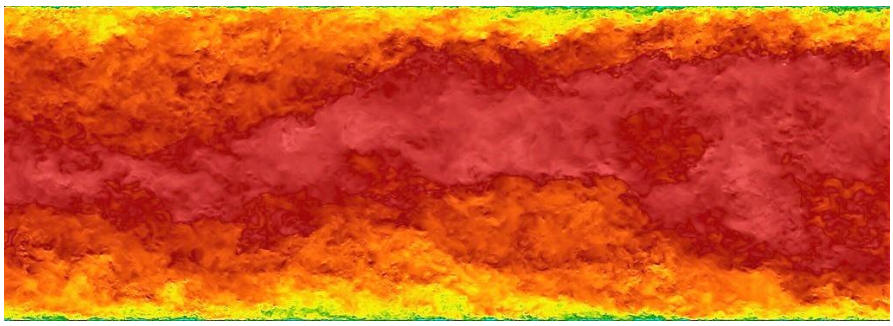
Research Internship (PRe)

Field of Study : Applied Mathematics
Scholar Year : 2020-2021

Neural Networks for Turbulence Modeling

Solving Reynolds-Averaged Navier-Stokes equations in Turbulent Channel Flow



Author : Félix Chavelli Promotion : 2022

ENSTA Paris Tutor : Zacharie Ales
Host Organism Tutor : David Sondak

Internship conducted from 17 / 05 / 2021 to 31 / 07 / 2021

Host organism : Harvard University
Adress : 20 Oxford street, 02138 Cambridge Massachusetts - United States

Acknowledgements

First of all I would like to deeply thank **Pavlos Protopapas**, Scientific Program Director and Lecturer at Institute for Applied Computational Science (IACS) for welcoming me to the laboratory and giving his precious time to mentor me during a part of the internship.

I would also like to express my gratitude to my internship supervisor **David Sondak**, Lecturer at IACS for allowing me to do this internship and for being at my side and available, despite the physical and temporal distance, always with patience and good humor.

I would finaly like to thank my colleagues **Haitz Saez de Ocariz Borde** and **Joao Esteves** for their answers, advice and availability throughout the project.

Résumé

La modélisation de la turbulence est nécessaire dans de nombreux domaines de l'ingénierie. Dans ce contexte, les équations de Navier-Stokes à moyenne de Reynolds ou *Reynolds-Averaged Navier–Stokes* (RANS) permettent de simplifier les équations et sont donc largement utilisées. Cependant, leur résolution nécessite une modélisation précise du tenseur des contraintes de Reynolds anisotrope. De nombreuses approches basées sur les données ont été développées ces dernières années pour résoudre ce problème et remplacer les méthodes traditionnelles n'étant pas fiables dans toutes les configurations d'écoulement. Cette approche par les données peut également être utilisée pour résoudre des équations différentielles. Dans ce projet, les réseaux de neurones sont utilisés à la fois pour modéliser le tenseur anisotrope des contraintes de Reynolds ainsi que pour résoudre les équations RANS associées au *Turbulent Channel Flow*. Différents modèles de fermeture sont ainsi entraînés, comparés et utilisés pour résoudre les équations RANS. Un modèle global exploitant deux réseaux de neurones pour modéliser le tenseur de contraintes de Reynolds anisotrope tout en résolvant les équations RANS est finalement présenté.

Mots clés:

modélisation de la turbulence, équations de Navier-Stokes à moyenne de Reynolds, réseaux de neurones

Abstract

Turbulence modeling is needed in many engineering fields. In this context, the Reynolds-averaged Navier-Stokes (RANS) equations allow to simplify the equations and are thus widely used. However, their solution requires an accurate modeling of the anisotropic Reynolds stress tensor. Numerous data-driven approaches have been developed in recent years to solve this problem and replace traditional methods which are not reliable in all flow configuration. This data-driven approach can also be used to solve differential equations. In this project, neural networks are used both to model the anisotropic Reynolds stress tensor and to solve the RANS equations associated with the turbulent channel flow. Different closure models are thus trained, compared and used to solve the RANS equations. A global model exploiting two neural networks to model the anisotropic Reynolds stress tensor while solving the RANS equations is finally presented.

Keywords:

turbulence modeling, Reynolds-averaged Navier-Stokes, neural networks

Contents

Introduction

Turbulence is involved in many practical fluid mechanics problems. Researchers and engineers therefore need turbulence models that are both reliable and efficient but also computationally tractable. Direct numerical simulation (DNS) allows very accurate modeling but is generally impossible to implement at the Reynolds numbers of fluid systems of interest. Therefore, simpler and more practical models have been developed. Reynolds-averaged Navier-Stokes (RANS) models are among the most popular. These models attempt to determine the average of the flow fields and the fluctuations around the average. However, these fluctuations require the modeling of the Reynolds stress tensor. The traditional models encountering difficulties to model this tensor, the last few years have been marked by the development of data-driven approaches (Ling, et al., 2016), (Wang, et al., 2017), (Fang, et al., 2019), (Sáez de Ocáriz Borde, et al., 2021). Neural networks are also of recent interest in the resolution of differential equations (Mattheakis, et al., 2020). In particular, some solutions of the Navier-Stokes equations, governing the dynamics of many fluids, can be approximated by neural networks.

The work presented in this report attempts to use neural networks to solve the RANS equations and model the associated Reynolds anisotropy tensor, in the context of one-dimensional Turbulent Channel Flow. This project was conducted within the StellarDNN team of the Institute for Applied Computational Science at Harvard University. I worked under the supervision of David Sondak, lecturer at the university, and Pavlos Protopapas, director of the team, while exchanging with other international interns. For 2 months and a half, while working on fluid mechanics, neural networks and differential equations, I had the exciting opportunity to bring together physics, computer science and applied mathematics. This project finally allowed me, step by step, using different previous research works, to build a resolution model.

The report is decomposed as follows. The first part presents the studied flow, the RANS equations as well as the neural networks and the DNS database used. In the second part, different closure models based on neural networks are trained and compared using the DNS database. The third part deals with the resolution of differential equations by neural network and their application to the population and then to the RANS equation. The fourth part is dedicated to the development of a complete model performing the resolution of the RANS equations in parallel with the modeling of the anisotropic Reynolds tensor. Finally, the last section deals with the conclusion and future research.

Chapter 1 – Background and methodology

1.1 Turbulent channel flow

1.1.1 Flow and study variables

The flow studied throughout the report is of a fluid confined between two infinite parallel plates in the $x - z$ plane and located at $y = 0$ et à $y = 2h$. A known pressure gradient directs the flow in the streamwise (x) direction. The flow is noted $\mathbf{u} = (u, v, w)$ where u is the velocity in the streamwise direction, v is velocity the wall-normal (y) direction and w is the spanwise (z) velocity. Since the flow of study is statistically one-dimensional and that u only depends on y, $\mathbf{u}$ can be noted $(u(y),0,0)$.

In the following, the friction Reynolds number is noted Re_τ and y^+ gives the non-dimensional distance from the wall. Non-dimensioned quantities will also be used and will be noted with stars. Thus, the spatial dimension y will be given by $y^* = y/h$. Likewise, the mean velocity becomes $\bar{u}^*$ and the Reynolds anisotropic tensor becomes $= a_{uv}^*$**.** Details about units are provided in the Appendix[i].

