ERCOFTAC symposium on Engineering, Turbulence, Modelling and Measurements (ETMM14)
in Mini-Symposium: Machine learning for turbulence, Dubrovnik on 22-24 September 2025
Eq. 6 is corrected

USING PHYSICS INFORMED NEURAL NETWORK (PINN) TO IMPROVE A $k-\omega$ TURBULENCE MODEL

Lars Davidson
M2 Fluid Dynamics Chalmers University of Technology
Gothenburg, Sweden, *lada@chalmers.se*

Abstract

The Wilcox $k-\omega$ turbulence model predicts turbulent boundary layers well, both fully-developed channel flows and flat-plate boundary layers. However, it predicts too low a turbulent kinetic energy. This is a feature it shares with most other two-equation turbulence models. When comparing the terms in the k equations with DNS data it is found that the production and dissipation terms are well predicted but the turbulent diffusion is not. In the present work the poor modeling of the turbulent diffusion is improved by making the turbulent diffusion constant, σ_k, a function of y/δ. By taking the production, the dissipation terms as well as the turbulent kinetic energy from DNS channel flow data, the k equation is turned into an ordinary differential equation for the turbulent viscosity which is solved using Physics Informed Neural Network (PINN). In order to not change the predicted turbulent viscosity – which is well predicted by the standard Wilcox $k-\omega$ turbulence model – a new function is added to the dissipation term and the destruction term in the ω equation. The new model, called the $k-\omega$ NN model, is shown to produce excellent velocity and turbulent kinetic profiles in channel flow at $Re_\tau = 2\,000$, $Re_\tau = 5\,200$, $Re_\tau = 10\,000$ as well as in flat-plate boundary layer flow. The Python PINN script and the pyCALC-RANS code can be downloaded (Davidson, 2025).

1 Introduction

Eddy-viscosity RANS (Reynolds-Averaged Navier-Stokes) turbulence models in the literature have been developed with the object of predicting a correct velocity field. $\bar{v}_i$. The most common turbulence models are the $k-\varepsilon$ and the $k-\omega$ models where k, ε and ω denote the turbulent kinetic energy, its dissipation rate and the specific dissipation $\omega = \varepsilon/(\beta^* k)$, respectively. Both turbulence models include five tuning constants. These constants can be improved using PINN.

Yazdani and Tahani (2024) use PINN to optimize the five constants α, β^*, β, σ_k and σ_ω in the $k-\omega$ model. DNS data of the flow over a periodic hill at $Re = 5600$ are used. Using these DNS data, they compute all terms in the 2D RANS equations comprising of the two momentum equations ($\bar{v}_1$ and $\bar{v}_2$), the continuity equation, the k and ω equations. The loss function is defined as the sum of $(\hat{Q}_i^n - \tilde{Q}_i^n)^2$ where $\hat{Q}$ and $\tilde{Q}$ denote DNS value and PINN value, respectively. Subscript i represents 5000 arbitrary chosen DNS data points and superscript n corresponds to $\bar{v}_1$, $\bar{v}_1$, continuity equation, k or ε. The residuals (square of L_2 norm) of the five governing equations, Q^n, are added to the loss function. They use PINN to find optimal values of the five coefficient in the $k-\omega$ model. PINN gives modified values for $\alpha = 2.9719$ and $\sigma_\omega = 1.2685$ (baseline values are 2 and 0.52) whereas the other three constants are not changed. Then they carry out RANS simulations of the flow over the same periodic hill that was used for training and compare with RANS simulations using the standard values for the coefficients. They find that the new PINN coefficients give somewhat better results.

