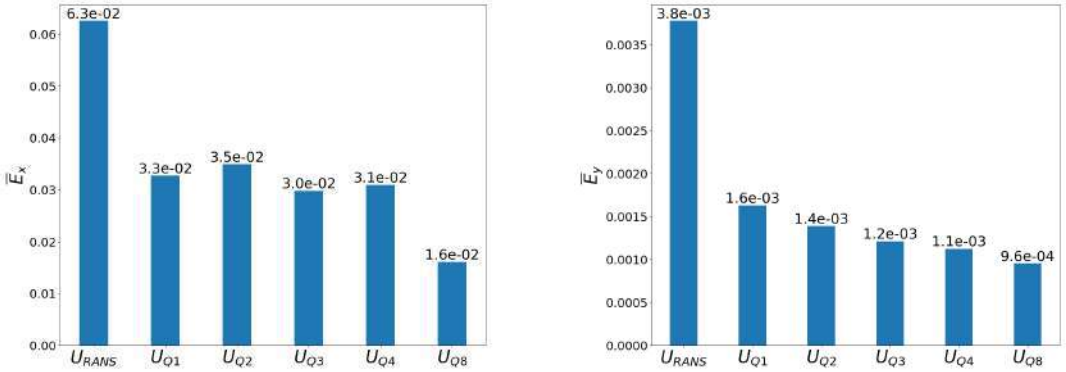
At last, Fig. 5.40 shows the results obtained when symmetry is imposed in $\mathbf{t}$ DNS data for the RF model. When comparing with previous results, shown in Fig. 5.32, it can be observed that, in general, imposing symmetry resulted in better accuracy for both the main and secondary directions.

These results show that, overall, ML models perform better when symmetry is imposed to the desired fields in ML applications on turbulence models.



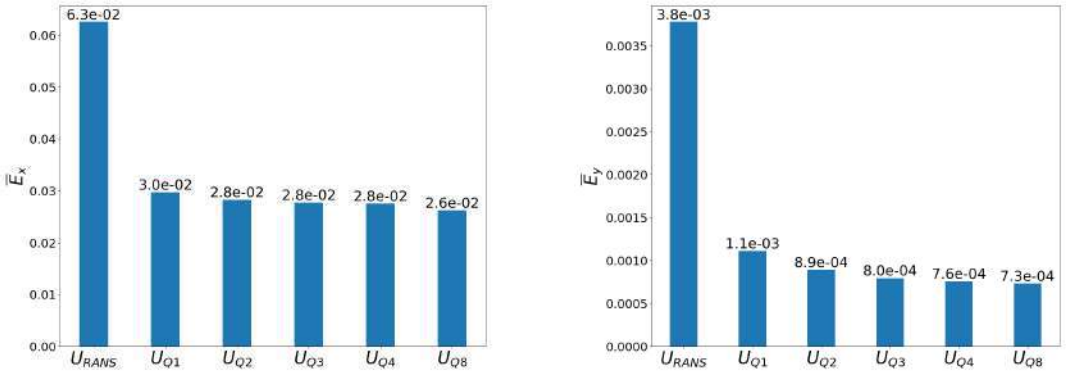
(a) U_X (b) U_Y

Figure 5.32: Global error of the velocity field for $Q3$ group injecting $\mathbf{t}$ predicted by the RF model.