1.1.2 Reynolds-Averaged Navier-Stokes (RANS) equations

Since the equations governing turbulence are most of the time impossible to solve, numerical simulations are commonly used. Given the computational cost of these simulations, it is usual to solve only simplified equations considering the mean flow and to leave aside the flows at small scales. The most common way to obtain these new equations is the Reynolds-Averaged method. This approach is based on a decomposition of the velocity into two components, the average velocity $\overline{\mathbf{u}}$ and the fluctuation $\mathbf{u}'$ around it. Then, $\mathbf{u} = \overline{\mathbf{u}} + \mathbf{u}'$, the objective being to obtain the average velocity $\overline{\mathbf{u}}$. The averaging operation, applied to the Navier-Stokes equations, allows to obtain the RANS equations. Details of the calculations are given in the appendix.[ii]

These equations show a new term: $\overline{\mathbf{u}' \otimes \mathbf{u}'}$, the Reynolds stress tensor. The latter must be modeled to close the RANS equations. Instead of modelling directly this tensor, it is usual to focus on the anisotropic Reynolds stress tensor $\mathbf{a} = \overline{\mathbf{u}' \otimes \mathbf{u}'} - (2k/3)\mathbf{I}$, as that is the portion responsible for turbulent transport. The only non-zero component of $\overline{\mathbf{u}}$**,** $\bar{u}(y)$ is thus directly linked to a single component of $\mathbf{a}$, the $u - v$ term, noted a_{uv}.

1.2 Turbulent channel flow dataset

1.2.1 Available data

The data used throughout the report to train and test the different models are derived from direct numerical simulation[1] (DNS) data of a turbulent channel flow from the Oden Institute turbulence file server. These simulations are performed for different friction Reynolds numbers $Re_\tau = [550, 1000, 2000, 5200]$. The original reference (Lee & Moser, 2015) provides details on the simulation. An extract of the dataset for $Re_\tau = 1000$ is shown below.

	y+	y	u+	u	index	du_dy	a_uv	a_uu	a_vv	a_ww	Re_tau
0	0.000000	0.000000	0.001383	0.000069	0	50.051144	0.000000e+00	1.423962e-38	-3.636333e-37	3.493937e-37	1000.512
1	0.002766	0.000003	0.005531	0.000277	1	50.050936	-5.412613e-11	1.750248e-09	-1.605746e-09	-1.445019e-10	1000.512
2	0.013829	0.000014	0.018438	0.000922	2	50.050290	-6.772994e-09	4.377421e-08	-4.010216e-08	-3.672055e-09	1000.512
3	0.038719	0.000039	0.045171	0.002260	3	50.048942	-1.490390e-07	3.435115e-07	-3.136778e-07	-2.983370e-08	1000.512

Figure 1 : *Extract of the dataset for Re_τ = 1000.*

1.2.2 Data exploration

The DNS data provide different flow quantities (velocity and anisotropic Reynolds stress tensors) for each of the considered friction Reynolds numbers. The velocity profiles are studied only for the lower half of the channel, the other half being symmetrical. The velocity will thus appear in abscissa (as a velocity profile) in a concern of physical representation. The figures below represent the anisotropic Reynolds stress tensors and the velocity profile for each friction number.

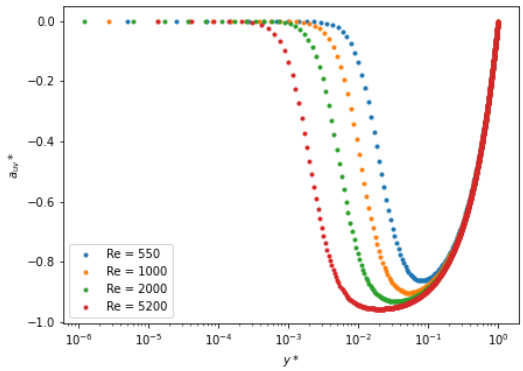


Figure 2 : *Anisotropic Reynolds stress tensor*

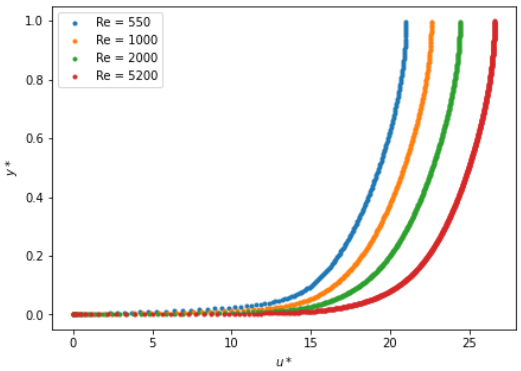


Figure 3 : *Velocity profiles*

[1] Direct numerical simulations (DNS) are numerical simulations in which the Navier-Stokes equations are solved numerically without using any turbulence model. The whole range of spatial and temporal scales of the turbulence must be solved. These simulations are therefore extremely expensive, even at low Reynolds, but they are a reliable source for the development of other models.

1.3 Neural Networks

1.3.1 Multilayer Perceptron

Neural networks are a type of machine learning algorithm used in many domains and are particularly effective for explaining complex non-linear relationships in high-dimensional data contexts. Feedforward neural networks, or multilayer perceptron in particular, are used to approximate functions. Formally, a neural network defines a mapping between an input x and an output y. In the case of MLPs, this mapping corresponds to a composition of functions forming a network of layers and nodes. The first layer of the network corresponds to the input x and the last to the output y. Each layer is composed of nodes and is connected to the next layer according to a set of weights w and bias b to be determined during the training. The layer h_{i+1} is thus given by $wh_i + b$. In the case of a completely connected network, each node of a layer is connected to the set of nodes of the previous layer (without loop or return) and applies an activation function, in general a non-linear smooth transformation σ such that $h_{i+1} = \sigma(wh_i + b)$.

1.3.2 Training process

The training of a neural network consists in comparing its prediction to data thanks to a loss function. An optimization step then uses an algorithm to adjust the weights and biases of the network to minimize the loss function. The algorithms used are usually gradient descent methods, where the parameters of the network are adjusted according to the gradient of the loss function with respect to the parameters and a certain learning rate.

1.3.3 Neural Networks for Fluid Mechanics

The main problems encountered in fluid mechanics concern reduction, modeling, control, sensing and closure. However, it appears that with the right formalism, it is possible to see these tasks as complex optimization problems (high dimensional, nonlinear, non-convex, multi-scale) (Brunton, et al., 2020). Neural networks being particularly efficient to solve this kind of problems, their use is increasingly common.

Chapter 2 – Model for the Anisotropic Reynolds Stress Tensor

2.1 Traditional models

The anisotropic Reynolds stress tensor previously defined must be modeled to close the RANS equations. Many closure models allow to approach its value; the most popular ones are the Eddy viscosity models like the $k-\varepsilon$ or the $k-\omega$. They are indeed simple to implement and fast to simulate. Other models, even simpler, do not require any additional equation resolution. This is the case of the Prandtl and Van Driest models. However, when applied to the interest flow, they remain rather inaccurate. They are nevertheless usefull because they will serve as an element of comparison as well as an initial model in the continuation of the project. The results for the turbulent channel flow are shown below.

