
# MACHINE LEARNING TECHNIQUES FOR ACCURACY IMPROVEMENT OF RANS SIMULATIONS

Matheus Altomare Cruz

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
Fevereiro de 2018

MACHINE LEARNING TECHNIQUES FOR ACCURACY IMPROVEMENT OF RANS SIMULATIONS

Matheus Altomare Cruz

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

Examinada por:

Prof. Roney Leon Thompson, D.Sc.

Prof. Gustavo Cesar Rachid Bodstein, Ph.D.

Prof. Aristeu da Silveira Neto, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2018

Cruz, Matheus Altomare

Machine learning techniques for accuracy improvement of RANS simulations/Matheus Altomare Cruz. – Rio de Janeiro: UFRJ/COPPE, 2018.

XII, 78 p.: il.; 29, 7cm.

Orientador: Roney Leon Thompson

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Mecânica, 2018.

Referências Bibliográficas: p. 75 – 78.

1. Turbulent flows. 2. Machine learning. 3. OpenFOAM. I. Thompson, Roney Leon. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Mecânica. III. Título.

*Gostaria de dedicar este trabalho*
*a mim mesmo, pois ele é prova*
*do que eu sou capaz.*

# Agradecimentos

Gostaria de agradecer primeiramente à minha família. Pois ela é o início de tudo, sendo nossa base e refúgio para qualquer momento difícil. Sem vocês, tenho certeza que não estaria aqui. Essa conquista é tão minha quanto de vocês.

A minha namorada Johanna. Que sempre me ajudou a caminhar na direção dos meus sonhos, estando sempre do meu lado para me apoiar, e confortar nas horas difíceis. Você é o meu anjo.

Ao meu orientador, e amigo, Prof. Roney Thompson. O senhor é uma verdadeira inspiração para todos que desejam seguir uma carreira acadêmica. Tenho muito o que agradecer por tudo que aprendi, e orgulho de ser seu aluno. Obrigado por toda a paciência e confiança depositadas em mim ao longo desta jornada.

Ao grupo maravilhoso do laboratório de Aerodinâmica da COPPE-RJ, que me acolheu com muito carinho. Todos os almoços e churrascos foram muito TOPs!

A todos da empresa Wikki Brasil, que me ajudaram muito ensinado-me a mexer no OpenFOAM. Isto me ajudou enormemente a desenvolver este trabalho.

Ao Prof. Luiz Eduardo e ao Raphael David que me ajudaram bastante a desenvolver os códigos e metodologias usadas neste trabalho.

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

# TÉCNICAS DE APRENDIZAGEM DE MÁQUINA PARA MELHORAR A PRECISÃO DE SIMULAÇÕES RANS

Matheus Altomare Cruz

Fevereiro/2018

Orientador: Roney Leon Thompson

Programa: Engenharia Mecânica

Há um grande número de escoamentos de interesse que são turbulentos. Como a Simulação Numérica Direta (DNS) e os experimentos são caros, o uso dos modelos RANS (Reynolds Average Navier-Stokes) torna-se uma necessidade. No entanto, tal abordagem possui pouca precisão. Este fato justifica a alta demanda por melhores modelos. Neste trabalho, uma técnica que usa o aprendizado de máquina, por meio de redes neurais, é usada para corrigir o modelo RANS $\kappa$ - $\epsilon$, considerando os dados DNS como ideais. As metodologias disponíveis na literatura empregam o tensor de tensão de Reynolds como a principal quantidade a ser corrigida. Uma vez que esta entidade é corrigida, o campo de velocidade é recalculado pelas equações de transporte RANS. Consequentemente, o campo de velocidade obtido se aproxima dos resultados de DNS. No entanto, no presente trabalho, essa metodologia é criticada devido à existência de incertezas no campo de tensões turbulentas fornecido pelos bancos de dados DNS. Sabe-se que os momentos estatísticos de segunda ordem (tensor de Reynolds) não são tão bem convergidos quanto os de primeira ordem (campos de velocidade e pressão médios) em simulações de DNS. Essas incertezas são propagadas e contaminam a velocidade média calculada a partir da mesma. Por esta razão, propõe-se, como nova metodologia, a correção do divergente do tensor de Reynolds, por ser é a única parte que de fato entra no balanço de momentum linear médio. Esta divergência pode ser calculada a partir dos campos de velocidade média e pressão, que são bem convergidos, utilizando a equação de balanço de quantidade de movimento linear médio. Os resultados obtidos até agora demonstraram que a correção do campo divergente das tensões turbulentas RANS é capaz de reconstruir os campos de velocidade média mais próximos do DNS do que a correção completa do tensor geralmente empregado na literatura.

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

MACHINE LEARNING TECHNIQUES FOR ACCURACY IMPROVEMENT OF RANS SIMULATIONS

Matheus Altomare Cruz

February/2018

Advisor: Roney Leon Thompson

Department: Mechanical Engineering

There is a wide number of applications where the flow is turbulent. Since Direct Numerical Simulation (DNS) and experiments are expensive, the use of Reynolds Average Navier-Stokes (RANS) models is a necessity. However, the obtained models from this approach have low accuracy. This fact justifies the high demand for better models. In this work, a technique that uses machine learning, by means of neural networks, is used to correct the $\kappa$-$\epsilon$ RANS model considering the DNS data as ideal. The methodologies available in the literature employ the Reynolds stress tensor as the main quantity to be corrected. Once this entity is corrected, the velocity field is recalculated by the RANS transport equations. Consequently, the obtained velocity field gets closer to DNS results. However, in the present work, such methodology is criticized due to the existence of uncertainties in the turbulent stress field provided by DNS databases. It is known that the second-order statistical moments (Reynolds stress tensor) are not as well converged as the first order ones (mean velocity and pressure fields) in DNS simulations. These uncertainties are propagated, and contaminate the mean velocity field calculated from it. For this reason, it is proposed, as a new methodology, the correction of the divergent of the Reynolds stress tensor, because it is the only part that is computed in the mean linear momentum balance. This divergence can be calculated from the mean velocity and pressure fields, which are well converged, using the mean linear momentum equation. The results obtained so far have demonstrated that the divergent correction of the RANS turbulent stress field is able to reconstruct mean velocity fields closer to the DNS than the complete tensor correction usually employed in the literature.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

There is a wide number of applications where the flow is turbulent. Since Direct Numerical Simulation (DNS) and experiments are prohibitively expensive for complex geometries and high Reynolds number, the use of Reynolds Average Navier-Stokes (RANS) models is a necessity. However, the obtained models from this approach have low accuracy. This fact justifies the high demand for better models. In this work, a technique that uses machine learning, by means of neural networks, will be used to correct $\kappa$-$\epsilon$ RANS model with the aim to approach DNS results. The methodologies present in the literature corrects the RANS Reynolds stress tensor with machine learning techniques. After that, the velocity field is recalculated by the RANS transport equations. Consequently, the obtained velocity field gets closer to DNS results. However, in the present work, such methodology is criticized due to the existence of uncertainties in the turbulent stress field provided by DNS databases. It is known that the second-order statistical moments (Reynolds stress tensor) are not as well converged as the first order ones (mean velocity and pressure fields) in DNS simulations. These uncertainties are propagated, and contaminate the velocity field calculated from it. For this reason, it is proposed, as a new methodology, the correction of the divergent of the Reynolds stress tensor, because it is the only part that is computed in the mean linear momentum balance. This divergence can be calculated from the mean velocity and pressure fields, which are better converged, using the mean linear momentum equation. Thus, it is demonstrated that the divergent correction of the RANS turbulent stress field obtains velocity fields closer to the DNS than the complete tensor correction usually employed in the literature in the axial direction of the square duct flow.

Reynolds-everaged Navier-Stokes (RANS) simulations are widely employed in industrial turbulent fluid dynamics applications. The most popular two-equations models, assumes the Boussinesq hypothesis [12], a linear relation between Reynolds stress tensor ($\mathbf{R}$) and the mean strain rate tensor ($\mathbf{D}$) defined by a turbulent viscosity. But, these models do not capture all the physics of some relevant flows, com-

promising their results reliability. HOFFMAN *et al.* [13] shows that those models do not provide satisfactory results in curvature flows, e.g., secondary flows in ducts. Flows with strong gradients and separation are poorly predicted when analysed by RANS models [14] as well.

In the past few years, some studies have been made to apply machine learning (ML) techniques into turbulent fluid mechanics problems in order to develop Reynolds stress closures. MILANO e KOUMOUTSAKOS [15] used a multiple hidden layer neural network (NN) to replicate a near-wall channel flow, however that NN does not predict the turbulence in forward flows. TRACEY *et al.* [7] used kernel regression to model turbulent stress anisotropy eigenvalues. They reported some difficulties in generalizing the results to new flows and to scale to a large amount of data. After that, TRACEY *et al.* [8] proposed a single hidden layer NN to model the source terms from the Spalart Allmaras RANS model. This attempt has shown the potential of NN for turbulence modeling. ZHANG e DURAISAMY [16] has used NN to correct the production term, only affecting the magnitude, but not the anisotropy, of $\mathbf{R}$. LING *et al.* [9] used a random forest regression to model the deviatoric part of $\mathbf{R}$. This technique presented a poor ability to predict this tensor because of difficulty to ensure Galilean invariance. Later, LING *et al.* [17] developed a NN to predict the anisotropy Reynolds stress tensor eigenvalues using a set of Galilean invariants inputs. That technique showed significant performance gains when compared to [9]. Those results evidenced the importance of the Galilean invariance in ML turbulence modelling. More recently, LING *et al.* [18] developed a deep specialized NN architecture witch ensured Galilean invariant anisotropic turbulent tensor. Those results showed that deep learning and the specialized architecture is able to provide a significant performance improvement.

In the present work, a different ML strategy is proposed to improve turbulence modeling. Instead of predicting the anisotropy Reynolds stress tensor, a NN is employed to correct its divergence. The main idea is to correct the term that really affects the mean linear momentum equation (MLME), $(\langle\mathbf{v}\rangle \cdot \boldsymbol{\nabla})\mathbf{v} - \nu\boldsymbol{\nabla}^2\langle\mathbf{v}\rangle + 1/\rho\boldsymbol{\nabla}\langle p\rangle = \mathbf{t}$. The proposed NN learns $\mathbf{t}$, and this prediction is used to correct the turbulent flow. Another original aspect of the new strategy is to employ objective tensorial entities as inputs for the NN training, what can ensure a more generic invariance than the ones proposed by POPE [19], and used in [18]. The motivation behind the use of $\mathbf{t}$ is the $\mathbf{R}_{\mathrm{DNS}}$ intrinsic uncertainties. THOMPSON *et al.* [20] have shown that turbulent stress is not as well converged as the mean velocities and pressure in DNS simulations, and this lack of convergence creates uncertainties that are propagated and amplified to the mean velocity and pressure. In other words, the mean velocity and pressure fields reconstructed from $\mathbf{R}_{\mathrm{DNS}}$, when compared to the given fields, are significantly different. This result implies that if a NN is

trained to achive $\mathbf{R}_{\mathrm{DNS}}$, the calculated velocity and pressure fields be will contaminated by the intrinsic DNS uncertainties. Creating a NN to correct $\mathbf{t}$ dribbles that problem, $\mathbf{t}_{\mathrm{DNS}}$ can be calculated from terms of the well-converged DNS fields by MLME, $\mathbf{t} = (\langle\mathbf{v}\rangle \cdot \boldsymbol{\nabla})\langle\mathbf{v}\rangle + 1/\rho\boldsymbol{\nabla}\langle p\rangle - \nu\boldsymbol{\nabla}^2\langle\mathbf{v}\rangle$. When the high-fidelity pressure field is not provided, $\mathbf{t}$ is redefined as $\mathbf{t} = -1/\rho\boldsymbol{\nabla}\langle p\rangle - \boldsymbol{\nabla}\cdot\mathbf{R}$ incorporating the pressure gradient term. In order to obtain the corrected velocity field, the equation $(\langle\mathbf{v}\rangle \cdot \boldsymbol{\nabla})\langle\mathbf{v}\rangle - \nu\boldsymbol{\nabla}^2\langle\mathbf{v}\rangle = \mathbf{t}$ will be solved. Both techniques, correcting $\mathbf{R}$ and $\mathbf{t}$, are evaluated and compared in this work.

# Chapter 2

# Turbulent flows

Historically, high viscous, or very low velocity flows, are called laminar flows because these flows are characterized by overlapping blades of moving fluid, WILCOX [21]. When small disturbances act on the flow, if the viscous effects are sufficiently high, then such perturbations will not be propagated and magnified. However, when such viscous effects are not sufficiently high, then these perturbations cause instability in the flow, TENNEKES e LUMLEY [22] causing the transition from a laminar to a turbulent state. The balance between the viscous and inertial forces is evaluated by the Reynolds number,

$$Re = \frac{UL}{\nu}, \tag{2.1}$$

where $U$ and $L$ are, respectively, characteristic velocity and length of the flow. The larger this dimensionless number, the less viscous effects compared to inertia. Therefore, the perturbations cause instabilities in the flow, leading them to the turbulent state, when this number is high. As the viscosity of most fluids is extremely low, then the turbulent regime is much more abundant in nature than the laminar, DAVIDSON [1].

## 2.1 Fundamental characteristics

Given the great complexity of this phenomenon, it is difficult to establish a precise definition. However, one can list fundamental characteristics of turbulence. In this session will be listed the main characteristics described by TENNEKES e LUMLEY [22].

### 2.1.1 Multiple scales

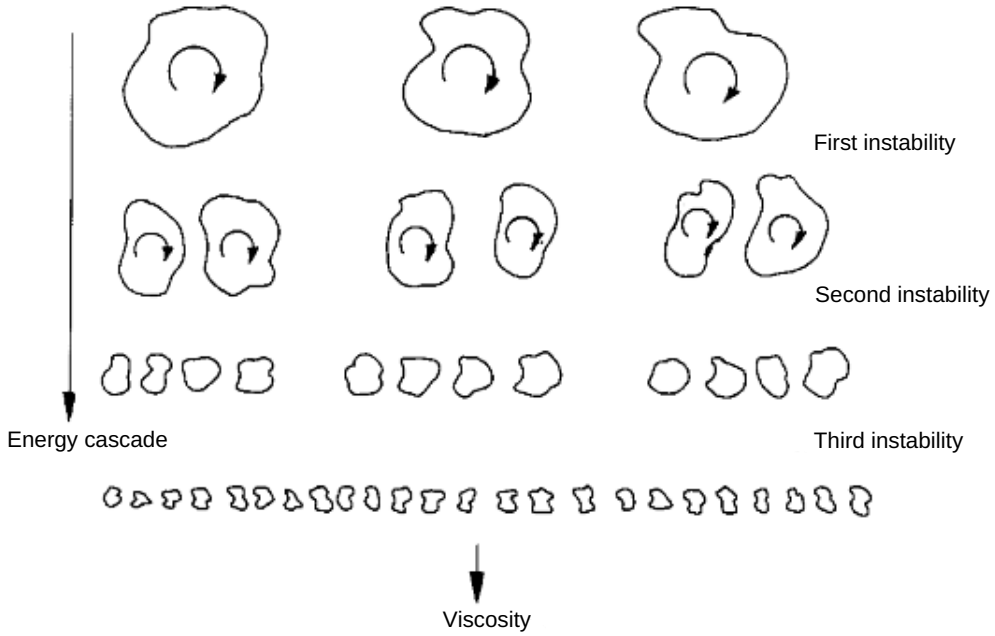



Figure 2.1: Energy scale, DAVIDSON [1].

The coherent structures present in turbulent flows, called vortex, are highly unstable. They are constantly decomposed into smaller structures. In turn, these smaller structures follow the same pattern of decomposition in even smaller vortices. This pattern repeats to a lower boundary threshold, which defines the size of smaller structures. RICHARDSON [23] then conjectured that this breakdown of structures in other minor vortex occurs through the transfer of energy from the larger structures to the smaller ones. This energy transfer from major structures to smaller ones is known as the energy cascade. This holds to the point where, at the smallest possible scales, viscous dissipation transforms the energy contained in the smaller structure into thermal energy. That process is illustrated in Fig. (2.1).

KOLMOGOROV [24], from the energy cascade concept, conjectured that in the smallest structures the viscous dissipation depends only on the turbulent kinetic energy ($\kappa$) and the turbulent dissipation ($\epsilon$). At this scale, the dissipation is isotropic, since it has completely lost information about the larger scales. The smaller structure possesses an equivalence of the viscous and inertial forces, that is,

$$Re_\eta = \frac{L_\eta U_\eta}{\nu} = 1. \tag{2.2}$$

Writing a characteristic velocity ($U_\eta$) of this scale in function of relevant quantities,

$$U_\eta \sim (\nu\epsilon)^{1/4}, \tag{2.3}$$

and using Eq. (2.2), it is possible to define a characteristic length of the smaller scale,

$$L_\eta \sim \left(\frac{\nu^3}{\epsilon}\right)^{1/4}. \tag{2.4}$$

By dimensional analysis, the turbulent dissipation is written in terms of the characteristic velocity and length of the larger scales,

$$\epsilon \sim \frac{U^3}{L}. \tag{2.5}$$

The ratio between the largest scale, referring to the geometry of the problem, and the smallest scale (Kolmogorov scale) can be calculated from the Eqs. (2.4) and (2.5),

$$\frac{L}{L_\eta} \sim L\left(\frac{\nu^3}{\epsilon}\right)^{-1/4} \sim \left(\frac{L^{-4}\nu^3}{LU^3}\right)^{-1/4} \sim \left(\frac{\nu}{LU}\right)^{-3/4},$$

$$\frac{L}{L_\eta} \sim Re^{3/4}. \tag{2.6}$$

So, the ratio between the characteristics volumes are

$$\left(\frac{L}{L_\eta}\right)^3 \sim \frac{V}{V_\eta} \sim Re^{9/4}. \tag{2.7}$$

In order to capture all the scales of a turbulent flow, it is necessary to have a mesh where the smallest dimension is comparable to that Kolmogorov scale. E.g., for a given flow with a fairly low Reynolds of 1000, by the Eq. (2.7), the simulation will must to have approximately 5.6 million centroids. That is, the Kolmogorov scale reveals that a full description of a turbulent flow requires a very high computational cost, which is prohibitive with the variable technology for the vast majority of applications. Even so, there is a class of simulations called direct numerical simulation (DNS) that perform this kind of analysis. They are extremely expensive, and are only made for simple geometries and low Reynolds numbers. In Fig. 2.2 it is seen the Reynolds number influence in the size of the turbulent structures.

The wide range of scales present in turbulent flows is characterized by very strong fluctuations. However, for a great number of engineering cases of interest, the description of an average flow is sufficient. This gives rise to the RANS simulations. That approach does not need to use such mesh refinement, enabling it for broader applications.

A summary of the theories of Richardson and Kolmogorov can be synthesized in Fig. 2.3, which illustrates the usual energy spectrum profile found for turbulence, together with the ideas developed by each author. The graph shows the energy

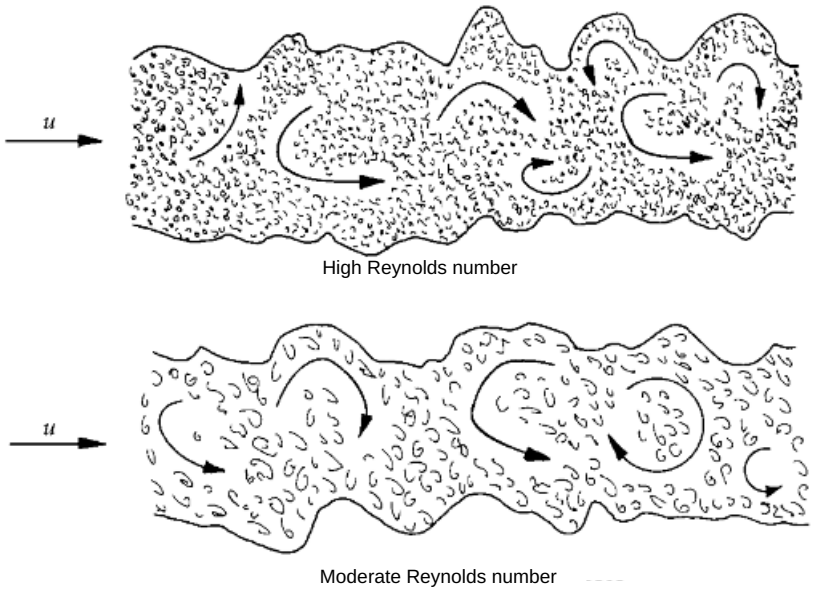



Figure 2.2: Influence of the Reynolds number on the Kolmogorov length scale, DAVIDSON [1].

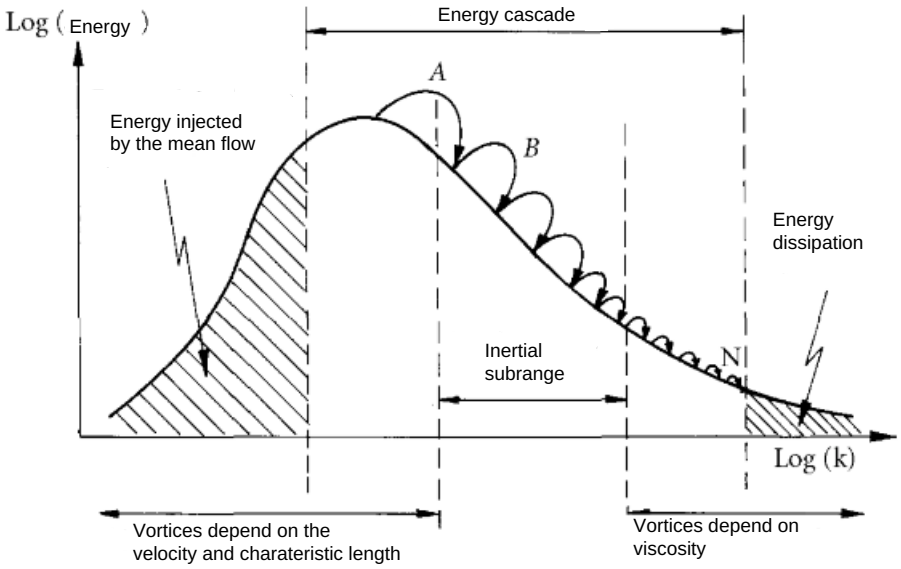



Figure 2.3: Energy spectrum, DAVIDSON [1].

contained in each vortex as a function of its wave number $k$, which corresponds to the inverse of characteristic length.

### 2.1.2 Diffusivity

Diffusivity is probably the most important characteristic of the turbulent flows for industrial applications, WILCOX [21]. Compared to laminar ones, turbulent flows present a greater ability to transport and mix properties of a fluid, what makes them very attractive in processes that require rapid mixing, such as air-fuel in the vehicle engine and reagents in chemical reactors, POPE [25]. The existence of the random fluctuations in the velocity field intensifies the diffusion exchanges in the fluid, while in the laminar regime they are given only by molecular means, KUNDU *et al.* [2]. An example that illustrates well this difference is the profile of the boundary layer in each of these flows, Fig. 2.4.

Due to the non-slip condition imposed by the wall, the fluid decelerates from its free velocity to the velocity of the wall in a short space given by the height of the boundary layer. This region, therefore, is characterized by high velocity gradients that give rise to surface drag. The turbulent flows, being more diffusive, promote a greater exchange of linear momentum between the outermost fluid and the innermost to the boundary layer, thus explaining why its profile is steeper near the wall compared to the laminar flows.

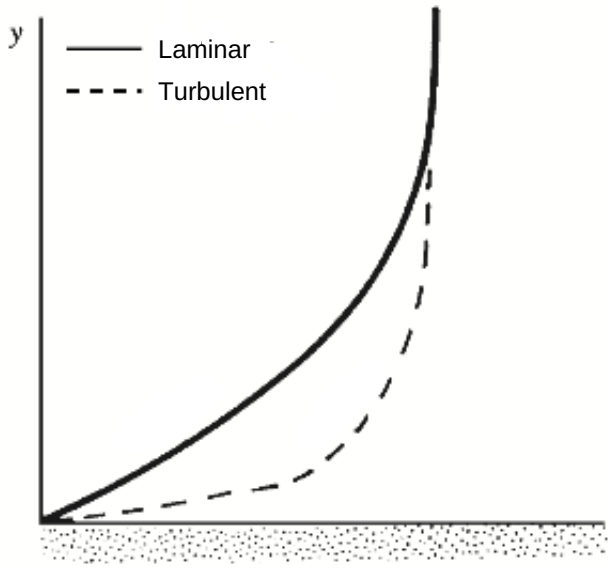



Figure 2.4: Profile of the boundary layer in a laminar flow and average profile in a turbulent flow, KUNDU *et al.* [2].

As a consequence of the difference in velocity gradient near the surface between a turbulent and laminar flow, the diffusivity caused by the first on the surface is greater than that of the second,

$$\mu \boldsymbol{\nabla}^2 \langle \mathbf{v} \rangle|_{\text{turb}} > \mu \boldsymbol{\nabla}^2 \langle \mathbf{v} \rangle|_{\text{lam}}. \tag{2.8}$$

Although it seems at at first time that superior drag in the turbulent flows is a disadvantage, there are actually applications in the study of aerodynamics that benefit from this fact. When the boundary layer that develops in a body moving in a fluid separates, a region of low pressure forms downstream. The imbalance of the pressure field around the body creates a force contrary to its displacement that can overlap in magnitude to the drag due to the friction of the fluid with the surface. Given the higher concentration of linear momentum within the turbulent boundary layer, deceleration of the fluid due to adverse surface pressure gradients until the start of the separation is reduced, which, for an airplane, means greater possible angles of attack before loss of support. Another example is the characteristic cavities of golf balls that immediately create a turbulent flow around them, delaying the separation of the boundary layer caused by the adverse pressure gradient, what allow a greater reach for the shot.