Luo et al. (2020) improve the five constants. C_μ, $C_{\varepsilon 1}$, $C_{\varepsilon 2}$, σ_k and σ_ε in the $k-\varepsilon$ model using PINN. They define the loss function as $(\hat{Q}_i^n - \tilde{Q}_i^n)^2$ (see above) at the DNS data points where Q^n is k or ε. Subscript i represents all DNS data points. The residuals of the transport equations of k and ε, multiplied by penalty functions, are added to the loss function. All terms in the k and ε equations are taken from DNS of a converging-diverging channel. New values of the constants are found by PINN (C_μ keeps its standard value of 0.09). Finally, RANS simulations are carried out for the converging-diverging channel flow (the sane flow that was used when training with PINN) using the new PINN-optimized $k-\varepsilon$ constants. Somewhat better results are obtained compared with the standard $k-\varepsilon$ constants.

Thakur et al. (2024) use PINN to predict a diffusion coefficient. They study an unsteady, two dimensional convection-diffusion concentration equation (i.e. a PDE), $c = c(x, y, t)$, with a spatially dependent diffusion coefficient, $D = D(x, y)$. The object is to predict D. The loss function, L, is

$$L = \frac{1}{N\sigma_c}\sum_{i=1}^{N}|\tilde{c} - c|^2 + \frac{1}{N^e\sigma_c}\sum_{i=1}^{N_e}|\tilde{c} - c^p|^2$$

where c denotes true data predicted with CFD using a prescribed (true), varying diffusion coefficient (D_{true} varies linearly, sin, tanh etc) and c^p is obtained from the explicit unsteady diffusion equation using $\tilde{c}$ at the old time step. σ_c is the standard deviation of the concentration. D and $\tilde{c}$ are obtained from two neural networks.

In simple flows including a boundary layer (chan-

nel flow, flat-plate boundary layer flow, jet flow etc) the turbulent shear stress, $\overline{v_1' v_2'}$, must be correctly predicted. The turbulent shear stress is in these models linearly coupled to the turbulent viscosity, ν_t, via the Boussinesq assumption. Hence, well-tuned eddy viscosity models correctly predict the mean flow, the turbulent shear stress and the turbulent viscosity. However, the turbulent, kinetic energy, k, is usually poorly predicted, see Fig. 1. Figure 1 present a prediction using the Wilcox $k-\omega$ turbulence model (Wilcox, 1988) of fully developed channel flow at $Re_\tau \equiv u_\tau \delta/\nu = 5\,200$ where u_τ and δ denote friction velocity and half-channel width, respectively. It can be seen that the $k-\omega$ model behaves as outlined above: the mean flow (and hence the turbulent shear stress and the turbulent viscosity) is well predicted but the predicted turbulent kinetic energy is much too small.

The object of the present paper is to improve the predicted, turbulent kinetic energy while not deteriorating the predicted mean flow. The predicted terms – production, dissipation and diffusion – in the k equation using the $k-\omega$ model are compared with DNS in Fig. 1c. It can be seen that the production term, P^k, and the sum of the viscous diffusion and the dissipation term, $D^\nu - \varepsilon$, agree fairly well with DNS but that the agreement for the turbulent diffusion term, D^k, is not so good. In order to improve the predicted k, we will use Physics Informed Neural Network, i.e. Physics Informed NN – usually called PINN – to improve the predicted diffusion term. In the works of Yazdani and Tahani (2024) and Luo et al. (2020) summarized above, new constant values of turbulent coefficients were optimized. In the present study, one turbulent constant, σ_k, will be turned into a function of y/δ where δ denotes the boundary layer thickness.

```
class NN(nn.Module):
 def __init__(self):
  self.lay_1=nn.Linear(1, 2) # Connection 0-1
  self.lay_2=nn.Linear(2, 1) # Connection 1-2
 def forward(self, x):
  x = torch.nn.functional.sigmoid(self.lay_1(x))
  out = torch.nn.functional.sigmoid(self.lay_2(x))
```

