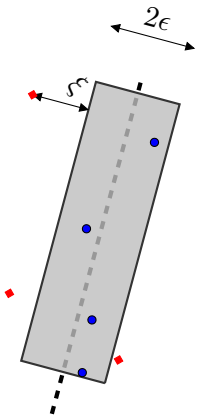
Using Machine Learning for formulating new wall functions for Large Eddy Simulation: A Second Attempt

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



Sketch of support vector regression (SVR). Data point inside (●) and outside (■) the tube (gray area). The gray dashed line is the hyperplane (regression plane) predicted by `svrLINEAR`.

Abstract

Machine Learning (ML) is used for developing wall functions for Large Eddy Simulations (LES). I use Improved Delayed Detached Eddy Simulations (IDDES) in fully-developed channel flow at a frictional Reynolds number of $5\,200$ to create the database. This database is used as a training set for the ML method (support vector regression). The input (i.e. the influence parameters) is y^+. The ML method is trained to predict U^+.

The support vector regression methods in Python are used. The trained ML model is saved to disk and it is subsequently uploaded into the Python CFD code **pyCALC-LES** [1]. IDDES is carried out on coarse – and semi-course – near-wall meshes and the wall-shear stress (using the local y^+ and $\bar{u}$) is predicted using the developed ML model. The test cases are channel flow at $Re_\tau = 16\,000$ and flat-plate boundary layer.

This work is a follow-up of that presented in [2].

Contents

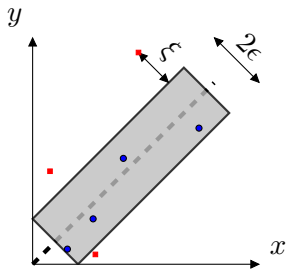


Figure 1: Sketch of support vector regression. Data point inside (●) and outside (■) the tube (gray area). The gray dashed line is the predicted solution which is called the hyperplane which may be linear (`svrLINEAR`) or non-linear (`svr`).

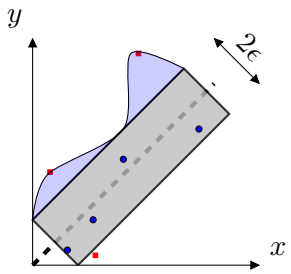


Figure 2: Sketch of vector regression. A large C enlarges the area of the tube (blue area).

1 Introduction

Machine Learning (ML) is a method where known data are used for teaching the algorithm to classify a set of data. The data may be photographs where the ML algorithm should recognize, for example, traffic lights or traffic signs [3]. Another example may be ECG signals where the ML algorithm should recognize certain unhealthy conditions of the heart [4]. A third example is detecting fraud for credit card payments [5]. ML methods such as Support Vector Machines (SVM) and neural networks are often used for solving this type of problems.

The examples above are classification problems using supervised learning (i.e. learning to recognise a traffic light, an unhealthy heart, learn what a customers usual credit card payment looks like). However, in the present work input and output are numerical values. In this case, ML in the form of regression methods should be used [4]; I will use support vector regression (SVR) methods available in Python.

In SVR a regression multi-dimensional "surface" is created which has as many dimensions as number of influence parameters (in the present work I use one influence parameter). Let's make a simple example. In Fig. 1 there is one influence

parameter, x, and one parameter to predict, y. Another two main input parameters are given to the SVR methods. The first is ϵ which determines the width of the tube around the hyperplane [1]. Points that lie inside this tube are considered as correct predictions and are not penalized by the algorithm. The support vectors are the points that lie outside the tube. The second parameter given to SVR models is the C value. It controls the "slack" (ξ), see Fig. 1, which is the distance to points outside the tube. If C is increased the size of the tube is increased so that some or all of the data points are located inside the tube.

There are not many studies in the literature on ML for improving wall functions. In [6] they use a time-averaged high-fidelity IDDES simulation to train a neural network for improving the predicted modeled turbulent kinetic used in wall functions (RANS). In Ref. [7] they use neural network to improve the predicted wall pressure to be used in fluid-structure interactions. Their target is the wall pressure spectrum and the input parameters are the pressure power spectra above the wall. In Ref [8] they use neural network to predict the wall pressure spectra. Their input data are boundary-layer thicknesses (physical, displacement and momentum), streamwise pressure gradient and wall shear stress which are taken from experiments and high-fidelity DNS/LES in the literature. In [9] they use an overly complicated neural network to create a pre-multiplication factor of the velocity-based wall model (VWM) and a log-law based wall model (LLWM). Then they introduce a reward factor, r_n, at each time step n.

2 Numerical method and turbulence model

The finite volume code **pyCALC-LES** [1] is used. It is written in Python and is fully vectorized (i.e. no `for` loops). The solution procedure is based on fractional step. Second-order central differencing is used in space for the momentum equations and Crank-Nicolson is used in time. For k and ε, the hybrid central/upwind scheme is used together with first-order fully-implicit time discretization.

All discretized equation (i.e. the sparse-matrix system) are solved on the GPU using the Algebraic MultiGrid library `pyAMGx` [10] based on AMGX.

2.1 The momentum equations

The equations read

$$\begin{aligned} \frac{\partial \bar{v}_i}{\partial x_i} &= 0 \\ \frac{\partial \bar{v}_i}{\partial t} + \frac{\partial}{\partial x_j}\left(\bar{v}_i \bar{v}_j\right) &= -\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left[\left(\nu + \nu_{sgs}\right)\frac{\partial \bar{v}_i}{\partial x_j}\right] \end{aligned}$$

[1] A hyperplane is a plane whose number of dimension is the same the number of influence parameters. For example, a two-dimensional hyperplane has two influence parameters.

2.2 The underlying RANS turbulence model

The AKN low-Reynolds number for IDDES (see Section 2.3) reads [11]

$$\begin{aligned}
\frac{\partial k}{\partial t}+\frac{\partial \bar{u}_j k}{\partial x_j} &= \frac{\partial}{\partial x_j}\left[\left(\nu+\frac{\nu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_j}\right]+P_k-\psi\varepsilon \\
\frac{\partial \varepsilon}{\partial t}+\frac{\partial \bar{u}_j \varepsilon}{\partial x_j} &= \frac{\partial}{\partial x_j}\left[\left(\nu+\frac{\nu_t}{\sigma_\varepsilon}\right)\frac{\partial \varepsilon}{\partial x_j}\right]+C_{\varepsilon 1}P_k\frac{\varepsilon}{k}-C_{\varepsilon 2}f_2\frac{\varepsilon^2}{k} \\
C_{\varepsilon 1} &= 1.5, \quad C_{\varepsilon 2}=1.9, \quad C_\mu = 0.09 \\
\nu_t &= C_\mu f_\mu \frac{k^2}{\varepsilon}, \quad \sigma_k = 1.4, \quad \sigma_\varepsilon = 1.4
\end{aligned} \tag{1}$$

where the damping functions are defined as

$$\begin{aligned}
f_2 &= \left[1-\exp\left(-\frac{y^*}{3.1}\right)\right]^2\left\{1-0.3\exp\left[-\left(\frac{R_t}{6.5}\right)^2\right]\right\} \\
f_\mu &= \left[1-\exp\left(-\frac{y^*}{14}\right)\right]^2\left\{1+\frac{5}{R_t^{3/4}}\exp\left[-\left(\frac{R_t}{200}\right)^2\right]\right\}
\end{aligned}$$

