Using Machine Learning for Improving a Non-Linear $k - \varepsilon$ Model: A First Attempt

L. Davidson
Div. of Fluid Dynamics
Dept. of Mechanics and Maritime Sciences (M2)
Chalmers University of Technology, Gothenburg, Sweden

September 11, 2023

Abstract

Machine Learning (ML) is used for improving a non-linear $k - \varepsilon$ model. A developing boundary-layer flow is used to train train the model. In boundary layer flow the model includes only two independent coefficients which are taken as output in a Neural Network (NN) model. Different inputs are evaluated including the velocity gradient and the square of the velocity gradient. They were scaled with either the friction velocity and the viscosity or the turbulence kinetic energy and its dissipation.

The input variables are computed using DNS of developing boundary layer at $Re_\theta = 8\,180$. The NN model is created using Python's `pytorch`. It turns out that the NN model acts as a low-Re model, replacing all low-Re modifications which are present in non-ML non-linear eddy-viscosity turbulence models.

After having trained the NN model in the boundary-layer flow, it is validated in two channel flows at $Re_\tau = 550$ Hoyas and Jimenez (2008) and $Re_\tau = 5\,200$ Lee and Moser (2015). Good agreement is obtained.

Contents

1 Introduction

Ling et al. (2016) used neural networks for predicting the turbulent Reynolds stress. They use a general non-linear Reynolds stress model in which the stresses are expressed in ten tensors, T_{ij}^n and five invariants, i.e. (see Eq. 5)

$$b_{ij} = \sum_{n=1}^{10} G^{(n)} T_{ij}^n, \quad b_{ij} = \overline{v_i' v_j'} - \frac{2}{3} k \delta_{ij}$$

where b_{ij} is the anisotropy tensor. They trained their neural network using the five invariants as input and the 10 tensors as output. Six cases were used for training, namely duct flow, channel flow, a jet in cross-flow, an inclined jet in cross-flow, flow around a square cylinder and flow through a converging-diverging channel. Two of the cases were used for testing, the duct flow and the flow over a wavy wall.

The objective of Wu et al. (2018) was also to develop a model for the turbulent stress tensor, $\overline{v_i' v_j'}$. Contrary to the work by Ling et al. (2016), they include more influence parameters. They include also the gradients of pressure and turbulent kinetic energy, i.e.

$$b_{ij} = f\left(|\bar{s}|, |\bar{\Omega}|, \partial \bar{p}/\partial x_i, \partial k/\partial x_i\right)$$

where $\bar{s} = (\bar{s}_{ij} \bar{s}_{ij})^{1/2}$ and $\bar{\Omega} = (\bar{\Omega}_{ij} \bar{\Omega}_{ij})^{1/2}$. They propose another three influence variables to account for viscous (low-Re number) effects near walls; they use local Reynolds number, $k^{1/2} y/\nu$ (y denotes wall-distance), turbulent kinetic energy and the ratio of turbulent to mean flow time scales. They use a machine learning method based on random forest regression.

Wang et al. (2017) use machine learning for improving the RANS-predicted Reynolds stresses. They perform a RANS simulation and compare with DNS. In their Machine Learning model they create a regression function which maps the input vector which includes 10 features (pressure gradient, streamline curvature, . . .). The output is the Reynolds stresses.

Duraisamy et al. (2019) gives a useful review on Machine Learning and turbulence modeling. They survey recent developments in bounding uncertainties in RANS models using physical constraints and they present methods how to use machine learning to improve turbulence models.

In the present work, the non-linear $k - \varepsilon$ model of Craft *et al.* is improved by optimizing the coefficients using 'Machine learning.

2 Non-linear algebraic models for the Reynolds stress tensor

2.1 Implicit and explicit algebraic stress models

Many years ago, Rodi (1976) proposed an algebraic Reynolds stress mode (denoted ASM, see also Section 11.11 in Davidson (2021))

$$\overline{v_i'v_j'} = \frac{2}{3}\delta_{ij}k + \frac{k}{\varepsilon}\frac{(1-c_2)\left(P_{ij} - \frac{2}{3}\delta_{ij}P^k\right) + \Phi_{ij,1w} + \Phi_{ij,2w}}{c_1 + P^k/\varepsilon - 1} \tag{1}$$

Equation 1 is an *implicit* equation for $\overline{v_i'v_j'}$, i.e. the Reynolds stresses appear both on the left and the right side of the equation. It would of course be advantageous to be able to get an *explicit* expression for the Reynolds stresses. Pope (1975) managed to derive an explicit expression for ASM in two dimensions (later Wallin and Johansson (2000) derived an explicit ASM in three dimensions). Pope assumed that the Reynolds stress tensor can be expressed in the strain-rate tensor, $\bar{s}_{ij}$, and the vorticity tensor, Ω_{ij}. Furthermore, he showed that the coefficients, $G^{(n)}$, in that expression can be a function of not more than the following five invariants

$$\tau^2\bar{s}_{ij}\bar{s}_{ji}, \quad \tau^2\bar{\Omega}_{ij}\bar{\Omega}_{ji}, \quad \tau^3\bar{s}_{ij}\bar{s}_{jk}\bar{s}_{ki} \tag{2}$$

$$\tau^3\bar{\Omega}_{ij}\bar{\Omega}_{jk}\bar{s}_{ki}, \quad \tau^4\bar{\Omega}_{ij}\bar{\Omega}_{jk}\bar{s}_{km}\bar{s}_{mi} \tag{3}$$

where $\tau = k/\varepsilon$ ($k - \varepsilon$ model) or ω ($k - \omega$ model). There are five invariants because when $\bar{s}_{ij}$ and Ω_{ij} are transformed to principal coordinates, there are three eigenvalue for each of them. Furthermore, $\bar{s}_{ii} = 0$ which means there are only five independent invariants.

In two dimensions the expression reads

$$\overline{v_i'v_j'} = \frac{2}{3}k\delta_{ij} + G^{(1)}\tau^2\bar{s}_{ij} + G^{(2)}\tau^3(\bar{s}_{ik}\bar{\Omega}_{kj} - \bar{\Omega}_{ik}\bar{s}_{kj}) \tag{4}$$

In general three-dimensional flow, the Reynolds stress tensor depends on 10 tensors, T_{ij}^n (Pope, 1975), i.e.