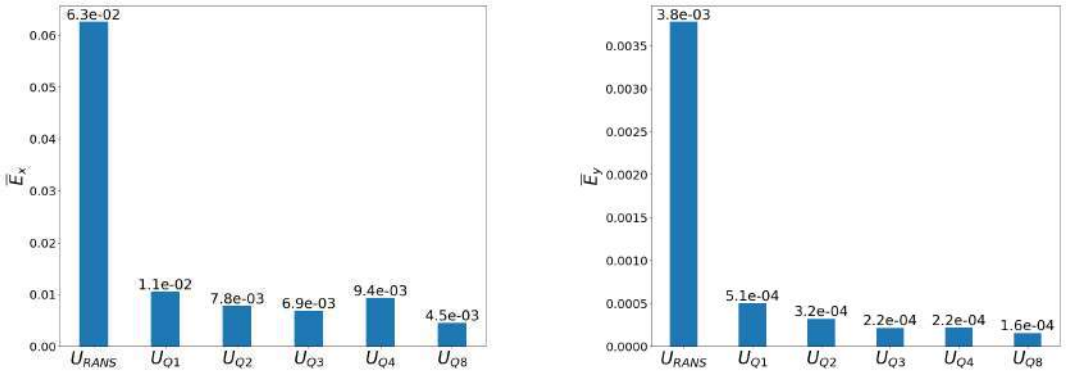
(a) U_X (b) U_Y

Figure 5.33: Global error of the velocity field for all quadrant groups injecting $\mathbf{R}$ predicted by the NN model.



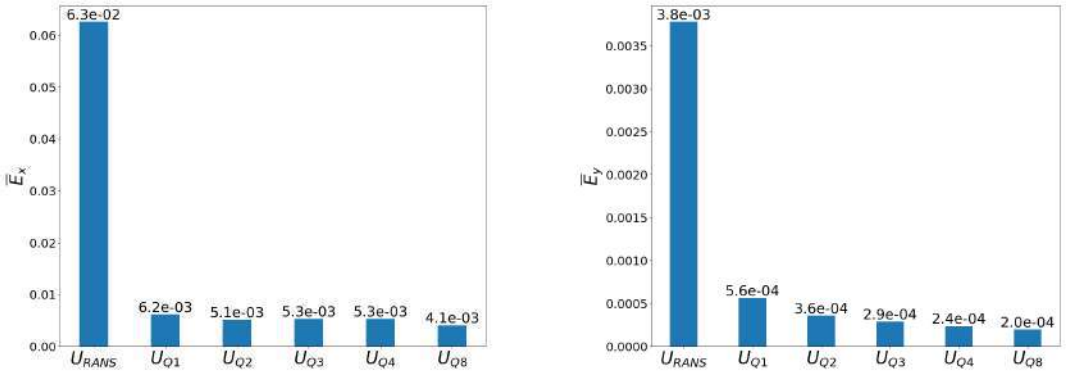
(a) U_X (b) U_Y

Figure 5.34: Global error of the velocity field for all quadrant groups injecting $\mathbf{R}$ predicted by the RF model.



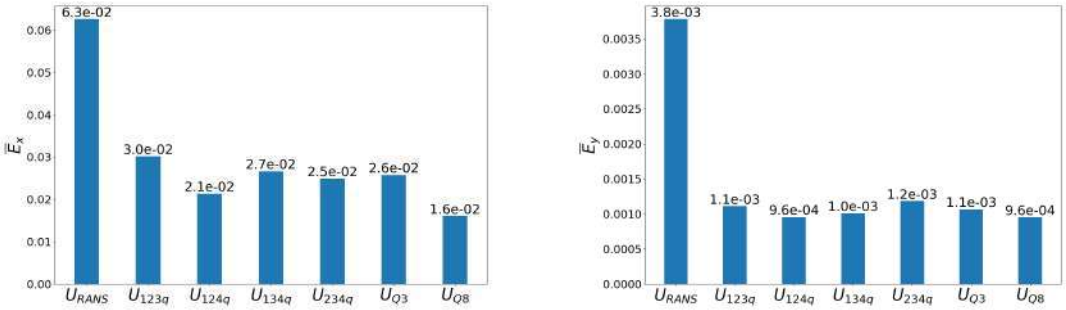
(a) U_X (b) U_Y

Figure 5.35: Global error of the velocity field for all quadrant groups injecting **t** predicted by the NN model.