2.3 The IDDES model

The Improved Delayed Detached Eddy Simulation method is used [12] used when creating the database as well as when using wall functions – based on ML or using Reichardt's law. The coefficient ψ in Eq. 1 is computed as

$$\psi = \frac{l_u}{L_{hyb}}, \quad l_u = \frac{k^{3/2}}{\varepsilon} \tag{2}$$

where L_{hyb} is the usual IDDES length scale [12]. For convenience, the procedure how to obtain the IDDES length scale is summarized below.

$$L_{hyb} = f_d(1+f_e)l_u + (1-f_d)l_c, \quad l_c = C_{DES}\Delta \tag{3}$$

where the Δ length scale is defined as

$$\Delta = \min\left\{\max\left[C_w d_w, C_w h_{max}, h_{wn}\right], h_{max}\right\}$$

and $C_w = 0.15$, d_w is the distance to the closest wall and h_{wn} is the grid step in the wall normal direction. The blending functions f_d and f_e read

$$f_d = \max\left\{\left(1-f_{dt}\right), f_B\right\} \tag{4}$$

$$f_e = \max\left\{\left(f_{e1}-1\right), 0\right\}f_{e2} \tag{5}$$

where the functions f_{dt} and f_B entering Eq. 4 are given by

$$f_{dt} = 1-\tanh\left[\left(8r_{dt}\right)^3\right] \tag{6}$$

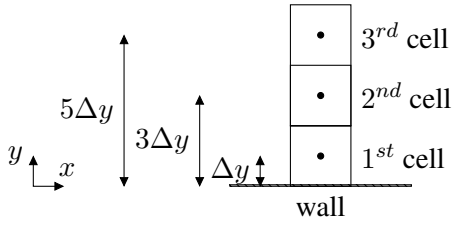


Figure 3: IDDES database.

$$f_B = \min\left\{2\exp\left(-9\alpha^2\right), 1\right\} \tag{7}$$

with

$$\alpha = 0.25 - d_w/h_{max}\ .$$

The functions f_{e1} and f_{e2} in Eq. 5 read

$$f_{e1} = \begin{cases} 2\exp\left(-11.09\alpha^2\right) & \text{if } \alpha \geq 0 \\ 2\exp\left(-9\alpha^2\right) & \text{if } \alpha < 0 \end{cases}$$

and

$$f_{e2} = 1 - \max\left\{f_t, f_l\right\}$$

where the functions f_t and f_l are given by

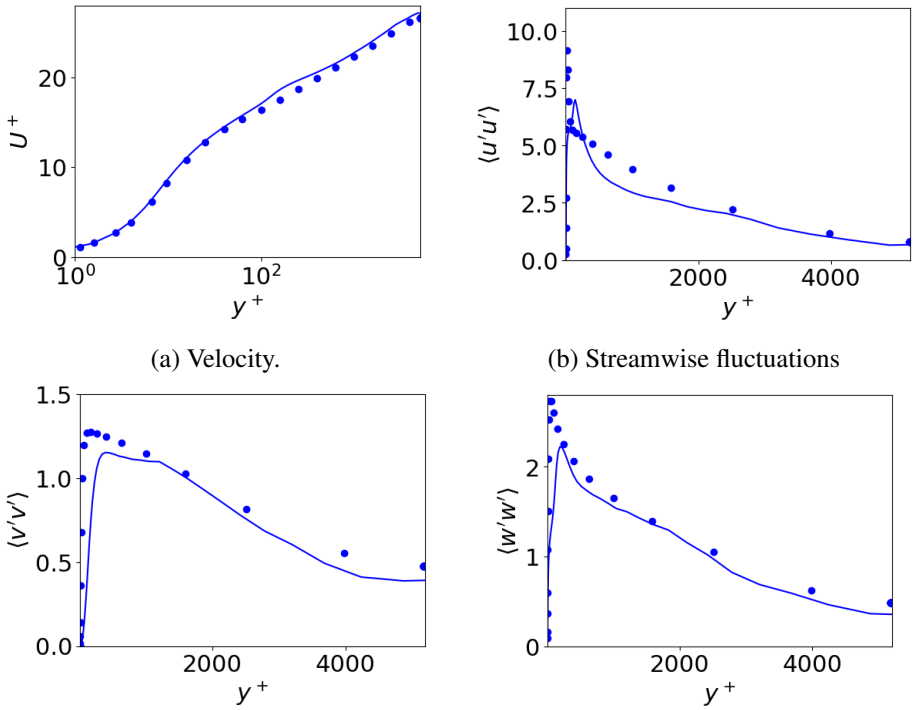
$$f_t = \tanh\left[\left(c_t^2 r_{dt}\right)^3\right]$$
$$f_l = \tanh\left[\left(c_l^2 r_{dl}\right)^{10}\right]\ .$$

The constants c_t and c_l given the same values as in the $k-\omega$ SST model, i.e. $c_t = 1.87$ and $c_l = 5$ [12]. The quantities r_{dt} (also entering Eq. 6) and r_{dl}, are defined as follows

$$r_{dt} = \frac{\nu_t}{\kappa^2 d_w^2 \max\left\{|\bar{s}|, 10^{-10}\right\}}$$
$$r_{dl} = \frac{\nu}{\kappa^2 d_w^2 \max\left\{|\bar{s}|, 10^{-10}\right\}}$$

3 Creating the database

To create a database which can be used for training the SVR I will carry out simulations using IDDES (see Section 2) of fully-developed channel flow. The size of the channel is $x_{max} = 3.2$ (streamwise, x or x_1), $y_{max} = 2$ (wall normal, y or x_2) and $z_{max} = 1.6$ (spanwise, z or x_3). The mesh has $96 \times 96 \times 96$ and the Reynolds



(a) Velocity.

(b) Streamwise fluctuations

(c) Wall-normal fluctuations.

(d) Spanwise fluctuations.

Figure 4: Mean flow and resolved turbulence. —— : IDDES; ●: DNS data [13].

	$\langle \Delta y^+ \rangle$
Location 1	12
Location 2	31
Location 3	49
Location 4	66
Location 5	76
Location 6	88
Location 7	135
Location 8	155
Location 9	207

Table 1: IDDES database for wall functions. Δy is defined in Fig. 3. The locations of the second and the third cell is obtained from Fig. 3.

number is 5 200 based on the friction velocity, $\langle u_\tau \rangle$ ($\langle \cdot \rangle$ denotes average in time, x_1 and x_3), and the half-channel width, δ. DNS was used in [2] on a much finer mesh and a lower Reynolds number. The accuracy of IDDES is consider to be sufficient.

Figure 4 presents comparison of the predicted velocity field and RSM fluctuations with DNS and – as can be seen – the agreement is reasonable.