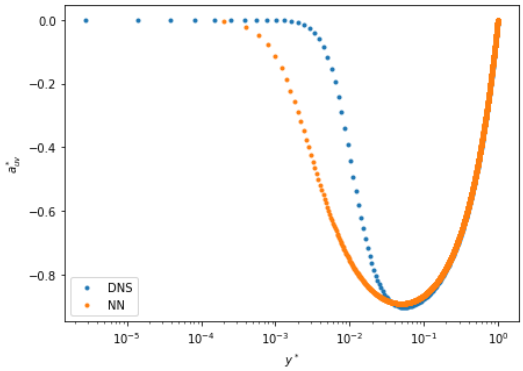


Figure 4 : *Prandtl model for the Reynolds anisotropy tensor*

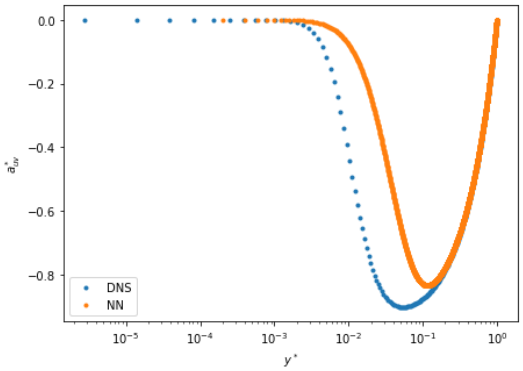


Figure 5 : *Van Driest model for the Reynolds anisotropy tensor*

2.2 Models using Neural Networks

To overcome the limitations of traditional models, machine learning algorithms have been progressively developed to adapt to all types of flows. In particular, the TBNN model, first proposed by (Ling, et al., 2016) allowed a significant improvement in accuracy compared to classical RANS models on a variety of flows. This model has since been outperformed by two other models presented in this section that were developed by (Fang, et al., 2019) and (Sáez de Ocáriz Borde, et al., 2021) during a previous research activity in the laboratory. They thus represent a brick of the complete model developed later. As these models has several variants, it was necessary first to analyse and train them, and then select the most suitable and modify it to fit the complete model.

2.2.1 Multilayer Perceptron - MLP

The basic network used to model the Reynolds anisotropy tensor is a fully connected neural network taking as input the mean velocity gradient of a channel flow $\frac{d\bar{u}^*}{dy^*}$ and returning as output the $u-v$ component of the normalized Reynolds anisotropic tensor, a^*_{uv}.

$$a^*_{uv}\left(\frac{d\bar{u}^*}{dy^*}\right) = MLP\left(\frac{d\bar{u}^*}{dy^*}\right)$$

Schematically, the model can be represented as follows. Bars and stars are not represented in order to simplify the notations.

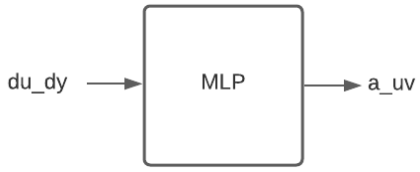


Figure 6 : *MLP diagram*

2.2.2 Adding Boundary Conditions – MLP-BC

To improve the network, it is possible to integrate physical conditions to obtain only realistic solutions. A first step in this direction is to consider the no-slip boundary condition. This is possible by reparametrizing the solution as follows, where the function G verifies $G(0) = 0$.

$$a^*_{uv}\left(\frac{d\bar{u}^*}{dy^*}, y^+\right) = G(y^+)\, MLP\left(\frac{d\bar{u}^*}{dy^*}\right)$$

The G used here is $G(y^+) = 1 - e^{-\theta y^+}$ with θ a hyperparameter. Thus, it is assured that at the edge of the channel, at $y^+ = 0$, $a^*_{uv} = 0$.

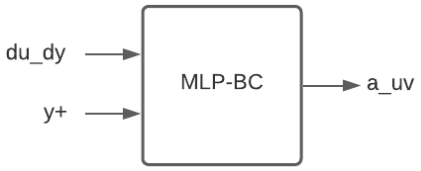


Figure 7 : *MLP-BC diagram*

2.2.3 Friction Reynolds number injection – MLP-BC-Re

Finally, it is also possible to add the friction Reynolds number in the model. Indeed, y^+ contains this information only indirectly. Re_τ is then injected on some layers of the network. This allows to add a new physical condition to the model and to improve its accuracy.

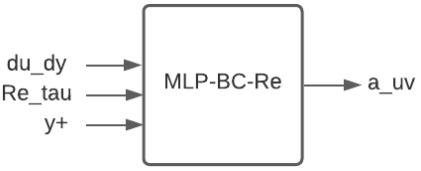


Figure 8 : *MLP-BC-Re diagram*

Another variant consisted in adding to this last model a non-locality condition. This model offers better results for some friction Reynolds number but is slightly less efficient on most of the Re_τ considered. The MLP-BC-Re model is therefore preferred for the continuation of the project.

2.2.4 Convolutional neural network

A last model, developed very recently by (Sáez de Ocáriz Borde, et al., 2021) has also been considered. This model uses a convolutional neural network to predict the normalized anisotropic Reynolds tensor with slightly better performance than the MLP-BC-Re. However, for the sake of flexibility and adaptability to another model, it was initially set aside.

2.2.5 Results of the selected model

The training is presented here for the MLP-BC-Re model. This model takes as input the DNS data of $\frac{d\overline{u}^*}{dy^*}$, Re_τ, and y^+ and predicts an output a^*_{uv}. This output is compared to the true value $a^*_{uv\_DNS}$ from the DNS data using mean squared error (MSE) loss (or squared L2 norm).

The training set is based on the data for $Re_\tau = [550, 2000, 5200]$, while the test set is performed with the values for $Re_\tau = 1000$. This allows the model to be evaluated on data that it has never seen. The performance of the model is similar for all the considered friction Reynolds numbers.

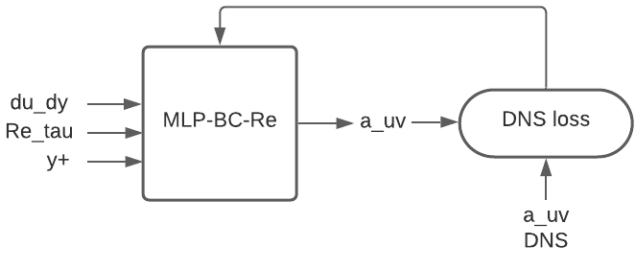


Figure 9 : *Training diagram of the MLP-BC-Re model*

Here are the results obtained at the end of the training. The figure shows a reasonable adequacy of the solution from the model (in orange) with the DNS data (in blue). The model is thus preserved for the continuation of the project.