### 2.1.3 Three-dimensionality

The three-dimensionality of the turbulence is closely related to the flow vorticity. More specifically, with the *vortex stretching* (I term of the Eq. 2.9) mechanism present in the vorticity transport equation,

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + (\mathbf{v} \cdot \boldsymbol{\nabla})\boldsymbol{\omega} = \underbrace{(\boldsymbol{\omega} \cdot \boldsymbol{\nabla})\mathbf{v}}_{\text{I}} + \nu \boldsymbol{\nabla}^2 \boldsymbol{\omega}. \tag{2.9}$$

The vortex stretching is a vorticity creation mechanism related with the rotation and stretching of the material filaments of the fluid. This mechanism is very important to turbulence flows because it is through it that energy is extracted from the mean flow for the formation of vortices in the energy cascade, DAVIDSON [1].

If the flow is bidimensional, $\mathbf{v} = v_1\hat{\mathbf{e}}_1 + v_2\hat{\mathbf{e}}_2$, the vorticity is orthogonal to the velocity field, $\boldsymbol{\omega} = \omega\hat{\mathbf{e}}_3$. When this happens, $(\boldsymbol{\omega} \cdot \boldsymbol{\nabla})\mathbf{v} = \mathbf{0}$. So, for bidimensional flows, there is no vortex stretching. As this phenomenon is of paramount importance for the energy cascade, it is concluded that two-dimensional flows can not become turbulent.

### 2.1.4 Dissipation

Turbulent flows are always dissipative. The energy cascade requires a continuous supply of energy for the average flow in order to keep the vortex stretching mechanism in operation. Once this supply has ceased, the turbulence decays rapidly due the transformation of the kinematic energy in thermal energy at the lower scales.

One of the simpler, and also older, problems formulated on turbulence referred

to how fast a turbulent flow dissipates turbulent kinetic energy when left isolated, DAVIDSON [1]. An everyday example would be the time required for the fluctuations induced in the coffee to stir it disappear. Although the problem is easy to understand, physicists and mathematicians have been trying to answer this question for over half a century and have yet to reach consensus on the topic, DAVIDSON [1].

### 2.1.5 Deterministic chaos

A dynamic system, that is, a system that has a certain set of equations that models its temporal evolution is deterministic. This means that, knowing the equations that model it, its boundary conditions and initial conditions, any configuration can be predicted at any desired instant of time. However, one does not always have complete knowledge of the phenomena that circumvent the dynamic system. An imprecision to the extent of certain quantities such as density or viscosity, or small perturbations in the boundary or initial conditions may generate uncertainties in the predictions.

The sensitivity to uncertainties depends on the number of degrees of freedom of the equations that describes the problem. In general, the more nonlinear the greater the dynamic system ability to propagate and amplify such uncertainties, this type of system is known as chaotic. An example of this is the comparison between the simple pendulum and the double pendulum. The simple pendulum has as only initial condition the angle of the pendulum $\theta$, its governing equation is the nonlinear ODE

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\sin(\theta) = 0, \tag{2.10}$$

where $l$ is the arm length and $g$ is the gravity. It is known that small uncertainties added in the initial angle of the pendulum will not profoundly modify the circular trajectory of its end significantly. This means that the single degree of freedom present in the equation is not enough to propagate such uncertainties.

However, the same does not occur with the double pendulum. This is governed

by four non-linear ODEs,

$$\begin{cases} \dfrac{d\theta_1}{dt} = \dfrac{6}{ml^2}\dfrac{p_{\theta 1} - 3\cos(\theta_1 - \theta_2)p_{\theta 2}}{16 - 9\cos^2(\theta_1 - \theta_2)} \\ \\ \dfrac{d\theta_2}{dt} = \dfrac{6}{ml^2}\dfrac{p_{\theta 2} - 8\cos(\theta_2 - \theta_2)p_{\theta 1}}{16 - 9\cos^2(\theta_1 - \theta_2)} \\ \\ \dfrac{dp_{\theta 1}}{dt} = -\dfrac{1}{2}ml^2\Big(\dfrac{d\theta_1}{dt}\dfrac{d\theta_2}{dt}\sin(\theta_1 - \theta_2) + 3\dfrac{g}{l}\sin(\theta_1)\Big) \\ \\ \dfrac{dp_{\theta 2}}{dt} = -\dfrac{1}{2}ml^2\Big(-\dfrac{d\theta_1}{dt}\dfrac{d\theta_2}{dt}\sin(\theta_1 - \theta_2) + \dfrac{g}{l}\sin(\theta_2)\Big) \end{cases}, \tag{2.11}$$

requiring two angles (referring to each arm) as initial conditions. It is observed that this system has a degree of non-linearity much greater than that observed in the simple pendulum. This impacts on the propagation of the uncertainties present in the initial conditions. This means that when two identical double pendulums are analyzed, but with very close but not identical initial conditions, their trajectories will be completely different because the uncertainty that differentiated both will be propagated and amplified by the non-linearities of the system, causing it to generate two completely different answers. An example of the trajectory of a double pendulum can be seen in Fig. 2.5.

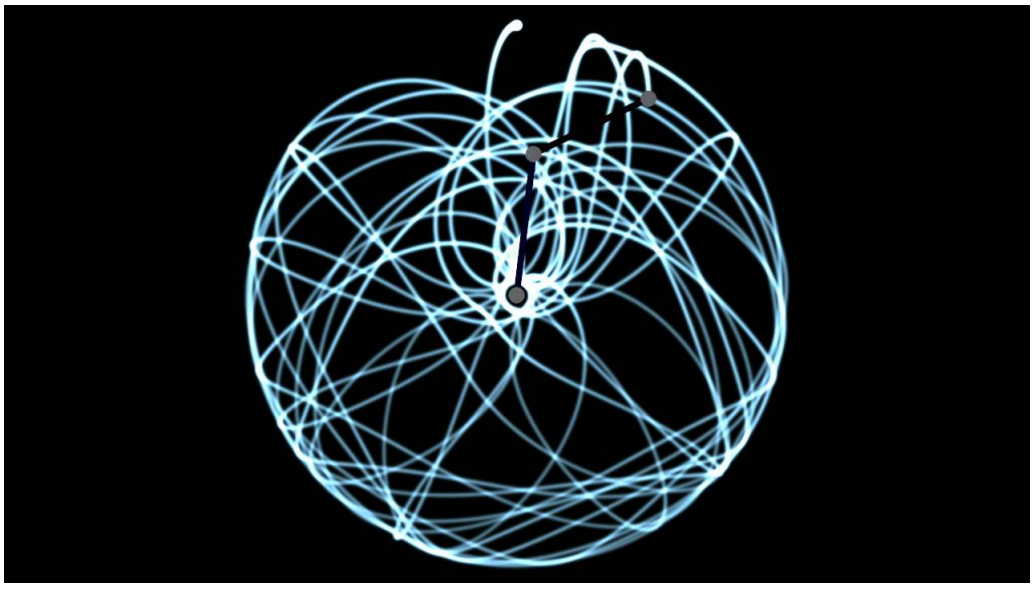


Figure 2.5: Chaotic trajectory of a double pendulum, [3].

A similar behavior can be seen in flows. As has already been said, on the basis of the Reynolds number, the equations that models the flow of a fluid may assume a chaotic behavior. Since the flow of fluids can be understood as a dynamic

system of many degrees of freedom having nonlinear interactions between them, small perturbations such as roughness, impurities and/or external vibrations, can be magnified causing the flow to transit to a turbulent state.

## 2.2 Turbulence modelling

As already mentioned in subsection 2.1.1, to capture all the nuances of a turbulent flow, the mesh should at least be refined on the Kolmogorov scale. What for most applications is not feasible because the Reynolds number is high. This undermines the large-scale application of DNS simulations.

For most engineering applications, the average flow is more than sufficient for engineering projects purposes. As can be seen in Fig. 2.6, the velocity profile obtained through the sample mean of a series of measurements made in a turbulent boundary layer represented well the physical phenomenon.

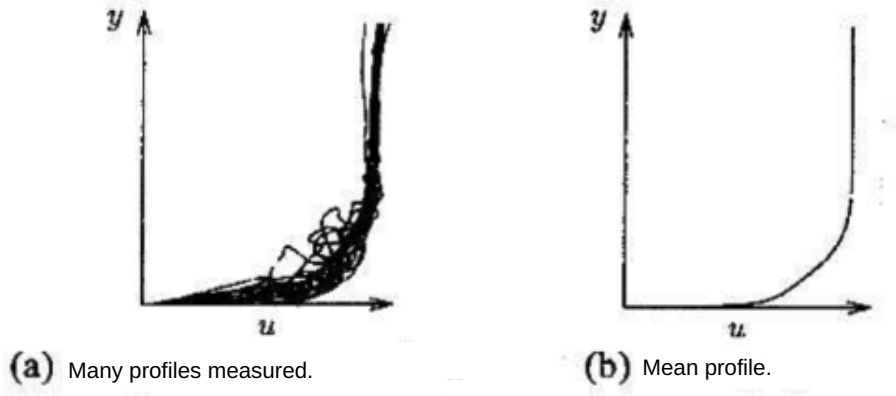



Figure 2.6: Turbulent boundary layer, WILCOX [4].

For this, the Reynolds decomposition is applied. Given any $\phi$ field, this can be decomposed into its mean, $\langle\phi\rangle$, and into a float around the same, $\phi'$,

$$\phi = \langle\phi\rangle + \phi', \tag{2.12}$$

where the mean field is calculated with respect its PDF function, $\rho_\phi$,

$$\langle\phi\rangle = \int_{\Phi_0}^{\Phi_f} \phi\rho_\phi \, \mathrm{d}\Phi, \tag{2.13}$$

in the sample space $\phi \in [\Phi_0, \Phi_f]$.

The average operator has some important properties. The first one is that the average of a given variable PHI is no longer in function of its sample space. Which

is obvious, since, for its achievement, it has been integrated in this space. This implies in the first important property, which says that the mean of an average is the average itself,

$$\langle\langle\phi\rangle\rangle = \int_{\Phi_0}^{\Phi_f} \langle\phi\rangle \rho_\phi \, \mathrm{d}\Phi = \langle\phi\rangle \underbrace{\int_{\Phi_0}^{\Phi_f} \rho_\phi \, \mathrm{d}\Phi}_{=1},$$

$$\langle\langle\phi\rangle\rangle = \langle\phi\rangle. \tag{2.14}$$

The second important property is that this operator is linear. From the linearity of the mean operator, and from the first property, Eq. 2.14, it is proved that the mean of the flotation is zero,

$$\langle\phi'\rangle = \langle\phi - \langle\phi\rangle\rangle = \langle\phi\rangle - \langle\langle\phi\rangle\rangle = \langle\phi\rangle - \langle\phi\rangle,$$

$$\langle\phi'\rangle = 0. \tag{2.15}$$

Another important property for future developments of this operator, concerns the product average of two fields,

$$\langle\phi\alpha\rangle = \Big\langle(\langle\phi\rangle + \phi')(\langle\alpha\rangle + \alpha')\Big\rangle = \Big\langle\langle\phi\rangle\langle\alpha\rangle + \phi'\langle\alpha\rangle + \langle\phi\rangle\alpha' + \phi'\alpha'\Big\rangle,$$

$$\langle\phi\alpha\rangle = \langle\phi\rangle\langle\alpha\rangle + \langle\phi'\rangle\langle\alpha\rangle + \langle\phi\rangle\langle\alpha'\rangle + \langle\phi'\alpha'\rangle,$$

$$\langle\phi\alpha\rangle = \langle\phi\rangle\langle\alpha\rangle + \langle\phi'\alpha'\rangle. \tag{2.16}$$

The last relevant property is also a direct consequence of the linearity of the mean operator and the derived operator. The derivative of the mean is the mean of the derivatives,

$$\Big\langle\frac{\partial\phi}{\partial x_i}\Big\rangle = \frac{\partial\langle\phi\rangle}{\partial x_i}. \tag{2.17}$$

Once the Reynolds decomposition, and all these properties of the average operator, have been established, it is sufficient to apply this theory in the velocity and pressure fields. However, there is still a deadlock, as if calculating these average fields without having access to their respective probability distribution functions. The ergodicity theorem solves this problem, since it postulates a connection between the mean calculated via probability density functions and sample and temporal means.

For a very large number of samples, this theorem guarantees that the sample

mean of a function converges to its statistical mean,

$$\int_{\Phi_0}^{\Phi_f} \phi \rho_\phi \, \mathrm{d}\Phi = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \phi_{(N)}, \tag{2.18}$$

and also ensures that for sufficiently long times, the temporal average will also converge to the statistical mean,

$$\int_{\Phi_0}^{\Phi_f} \phi \rho_\phi \, \mathrm{d}\Phi = \lim_{T\to\infty} \frac{1}{T} \int_{t}^{t+T} \phi \, \mathrm{d}\tau. \tag{2.19}$$

With this tool, it is now possible to extract average velocity and pressure fields from experiments, or even from DNS simulations, provided that a sufficient number of spatial or temporal samples are available to guarantee this convergence between the means. An example of this can be seen in Fig. 2.6.

### 2.2.1 Reynolds averaged Navier-Stokes equations - RANS

For the sake of simplicity, this work will only analyze the flows of incompressible Newtonian fluids, with constant properties and without body forces.

To deal with the average fields, we need the transport equations of these average quantities. This is what the Reynolds averaged Navier-Stokes (RANS) approach. From the continuity equations and linear momentum balance,

$$\boldsymbol{\nabla} \cdot \mathbf{v} = 0, \tag{2.20a}$$

$$\frac{\partial \mathbf{v}}{\partial t} + \boldsymbol{\nabla} \cdot (\mathbf{v} \otimes \mathbf{v}) = -\frac{1}{\rho} \boldsymbol{\nabla} p + \nu \boldsymbol{\nabla}^2 \mathbf{v}, \tag{2.20b}$$

the average operator, defined in the Eq. 2.13, is applied to obtain the average equations. Using the properties listed in the previous session, you get

$$\boldsymbol{\nabla} \cdot \langle \mathbf{v} \rangle = 0, \tag{2.21a}$$

$$\frac{\partial \langle \mathbf{v} \rangle}{\partial t} + \boldsymbol{\nabla} \cdot (\langle \mathbf{v} \rangle \otimes \langle \mathbf{v} \rangle) + \boldsymbol{\nabla} \cdot \langle \mathbf{v'} \otimes \mathbf{v'} \rangle = -\frac{1}{\rho} \boldsymbol{\nabla} \langle p \rangle + \nu \boldsymbol{\nabla}^2 \langle \mathbf{v} \rangle. \tag{2.21b}$$

Overall, the demonstration is very immediate. The mean operator linearity and the mean property of a derivative are used, Eq. 2.17. What is interesting is when the operator is applied on the advective term of the linear momentum balance. The nonlinearity of this term produces an extra term for the equation in its average

version, through the property described in Equation 2.16,

$$\Big\langle \boldsymbol{\nabla}\cdot(\mathbf{v}\otimes\mathbf{v})\Big\rangle = \Big\langle \partial_m(v_m v_i)\hat{\mathbf{e}}_i\Big\rangle,$$

$$= \partial_m(\underbrace{\langle v_m v_i\rangle}_{\text{Eq. 2.16.}})\hat{\mathbf{e}}_i,$$

$$= \partial_m(\langle v_m\rangle\langle v_i\rangle + \langle v'_m v'_i\rangle)\hat{\mathbf{e}}_i,$$

$$= \big(\partial_m(\langle v_m\rangle\langle v_i\rangle) + \partial_m\langle v'_m v'_i\rangle\big)\hat{\mathbf{e}}_i,$$

$$= \boldsymbol{\nabla}\cdot(\langle\mathbf{v}\rangle\otimes\langle\mathbf{v}\rangle) + \boldsymbol{\nabla}\cdot\langle\mathbf{v'}\otimes\mathbf{v'}\rangle.$$

This extra term that arises is a tensor that is associated with the product average of the instantaneous velocity field fluctuations. That is, this tensorial entity has to do with the second statistical moment of the velocity field, the covariance of $\mathbf{v}$. Defining this extra term as $\mathbf{R} \equiv -\langle\mathbf{v'}\otimes\mathbf{v'}\rangle$, the RANS equations are

$$\boldsymbol{\nabla}\cdot\langle\mathbf{v}\rangle = 0, \tag{2.22a}$$

$$\frac{\partial\langle\mathbf{v}\rangle}{\partial t} + \boldsymbol{\nabla}\cdot(\langle\mathbf{v}\rangle\otimes\langle\mathbf{v}\rangle) = -\frac{1}{\rho}\boldsymbol{\nabla}\langle p\rangle + \nu\boldsymbol{\nabla}^2\langle\mathbf{v}\rangle + \boldsymbol{\nabla}\cdot\mathbf{R}. \tag{2.22b}$$

Comparing the instantaneous equations with the RANS equations, we can see that the only difference between them is the presence of the divergent of this tensor. Since this entity is a function of velocity fluctuations, it is concluded that the whole effect of the turbulence on the average flow occurs through $\mathbf{R}$. By rewriting the right side of the mean linear momentum balance, we can establish a physical interpretation for this extra term,

$$\frac{\partial\langle\mathbf{v}\rangle}{\partial t} + \boldsymbol{\nabla}\cdot(\langle\mathbf{v}\rangle\otimes\langle\mathbf{v}\rangle) = \boldsymbol{\nabla}\cdot\Big(-\frac{1}{\rho}\langle p\rangle\mathbf{I} + \nu\boldsymbol{\nabla}\langle\mathbf{v}\rangle + \mathbf{R}\Big),$$

where $\mathbf{I}$ is the identity tensor. On the right side, inside the divergent, three portions of stress acting on the fluid are observed. The first two are related to pressure and molecular viscosity, analogous to their instantaneous formulation. The third portion, which is the extra tensor itself, corresponds to a extra stress, called the turbulent stress tensor or Reynolds stress tensor, that acts on the fluid. This term acts along with the term associated with molecular viscosity, increasing the diffusivity.

### 2.2.2 The closure problem

With the appearance of the Reynolds tensor, the system of equations now becomes indeterminate since the six components of $\mathbf{R}$ (symmetric tensor) are extra variables.

So we need six more equations to close the problem. A first idea would be to define a transport equation for R, to do so, we compute the transport equation of the turbulent fluctuations by subtracting from the instantaneous equation the mean equation. With this velocity fluctuation transport, the tensor product is applied on both sides of this equation to define an equation for $\mathbf{v'}\otimes\mathbf{v'}$. The transport equation of $\mathbf{R}$ is obtained from the mean of this result. However, the same problem of the average operator acting on the non-linear term occurs in this case, this gives rise to a third-order statistical moment. That is, a new third order tensorial entity, $\langle \mathbf{v'} \otimes \mathbf{v'} \otimes \mathbf{v'} \rangle$ , representing twenty-seven variables.

A new transport equation can be deduced from this term, however, from the advective term of the equation of $\mathbf{R}$, there will be a new larger order entity with even more variables to be determined. This problem suggests that it is impossible to deduce a closure for the system of equations, this configures the problem of closing the RANS equations [26].

The alternative then becomes the modeling of turbulent stresses.

### 2.2.3 RANS modeling

There is a wide range of closure models, each with its own peculiarities. This fact denounces the inexistence of a universal model that can solve the problem of closure. The various models are generally classified by the number of transport equations plus the Reynolds equations that it adds to the problem. The most widely used models in the industry are with two equations.

The two-equation models are based on the BOUSSINESQ [12] hypothesis, which are also called closure viscosity models. This is because the Reynolds tensor is defined in a manner analogous to the stress tensor for Newtonian fluids, through a linear relationship between the anisotropic part of $\mathbf{R}$ and the mean strain rate of the flow defined by a turbulent viscosity $\nu_T$,

$$\mathbf{R} = 2\nu_T\langle\mathbf{D}\rangle - \frac{2}{3}\kappa\mathbf{I}, \tag{2.23}$$

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

Unlike molecular viscosity which is a property of the fluid, the turbulent viscosity depends on the flow pattern itself, WILCOX [21].

With this, the average balance of linear momentum is,

$$\frac{\partial\langle\mathbf{v}\rangle}{\partial t} + \boldsymbol{\nabla}\cdot(\langle\mathbf{v}\rangle\otimes\langle\mathbf{v}\rangle) = -\frac{1}{\rho}\boldsymbol{\nabla}\Big(\langle p\rangle + \frac{2}{3}\rho\kappa\Big) + \boldsymbol{\nabla}\cdot\big((\nu+\nu_T)\boldsymbol{\nabla}\langle\mathbf{v}\rangle\big),$$

where $\nu_T \equiv \mu_T/\rho$. Generally an equivalent pressure is defined, adding in this way

the average pressure with the term as a function of the turbulent kinetic energy,

$$p^* \equiv \langle p \rangle + \frac{2}{3}\rho\kappa. \tag{2.24}$$

With the application of this hypothesis, the problem of closure has not yet been solved. However, it was reduced from six extra variables to only one, which is the turbulent viscosity,

$$\frac{\partial\langle \mathbf{v} \rangle}{\partial t} + \boldsymbol{\nabla} \cdot (\langle \mathbf{v} \rangle \otimes \langle \mathbf{v} \rangle) = -\frac{1}{\rho}\boldsymbol{\nabla} p^* + \boldsymbol{\nabla} \cdot \big((\nu + \nu_T)\boldsymbol{\nabla}\langle \mathbf{v} \rangle\big). \tag{2.25}$$

In a first analysis, this hypothesis seems quite reasonable. For it implies an increase in the effective viscosity of the medium flow. This is in line with the discursive characteristics of the turbulent flows as high diffusivity and dissipation.

However, as has been said, there is no universal model for turbulence. in this case, the models that use the Boussinesq hypothesis have certain limitations. According to POPE [19], an explicit description of the turbulent stresses as a function of mean strain rate is not correct. This happens because the Reynolds stress tensor does not change instantaneously with the mean strain rate, there is a certain response delay that has been experimentally detected. The second criticism made by POPE [19] is that, even if there were an explicit relation between $\mathbf{R}$ and $\langle \mathbf{D} \rangle$, this relation would not be linear. Even for simple shear flows, it is observed that the linear hypothesis misses significantly to represent the Reynolds stress tensor. It was concluded that the linearity is not enough to capture all the anisotropy of $\mathbf{R}$.

DESCHAMPS [27] lists the cases where the Boussinesq hypothesis fails:

- Flow with significant curvature of the streamlines;
- Flow with adverse pressure gradient;
- Flow with separation regions;
- Jet flows;
- Flow with body forces.

Despite these limitations, the results obtained by RANS models using the Boussinesq hypothesis are sufficiently satisfactory, and computationally cheap, for a wide variety of applications, SCHMITT [28].

The next step of closure is to model the turbulent viscosity. For this, a dimensional analysis is made,

$$[\nu_T] = \left[\frac{L^2}{T}\right] = [L] \times \left[\frac{L}{T}\right],$$
$$\nu_T = c\mathcal{L}\mathcal{U}. \tag{2.26}$$

The turbulent viscosity is written as a function of a dimensionless $c$, a characteristic length $\mathcal{L}$ and a characteristic velocity $\mathcal{U}$.

Next, the closure model that will be used in this work will be described. The characteristic length and velocity are described to describe the turbulent viscosity.

### $\kappa$-$\epsilon$ RANS model

The $\kappa$-$\epsilon$ model uses as characteristic velocity the square root of the turbulent kinetic energy [11],

$$\mathcal{U} \equiv \kappa^{1/2}. \tag{2.27}$$

Due to this, a transport equation for $\kappa$ is required. The turbulent kinetic energy can be calculated from the average of the scalar product of the velocity field fluctuation with itself,

$$\kappa = \frac{1}{2}\langle \mathbf{v}' \cdot \mathbf{v}' \rangle.$$

Similarly, to find the $\kappa$ equation, the same methodology is applied in the equation of transport of the velocity fluctuation. This equation is obtained from the instantaneous linear momentum discounted from the average balance,

$$\frac{\partial \mathbf{v}'}{\partial t} + \boldsymbol{\nabla} \cdot (\mathbf{v} \otimes \mathbf{v} - \langle \mathbf{v} \rangle \otimes \langle \mathbf{v} \rangle) = -\frac{1}{\rho}\boldsymbol{\nabla} p' + \nu \boldsymbol{\nabla}^2 \mathbf{v}' - \boldsymbol{\nabla} \cdot \mathbf{R}.$$

Similar to what was done earlier, the inner product is applied on both sides of the floating linear momentum transport equation with the floating velocity field itself, and then the average operator is applied on both sides,

$$\Big\langle \frac{\partial \mathbf{v}'}{\partial t} \cdot \mathbf{v}' + \boldsymbol{\nabla} \cdot (\mathbf{v} \otimes \mathbf{v} - \langle \mathbf{v} \rangle \otimes \langle \mathbf{v} \rangle) \cdot \mathbf{v}' \Big\rangle = \Big\langle -\frac{1}{\rho}\boldsymbol{\nabla} p' \cdot \mathbf{v}' + \nu \boldsymbol{\nabla}^2 \mathbf{v}' \cdot \mathbf{v}' - \boldsymbol{\nabla} \cdot \mathbf{R} \cdot \mathbf{v}' \Big\rangle,$$

$$\frac{\partial \kappa}{\partial t} + \boldsymbol{\nabla} \cdot (\langle \mathbf{v} \rangle \kappa) = \mathcal{P} + \boldsymbol{\nabla} \cdot \Big( \nu \boldsymbol{\nabla} \kappa - \frac{1}{2}\langle \mathbf{v}' \otimes \mathbf{v}' \cdot \mathbf{v}' \rangle - \frac{1}{\rho}\langle p' \mathbf{v}' \rangle \Big) - \epsilon, \tag{2.28}$$

where $\mathcal{P}$ is the turbulent production,

$$\mathcal{P} \equiv 2\nu_T \langle \mathbf{D} \rangle : \boldsymbol{\nabla} \langle \mathbf{v} \rangle, \tag{2.29}$$

and $\epsilon$ is the turbulent dissipation,

$$\epsilon \equiv \nu \langle \boldsymbol{\nabla} \mathbf{v}' : \boldsymbol{\nabla} \mathbf{v}'^T \rangle. \tag{2.30}$$

The turbulent production $\mathcal{P}$ is the energy rate that leaves the medium flow and turns into turbulent kinetic energy. As previously stated in the text, turbulent

dissipation $\epsilon$ is the rate of conversion of turbulent kinetic energy into thermal energy at the smaller scales in the energy cascade. Although the classical terms used are production and dissipation, both terms represent rates of energy transformation. The diffusive term has three parcels: a portion of molecular diffusion $\boldsymbol{\nabla} \cdot (\nu \boldsymbol{\nabla} \kappa)$, a term of diffusion through a turbulent transport $\boldsymbol{\nabla} \cdot (-1/2 \langle \mathbf{v}' \otimes \mathbf{v}' \cdot \mathbf{v}' \rangle)$ and a term of diffusion by pressure $\boldsymbol{\nabla} \cdot (-1/\rho \langle p' \mathbf{v}' \rangle)$. These last two terms need modeling since they are written as functions of velocity and pressure field fluctuations. Those terms were modeled with a gradient-diffusion approach [11],

$$\frac{1}{2} \langle \mathbf{v}' \otimes \mathbf{v}' \cdot \mathbf{v}' \rangle - \frac{1}{\rho} \langle p' \mathbf{v}' \rangle \approx - \frac{\nu_T}{\sigma_\kappa} \boldsymbol{\nabla} \kappa,$$

where $\sigma_\kappa$ dimensionless constant.

