Using physics informed neural network (PINN) and neural network (NN) to improve a k-ω turbulence model

Downloaded from: https://research.chalmers.se, 2026-08-08 21:24 UTC

Citation for the original published paper (version of record):
Davidson, L. (2026). Using physics informed neural network (PINN) and neural network (NN) to improve a k-ω
turbulence model. Journal of Turbulence, 27(7): 187-208.
http://dx.doi.org/10.1080/14685248.2026.2665148

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

Journal of Turbulence

ISSN: 1468-5248 (Online) Journal homepage: www.tandfonline.com/journals/tjot20

Using physics informed neural network (PINN) and neural network (NN) to improve a $k-\omega$ turbulence model

Lars Davidson

To cite this article: Lars Davidson (2026) Using physics informed neural network (PINN) and neural network (NN) to improve a $k-\omega$ turbulence model, Journal of Turbulence, 27:7, 187-208, DOI: 10.1080/14685248.2026.2665148

To link to this article: https://doi.org/10.1080/14685248.2026.2665148

© 2026 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group.

Published online: 02 May 2026.

Submit your article to this journal

Article views: 1741

View related articles

View Crossmark data

Citing articles: 1 View citing articles

Full Terms & Conditions of access and use can be found at
https://www.tandfonline.com/action/journalInformation?journalCode=tjot20

JOURNAL OF TURBULENCE
2026, VOL. 27, NO. 7, 187–208
https://doi.org/10.1080/14685248.2026.2665148

Taylor & Francis
by informa

OPEN ACCESS Check for updates

Using physics informed neural network (PINN) and neural network (NN) to improve a $k-\omega$ turbulence model

Lars Davidson

Div. of Fluid Dynamics, Dept of Mechanical Engineering, Chalmers University of Technology, Gothenburg, Sweden

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 modelling of the turbulent diffusion is improved using Physics Informed Neural Network (PINN) and Neural Network (NN). The k equation is turned into an ordinary differential equation for the turbulent viscosity in the k equation, $\nu_{t,PINN}$, which is solved using PINN. A new turbulent Prandtl number is then computed as $\sigma_{k,PINN} = \nu_t/\nu_{t,PINN}$ where $\nu_t = k/\omega$. Hence, the turbulent Prandtl number, $\sigma_{k,PINN}$, is determined using PINN, followed by the use of DNS data for estimating $C_{k,PINN}$ and $C_{\omega 2,PINN}$ which appear in the destruction terms in the k and ω equation, respectively. Neural networks are then used to generalise these results and thus construct a turbulence model. All Python PINN, NN and pySR scripts as well as the Python CFD code can be downloaded [Davidson. Using physical informed neural network (PINN) and neural network (NN) to improve a $k-\omega$ turbulence model: python CFD code and PINN script. In: Division of fluid dynamics. Gothenburg: Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology; 2025].

ARTICLE HISTORY
Received 17 November 2025
Accepted 21 April 2026

KEYWORDS
Machine learning; turbulence model; PINN; NN; symbolic regression;

1. Introduction

Eddy-viscosity RANS (Reynolds-Averaged Navier–Stokes) turbulence models 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 $\omega = \varepsilon/(\beta^* k)$ (β^* is a constant), respectively. Both turbulence models include five tuning constants. These constants can be improved using PINN.

In [1] they use PINN to optimise the five constants α, β^*, β, σ_k and σ_ω in the $k-\omega$ model. The constant β^* appears in the dissipation term in the k equation. σ_k and σ_ω are included in the turbulent diffusion term in the k and ω equation, respectively. The constants α and β are found in the production and destruction term, respectively, in the ω equation. 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 $\bar{v}_1$, $\bar{v}_2$, 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.

CONTACT Lars Davidson lada@chalmers.se

© 2026 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group.
This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. The terms on which this article has been published allow the posting of the Accepted Manuscript in a repository by the author(s) or with their consent.

Luo et al. [2] improve the five constants C_μ, $C_{\varepsilon 1}$, $C_{\varepsilon 2}$, σ_k and σ_ε in the $k-\varepsilon$ model using PINN. The turbulent viscosity is linearly dependent on C_μ. The constants $C_{\varepsilon 1}$ and $C_{\varepsilon 2}$ are part of the production and dissipation, respectively, in the ε equation. σ_k and σ_ε appear in turbulent diffusion terms in the k and ε equation, respectively. The authors 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. Superscript $n=1$ for the k equation and $n=2$ for the ε equation. 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 same flow that was used when training the PINN) using the new PINN-optimised $k-\varepsilon$ constants. Somewhat better results are obtained compared with the standard $k-\varepsilon$ constants.