The instantaneous velocity, $\bar{u}$, is stored at nine locations in the database along with the friction velocity at the same (x, z) position. It may be noted that in [2] $\bar{u}$ was integrated over $2\Delta y$ (giving $\bar{U}$) where $2\Delta y$ was relevant for the wall-normal cell size when using wall functions. At the end of [2], it has been found that the correlation between $\bar{U}$ and the friction velocity is very low and hence integrating in y is not a good procedure. The grid in the wall-parallel plane is finer than in a typical wall-function mesh. Hence, in the present study the instantaneous $\bar{u}$ velocities and the friction velocities are integrated over ΔX and ΔZ where ΔX and ΔZ correspond to typical cell size in a wall-function mesh, i.e.

$$\begin{aligned} \bar{U}(x,y,z) &= \frac{1}{\Delta X \Delta Z} \int_{x-0.5\Delta X, z-0.5\Delta Z}^{x+0.5\Delta X, z+0.5\Delta Z} \bar{u}(x',y,z')dx'dz' \\ \bar{u}_\tau(x,z) &= \frac{1}{\Delta X \Delta Z} \int_{x-0.5\Delta X, z-0.5\Delta Z}^{x+0.5\Delta X, z+0.5\Delta Z} u_\tau(x'z')dx'dz' \end{aligned} \tag{8}$$

ΔX and ΔZ are set to 0.1 and 0.05, respectively. The nine locations of the first cell are given in Table 1. The locations of the second and third cells are shown in Fig. 3. The $\bar{U}$ velocity at the second and the third cells are stored in order to be able to compute the first and the second velocity derivative which could be used as additional input (i.e. influence) parameters. They are not used in the present work.

4 Standard wall functions

The ML wall functions will be compared to wall functions based on Reichardt's law

$$\frac{\bar{u}_P}{u_\tau} \equiv U^+ = \frac{1}{\kappa}\ln(1-0.4y^+)+7.8\left[1-\exp\left(-y^+/11\right)-(y^+/11)\exp\left(-y^+/3\right)\right] \tag{9}$$

The friction velocity is then obtained by solving the implicit equation

$$u_\tau - \bar{u}_P\left(\ln(1-0.4y^+)/\kappa + 7.8\left[1-\exp\left(-y^+/11\right)-(y^+/11)\exp\left(-y^+/3\right)\right]\right)^{-1} = 0 \tag{10}$$

using the Newton-Raphson method `scipy.optimize.newton` in Python. $\bar{u}_P$ and y^+ denote the non-dimensional wall-parallel velocity and wall distance, respectively, at the first, second or third wall-adjacent cells.

5 Wall functions based on ML methods

As indicated in the introduction, I will use SVR (Support Vector Regression). Two different packages are used, `svr` and `svrLINEAR`, both available in Python. The former method is non-linear and the latter is linear. The ML method consists of a learning part and a testing part. In the learning part, the ML method is trained and in the testing part it is tested.

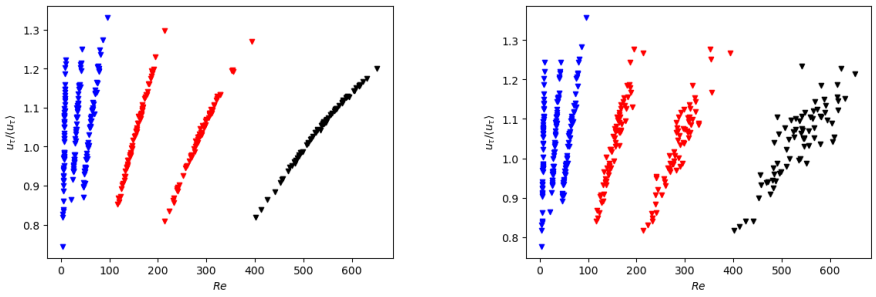
First, I need to determine which input variable (influence parameters) that should be used. In standard wall functions such as Eq. 9, the input parameters are wall-parallel velocity, $\bar{v}_P$ and the non-dimensional wall distance, y^+ (which includes the friction velocity, u_τ); the output is the friction velocity. Hence, the friction is both input and output. In a ML method, it may be a bad idea to let a parameter be part of both the input and output parameters. Hence, in [2] I choose the local Reynolds number

$$Re = \frac{\bar{U}\Delta y}{\nu}, \tag{11}$$

and the non-dimensional velocity gradient as input parameters. The friction velocity was taken as output, i.e.

$$u_\tau = f\left(Re, \langle y^+\rangle, \frac{\partial \bar{U}}{\partial y}\right) \tag{12}$$

Figure 5 presents the resulting scatter plots. The three different markers and two different colors represent different $\langle y^+\rangle$, i.e. different locations. It turns out that this choice of influence parameters and out parameter is not a good one since for certain combinations of $\langle y^+\rangle$ and Re multiply values of the output parameter,



(a) Testing (prediction). (b) Target data (IDDES datan).

Figure 5: 2D scatter plot of the friction velocity (Eq. 12) using Re (Eq. 11) and the time-averaged y^+ (Eq. 13) as influence parameter and u_τ as output parameter. ▲: Location 1; ▲: Location 2; ▲: Location 3; ▼: Location 4; ▼: Location 5; ▼: Location 6 (see Table 1).

u_τ may be obtained. As a remedy, a third input variable was added, name;y the averaged non-dimensional wall distance, i.e.

$$\langle y^+ \rangle = \frac{\langle u_\tau \rangle \Delta y}{\nu} \tag{13}$$

This ML wall function gave fairly good results in fully-developed channel flow [2] but in the present work it was found that it fails in flat-plate boundary layers (not shown).

Hence, I decided to pick a different set of influence and output parameters. Let's take guidance of the Reichard's law (Eq. 9) and the log-law and the linear law

$$\frac{\bar{u}}{u_\tau} \equiv U^+ = \frac{1}{\kappa} \ln(Ey^+), \quad U^+ = y^+$$

All three are written on the form

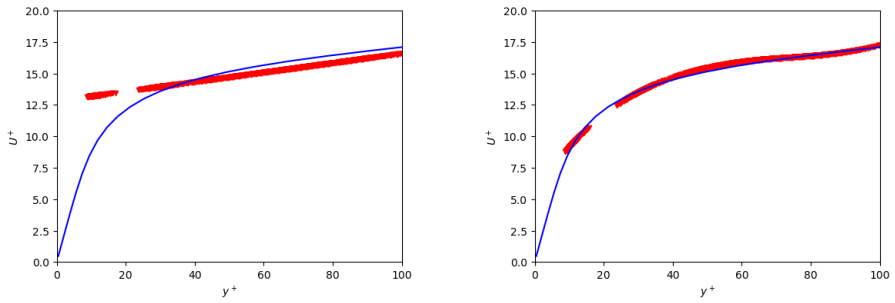
$$\frac{\bar{u}}{u_\tau} = f\left(y^+\right) \equiv f\left(\frac{u_\tau y}{\nu}\right)$$

I choose to use the same output and input in the ML wall functions, i.e.

$$\begin{aligned} y^+ &: \quad \text{influence parameter} \\ U^+ &: \quad \text{output parameter} \end{aligned}$$

The potential drawback is that u_τ appears both in the influence and output parameter which could necessitate an iterative procedure (as when solving Eq. 9) predicting U^+ a couple of times with updated y^+. In the results section it is found that that is not needed.

Next, I will train the ML method using the IDDES data created in Section 3. I use 300 independent samples at each of the nine locations, see Table 1 and Fig. 3.



(a) `svrLINEAR`. (b) `svr`.

Figure 6: Scatter plot of predicted (testing) U^+ using $C = 10$ and $\varepsilon = 0.001$.

U^+. I pick 80% of the data randomly and define that as the training set. The remaining 20% is then used for testing, i.e. predicting.