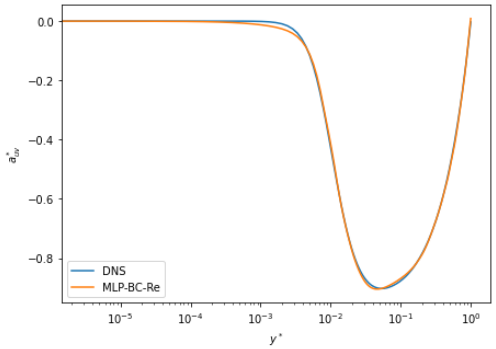


Figure 10 : *Anisotropic Reynolds tensor from MLP-BC-RE*

With the Reynolds anisotropy tensor modeled, it is then possible to solve the associated RANS differential equation. This is the subject of the next chapter.

Chapter 3 – Neural Networks for solving differential equations

3.1 Motivation

3.1.1 Neural network solvers

The idea of using neural networks to solve differential equations is not new: it was first proposed by (Lagaris, et al., 1998). If many numerical methods for solving differential equations exist, the interest of neural networks is twofold: to solve PDEs faster than classical solvers but also to be able, once the network is trained, to solve a whole family of PDEs without having to retrain the model.

3.1.2 Problem dynamics (RANS)

In the context of our problem, and once the Reynolds anisotropy tensor is modeled, it is possible to solve the RANS equation. It will then be necessary to solve for $\bar{u}$ the following differential equation.

$$\frac{1}{Re_\tau}\frac{d^2\bar{u}}{dy^2} - \frac{da^*_{uv}}{dy} + 1 = 0$$

The final objective being to develop a model based on neural networks, a neural network will again be used to solve this equation.

3.2 Population equation

Before developing the network dedicated to the RANS equation, a simpler equation is first studied, both to get to grips with the tools and to understand the mechanisms underlying the use of neural networks.

The studied equation here is the population equation :

$$\begin{cases} u'(t) = \lambda u(t) \\ u(0) = u_0 \end{cases}$$

where $t \in [0, 10]$, $u_0 = 1$ and $\lambda = -0.25$.

3.2.1 Model

Since neural networks are universal approximators, it is then possible to approximate the solution of the equation with : $u_{nn}(t) \approx u(t)$, u being the analytical solution and u_{nn} the solution of the neural network.

It is then expected that the solution respects

$$\begin{cases} u'_{nn}(t) \approx u'(t) = f(u,t) \\ u_{nn}(0) \approx u(0) = u_0 \end{cases}$$

These conditions can then be integrated into a loss function l to be minimized:

$$l = (u_{nn}(0) - u_0)^2 + \sum_i \left(\frac{du_{nn}(t_i)}{dx} - f(u_{nn}, t_i) \right)^2$$

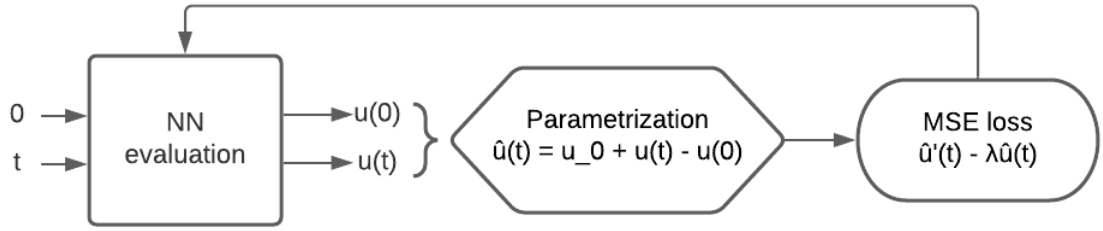
The loss function thus contains all the necessary information, but consists of two terms, which has an impact on the stability of the training. A new solution $\hat{u}_{nn}$ is then being defined, satisfying by construction the boundary condition.

$$\hat{u}_{nn} = u_0 + u_{nn} - u_{nn}(0)$$

In this way the loss function becomes simply

$$l = \sum_i \left(\frac{d\hat{u}_{nn}(t_i)}{dx} - \lambda \hat{u}_{nn}(t_i) \right)^2$$

The training process can then be represented as follows. A discretization of the study interval provides a training set for t, passed as input to the network. The latter is called twice, to predict $u(t)$ as well as $u(0)$. These elements are used to compute $\hat{u}(t)$. The latter is used to calculate the loss, here using MSE loss. Once the loss is calculated, an optimization step (represented by the arrow from the loss function to the network) is performed with respect to the weight of the network.

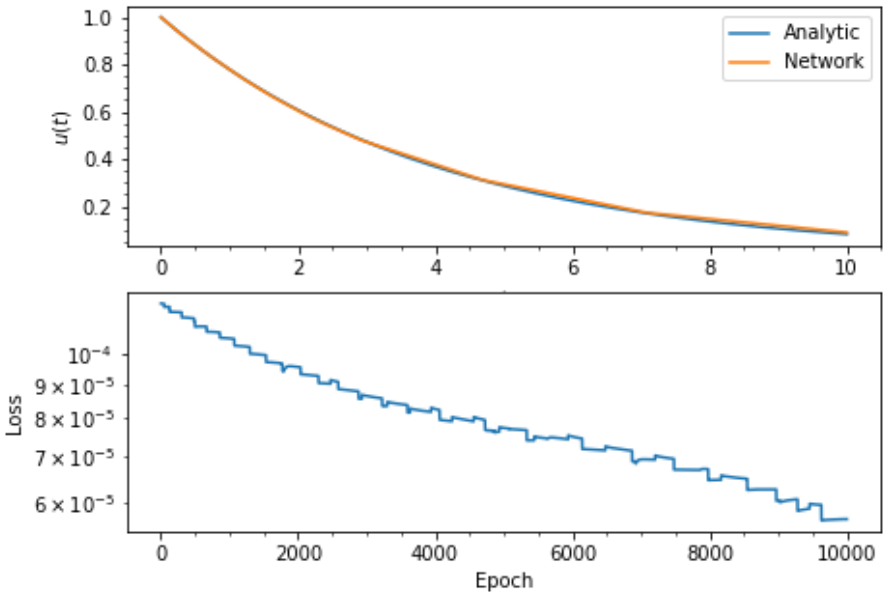


3.2.2 Results

Once the network is trained, it is tested over a certain interval. It is then possible to compare the obtained solution $\hat{u}_{nn}$ with the analytical solution $u(t) = e^{\lambda t}$. The analytical solution is shown in blue on the graph below, while the solution from the network is shown in orange.

The curves are close but not completely merged. Indeed, the training has been done here with a particularly simple network consisting of a single hidden layer. We notice however that the loss function continues to decrease at the end of the training. The network can thus be improved with a longer training. Here, the training already lasts 10.000 epochs[2]. Here again it is possible to do better, by optimizing the learning rate during the training.