$$\overline{v_i'v_j'} - \frac{2}{3}k\delta_{ij} = \sum_{n=1}^{10} G^{(n)}T_{ij}^n$$

$$T_{ij}^1 = \bar{s}_{ij}, \quad T_{ij}^2 = \bar{s}_{ik}\bar{\Omega}_{kj} - \bar{s}_{jk}\bar{\Omega}_{ki}, \quad T_{ij}^3 = \bar{s}_{ik}\bar{s}_{kj} - \frac{1}{3}\delta_{ij}\bar{s}_{mk}\bar{s}_{km}$$

$$T_{ij}^4 = \bar{\Omega}_{ik}\bar{\Omega}_{kj} - \frac{1}{3}\delta_{ij}\bar{\Omega}_{ik}\bar{\Omega}_{ki}, \quad T_{ij}^5 = \bar{\Omega}_{ik}\bar{s}_{km}\bar{s}_{mj} - \bar{s}_{im}\bar{s}_{mk}\bar{\Omega}_{kj}$$

$$T_{ij}^6 = \bar{\Omega}_{im}\bar{\Omega}_{mk}\bar{s}_{kj} + \bar{s}_{ik}\bar{\Omega}_{km}\bar{\Omega}_{mj} - \frac{2}{3}\delta_{ij}\bar{\Omega}_{pm}\bar{\Omega}_{mk}\bar{s}_{kp} \tag{5}$$

$$T_{ij}^7 = \bar{\Omega}_{im}\bar{s}_{mk}\bar{\Omega}_{kn}\bar{\Omega}_{nj} - \bar{\Omega}_{im}\bar{\Omega}_{mk}\bar{s}_{kn}\bar{\Omega}_{nj}, \quad T_{ij}^8 = \bar{s}_{im}\bar{\Omega}_{mk}\bar{s}_{kn}\bar{s}_{nj} - \bar{s}_{im}\bar{s}_{mk}\bar{\Omega}_{kn}\bar{s}_{nj}$$

$$T_{ij}^9 = \bar{\Omega}_{im}\bar{\Omega}_{mk}\bar{s}_{kn}\bar{s}_{nj} - \bar{s}_{im}\bar{s}_{mk}\bar{\Omega}_{kn}\bar{\Omega}_{nj} - \frac{2}{3}\delta_{ij}\bar{\Omega}_{pm}\bar{\Omega}_{mk}\bar{s}_{kn}\bar{s}_{np}$$

$$T_{ij}^{10} = \bar{\Omega}_{im}\bar{s}_{mk}\bar{s}_{kn}\bar{\Omega}_{np}\bar{\Omega}_{pj} - \bar{\Omega}_{im}\bar{\Omega}_{mk}\bar{s}_{kn}\bar{s}_{np}\bar{\Omega}_{pj}$$

where T_{ij}^n may depend on the five invariants in Eq. 2. Equation 5 is a general form of a non-linear eddy-viscosity model. Any ASM may be written on the form of Eq. 5.

It may be noted that Eq. 5 includes only linear and quadratic terms of $\bar{s}_{ij}$ and $\bar{\Omega}_{ij}$. That is because of the Cayley-Hamilton theorem which states that a second-order tensor satisfies its own characteristic equation (see Section 1.20 in (Mase, 1970)); hence cubic terms or higher can recursively be expressed in linear ($\bar{s}_{ij}$) and quadratic tensors ($\bar{s}_{ik}\bar{s}_{kj}$). Furthermore, note that all terms in Eq. 5 are symmetric and traceless as required by the left side, $\overline{v_i'v_j'} - 2\delta_{ij}k/3$.

2.2 Non-linear Eddy-viscosity Models

In traditional eddy-viscosity models the turbulent stress $\overline{v_i'v_j'}$ is formulated from the Boussinesq assumption, i.e.

$$
\begin{aligned}
a_{ij} &= -2\nu_t \frac{\bar{s}_{ij}}{k} \\
\bar{s}_{ij} &= \frac{1}{2}\left(\frac{\partial \bar{v}_i}{\partial x_j} + \frac{\partial \bar{v}_j}{\partial x_i}\right)
\end{aligned}
\tag{6}
$$

where the anisotropy tensor is defined as

$$
a_{ij} \equiv \frac{\overline{v_i'v_j'}}{k} - \frac{2}{3}\delta_{ij}
\tag{7}
$$

The relation between the stress $\overline{v_i'v_j'}$ and the velocity gradient in Eq. 6 is, as can be seen, linear. As shown above, one way to make eddy-viscosity models more general is to include non-linear terms of the strain-rate (i.e. the velocity gradient) (Pope, 1975). In Craft et al. (1997) they use a slightly simplified expression (see also see Section 13 in Davidson (2021))

$$
\begin{aligned}
a_{ij} =\ &\boxed{-2c_\mu\tau\bar{s}_{ij}} \tag{8}\\
&+\ c_1\tau^2\left(\bar{s}_{ik}\bar{s}_{kj} - \frac{1}{3}\bar{s}_{\ell k}\bar{s}_{\ell k}\delta_{ij}\right) + c_2\tau^2\left(\bar{\Omega}_{ik}\bar{s}_{kj} - \bar{s}_{ik}\bar{\Omega}_{kj}\right) \\
&+\ c_3\tau^2\left(\bar{\Omega}_{ik}\bar{\Omega}_{jk} - \frac{1}{3}\bar{\Omega}_{\ell k}\bar{\Omega}_{\ell k}\delta_{ij}\right) + c_4\tau^3\left(\bar{s}_{ik}\bar{s}_{k\ell}\bar{\Omega}_{\ell j} - \bar{\Omega}_{i\ell}\bar{s}_{\ell k}\bar{s}_{kj}\right) \\
&+\ c_5\tau^3\left(\bar{\Omega}_{i\ell}\bar{\Omega}_{\ell m}\bar{s}_{mj} + \bar{s}_{i\ell}\bar{\Omega}_{\ell m}\bar{\Omega}_{mj} - \frac{2}{3}\bar{\Omega}_{mn}\bar{\Omega}_{n\ell}\bar{s}_{\ell m}\delta_{ij}\right) \\
&+\ c_6\tau^3\bar{s}_{k\ell}\bar{s}_{k\ell}\bar{s}_{ij} + c_7\tau^3\bar{\Omega}_{k\ell}\bar{\Omega}_{k\ell}\bar{s}_{ij} \\
\bar{\Omega}_{ij} =\ &\frac{1}{2}\left(\frac{\partial \bar{v}_i}{\partial x_j} - \frac{\partial \bar{v}_j}{\partial x_i}\right)
\end{aligned}
$$

The tensor groups correspond to a subset of Eq. 5:

Line 1: T_{ij}^1,

Line 2: T_{ij}^3 and T_{ij}^2

Line 3: T_{ij}^4 and T_{ij}^5

Line 4: T_{ij}^6

Line 5: T_{ij}^1 multiplied by the invariants $\bar{s}_{k\ell}\bar{s}_{k\ell}$ and $\bar{\Omega}_{k\ell}\bar{\Omega}_{k\ell}$

The constants are set to (Craft et al., 1997)

$$
\begin{aligned}
c_1 &= -0.05, \quad c_2 = 0.11, \quad c_3 = 0.21, \quad c_4 = -0.8 \\
c_5 &= 0, \quad c_6 = -0.5, \quad c_7 = 0.5
\end{aligned}
\tag{9}
$$

The expression in Eq. 8 is cubic in $\partial \bar{v}_i / \partial x_j$. However, note that it is only quadratic in $\bar{s}_{ij}$ and $\bar{\Omega}_{ij}$. As mention above, this is due to the Cayley-Hamilton theorem. a_{ij} is symmetric and its trace is zero; it is easily verified that the right side of Eq. 8 also has these properties.

Examples of non-linear models (sometimes also called *explicit* algebraic Reynolds stress models, EARSM) in the literature are the models presented by Gatski and Speziale (1993); Shih et al. (1995); Craft et al. (1997); Wallin and Johansson (2000).