Figure 6 presents scatter plots of predicted friction velocities using `svr` and `svrLINEAR` for all testing data (in total 540 data points, i.e. 20% of 300 independent samples at nine locations, see Table 1 and Fig 3). It is seen that for `svrLINEAR` u_τ varies linearly (which is expected since `svrLINEAR` is linear) whereas `svr` closely follows the target data. Hence, `svr` is chosen. The slack parameter, C, and ε (see Section 1) are set to 10 and 0.001, respectively. The error, e, between the predicted U^+ (testing samples) using `svr` and the IDDES database

$$e = \frac{\texttt{std}\left(U^+_{pred} - U^+_{IDDES}\right)}{\texttt{mean}\left(U_{pred}\right)} \tag{14}$$

is approximately 9%.

5.1 Python code

Here I present some of the Python commands. First, I scale the input data

```
scaler_yplus=StandardScaler()
yplus_in=scaler_yplus.fit_transform(yplus_in)
```

Then I put the input and output data on generic form

```
y=uplus_out    # output
X=np.zeros((n_svr,1))    #input
X[:,0]=yplus_in[:,0]
```

Next I pick a model

```
C=10
eps=0.001
model = SVR(epsilon = eps, C = C)
```

I train the model

```
svr = model.fit(X, y.flatten())
```

Now comes the testing part. I scale my test data

```
# Use MinMax scaling
yplus_in_test=yplus_in_test.reshape(-1, 1)
yplus_in_test=scaler_yplus.transform(yplus_in_test)

# test
X_test=np.zeros((n_test,1))
X_test[:,0]=yplus_in_test[:,0]
```

Then I predict U^+

```
y_svr = model.predict(X_test)
```

Now I want to find how accurate my predictions are. I compare with IDDES data

```
# find difference
uplus_rms=np.std(y_svr-uplus_out_test)/\
np.mean(uplus_out_test.flatten())
print('rms_error',uplus_rms)
```

Finally I store the model on disk.

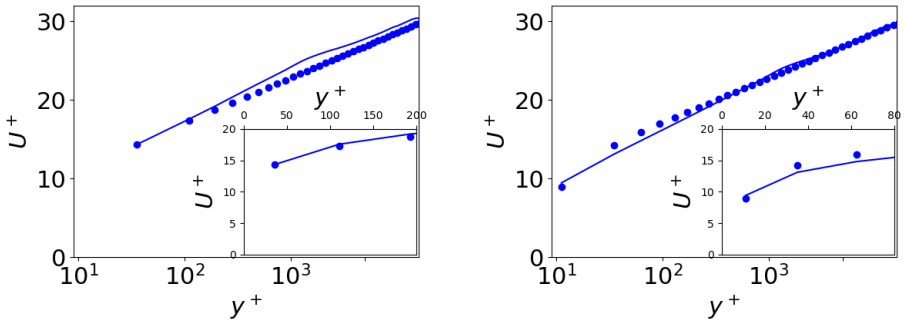
```
# save the model to disk
filename = 'model-svr-C-10-eps-0.001.bin'
dump(model, filename)
dump(scaler_yplus,'model-svr-C-10-eps-0.001-yplus.bin')
np.savetxt('min-max-svr-C-10-eps-0.001.txt', \
[yplus_max, yplus_min])
```

When I do the IDDES simulations with **pyCALC-LES** I load the model.

```
if itstep == 0 and iter == 0:
# load model
   folder='~noback/pycalc-les/channel-5200-IDDES-96-86-96-ML/'

   filename=str(folder)+'model-svr-C-10-eps-0.001.bin'
   model = load(filename)
   scaler_yplus = load(str(folder)+'model-svr-C-10-eps-0.001-yplus.bin')
   yplus_max,yplus_min=
    np.loadtxt(str(folder)+'min-max-svr-C-10-eps-0.001.txt')

# initialize
   ustar_south=np.ones((ni,nk))
```

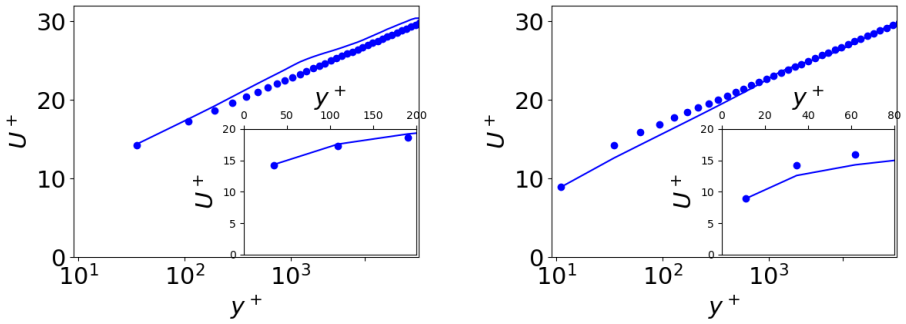
When I predict U^+, I do as in the testing phase above at both wall. For example, at the south wall

```
u2d_wall=abs(u3d[:,0,:])  # first cell
#u2d_wall=abs(u3d[:,1,:])  # 2nd cell
#u2d_wall=abs(u3d[:,2,:])  # 3rd cell

# wall distance
dy_wall=dist3d[0,0,0]  # first cell
#dy_wall=dist3d[0,1,0]  # 2nd cell
#dy_wall=dist3d[0,2,0]  # 3rd cell

yplus=ustar_south*dy_wall/viscos

#flatten
yplus=yplus.flatten()

# count values larger/smaller than max/min
yplus_min_number= (yplus < yplus_min).sum()
yplus_max_number= (yplus > yplus_max).sum()

print('south: yplus_min_number',yplus_min_number)
print('south: yplus_max_number',yplus_max_number)

# set limits
yplus=np.minimum(yplus,yplus_max)
yplus=np.maximum(yplus,yplus_min)

#size
N=len(yplus)

# re-scale
yplus=yplus.reshape(-1, 1)
yplus=scaler_yplus.transform(yplus)

# predict
X=np.zeros((N,1))
X[:,0]=yplus[:,0]

# compute uplus
y_svr = model.predict(X)

uplus=np.reshape(y_svr,(ni,nk))
```



(a) $N_y = 66$, stretching 11%. (b) $N_y = 68$, stretching 14.7%.

Figure 7: Channel flow. `svr`. $Re_\tau = 16\,000$. Velocity. •: Reichardt's law, Eq. 9.



(a) $N_y = 66$, stretching 11%. (b) $N_y = 68$, stretching 14.7%.

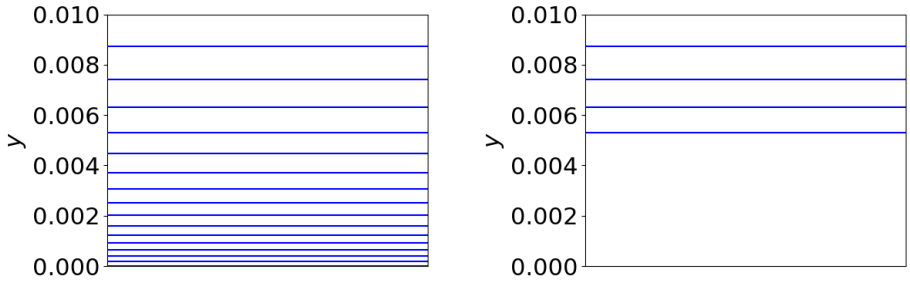
Figure 8: Channel flow. Reichardt's wall function. $Re_\tau = 16\,000$. Velocity. •: Reichardt's law, Eq. 9.

```
# compute ustar
ustar=np.divide(u2d_wall,uplus)
ustar_south=ustar
```

I compute U^+ using $\bar{u}$ and y^+ at the first wall-adjacent cells. In code lines colored red and blue I indicate how to use the second and third wall-adjacent cells, respectively. Furthermore, when I created the ML model, I stored `min` and `max` of y^+ and I use them to make sure that y^+ does not go outside the limits of the training process.