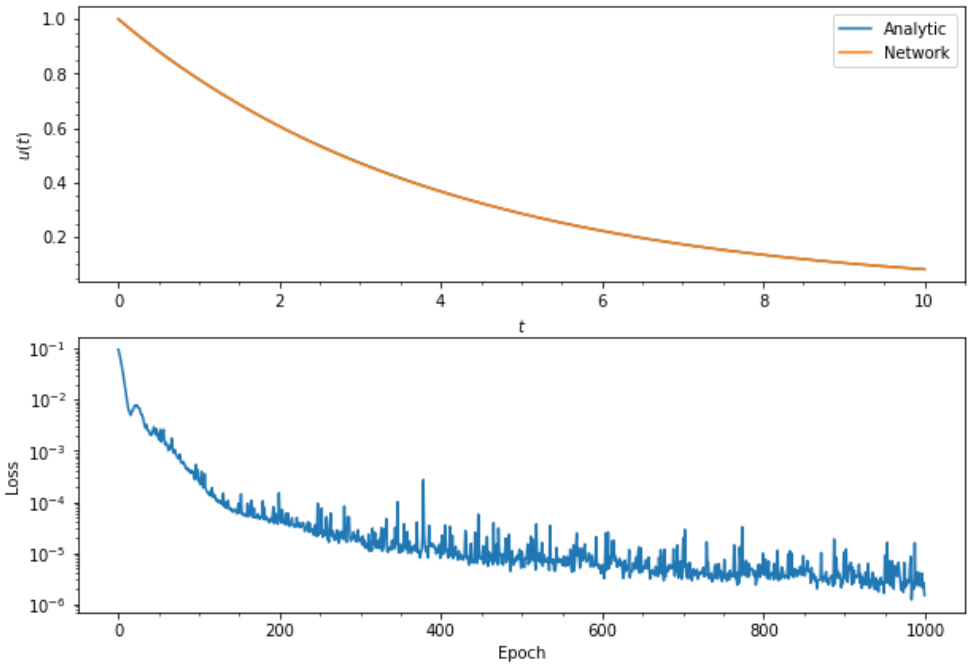
[2] An epoch refers to the number of runs of the training dataset that the machine learning algorithm has made. Data sets are usually grouped into batches (especially when the amount of data is very large).



3.2.3 Using the NeuroDiffEq package

Another solution is based on the use of a package dedicated to the resolution of differential equations by neural networks. The laboratory is currently developing its own, named NeuroDiffEq.

For comparison purposes, the solver of the package has been used to solve the population equation. The result is presented in the graph below. It appears here that the network solution fits perfectly with the analytical solution, all with a much shorter training time.



3.3 Application to the RANS equation

3.3.1 Problem definition

The goal is now to implement a network to solve the RANS equation governing the dynamics of the system. In order to lighten the notations, $\overline{u}$ will simply be noted u and the manipulated quantities will implicitly correspond to their normalized star values (y thus denotes y^* for example).

The equation considered is then the following.

$$\frac{1}{Re_\tau}\frac{d^2u}{dy^2} - \frac{da_{uv}}{dy} + 1 = 0$$

In addition to this relation, there are two boundary conditions shown below:

- the **no-slip condition**, which requires the velocity to be zero at the bottom of the channel, i.e. $u(0) = 0$.
- the **symmetry condition**, which requires the velocity gradient to cancel out at mid-height of the channel, i.e. $\frac{du}{dy}(1) = 0$

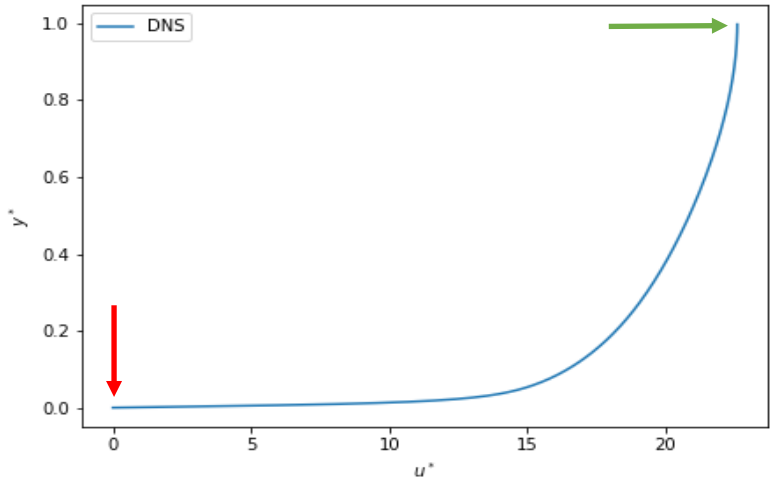


Figure 12 : *Speed profile for Re_τ = 1000 from DNS data with boundary conditions*

3.3.2 Model

It is then possible to solve the RANS equation using the NeuroDiffEq package. The loss function of the network is then the RANS equation. The term derived from the anisotropic Reynolds tensor is then obtained from the pre-trained network defined in the previous section. The boundary conditions are defined in the form of a Dirichlet-Neumann constraint to reparametrize the solution. The following equation and conditions are then passed as arguments to the solver.

$$\begin{cases} \frac{1}{Re_\tau}\frac{d^2u}{dy^2} - \frac{da_{uv}}{dy} + 1 = 0 \\ u(0) = 0 \\ \frac{du}{dy}(1) = 0 \end{cases}$$

In addition, other inputs are passed to the solver:

- a discretization of the interval of interest $I = [0,1]$ in n points,
- the network parameters (weights and biases) with which to start training: randomly defined or from a pre-trained model,
- the choice of the optimization algorithm ("Adam", "SGD" or other).

3.3.3 Results

In order to speed up the learning process, the network is initialized using the Van Driest model defined earlier. The starting model is shown on the left while the trained model is shown on the right. The solution curve obtained by the neural network (in orange) is presented in comparison with the solution from the DNS data (in blue).

It appears that the solution proposed by the model corresponds well to the DNS data. Moreover, the learning time is very short: the loss function stabilizes around $1{,}5.10^{-4}$ after 1000 epochs.

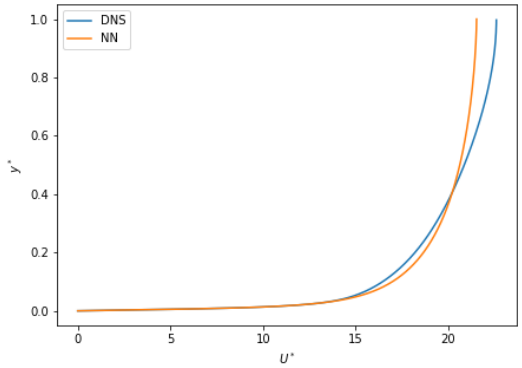


Figure 13 : *Velocity profile with Van Driest model*

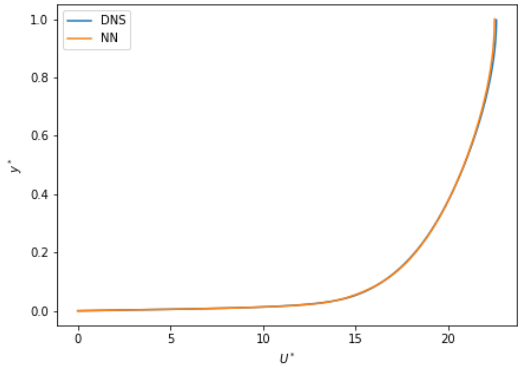


Figure 14 : *Velocity profile with neural network model*

If the solution of the differential equation is satisfactory, this task remains distinct from the modeling of the Reynolds anisotropy tensor. The following chapter thus proposes to group these tasks in a single model.