Thakur et al. [3] use PINN to predict a diffusion coefficient. They study an unsteady, two dimensional convection-diffusion concentration equation, $c = c(x, y, t)$, with a spatially dependent diffusion coefficient, $D = D(x, y)$. The object is to predict D. The loss function, L, is

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

where c_i denotes true data predicted with CFD using a prescribed (true), varying diffusion coefficient (D_{true} – which is the prescribed D in the CFD simulation – varies linearly, sin, *tanh* etc) and c^p is obtained from the explicit unsteady diffusion equation; in the unsteady term, $\tilde{c}$ is used as the old time value of c^p. 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 (channel flow, flat-plate boundary layer flow, jet flow, etc) the turbulent shear stress, $\overline{u'v'}$, 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. Figure 1 presents a prediction using the Wilcox $k-\omega$ turbulence model [4] of fully developed channel flow at $Re_\tau \equiv u_\tau\delta/\nu = 5200$ 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. The predicted terms in the k equation using the $k-\omega$ model are compared with DNS in Figure 1(c). 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. Regarding the sum of the viscous diffusion term and the dissipation term, it may be noted that there is a slight spatial offset.

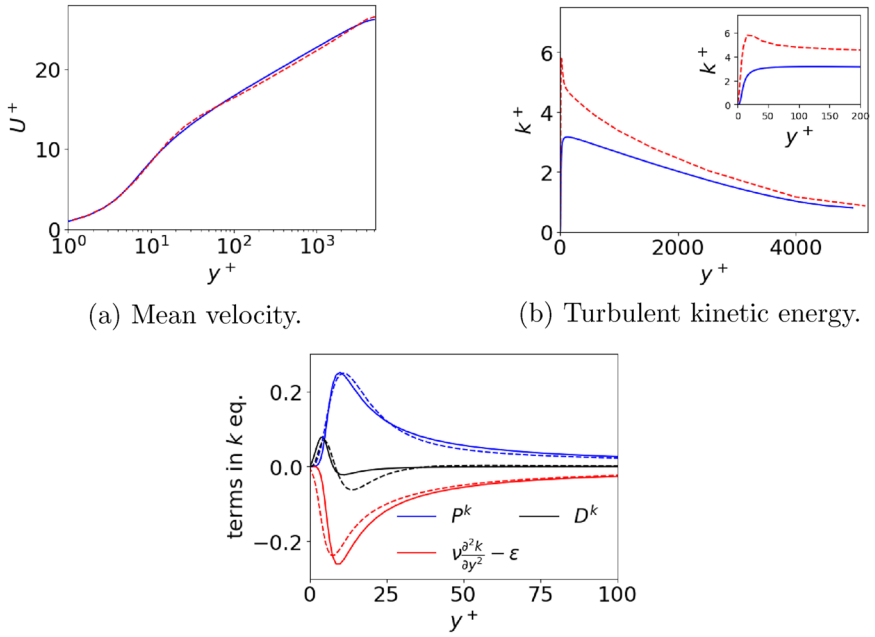
In order to improve the predicted k, we will use Physics Informed Neural Network, i.e. Physics Informed NN – usually called PINN – to improve the predicted diffusion term. In [1, 2] summarised above, new constant values of turbulent coefficients were optimised. In the present study, one turbulent coefficient, σ_k, is first turned into a function of y/δ, i.e. $\sigma_{k,PINN}(y/\delta)$. This coefficient appears implicitly in the diffusion term in the k equation which in fully-developed channel flow reads

$$\frac{\mathrm{d}}{\mathrm{d}y}\left[\left(\nu + \nu_{t,PINN}\right)\frac{\mathrm{d}k}{\mathrm{d}y}\right] + P^k - \varepsilon = 0 \tag{1}$$

where ν denotes the physical kinematic viscosity,

$$\sigma_{k,PINN} = \frac{\nu_t}{\nu_{t,PINN}} \tag{2}$$

and $\nu_t = k/\omega$. The two last terms are the production and dissipation term, respectively. The $\sigma_{k,PINN}$ function is obtained as follows. By taking the production, the dissipation terms as well as the turbulent kinetic energy from DNS channel flow data, Equation (1) is turned into an ordinary differential equation for the turbulent viscosity, $\nu_{t,PINN}$, which is solved using Physics Informed Neural Network (PINN).



(a) Mean velocity.

(b) Turbulent kinetic energy.

(c) Terms in the k equation. D^k is turbulent diffusion; for the $k-\omega$ simulation $D^k = \frac{\partial}{\partial y}\left(\nu_t \frac{\partial k}{\partial y}\right)$.

Figure 1. Fully-developed channel flow. $Re_\tau = 5200$. Solid lines: $k-\omega$; dashed lines:DNS [5]. (a) Mean velocity. (b) Turbulent kinetic energy and (c) Terms in the k equation. D^k is turbulent diffusion; for the $k-\omega$ simulation $D^k = \frac{\partial}{\partial y}(\nu_t \frac{\partial k}{\partial y})$.