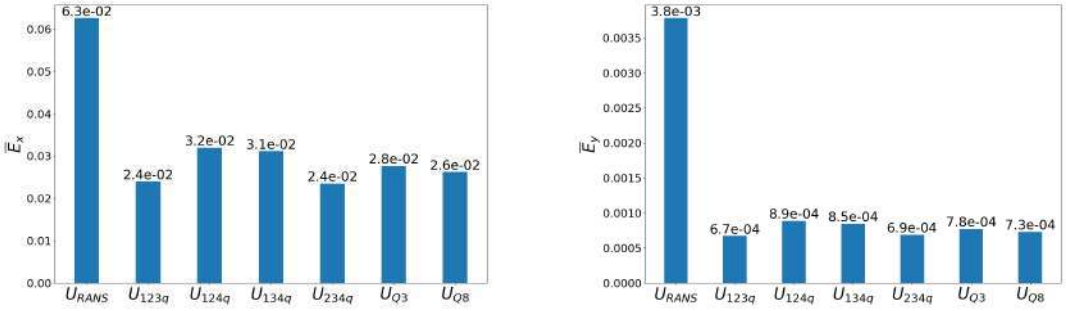
(a) U_X (b) U_Y

Figure 5.36: Global error of the velocity field for all quadrant groups injecting **t** predicted by the RF model.



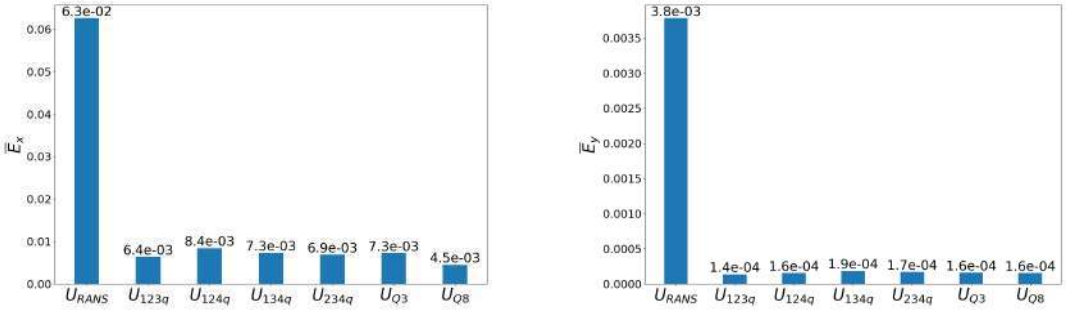
(a) U_X (b) U_Y

Figure 5.37: Global error of the velocity field for $Q3$ group injecting a symmetric **R** predicted by the NN model.



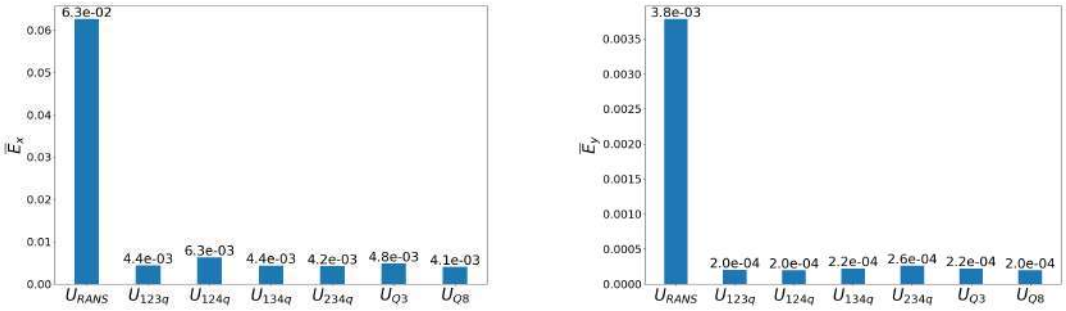
(a) U_X (b) U_Y

Figure 5.38: Global error of the velocity field for $Q3$ group injecting a symmetric **R** predicted by the RF model.



(a) U_X (b) U_Y

Figure 5.39: Global error of the velocity field for $Q3$ group injecting a symmetric **t** predicted by the NN model.



(a) U_X (b) U_Y

Figure 5.40: Global error of the velocity field for $Q3$ group injecting a symmetric **t** predicted by the RF model.

Chapter 6

Conclusion

In this work, NN and RF models were built using the KERAS and SCIKIT-LEARN libraries, respectively, in order to predict turbulent properties expressed by $\mathbf{R}$ and $\mathbf{t}$. These predictions were injected in OF to obtain corrected mean velocity fields with more accurate results for a turbulent square duct flow. The DNS database, made available by PINELLI *et al.* (2010), was manipulated by analyzing the available data in quadrants, in order to emulate longer DNS averaging simulation times and hence, obtain better convergence of the respective results. These data were used as output of ML models, with data from RANS simulations used as input.