Chapter 4 – Complete resolution model

Until now, the modeling of the anisotropy Reynolds tensor and the solution of the flow dynamics equation are done independently by two distinct models. This part is devoted to the development of a model that allows to perform both tasks simultaneously. The final model then simply takes as input y and returns the average fluid velocity $\bar{u}(y)$ for a given friction Reynolds number of interest.

4.1 Network selection

In order to solve this problem, the proposed model relies on two distinct but simultaneously trained neural networks. The first one to predict the anisotropic Reynolds tensor and the second to solve the associated differential equation. The network dedicated to the modeling of the anisotropic Reynolds tensor then uses the previous MLP-BC-Re architecture while the network dedicated to the resolution of the equation is new and developed from a MLP. Indeed, despite its efficiency, the model based on the NeuroDiffEq package is not flexible enough. Moreover, a model developed from scratch allows more control and adaptability.

4.2 Architecture and training process

4.2.1 RANS neural network

Once the two models are defined and initialized, the training proceeds as follows and is represented schematically below. A batch of the training set for y is passed into the first MLP network, which predicts an output u. This output is then used in two ways. It is derived to obtain du/dy and passed to the second MLP-BC-Re network with the Re_τ of interest and y^+ to obtain a_{uv} on the one hand and derived again to provide a term of the RANS loss function on the other hand. The a_{uv} obtained is also derived to provide the second term of the RANS loss function. Once the RANS loss function is calculated, an optimization step is performed with respect to the weight of the first network. The process is shown below.

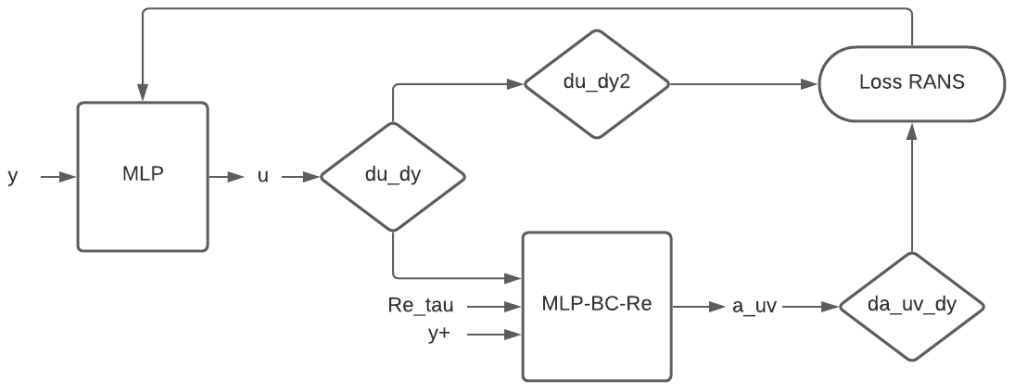


Figure 15 : *Diagram of the training process for the RANS network*

4.2.2 Anisotropic Reynolds tensor neural network

In parallel, the MLP-BC-Re, designated by Anisotropic Reynolds stress tensor (ARST) network is trained. For this purpose, a batch of a second training set containing values of du/dy, Re_τ and y^+ from DNS data for friction numbers different from the Re_τ of interest are passed through the network to predict an output a_{uv}. As before, this value is used to calculate a loss function (DNS loss). This loss function is combined with the RANS loss thanks to a loss combination function to form a final loss function for the network. An optimization step is then performed with respect to the weight of the MLP-BC-Re network. The training diagram is shown below.

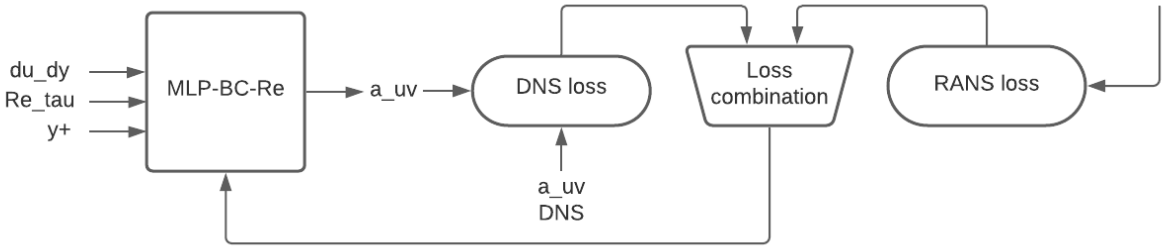


Figure 16 : *Diagram of the training process for the MLP-BC-Re network*

4.2.3 Complete model architecture

The complete model is obtained by joining the two previously defined networks as shown below. The data associated with the friction Reynolds numbers of interest (Re_i) are shown on a blue background. The data represented on yellow background correspond to the other friction Reynolds numbers.

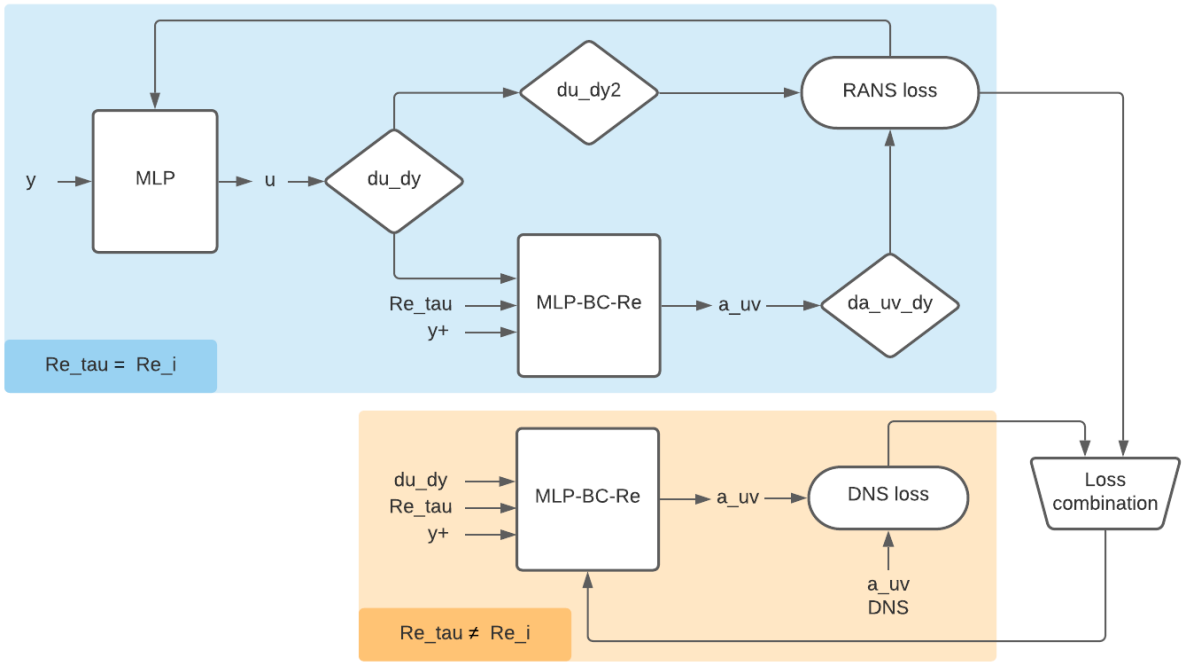


Figure 17 : *Diagram of the training process of the complete resolution model*

4.3 Initialization with Van Driest loss function

4.3.1 Model

As before, in order to improve the training speed of the model, it is preferable to initialize it with an approximation of the solution. The RANS network is then trained alone with the loss function associated with the Van Driest model. This loss function still corresponds to the RANS equation, but where the term associated with the anisotropic Reynolds tensor is given directly by