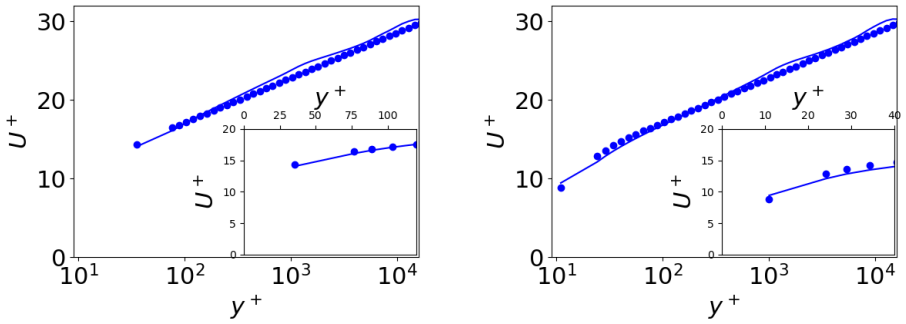
6 Results

Here I evaluate `svr` and Reichardt's wall function in fully-developed channel flow and a flat-plate boundary layer. The Python code **pyCALC-LES** is used for all



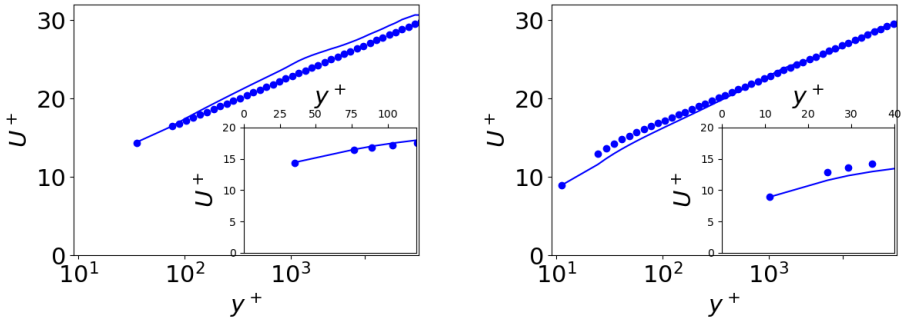
(a) Low-Re number IDDES grid. (b) Wall function grid. New grid strategy.

Figure 9: Different grids. —— : grid lines.



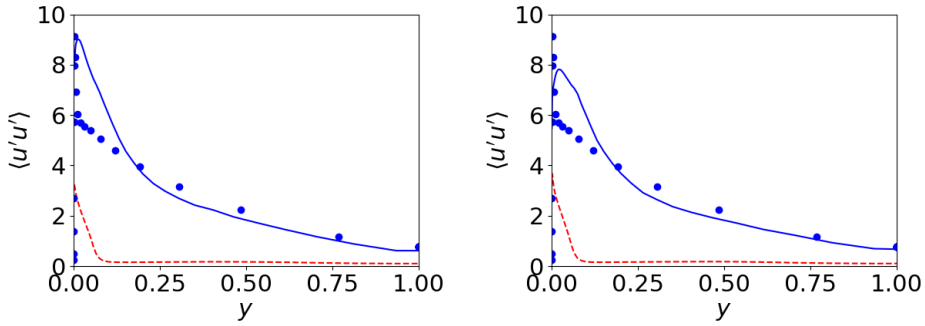
(a) $N_y = 78$. (b) $N_y = 92$.

Figure 10: Channel flow. $Re_\tau = 16\,000$. Velocity. `svr`. •: Reichardt's law, Eq. 9.



(a) $N_y = 78$. (b) $N_y = 92$.

Figure 11: Channel flow. $Re_\tau = 16\,000$. Velocity. Reichardt's wall function. •: Reichardt's law, Eq. 9.



(a) $N_y = 78$. (b) $N_y = 92$.

Figure 12: Channel flow. `svr`. $Re_\tau = 16\,000$. Streamwise fluctuations. ——: resolved plus modelled; – – : modeled. •: DNS data [13] at $Re_\tau = 5\,200$.

simulations together with the IDDES model, see Section 2.3.

6.1 Channel flow

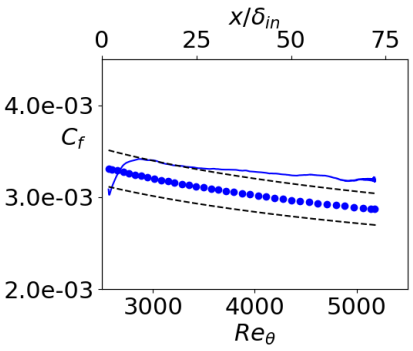
The Reynolds number is $Re_\tau = 16\,000$ based on the friction velocity and channel half width. The extent of the domain in the x and z direction is 3.2 and 1.6, respectively, covered by 32 cells in each direction.

First, I use a typical wall-function mesh ($N_y = 66$, stretching 11%) placing the wall-adjacent cells at $y^+ = 34$. The velocity profiles are shown in Figs. 7a and 8a and the agreement with Reichardt's law is excellent. However, when the stretching is increased to 14.7% the agreement deteriorates, see Figs. 7b and 8b. The reason for the poor agreement is probably that the cells further away from the wall (2nd, 3rd . . . cell) are too coarse.

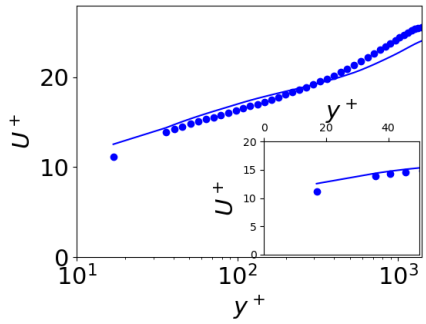
6.2 New grid strategy

Since the grid seems to be too coarse away from the figure in Figs. 7b and 8b a new grid strategy is proposed. Figure 9 presents two distributions of grid lines in the wall-normal direction. A low-Reynolds grid used in RANS and IDDES in Fig. 9a with stretching of 15%. Figure. 9b present the new grid in which a number of the near-wall cells are merged into one large wall-adjacent cell where the cell center should be located at $10 \leq y^+ \leq 100$. This strategy was used in [14] for channel flow and impinging jets (RANS). The new grid strategy is used in all simulations presented below.