As expected, the global error associated with the predicted values of $\mathbf{R}$ was higher than the error associated with the predicted values of $\mathbf{t}$, which corroborate the results obtained by CRUZ *et al.* (2019) and the expected results for the injection of these quantities in OF, shown in Figs. 4.9 and 4.10.

Results presented in this work showed that, as expected, the convergence of the DNS database, frequently used in ML applications for turbulence modeling, has a direct influence in the capability of a ML model to learn from the available data and predict quantities with better accuracy, which implies in better results for the corrected mean velocity fields. As expected, the error associated to the corrected mean velocity field, in comparison to the respective DNS mean velocity field, tend to decrease when emulating longer DNS averaging simulation times, which corroborate the results obtained by RANGEL (2019). The use of Euclidean invariant databases resulted in satisfactory predictions and corrected velocity fields for all cases studied, which is possible to assert comparing results for $\mathbf{R}$ and $\mathbf{t}$ obtained by RANGEL (2019) in Figs. 3.10 and 3.11 and the results presented in Figs. 5.33 to 5.36.

Analyzing all ML models and turbulent quantities predicted, $\mathbf{R}$ or $\mathbf{t}$, the best results were obtained for the target that presented the full symmetrical domain, represented by the quadrant group $Q8$. The use of the full symmetrical domain for ML applications is recommended as the computational cost to obtain this data is

significantly low, when compared to the costs to train ML models and run the OF simulations, and the predictions result in better corrected velocity fields.

Results showed that for velocity fields based on $\mathbf{R}$ predictions, the RF models presented, when compared to NN, an overall lower global error for both main and secondary directions of the flow. For velocity fields based on $\mathbf{t}$ predictions, RF models had lower global error in the main direction of the flow, but NN models captured the physics of the secondary flow with more accuracy. Therefore, analyzing the results obtained, the use of RF on ML models for turbulence application is recommended, as it yields satisfactory results for all cases studied and, when compared to NN, have a lower computational cost and are easier to understand and implement.

So, for ML applications where the DNS database presents some kind of symmetry in their geometry, as in the square duct pattern, the usage of symmetry filters that emulate longer DNS simulation averaging times and the usage of invariant ML databases are essential steps in order to obtain better ML predictions and, consequently, better corrected velocity fields.

6.1 Future Work

In order to expand the use of the methodology applied, different and more robust ML models, when compared to NN and RF, such as gradient boosting regression, could be implemented, although a higher computational cost is associated with them.

Another line of improvement is the use of optimization algorithms, such as a Bayesian optimization, to tune all hyper-parameters involved in each ML model developed, instead of using the same hyper-parameters for all models involved. Although involving a higher computational cost, this would yield the best ML architecture for each model and best set of outputs from the high-fidelity database.

This methodology can also be applied to another ML based turbulence models, such as the methodology proposed by WU *et al.* (2018), decomposing the quantity of interest into an implicit linear and explicit nonlinear terms.

References

ANDRADE, J. R., MARTINS, R. S., THOMPSON, R. L., et al., 2018, "Analysis of uncertainties and convergence of the statistical quantities in turbulent wall-bounded flows by means of a physically based criterion", *Physics of Fluids*, v. 30, n. 4 (abr.), pp. 045106. doi: 10.1063/1.5023500. Available at: <`https://doi.org/10.1063/1.5023500`>.

BANERJEE, S., ERTUNC, O., DURST, F., 2008, "Anisotropy Properties of Turbulence". In: *Proceedings of the 13th WSEAS International Conference on Applied Mathematics*, p. 26–57, Stevens Point, Wisconsin, USA. World Scientific and Engineering Academy and Society (WSEAS). ISBN: 9789604740345.

BOUSSINESQ, J., 1877, *Essai Sur La Théorie Des Eaux Courantes*. Mem. Pres. Acad. Sci.

BREIMAN, L., 2017, *Classification and regression trees*. Boca Raton, Chapman & Hall/CRC. ISBN: 1138469521.