Recall that the turbulent viscosity is well predicted by the standard Wilcox $k-\omega$ turbulence model. If we modify the turbulent kinetic energy, the turbulent viscosity will also be modified and the predicted mean flow would then deteriorate. Hence, we must also modify the predicted ω so that the new $k-\omega$ model predicts the same turbulent viscosity, $\nu_t = k/\omega$, as the Wilcox $k-\omega$ model. In order to not change the predicted turbulent viscosity a new function, $C_{k,NN}$, is added to the dissipation term in the k equation and another, $C_{\omega 2,NN}$, is added to the destruction term in the ω equation. Finally, three NN models are created for $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ which are made functions of two input parameters, τ_{tot}/u_τ^2 and $\nu_t/(yu_\tau)$. The total stress, τ_{tot}, reads

$$\tau_{tot} = 2\left(\nu + \nu_t\right)\left(\bar{s}_{ij}\bar{s}_{ij}\right)^{1/2}, \quad \bar{s}_{ij} = \frac{1}{2}\left(\frac{\partial \bar{v}_i}{\partial x_j} + \frac{\partial \bar{v}_j}{\partial x_i}\right). \tag{3}$$

The difference between $\sigma_{k,PINN}$ and $\sigma_{k,NN}$ is that the former is obtained from PINN and the latter is given by the NN model using $\sigma_{k,PINN}$ as the target.

The paper is organised as follows. In the two following sections, the two-dimensional solver (channel flows and flat-plate boundary layer) and three-dimensional solver (periodic hill flow) are briefly described. Then the PINN and NN models are presented. Next, the results are discussed and presented and in the final section we give some concluding remarks.

2. Equations and the numerical method for the 2D RANS simulations

The steady RANS equations for the fully-developed channel flow and the flat-plate boundary layer read

$$\frac{\partial \bar{v}_i}{\partial x_i} = 0 \tag{4}$$

$$\frac{\partial\left(\bar{v}_i\bar{v}_j\right)}{\partial x_j} = \delta_{1i} - \frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left[\left(\nu + \nu_t\right)\frac{\partial \bar{v}_i}{\partial x_j}\right] \tag{5}$$

The first term on the right-hand side of Equation (5) is the driving pressure gradient used in fully-developed channel flow; it is not present in the flat-plate boundary layer. The $k-\omega$ turbulence model reads

$$\begin{aligned}\frac{\partial\left(\bar{\nu}_j k\right)}{\partial x_j} &= P^k + \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_{k,NN}}\right)\frac{\partial k}{\partial x_j}\right] - C_\mu C_{k,NN} k\omega \\ \frac{\partial\left(\bar{\nu}_j \omega\right)}{\partial x_j} &= C_{\omega_1}\frac{P^k}{\nu_t} + \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_{\omega,NN}}\right)\frac{\partial \omega}{\partial x_j}\right] - C_{\omega 2,NN}\omega^2 \\ P^k &= \nu_t\left(\frac{\partial \bar{\nu}_i}{\partial x_j} + \frac{\partial \bar{\nu}_j}{\partial x_i}\right)\frac{\partial \bar{\nu}_i}{\partial x_j}, \quad \nu_t = \frac{k}{\omega}\end{aligned} \tag{6}$$

At the wall-adjacent cells, ω is not solved but is set as

$$\omega_w = \frac{6\nu}{C_{\omega 2,NN} y^2} \tag{7}$$

where y is the wall distance between the wall-adjacent cell centre and the wall. In the standard $k-\omega$ model $C_{k,NN} = 1$, $\sigma_{k,NN} = \sigma_k$, $\sigma_{\omega,NN} = \sigma_\omega$ and $C_{\omega 2,NN} = C_{\omega 2}$. The coefficients have the values $C_{\omega 1} = 5/9$, $C_{\omega 2} = 3/40$, $\sigma_k = \sigma_\omega = 2$ and $C_\mu = 0.09$.

It turns out that the turbulent Prandtl number, $\sigma_{k,NN}$, is much smaller than σ_k. For that reason, also the turbulent Prandtl number of the ω equation, σ_ω, is modified as

$$\sigma_{\omega,NN} = \min\left(2\sigma_{k,NN}, \sigma_\omega\right) \tag{8}$$

The **pyCALC-RANS** code is used [6] for solving the discretised equations. It is an incompressible, finite volume code written in Python. It is fully vectorised (i.e. no for loops). The convective terms in the momentum equations are discretised using the MUSCL scheme [7] (a second-order bounded upwind scheme) and the hybrid 1st-order upwind/2nd-order central scheme is used for k and ω equations. The numerical procedure is based on the pressure-correction method, SIMPLEC, and a collocated grid arrangement using Rhie–Chow interpolation [8].