$$a_{uv} = -k^2 y^2 \left(\frac{du}{dy}\right)^2 (1 - e^{-Cy})^2$$

with $C = \frac{\delta u_\tau}{A^+ \nu}$, where A^+ is the Van Driest damping and k is the Van Karman constant, given (as u_τ and ν) by the DNS data. This model, although imprecise, allows to approach the solution in a very fast way. The RANS network model can then be trained over a smaller number of iterations with the classical loss function.

4.3.2 Initialisation results

The results are presented graphically below. It can be noted that the velocity profile obtained is then similar to the one obtained using the NeuroDiffEq package, so this result is reassuring. Moreover, the accuracy of the model can hardly be improved. The loss function oscillating strongly near 10.000 epochs stops decreasing afterwards, even when using a lower learning rate. The model thus seems to reach its limit.

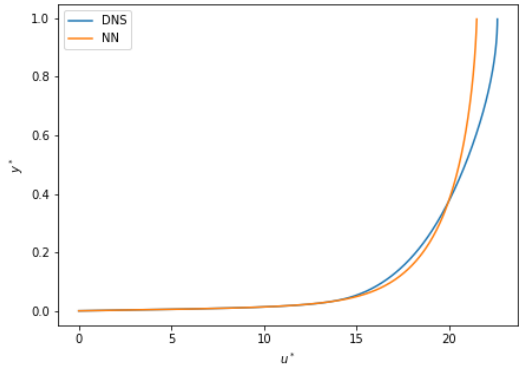


Figure 18 : *Velocity profile of RANS network trained with Van Driest closure model*

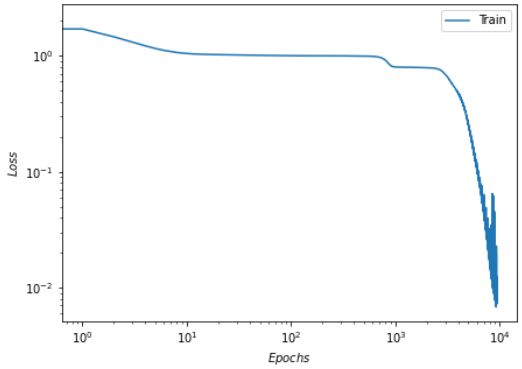


Figure 19 : *Loss function of RANS network training with Van Driest closure model*

4.4 Implementation

4.4.1 Loss functions and hyperparameters

The global model presented above requires several functions and parameters to be defined before training. First, it is necessary to ensure

- the modeling of the anisotropic Reynolds stress tensor: for this, the loss function of the ARST network uses the MSE as before;
- the satisfaction of the RANS equation: to do so, the loss function of both networks

integrates the MSE on the RANS equation directly;

- the respect of the boundary conditions : to ensure $u(0) = 0$ a parametrization similar to the one used for the population equation is used, while the constraint on the derivative $\frac{du}{dy}(1) = 0$ is handled by an additional MSE in the loss function of the RANS network.

Finally, the loss functions of the two networks are defined as follows:

$$loss_{RANS} = \gamma MSE\left(\frac{du}{dy}(1)\right) + MSE\left(\frac{1}{Re_\tau}\frac{d^2u}{dy^2} - \frac{da_{uv}}{dy} + 1\right)$$

$$loss_{ARST} = \beta MSE\left(a_{uv} - a_{uv_{DNS}}\right) + MSE\left(\frac{1}{Re_\tau}\frac{d^2u}{dy^2} - \frac{da_{uv}}{dy} + 1\right)$$

where β and γ are hyperparameters to be chosen. In the following, $\beta = 1$ and $\gamma = 0.01$. This choice balances the terms of the loss function and accelerates learning. It is now possible to train the whole model. Each iteration involves both networks and optimizes them simultaneously.

4.4.2 Results and discussion

The results obtained for the different variables of interest at the end of the training are shown below. It appears that the velocity profile given by the model approaches the solution given by the DNS data. However, the result is not perfect. The derivative of du/dy is indeed quite close to the desired solution but the modeling of the anisotropic Reynolds tensor does not seem to be good enough to estimate the solution correctly. The results do not improve by increasing the value of β associated with the error made on the anisotropic Reynolds tensor term. The MLP-BC-Re model thus does not seem to perform well enough.

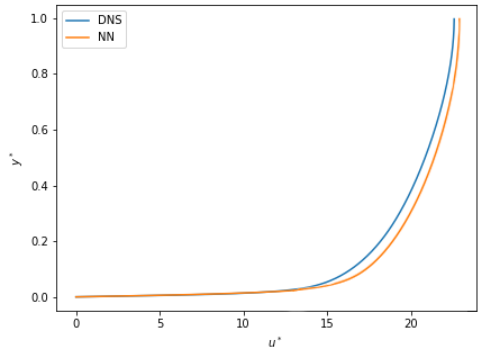


Figure 20 : *Velocity profile from RANS network*

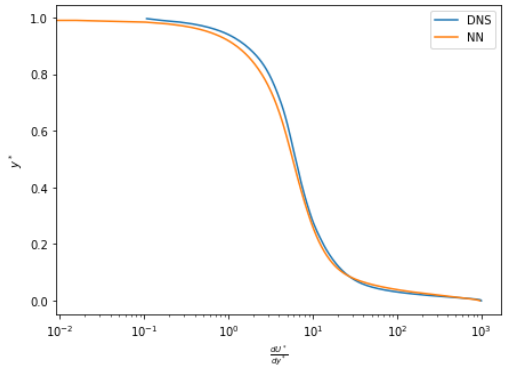


Figure 21 : *Derivative of the velocity profile from RANS network*

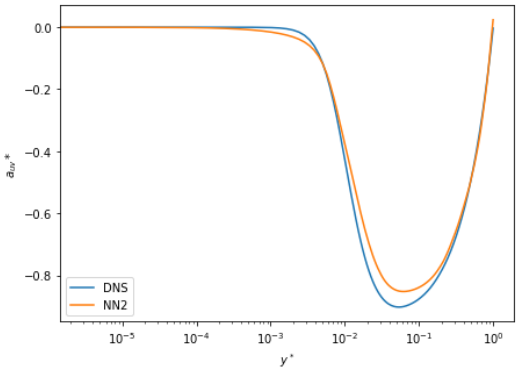


Figure 22 : *Anisotropic Reynolds stress tensor from ARST network*

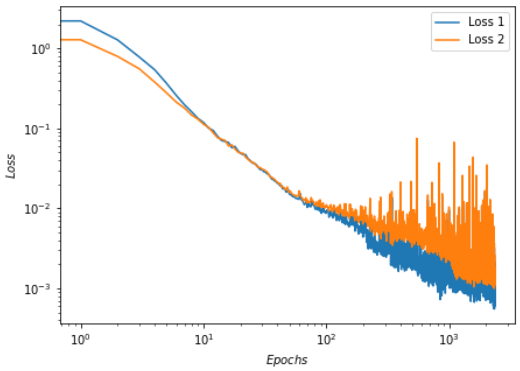


Figure 23 : *Evolution of loss functions during training (Loss 1 : RANS, Loss 2 : ARST)*