The turbulent dissipation, as well as the latter two terms, is also written in terms of the fluctuation of the velocity field. However, unlike these portions, $\epsilon$ will not be modeled explicitly. Instead, along with the turbulent kinetic energy, it will be used to define a characteristic length [11],

$$[\mathcal{L}] = \frac{[\kappa]^{3/2}}{[\epsilon]}. \tag{2.31}$$

Having defined velocity and characteristic length, one can then define the turbulent viscosity (Eq. 2.26),

$$\nu_T = C_\mu \frac{\kappa^2}{\epsilon}. \tag{2.32}$$

In order to close the problem, there is now only the transport equation of turbulent dissipation. A transport equation can be derived just as it was done for the turbulent kinetic energy, but this would be so laborious and complex that it becomes more advantageous to model the equation entirely. It is then written in a manner analogous to the $\kappa$ equation, but with other constants to fit the model. The extra transport equations are then,

$$\frac{\partial \kappa}{\partial t} + \boldsymbol{\nabla} \cdot (\langle \mathbf{v} \rangle \kappa) = \mathcal{P} + \boldsymbol{\nabla} \cdot \left[ \left( \nu + \frac{\nu_T}{\sigma_\kappa} \right) \boldsymbol{\nabla} \kappa \right] - \epsilon, \tag{2.33a}$$

$$\frac{\partial \epsilon}{\partial t} + \boldsymbol{\nabla} \cdot (\langle \mathbf{v} \rangle \epsilon) = C_{\epsilon 1} \frac{\epsilon}{\kappa} \mathcal{P} - C_{\epsilon 2} \frac{\epsilon^2}{\kappa} + \boldsymbol{\nabla} \cdot \left[ \left( \nu + \frac{\nu_T}{\sigma_\epsilon} \right) \boldsymbol{\nabla} \epsilon \right]. \tag{2.33b}$$

The parameters are described in Tab. (2.1).

As in the turbulent kinetic energy equation, the turbulent dissipation equation has a production and dissipation term, where it is assumed that these are proportional to the $\kappa/\epsilon$ ratio. The parameters $C_{\epsilon 1}$ and $C_{\epsilon 2}$ are defined to adjust such proportionality. What justifies such a hypothesis is the fact that the production of

Table 2.1: $\kappa$-$\epsilon$ model's dimensionless parameters [11].

| $C_\mu$ | $\sigma_\kappa$ | $\sigma_\epsilon$ | $C_{\epsilon 1}$ | $C_{\epsilon 2}$ |
|---|---|---|---|---|
| 0.09 | 1.00 | 1.30 | 1.44 | 1.92 |

turbulent kinetic energy is correlated with dissipation, where one is high, the other is also [29].

The main deficiency of this model is the description of the flow near the solid surfaces. In these regions, viscous effects predominate over turbulent effects, and the $\kappa$-$\epsilon$ model does not contemplate this phenomenon. This is mainly because of the uncertainties of the transport equation of the turbulent dissipation which is excessively modeled. There are two ways to circumvent this problem, the first is to change the $\epsilon$ transport equation by introducing damping functions as a function of the local Reynolds number to model the effect of viscosity in these regions. The second option is through the wall functions, which are explicit solutions of the velocity field in these regions through the Law of the Wall.

Despite the shortcomings of the Boussinesq hypothesis and the turbulent dissipation transport equation cited above, the $\kappa$-$\epsilon$ model is the most widely used in industry. This is justified by its relative success in several problems of interest, and for being the most validated literature.

In this work we will use the second approach, which uses the wall laws to better represent the solution near the walls.

### Wall functions

In close proximity to the solid surfaces, the viscous effects are relevant, this can be noticed through a simple dimensional analysis. This region, which defines the boundary layer, to be characterized in a CFD simulation, requires a high mesh refining. Wall functions are intended to use wall laws to directly calculate average velocity and velocity fields. This is important because the turbulence model has difficulties in describing this region of the flow.

In this region, the characteristic quantities are the shear stress, due to the proximity to the wall, and the kinematic viscosity due to the importance of the viscous effects. With the shear stress, a characteristic velocity is defined, the friction velocity,

$$v_\tau \equiv \sqrt{\frac{||\tau_w||}{\rho}}. \tag{2.34}$$

With this characteristic velocity and viscosity, the distance from the wall, and

the tangential velocity can be made dimensionless as,

$$y^+ \equiv \frac{v_\tau y_\perp}{\nu}, \tag{2.35a}$$

$$v^+ \equiv \frac{v}{v_\tau}. \tag{2.35b}$$

It is known that the region to which the wall law will act is divided into three: A linear region where the viscous forces are practically balanced only by the shear stress. The region of logarithmic law, where the turbulent stresses increase sufficiently to come into equilibrium with the viscosity and the shear stress. And the transition region between the two, the buffer layer. The layers can be seen in Fig. 2.7.

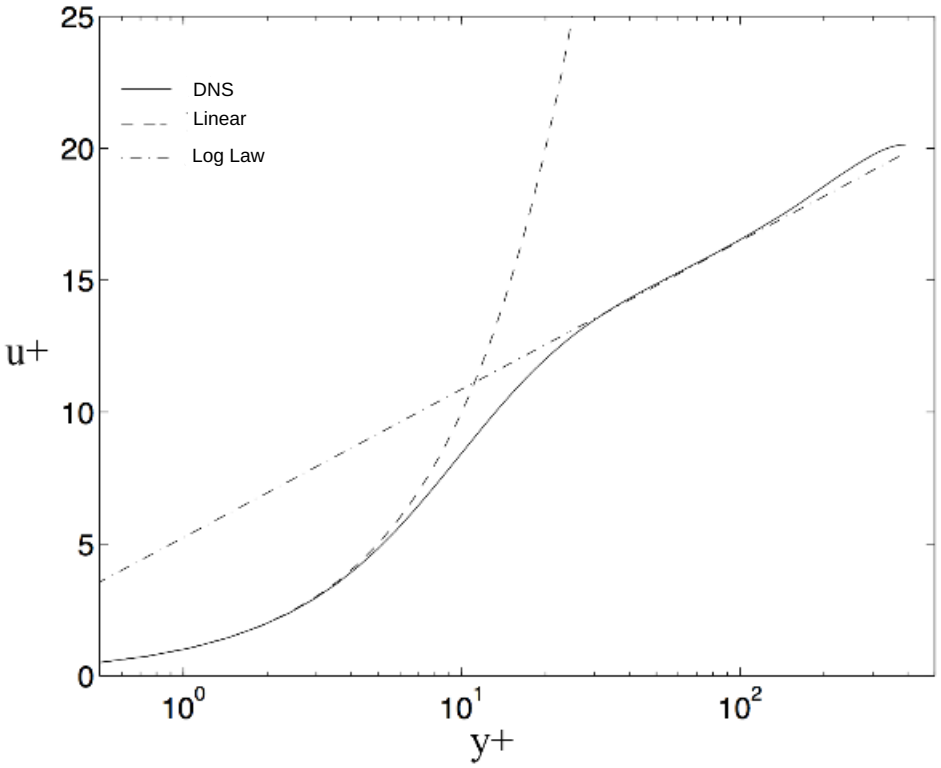



Figure 2.7: Law of the wall, BREDBERG [5].

The equations that govern each of the regions may be

$$v^+ = \begin{cases} y^+ & , y^+ \leq 5 \\ \frac{1}{\kappa}\log\left(Ey^+\right) & , 30 < y^+ < 200 \end{cases} . \tag{2.36}$$

where $\kappa = 0.41$ and $E = 9.8$ for smooth walls.

The wall functions were developed from the wall law. The strategy is to find an expression for the turbulent viscosity from it. With this $\nu_T$ calculated by the

wall function, we can calculate the velocity and pressure field from the average momentum equations modeled by the Boussinesq hypothesis.

To find an expression for the turbulent viscosity, a first order approximation is used between the wall and the first mesh element for the derivative. Next, the definition of friction velocity is used to find a second expression for the shear stress,

$$\tau_w = \rho(\nu + \nu_T)\frac{v}{y} \tag{2.37a}$$

$$\tau_w = \rho v_\tau^2 \tag{2.37b}$$

Equating both expressions we can find the turbulent viscosity in terms of $y^+$,

$$\rho(\nu + \nu_T)\frac{v}{y} = \rho v_\tau^2,$$

$$\nu_T = \frac{v_\tau^2}{v} y - \nu,$$

$$\nu_T = \frac{1}{\left(\frac{v}{v_\tau}\right)} \frac{v_\tau y}{\nu}\nu - \nu,$$

$$\nu_T = \nu\left(\frac{y^+}{v^+} - 1\right),$$

$$\nu_T = \nu\left(\frac{y^+\kappa}{\log\left(Ey^+\right)} - 1\right). \tag{2.38}$$

In order to be able to calculate the turbulent viscosity, it is necessary to compute the $y^+$. For this, of the logarithmic law,

$$\frac{v}{v_\tau} = \frac{1}{\kappa}\log\left(Ey^+\right),$$

$$v_\tau = \frac{v\kappa}{\log\left(Ey^+\right)},$$

$$v_\tau \frac{y}{\nu} = \frac{v\kappa}{\log\left(Ey^+\right)}\frac{y}{\nu},$$

$$y^+ \log\left(Ey^+\right) - \frac{\kappa v y}{\nu} = 0.$$

The desired dimensionless distance is the roots of the function

$$f(y^+) = y^+ \log\left(Ey^+\right) - \frac{\kappa v y}{\nu}. \tag{2.39}$$

Which can be calculated by Newton's method for example.

This is the so-called standard wall function. It has the limitation of not describing cases out of equilibrium, ie, cases where there is a pressure gradient large enough to peel off the boundary layer (zero shear stress). To circumvent this problem, a new characteristic velocity is proposed using turbulent kinetic energy, and no more $\tau_w$ [30],

$$u^* \equiv C_\mu^{1/4}\sqrt{\kappa}, \tag{2.40}$$

where $\kappa$ is calculated from a law of the wall. The other details are similar to the previous ones, just changing the characteristic velocities.

# Chapter 3

# Neural networks

Machine learning (ML) is a computation and artificial intelligence branch. The main objective of this field is the modelling of systems from some given database solely. The machine is able to learn from this available data, understanding patterns, without the necessity of previous implementations.

There are many ML techniques, e.g.: supported vector machine (SVM), neural networks (NN), genetic algorithm, decision trees and random forests. Each one of them can be used for different applications.

In this work, NN was chosen for the flexibility on building their structure and the promising results previously obtained [18]. The theory of neural networks will be discussed in this chapter.

## 3.1 Artificial neuron

In order to understand the concept of a artificial neuron, it is important to first understand the basic functionally of the biological brain, and how it processes data. The brain is a powerful and complex system that is capable to deal with a massive amount of data, that can be noisy and inconsistent, but still capable to produce right answers [31].

Comparing with computation science, the brain could be interpreted as a massive parallel processing performed by approximated 85 billions processors. Those processors are the nervous system cell called neurons [31]. In Fig. (3.1), a neuron cell structure is shown.

- **Cell body**: That's the most important part of the neuron, where all the input signals that comes from other neurons are processed. It is where all the essential parts of a cell are located, and the place where all the metabolic processes. This part is also called 'Sum', because it sums all the signals that comes from other neurons, as has already been said, and creates a new one

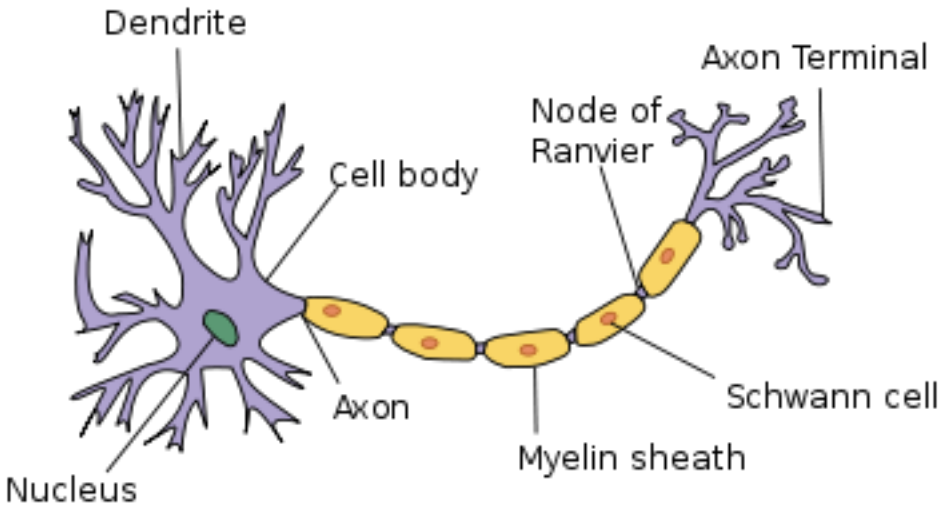



Figure 3.1: Biological neuron basic structure.[6]

that will be transmitted by the axon. This concept of sum of signals is very important for the artificial neuron concept.

- **Axon**: Similar to a energy cable, it transmits the electrical signal generated in the cell body by the metabolic processes for other cells, like another neuron. It's covered by a insulating layer called Myelin, what speeds up the signal streaming.

- **Dendrite**: Are the extensions that leave the cell body, and which receive signals from other neurons.

- **Synapse**: It's the connection between the axon and another neuron's dendrites.

The neurons transmit information by electrical signals that are received by the dendrites. That reception occurs when these cell body dendrites are connected with others neurons by their synapses, what is called synaptic connection. This connection is possible because of the existence of chemicals between the dendrites and the synapses called neurotransmitters [32].

These described interactions occur in all the $85 \times 10^9$ neurons in the human brain. This huge amount of synaptic connections defines the powerful parallel process told before, and it is constantly changing. Providing to the brain the power to recognize patterns, learning, solve complex problems and for make predictions [31].

Based on this basic biological neuron conception, MCCULLOCH e PITTS [33] proposed the fist mathematical description of a artificial neuron, whose characteristics are listed below:

- **Inputs** (in): Simulating the multiple electrical signals that come from other

neurons by the dendrites, the artificial neuron receives the data that comes from others connected to it.

- **Weights** (w): Each data that is received by the artificial neuron is weighted. It simulates the synapses connections, those with higher weights are more important then the others.

- **Activation function** ($\varphi$): It represents the metabolic process that happens in the Cell body, where a new signal is created from the sum of all the weighted signals arriving in the neuron. MCCULLOCH e PITTS [33] first presented that function as a binary function, but later other mathematical expressions were proposed, as the bias term. Bias is a number summed to the sum of all weighted signals that arrives to the neuron. These updates improved the performance of the method.

- **Output** (out): Represents the signal transmitted by the Axon. It is the result of the application of the activation function on the weighted sum of all the signals that arrive in the neuron plus the bias.

Therefore, the $j^{\text{th}}$ neuron's output, of the $m^{\text{th}}$ layer (the neurons are presented in the network organized by layers connected to each other, this architecture will be better explained in the next session), ($out_j^{(m)}$) is the application of the activation function $\varphi$ in the weighted sum of signals that outputs from each one of the back neuron's layer $n$,

$$out_j^{(m)} = \varphi(in_j^{(m)}) = \varphi\left(\sum_{i=1}^{N} w_i^{(m)} out_i^{(n)} + b_j^{(m)}\right). \tag{3.1}$$

As shown in Fig. (3.2).

The most popular activation functions are listed below:

- **Limiar function**:

$$\varphi(in_j^{(m)}) = \begin{cases} 1, \, in_j^{(m)} \geq 1 \\ 0, \, in_j^{(m)} < 1 \end{cases} \tag{3.2}$$

- **Linear function**:

$$\varphi(in_j^{(m)}) = k * in_j^{(m)} \tag{3.3}$$

- **Sigmoid function**: Or the **Logistic function**, is a continous smooth function witch ranges between 0 and 1.

$$\varphi(in_j^{(m)}) = \frac{1}{1 + \exp\left(-in_j^{(m)}\right)} \tag{3.4}$$

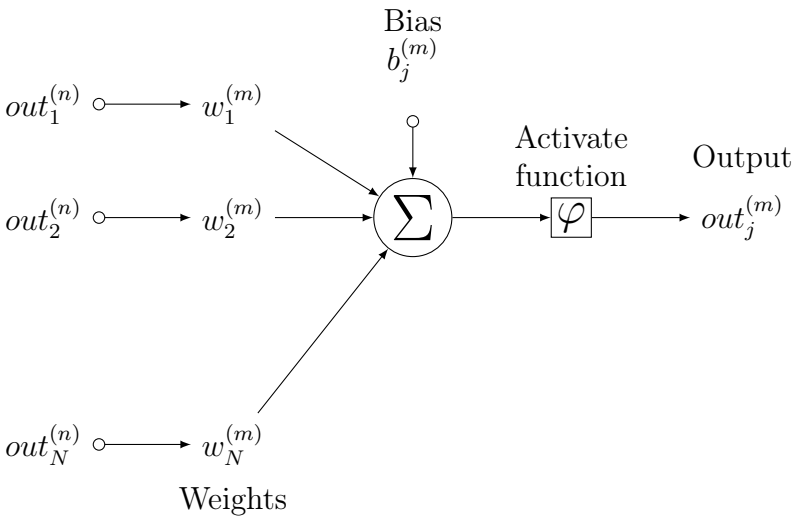



Figure 3.2: $j^{\text{th}}$ neuron of the $m^{\text{th}}$ layer.

- **Hyperbolic tangent function**: Its similar to the Sigmoid function, but it ranges between $-1$ and 1.

$$\varphi(in_j^{(m)}) = \frac{1 - \exp\left(-in_j^{(m)}\right)}{1 + \exp\left(-in_j^{(m)}\right)} \tag{3.5}$$

The multi-interactivity non-linear activation functions, as Sigmoid and Hyperbolic, from a neuron's layer into another that gives to NN the capability to perform non-linear mapping between the inputs $\mathbf{X}$ and a target $\mathbf{Y}$,

$$\mathbf{Y} = NN(\mathbf{w}, \mathbf{b}; \mathbf{X}). \tag{3.6}$$

CIBENKO [34] demosntrated a Universal Approximation Theorem that guaratees the existence of a NN capable of mapping $\mathbf{X}$ to $\mathbf{Y}$. Another benefit in the application of those non-linear activation functions is the fact that they are diferenciable, and their derivates are easy to be calculated, what helps in the NN tranning process(backward).

## 3.2 Neural network architecture - Feedforward NN

Once the concept of artificial neuron was established in the previous session, it will be discussed how they connect to each other in order to form the neural networks properly said.

A typical NN topology presents neuron layers distributions. These layers represent a collection of neurons that are connected with the previous neuron's layer, and are also connected with the layer ahead. The first layer is called input layer, the

last one is the output layer, and all the others in the middle are the hidden layers. Neural networks whose outputs from each layer do not connect with previous layers, that is, only connect to the layers forward, are called feedforward networks. There are other types of architectures, e.g., the feedback neural nets, but these will not be discussed in the present work because they are not part of the scope of the work. A feedforward NN that presents non-linear neuron (with a non-linear activation function) in their hidden layer is also called a multi layer perceptron (MLP). In Fig. (3.3) there is a MLP example with four inputs, one output and two hidden layers with five neurons each. Because of its feedforward structure, the data flows from the input layer to the output layer, passing through the hidden layers.

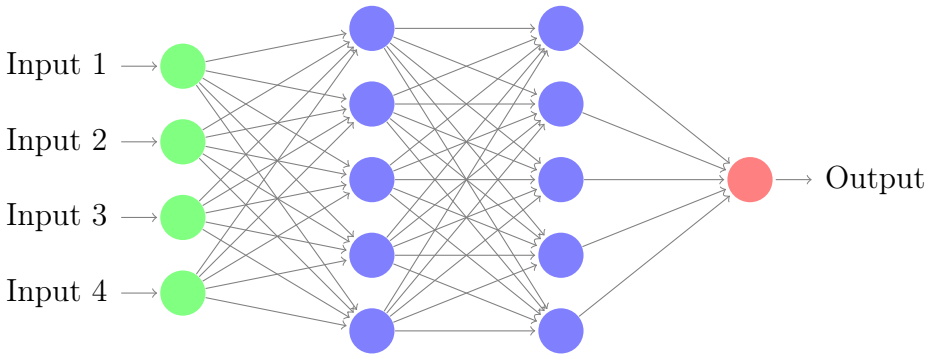



Figure 3.3: MLP NN example.

Therefore, what constitutes a MLP architecture is the number of neurons in each layer, the number of hidden layers and the activation functions. That topology will depend on the project, if the problem has a strong non-linearity, then more neurons and more hidden layers with nonlinear activation functions, e.g., sigmoid or hyperbolic tangent, will probably be needed to better represent it. On the other hand, an excess of neurons in the network can cause overfitting, that is, the network loses generalization capacity. The opposite also occurs, if a small number of neurons is used the network loses abstraction capacity and, therefore, cannot capture all the patterns present in the data used in training stage. In this case, it is said that the problem is underfitted by the NN.

## 3.3 Training a NN - Backpropagation

Usually, the NN weights and bias are initialized by random numbers between $-1$ and 1. So, there will be an error ($E$) between the regression NN results ($\mathbf{Y}$) and the target ($\tilde{\mathbf{Y}}$). That error is a function of all the weights and bias of the NN,

$$E = E(\mathbf{w}, \mathbf{b}).$$

Training a NN is the process of finding the weights and bias that minimizes $E$. The algorithm that does that is the backpropagation.

Defining the error function as

$$E = \sum_{b \in L_1} \frac{1}{2}(\tilde{y}_b - y_b)^2. \tag{3.7}$$

Given a NN backward enumerated, the output layer is the first and the hidden layer before the input layer is the $N^{\text{th}}$, Fig. (3.4). The derivate of the square error function $E = E(\mathbf{w}, \mathbf{b})$ with respect each one of neuron's weights, and bias, from every layer are calculated, $\frac{\partial E}{\partial w_{tv}^{(m)}}$ and $\frac{\partial E}{\partial b_v^{(m)}}$, respectively. Those derivate will be required to update the weights and bias, minimizing the error function by some optimization technique, e.g., gradient descent.

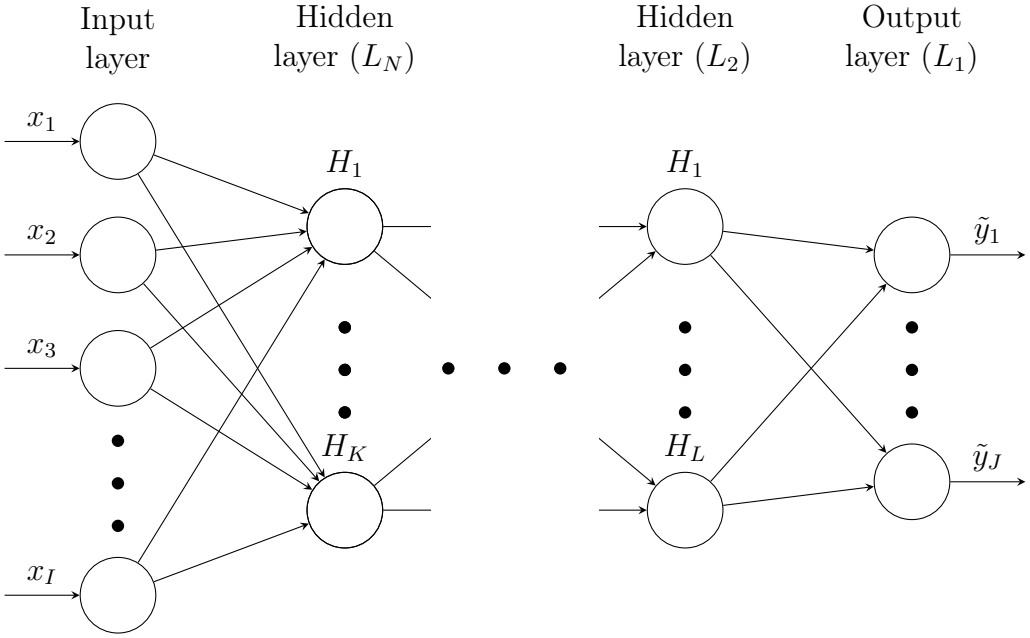



Figure 3.4: Neural net topology example.