Let's take a closer look on Eq. 8 in fully developed channel flow ($\bar{v}_2 = \bar{v}_3 = \partial/\partial x_1 = \partial/\partial x_3 \equiv 0$); the terms $T_{ij}^2 = 0$ (Line 2) and T_{ij}^1 multiplied by the invariants $\bar{s}_{k\ell}\bar{s}_{k\ell}$ and $\bar{\Omega}_{k\ell}\bar{\Omega}_{k\ell}$ (Line 5) are zero since $c_6 = -c_7$, see Eq. 9; we obtain

$$
\begin{aligned}
a_{11} &= \frac{1}{12}\tau^2 \left(\frac{\partial \bar{v}_1}{\partial x_2}\right)^2 (c_1 + 6c_2 + c_3) \\[2mm]
a_{22} &= \frac{1}{12}\tau^2 \left(\frac{\partial \bar{v}_1}{\partial x_2}\right)^2 (c_1 - 6c_2 + c_3) \\[2mm]
a_{33} &= -\frac{1}{6}\tau^2 \left(\frac{\partial \bar{v}_1}{\partial x_2}\right)^2 (c_1 + c_3) \\[2mm]
a_{12} &= -c_\mu \tau \frac{\partial \bar{v}_1}{\partial x_2} + \frac{1}{4}\tau^3 \left(\frac{\partial \bar{v}_1}{\partial x_2}\right)^3 (-c_5 + c_6 + c_7)
\end{aligned}
\tag{10}
$$

2.3   The low-Re number non-linear $k - \varepsilon$ model of Craft et al. (1997)

The non-linear model in Eqs. 8 and 10 were adapted for low-Re numbers (i.e. using the model also close to the wall down $y^+ < 1$). For fully-developed channel flow it reads

$$
\begin{aligned}
\overline{v_1'^2} &= \frac{k\nu_t}{12\tilde{\varepsilon}} \left(\frac{\partial \bar{v}_1}{\partial x_2}\right)^2 (c_0 + 6c_2) + \frac{2}{3}k \\[2mm]
\overline{v_2'^2} &= \frac{k\nu_t}{12\tilde{\varepsilon}} \left(\frac{\partial \bar{v}_1}{\partial x_2}\right)^2 (c_0 - 6c_2) + \frac{2}{3}k \\[2mm]
\overline{v_3'^2} &= -\frac{k\nu_t}{6\tilde{\varepsilon}} \left(\frac{\partial \bar{v}_1}{\partial x_2}\right)^2 c_0 + \frac{2}{3}k
\end{aligned}
$$

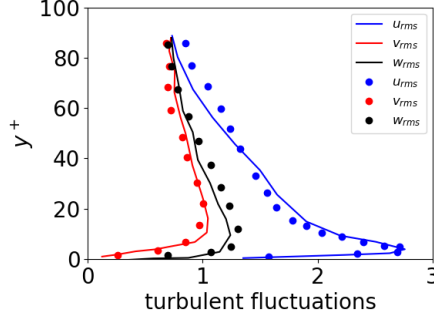Figure 1: Turbulent fluctuations in fully developed-channel flow at $Re_b = 13\,750$. Lines: predictions taken from Craft et al. (1997); symbols: DNS data taken from Craft et al. (1997)

$$
\begin{aligned}
\overline{v_1' v_2'} &= -\nu_t \frac{\partial \bar{v}_1}{\partial x_2} \\
\tilde{\varepsilon} &= \varepsilon - \nu \left(\frac{\partial k^{1/2}}{\partial y} \right)^2, \quad \nu_t = c_\mu \frac{k^2}{\tilde{\varepsilon}} \\
c_\mu &= \frac{0.667 r_\eta \left[1 - \exp(-0.415 \exp(1.3 \eta^{5/6}) \right]}{1 + 1.8 \eta} \\
\eta &= r_\eta \frac{k}{\tilde{\varepsilon}} \frac{\partial \bar{v}_1}{\partial x_2}, \quad R_t = \frac{k^2}{\nu \tilde{\varepsilon}} \\
A_2 &= a_{11} a_{22} + a_{11} a_{33} + a_{22} a_{33} - 2 a_{12} a_{21} \\
f_\mu &= 1.1 (\tilde{\varepsilon}/\varepsilon)^{1/2} \frac{1 - 0.8 \exp(-R_t/30))}{1 + 0.6 A_2 + 0.2 A_2^{3.5}} \\
r_\eta &= 1 + (1 - \exp(-(2 A_2)^3)) \left[1 + 4 \left(\exp(-R_t/20) \right)^{1/2} \right] \\
f_q &= \frac{r_\eta}{1 + 0.0086 \eta^2)^{1/2}} \\
c_1 &= -0.05 \frac{f_q}{f_\mu}, \quad c_2 = 0.11 \frac{f_q}{f_\mu}, \quad c_3 = 0.21 \frac{f_q}{f_\mu}
\end{aligned}
\tag{11}
$$

Craft et al. (1997) presents excellent agreement with DNS data, see Fig. 1.

3   Improving a non-linear $k - \varepsilon$ model using Machine Learning

Instead of using the constant coefficients, $c_1 - c_3$ (see Eq. 9) or using low-Re number modifications as in Section 2.3, I will here present a method how to make the coefficients in Eq. 10 non-constant using Machine Learning.

We find that the coefficients $c_0 \equiv c_1 + c_3$ appear everywhere in Eq. 10. Hence, we re-write the equation system as

$$\overline{v_1'^2} = \frac{k}{12}\tau^2 \left(\frac{\partial \bar{v}_1}{\partial x_2}\right)^2 (c_0 + 6c_2) + \frac{2}{3}k \tag{12}$$

$$\overline{v_2'^2} = \frac{k}{12}\tau^2 \left(\frac{\partial \bar{v}_1}{\partial x_2}\right)^2 (c_0 - 6c_2) + \frac{2}{3}k \tag{13}$$

$$\overline{v_3'^2} = -\frac{k}{6}\tau^2 \left(\frac{\partial \bar{v}_1}{\partial x_2}\right)^2 c_0 + \frac{2}{3}k \tag{14}$$

$$\overline{v_1'v_2'} = -c_\mu \tau \frac{\partial \bar{v}_1}{\partial x_2}$$

Equation 14 gives

$$c_0 = -\frac{6a_{33}}{\tau^2 \left(\frac{\partial \bar{v}_1}{\partial x_2}\right)^2} \tag{15}$$

Inserting Eq. 15 into Eq. 12 gives

$$c_2 = \frac{2a_{11}}{\tau^2 \left(\frac{\partial \bar{v}_1}{\partial x_2}\right)^2} - \frac{c_0}{6} = \frac{2a_{11} + a_{33}}{\tau^2 \left(\frac{\partial \bar{v}_1}{\partial x_2}\right)^2} \tag{16}$$