3. Equations and the numerical method for the hill-flow simulations

The unsteady momentum equations for the periodic hill flow read

$$\frac{\partial \bar{\nu}_i}{\partial t} + \frac{\partial\left(\bar{\nu}_j \bar{\nu}_i\right)}{\partial x_j} = \beta\delta_{1i} - \frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left[(\nu + \nu_t)\frac{\partial \bar{\nu}_i}{\partial x_j}\right] \tag{9}$$

where the term $\beta\delta_{1i}$ is the driving pressure gradient in the streamwise direction. The coefficient β is given in Equation (21).

The finite volume code **pyCALC-LES** [9] is used. It is written in Python and is fully vectorised (i.e. no `for` loops). The solution procedure is based on fractional step. The second-order MUSCL scheme [7] is used for the convective terms in the $\bar{\nu}_i$ equations. The k and ω equations are given in Equation (6) (adding the unsteady term $\partial/\partial t$) using the same discretisation.

The only reason why **pyCALC-RANS** is not used for this flow is that the author did not succeed in adjusting the β coefficient, in a reasonable manner, see Equation (21). The discretisation in space in **pyCALC-LES** is identical to that in **pyCALC-RANS**. The main difference is how the pressure-velocity coupling is treated; furthermore the latter code was developed for unsteady, three-dimensional flow whereas the former is used for steady, two-dimensional flow.

Above, we have used tensor notation for the velocity vector $(\bar{\nu}_1, \bar{\nu}_2, \bar{\nu}_3)$ and for the space vector (x_1, x_2, x_3). Below, we will replace them with $\bar{u}$, $\bar{v}$, $\bar{w}$ and x,y,z.

4. Development of the new $k-\omega$ model using PINN and NN

The implementation of PINN is briefly explained in Appendix 1. Now, let's involve our differential equation, the k equation (see Equation (6)). The equation for turbulent kinetic energy in fully-developed channel flow

is given in Equation (1) and it is re-written as

$$\left(\nu + \nu_{t,PINN}\right)\frac{\mathrm{d}^2k}{\mathrm{d}y^2} + \frac{\mathrm{d}k}{\mathrm{d}y}\frac{\mathrm{d}\nu_{t,PINN}}{\mathrm{d}y} + P^k - \varepsilon = Q \tag{10}$$

where we have added a right-hand side, Q (which is an error term that should be reduced to zero, see below). We want to find a new turbulent viscosity, $\nu_{t,PINN}$, that gives a turbulent diffusion that agrees with the DNS turbulent diffusion term in Figure 1(c). Hence, $\nu_{t,PINN}$ is the unknown variable in Equation (10) and k, P^k and ε are taken from DNS. Equation (10) is solved in a half-width channel at $Re_\tau = 5200$. The boundary condition at the wall ($y = 0$) is $\nu_{t,PINN} = 0$ and at the centre ($y = \delta$) it is $\nu_{t,PINN} = \nu_{t,DNS}$ where $\nu_{t,DNS} = |\frac{\overline{u'v'}}{\partial\overline{v}/\partial y}|_{DNS}$.

The turbulent viscosity, $\nu_{t,PINN}$ in Equation (10), will be predicted by PINN while minimising the error Q^2. The loss function, `loss_fn`, in Listing 2 in the Appendix is replaced with Equation (10) and the Python code is given in Listing 3. `nut` and `k` in Listing 3 are $\nu_{t,PINN}$ and k in Equation (10), respectively. `k_y`, for example, is dk/dy. Note that `k_y` and `k_yy` are known (taken from DNS). There are two losses in Listing 3, one, `ODE_loss`, which relates to the ordinary differential equation (ODE) and one, `BC_loss`, which relates to the boundary conditions (BC). In this way the PINN is forced to satisfy both the ODE and the boundary conditions. It should be mentioned that we had serious problems in making the PINN converge. The first problem was that the PINN is sensitive to the initialisation of the weights and biases. Two subsequent runs gave very different convergence histories. The remedy was to save the initial weights and biases of a successful run and then load and use them in subsequent runs. The second remedy was to increase the number of neurons from 10 to 30, switch from the `tanh` actuator to the `sigmoid` actuator and increase the number of hidden layers from 3 to 5, see `class MyNet` in Listing 3. The Adam optimiser was used with an initial learning rate of 0.01 which was reduced by 0.7 (`gamma = 0.7`) using the `MultiStepLR` scheduler at `milestones` equal to `3e4, 6e4, 8e4, 1e5, 1.3e5, 1.5e5, 1.8e5`. This scheduler was found to be more suitable than the `ReduceLROnPlateau` since it was difficult to find a good `patience` parameter (i.e. how long a plateau should be defined before the learning rate should be reduced). The loss, defined as $\sum(Y - y_{pred})^2$ is during 200, 000 epochs reduced from $1.4 \cdot 10^{13}$ to 4. Recall that all Python scripts can be downloaded [10].

In the NN models, contrary to the PINN solver, the `relu` actuator function was found to be superior to the `sigmoid` actuator function.