The first layer, the output one, will be calculated. Using the chain rule,

$$\frac{\partial E}{\partial w_{lj}^{(1)}} = \frac{\partial in_j^{(1)}}{\partial w_{lj}^{(1)}} \frac{\partial out_j^{(1)}}{\partial in_j^{(1)}} \frac{\partial E}{\partial out_j^{(1)}},$$

and applying the $in_j^{(1)}$ and $out_j^{(1)}$ definitions, Eq. (3.1). And Eq. (3.7),

$$\frac{\partial E}{\partial w_{lj}^{(1)}} = \frac{\partial}{\partial w_{lj}^{(1)}} \left( \sum_{a \in L_1} w_{aj}^{(1)} out_a^{(2)} + b_j^{(1)} \right) \frac{\partial \varphi(in_j^{(1)})}{\partial in_j^{(1)}} \frac{\partial}{\partial out_j^{(1)}} \left( \sum_{b \in L_1} \frac{1}{2}(\tilde{y}_b - y_b)^2 \right),$$

$$\frac{\partial E}{\partial w_{lj}^{(1)}} = out_l^{(2)} \varphi'(in_j^{(1)}) \frac{\partial}{\partial \tilde{y}_j} \left( \sum_{b \in L_1} \frac{1}{2} (\tilde{y}_b - y_b)^2 \right),$$

$$\frac{\partial E}{\partial w_{lj}^{(1)}} = out_l^{(2)} \varphi'(in_j^{(1)}) (\tilde{y}_j - y_j).$$

A delta function related to the first layer is defined as,

$$\delta_j^{(1)} \equiv \varphi'(in_j^1)(\tilde{y}_j - y_j), \tag{3.8}$$

and the with respect the weights of the first layer is

$$\frac{\partial E}{\partial w_{lj}^{(1)}} = out_l^{(2)} \delta_j^{(1)}. \tag{3.9}$$

The same procedure done in the first layer is done in the second one,

$$\frac{\partial E}{\partial w_{ml}^{(2)}} = \frac{\partial in_l^{(2)}}{\partial w_{ml}^{(2)}} \frac{\partial out_l^{(2)}}{\partial in_l^{(2)}} \frac{\partial E}{\partial out_l^{(2)}},$$

$$\frac{\partial E}{\partial w_{ml}^{(2)}} = \frac{\partial}{\partial w_{ml}^{(2)}} \left( \sum_{a \in L_2} w_{al}^{(2)} out_a^{(3)} + b_l^{(2)} \right) \frac{\partial \varphi(in_l^{(2)})}{\partial in_l^{(2)}} \frac{\partial}{\partial out_l^{(2)}} \left( \sum_{b \in L_1} \frac{1}{2} (\tilde{y}_b - y_b)^2 \right),$$

$$\frac{\partial E}{\partial w_{ml}^{(2)}} = out_m^{(3)} \varphi'(in_l^{(2)}) \frac{\partial}{\partial out_l^{(2)}} \left( \sum_{b \in L_1} \frac{1}{2} (\tilde{y}_b - y_b)^2 \right),$$

$$\frac{\partial E}{\partial w_{ml}^{(2)}} = out_m^{(3)} \varphi'(in_l^{(2)}) \sum_{b \in L_1} \left( (\tilde{y}_b - y_b) \frac{\partial \tilde{y}_b}{\partial out_l^{(2)}} \right).$$

Applying the chain rule in $\frac{\partial \tilde{y}_b}{\partial out_l^{(2)}}$,

$$\frac{\partial E}{\partial w_{ml}^{(2)}} = out_m^{(3)} \varphi'(in_l^{(2)}) \sum_{b \in L_1} \left( (\tilde{y}_b - y_b) \frac{\partial \tilde{y}_b}{\partial in_b^{(1)}} \frac{\partial in_b^{(1)}}{\partial out_l^{(2)}} \right),$$

and using the fact that the predicted $y$ ($\tilde{y}_b$) is equal to the $b^{\text{th}}$ first layer neuron output ($\tilde{y}_b = out_b^{(1)} = \varphi(in_b^{(1)})$),

$$\frac{\partial E}{\partial w_{ml}^{(2)}} = out_m^{(3)} \varphi'(in_l^{(2)}) \sum_{b \in L_1} \left( (\tilde{y}_b - y_b) \frac{\partial \varphi(in_b^{(1)})}{\partial in_b^{(1)}} \frac{\partial in_b^{(1)}}{\partial out_l^{(2)}} \right).$$

Utilizing the definition Eq. (3.1) in $\frac{\partial in_b^{(1)}}{\partial out_l^{(2)}}$,

$$\frac{\partial E}{\partial w_{ml}^{(2)}} = out_m^{(3)} \varphi'(in_l^{(2)}) \sum_{b \in L_1} \left( \underbrace{(\tilde{y}_b - y_b)\varphi'(in_b^{(1)})}_{\text{eq. 3.8.}} w_{lb}^{(1)} \right),$$

$$\frac{\partial E}{\partial w_{ml}^{(2)}} = out_m^{(3)} \varphi'(in_l^{(2)}) \sum_{b \in L_1} \left( \delta_b^{(1)} w_{lb}^{(1)} \right).$$

A delta function related to the second layer is also defined as,

$$\delta_l^{(2)} = \varphi'(in_l^{(2)}) \sum_{b \in L_1} \left( \delta_b^{(1)} w_{lb}^{(1)} \right), \tag{3.10}$$

and the derivate with respect the weights of the second layer is

$$\frac{\partial E}{\partial w_{ml}^{(2)}} = out_m^{(3)} \delta_l^{(2)}. \tag{3.11}$$

To complete this demonstration, again, the same procedure is done in the third layer,

$$\frac{\partial E}{\partial w_{nm}^{(3)}} = \frac{\partial in_m^{(3)}}{\partial w_{nm}^{(3)}} \frac{\partial out_m^{(3)}}{\partial in_m^{(3)}} \frac{\partial E}{\partial out_m^{(3)}},$$

$$\frac{\partial E}{\partial w_{nm}^{(3)}} = \frac{\partial}{\partial w_{nm}^{(3)}} \left( \sum_{a \in L_3} w_{am}^{(3)} out_a^{(4)} + b_m^{(3)} \right) \frac{\partial \varphi(in_m^{(3)})}{\partial in_m^{(3)}} \frac{\partial}{\partial out_m^{(3)}} \left( \sum_{b \in L_1} \frac{1}{2} (\tilde{y}_b - y_b)^2 \right),$$

$$\frac{\partial E}{\partial w_{nm}^{(3)}} = out_n^{(4)} \varphi'(in_m^{(3)}) \frac{\partial}{\partial out_m^{(3)}} \left( \sum_{b \in L_1} \frac{1}{2} (\tilde{y}_b - y_b)^2 \right),$$

$$\frac{\partial E}{\partial w_{nm}^{(3)}} = out_n^{(4)} \varphi'(in_m^{(3)}) \sum_{b \in L_1} \left( (\tilde{y}_b - y_b) \frac{\partial \tilde{y}_b}{\partial out_m^{(3)}} \right),$$

$$\frac{\partial E}{\partial w_{nm}^{(3)}} = out_n^{(4)} \varphi'(in_m^{(3)}) \sum_{b \in L_1} \left( (\tilde{y}_b - y_b) \frac{\partial \tilde{y}_b}{\partial in_b^{(1)}} \frac{\partial in_b^{(1)}}{\partial out_m^{(3)}} \right),$$

$$\frac{\partial E}{\partial w_{nm}^{(3)}} = out_n^{(4)} \varphi'(in_m^{(3)}) \sum_{b \in L_1} \left( (\tilde{y}_b - y_b) \frac{\partial \varphi(in_b^{(1)})}{\partial in_b^{(1)}} \frac{\partial in_b^{(1)}}{\partial out_m^{(3)}} \right),$$

$$\frac{\partial E}{\partial w_{nm}^{(3)}} = out_n^{(4)} \varphi'(in_m^{(3)}) \sum_{b \in L_1} \left( \underbrace{(\tilde{y}_b - y_b)\varphi'(in_b^{(1)})}_{\text{eq. 3.8.}} \frac{\partial in_b^{(1)}}{\partial out_m^{(3)}} \right).$$

Using again the chain rule in $\frac{\partial in_b^{(1)}}{\partial out_m^{(3)}}$,

$$\frac{\partial E}{\partial w_{nm}^{(3)}} = out_n^{(4)} \varphi'(in_m^{(3)}) \sum_{b \in L_1} \left( \delta_b^{(1)} \sum_{c \in L_2} \left( \frac{\partial in_c^{(2)}}{\partial out_m^{(3)}} \frac{\partial out_c^{(2)}}{\partial in_c^{(2)}} \frac{\partial in_b^{(1)}}{\partial out_c^{(2)}} \right) \right),$$

and applying the Eq. (3.1),

$$\frac{\partial E}{\partial w_{nm}^{(3)}} = out_n^{(4)} \varphi'(in_m^{(3)}) \sum_{b \in L_1} \left( \delta_b^{(1)} \sum_{c \in L_2} \left( w_{mc}^{(2)} \frac{\partial \varphi(in_c^{(2)})}{\partial in_c^{(2)}} w_{cb}^{(1)} \right) \right),$$

$$\frac{\partial E}{\partial w_{nm}^{(3)}} = out_n^{(4)} \varphi'(in_m^{(3)}) \sum_{b \in L_1} \left( \delta_b^{(1)} \sum_{c \in L_2} \left( w_{mc}^{(2)} \varphi'(in_c^{(2)}) w_{cb}^{(1)} \right) \right),$$

$$\frac{\partial E}{\partial w_{nm}^{(3)}} = out_n^{(4)} \varphi'(in_m^{(3)}) \sum_{c \in L_2} \left( \underbrace{\varphi'(in_c^{(2)}) \sum_{b \in L_1} \left( \delta_b^{(1)} w_{cb}^{(1)} \right)}_{\text{eq. 3.10.}} w_{mc}^{(2)} \right),$$

$$\frac{\partial E}{\partial w_{nm}^{(3)}} = out_n^{(4)} \varphi'(in_m^{(3)}) \sum_{c \in L_2} \left( \delta_c^{(2)} w_{mc}^{(2)} \right).$$

A delta function related to the third layer is also defined as,

$$\delta_m^{(3)} = \varphi'(in_m^{(3)}) \sum_{c \in L_2} \left( \delta_c^{(2)} w_{mc}^{(2)} \right), \tag{3.12}$$

and with respect the weights of the second layer is

$$\frac{\partial E}{\partial w_{nm}^{(3)}} = out_n^{(4)} \delta_m^{(3)}. \tag{3.13}$$

Observing the Eqs. (3.8), (3.10) and (3.12), a recursive definition of delta function is observed. Then, by induction, the general definition of the each layer delta function is,

$$\delta_v^{(m)} = \begin{cases} \varphi'(in_v^{(1)})(\tilde{y}_v - y_v), \, m = 1 \\ \varphi'(in_v^{(m)}) \sum\limits_{b \in L_{m-1}} \left( \delta_b^{(m-1)} w_{vb}^{(m-1)} \right), \, N \geq m > 1 \end{cases}, \tag{3.14}$$

and the general error with respect each one of the $N$ layer's nodes definition is

$$\frac{\partial E}{\partial w_{tv}^{(m)}} = out_t^{(m+1)} \delta_v^{(m)}. \tag{3.15}$$

The bias of each node must be corrected as well. The procedure to calculate the error with respect to the bias is similar with the one used before for the weights,

$$\frac{\partial E}{\partial b_j^{(1)}} = \frac{\partial in_j^{(1)}}{\partial b_j^{(1)}} \frac{\partial out_j^{(1)}}{\partial in_j^{(1)}} \frac{\partial E}{\partial out_j^{(1)}},$$

$$\frac{\partial E}{\partial b_j^{(1)}} = \frac{\partial}{\partial b_j^{(1)}} \left( \sum_{a \in L_1} w_{aj}^{(1)} out_a^{(2)} + b_j^{(1)} \right) \frac{\partial \varphi(in_j^{(1)})}{\partial in_j^{(1)}} \frac{\partial}{\partial out_j^{(1)}} \left( \sum_{b \in L_1} \frac{1}{2} (\tilde{y}_b - y_b)^2 \right),$$

$$\frac{\partial E}{\partial b_j^{(1)}} = \varphi'(in_j^{(1)}) \frac{\partial}{\partial \tilde{y}_j} \left( \sum_{b \in L_1} \frac{1}{2} (\tilde{y}_b - y_b)^2 \right),$$

$$\frac{\partial E}{\partial b_j^{(1)}} = \underbrace{\varphi'(in_j^{(1)})(\tilde{y}_j - y_j)}_{\text{eq. 3.8.}},$$

$$\frac{\partial E}{\partial b_j^{(1)}} = \delta_j^{(1)}.$$

That definition will be recursive as well. Then,

$$\frac{\partial E}{\partial b_v^{(m)}} = \delta_v^{(m)}. \tag{3.16}$$

Because of that recursive definition of $\delta_v^{(m)}$, where, in order to calculate the $m^{\text{th}}$ layer's delta, it is needed the backward one ($\delta_l^{(m-1)}$). It is said that the error is back-propagated from the output layer to the first hidden layer before the inputs ($N^{\text{th}}$ layer).

Once each one of the error derivates with respect to all the weights and bias of all layers are calculated, then that error function can be minimized by some of the techniques listed below.

### 3.3.1 Gradient descent

The gradient descent is the most popular, and intuitive one. Basically the increment that will be summed to correct the weights and bias updates the error function in the opposite direction of its gradient (that's why there is a negative sign), Eqs. (3.17) and (3.18). That is, the updates make the error function to decrease in each step (also called epochs). That update is regulated by $\eta$, called learning rate. The value prescribed for $\eta$ is crucial because if it is too high, the minimum error can be

missed, and if it is too low will overly slow the minimization.

$$\Delta w_{tv}^{(m)} = -\eta \frac{\partial E}{\partial w_{tv}^{(m)}} \tag{3.17}$$

$$\Delta b_{v}^{(m)} = -\eta \frac{\partial E}{\partial b_{v}^{(m)}} \tag{3.18}$$

### 3.3.2 RMSProp

TIELEMAN e HINTON [35] proposed a variable learning rate method, where $\eta$ is divided by the recent gradient magnitude average. $\gamma$ is a forgetting factor, that regulates the weight of the previous gradient magnitude in the minimization process. That adaptive learning rate helps the convergence, creating a robust optimizer. Below, we can find how the forgetting factor is applied, whose value used in the literature is usually between 0.9 and 0.95.

$$v_w(t+1) = \gamma v_w(t) + (1-\gamma)\left(\frac{\partial E}{\partial w_{tv}^{(m)}}\right)^2 \tag{3.19a}$$

$$\Delta w_{tv}^{(m)} = -\frac{\eta}{\sqrt{v_w(t+1)}} \frac{\partial E}{\partial w_{tv}^{(m)}} \tag{3.19b}$$

$$v_b(t+1) = \gamma v_b(t) + (1-\gamma)\left(\frac{\partial E}{\partial b_{v}^{(m)}}\right)^2 \tag{3.20a}$$

$$\Delta b_{v}^{(m)} = -\frac{\eta}{\sqrt{v_b(t+1)}} \frac{\partial E}{\partial b_{v}^{(m)}} \tag{3.20b}$$

### 3.3.3 ADAM

KINGMA e BA [36] updated the RMSProp optimizer. It obeys the same concept, but the adaptive learning rate also take into account the second moment of the gradient, with the need of two corresponding forgetting factors, $\beta_1$ and $\beta_2$. That algorithm achieves good results in deep neural nets.

$$m_w(t+1) = \beta_1 m_w(t) + (1-\beta_1)\frac{\partial E}{\partial w_{tv}^{(m)}} \tag{3.21a}$$

$$v_w(t+1) = \beta_2 v_w(t) + (1-\beta_2)\left(\frac{\partial E}{\partial w_{tv}^{(m)}}\right)^2 \tag{3.21b}$$

$$\hat{m}_w(t+1) = \frac{m_w(t+1)}{1-\beta_1} \tag{3.21c}$$

$$\hat{v}_w(t+1) = \frac{v_w(t+1)}{1-\beta_2} \tag{3.21d}$$

$$\Delta w_{tv}^{(m)} = -\eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w}} \frac{\partial E}{\partial w_{tv}^{(m)}} \tag{3.21e}$$

$$m_b(t+1) = \beta_1 m_b(t) + (1-\beta_1) \frac{\partial E}{\partial w_{tv}^{(m)}} \tag{3.22a}$$

$$v_b(t+1) = \beta_2 v_b(t) + (1-\beta_2) \left( \frac{\partial E}{\partial w_{tv}^{(m)}} \right)^2 \tag{3.22b}$$

$$\hat{m}_b(t+1) = \frac{m_b(t+1)}{1-\beta_1} \tag{3.22c}$$

$$\hat{v}_b(t+1) = \frac{v_b(t+1)}{1-\beta_2} \tag{3.22d}$$

$$\Delta b_v^{(m)} = -\eta \frac{\hat{m}_b}{\sqrt{\hat{v}_b}} \frac{\partial E}{\partial b_v^{(m)}} \tag{3.22e}$$

## 3.4 Validation and test groups

During the process of minimizing the error function, to face iteration of the same (epochs), the neural network is applied in a set of data called validation data. That is, for a given time, the neural network is applied on a training input and on a validation input. Both outputs generated training and validation errors. The minimization process acts on the training error, whereas this validation error is used as the training stop criterion. This is important because it is normal for a training to minimize the training error, but the network loses the power of generalization for data that was not part of the training process. This is the so-called overfitting of the neural network, in other words, it has only learned from the data that belongs to the training group, but it can not extrapolate results to unprecedented inputs. One possible stopping criterion is the so-called early stoppig, which interrupts minimization when the validation error increases for a number of consecutive times.

After the neural network is converged, at the end of its training, it is evaluated. The data group that is intended for this evaluation is the so-called test group. The neural network basically reads a test input, and predicts a output. This output is compared to the test output, thus generating an error metric associated with this neural network. The test group is distinct from both the training group and the validation group. Soon this evaluation also helps to study the power of generalization of the network.

## 3.5 Cross-validation

The result of the test error, the error that actually evaluates the neural network, has a certain volatility. Initially the weights and bias of the network are chosen randomly, so a second training, with the same training and validation data, can lead to minimization for a different response. So impacting the test error. In order to have a statistically significant response, several training sessions with different training, validation and test groups are carried out. And, in the end, there is an average error with a given standard deviation. This average error will, in fact, validate the neural network.

A normal practice is to consider a single large database and allocate, for example, 60% of it for training, 20% for validation and 20% for testing. These groups are chosen randomly for each training process.

# Chapter 4

# Machine learning and turbulence

In recent years, machine learning has entered the universe of computational fluid dynamics simulations of turbulent flows aiming to correct the shortcomings of the turbulence models of describing phenomena such as separation of the boundary layer and secondary flows. In this chapter, the main works done in this segment are summarized, in order to better contextualize what is proposed in this present work.

The various machine learning techniques act in this segment as nonlinear regression processes for the description of some aspect of turbulence modeling, for example, to directly describe the Reynolds stress tensor. Nonlinear regression is created from data extracted from high fidelity databases, DNS or LES, that serves as targets of the various machine learning methods used in the literature. Basically, from certain inputs extracted from the RANS simulations, a non-linear regression is performed, which will, for example, predict a new $\mathbf{R}$ field that is intended to be close to a DNS or LES quality established as a target.

TRACEY *et al.* [7] used a machine learning technique called kernel regression to correct turbulent stress anisotropy eigenvalues, using as target some DNS database. The eigenvalues $\lambda_1$, $\lambda_2$ and $\lambda_3$ of the deviatoric part of $\mathbf{R}$, normalized with respect to the turbulent kinetic energy $\kappa$,

$$\mathbf{a} \equiv \frac{1}{2\kappa}\mathbf{R} - \frac{1}{3}\boldsymbol{\delta}, \tag{4.1}$$

are written in a space defined by these three quantities,

$$C_{1c} = \lambda_1 - \lambda_2, \tag{4.2a}$$

$$C_{2c} = 2(\lambda_2 - \lambda_3), \tag{4.2b}$$

$$C_{3c} = 3\lambda_3 + 1. \tag{4.2c}$$

Once these quantities are determined, a point inside the so-called barycentric map

can be located with

$$x_{bary} = C_{1c}x_{1c} + C_{2c}x_{2c} + C_{3c}x_{3c}, \tag{4.3a}$$

$$y_{bary} = C_{1c}y_{1c} + C_{2c}y_{2c} + C_{3c}y_{3c}. \tag{4.3b}$$

This map is defined inside a triangle, a color map is used to locate the point inside the map, as shown in Fig. (4.1). This map shows the limits of turbulence flows. Each

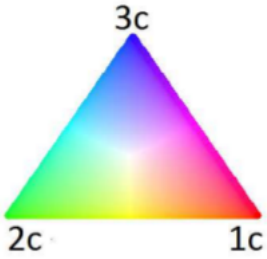
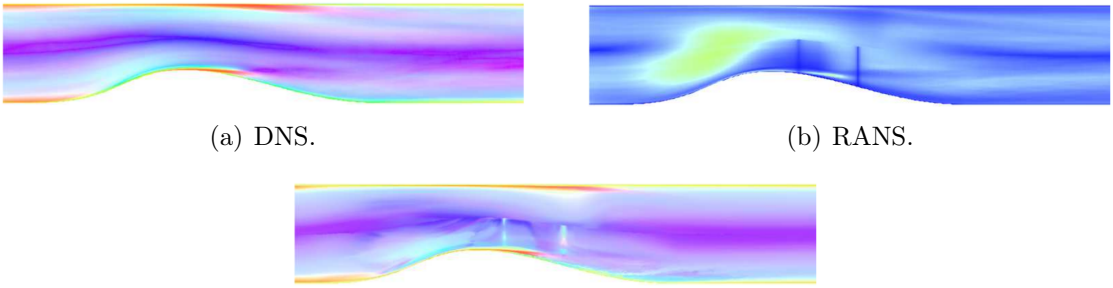



Figure 4.1: Color map representing the location in the barycentric map. [7]

one of the corners of the map is associated with a specific turbulent flow pattern. In order to the flow to be realizable, the anisotropy of the turbulent field must be inside this equilateral triangle.

The authors of this work have used machine learning to predict a corrected anisotropy turbulent field, written in barycentric map coordinates, of a periodic hill flow pattern in order to correct a RANS simulation that used a $\kappa$-$\omega$ SST turbulent model. It was used a target a related DNS simulation. In Fig. (4.2), it is notable that after the correction the anisotropic field is closer to the DNS result, showing the capability of correcting the eigenvalues of the deviatoric part of $\mathbf{R}$. It was reported some difficulties in generalizing the results to new flows and to scale to a large amount of data.



(a) DNS.

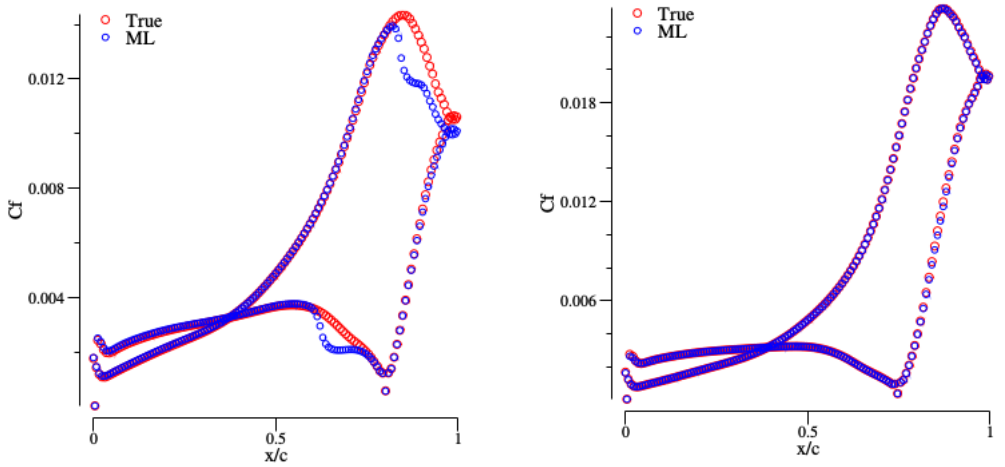
(b) RANS.

(c) Predicted.

Figure 4.2: Anisotropy in a periodic hill. [7]

After that, TRACEY *et al.* [8] proposed a single hidden layer NN to model the source terms from the Spalart Allmaras RANS model. In this case, it no longer fixes the tensor $\mathbf{R}$ directly, but rather a specific portion of a RANS model. This example constitutes a hybrid model, which still carries RANS modeling and uses machine

learning to describe turbulence. In that work, it was analysed a external flow with respect to a NACA 0012 wing, and its friction coefficient correction performed by the prediction of the source term of the turbulent viscosity transport equation of the Spalart Allmaras turbulent model. In Fig. (4.3) it can be seen two results in, each one of them different training data are used: in Fig. (4.3(a)) pressure driven channels data, and in Fig. (4.3(b)) flat plate solutions. This attempt has shown the potential of NN for turbulence modeling.

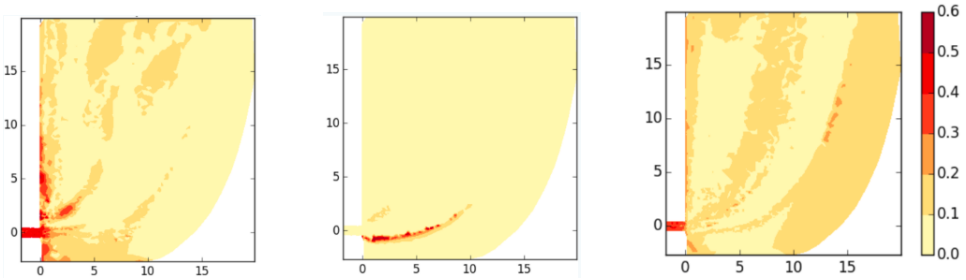



(a) NACA 0012 airfoil using training data from the pressure driven channels.