The objective is now to use Machine Learning to adapt the coefficients c_0 and c_2 so that the normal stresses using Eqs. 12, 13 and 14 agree with DNS of developing boundary layer at $Re_\theta = 8\,180$ (Eitel-Amor et al., 2014a). The mean velocity, $\bar{u}$, k and ε in Eqs. 12 – 14 are taken from DNS (Eitel-Amor et al., 2014a). The DNS data non-dimensional scaled with u_τ and ν.
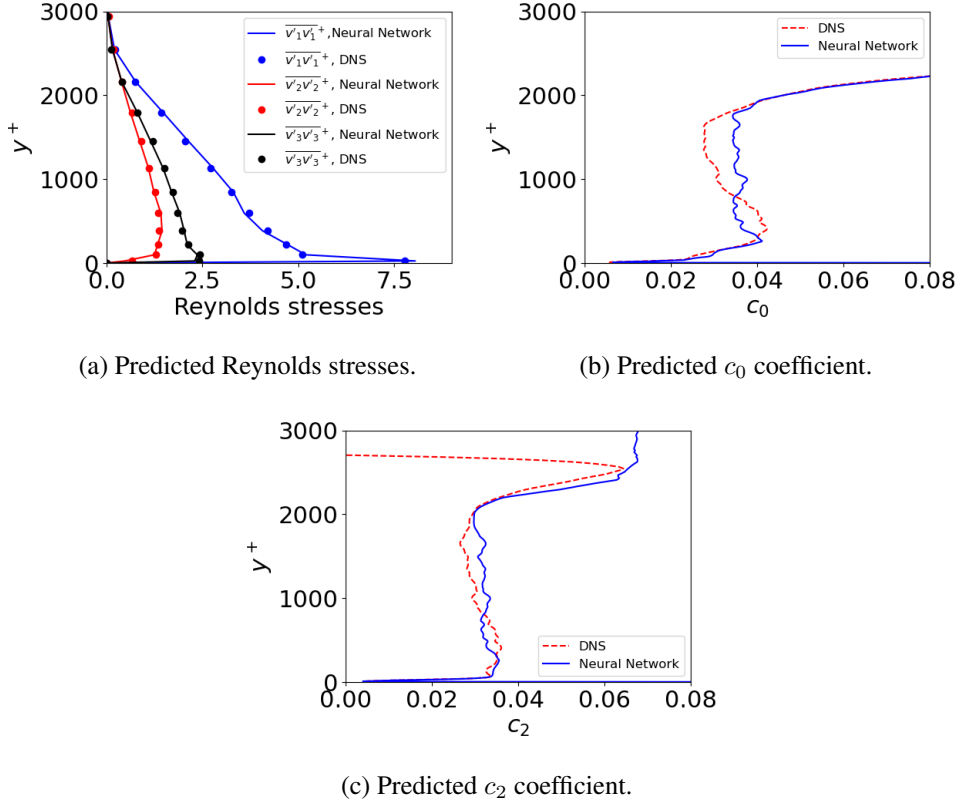
In Davidson (2023), I used Support Vector Machines (SVR) in Python to predict the wall-shear stress, i.e. I had one output parameter. In this work I have two output parameters, c_0 and c_2. SVR can handle only one output parameter. Hence, in this work I will use Neural Network and I choose `pytorch` in Python.

The next question is which input variable should be used. Looking at Eq. 12 – 14, the square of the velocity gradient seems to be an obvious choice. Note that the input variables should be non-dimensional so that the Neural Network Model can be used to predict other flow cases and Reynolds numbers. As mentioned above, the DNS data are already non-dimensional. Recall that the velocity gradients must be taken with respect to y^+ (not y) in order to get a non-dimensional velocity gradient.

However, I did not manage to make the make Neural Network converge with this input. Hence, I added one input variable. There are a number of different suitable choices. Here are some of the combinations of input variables I tried

1. $(\partial U^+/\partial y^+)^2$ and $(\partial U^+/\partial y^+)^{-1}$

2. $T^2 (\partial U/\partial y)^2$ and $T (\partial U/\partial y)^{-1}$, $T = k/\varepsilon$

(a) Predicted Reynolds stresses.

(b) Predicted c_0 coefficient.

(c) Predicted c_2 coefficient.

Figure 2: Developing boundary-layer flow at $Re_\theta = 8\,180$.

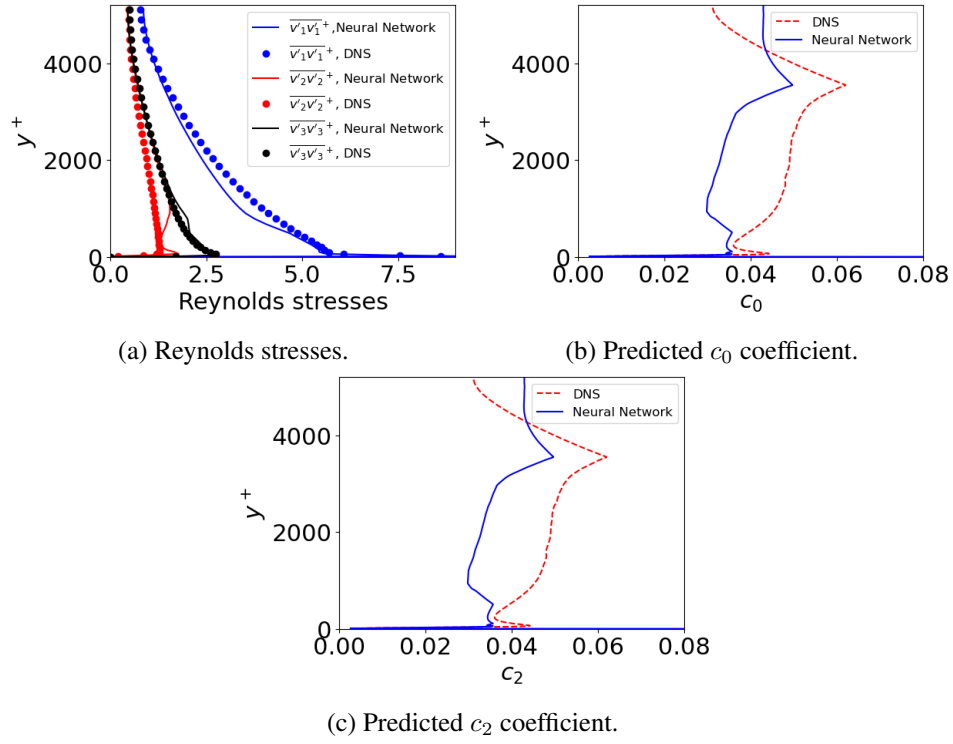3. $(\partial U^+/\partial y^+)^2$ and k^+/ε^+

The advantage of Option 2 is that the scaling is local; the other options depend on scaling based on wall quantities which may be difficult to use in complex geometries. Here I use Option 2 (Option 1 gives virtually identical results).
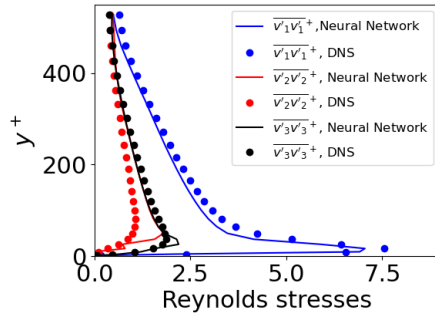
4 Implementing the Neural Network model in Python

As a start for my Python code, I started with the Python code in Batlouni et al. (2023); this code was written in a BSc thesis project in Spring 2023 which I supervised.
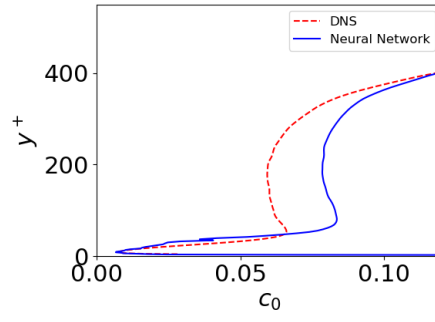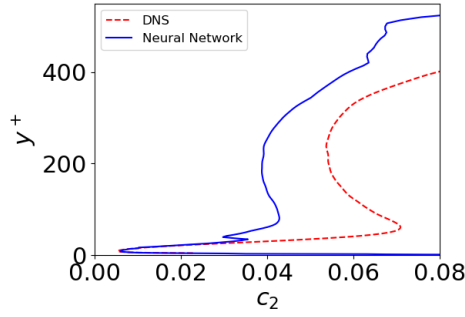
First, I load the required Python packages

```
import numpy as np
import torch
import sys
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
```

(a) Reynolds stresses.