Figure 2(a–c) present the predicted turbulent viscosity in the k equation, $\nu_{t,PINN}$, and the turbulent Prandtl number, $\sigma_{k,PINN}$, where $\sigma_{k,PINN} = \nu_t/\nu_{t,PINN}$. It is seen that the PINN turbulent viscosity for $y^+ \lesssim 15$ is slightly larger than that predicted by the $k-\omega$ model. But it should be noted that the object of PINN is not that $\nu_{t,PINN}$ should agree with that of the $k-\omega$ model or DNS but to predict a turbulent diffusion that agrees with DNS (see Figure 2(d)). The DNS turbulence diffusion includes the contribution from both the triple correlation and the pressure-velocity correlation; the latter term was (by mistake) omitted in [11, Figure 3c]. The DNS and PINN lines in Figure 2(d) are on top of each other. In the inset in Figure 2(b) a small non-physical oscillation can be seen. It occurs at the point where dk_{DNS}/dy changes sign.

It should be noted that $\nu_{t,PINN}$ in Figure 2(b) is for $y^+ \gtrsim 30$ very different to that shown in [11, Figure 3a]. The reason is that Equation (10) was poorly converged for $y^+ \gtrsim 30$ [11]. However, the predicted turbulent diffusion in [11, Figure 3c] is very similar to that in Figure 2(d).

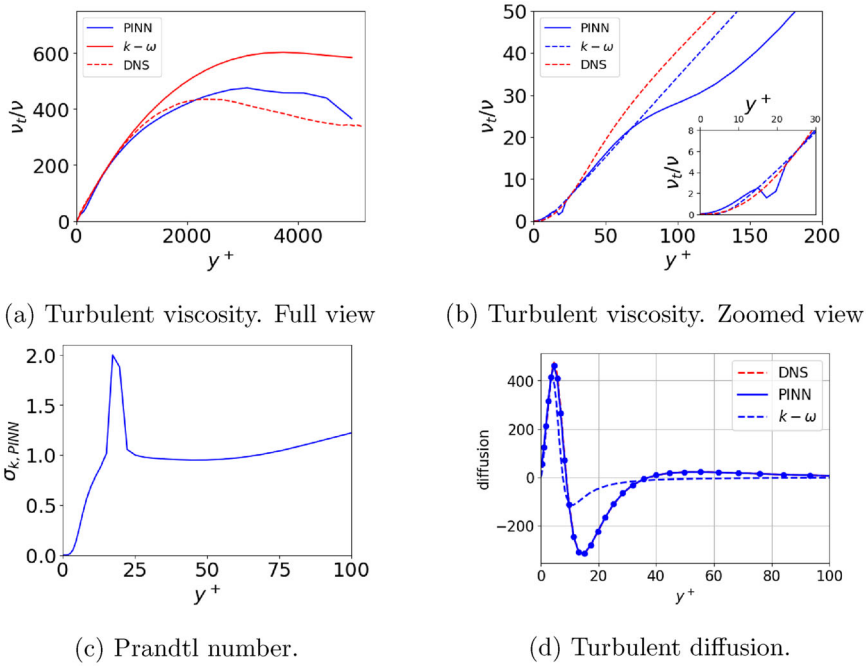
Let's verify that a CFD solver does predict a correct turbulent kinetic energy using the PINN turbulent Prandtl number (see Figure 2(c)). In fully-developed channel flow, the k equation (see Equation (6)) reads

$$\frac{\mathrm{d}}{\mathrm{d}y}\left(\left(\nu + \frac{\nu_t}{\sigma_{k,PINN}}\right)\frac{\mathrm{d}k}{\mathrm{d}y}\right) + P^k - \varepsilon = 0 \tag{11}$$

where P^k and ε are again taken from DNS. The turbulent viscosity, ν_t, is computed as $\nu_t = k/\omega_{DNS}$ where ω_{DNS} is taken from Equation (13). The reason is that the turbulence viscosity, ν_t, must be the same as that in the $k-\omega$ model (and DNS), see Equation (13).

We solve Equation (11) using the CFD solver **pyCALC-RANS**. Figure 3 presents the predicted k profiles. We find that the agreement is excellent.

Until 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 will solve both the k and ω equations computing the source terms as in Equation (6). Recall that P^k in the Wilcox $k-\omega$



(a) Turbulent viscosity. Full view

(b) Turbulent viscosity. Zoomed view

(c) Prandtl number.

(d) Turbulent diffusion.

Figure 2. Prediction by PINN compared to DNS and the Wilcox $k-\omega$ model. $Re_\tau = 5200$. (a) Turbulent viscosity. Full view. (b) Turbulent viscosity. Zoomed view. (c) Prandtl number and (d) Turbulent diffusion.