CRAFT, T., LAUNDER, B., SUGA, K., 1996, "Development and application of a cubic eddy-viscosity model of turbulence", *International Journal of Heat and Fluid Flow*, v. 17, n. 2 (abr.), pp. 108–115.

CRUZ, M. A., THOMPSON, R. L., SAMPAIO, L. E., et al., 2019, "The use of the Reynolds force vector in a physics informed machine learning approach for predictive turbulence modeling", *Computers & Fluids*, v. 192 (out.), pp. 104258. doi: 10.1016/j.compfluid.2019.104258. Available at: <`https://doi.org/10.1016/j.compfluid.2019.104258`>.

CYBENKO, G., 1989, "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals, and Systems*, v. 2, n. 4 (dez.), pp. 303–314.

DAVIDSON, P. A., 2004, *Turbulence : an introduction for scientists and engineers*. Oxford, UK New York, Oxford University Press.

GENEVA, N., ZABARAS, N., 2019, "Quantifying model form uncertainty in Reynolds-averaged turbulence models with Bayesian deep neural networks", *Journal of Computational Physics*, v. 383 (abr.), pp. 125–147.

GURTIN, M., 2010, *The mechanics and thermodynamics of continua*. New York, Cambridge University Press. ISBN: 978-0-511-76980-1.

HINTON, G., 2012, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." *Coursera Lecture*. Available at: <http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf>.

JONES, W., LAUNDER, B., 1972, "The prediction of laminarization with a two-equation model of turbulence", *International Journal of Heat and Mass Transfer*, v. 15, n. 2 (fev.), pp. 301–314.

KAANDORP, M. L., DWIGHT, R. P., 2020, "Data-driven modelling of the Reynolds stress tensor using random forests with invariance", *Computers & Fluids*, v. 202 (abr.), pp. 104497. doi: 10.1016/j.compfluid.2020.104497. Available at: <https://doi.org/10.1016/j.compfluid.2020.104497>.

KERAS. "Keras Documentation". Available at: <https://keras.io/>. Accessed: 2020-04-15.

KINGMA, D. P., BA, J., 2015, "Adam: A Method for Stochastic Optimization", *CoRR*, v. abs/1412.6980.

LAUNDER, B. E., SHARMA, B. I., 1974, "Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc", *Letters Heat Mass Transfer*, v. 1 (dez.), pp. 131–137.

LING, J., JONES, R., TEMPLETON, J., 2016a, "Machine learning strategies for systems with invariance properties", *Journal of Computational Physics*, v. 318 (ago.), pp. 22–35.

LING, J., KURZAWSKI, A., TEMPLETON, J., 2016b, "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance", *Journal of Fluid Mechanics*, v. 807 (out.), pp. 155–166.

LING, J., RUIZ, A., LACAZE, G., et al., 2016c, "Uncertainty Analysis and Data-Driven Model Advances for a Jet-in-Crossflow", *Journal of Turbomachinery*, v. 139, n. 2 (out.), pp. 021008.

MARSLAND, S., 2014, *Machine Learning: An Algorithmic Perspective, Second Edition (Chapman & Hall/Crc Machine Learning & Pattern Recognition)*. Chapman and Hall/CRC.

MCCULLOCH, W. S., PITTS, W., 1943, "A logical calculus of the ideas immanent in nervous activity", *The Bulletin of Mathematical Biophysics*, v. 5, n. 4 (dec), pp. 115–133.

MILANO, M., KOUMOUTSAKOS, P., 2002, "Neural Network Modeling for Near Wall Turbulent Flow", *Journal of Computational Physics*, v. 182, n. 1 (out.), pp. 1–26.

MUCK, K. C., HOFFMANN, P. H., BRADSHAW, P., 1985, "The effect of convex surface curvature on turbulent boundary layers", *Journal of Fluid Mechanics*, v. 161, n. -1 (dez.), pp. 347.

PINELLI, A., UHLMANN, M., SEKIMOTO, A., et al., 2010, "Reynolds number dependence of mean flow structure in square duct turbulence", *Journal of Fluid Mechanics*, v. 644 (fev.), pp. 107–122. doi: 10.1017/s0022112009992242. Available at: <https://doi.org/10.1017/s0022112009992242>.