(b) Predicted c_0 coefficient.

(c) Predicted c_2 coefficient.

Figure 3: Channel flow at $Re_\tau = 5\,200$.

(a) Reynolds stresses.

(b) Predicted c_0 coefficient.

(c) Predicted c_2 coefficient.

Figure 4: Channel flow at $Re_\tau = 550$.

(a) Reynolds stresses.

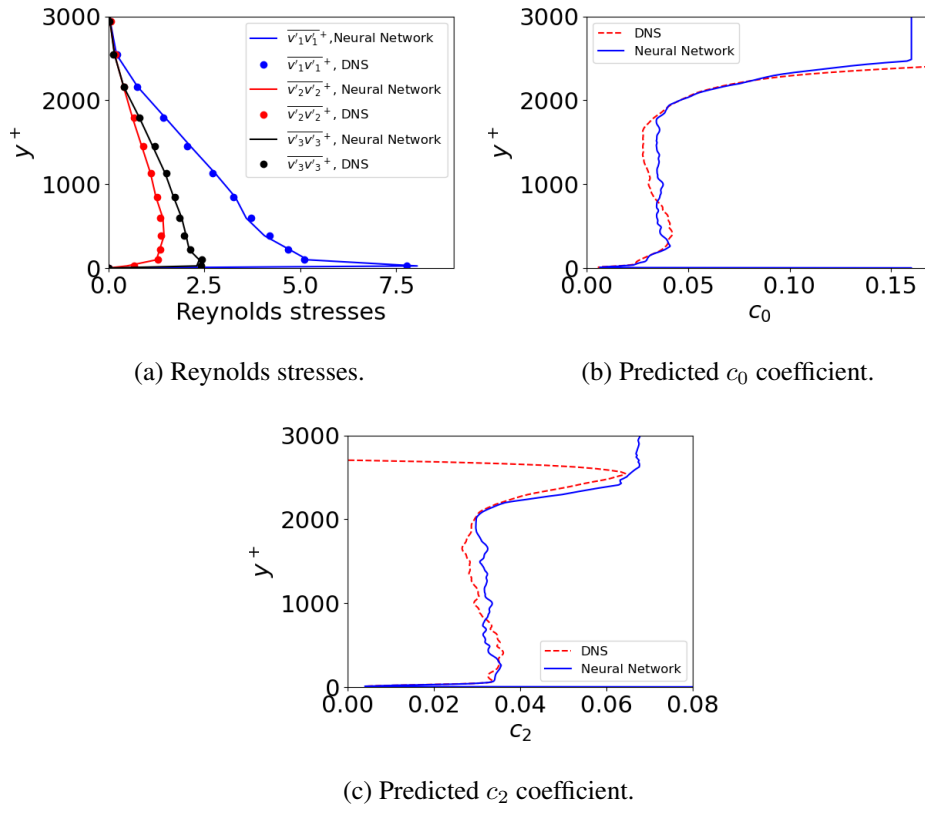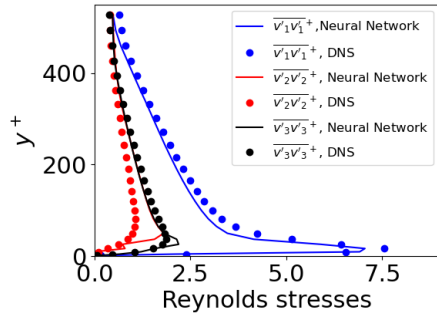(b) Predicted c_0 coefficient.
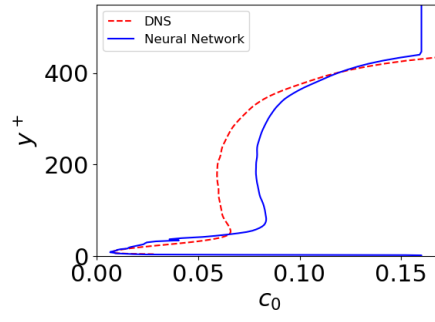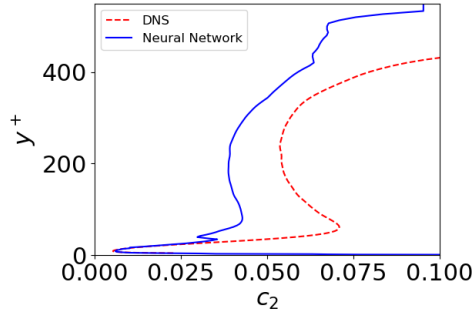

(c) Predicted c_2 coefficient.

Figure 5: Developing boundary-layer flow at $Re_\theta = 8\,180$. $c_{0,max} = 0.16$, $c_{2,max} = 0.11$

(a) Reynolds stresses.

(b) Predicted c_0 coefficient.

(c) Predicted c_2 coefficient.

Figure 6: Channel flow at $Re_\tau = 550$. $c_{0,max} = 0.16$, $c_{2,max} = 0.11$

(a) Reynolds stresses.

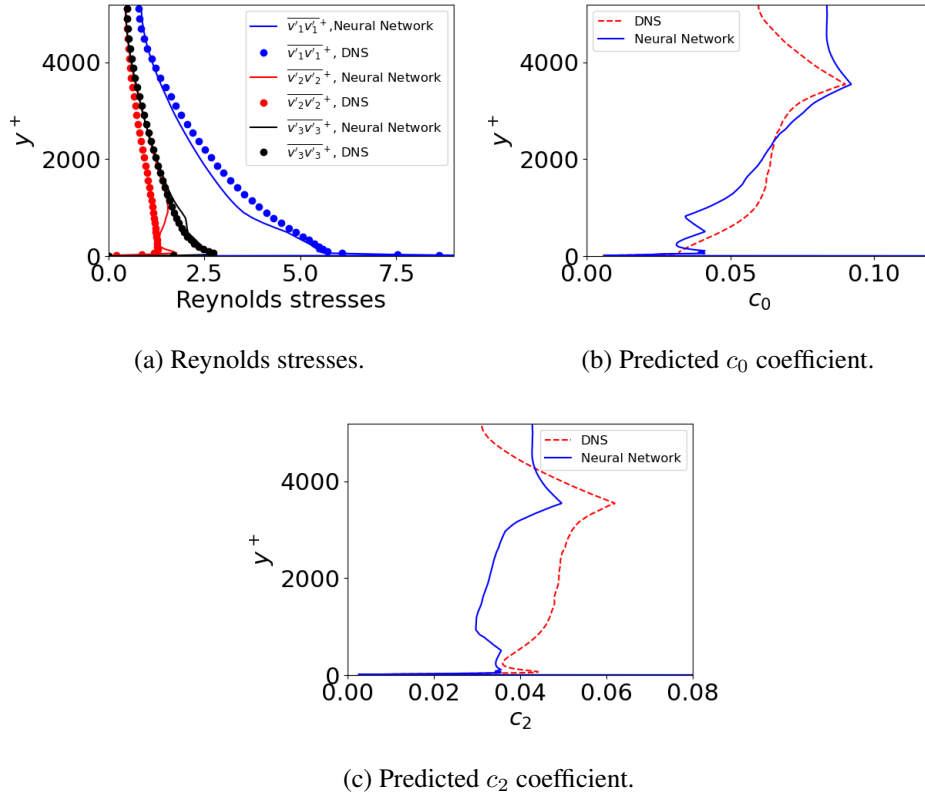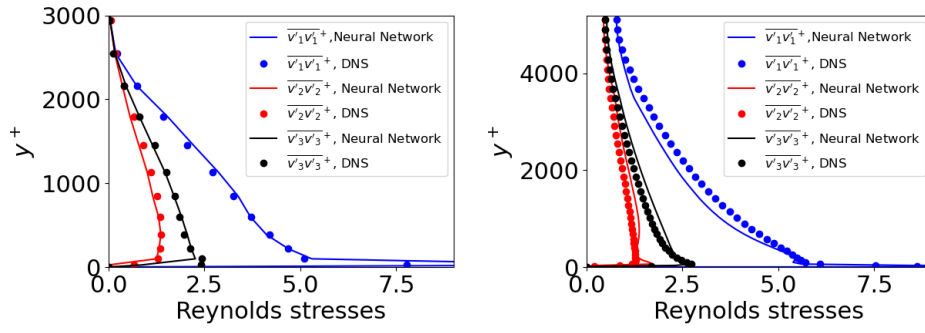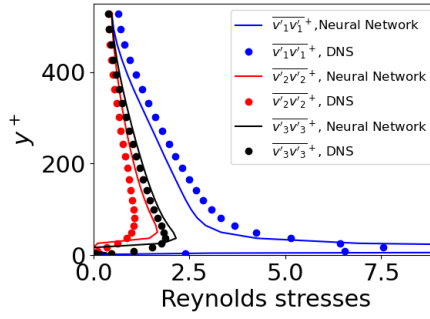(b) Predicted c_0 coefficient.