Listing 1: Simple NN. Initiation and forward step

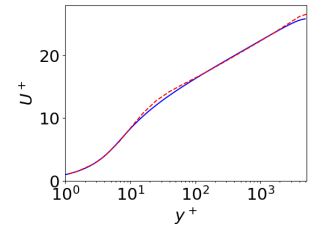
2 Equations and the numerical method

The RANS equations are given by

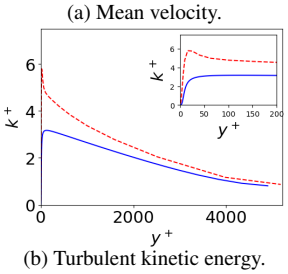
$$\begin{aligned}
\frac{\partial \bar{v}_i}{\partial x_i} &= 0 \\
\frac{\partial \bar{v}_i \bar{v}_j}{\partial x_j} &= -\frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left[(\nu+\nu_t)\frac{\partial \bar{v}_i}{\partial x_j}\right]
\end{aligned}$$

The Wilcox $k-\omega$ turbulence model reads

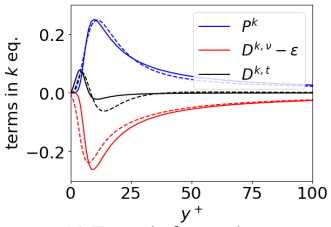
$$\begin{aligned}
\frac{\partial \bar{v}_j k}{\partial x_j} &= P^k + \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_j}\right] - C_\mu k\omega \\
\frac{\partial \bar{v}_j \omega}{\partial x_j} &= C_{\omega_1}\frac{P^k}{\nu_t} + \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_\omega}\right)\frac{\partial \omega}{\partial x_j}\right] - C_{\omega 2}\omega^2 \\
P^k &= \nu_t\left(\frac{\partial \bar{v}_i}{\partial x_j} + \frac{\partial \bar{v}_j}{\partial x_i}\right)\frac{\partial \bar{v}_i}{\partial x_j}, \quad \nu_t = \frac{k}{\omega}
\end{aligned} \tag{1}$$



(a) Mean velocity.



(b) Turbulent kinetic energy.



(c) Terms in k equation.

Figure 1: Fully-developed channel flow. Solid lines: $k-\omega$; dashed lines:DNS (Lee and Moser, 2015).

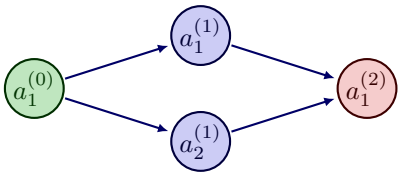


Figure 2: Schematic of Simple NN

The standard coefficients are used, i.e. $C_{\omega 1} = 5/9$, $C_{\omega 2} = 3/40$, $\sigma_k = \sigma_\omega = 2$ and $C_\mu = 0.09$.

The **pyCALC-RANS** code is used (Davidson, 2021) for solving the discretized equations. It is an incompressible, finite volume code written in Python. It is fully vectorized (i.e. no for loops). The convective terms in all equations are discretized using the Hybrid central/upwind scheme. The numerical procedure is based on the pressure-correction method, SIMPLEC, and a collocated grid arrangement using Rhie-Chow interpolation.

3 Physics informed NN (PINN)

Let's create a simple NN that finds a damping function, $Y \equiv f$, as a function of $X \equiv y^+$, see Fig. 2. It has one input ($X = a_1^{(0)}$), one hidden layer with two neurons ($a_1^{(1)}, a_2^{(1)}$) and one output ($Y = a_1^{(2)}$). In Listing 1, you find the line labeled `Connection 0-1` which connects $a_1^{(0)}$ and $\left(a_1^{(1)}, a_2^{(1)}\right)$ and the line labeled `Connection 1-2`

which connects $\left(a_1^{(1)}, a_1^{(1)}\right)$ and $a_1^{(2)}$.