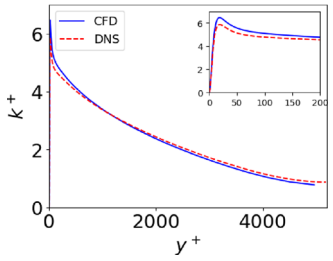


Figure 3. CFD prediction of Equation (11) using $\sigma_{k,PINN}$.

model is correct since the model does predict the velocity profile correctly, see Figure 1, and hence it also predicts the turbulent viscosity correctly. Recall that the total shear stress is predicted as

$$y/\delta - 1 \tag{12}$$

by any turbulence models in a finite volume method since this is the exact solution when integrating the momentum equation [12, Section 6.2]. The k predicted by DNS near the wall is much larger than that predicted by the Wilcox $k-\omega$ model, see Figure 1(b). This means that ω must be modified in order to give the same turbulent viscosity as the Wilcox $k-\omega$ model, i.e.

$$\nu_{t,k-\omega} = \frac{k_{k-\omega}}{\omega_{k-\omega}} = \nu_{t,DNS} = \frac{k_{DNS}}{\omega_{DNS}} \tag{13}$$

which gives

$$\omega_{DNS} = k_{DNS}/\nu_{t,k-\omega}. \tag{14}$$

Now we have an ω_{DNS} that gives the same turbulent viscosity as DNS. Next, the dissipation term $C_\mu k\omega$ in the k equation in Equation (6) 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, $C_{k,PINN} = C_{k,PINN}(y/\delta)$, so that the k equation in

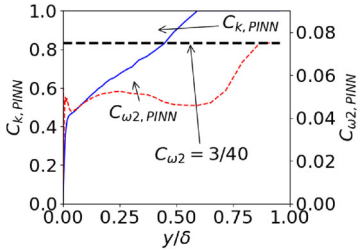


Figure 4. $C_{k,PINN}$ and $C_{\omega2,PINN}$.

Equation (6) is satisfied with $k = k_{DNS}$ and $\omega = \omega_{DNS}$, i.e. (see Equation (11))

$$C_{k,PINN} = \frac{D^k_{DNS} + P^k_{DNS}}{C_\mu k_{DNS} \omega_{DNS}} \tag{15}$$

Note that the viscous diffusion has been omitted in order to prevent a large gradient of $C_{k,PINN}$ close to the wall.

Finally, we must make sure that the ω equation in Equation (6) predicts $\omega = \omega_{DNS}$. This is achieved by making $C_{\omega2,PINN} = C_{\omega2,PINN}(y/\delta)$. From the ω equation in Equation (6) we get

$$C_{\omega2,PINN} = \frac{\frac{\mathrm{d}}{\mathrm{d}y}\left(\frac{\nu_t}{\sigma_\omega}\frac{\mathrm{d}\omega_{DNS}}{\mathrm{d}y}\right) + C_{\omega1}\frac{P^k_{DNS}}{\nu_{t,DNS}}}{\omega^2_{DNS}} \tag{16}$$

Again, the viscous diffusion has been omitted in order to prevent a large gradient of $C_{\omega2,PINN}$ close to the wall.

Figure 4 shows the two damping functions computed with Equations (15) and (16). The thick, black, dashed line is the standard value of $C_{\omega,2} = 3/40$, see below Equation (6). The damping function, $C_{k,PINN}$, is in the outer region much affected by the dominator in Equation (15) which essentially is $(k_{DNS}/k_{k-\omega})^2$: first, k_{DNS} is larger than $k_{k-\omega}$ (see Figure 3), and, second, ω_{DNS} is larger than $\omega_{k-\omega}$, see Equation (13). This explains why $C_{k,PINN}$ is much smaller than one for $y/\delta \lesssim 0.5$.

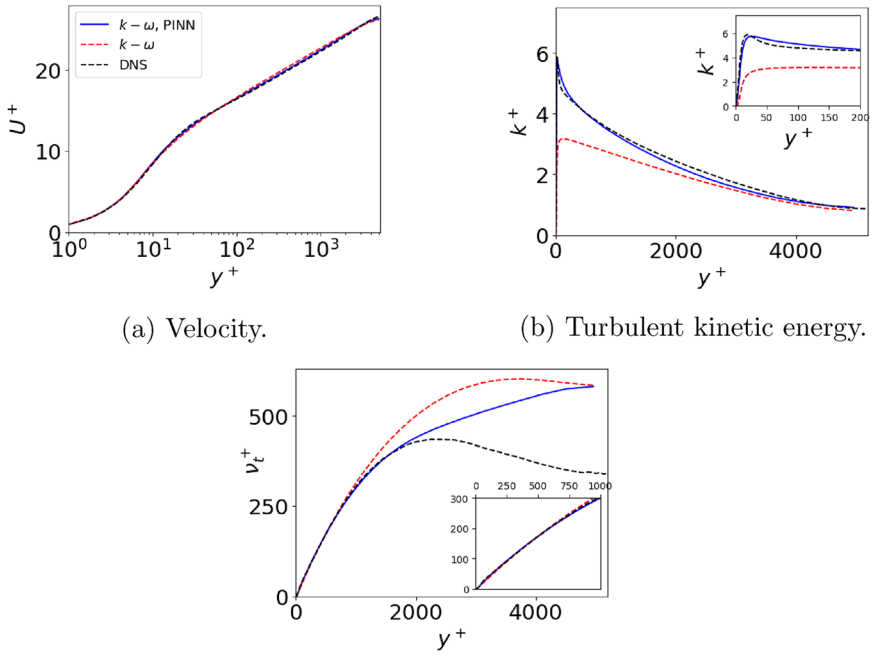
Finally, we insert our expression of $\sigma_{k,PINN}$ (obtained by PINN, Equation (10) and Equation (2)), $C_{k,PINN}$ Equation (15) and $C_{\omega2,PINN}$ Equation (16) into Equation (6) and solve the full equation system ($\bar{\nu}_1$, k and ω) using **pyCALC-RANS**. The predictions are compared with DNS in Figure 5 and we find that the agreement is very good.