(c) Predicted c_2 coefficient.

Figure 7: Channel flow at $Re_\tau = 5200$. $c_{0,max} = 0.16$, $c_{2,max} = 0.11$
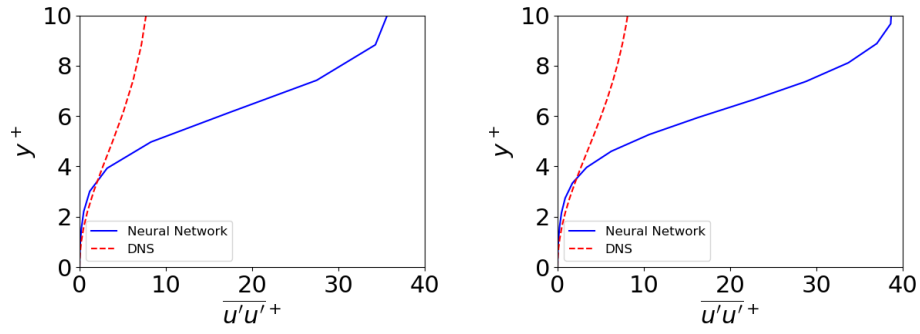
(a) Boundary-later flow. $Re_\theta = 8\,180$.
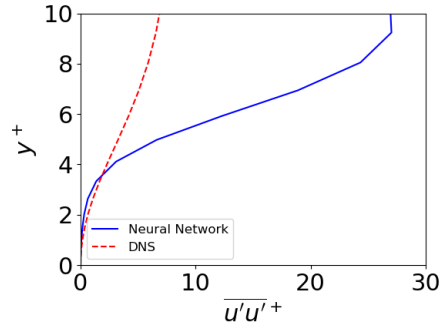
(b) Channel flow. $Re_\tau = 5\,200$.

(c) Channel flow. $Re_\tau = 550$.

Figure 8: Reynolds stresses with $c_0 = c_2 = 0.035$.

(a) Boundary-later flow. $Re_\theta = 8\,180$.

(b) Channel flow. $Re_\tau = 5\,200$.

(c) Channel flow. $Re_\tau = 550$.

Figure 9: $\overline{v_1' v_1'}^+$ with $c_0 = c_2 = 0.035$. A zoomed-in view.

```
7  from torch.utils.data import TensorDataset, DataLoader
8  from sklearn.model_selection import train_test_split
9  from sklearn.preprocessing import MinMaxScaler
10 from joblib import dump, load
11
12 plt.rcParams.update({'font.size': 22})
13 plt.interactive(True)
14 plt.close('all')
```

Next step is to load the DNS data ([Eitel-Amor et al., 2014b](#))

```
1  # load DNS data
2  vel_DNS=np.genfromtxt("vel_11000_DNS_no-text.dat", comments="%")
3
4  y_DNS=vel_DNS[:,0]
5  yplus_DNS=vel_DNS[:,1]
6  u_DNS=vel_DNS[:,2]
7  uu_DNS=vel_DNS[:,3]**2
8  vv_DNS=vel_DNS[:,4]**2
9  ww_DNS=vel_DNS[:,5]**2
10 uv_DNS=vel_DNS[:,6]
11 dudy_DNS  = np.gradient(u_DNS,yplus_DNS)
12 k_DNS   = 0.5*(uu_DNS+vv_DNS+ww_DNS)
13 DNS_RSTE = np.genfromtxt("bud_11000_no-text.prof",comments="%")
14 eps_DNS = -DNS_RSTE[:,4]
15
16
17 # fix wall
18 eps_DNS[0]=eps_DNS[1]
```

Near the wall, the velocity gradients become very large which hampers the convergence of the Neural Network. Furthermore, the Reynolds stresses are negligible in the viscous sub-layer. Hence, the DNS data for $y^+ < 9$ are omitted. Also, at the edge of the boundary layer the velocity gradients go to zero. To promote the convergence rate I set a lower limit on the velocity gradient ($4 \cdot 10^{-4}$).

```
1  #-----------------Data_manipulation-------------------
2  # choose values based on  y+
3  index_choose=np.nonzero((yplus_DNS > 9 )  & (yplus_DNS< 2200 ))
4
5  # set a min on dudy
6  dudy_DNS = np.maximum(dudy_DNS,4e-4)
7
8  uv_DNS      =  uv_DNS[index_choose]
9  uu_DNS      =  uu_DNS[index_choose]
10 vv_DNS      =  vv_DNS[index_choose]
11 ww_DNS      =  ww_DNS[index_choose]
12 k_DNS       =  k_DNS[index_choose]
13 eps_DNS     =  eps_DNS[index_choose]
14 dudy_DNS    =  dudy_DNS[index_choose]
15 yplus_DNS   =  yplus_DNS[index_choose]
16 y_DNS       =  y_DNS[index_choose]
17 u_DNS       =  u_DNS[index_choose]
18
```
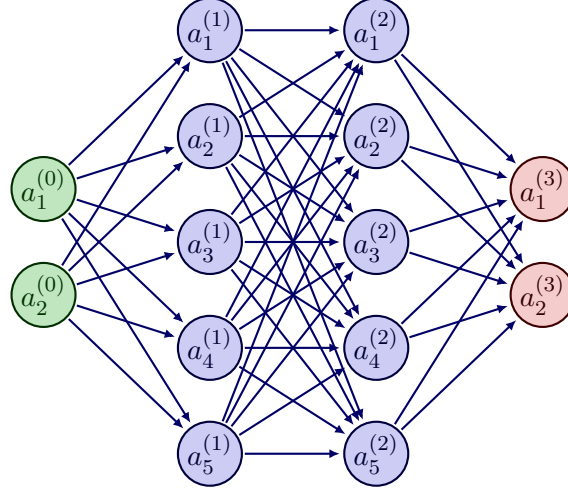
Figure 10: The Neural Network with two inputs variables and two output variables. There are five neurons in this figure; in the simulations I have 50, see Code listing 1.

```
19  tau_DNS  =  k_DNS/eps_DNS
```