POPE, S. B., 1975, "A more general effective-viscosity hypothesis", *Journal of Fluid Mechanics*, v. 72, n. 02 (nov.), pp. 331.

POPE, S. B., 2000, *Turbulent flows*. Cambridge New York, Cambridge University Press. ISBN: 978-0-521-59886-6.

RANGEL, V. B., 2019, *Influência do tratamento da base de dados DNS na aplicação de técnicas de aprendizagem de máquina para melhorar acurácia de simulações RANS*. Tese de Mestrado, Universidade Federal do Rio de Janeiro. COPPE Programa de Engenharia Mecânica.

REYNOLDS, O., 1895, "On the Dynamical Theory of Incompressible Viscous Fluids and the Determination of the Criterion." *Philosophical Transactions of the Royal Society A*, v. 186, pp. 123–164.

RUDER, S., 2016, "An overview of gradient descent optimization algorithms", v. abs/1609.04747. Available at: <http://arxiv.org/abs/1609.04747>.

RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J., 1986, "Learning representations by back-propagating errors", *Nature*, v. 323, n. 6088 (out.), pp. 533–536.

SCHMITT, F. G., 2007, "About Boussinesq turbulent viscosity hypothesis: historical remarks and a direct evaluation of its validity", *Comptes Rendus Mecanique*, v. 335, n. 9-10 (set.), pp. 617–627.

SCIKIT-LEARN. "Scikit-Learn User Guide". Available at: <https://scikit-learn.org/stable/user_guide.html>. Accessed: 2020-04-10.

SNOEK, J., LAROCHELLE, H., ADAMS, R. P., 2012, "Practical Bayesian Optimization of Machine Learning Algorithms", v. arXiv: 1206.2944. Available at: <https://arxiv.org/pdf/1206.2944.pdf>.

SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., et al., 2014, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, v. 15, n. 56, pp. 1929–1958. Available at: <http://jmlr.org/papers/v15/srivastava14a.html>.

SUPERDATASCIENCE. "Biological neuron structure". Available at: <https://www.superdatascience.com/courses/deep-learning-az>. Accessed: 2019-05-22.

TENNEKES, H., LUMLEY, J. L., 1972, *A first course in turbulence*. Cambridge, Mass, MIT Press.

THOMPSON, R. L., MENDES, P. R. S., 2005, "Persistence of straining and flow classification", *International Journal of Engineering Science*, v. 43, n. 1-2 (jan.), pp. 79–105.

THOMPSON, R. L., SAMPAIO, L. E. B., DE BRAGANCA ALVES, F. A., et al., 2016, "A methodology to evaluate statistical errors in DNS data of plane channel flows", *Computers & Fluids*, v. 130 (maio), pp. 1–7.

TRACEY, B., DURAISAMY, K., ALONSO, J., 2013, "Application of Supervised Learning to Quantify Uncertainties in Turbulence and Combustion Modeling". In: *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. American Institute of Aeronautics and Astronautics, jan.

TRACEY, B. D., DURAISAMY, K., ALONSO, J. J., 2015, "A Machine Learning Strategy to Assist Turbulence Model Development". In: *53rd AIAA Aerospace Sciences Meeting*. American Institute of Aeronautics and Astronautics, jan.

WILCOX, D., 2006, *Turbulence modeling for CFD*. 5354 Palm Drive, La Canada, California, DCW Industries.

WU, J.-L., XIAO, H., PATERSON, E., 2018, "Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework", *Physical Review Fluids*, v. 3, n. 7 (jul.).

XIAO, H., WU, J.-L., WANG, J.-X., et al., 2016, "Quantifying and reducing model-form uncertainties in Reynolds-averaged Navier-Stokes simulations: A data-driven, physics-informed Bayesian approach", *Journal of Computational Physics*, v. 324 (nov.), pp. 115–136.

YARLANKI, S., RAJENDRAN, B., HAMANN, H., 2012, "Estimation of turbulence closure coefficients for data centers using machine learning algorithms". In: *13th InterSociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*. IEEE, maio.