(b) NACA 0012 airfoil using training data from the three flat plate solutions.

Figure 4.3: Friction coefficient. [8]

LING *et al.* [9] used a random forest regression to model the deviatoric part of $\mathbf{R}$. In Fig. (4.4) it is possible to see the correction of the second invariant of the anisotropy of $\mathbf{R}$ of a $\kappa$-$\epsilon$ RANS simulation, using as a target a jet-in-cross flow LES results. This technique presented a poor ability to predict this tensor because




(a) LES. (b) RANS. (c) Predicted.

Figure 4.4: Second anisotropy invariant $II_a$ in a periodic hill. [9]

of difficulty to ensure Galilean invariance. If a certain DNS database is used as training data and then this same database is used for a second training but written in another reference system, this new regression will be distinct from the first. This is because what is being predicted by the network, in this case a tensor field, is

not invariant under Galilean transformations. This compromises the generalization potential of the regression process because the resulting neural network can only be used in RANS simulations that have a coordinate system similar to that used in the DNS base.

Later, LING *et al.* [17] developed a NN to predict the anisotropy Reynolds stress tensor eigenvalues using a set of Galilean invariants inputs. That technique showed significant performance improvements when compared to [9]. Those results evidenced the importance of the Galilean invariance in ML turbulence modelling.

More recently, LING *et al.* [18] developed a deep specialized NN architecture witch ensured Galilean invariant anisotropic turbulent tensor. Those results showed that deep learning and the specialized architecture is able to provide a significant performance improvement.

# Chapter 5

# Square duct flow

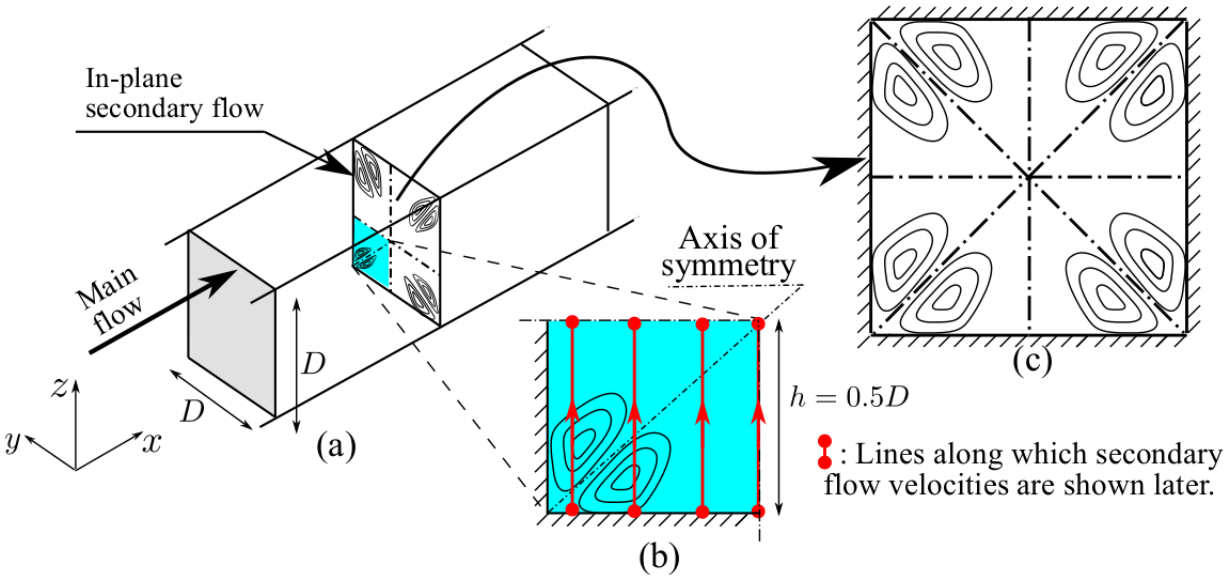



Figure 5.1: Schematic drawing of the square duct flow [10].

The geometry of the square duct flow is presented in the Fig. (5.1). The Reynolds number that rules the problem is computed with respect to the hydraulic diameter $D$ and the bulk velocity. As it can be seen, only a quarter of the duct section is numerically analyzed (the colored one). This flow pattern was chosen due to the existence of a secondary flow located in the cross-plane. A secondary flow is a minor flow that is superposed by a primary flow. The primary flow is close to the magnitude of total flow, since the secondary velocity components are order of magnitude smaller than the primary flow. For the square duct flow, in each one of the four edges of this cross-plane, there is a counter-rotating recirculation pair.

It is known that two equations RANS models have difficulty on capturing this type of phenomenon because of linear eddy viscosity hypothesis [13], as mentioned in Turbulent flows sections. The inability to describe this secondary flow will be overcome by the correction of turbulent stresses via machine learning.

The chosen RANS model was the $\kappa$-$\epsilon$ [30] implemented in OpenFOAM-4.x (OF). The employed boundary conditions are the non-slip condition and null pressure gradient on the solid faces (green faces in Fig. 5.2), symmetry in the cut-off regions of the quarter section (red faces in Fig. 5.2). In the direction of the main flow ($x$ direction in Fig. 5.2) the periodic condition and the constant section average velocity are defined. This constant bulk velocity condition basically creates an artificial component of pressure gradient in the axial direction of the duct, this is necessary because in the inlet and outlet faces the boundary conditions are periodic. This component then passes through an iterative process whose stopping criterion is defined with respect to the bulk velocity difference obtained at each iteration and the desired one. The turbulent kinetic energy in the walls is zero, and for turbulent dissipation the OF wall function epsilonWallFunction was used.

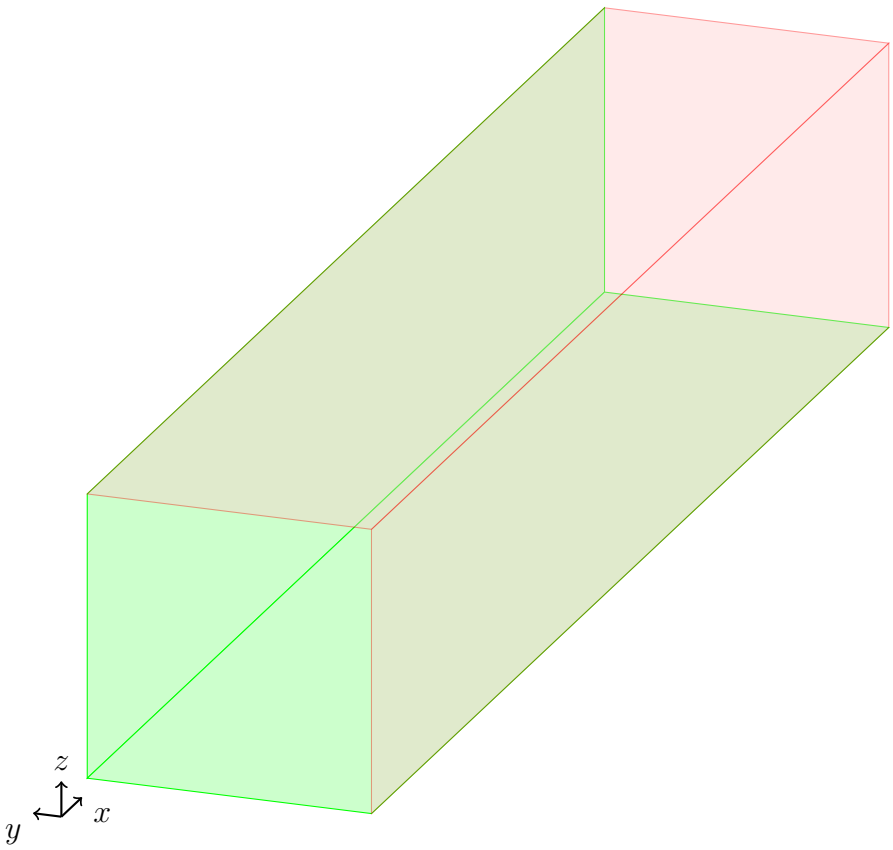



Figure 5.2: Boundary conditions.

It was used the simpleFoam, a finite volume OF solver. It is a solver for non-steady and incompressible flows of Newtonian fluids. It uses the SIMPLE algorithm of pressure and velocity decoupling. The discretization of divergent operators from the transport equations chosen for the RANS simulations was Gauss upwind, the gradient operators were discretized with linear Gauss. For the laplacians, linear corrected Gauss was chosen, and the point-to-point interpolation scheme used was linear.

For the solution of the discretized pressure equation, the Preconditioned Conjugate Gradient (PCG) solver was used, with the Diagonal Incomplete-Cholesky (DIC) preconditioner. For the velocity, $\kappa$ and $\epsilon$ was used the smooth solver, with the Gauss Seidel symmetrical smoother. The tolerances for all four variables were $1e-07$, with relaxation factors for the pressure field of 0.5, from 0.15 for the velocity equation and 0.5 for the turbulent fields.

In order to create a neural network capable of correcting the turbulent flow predicted by the $\kappa$-$\epsilon$, it is necessary to provide a reference database. The machine, to carry out its learning process, will use as reference the turbulence coming from a high fidelity and very computationally expensive database, DNS simulations. For this work, it will be used a set of flow cases corresponding to the following values for the Reynolds number: 2200, 2400, 2600, 2900, 3200 and 3500. The square duct DNS data was made available by PINELLI *et al.* [37]. For each one of those six DNS, a corresponding RANS simulation is performed. With these DNS and RANS data, a NN will be trained in order to correct the turbulent flow obtained from a RANS model using the results provided by the DNS as targets.

For all the simulations, it was used a bulk velocity of $0.4819\mathrm{m\,s^{-1}}$. The solved domain has $1\mathrm{m} \times 1\mathrm{m} \times 10\mathrm{m}$, that is $D = 1\mathrm{m}$. Each simulations is differentiated by the kinematic viscosities listed in tab. 5.1.

| $Re$ | $\nu[\mathrm{m^2\,s^{-1}}] \times 10^{-4}$ |
|---|---|
| 2200 | 2.1858 |
| 2400 | 2.0082 |
| 2600 | 1.8537 |
| 2900 | 1.6619 |
| 3200 | 1.5061 |
| 3500 | 1.3770 |

Table 5.1: Simulations viscosities.

The mesh topology can be seen in Fig. (5.3). The cross-section has $40 \times 40$ nodes, where the mesh is more refined near the walls in such a way to ensure $y^+ < 0.2$. In the main flow direction, the mesh is equally divided in ten parts. Then the mesh has a total of 16000 centroids.

Although the $\kappa$ - $\epsilon$ model is a high Reynolds number model, we chose to perform a extra-refinement near the walls so that there were more points in this region and, thus, the neural network could correct there as well. OF was set to use wall laws even though there are centroids within the viscous sublayer.

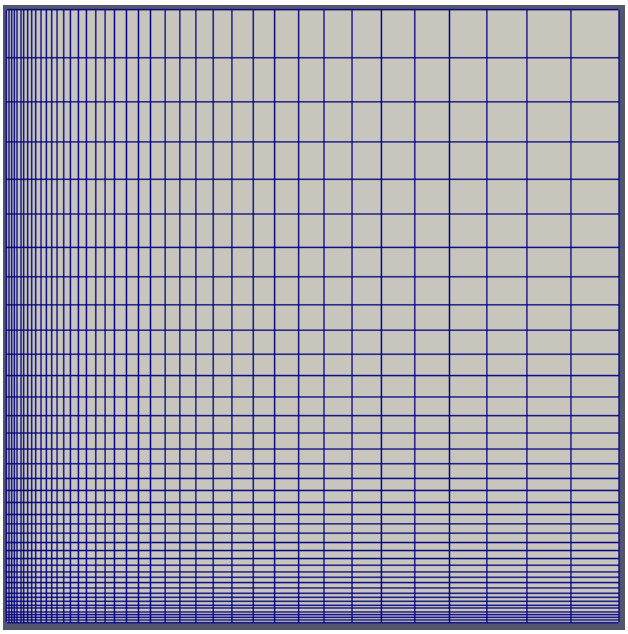


Figure 5.3: Mesh's topology.

# Chapter 6

# Methodology

This chapter explains how to use neural networks to correct RANS simulations. The first thing to do is to define which inputs the neural network will receive to correct the turbulence obtained by the RANS simulations. In this work we selected quantities associated with mean kinematics and turbulence, as can be seen in Tab. 6.1. The

Table 6.1: Inputs.

| | |
|---|---|
| $\mathbf{D}$ | $\nabla \cdot \mathbf{D}$ |
| $\mathbf{P}$ | $\nabla \cdot \mathbf{P}$ |
| $\mathbf{D}^2$ | $\nabla \cdot \mathbf{D}^2$ |
| $\mathbf{P}^2$ | $\nabla \cdot \mathbf{P}^2$ |
| $\mathbf{D} \cdot \mathbf{P} + \mathbf{P} \cdot \mathbf{D}$ | $\nabla \cdot (\mathbf{D} \cdot \mathbf{P} + \mathbf{P} \cdot \mathbf{D})$ |
| $\mathbf{D}^2 \cdot \mathbf{P} + \mathbf{P} \cdot \mathbf{D}^2$ | $\nabla \cdot (\mathbf{D}^2 \cdot \mathbf{P} + \mathbf{P} \cdot \mathbf{D}^2)$ |
| $\mathbf{P}^2 \cdot \mathbf{D} + \mathbf{D} \cdot \mathbf{P}^2$ | $\nabla \cdot (\mathbf{P}^2 \cdot \mathbf{D} + \mathbf{D} \cdot \mathbf{P}^2)$ |
| $\mathbf{R}$ | $\nabla \cdot \mathbf{R}$ |
| $\mathbf{R}^2$ | $\nabla \cdot \mathbf{R}^2$ |
| $\mathbf{D} \cdot \mathbf{R} + \mathbf{R} \cdot \mathbf{D}$ | $\nabla \cdot (\mathbf{D} \cdot \mathbf{R} + \mathbf{R} \cdot \mathbf{D})$ |
| $\mathbf{D}^2 \cdot \mathbf{R} + \mathbf{R} \cdot \mathbf{D}^2$ | $\nabla \cdot (\mathbf{D}^2 \cdot \mathbf{R} + \mathbf{R} \cdot \mathbf{D}^2)$ |
| $\mathbf{R}^2 \cdot \mathbf{D} + \mathbf{D} \cdot \mathbf{R}^2$ | $\nabla \cdot (\mathbf{R}^2 \cdot \mathbf{D} + \mathbf{D} \cdot \mathbf{R}^2)$ |

list of features that the neural network is calculated as a function of the strain rate tensor $\mathbf{D}$, the non-persistence-of-straining tensor $\mathbf{P}$ and the Reynolds stress tensor $\mathbf{R}$ computed from the RANS simulations. In total there are twelve symmetrical tensors, and the vectors that correspond to the divergence operation applied to these tensors. Each component of each of these tensors and vectors corresponds to a feature, that is, in total the NN receives an input with 108 features.

The tensor $\mathbf{P}$ is the non-persistence-of-straining tensor [38],

$$\mathbf{P} = \mathbf{D} \cdot \overline{\mathbf{W}} - \overline{\mathbf{W}} \cdot \mathbf{D}. \tag{6.1}$$

$\overline{\mathbf{W}}$ is the relative vorticity [38], it is the regular vorticity tensor discounted from the

tensor that represents the rate of rotation of the eigenvectors of $\mathbf{D}$ ($\mathbf{\Omega^D}$),

$$\overline{\mathbf{W}} = \mathbf{W} - \mathbf{\Omega}^D. \tag{6.2}$$

Both entities are objective. The non-persistence-of-straining tensor is a deformation local measurement. If the tensor is zero, it means that the fluid is in a configuration of maximum deformation. Which makes sense because, observing $\mathbf{P}$ in $\mathbf{D}$ eigenvectors base,

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

the tensor only cancels out in two cases: Having all $\mathbf{D}$ eigenvalues equal, indicating that any direction is a direction of maximum stretching. Or if the relative vorticity is zero, which means that the filament rotates along with the eigenvectors of $\mathbf{D}$, so that it always stays at maximum stretching direction. More details about $\mathbf{P}$ can be found in [39].

The samples that will be drawn to compose the NN input are taken from the centroids of each of the RANS simulations. In this work we use six RANS simulations, differentiated by the Reynolds number, each with 16000 centroids. For network training, four of these six RANS are always used, for validation one simulation and for testing the converged network, the last simulation. This means that the training input will have 64000 samples, while the test and validation inputs have 16000 each.

In this work, a open source python library, called Keras [40], is used to construct the neural networks architectures, performing the training and to validate the NN itself. Keras library uses other famous libraries like GOOGLE's TensorFlow, and Theano with the advantage of creating a more user-friendly interface.

The input must be defined as an array whose columns represent the features defined in the table 6.1, and each lines represent each sample, in this case each of the centroids of the RANS simulations. The training input is a $64000 \times 108$ matrix and is defined as $X_{\text{train}}$. The inputs for the validation and testing of the neural network will be both matrices of $18000 \times 108$ defined as $X_{val}$ and $X_{\text{test}}$ respectively.

The output is what is wanted from the NN to be able to predict. The literature-based output is a set composed by the six components of the Reynolds stress tensor provided by DNS databases. The purpose of the neural network is to correct the turbulence predicted by RANS models, making it as close as possible to the turbulence obtained from a DNS. Therefore, the choice of output as being the $\mathbf{R}$ components of a DNS database is justified. The training output is a $64000 \times 6$ array, $Y_{\text{train}}$ array. And the validation and test outputs are $18000 \times 6$ matrices, called $Y_{\text{val}}$ and $Y_{\text{test}}$

respectively.

A common practice in neural networks is to conduct all data, both input and output, go through a preprocessing step. For example, each of the columns in some of the input arrays represent distinct features. One can represent the value of some component of the tensor $\mathbf{D}$, while another column may represent some component of the divergence of the tensor $\mathbf{P}$. These columns may have different order of magnitudes. When these order of magnitudes are very distinct, this is likely to make the convergence of the minimization that occurs in the NN training stage very difficult. Because of this, it is necessary to pre-process the data, including the input and output. The preprocessing in question is the application of some kind of normalization in each of the columns of the matrices. In this work the standard normalization was chosen. For each column of training inputs and outputs, the standard deviation ($\sigma$) and the mean ($\mu$) are calculated, and then each of these columns is normalized using their respective means and standard deviations,

$$\text{normalized column} = \frac{\text{column} - \mu_{\text{column}}}{\sigma_{\text{column}}}. \tag{6.3}$$

This normalization is applied to all data, inputs and outputs for training, validation and testing is defined only in terms of the means and standard deviations computed from the columns of the training data. Normalization is expected to be done for the validation and test data as these are well represented by the training data. The resulting neural network provides a normalized Reynolds tensor, what implies that afterwards the results need to be rescaled,

$$\text{column} = \sigma_{\text{column}}\,(\text{normalized column}) + \mu_{\text{column}}. \tag{6.4}$$

To do this in Python, you use a widely known library of machine learning called scikit-learn [41] using the following code.

```
1   import numpy as np
2   from sklearn.preprocessing import StandardScaler
3
4   # normalizing the dataset
5   x_scaler = StandardScaler()
6   Xn_train = x_scaler.fit_transform(X_train)
7   y_scaler = StandardScaler()
8   Yn_train = y_scaler.fit_transform(Y_train)
9   Xn_val  = x_scaler.transform(X_val)
10  Yn_val  = y_scaler.transform(Y_val)
11  Xn_test = x_scaler.transform(X_test)
12  Yn_test = y_scaler.transform(Y_test)
```

With the normalized data, the architecture of the neural network itself is now

projected. For this work two hidden layers are used with 100 neurons in each totally connected. The first layer of neurons, the inputs layer, must have the same number of input data features, that is, 108 neurons in the first layer of the NN. The last layer must have the same number of neurons as the number of elements to be predicted with the network, in the case of the methodology based on the literature, the 6 components of the Reynolds stress tensor. So the last layer will have 6 neurons.

The activation functions chosen were the sigmoid for the hidden layers, and linear for the last layer, the choice of this activation function for the output layer is classic for regression tasks. Below you can see how this architecture is implemented using Keras.

```
1  from keras.models import Sequential
2  from keras.layers import Activation,Dense
3
4  # create model
5  hidenNeurons=100
6  features=108
7
8  model = Sequential()
9  model.add(Dense(hidenNeurons, input_dim=features))
10 model.add(Activation('sigmoid'))
11 model.add(Dense(hidenNeurons))
12 model.add(Activation('sigmoid'))
13 model.add(Dense(6))
14 model.add(Activation('linear'))
```

The employed numerical method of minimizing bias and weights of NN is ADAM. The ADAM coefficients are the Keras standards, there was no need for line 5 of the code below, it was only displayed in order to show the possibility that these minimization parameters may be customizable. The error function that will be minimized is the mean square error (mse). The so-called 'early stopping' was used as the stopping criterion. It is a criterion that interrupts the minimization process when the error of the validation group begins to grow after a certain number of epochs. In the present case it was set for this to occur if the validation error goes up for 10 consecutive epochs. The other criterion chosen was a maximum number of epochs equal to 500.

```
1  from keras.callbacks import EarlyStopping
2  from keras import optimizers
3
4  # Compile model
5  adam=optimizers.Adam(lr=1.0e-08, beta_1=0.9, beta_2=0.999, epsilon
       =1e-08, decay=0.0)
6  model.compile(loss='mse', optimizer='adam')
7
```

```
    # Fit the network
    earlyStopping=EarlyStopping(monitor='val_loss', patience=10,
        verbose=0, mode='auto')
    history = model.fit(Xn_train, Yn_train, epochs=500, callbacks=[
        earlyStopping], validation_data=(Xn_val,Yn_val))
```

With the neural network converged, its architecture is saved in a 'json' file and the weights and bias in an 'h5' file.

```
    #save
    # serialize model to JSON
    model_json = model.to_json()
    with open("model.json", "w") as json_file:
        json_file.write(model_json)
    # serialize weights to HDF5
    model.save_weights("model.h5")
    print("Saved model to disk")
```

At this point the NN can be used to predict new Reynolds stress tensors for a new set of data that was not part of its training. A successful network will predict a tensor field of DNS-quality turbulent voltages from the test input data. As its name suggests, this is a way of testing whether the network has the ability to generalize what was learned in the training stage. The network will then predict a field of $\mathbf{R}$ that will be normalized , so this result must be post-processed to be rescaled ($Y_{\text{pred}}$) by applying the inverse transformation described in Eq. 6.4.

```
    # Make predictions
    Yn_pred = model.predict(Xn_test)
    Y_pred = y_scaler.inverse_transform(Yn_pred)
    np.savetxt('Y_pred.dat',Y_pred)
```

To evaluate the neural network, we compare the predicted output with the test output. The error metric used in this is the coefficient of variation

$$vc = 100\frac{\sqrt{mse}}{||mean(Y_{\text{test}})||}. \tag{6.5}$$

Which computes the root mean square error percentage with respect to the expected average value.

```
    # Evaluate the network
    from sklearn.metrics import mean_squared_error

    mse=mean_squared_error(Y_test, Y_pred, multioutput='raw_values')

    erms=np.sqrt(mse)

    vc=100*np.divide(erms,np.absolute(np.mean(Y_test)))
```

The complete code, which includes the preprocessing, NN creation and prediction of the corrected Reynolds tensor field can be seen below.

Listing 6.1: Complete code.

```
def main():
    import numpy as np
    from sklearn.preprocessing import StandardScaler

    # normalizing the dataset
    x_scaler = StandardScaler()
    Xn_train = x_scaler.fit_transform(X_train)
    y_scaler = StandardScaler()
    Yn_train = y_scaler.fit_transform(Y_train)
    Xn_val  = x_scaler.transform(X_val)
    Yn_val  = y_scaler.transform(Y_val)
    Xn_test = x_scaler.transform(X_test)
    Yn_test = y_scaler.transform(Y_test)




    import numpy as np
    from sklearn.preprocessing import StandardScaler

    # normalizing the dataset
    x_scaler = StandardScaler()
    Xn_train = x_scaler.fit_transform(X_train)
    y_scaler = StandardScaler()
    Yn_train = y_scaler.fit_transform(Y_train)
    Xn_val  = x_scaler.transform(X_val)
    Yn_val  = y_scaler.transform(Y_val)
    Xn_test = x_scaler.transform(X_test)
    Yn_test = y_scaler.transform(Y_test)




    from keras.models import Sequential
    from keras.layers import Activation,Dense

    # create model
    hidenNeurons=100
    features=108

    model = Sequential()
    model.add(Dense(hidenNeurons, input_dim=features))
    model.add(Activation('sigmoid'))
```

```
44      model.add(Dense(hidenNeurons))
45      model.add(Activation('sigmoid'))
46      model.add(Dense(6))
47      model.add(Activation('linear'))
48
49
50
51
52      from keras.callbacks import EarlyStopping
53      from keras import optimizers
54
55      # Compile model
56      adam=optimizers.Adam(lr=1.0e-08, beta_1=0.9, beta_2=0.999, epsilon
            =1e-08, decay=0.0)
57      model.compile(loss='mse', optimizer='adam')
58
59      # Fit the network
60      earlyStopping=EarlyStopping(monitor='val_loss', patience=10,
            verbose=0, mode='auto')
61      history = model.fit(Xn_train, Yn_train, epochs=500, callbacks=[
            earlyStopping], validation_data=(Xn_val,Yn_val))
62
63
64
65
66      #save
67      # serialize model to JSON
68      model_json = model.to_json()
69      with open("model.json", "w") as json_file:
70          json_file.write(model_json)
71      # serialize weights to HDF5
72      model.save_weights("model.h5")
73      print("Saved model to disk")
74
75
76
77
78      # Make predictions
79      Yn_pred = model.predict(Xn_test)
80      Y_pred = y_scaler.inverse_transform(Yn_pred)
81      np.savetxt('Y_pred.dat',Y_pred)
82
83
84
85
86      # Evaluate the network
87      from sklearn.metrics import mean_squared_error
```

```
88
89      mse=mean_squared_error(Y_test, Y_pred, multioutput='raw_values')
90
91      erms=np.sqrt(mse)
92
93      vc=100*np.divide(erms,np.absolute(np.mean(Y_test)))
94  main()
```

This code is then able to use machine learning to predict a new Reynolds stress field with a quality closer to the one obtained from DNS. With this new field, the average velocity and pressure fields are recalculated with the same solver, hoping that these are also closer to the DNS. Below is a summary of the methodology used.