Now we formulate the NN with weights, w, and biases, b and add Sigmoid activators, s, to both neurons, i.e.

$$\begin{aligned}
\text{Activation 1:} \quad & a_1^{(1)} = s_1^{(1)}\left(w_1^{(0)} a_1^{(0)} + b_1^{(0)}\right) \\
\text{Activation 2:} \quad & a_2^{(1)} = s_2^{(1)}\left(w_2^{(0)} a_1^{(0)} + b_2^{(0)}\right) \\
\text{Output:} \quad & a_1^{(2)} = s_1^{(2)}\left(w_1^{(1)} a_1^{(1)} + b_1^{(1)}\right. \\
& \left. + w_2^{(1)} a_2^{(1)} + b_2^{(1)}\right) \equiv Y
\end{aligned}$$

The Python code is given in Listing 2.

```
#  initiate the NN model
model = NN()
# define input, X
X=np.zeros(nj,1))
X[:,0] = scaler_yplus.fit_transform(yplus)[:,0]
# define output, Y (f is known)
Y = f
# Training loop
for epoch in range(max_no_epoch):
# Compute prediction and loss, L
 o = model(X) #prediction
 L = loss_fn(o, Y)   # L = |o-Y|_2
 L.backward()
```

Listing 2: Simple NN. Prediction and backward step

The `loss.backward()` command computes the gradients of the loss, L, with respect to the weights, biases and activators, (i.e. $\partial L/\partial w_1$, $\partial L/\partial b_1$, $\partial L/\partial s_1 \ldots$) in order to get new improved $w_1, b_1, s_1 \ldots$

Now, let's involve our differential equation, the k equation (see Eq. 1). In fully-developed channel flow it is a diffusion equation with source terms which reads

$$\frac{d}{dy}\left(\nu + \nu_{t,NN}\frac{dk}{dy}\right) + P^k - \varepsilon = Q.$$

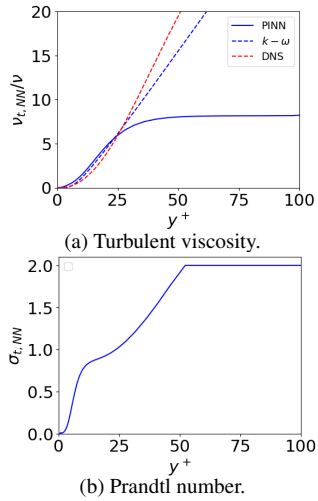
It is re-written as

$$(\nu + \nu_{t,NN})\frac{d^2k}{dy^2} + \frac{dk}{dy}\frac{d\nu_{t,NN}}{dy} + P^k - \varepsilon = Q \quad (2)$$

We want to find a new turbulent viscosity, $\nu_{t,NN}$, that gives a turbulent diffusion that agrees with the DNS turbulent diffusion term in Fig. 1c. Hence, $\nu_{t,NN}$ is the unknown variable in Eq. 2 and k, P^k and ε are taken from DNS. Equation 2 is solved in half a channel at $Re_\tau = 5\,200$. The boundary condition at the wall ($y = 0$) is $\nu_{t,NN} = 0$ and at the center ($y = \delta$) it is $\nu_{t,DNS}$.

The turbulent viscosity, $\nu_{t,NN}$ in Eq. 2, will be predicted by PINN while minimizing the error Q^2. The loss function, `loss_fn`, in Listing 2 is replaced with Eq. 2 and the Python code is given in Listing 3.

```
def ODE(y, nut):
 nut_y = grad(nut, y, torch.ones\
   (x.size()[0], 1,),create_graph=True)[0]
# Differential equation loss
 ODE_loss = (nu+nut)*k_yy + k_y*nut_y + Pk - eps
 ODE_loss = torch.sum(ODE_loss ** 2)
# b.c. loss
```



(a) Turbulent viscosity.

(b) Prandtl number.

(c) Turbulent diffusion.

Figure 3: Prediction by PINN compared to DNS and the Wilcox $k-\omega$ model. $Re_\tau = 5\,200$.

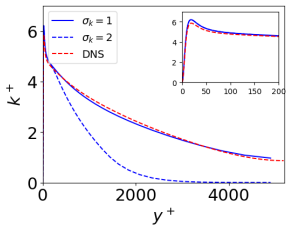


Figure 4: CFD predictions with different $\sigma_{k,max}$.

```
 BC_loss = (nut[0] - nut_0) ** 2
 return ODE_loss, BC_loss
loss_ODE, loss_bc = ODE(y,o)
```

Listing 3: Python code for PINN.