4.1. Compute $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega2,NN}$ using neural network (NN)

It was shown [11] that the $k-\omega$ model, using $\sigma_{k,PINN}$, $C_{k,PINN}$ and $C_{\omega2,PINN}$, accurately predicts fully-developed channel flow at $Re_\tau = 2000$, $Re_\tau = 5200$, $Re_\tau = 10,000$ as well as a flat-plate boundary layer. But the drawback is that this model is not applicable to re-circulating flow since the three coefficients are expressed in y/δ. In this section, we will use NN models to predict $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega2,NN}$. Many different input parameters to the NN models were investigated such as P_k/ε, P_k^+, ε^+, $\nu_t/(yu_\tau)$, etc, It is important that they are properly non-dimensionalised so that they can be used in other flows as well at other Reynolds numbers. In the end, a good combination was found: the input parameters to the NN models for $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega2,NN}$ were taken as

$$\frac{\tau_{tot}}{u^2_\tau} \quad \text{and} \quad \frac{\nu_t}{yu_\tau} \tag{17}$$

where y is the distance to the nearest wall and τ_{tot} is given in Equation (3). Using these input parameters the NN models will be applicable to complex, recirculating flow. In complex geometries with multiple bodies, it may be costly to find the wall distance to the closest wall. An efficient to find the wall distance is to solve a Poisson equation [14]. The input parameters, τ_{tot}/u^2_τ and $\nu_t/(yu_\tau)$, are in the training process taken from the $k-\omega$-PINN predictions in Figure 5. The targets/outputs are $\sigma_{k,PINN}$ Equation (2), $C_{k,PINN}$ Equation (15) and



(a) Velocity.

(b) Turbulent kinetic energy.

(c) Turbulent viscosity.

Figure 5. Fully-developed channel flow comparing $k-\omega$ PINN and standard $k-\omega$ model. $\sigma_{k,PINN}$ (Eqs. 10 and 2), $C_{k,PINN}$ Equation (15) and $C_{\omega 2,PINN}$ Equation (16) are used in Equation (6). $Re_\tau = 5200$. (a) Velocity. (b) Turbulent kinetic energy and (c) Turbulent viscosity.

$C_{\omega 2,PINN}$ Equation (16). It should be mentioned that the non-physical oscillation in Figure 2(b) was smoothed when $\sigma_{k,PINN}$ was used as target in the NN model for $\sigma_{k,NN}$ below.

Minimum and maximum of the input parameters, τ_{tot}/u_τ^2 and $\nu_t/(yu_\tau)$, as well as the output parameters, $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$, are in the training process stored on disk. They are then used in the NN model in the CFD code to set limits on the CFD input parameters, see lines 42-49 in Listing 4 in the Appendix, and the output parameters, see lines 65-67. The difference between $\sigma_{k,PINN}$ and $\sigma_{k,NN}$ is that the former is obtained from Equation (2) where $\nu_{t,PINN}$ is given by solving an ODE Equation (11) using PINN; the latter is predicted by the NN model using $\sigma_{k,PINN}$ as the target. The differences between $C_{k,PINN}/C_{k,NN}$ and $C_{\omega 2,PINN}/C_{\omega 2,NN}$ are essentially the same: $C_{k,PINN}$ and $C_{\omega 2,PINN}$ are given by Eqs. 15 and 16 whereas $C_{k,NN}$ and $C_{\omega 2,NN}$ are predicted by the NN models using $C_{k,PINN}$ and $C_{\omega 2,PINN}$ as targets.

All Python PINN and NN scripts and the Python CFD codes can be downloaded [10].

5. Results

NN models for the turbulent Prandtl number, $\sigma_{k,NN}$, the coefficients $C_{k,NN}$ and $C_{\omega 2,NN}$ were developed in the previous section. They are in this section used in Equation (6) when solving the k and ω equations. The new model is called the $k-\omega$-PINN-NN model.