**Training**

- Select a subset of DNS cases (geometry ans flow conditions);
- Run a RANS simulation for the same subset of cases;
- Extract the inputs described in tab. 6.1 from all RANS simulations, from each centroids, creating $X_{\text{train}}$ and $X_{\text{val}}$;
- Extract the six components of $\mathbf{R}$ from DNS database, defining the $Y_{\text{train}}$ and $Y_{\text{val}}$ outputs;
- Run the python code in order to create (training) the NN.

**Correcting**

- Run the RANS;
- Post process the converged results in order to get the NN inputs described in tab. 6.1 from each RANS centroids;
- Run the NN with the extracted inputs before, and predicts the Reynolds stress tensor $\mathbf{R}_{\text{pred}}$ using the Python code;
- Replaces the $\mathbf{R}_{\text{rans}}$ field with $\mathbf{R}_{\text{pred}}$ in OpenFOAM;
- With that turbulent stress corrected, rerun the used solver in order to obtain the mean corrected velocity and pressure fields.

## 6.1 Correcting $\nabla \cdot \mathbf{R}$

Since the primary objective is to correct the turbulence obtained from the RANS model through neural networks, it would be considered as a first analysis to correct the turbulent stresses. However, this is not the ideal choice. THOMPSON *et al.*

[20] have shown fields that the Reynolds stress tensor is not as well converged as the mean velocity and pressure fields in DNS simulations, and this lack of convergence creates uncertainties that are propagated and amplified to the mean velocity and pressure fields if this tensor is plugged in the balance of mean momentum equation. What implies that if a NN is trained to achieve $\mathbf{R}_{\text{DNS}}$, the velocity and pressure calculated will be contaminated by the intrinsic DNS uncertainties.

To bypass this problem, one can correct the divergence of the Reynolds stress tensor. There are two ways to calculate the Reynolds stress tensor divergent, the first one is directly applying the divergence operator in the tensorial field provided by the DNS database,

$$\boldsymbol{\nabla} \cdot \mathbf{R}|_{\text{DNS}} = \boldsymbol{\nabla} \cdot \mathbf{R}_{\text{DNS}}. \tag{6.6}$$

As were discussed, this resulting vector field will carry uncertainties originated from the DNS data. To calculate this divergence bypassing this problem, it is necessary to calculate it only in terms of the given first order statistical momentum entities, the DNS pressure and velocity fields. That can be done using the averaged momentum equation,

$$\boldsymbol{\nabla} \cdot \mathbf{R}|_{\text{DNS}} = -(\langle \mathbf{v} \rangle_{\text{DNS}} \cdot \boldsymbol{\nabla})(\langle \mathbf{v} \rangle_{\text{DNS}}) - \frac{1}{\rho} \boldsymbol{\nabla} \langle p \rangle_{\text{DNS}} + \nu \boldsymbol{\nabla}^2 \langle \mathbf{v} \rangle_{\text{DNS}}. \tag{6.7}$$

The vector field $\boldsymbol{\nabla} \cdot \mathbf{R}|_{\text{DNS}}$ calculated by Eq. 6.7 will be more accurate than the one calculated by Eq. 6.6. Because of that, this one will be used as target for the NN correction, that is the neural network will correct the $\boldsymbol{\nabla} \cdot \mathbf{R}|_{\text{RANS}}$ in the direction of $\boldsymbol{\nabla} \cdot \mathbf{R}|_{\text{DNS}}$ that was extract from DNS data by Eq. 6.7.

However, for this specific DNS database [37], the mean pressure field was not given. So, a auxiliary vector $\mathbf{t}$ is defined as,

$$\mathbf{t} \equiv \boldsymbol{\nabla} \cdot \mathbf{R} + \frac{1}{\rho} \boldsymbol{\nabla} \langle p \rangle, \tag{6.8}$$

and, this vector field can be extracted from the DNS database in terms of the given velocity field only as,

$$\mathbf{t}_{\text{DNS}} = -(\langle \mathbf{v} \rangle_{\text{DNS}} \cdot \boldsymbol{\nabla})(\langle \mathbf{v} \rangle_{\text{DNS}}) + \nu \boldsymbol{\nabla}^2 \langle \mathbf{v} \rangle_{\text{DNS}}. \tag{6.9}$$

In the present work we propose a new methodology which consists on correcting the $\mathbf{t}$ vector field. The output corresponding is the $\mathbf{t}$ field computed from the DNS data, $\mathbf{t}_{\text{DNS}}$. Therefore, the number of columns of the training, validation and test outputs is three, one for the witch component of the vector $\mathbf{t}$, evaluated at each centroid of each of the simulations.

The algorithm to correct this vector is similar to the previous one, the only

difference between them is the number of columns of the output, defined in line 46 of the code 6.1, witch becomes three. The summary of this methodology is shown below.

**Training**

- Select a subset of DNS cases (geometry and flow conditions);
- Run a RANS simulation for the same subset of cases;
- Extract the inputs from described in tab. 6.1 from all RANS simulations, from each centroids, creating $X_{\text{train}}$ and $X_{\text{val}}$;
- Extract the three components of $\mathbf{t}$, calculated with Eq. 6.9, from DNS database, defining the $Y_{\text{train}}$ and $Y_{\text{val}}$ outputs;
- Run the python code in order to create (training) the NN.

**Correcting**

- Run the RANS;
- Post process the converged results in order to get the NN inputs described in tab. 6.1 from each RANS centroids;
- Run the NN with the extracted inputs before, and predicts the Reynolds stress tensor $\mathbf{t}_{\text{pred}}$ using the Python code;
- Replaces the $\mathbf{t}_{\text{rans}}$ field with $\mathbf{t}_{\text{pred}}$ in OpenFOAM;
- With the corrected $\mathbf{t}$, solve

$$(\langle\mathbf{v}\rangle \cdot \boldsymbol{\nabla})\langle\mathbf{v}\rangle + \mathbf{t}_{\text{pred}} = \nu\boldsymbol{\nabla}^2\langle\mathbf{v}\rangle. \tag{6.10}$$

Note that the Eq. 6.10 does not use the SIMPLE algorithm, as it is not solving the average pressure, only the mean velocity field.

# Chapter 7

# Results

To motivate the proposed methodology, it was plotted the velocity fields calculated from provided DNS Reynolds tensor, in order to show uncertainty propagations caused by its lack of convergence.

In Fig. 7.1, you can see three columns of figures. The first column refers to the velocity field calculated from the Reynolds tensor field provided by the $Re = 3200$ DNS database. This is calculated by the linear momentum average equation,

$$(\langle \mathbf{v} \rangle_{\text{rec}} \cdot \boldsymbol{\nabla}) \langle \mathbf{v} \rangle_{\text{rec}} = -\frac{1}{\rho} \boldsymbol{\nabla} \langle p \rangle_{\text{rec}} + \nu \boldsymbol{\nabla}^2 \langle \mathbf{v} \rangle_{\text{rec}} + \boldsymbol{\nabla} \cdot \mathbf{R}_{\text{DNS}}. \tag{7.1}$$

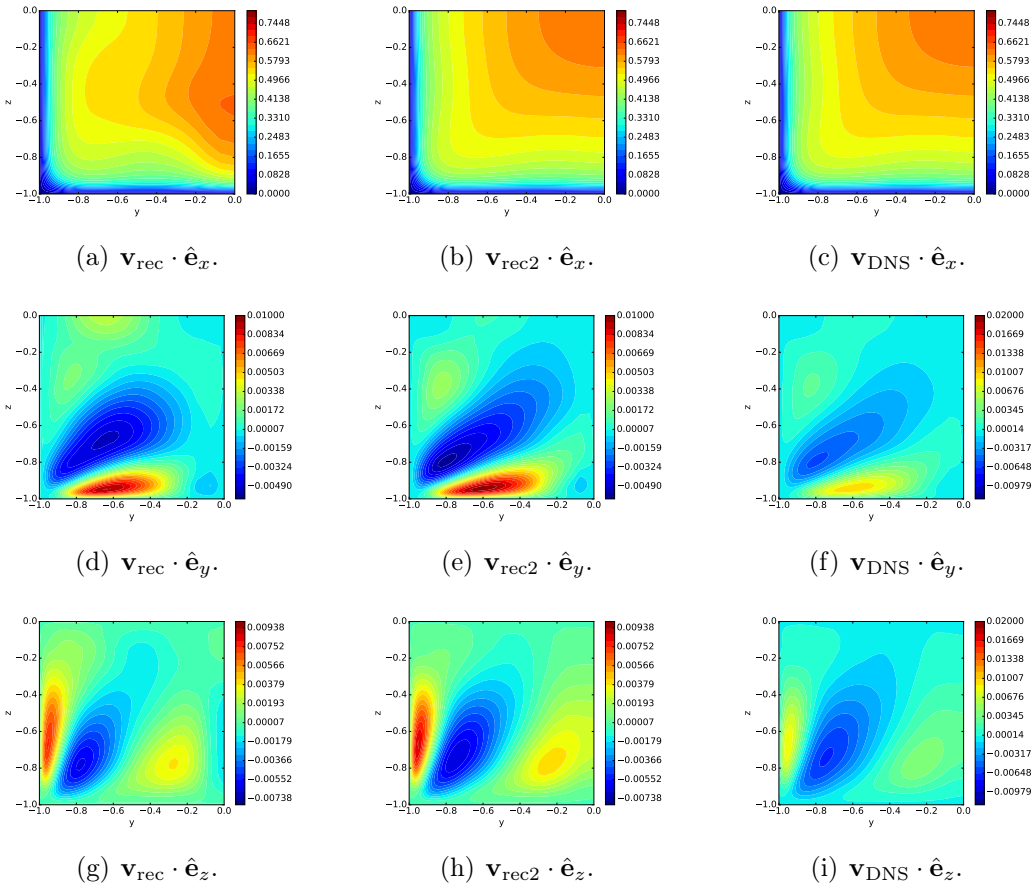
The result is the mean velocity field reconstructed by $\mathbf{R}_{\text{DNS}}$ ($\langle \mathbf{v} \rangle_{\text{rec}}$), Figs. (7.1(a)), (7.1(d)) and (7.1(g)). The second one is the velocity field obtained from the $\mathbf{t}_{\text{DNS}}$ extracted from the $Re = 3200$ DNS database by solving the modified linear momentum average equation

$$(\langle \mathbf{v} \rangle_{\text{rec2}} \cdot \boldsymbol{\nabla}) \langle \mathbf{v} \rangle_{\text{rec2}} + \mathbf{t}_{\text{DNS}} = \nu \boldsymbol{\nabla}^2 \langle \mathbf{v} \rangle_{\text{rec2}}. \tag{7.2}$$

The result of this equation is the mean velocity field reconstructed by $\mathbf{t}_{\text{DNS}}$ ($\langle \mathbf{v} \rangle_{\text{rec2}}$), Figs. (7.1(b)), (7.1(e)) and (7.1(h)). The third column is the average velocity field provided by the DNS database itself ($\langle \mathbf{v} \rangle_{\text{DNS}}$), Figs. (7.1(c)), (7.1(f)) and (7.1(i)).

The uncertainties of the Reynolds stress tensor provided by the DNS database, as expected, were propagated to the velocity field. In the main flow direction ($x$) the error was higher than in the others. But the secondary flow was also affected. This shows that even in the case of a perfect correction applied to the RANS Reynolds stress tensor towards the direction of the DNS Reynolds stress tensor, as is done in the literature, the velocity field obtained from this will be contaminated by the uncertainties of the DNS database itself.

On the other hand, the velocity fields obtained by $\mathbf{t}$ are satisfactorily similar to the DNS field, showing that if this entity is corrected in the RANS simulations to approach DNS, the velocity field obtained from it will be very close to a DNS



(a) $\mathbf{v}_{\text{rec}} \cdot \hat{\mathbf{e}}_x$. (b) $\mathbf{v}_{\text{rec2}} \cdot \hat{\mathbf{e}}_x$. (c) $\mathbf{v}_{\text{DNS}} \cdot \hat{\mathbf{e}}_x$.

(d) $\mathbf{v}_{\text{rec}} \cdot \hat{\mathbf{e}}_y$. (e) $\mathbf{v}_{\text{rec2}} \cdot \hat{\mathbf{e}}_y$. (f) $\mathbf{v}_{\text{DNS}} \cdot \hat{\mathbf{e}}_y$.

(g) $\mathbf{v}_{\text{rec}} \cdot \hat{\mathbf{e}}_z$. (h) $\mathbf{v}_{\text{rec2}} \cdot \hat{\mathbf{e}}_z$. (i) $\mathbf{v}_{\text{DNS}} \cdot \hat{\mathbf{e}}_z$.

Figure 7.1: Reconstructed velocity field by $\mathbf{R}_{\text{DNS}}$ ($\mathbf{v}_{\text{rec}}$), by $\mathbf{t}_{\text{DNS}}$ ($\mathbf{v}_{\text{rec2}}$) and the velocity field given by DNS $Re = 3200$ database.

quality.

## 7.1 Correction using the Reynolds stress tensor as the target

The architecture of the neural network was evaluated through a cross-validation performed in ten different training, validation and test groups. After that, the means of each of the coefficients of variation, the error metric that calculates the percentage of the mean square error in relation to the expected average

$$vc = 100 \times \frac{rms}{\langle test \rangle}, \tag{7.3}$$

of each of the six components predicted by the network. Each of the cross-validation groups were composed by simulations numbered from 1 to 6 corresponding to the Reynolds numbers of 2200, 2400, 2600, 2900, 3200 and 3500, respectively.

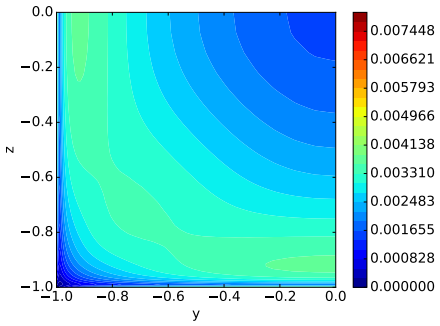
In the Tab. 7.1, we can see that the highest error was found in the forecast of

the xx component of the Reynolds tensor, 4.06%. For the other components, the errors were all less than 1%. This shows the ability of the network to correct this turbulent tensor field.

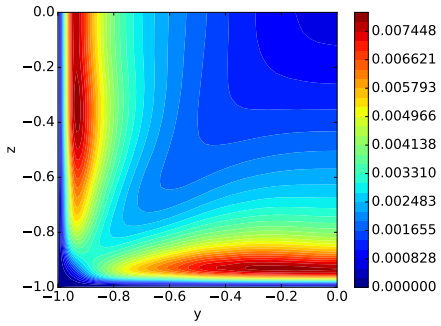
Table 7.1: **R** cross-validation errors (variation coefficients described in Eq. 7.3) [%].

| train | val | test | $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_x$ | $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_y$ | $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ | $\hat{\mathbf{e}}_y \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_y$ | $\hat{\mathbf{e}}_y \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ | $\hat{\mathbf{e}}_z \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ |
|---|---|---|---|---|---|---|---|---|
| 1-2-4-6 | 3 | 5 | 2.38 | 0.28 | 0.31 | 0.41 | 0.03 | 0.51 |
| 1-2-3-4 | 5 | 6 | 8.80 | 0.84 | 0.87 | 0.99 | 0.09 | 1.00 |
| 3-4-5-6 | 1 | 2 | 6.29 | 0.44 | 0.58 | 0.55 | 0.06 | 0.68 |
| 1-4-5-6 | 3 | 2 | 3.69 | 0.43 | 0.41 | 0.54 | 0.03 | 0.62 |
| 1-2-5-6 | 3 | 4 | 2.95 | 0.34 | 0.34 | 0.53 | 0.04 | 0.53 |
| 1-3-4-5 | 2 | 6 | 3.78 | 0.39 | 0.41 | 0.68 | 0.05 | 0.49 |
| 1-4-5-6 | 2 | 3 | 3.28 | 0.39 | 0.20 | 0.43 | 0.04 | 0.65 |
| 2-3-4-5 | 1 | 6 | 3.00 | 0.39 | 0.53 | 0.94 | 0.05 | 0.60 |
| 2-3-5-6 | 1 | 4 | 2.09 | 0.22 | 0.21 | 0.50 | 0.04 | 0.40 |
| 2-4-5-6 | 1 | 3 | 4.32 | 0.51 | 0.33 | 0.47 | 0.03 | 0.52 |
| average | | | 4.06 | 0.42 | 0.42 | 0.60 | 0.05 | 0.60 |

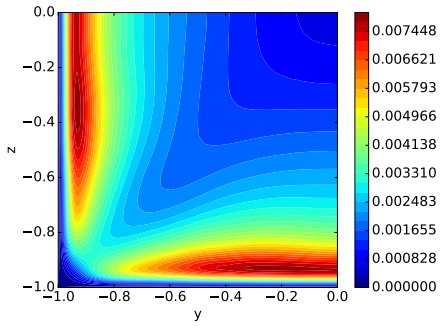
In this session it can be observed that the methodology based on the literature was able to correct very efficiently the tensor field of the turbulent stresses. From the Fig. 7.2 to the Fig. 7.13, it is possible to observe that all the tensorial components were well corrected by the neural network.




(a) $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_x$ RANS.

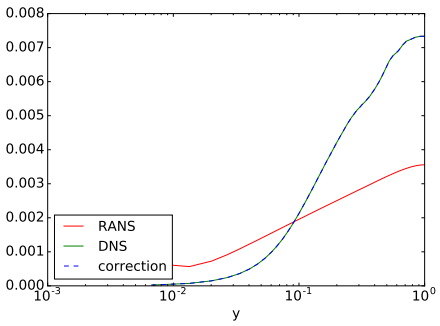



(b) $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_x$ DNS.

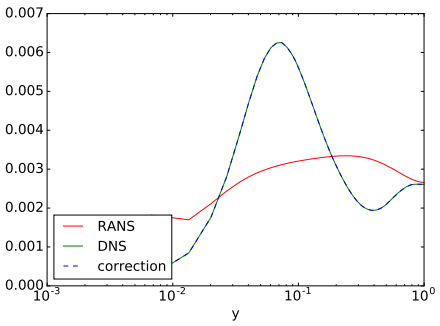



(c) $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_x$ corrected.

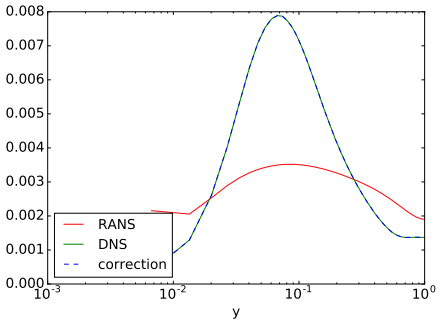
Figure 7.2: $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_x$.



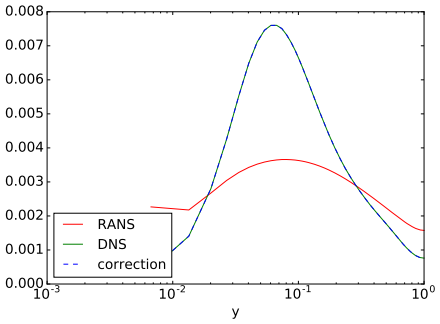

(a) $z = -0.95$.




(b) $z = -0.65$.

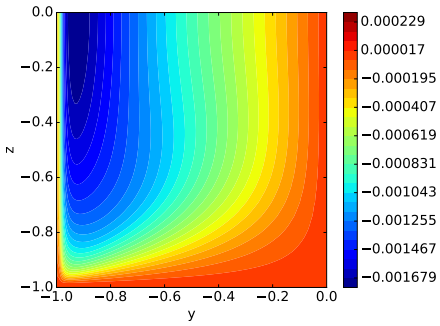



(c) $z = -0.35$.

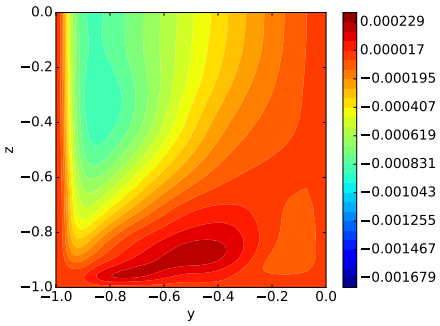



(d) $z = -0.05$.

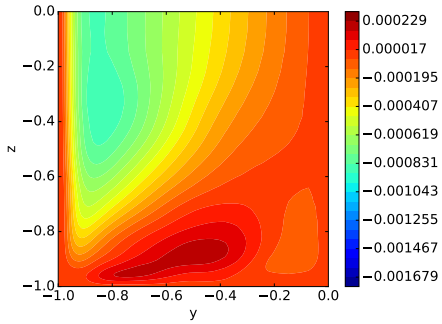
Figure 7.3: $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_x$ component samples.




(a) $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_y$ RANS.

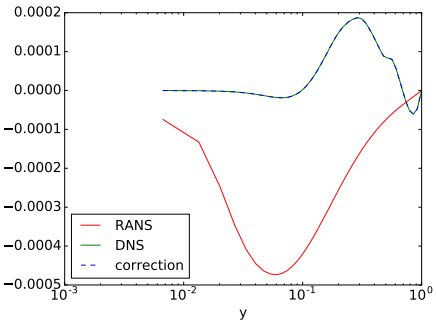



(b) $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_y$ DNS.

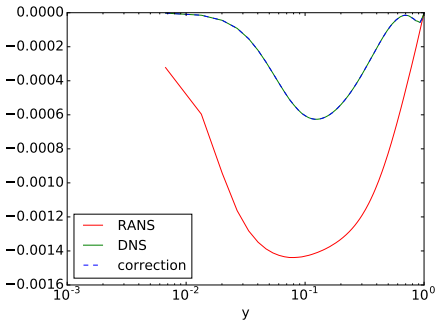



(c) $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_y$ corrected.

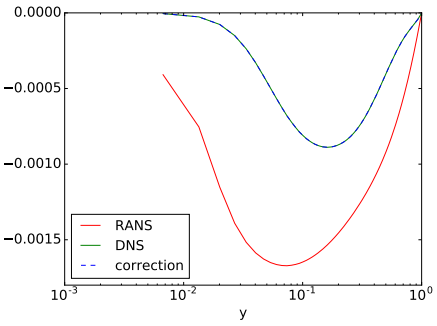
Figure 7.4: $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_y$.



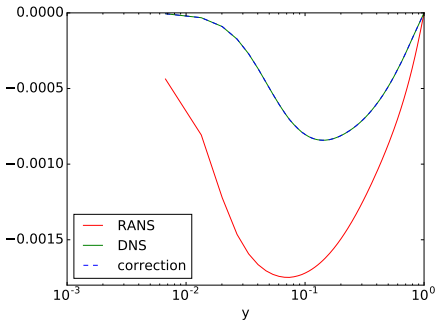

(a) $z = -0.95$.




(b) $z = -0.65$.

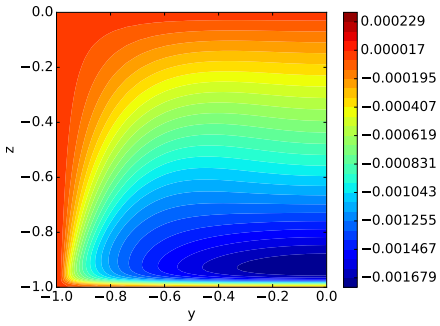



(c) $z = -0.35$.

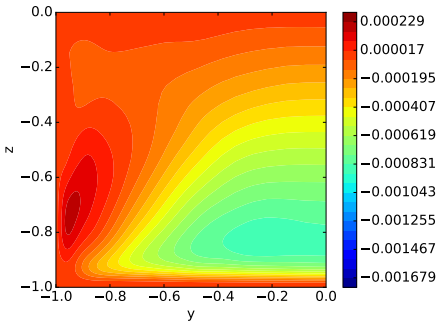



(d) $z = -0.05$.

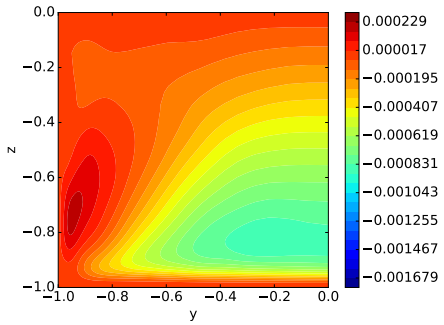
Figure 7.5: $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_y$ component samples.




(a) $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ RANS.

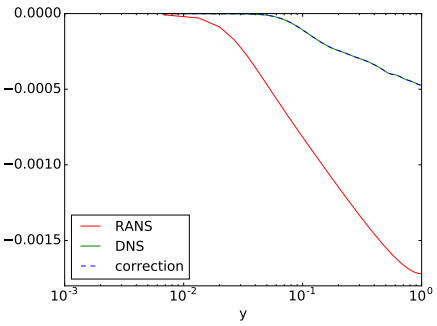



(b) $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ DNS.