The Neural Network Machine is set-up using two hidden layers and 50 neurons. The two input variables rn Fig. 10 are

$$
\begin{aligned}
a_1^{(0)} &= \left(\frac{\partial U^+}{\partial y^+}\right)^2 T^2 \\
a_2^{(0)} &= \frac{1}{T}\left(\frac{\partial U^+}{\partial y^+}\right)^{-1}
\end{aligned}
\tag{17}
$$

where $T = k/\varepsilon$

$$
\begin{aligned}
a_1^{(3)} &= c_0 \\
a_2^{(3)} &= c_2.
\end{aligned}
$$

As you see in Eq. 17, I use the inverse of the velocity gradient as the second input variable. I think it works with $\partial \bar{v}_1^+/\partial x_2^+$ as well.

```
1  # Let's set up a neural network:
2
3  class ThePredictionMachine(nn.Module):
4
5      def __init__(self):
6
7          super(ThePredictionMachine, self).__init__()
8
9          self.input   = nn.Linear(2, 50)   # axis 0: dimension of X
```

```
10        self.hidden1 = nn.Linear(50, 50)
11        self.hidden2 = nn.Linear(50, 2)   # axis 1: dimension of y
12
13    def forward(self, x):
14        x = nn.functional.relu(self.input(x))
15        x = nn.functional.relu(self.hidden1(x))
16        x = self.hidden2(x)
17
18        return x
```

Listing 1: The Neural Network

I have tried up to eight hidden layers, but when I increase the number of hidden layers the convergence rate deteriorates. Two hidden layers seem to give best convergence rate.

The input, X, and the output variables, y, are in `pytorch` setup as:

```
1  c = np.array([c_0_DNS,c_2_DNS])
2
3  # transpose the target vector to make it a column vector
4  y = c.transpose()
5
6  dudy_squared_DNS = (dudy_DNS**2)
7  # scale with k and eps
8  # dudy [1/T]
9  # dudy**2 [1/T**2]
10 T = tau_DNS
11 dudy_squared_DNS_scaled = dudy_squared_DNS*T**2
12 dudy_DNS_inv = 1/dudy_DNS/T
13 # re-shape
14 dudy_squared_DNS_scaled = dudy_squared_DNS_scaled.reshape(-1,1)
15 dudy_DNS_inv_scaled = dudy_DNS_inv.reshape(-1,1)
16 # use MinMax scaler
17 scaler_dudy2 = MinMaxScaler()
18 scaler_dudy = MinMaxScaler()
19 X=np.zeros((len(dudy_DNS),2))
20 X[:,0] = scaler_dudy2.fit_transform(dudy_squared_DNS_scaled)[:,0]
21 X[:,1] = scaler_dudy.fit_transform(dudy_DNS_inv_scaled)[:,0]
```

There are three main input parameters to the Neural Network Model, namely

```
1  # Let's set up a neural network:
2
3  # Set up hyperparameters
4  learning_rate = 9e-1
5  my_batch_size = 5
6  epochs = 40000
```

The `learning_rate` is some sort of under-relaxation; the larger, the faster the model converges, but if it is too large the residual does not decrease but it starts to oscillate. My experience is that the number of batches should be fairly small. It is imperative to drive down the residual to a small value. For this choice of input variables, the `MSELoss()` error is $1.26 \cdot 10^{-6}$. This is an error which is

not normalized and it depends on the magnitude of the output variables. Hence its magnitude is rather useless for estimating the level of convergence. A much better estimate is to normalize the error as

```python
# compute the error
c0_std=np.std(c0-c0_DNS_test)/(np.mean(c0.flatten()**2))**0.5
c2_std=np.std(c2-c2_DNS_test)/(np.mean(c2.flatten()**2))**0.5
```

The errors for the training in Fig. 2 are $4.5 \cdot 10^{-2}$ and $1.6 \cdot 10^{-2}$, respectively. When I tested other input variables these error had to be smaller than $1 \cdot 10^{-3}$; if they were larger, the Neural Network Model predicted too large $\overline{v_1'^2}$ values near the wall (maybe 30% too large) and the model could also give a few negative samples of $\overline{v_2'^2}$. But when I managed to drive down the residuals (by tuning `learning_rate` and/or `my_batch_size` and increasing `epochs`) the non-physical samples disappeared. I found that it is more difficult to converge the NN model when Option 2 for scaling is used, see p. 8 (hence the large number of `epochs`, see above). The convergence with Option 1 is much better.

The training loop of the Neural Network model reads

```python
# Instantiate a neural network
neural_net = ThePredictionMachine()

# Initialize the loss function
loss_fn = nn.MSELoss()

# Choose loss function, check out https://pytorch.org/docs/stable/
    optim.html for more info
# In this case we choose Stocastic Gradient Descent
optimizer = torch.optim.SGD(neural_net.parameters(), lr=
    learning_rate)


for t in range(epochs):
    print(f"Epoch {t+1}\n-----------------------------")
    train_loop(train_loader, neural_net, loss_fn, optimizer)
    test_loss = test_loop(test_loader, neural_net, loss_fn)
print("Done!")

# make the testing (i.e. predicion)
preds = neural_net(X_test_tensor)

#transform from tensor to numpy
c_NN = preds.detach().numpy()

c0=c_NN[:,0]
c2=c_NN[:,1]
```

Finally, the Neural Network model is saved to disk

```python
torch.save(neural_net, 'model.pth')
dump(scaler_dudy2,'model-scaler-dudy2.bin')
dump(scaler_dudy,'model-scaler-dudy.bin')
```

Listing 2: Saving the model

The `numpy` arrays `c0` and `c2` are the predicted coefficients which are then used in Eqs. 12, 13 and 14 to compute the Reynolds stresses $\overline{v_1'^2}$, $\overline{v_2'^2}$ and $\overline{v_3'^2}$, see Fig. 2a. The agreement is – as can be seen – excellent. But this is to be expected since the c_0 and c_2 where trained on the DNS data in Fig. 2a

A better test is to apply the model to boundary flow at different Reynolds numbers. When I make predictions with the NN model I have to load the NN model (Python script `predict.py`) which I saved in Code listing 2 as

```
1   load pytorch model
2  folder='./'
3  filename=str(folder)+'model.pth'
4  neural_net = torch.load(filename)
5  scaler_dudy2 = load(str(folder)+'scaler-dudy2.bin')
6  scaler_dudy = load(str(folder)+'scaler-dudy.bin')
```