Figures 10a and 11a present the velocity profiles using `svr` and Reichardt's wall function where the wall-adjacent cells are located at $y^+ = 35$ and the agreement is good (slightly better with `svr`). The agreement is also good when the wall-adjacent cells are located at $y^+ = 10$, see Figs. 10b and 11b (again, slightly

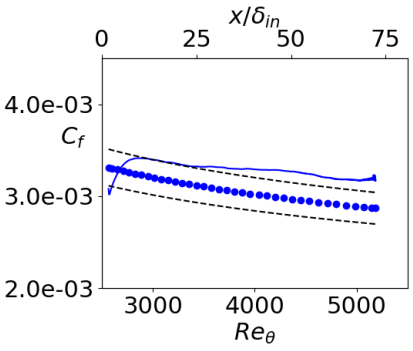


(a) Skin friction. Dashed lines: $\pm 6\%$. •: $2 \cdot ((1/0.384)\ln(Re_\theta) + 4.127)^{-2}$

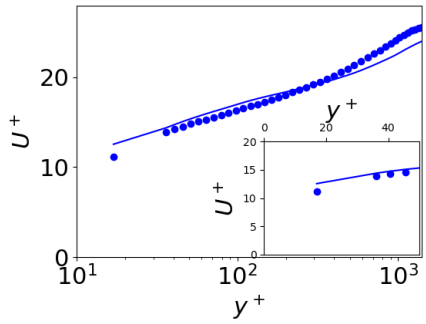


(b) Velocity at $Re_\theta = 4\,000$. $y^+ = 16$. Markers: DNS [15]

Figure 13: Boundary layer flow. `svr`. $N_y = 85$



(a) Skin friction. Dashed lines: $\pm 6\%$. •: $2 \cdot ((1/0.384)\ln(Re_\theta) + 4.127)^{-2}$



(b) Velocity at $Re_\theta = 4\,000$. $y^+ = 16$. Markers: DNS [15]

Figure 14: Boundary layer flow. Reichardt's wall function. $N_y = 85$

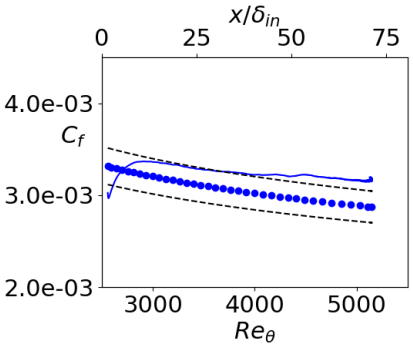
better with `svr`).

The streamwise fluctuations are shown in Fig. 12 for the two grids. The agreement with DNS data is good for $y \gtrsim 0.2$; the modeled fluctuations are negligible for $y \gtrsim 0.2$.

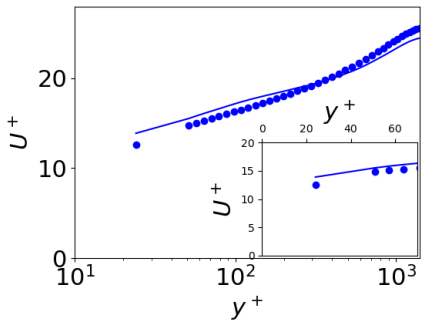
6.3 Flat-plate boundary layer

Now I will simulate a flat–plate boundary layer. The inlet Reynolds number based on the momentum thickness is $Re_\theta = 2\,550$. The data presented below are averaged in z direction and time.

The mean inlet profiles are taken from a 2D RANS solution at $Re_\theta = 2\,550$. Synthetic fluctuations [16, 17] are superimposed to the RANS velocity profile. The mesh has 550×64 cells in streamwise (x) and spanwise (z) directions. It has 82

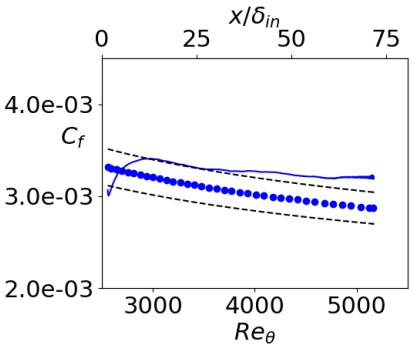


(a) Skin friction. Dashed lines: $\pm 6\%$. •: $2 \cdot ((1/0.384)\ln(Re_\theta) + 4.127)^{-2}$

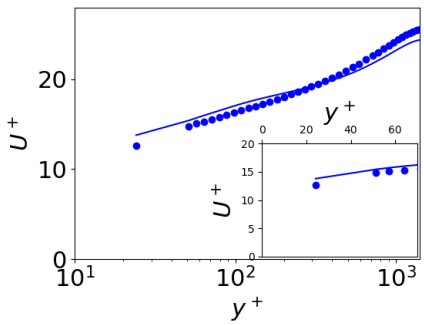


(b) Velocity at $Re_\theta = 4\,000$. $y^+ = 24$. Markers: DNS [15]

Figure 15: Boundary layer flow. `svr`. $N_y = 82$

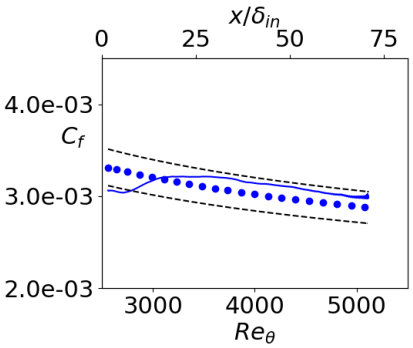


(a) Skin friction. Dashed lines: $\pm 6\%$. •: $2 \cdot ((1/0.384)\ln(Re_\theta) + 4.127)^{-2}$

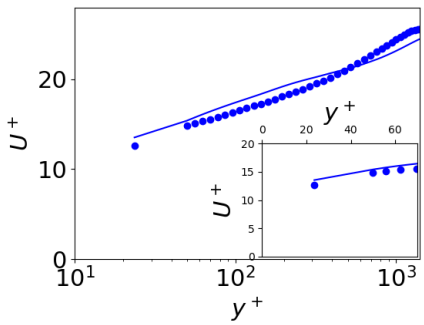


(b) Velocity at $Re_\theta = 4\,000$. $y^+ = 24$. Markers: DNS [15]

Figure 16: Boundary layer flow. Reichardt's wall function. $N_y = 82$

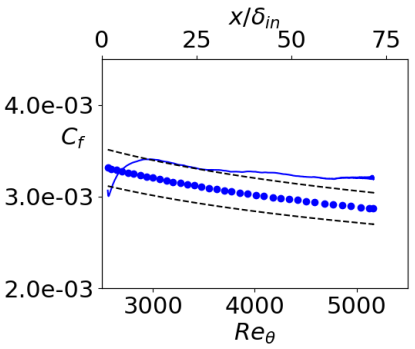


(a) Skin friction. Dashed lines: $\pm 6\%$. •: $2 \cdot ((1/0.384)\ln(Re_\theta) + 4.127)^{-2}$

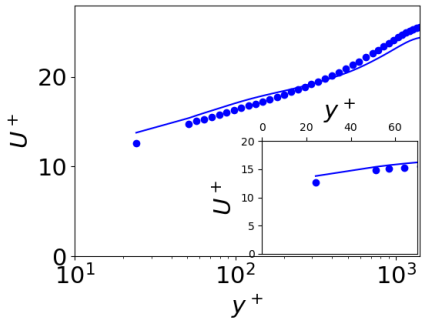


(b) Velocity at $Re_\theta = 4\,000$. $y^+ = 24$. Markers: DNS [15]

Figure 17: Boundary layer flow. `svr`. $N_y = 82$, $N_k = 32$, $\Delta x_{in} = 2\Delta x_{in,base}$

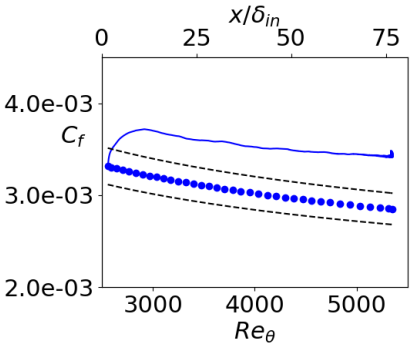


(a) Skin friction. Dashed lines: $\pm 6\%$. •: $2 \cdot ((1/0.384)\ln(Re_\theta) + 4.127)^{-2}$