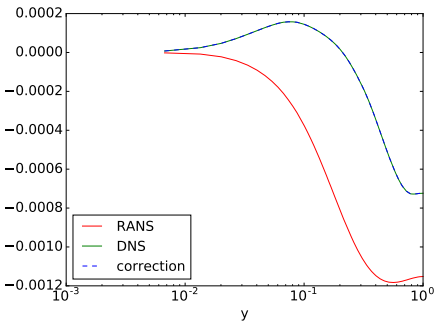



(c) $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ corrected.

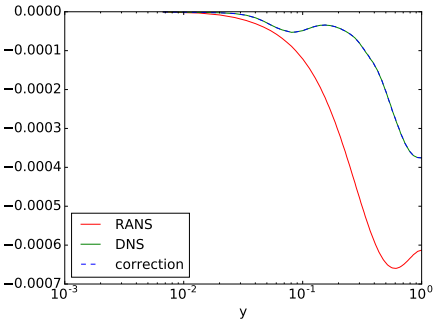
Figure 7.6: $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$.



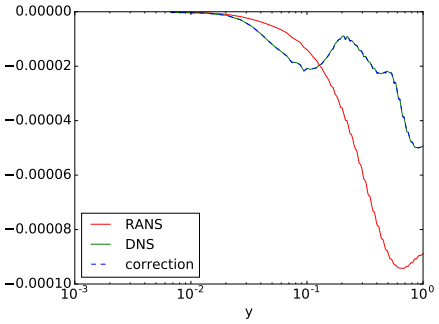

(a) $z = -0.95$.




(b) $z = -0.65$.

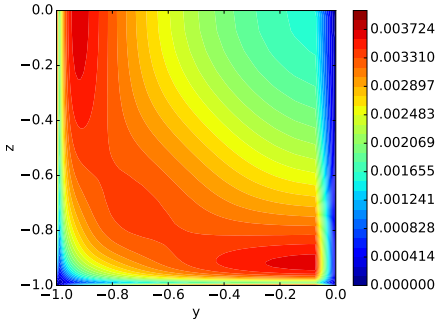



(c) $z = -0.35$.

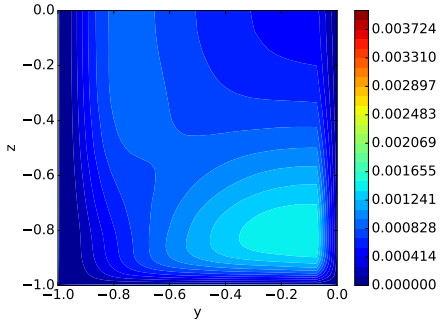



(d) $z = -0.05$.

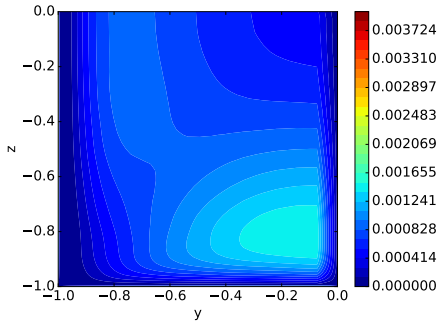
Figure 7.7: $\hat{\mathbf{e}}_x \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ component samples.




(a) $\hat{\mathbf{e}}_y \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_y$ RANS.

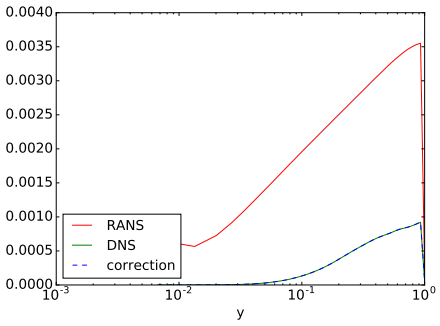



(b) $\hat{\mathbf{e}}_y \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_y$ DNS.

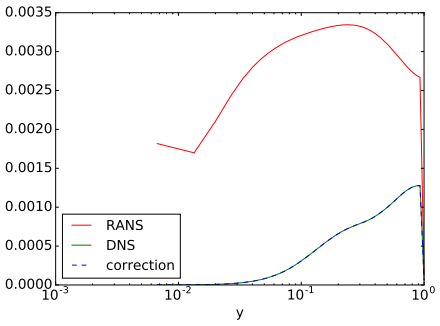



(c) $\hat{\mathbf{e}}_y \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_y$ corrected.

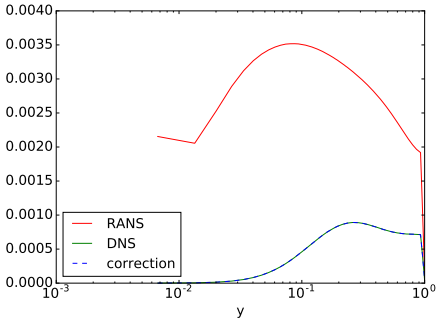
Figure 7.8: $\hat{\mathbf{e}}_y \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_y$.



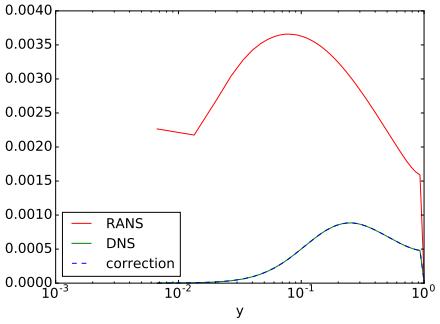

(a) $z = -0.95$.




(b) $z = -0.65$.

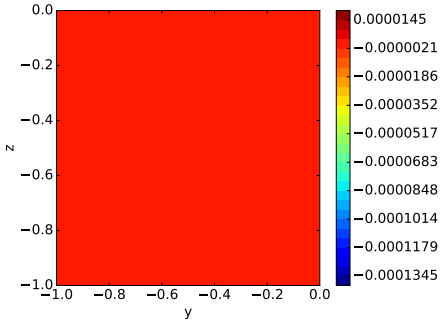



(c) $z = -0.35$.

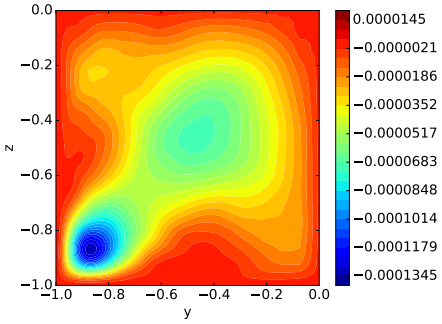



(d) $z = -0.05$.

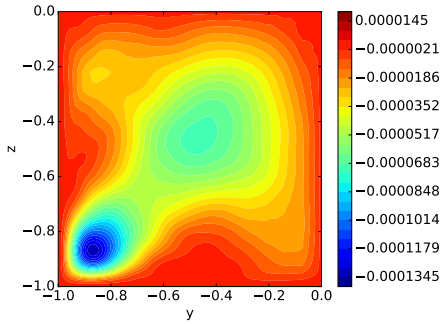
Figure 7.9: $\hat{\mathbf{e}}_y \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_y$ component samples.




(a) $\hat{\mathbf{e}}_y \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ RANS.

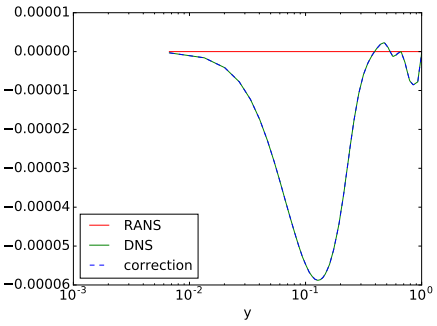



(b) $\hat{\mathbf{e}}_y \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ DNS.

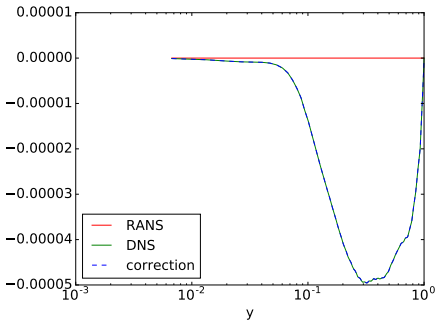



(c) $\hat{\mathbf{e}}_y \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ corrected.

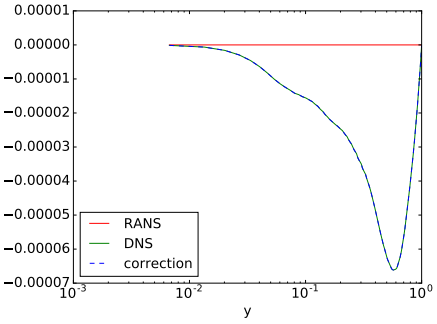
Figure 7.10: $\hat{\mathbf{e}}_y \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$.



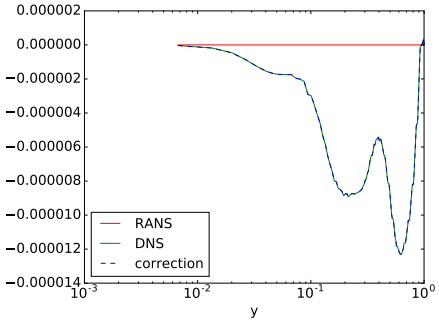

(a) $z = -0.95$.




(b) $z = -0.65$.

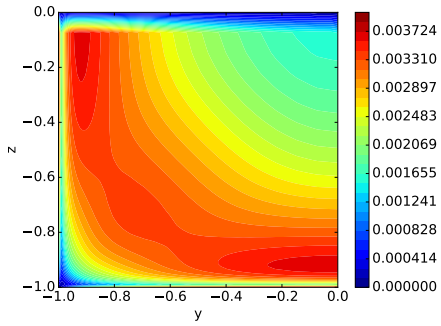



(c) $z = -0.35$.

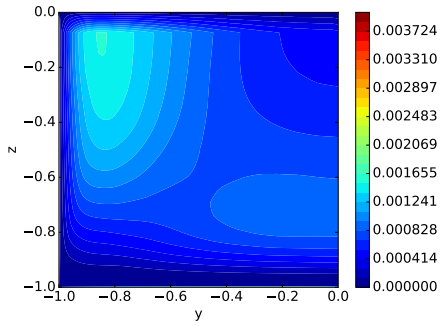



(d) $z = -0.05$.

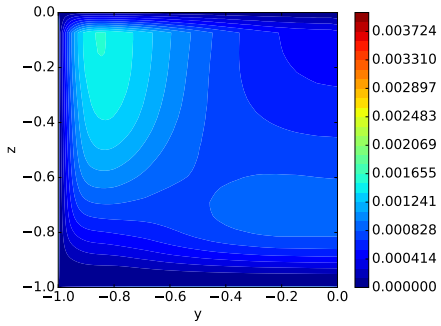
Figure 7.11: $\hat{\mathbf{e}}_y \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ component samples.




(a) $\hat{\mathbf{e}}_z \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ RANS.

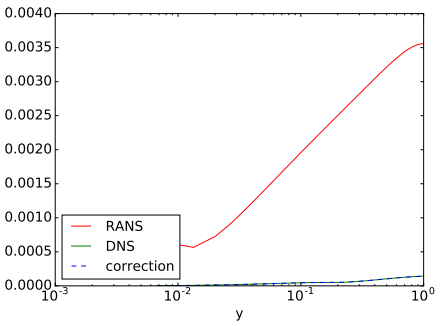



(b) $\hat{\mathbf{e}}_z \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ DNS.

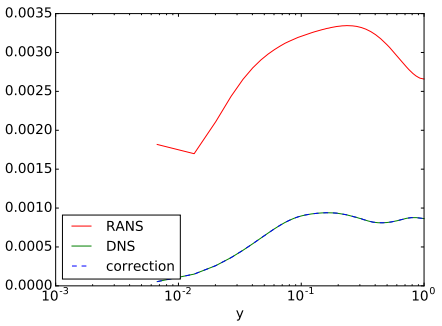



(c) $\hat{\mathbf{e}}_z \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ corrected.

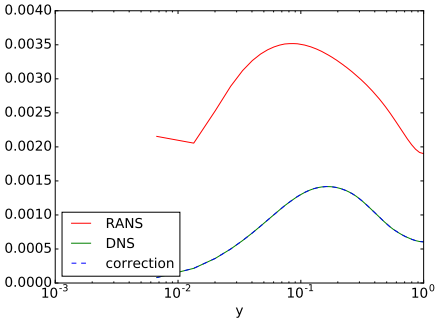
Figure 7.12: $\hat{\mathbf{e}}_z \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$.



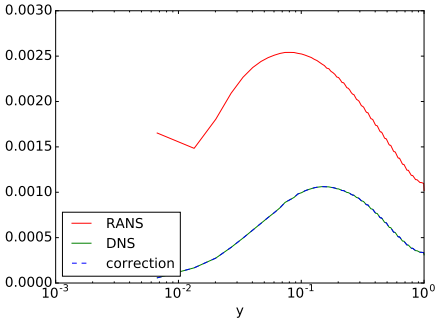

(a) $z = -0.95$.




(b) $z = -0.65$.




(c) $z = -0.35$.




(d) $z = -0.05$.

Figure 7.13: $\hat{\mathbf{e}}_z \cdot \mathbf{R} \cdot \hat{\mathbf{e}}_z$ component samples.

## 7.2 Correction using the t vector as a target

Comparing to the correction of the Reynolds tensor field, the errors obtained by the proposed technique reach higher percentage values. As occurred in the correction of **R**, the largest error is associated with the main direction of the flow, 16.19%. In the secondary directions, the error was of the order of 2%, Tab. 7.2.

Table 7.2: **t** cross-validation errors (variation coefficients described in Eq. 7.3) [%].

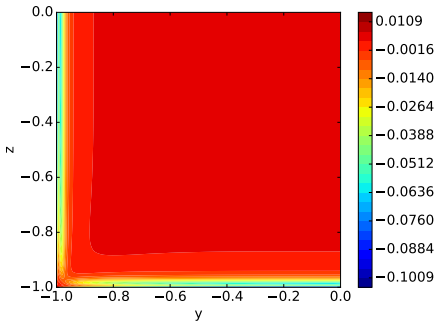
| train | val | test | $\mathbf{t} \cdot \hat{\mathbf{e}}_x$ | $\mathbf{t} \cdot \hat{\mathbf{e}}_y$ | $\mathbf{t} \cdot \hat{\mathbf{e}}_z$ |
|---|---|---|---|---|---|
| 1-2-4-6 | 3 | 5 | 13.15 | 1.37 | 1.14 |
| 1-2-3-4 | 5 | 6 | 26.39 | 3.02 | 3.65 |
| 3-4-5-6 | 1 | 2 | 13.24 | 1.11 | 1.57 |
| 1-4-5-6 | 3 | 2 | 13.76 | 1.04 | 1.09 |
| 1-2-5-6 | 3 | 4 | 11.85 | 1.01 | 1.54 |
| 1-3-4-5 | 2 | 6 | 20.46 | 2.44 | 2.39 |
| 1-4-5-6 | 2 | 3 | 15.39 | 0.98 | 1.09 |
| 2-3-4-5 | 1 | 6 | 18.56 | 2.57 | 2.43 |
| 2-3-5-6 | 1 | 4 | 11.02 | 1.10 | 0.97 |
| 2-4-5-6 | 1 | 3 | 18.05 | 2.01 | 4.06 |
| average | | | 16.19 | 1.66 | 1.99 |

The field is well corrected in the main direction of the flow, as can be seen in Figs. (7.14), (7.15),

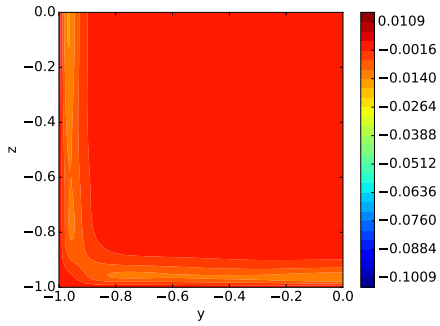
In the $y$ direction, the field is also well corrected. Rebuilding the existing structures near the bottom solid wall from a null RANS field, Fig. (7.16).

However, in the Fig. (7.17) it is observed that the NN generates a result as it moves away from the solid wall, towards the center of the flow.

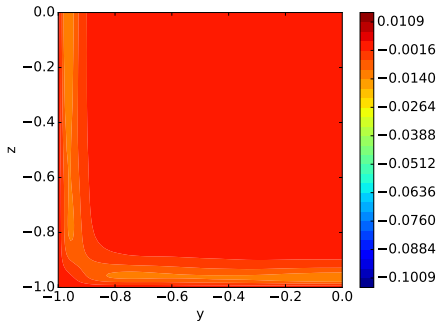
Due to the symmetry of the flow, a behavior similar to that observed in $y$ is seen in the $z$-direction. The field is reconstructed from a null RANS field, and the correction worsens in regions far from the wall, Figs. (7.18) and (7.19).




(a) $\mathbf{t} \cdot \hat{\mathbf{e}}_x$ RANS.

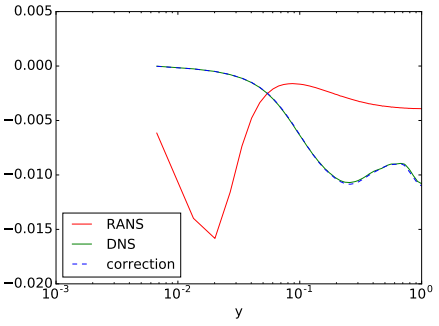



(b) $\mathbf{t} \cdot \hat{\mathbf{e}}_x$ DNS.

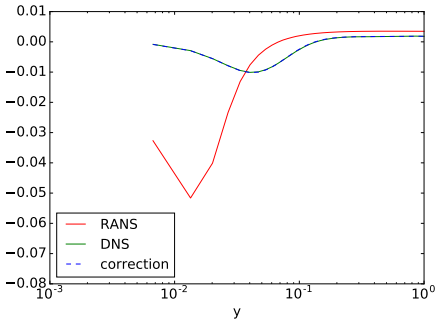



(c) $\mathbf{t} \cdot \hat{\mathbf{e}}_x$ corrected.

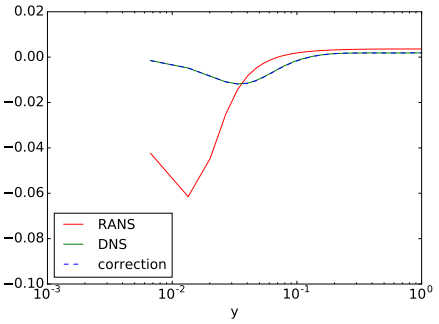
Figure 7.14: $\mathbf{t} \cdot \hat{\mathbf{e}}_x$.



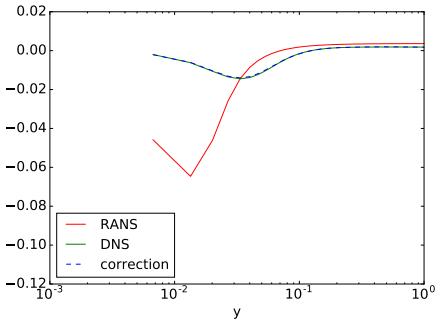

(a) $z = -0.95$.




(b) $z = -0.65$.

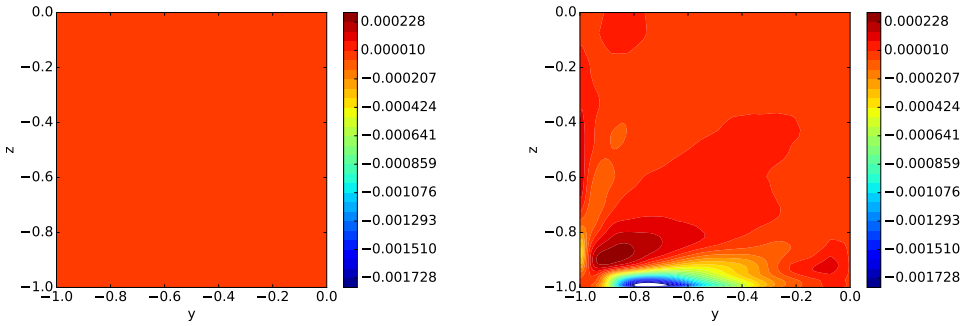



(c) $z = -0.35$.




(d) $z = -0.05$.

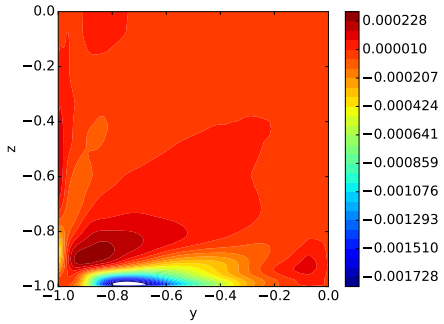
Figure 7.15: $\mathbf{t} \cdot \hat{\mathbf{e}}_x$ component samples.




(a) $\mathbf{t} \cdot \hat{\mathbf{e}}_y$ RANS.

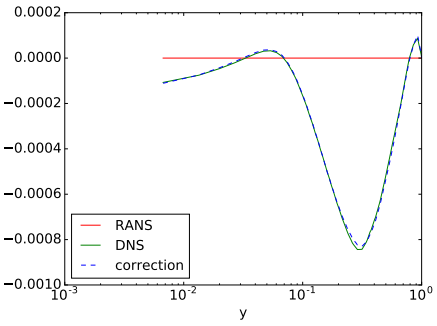



(b) $\mathbf{t} \cdot \hat{\mathbf{e}}_y$ DNS.

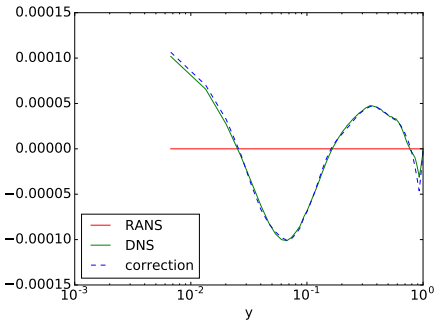



(c) $\mathbf{t} \cdot \hat{\mathbf{e}}_y$ corrected.

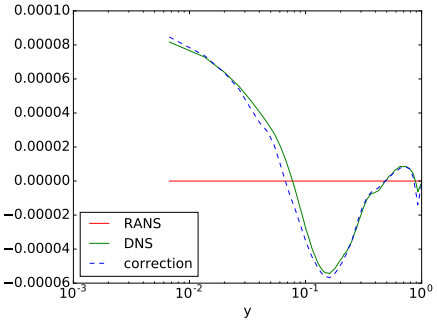
Figure 7.16: $\mathbf{t} \cdot \hat{\mathbf{e}}_y$ component.



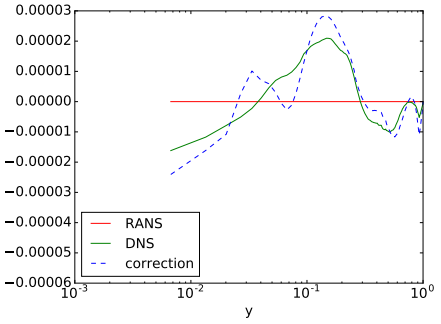

(a) $z = -0.95$.




(b) $z = -0.65$.

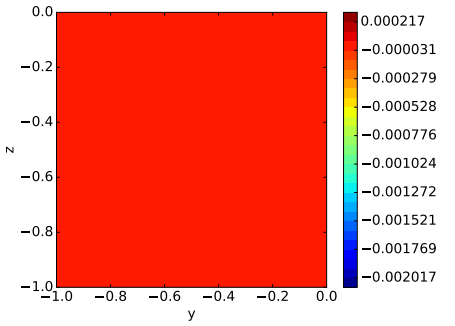



(c) $z = -0.35$.

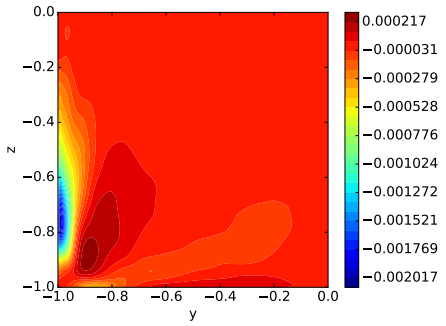



(d) $z = -0.05$.

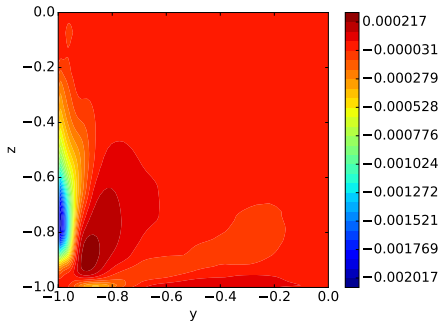
Figure 7.17: $\mathbf{t} \cdot \hat{\mathbf{e}}_y$ component samples.




(a) $\mathbf{t} \cdot \hat{\mathbf{e}}_z$ RANS.

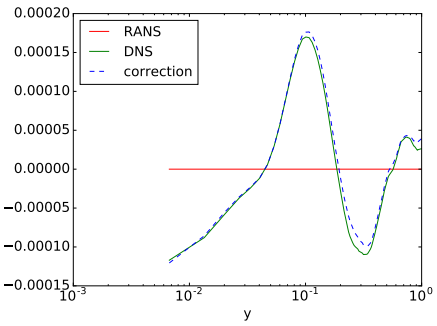



(b) $\mathbf{t} \cdot \hat{\mathbf{e}}_z$ DNS.

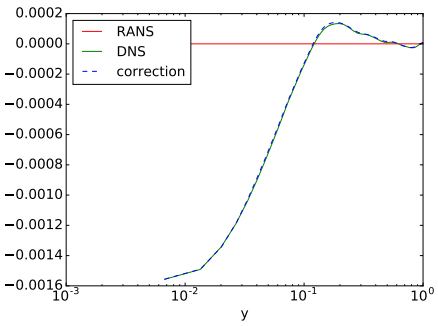



(c) $\mathbf{t} \cdot \hat{\mathbf{e}}_z$ corrected.