`nut` and `k` in Listing 3 are $\nu_{t,NN}$ and k in Eq. 2, respectively. `k_y`, for example, is dk/dy. Note that `k_y` and `k_yy` are known and constant. There are two losses in Listing 3, one, `ODE_loss`, which relates to the ODE and one, `BC_loss`, which relates to the boundary conditions. In this way the PINN is forced to satisfy both the ODE and the boundary conditions.

Figures 3a and 3b present the predicted turbulent viscosity in the k equation, $\nu_{t,NN}$, and the turbulent Prandtl number, $\sigma_{t,NN}$, where $\sigma_{t,NN} = \nu_t/\nu_{t,NN}$ (ν_t is the turbulent viscosity predicted by the Wilcox $k-\omega$). It is compared to the DNS turbulent diffusion in Fig. 3c, and the agreement is good. The turbulent diffusion goes to zero at $y^+ \simeq 40$ and the turbulent diffusion is negligible for $y^+ \gtrsim 40$ (see Fig. 1c). Hence, the value of $\nu_{t,NN}$ (and $\sigma_{t,NN}$) is not relevant in the outer region and $\sigma_{t,NN}$ is set to a constant in this re-

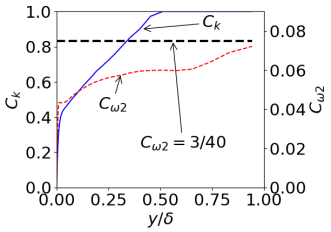


Figure 5: C_k and $C_{\omega 2}$ vs. y/δ.

gion (to be determined).

Now we have modified the turbulent Prandtl number in the k equation so that we – with exact source terms taken from DNS, P^k and ε – predict a correct near-wall behaviour of k. In the next step, we have to take the source terms from the $k-\omega$ model. Recall that P^k in the $k-\omega$ model is correct since the model does predict the velocity profile correctly, see Fig. 1 and hence the turbulent viscosity (the total shear stress is predicted as $y-1$ by all turbulence models). The k predicted by DNS is near the wall much larger than that predicted by the $k-\omega$ model, see Fig. 1b. This means that ω must be modified in order to give the same turbulent viscosity as the $k-\omega$ model, i.e.

$$\nu_{t,k-\omega} = \frac{k_{k-\omega}}{\omega_{k-\omega}} = \nu_{t,DNS} = \frac{k_{DNS}}{\omega_{DNS}} = \nu_{t,NN} \quad (3)$$

which gives

$$\omega_{DNS} = k_{DNS}/\nu_{t,DNS}. \quad (4)$$

The turbulent Prandtl number in the k equation has now been modified (Fig. 3b) and we have found an expression for a correct ω (Eq. 4). Let's verify that a CFD solver does predict a correct turbulent kinetic energy. Equation 2 is formulated as an equation for k, i.e.

$$\frac{d}{dy}\left(\left(\nu + \frac{\nu_{t,k-\omega}}{\sigma_{t,NN}}\right)\frac{dk}{dy}\right) + P^k - \varepsilon = 0 \quad (5)$$

where P^k and ε are again taken from DNS. The turbulent viscosity, $\nu_{t,k-\omega}$, is taken from the standard $k-\omega$ model and the turbulent Prandtl number is computed using the DNS value of ω, i.e.

$$\sigma_{t,NN} = \frac{k/\omega_{DNS}}{\nu_{t,NN}} \quad (6)$$

We solve Eq. 5 using the CFD solver **pyCALC-RANS** . Figure 4 presents the predicted k profiles using two different turbulent Prandtl numbers. We find that when using $\sigma_{k,max} = 1$ the agreement is excellent. The larger value $\sigma_{k,max} = 2$ gives also excellent agreement near the wall for $y^+ < 200$ but further out is gives much too small a k. The reason is that the diffusion in the outer region is too small because of a small dk_{DNS}/dy. However, when solving the full equation system (i.e. $\bar{v}_1$, k and ω) the k profile in the