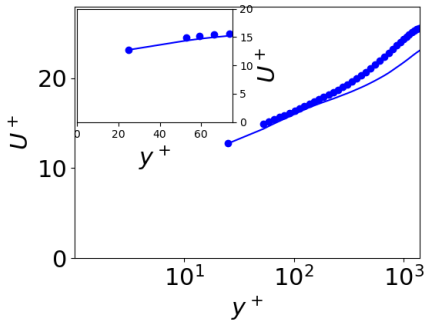


(b) Velocity at $Re_\theta = 4\,000$. $y^+ = 24$. Markers: DNS [15]

Figure 18: Boundary layer flow. Reichardt's wall function. $N_y = 82$, $N_k = 32$, $\Delta x_{in} = 2\Delta x_{in,base}$

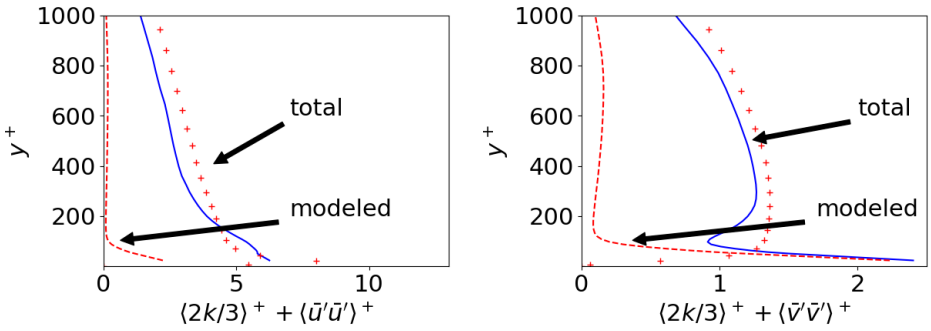


(a) Skin friction. Dashed lines: $\pm 6\%$. •: $2 \cdot ((1/0.384)\ln(Re_\theta) + 4.127)^{-2}$



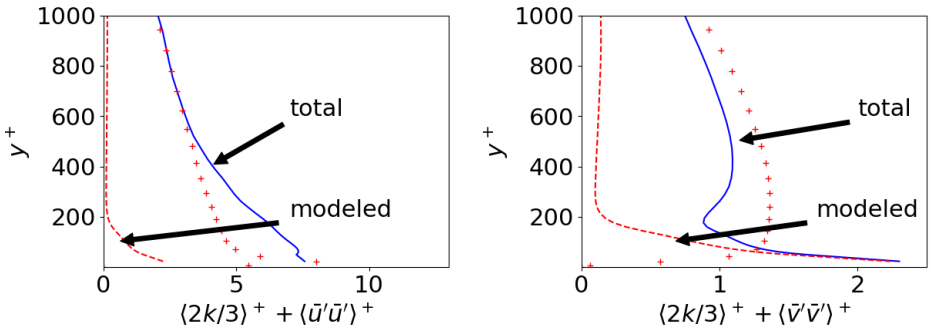
(b) Velocity at $Re_\theta = 4\,000$. $y^+ = 24$. Markers: DNS [15]

Figure 19: Boundary layer flow. `svr`. $N_y = 82$. u_τ is computed using the wall-adjacent cells.



(a) Streamwise. (b) Wall-normal.

Figure 20: Boundary layer flow. `svr`. $N_y = 82$. ——: total fluctuations; – – –: modeled fluctuations; Markers: DNS [15].



(a) Streamwise. (b) Wall-normal.

Figure 21: Boundary layer flow. `svr`. $N_y = 82$, $N_k = 32$, $\Delta x_{in} = 2\Delta x_{in,base}$ ——: total fluctuations; – – –: modeled fluctuations; Markers: DNS [15].

or 85 cells in the wall-normal direction (y). The domain size is $8.5 \times 4.6 \times 3.2$. The grid is stretched by 10% when $\Delta y < 0.05$ and when $y > 2$; $\Delta y_{max} = 0.1$. In the streamwise direction $\Delta x_{in} = 0.086$ and a 0.1% stretching is used. The inlet boundary-layer thickness is $\delta_{in} \simeq 0.8$.

In the k equation, I add a commutation term including $\partial f_k/\partial x$ at the plane adjacent to the inlet [18] (Model 3). The commutation term, C reads

$$C = U_{in} k_{tot} \frac{\partial f_k}{\partial x}$$

which is discretized as

$$C = U_{in} k_{tot,0} \frac{f_{k,0} - f_{k,in}}{\Delta x}$$

where subscript 0 denotes the cells adjacent to the inlet, $f_{k,in} = 1$ (RANS) and $k_{tot} = k_{res} + k$ where k_{res} is computed as a running average. $f_{k,0}$ is computed using the equivalence criterion, Eq. 16 in [19]. The purpose of the commutation term, C, is to reduce k near the inlet k from RANS (i.e. inlet) values to IDDES values.

Figures 13, 14, 15 and 16 present skin friction and velocity profiles using `svr` and Reichardt's wall function for two different resolutions in the wall-normal direction, namely the wall-adjacent cells at $y^+ = 16$ (Fig. 13 and 14) and $y^+ = 24$ (Figs. 15 and 16). The agreement is fairly good for both models and both grids (slightly better with `svr`).

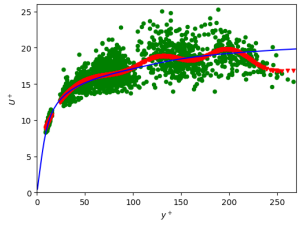
Figures 17 (`svr`) and 18 (Reichardt's wall function) show skin friction and velocity profiles using 50% fewer cells in x and z direction. The agreement is good with both models (again, slightly better with `svr`)

In the channel flow the wall-adjacent cells were used when computing u_τ (i.e. $\bar{u}_P$ and y^+ were taken at the first cell). In this flow, I use the third wall-adjacent cells (as proposed in [20, 21]), both for `svr` (see blue Python coding lines at p. 14) and Reichardt's wall function. Using the third cell in the channel flow gives negligible difference compared to the first cells (not shown). When I use the first cells in the boundary layer flow the agreement with experiment gets worse, see Figs. 19. Using Reichardt's wall function (not shown) gives virtually identical results.

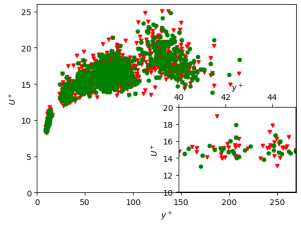
Finally, the total and modeled fluctuations are presented in Figs. 20 (baseline grid) and 21 (coarse grid). As can be seen, the agreement is satisfying but near the wall the streamwise fluctuations are somewhat over-predicted and the wall-normal are slightly under-predicted for both grid. The modeled fluctuation are negligible for $y^+ \gtrsim 150$.

7 Conclusions

A Machine Learning (ML) method (`svr`) is proposed for wall functions. IDDES is used for creating the training data. The IDDES is also used when doing the



(a) ——: $\langle\bar{u}\rangle$, IDDES; ▼: `svr`: ●: IDDES, test data. 9% normalized error.