4.5 Future work

It appears that the complete model can find an approximation of the solution of the RANS equations while modeling the anisotropic Reynolds stress tensor. Nevertheless, to obtain satisfactory results, it is necessary to improve the predictions of the ARST network. An interesting approach could be the use of the convolutional network developed by (Sáez de Ocáriz Borde, et al., 2021) instead of the MLP-BC-Re. The latter offers better predictions for Turbulent Channel Flow.

Other avenues for future research, including the improvement of the current model, the extension to the three-dimensional problem and the study of other types of flows are mentioned in conclusion.

Conclusion

The resolution of RANS equations is necessary for many engineering applications. In the case of turbulent channel flow, a data-driven approach allows both the modeling of the Reynolds anisotropy tensor and the solution of the RANS equation.

Regarding the closure model, this is possible by imposing physical conditions on the model, including reparametrizing the solution and injecting the friction Reynolds number. This method thus allows to obtain physically viable solutions. Concerning the resolution of RANS, it is also possible to exploit neural networks, taking advantage of their capacity as universal approximators to obtain satisfactory solutions.

Finally, an attempt has been made to combine these two processes within the same model, to integrate the closure model into the solution of the equation and have a single tool to perform the entire resolution. The results of this method can certainly be improved by using a more powerful model for modeling the anisotropic Reynolds tensor, as the convolutional neural network presented in Chapter 1. Moreover, if a quick search for hyperparameters has been performed in this project, a rigorous optimization of the hyperparameters could improve the performance of the current model. Finally, other choices of architecture for the complete model are also possible, as well as other methods of reparametrization or penalization of solutions to enforce the physical conditions.

Future work may also focus on extending the model to solve other problems. This work only deals with the unidimensional case, it will thus be possible to extend the study to a multidimensional flow by using the other components of the Reynolds anisotropic tensor, and by imposing the adapted symmetries. Finally, another avenue of research concerns the study of other types of more complex flows, ideally in 3 dimensions.

Glossary

ARST : Anisotropic Reynolds stress tensor / Tenseur des contraintes anisotropes de Reynolds

DNS : Direct Numerical Simulation / Simulation numérique directe

MLP : Multilayer Perceptron / Perceptron multicouche

MSE : Mean Square Error / Erreur quadratique moyenne

NN : Neural Network / Réseau de neurones

PDE : Partial Differential Equation / Équation différentielle partielle

RANS : Reynolds-Averaged Navier-Stokes / Navier-Stokes avec moyenne de Reynolds

List of Figures

Bibliography

Brunton, S., Noack, B. & Koumoutsakos, P., 2020. Machine Learning for Fluid Mechanics. *Annual Review of Fluid Mechanics*.
Duraisamy, K., Iaccarino, G. & Xiao, H., 2019. Turbulence modeling in the age of data. *Annual Review of Fluid Mechanics*, Volume 51, pp. 357-377.
Fang, R., Sondak, D., Protopapas, P. & Succi, S., 2019. Neural Network Models for the Anisotropic Reynolds Stress Tensor.
Fröhlich, J. & Von Terzi, D., 2008. Hybrid les/rans methods for the simulation of turbulent flows. *Progress in Aerospace Sciences*, Volume 44 (5), pp. 349-377.
Lagaris, I., A., L. & D.I., F., 1998. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5), pp. 987 - 1000.
Lee, M. & Moser, R. D., 2015. Direct numerical simulation of turbulent channel up to Re_t = 5200. *Journal of Fluid Mechanics*, pp. 395-415.
Ling, J., Jones, R. & Templeton, J., 2016. Machine learning strategies for systems with invariance properties. *Journal of Computational Physics*, Volume 318, pp. 22-35.
Ling, J., Kurzawsky, A. & Templeton, J., 2016. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, Volume 807, pp. 155-166.
Mattheakis, M., Sondak, D., Dogra, S. & Protopapas, P., 2020. *Hamiltonian Neural Networks for Solving Differential Equations.* [Online]
Available at: https://arxiv.org/abs/2001.11107
Sáez de Ocáriz Borde, H., Sondak, D. & Protopapas, P., 2021. Convolutional Neural Network Models for the Anisotropic Reynolds Stress Tensor in Turbulent One-dimensional Flows.
Wang, J.-X., Wu, J.-L. & Xiao, H., 2017. Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data. *Physical Review Fluids*, Volume 2 (3).

Appendix

Notations

Fluid density : ρ

Kinetic viscosity : ν

Turbulent kinetic energy : k

Wall shear stress: τ_{wall} (proportional to the pressure gradient)

Friction velocity: $u_\tau = \sqrt{\tau_{wall}/\rho}$

Friction Reynolds : $Re_\tau = u_\tau h/\nu$

Non-dimensional distance from the wall : $y^+ = u_\tau y/\nu = y * Re_\tau$

Normalization

$y^* = y/h$

$\bar{u}^* = \bar{u}/u_b$

$a_{uv}^* = \overline{u'v'}^* = \overline{u'v'}/u_b^2$, where $u_b = \frac{1}{h}\int_0^h \bar{u}\, dy$ is the bulk velocity.

RANS equations associated with the turbulent channel flow

Given the unidimensional Navier-Stokes equations :

$$\begin{cases} \dfrac{\partial u}{\partial t} + (u \cdot \nabla)u = -\nabla p + \dfrac{1}{R_e}\nabla^2 u \\ \nabla \cdot u \;=\; 0 \end{cases}$$

Writing u as, $u(x,t) = \overline{u}(x) + u'(x,t)$, it comes

$$\overline{u}\frac{\partial \overline{u}}{\partial x} + \overline{v}\frac{\partial \overline{u}}{\partial y} + \overline{w}\frac{\partial \overline{u}}{\partial z} + \frac{\partial \overline{u'u'}}{\partial x} + \frac{\partial \overline{u'v'}}{\partial y} + \frac{\partial \overline{u'w'}}{\partial z} = -\nabla \overline{p} + \frac{1}{R_e}\nabla^2 \overline{u}$$

The last three terms of the left-hand member make up the Reynolds stress tensor.

In the case of the turbulent channel flow, $\overline{\boldsymbol{u}} = (\overline{u}(y), 0, 0)$. The equation then becomes

$$\nabla \cdot \boldsymbol{a} = -\nabla \overline{p} + \frac{1}{R_e}\nabla^2 \overline{u}$$

with $\boldsymbol{a} = \overline{u'v'}$.

[i] Details of the flow variables

[ii] Details of the RANS equations calculations