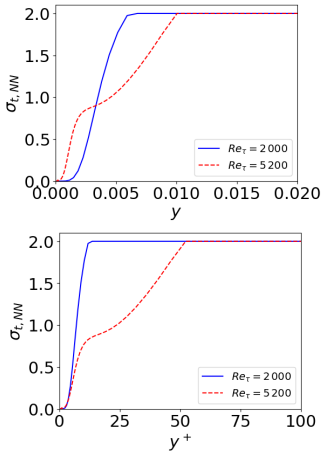


Figure 6: Turbulent Prandtl number vs. y and y^+.

outer region must adapt so that the total shear stress satisfies $y-1$. In Section it is found that $\sigma_{k,max} = 1$ and using no limit give identical predictions. The reason is that the turbulent diffusion in the k equation is negligible in the outer region, see Fig. 1c. In the Wilcox $k-\omega$ model $\sigma_k = 2$ and hence $\sigma_{k,max}$ is set to two.

Now we know ω_{DNS}. Next, the dissipation term $C_\mu k\omega$ in the k equation in Eq. 1 must be modified so that it agrees with $\varepsilon_{DNS} = C_\mu k_{DNS}\omega_{DNS}$. This is achieved by multiplying the dissipation term by a damping function, $C_k = C_k\,(y/\delta)$, so that the k equation in Eq. 1 is satisfied with $k = k_{DNS}$ and $\omega = \omega_{DNS}$, i.e. (see Eq. 5)

$$C_k = \frac{\frac{d}{dy}\left(\frac{\nu_t}{\sigma_{t,NN}}\frac{dk_{DNS}}{dy}\right) + P^k_{DNS}}{C_\mu k_{DNS}\omega_{DNS}} \quad (7)$$

Note that the viscous diffusion has been omitted in order to prevent a large gradient of C_k close to the wall.

Finally, we must make sure that the ω equation in Eq. 1 predicts $\omega = \omega_{DNS}$. This is achieved by making $C_{\omega 2} = C_{\omega 2}\,(y/\delta)$. From the ω equation in Eq. 1 we get

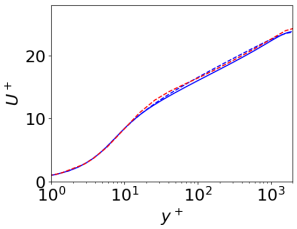
$$C_{\omega 2} = \frac{\frac{d}{dy}\left(\frac{\nu_t}{\sigma_\omega}\frac{d\omega_{DNS}}{dy}\right) + C_{\omega 1}\frac{P^k_{DNS}}{\nu_{t,DNS}}}{\omega^2_{DNS}} \quad (8)$$

Again, the viscous diffusion has been omitted.

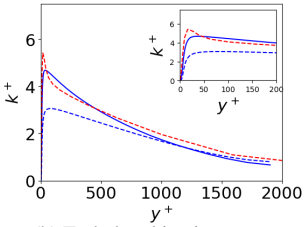
Figure 5 shows the two damping functions. The thick, black, dashed line indicates the standard value of $C_{\omega,2} = 3/40$, see below Eq. 1. The damping function, C_k, is essentially $(k_{DNS}/k_{k-\omega})^2$ because of the denominator in Eq. 7: first, k_{DNS} is larger than $k_{k-\omega}$ (see Fig. 4), and, second, ω_{DNS} is larger than $\omega_{k-\omega}$, see Eq. 3.

4 Results

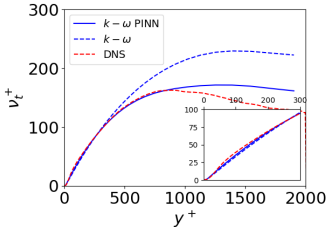
The turbulent Prandtl number, $\sigma_{t,NN} = \sigma_{t,NN}\,(y/\delta)$, $C_k = C_k\,(y/\delta)$ and $C_{\omega 2} = C_{\omega 2}\,(y/\delta)$ (see Figs. 3b and 5) are obtained using PINN and DNS



(a) Velocity.