(b) Nearest neighbor using Python's `scipy.spatial.KDTree` ●: IDDES, nearest neighbour.; ▼: IDDES training samples. 0.7% normalized error.

wall-function simulations, Good results are obtained for channel flow and flat-plate boundary for the Machine-Learning-Based wall functions, slightly better than wall functions based on Reichardt's law. In the channel-flow simulation the results are insensitive to the location of $\bar{U}$ and y^+ used for computing the wall-shear stress. For the flat-plate boundary layer, much better results are obtained by placing using $\bar{U}$ and y^+ at the third wall-adjacent cell.

Instantaneous IDDES data have been used for training `svr` (green markers in Fig. 22a) but the predicted `svr` data (red markers in Fig. 22a; this is called the hyper surface, see Fig. 1) follow the time-averaged IDDES data (blue line in Fig. 22a)). If I'm interested in predicting the *instantaneous* u_τ, I could find try to find nearest neighbour using Python's `scipy.spatial.KDTree` (shown by ● in Fig. 22b). The error between the `svr` predicted data and the IDDES data is then reduced from 9% (`svr`, Fig. 22a) to 0.7%, Fig. 22b). I'd call this model a *data-driven* wall function.

An alternative could be to train the ML using *time-averaged* data rather than instantaneous. In this case many time-averaged $\langle\bar{u}\rangle$ (and $\langle\bar{p}\rangle$) profiles can be used for training, both in attached and separated flows. Maybe `svm` (Support Vector Machines) and/or Neural Network could be used for finding the most appropriate profile. Then `svr` could be used for finding regression lines (or surfaces) in the selected profile.

Acknowledgments

This study was partly financed by *Strategic research project on Chalmers on hydro- and aerodynamics*.

The computations were enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC) at NSC partially funded by the Swedish Research Council through grant agreement no. 2018-05973.

References

[1] L. Davidson. pyCALC-LES: a Python code for DNS, LES and Hybrid LES-RANS. Division of Fluid Dynamics, Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, 2021.

[2] L. Davidson. Using Machine Learning for formulating new wall functions for Large Eddy Simulation: A first attempt. Technical report, Division of Fluid Dynamics, Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, 2018.

[3] Sudarshana S Rao and Santosh R Desai. Machine learning based traffic light detection and ir sensor based proximity sensing for autonomous cars. In *Proceedings of the International Conference on IoT Based Control Networks & Intelligent Systems – ICICNIS*, 2021. URL http://dx.doi.org/10.2139/ssrn.3883931.

[4] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas Schön. *Machine Learning: A First Course for Engineers and Scientists*. Cambridge University Press, 2022. ISBN 9781108843607. doi: 10.1017/9781108919371. URL http://smlbook.org/.

[5] Menneni Rachana, Jegadeesan Ramalingam, Gajula Ramana, Adigoppula Tejaswi, Sagar Mamidala, and G Srikanth. Fraud detection of credit card using machine learning. *GIS-Zeitschrift für Geoinformatik*, 8:1421–1436, 10 2021.

[6] Lorenzo Tieghi, Alessandro Corsini, Giovanni Delibra, and Francesco Aldo Tucci. A machine-learnt wall function for rotating diffusers. volume Volume 1: Aircraft Engine; Fans and Blowers of *Turbo Expo: Power for Land, Sea, and Air*, 09 2020. doi: 10.1115/GT2020-s̄15353. URL https://doi.org/10.1115/GT2020-s̄15353.

[7] Julia Ling, Matthew F. Barone, Warren Davis, Kamaljit Chowdhary, and Jeffrey Fike. Development of machine learning models for turbulent wall pressure fluctuations. In *55th AIAA Aerospace Sciences Meeting*, 2017. doi: 10.2514/6.2017-s̄0755. URL https://arc.aiaa.org/doi/abs/10.2514/6.2017-s̄0755.

[8] J. Dominique, J. Van den Berghe, C. Schram, and M. A. Mendez. Artificial neural networks modeling of wall pressure spectra beneath turbulent boundary layers. *Physics of Fluids*, 34(3):035119, 2022. doi: 10.1063/5.0083241. URL https://doi.org/10.1063/5.0083241.

[9] H.J Bae and P. Koumoutsakos. Scientific multi-agent reinforcement learning for wall-models of turbulent flows. *Nat Commun*, 13(1443), 2022. doi: https://doi.org/10.1038/s41467-s̄022-s̄28957-s̄7. URL https://doi.org/10.1038/s41467-s̄022-s̄28957-s̄7.

[10] L. N. Olson and J. B. Schroder. PyAMG: Algebraic multigrid solvers in Python v4.0, 2018. URL https://github.com/pyamg/pyamg. Release 4.0.

[11] K. Abe, T. Kondoh, and Y. Nagano. A new turbulence model for predicting fluid flow and heat transfer in separating and reattaching flows - 1. Flow field calculations. *Int. J. Heat Mass Transfer*, 37(1):139–151, 1994.

[12] M. L. Shur, P. R. Spalart, M. Kh. Strelets, and A. K. Travin. A hybrid RANS-LES approach with delayed-DES and wall-modelled LES capabilities. *International Journal of Heat and Fluid Flow*, 29:1638–1649, 2008. URL https://doi.org/10.1016/j.ijheatfluidflow.2008.07.001.

[13] 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.

[14] J.-A. Bäckar and L. Davidson. Evaluation of numerical wall functions on the axisymmetric impinging jet using OpenFOAM. *International Journal of Heat and Fluid Flow*, 67:27–42, 2017.

[15] J.A. Sillero, J. Jimenez, and R.D. Moser. One-point statistics for turbulent wall-bounded flows at Reynolds numbers up to $\delta^+ \simeq 2000$. *Physics of Fluids*, 25(105102), 2014.

[16] M. Shur, P.R. Spalart, M.K. Strelets, and A.K. Travin. Synthetic turbulence generators for RANS-LES interfaces in zonal simulations of aerodynamic and aeroacoustic problems. *Flow, Turbulence and Combustion*, 93:69–92, 2014.

[17] M. Carlsson, L. Davidson, S.-H. Peng, and S. Arvidson. Investigation of turbulence injection methods in large eddy simulation using a compressible flow solver. In *AIAA Science and Technology Forum and Exposition, AIAA SciTech Forum*, 2022.

[18] L. Davidson. Zonal PANS: evaluation of different treatments of the RANS-LES interface. *Journal of Turbulence*, 17(3):274–307, 2016. URL http://dx.doi.org/10.1080/14685248.2015.1093637.

[19] L. Davidson and C. Friess. A new formulation of f_k for the PANS model. *Journal of Turbulence*, pages 1–15, 2019. doi: 10.1080/14685248.2019.1641605. URL http://dx.doi.org/10.1080/14685248.2019.1641605.

[20] Soshi Kawai1 and Johan Larsson. Wall-modeling in large eddy simulation: Length scales, grid resolution, and accuracy. *Physics of Fluids*, 24:015105, 2012. URL https://doi.org/10.1063/1.3678331.

[21] T. Mukha, R.E. Bensow, and M. Liefvendahl. Predictive accuracy of wall-modelled large-eddy simulation on unstructured grids. *Computers & Fluids*, 221:104885, 2021. ISSN 0045-7930. doi: https://doi.org/10.1016/j.compfluid.2021.104885. URL https://www.sciencedirect.com/science/article/pii/S0045793021000517.