Listing 3: Loading the model

The `ThePredictionMachine` is also included in `predict.py`. The prediction is coded in exactly the same way as when doing the testing, see below.

```
1  dudy_squared_DNS   = dudy_DNS**2
2  # re-shape
3  # scale with k and eps
4  # dudy [1/T]
5  # d2udy2 [1/(LT)]  L = k**1.5/eps,   LT =
6  T = tau_DNS
7  dudy_squared_DNS_scaled = dudy_squared_DNS*T*T
8  dudy_DNS_inv = 1/dudy_DNS/T
9  # re-shape
10 dudy_squared_DNS_scaled = dudy_squared_DNS_scaled.reshape(-1,1)
11 dudy_DNS_inv_scaled = dudy_DNS_inv.reshape(-1,1)
12 X=np.zeros((len(dudy_DNS),2))
13 X[:,0] = scaler_dudy2.transform(dudy_squared_DNS_scaled)[:,0]
14 X[:,1] = scaler_dudy.transform(dudy_DNS_inv_scaled)[:,0]
15
16 X_tensor = torch.tensor(X, dtype=torch.float32)
17
18 preds = neural_net(X_tensor)
19 # transform from tensor to numpy
20 c_NN = preds.detach().numpy()
21
22 c0=c_NN[:,0]
23 c2=c_NN[:,1]
```

Listing 4: Predicting the coefficients

Figure 3a presents the predicted stresses in channel flow at $Re_\tau = 5\,200$ and it is seen that at $y^+ \simeq 1000$ the streamwise stress is somewhat underpredicted and the other two stresses exhibit a small local maxima at $y^+ \simeq 1000$. Looking

at the predicted coefficients in Figs. 3b and 3c it seems that the poor predictions at $y^+ \simeq 1000$ are related to local minima in c_0 and c_2. The predicted stresses for $Re_\tau = 550$ in Fig. 4a are in much better agreement with DNS than those in Fig. 3a. The coefficients in Figs. 4b and 4c do not exhibit any local minimum. However, in the outer region they increase strongly.

The fact that the coefficients increase strongly in the outer region where the velocity gradient goes to zero could cause problems when the model is used in general flows including re-circulations regions. One obvious remedy for the large c_0 and c_2 in the outer region is to introduce an upper limit. The upper limit is taken as the constants used by Craft et al. (1997), i.e. $c_0 = 0.16$ and $c_2 = 0.11$, see Eq. 9. Figure 5a presents the Reynolds stresses for the boundary layer using the upper limits. They do not seem to be affected at all (cf. Fig. 2a). For the channel flow at $Re_\tau = 550$, the Reynolds stresses are not affected by the limit because the maximum limits of c_0 and c_2 are not reached. The Reynlds stresses for the channel flow at $Re_\tau = 5\,200$ are also not modified by the upper limit, see Fig. 7a.

We can see in Figures 2, 3 and 4 that if we would choose a constant value for c_0 and c_2, a value of 0.035 seems rather good. Figure 8 presents the Reynolds stresses using constant $c_0 = c_2 = 0.035$ and it seems that better agreement is obtained for all three cases. However, the peak values of the streamwise stress are much overpredicted. Figure 9 shows that they are at least three times larger than the DNS values. This shows that the present NN model is essentially a low-Re number extension of Eq. 10.

5   Conclusions

A Neural Network (NN) model has been used to improve the non-linear $k - \varepsilon$ model of Craft et al. (1997) for predicting the normal stresses. The NN model has been trained in developing boundary layer flow using DNS data. Then the NN model was used for predicting normal stresses in channel flow at and channel flow. Fairly good agreement was obtained. It is found that the best agreement is obtained by using the NN model near the wall (e.g. n $y^+ < 100$) and new constant values on c_0 and c_2 (both equal to 0.035) further away from the wall.

References

F. Batlouni, B. Elm Jonsson, O. Fjeldså, N. Persson, and L. Ånestrand. Utveckling av turbulensmodeller med hjälp av maskininlärning i Python (*Eng.:* development of turbulence models using machine learning in Python). Bsc thesis, Division of Fluid Dynamics, Department of Mechanics and Maritime Sciences, Chalmers University of Technology, Göteborg, Sweden, 2023.

T. J. Craft, B. E. Launder, and K. Suga. Prediction of turbulent transitional phenomena with a nonlinear eddy-viscosity model. *International Journal of Heat and Fluid Flow*, 18:15–28, 1997.

L. Davidson. Fluid mechanics, turbulent flow and turbulence modeling. eBook, Division of Fluid Dynamics, Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, 2021.

L. Davidson. Using machine learning for formulating new wall functions for detached eddy simulation. In *14th International ERCOFTAC Symposium on Engineering Turbulence Modelling and Measurements (ETMM14), barcelona/Digital, Spain 6–8 September*, 2023.

K. Duraisamy, G. Iaccarino, and H. Xiao. Turbulence modeling in the age of data. *Annual Review of Fluid Mechanics*, 51(1):357–377, 2019. doi: 10.1146/annurev-s̄fluid-s̄010518-s̄040547. URL https://doi.org/10.1146/annurev-s̄fluid-s̄010518-s̄040547.

G. Eitel-Amor, R. Orlu, and P. Schlatter. Simulation and validation of a spatially evolving turbulent boundary layers up to $Re_\theta = 8300$. *International Journal of Heat and Fluid Flow*, 47:57–69, 2014a.

G. Eitel-Amor, R. Orlu, and P. Schlatter. Simulation and validation of a spatially evolving turbulent boundary layers up to $Re_\theta = 8300$, DNS Data base. 2014b. URL https://www.flow.kth.se/flow-s̄database/simulation-s̄data-s̄1.791810.

T. B. Gatski and C. G. Speziale. On explicit algebraic stress models for complex turbulent flows. *Journal of Fluid Mechanics*, 154:59–78, 1993.

S. Hoyas and J. Jimenez. Reynolds number effects on the reynolds-stress budgets in turbulent channels,http://torroja.dmt.upm.es/channels/. *Physics of Fluids A*, 20(101511), 2008. doi: http://dx.doi.org/10.1063/1.3005862.

M. Lee and R. D. Moser. Direct numerical simulation of turbulent channel flow up to $Re_\tau \approx 5200$. *Journal of Fluid Mechanics*, 774:395–415, 2015. doi: 10.1017/jfm.2015.268. URL https://doi.org/10.1017/jfm.2015.268.

J. Ling, A. Kurzawski, and J. Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016. doi: 10.1017/jfm.2016.615.

G. E. Mase. *Continuum Mechanics*. Schaum's Outline Series. McGraw-Hill, 1970.

S. B. Pope. A more general effective-viscosity hypothesis. *Journal of Fluid Mechanics*, 472:331–340, 1975.

W. Rodi. A new algebraic relation for calculating the Reynolds stresses. *ZAMM*, 56:T219–T221, 1976.

T.-H. Shih, J. Zhu, and J. L. Lumley. A new Reynolds stress algebraic equation model. *Comput. Methods Appl. Engng.*, 125:287–302, 1995.

S. Wallin and A. V. Johansson. A new explicit algebraic Reynolds stress model for incompressible and compressible turbulent flows. *Journal of Fluid Mechanics*, 403:89–132, 2000.

J.-X. Wang, J.-L. Wu, and H. Xiao. Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data. *Phys. Rev. Fluids*, 2:034603, Mar 2017. doi: 10.1103/PhysRevFluids.2.034603. URL https://link.aps.org/doi/10.1103/PhysRevFluids.2.034603.

J.-L. Wu, H. Xiao, and E. Paterson. Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework. *Phys. Rev. Fluids*, 3:074602, Jul 2018. doi: 10.1103/PhysRevFluids.3.074602. URL https://link.aps.org/doi/10.1103/PhysRevFluids.3.074602.