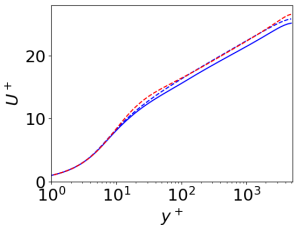


(b) Turbulent kinetic energy.

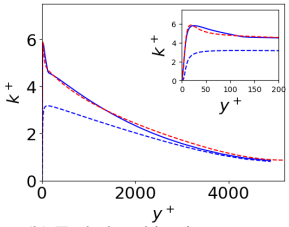


(c) Turbulent viscosity.

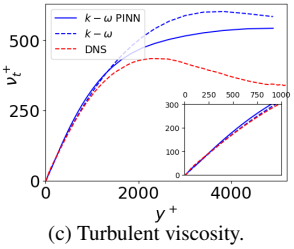
Figure 7: Fully-developed channel flow. $Re_\tau = 2\,000$.



(a) Velocity.

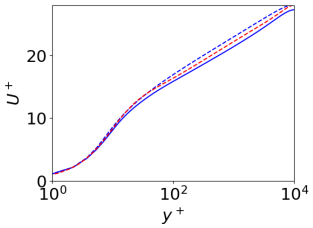


(b) Turbulent kinetic energy.

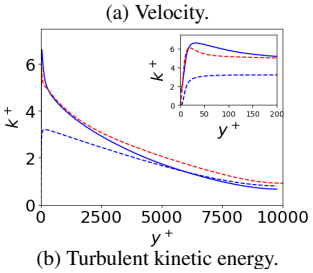


(c) Turbulent viscosity.

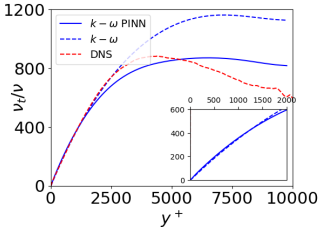
Figure 8: Fully-developed channel flow. $Re_\tau = 5\,200$.



(a) Velocity.

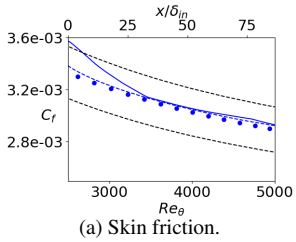


(b) Turbulent kinetic energy.

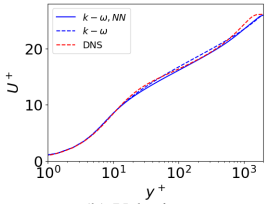


(c) Turbulent viscosity.

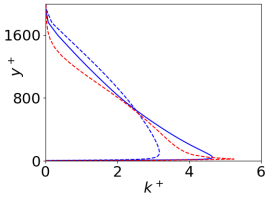
Figure 9: Fully-developed channel flow. $Re_\tau = 10\,000$.



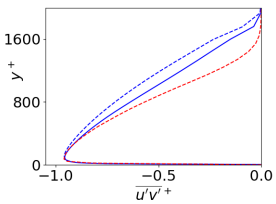
(a) Skin friction.



(b) Velocity.



(c) Turbulent kinetic energy.



(d) Turbulent shear stress.

Figure 10: Flat-plate boundary layer. Profiles at $Re_\theta = 4\,500$.

data for fully-developed channel flow at $Re_\tau = 5\,200$ in the previous section. They are in this section used in the $k-\omega$ model. The new model is called the $k-\omega$-NN model. Note that $\sigma_{t,NN}$, C_k and $C_{\omega 2}$ are expressed as function of y/δ rather than of y^+. The reason is that when repeating the predictions in the previous section using data at $Re_\tau = 2\,000$ it is found that $\sigma_{t,NN}$ scales much better with y/δ than with y^+, see Fig. 6.