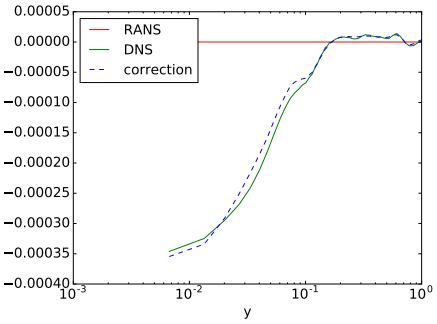
Figure 7.18: $\mathbf{t} \cdot \hat{\mathbf{e}}_z$ component.



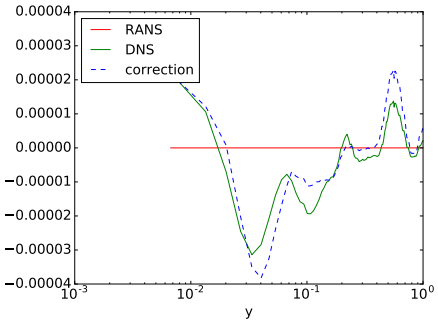

(a) $z = -0.95$.




(b) $z = -0.65$.




(c) $z = -0.35$.

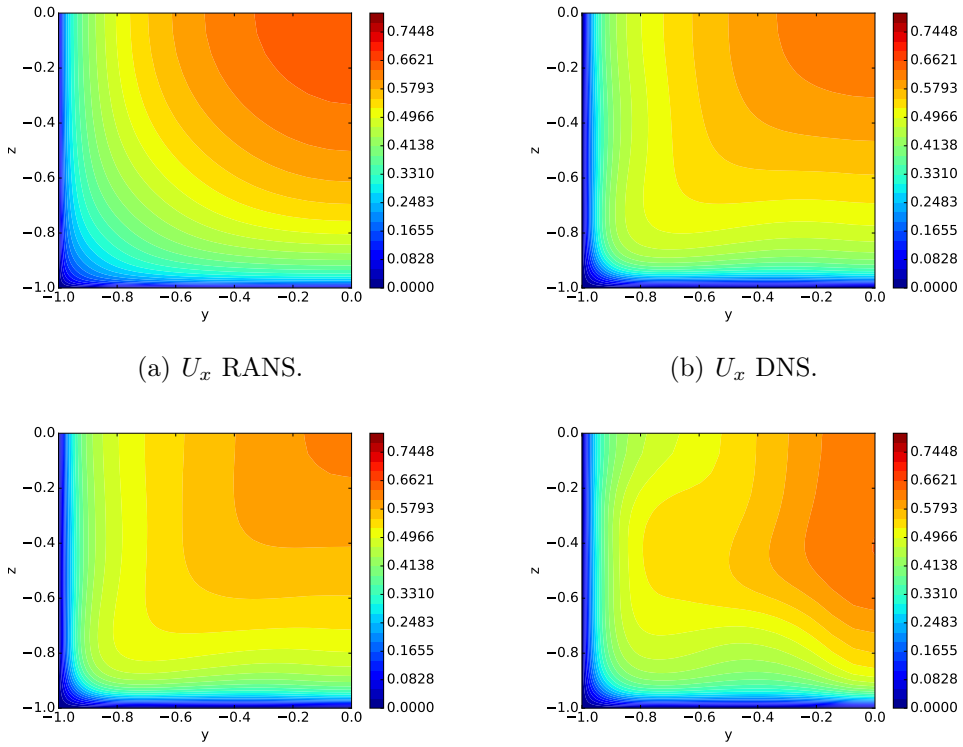



(d) $z = -0.05$.

Figure 7.19: $\mathbf{t} \cdot \hat{\mathbf{e}}_z$ component samples.

## 7.3 Velocity and pressure corrections

The correction of the velocity field in the main direction is more accurate when correcting the field **t** than when correcting the field **R**, as can be seen in Figs. (7.20) and (7.21). This is due to the uncertainties propagated from the DNS database itself, as already mentioned.




(a) $U_x$ RANS.

(b) $U_x$ DNS.

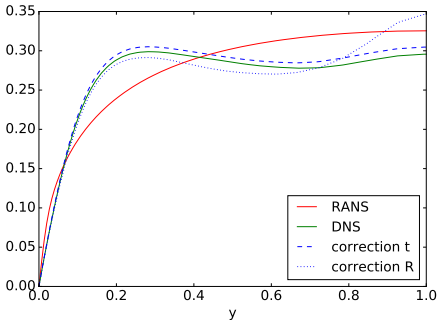
(c) $U_x$ corrected by **t**.

(d) $U_x$ corrected by **R**.

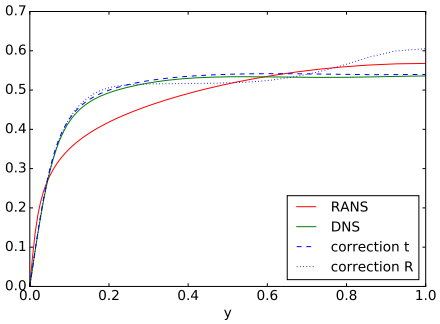
Figure 7.20: $U_x$ component.

Both techniques reconstruct the structures observed in the DNS field, from a null RANS field, as shown in Fig. (7.22), in the $y$-direction. However, the **R** correction is closer to DNS in the entire domain, as shown in Fig. (7.23). The same is seen in the $z$-direction, as can be seen in Figs. (7.24) and (7.25).

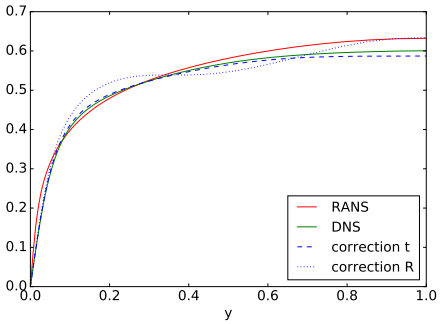
The secondary flow is plotted in Fig. (7.26), both techniques reconstruct the recirculation cells that are not seen in the RANS result, but the correction via **R** is closer to the DNS result, as has already been said.



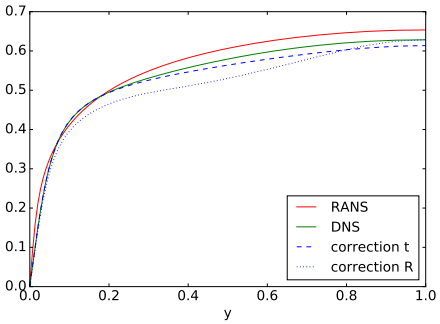

(a) $z = -0.95$.




(b) $z = -0.65$.

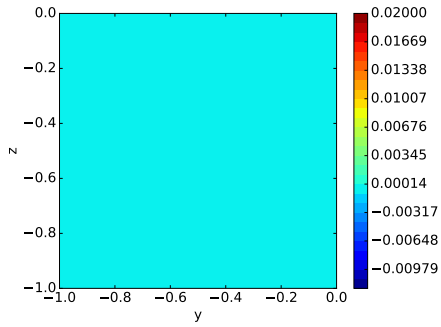



(c) $z = -0.35$.

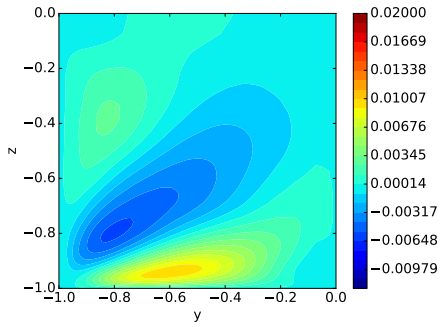



(d) $z = -0.05$.

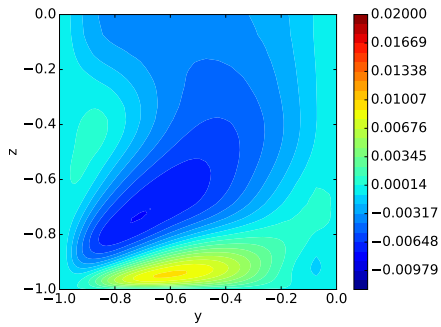
Figure 7.21: $U_x$ component samples.



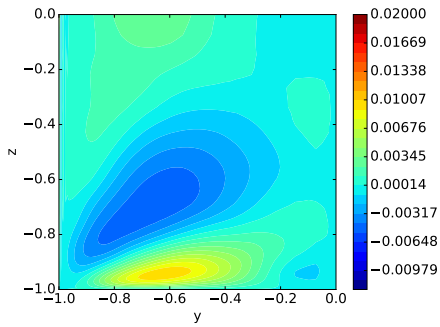

(a) $U_y$ RANS.




(b) $U_y$ DNS.

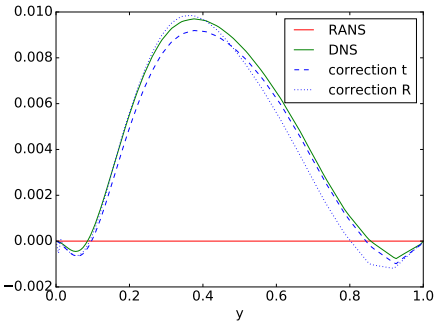



(c) $U_y$ corrected by $\mathbf{t}$.

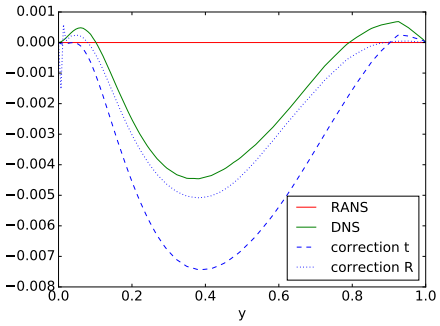



(d) $U_y$ corrected by $\mathbf{R}$.

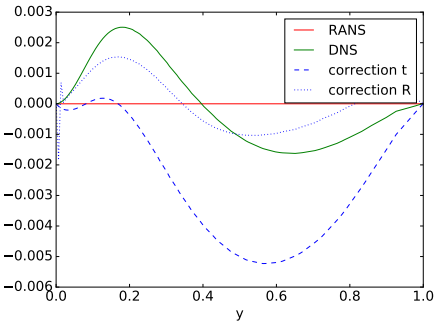
Figure 7.22: $U_y$ component.



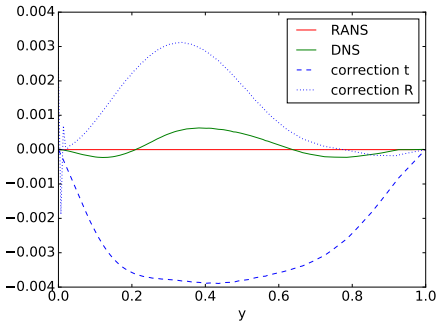

(a) $z = -0.95$.




(b) $z = -0.65$.

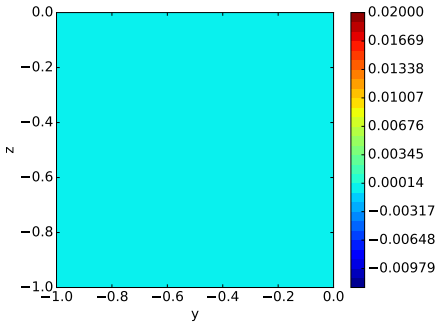



(c) $z = -0.35$.

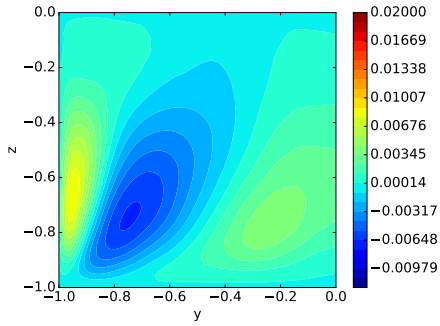



(d) $z = -0.05$.

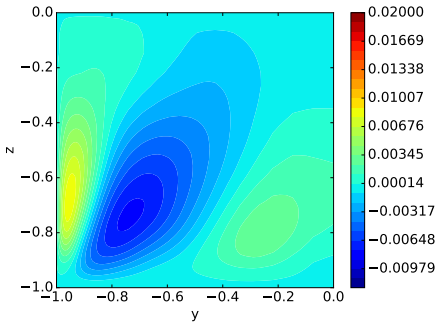
Figure 7.23: $U_y$ component samples.



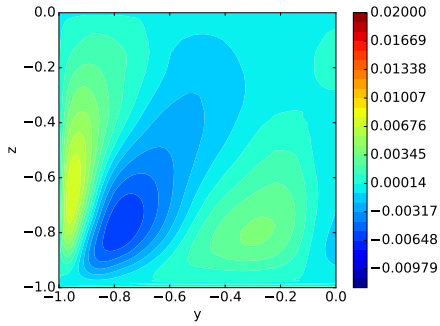

(a) $U_z$ RANS.




(b) $U_z$ DNS.

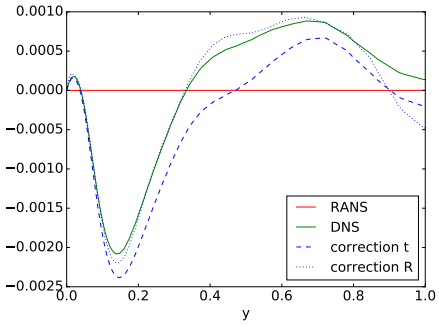



(c) $U_z$ corrected by $\mathbf{t}$.

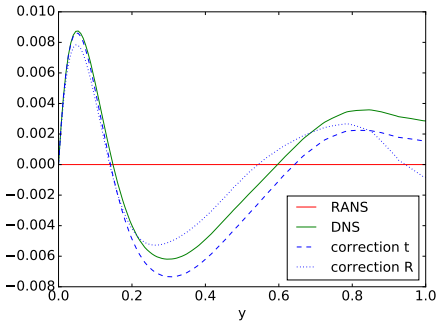



(d) $U_z$ corrected by $\mathbf{R}$.

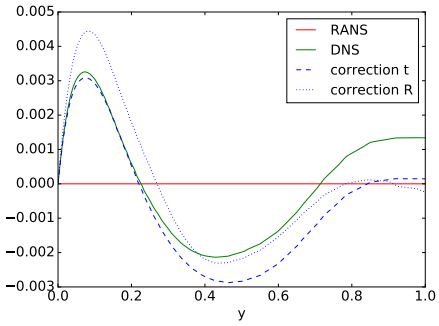
Figure 7.24: $U_z$ component.



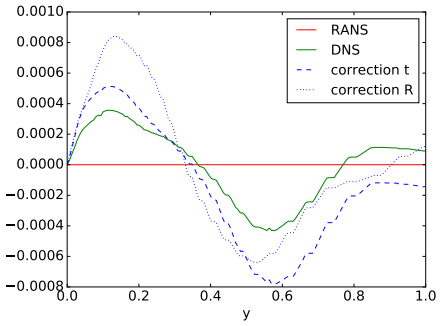

(a) $z = -0.95$.




(b) $z = -0.65$.

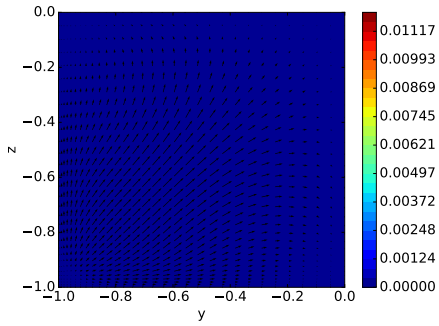



(c) $z = -0.35$.

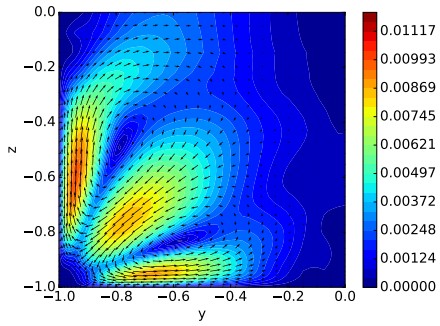



(d) $z = -0.05$.

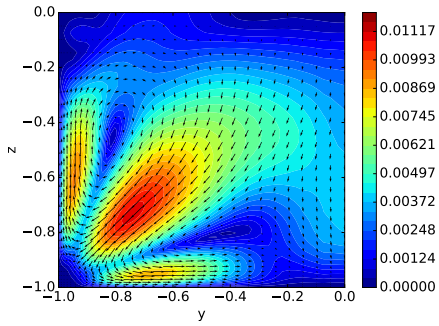
Figure 7.25: $U_z$ component samples.



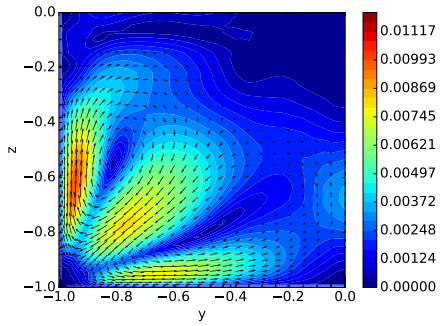

(a) RANS.




(b) DNS.




(c) Corrected by $\mathbf{t}$.




(d) Corrected by $\mathbf{R}$.

Figure 7.26: Secondary flow magnitude contour levels ($\sqrt{U_y^2 + U_z^2}$), and stream lines.

Both techniques have well corrected the pressure gradient that feeds the flow, Fig. (7.27).

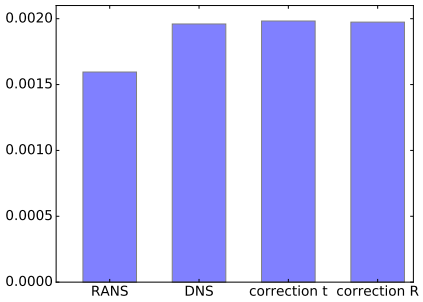



Figure 7.27: $-\frac{1}{\rho}\frac{\partial p}{\partial x}$.

# Chapter 8

# Conclusions

It is concluded that, in fact, it is problematic to target the Reynolds stress tensor field provided by DNS databases because they have uncertainties associated with poor convergence of the second order statistics they provide. Such uncertainties will then always be propagated to the average velocity field, no matter how well designed the neural networks.

The main direction flow component is very well corrected by $\mathbf{t}$, but the same does not occur with the correction by $\mathbf{R}$. This is due to the propagation of the uncertainties mentioned above.

The secondary flow that is not captured by the RANS model, is recovered by both techniques. However, the $\mathbf{R}$-correction arrives closer to the DNS result than via $\mathbf{t}$.

The pressure gradient that feeds the flow is corrected by both techniques.

The non-persistence-of-straining tensor ($\mathbf{P}$) was successful as input of the neural network that corrects $\mathbf{R}$ and $\mathbf{t}$. Showing its your application is promising.

In general, the proposed methodology surpasses the methodology available in the literature, because its primary flow correction is much better, and the secondary flow and pressure gradient corrections were similar to the results obtained in the literature based methodology. That shows the potential of machine learning in improving the turbulent flows simulations.

# Chapter 9

# Future works

The next step of the research is to add more flow patterns to evaluate the abstraction power of the neural network.

When the divergence of the Reynolds stress tensor was chosen to be corrected instead of the tensor itself, the problem of the the DNS databases uncertainties was solved. But, at the same time, the universality of the learning capability decreased, because the tensorial entity is more general than its divergence. In order to increase the universality of the machine learning correction, a new methodology will be proposed to correct the Helmholtz decomposition of the $\boldsymbol{\nabla} \cdot \mathbf{R}$ field,

$$\boldsymbol{\nabla} \cdot \mathbf{R} = \boldsymbol{\nabla}\phi + \boldsymbol{\nabla} \times \boldsymbol{\psi}.$$

The potential fields $\phi$ and $\boldsymbol{\psi}$ obtained from the decomposition will be corrected.

# Bibliography


[1] DAVIDSON, P. A. *Turbulence: An Introduction for Scientists and Engineers.* USA: Oxford University Press, 2004.

[2] KUNDU, P. K., COHEN, I. M., DOWLING, D. R. *Fluid Mechanics.* USA: Academic Press, 2012.

[3] `https://i.ytimg.com/vi/nauLXHIkvf0/maxresdefault.jpg`. Accessed: 2016-12-26.

[4] WILCOX, D. C. *Turbulence Modeling for CFD.* USA: DCW Industries, 2006.

[5] BREDBERG, J. *On the Wall Boundary Condition for Turbulence Models.* Sweden:Göteborg, 2000.

[6] `http://www.newworldencyclopedia.org/entry/File:Neuron.svg`. Accessed: 2016-08-14.

[7] TRACEY, B., DURAISAMY, K., ALONSO, J. J. "Application of supervised learning to quantify uncertainties in turbulence and combustion modeling." *AIAA Aerospace Sciences Meeting*, v. 0259, 2013.

[8] TRACEY, B., DURAISAMY, K., ALONSO, J. J. "A machine learning strategy to assist turbulence model development." *AIAA Aerospace Sciences Meeting*, v. 1287, 2015.

[9] LING, J., RUIZ, A., LACANZE, G., etal. "Uncertainty analysis and data-driven model advances for a jet-in-crossflow." *ASME Turbo Expo*, 2016.

[10] WANG, J. X., WU, J. L., IACCARINO, G., etal. "Physics-Informed Machine Learning for Predictive Turbulence Modeling:Towards a Complete Framework." 2016.

[11] JONES, W. P., LAUNDER, B. E. "The prediction of laminarization with a two-equation model of turbulence." *International Journal of Heat and Mass Transfer*, pp. 301–314, 1972.

[12] BOUSSINESQ, J. "Essai sur la theorie des eaux curantes", *Mem. Pres. Acad. Sci.*, , n. XXIII, pp. 46, 1877.

[13] HOFFMAN, P. H., MUCK, K. C., BRADSHAW, P. "The effect of a concave surface curvature on turbulent boundary layers", *J.F.M*, v. 161, pp. 371, 1985.

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

[15] MILANO, M., KOUMOUTSAKOS, P. "Neural network modeling for near wall turbulent flow", *J. Comput. Phys*, v. 182, pp. 1–26, 2002.

[16] ZHANG, Z. J., DURAISAMY, K. "Machine learning methods for data-driven turbulence modeling." *AIAA Computational Fluid Dynamics*, v. 2460, 2015.

[17] LING, J., JONES, R., TEMPLETON, J. "Machine learning strategies for systems with invariance properties." *J. Comput. Phys.*, v. 318, pp. 22–35, 2016.

[18] LING, J., KURZAWSKI, A., TEMPLETON, J. "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance." *J. Fluid Mech.*, v. 807, pp. 155–166, 2016.

[19] POPE, S. B. "A more general effective-viscosity hypothesis." *J. Fluid Mech.*, v. 72, pp. 311–340, 1975.

[20] THOMPSON, R. L., SAMPAIO, L. E. B., ALVES, F. A. V. B., etal. "A methodology to evaluate statistical errors in DNS data of plane channel flows." *Computers and Fluids*, v. 130, pp. 1–7, 2016.

[21] WILCOX, D. C. *Turbulence Modeling for CFD.* USA: DCW Industries, 1993.

[22] TENNEKES, H., LUMLEY, J. L. *A First Course in Turbulence.* The MIT Press, 1972.

[23] RICHARDSON, L. F. "Weather Prediction by Numerical Process", *Cambridge:Cambridge University Press*, 1922.

[24] KOLMOGOROV, A. N. "The local structure of turbulence in incompressible viscous fluids at very large reynolds number", *Dolk. Akad Nauk. SSSR*, 1941.

[25] POPE, S. B. *Turbulent Flows.* USA: Cambridge University Press, 2000.

[26] FERZIGER, J. H., PERIC, M. *Computational Methods for Fluid Dynamics.* Germany: Springer, 2002.

[27] DESCHAMPS, C. J. "Modelos Algébricos e Diferencais". In: Freire, A. P. S., Menut, P. P. H. M., Su, J. (Eds.), *Turbulência*, ABCM, cap. 3, ABCM, 2002.

[28] SCHMITT, F. G. "About boussinesq's tubulent viscosity hypothesis: historical remarks and a direct evaluation of its validity." *Comptes Rendus Mecanique*, pp. 617–627, 2007.

[29] VERSTEEG, H. K., MALALASEKERA, W. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method.* London, UK: Pearson, 2007.

[30] LAUNDER, B., SPALDING, D. "The Numerical Computation of Turbulent Flow Computer Methods", v. 3, pp. 269–289, 03 1974.

[31] MARSLAND, S. "Machine Learning - An Algorithmic Perspective." *Florida, CRC Press*, v. 1, 2009.

[32] LEVADA, M. M. O., FIERI, W. J., PIVESSO, M. S. G. *Apontamentos Teóricos de Citologia, Histologia e Embriologia.* Catálise Editora, 1996.

[33] MCCULLOCH, W. S., PITTS, W. H. "A Logical Calculus of the ideas immanent in nervous activity." *Bulletin of Mathematical Biophysics*, v. 5, pp. 115–133, 1943.

[34] CIBENKO, G. "Approximation by Superpositions of a Sigmoidal Function." *Mathematics of Control, Signals and Systems*, v. 2, pp. 303–314, 1989.

[35] TIELEMAN, T., HINTON, G. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." Coursera Lecture, 2012.

[36] KINGMA, D. P., BA, J. L. "Adam: A Method for Stochastic Optimization." *International Conference for Learning Representations*, 2015.

[37] PINELLI, A., UHLMANN, M., SEKIMOTO, A., etal. "Reynolds number dependence of mean flow structure in square duct turbulence." *J. Fluid Mech*, v. 644, pp. 107–122, 2010.

[38] THOMPSON, R. L., MENDES, P. R. S. "Persistence of straining and flow classification", *International Jornal of Engeneering Sciense*, , n. 43, pp. 79–75, 2005.

[39] THOMPSON, R. L. "Some perspectives on the dynamic history of a material element." *International Journal of Engineering Science*, v. 46, pp. 224–249, 2008.

[40] CHOLLET, F., OTHERS. "Keras". `https://github.com/fchollet/keras`, 2015.

[41] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., etal. "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, v. 12, pp. 2825–2830, 2011.