Fully-developed channel flows at $Re_\tau = 2\,000$, $Re_\tau = 5\,200$ and $Re_\tau = 10\,000$ are simulated using **pyCALC-RANS** . The predicted velocity, turbulent kinetic energy and turbulent viscosity using the Wilcox $k-\omega$ model and the $k-\omega$-NN model are compared with DNS in Figs. 7, 8 and 9. The velocity profiles are well predicted with both models but the turbulent kinetic energy is much better predicted with the $k-\omega$-NN model. It can be seen that the turbulent viscosities predicted by the two turbulence models are very similar in the inner part of the boundary layer ($y^+ \lesssim 300$, $y^+ \lesssim 1\,000$ and $y^+ \lesssim 2\,000$ for $Re_\tau = 2\,000$, $Re_\tau = 5\,200$ $Re_\tau = 10\,000$, respectively). It may be recalled that this was indeed a requirement when developing the $k-\omega$ NN model in the previous section, see Eq. 3.

5 Conclusions

The present work proposes a methodology for using PINN for improving turbulence models. A new $k-\omega$ turbulence model, $k-\omega$ NN, has been presented. By making the turbulent Prandtl number in the k equation a function of y/δ we were able to predict k near the wall in agreement with DNS which the Wilcox $k-\omega$ model fails to do. The $k-\omega$ NN model predicts a larger k near the wall but the same turbulent viscosity (which is necessary in order to predict an accurate velocity field). In order to keep ν_t the same in the $k-\omega$ NN model as in the Wilcox $k-\omega$ model, we had to introduce $C_k = C_k\,(y/\delta)$ in front of the dissipation term in the k equation and $C_{\omega 2}$ in the ω equation was made a function of y/δ.

In this work σ_k, C_K and $C_{\omega 2}$ are made functions of y/δ. Hence, the current formulation of the model is not applicable to re-circulating flow. Using Neural Network (NN), we have tried to make them functions of P_k/ε, P_k^+, $\nu_t/(yu_\tau)$, etc. It is important that they are properly non-dimensionalized so that can be used in other flows as well at other Reynolds numbers. In the end, a good combination was found: σ_k, C_k and $C_{\omega 2}$ can be made functions of $\overline{u'v'}/u_\tau^2$ and $\nu_t/(yu_\tau)$. The predictions (not shown) using σ_k, C_k and $C_{\omega 2}$ based on NN are very good for all three channel flows. However, the skin friction for the flat-plate boundary layer is over-predicted by some 10%.

References

Davidson L. pyCALC-RANS: a 2D Python code for RANS. Division of Fluid Dynamics, Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, 2021. URL https://www.tfd.chalmers.se/~lada/pyCALC-RANS.html.

Davidson L. Using physical informed neural network (PINN) to improve a $k-\omega$ turbulence model: Python CFD code and PINN script. Division of Fluid Dynamics, Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, 2025. URL https://www.tfd.chalmers.se/~lada/Using-Physical-Informed-Neural-Network-PINN-improve-a-k-omega-turbulence-model.html.

Lee M. and Moser R. D. 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.

Luo S., Vellakal M., Koric S., Kindratenko V., and Cui J. Parameter identification of RANS turbulence model using physics-embedded neural network. In Jagode H., Anzt H., Juckeland G., and Ltaief H., editors, *High Performance Computing*, pages 137–149, Cham, 2020. Springer International Publishing. URL https://link.springer.com/book/10.1007/978-3-030-59851-8.

Thakur S., Esmaili E., Libring S., Solorio L., and Ardekani A. M. Inverse resolution of spatially varying diffusion coefficient using physics-informed neural networks. *Physics of Fluids*, 36(8):081915, 08 2024. ISSN 1070-6631. doi: 10.1063/5.0207453. URL https://doi.org/10.1063/5.0207453.

Wilcox D. C. Reassessment of the scale-determining equation. *AIAA Journal*, 26(11):1299–1310, 1988.

Yazdani S. and Tahani M. Data-driven discovery of turbulent flow equations using physics-informed neural networks. *Physics of Fluids*, 36(3):035107, 03 2024. ISSN 1070-6631. doi: 10.1063/5.0190138. URL https://doi.org/10.1063/5.0190138.