The NN models are incorporated in the CFD code **pyCALC-RANS** as follows (the procedure is identical in **pyCALC-LES**).

(1) At the first iteration, load the NN model into **pyCALC-RANS**
(2) Solve the $\bar{\nu}_1$ and $\bar{\nu}_2$ equations Equation (5) and the continuity equation (Equation (4), i.e. the pressure correction equation).
(3) Call the NN models to compute $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$
(4) Solve k and ω equations.
(5) Compute $\nu_t = k/\omega$
(6) End of global iteration. Repeat from Step 2 until convergence (1000s of iterations)

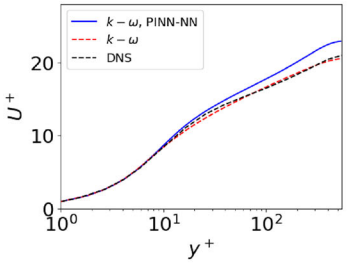
Table 1. Channel flow. Reynolds number (Re_τ), number of cells in the wall-normal direction (N_y), grid stretching (s_y) and location of wall-adjacent cell centre (y^+).

Re_τ	N_y	s_y	y^+
550	60	1.07	0.3
2000	60	1.11	0.2
5200	70	1.1	0.3
10, 000	150	1.05	0.2

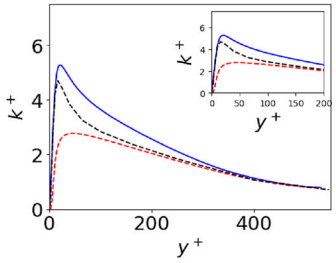
The new model is validated in three types of flows: fully-developed channel flow ($Re_\tau = 550$, 2000, 5200 and $Re_\tau = 10,000$), flat-plate boundary layer flow and flow over a hill. The boundary conditions at the walls are $\bar{u} = \bar{v} = k = 0$; ω is at the centre of the wall-adjacent cells fixed according to Equation (7).

5.1. Fully-developed channel flow

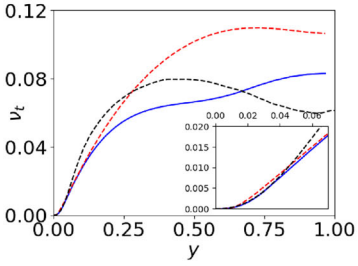
Fully-developed, i.e. one-dimensional, channel flows are simulated using **pyCALC-RANS**. The $\bar{u}$, k and ω equations are solved. The domain covers only the lower half of the channel. The wall-normal gradient is set to zero for all variables at the upper boundary. The flow is driven by a prescribed pressure gradient (the first term on the right-hand side of Equation (5)). Details of the grids are provided in Table 1. All quantities are



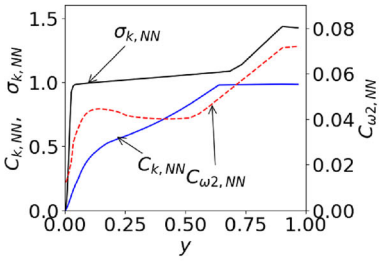
(a) Velocity.



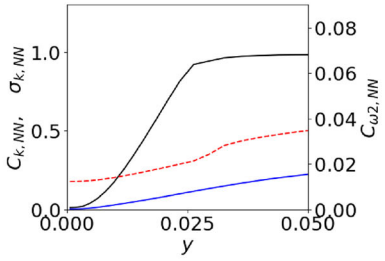
(b) Turbulent kinetic energy.



(c) Turbulent viscosity.

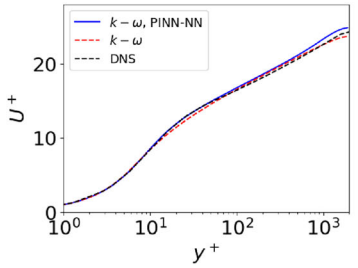


(d) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model.

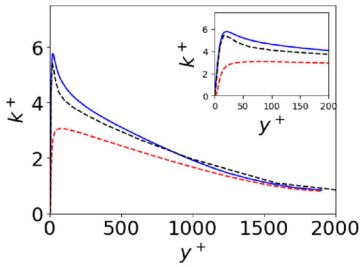


(e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model. Zoomed-in view.

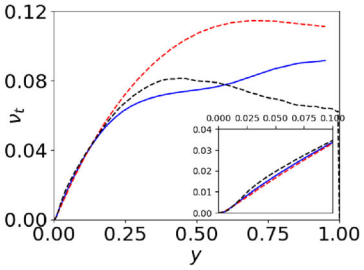
Figure 6. Fully-developed channel flow. $Re_\tau = 550$. (a) Velocity. (b) Turbulent kinetic energy. (c) Turbulent viscosity. (d) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model and (e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model. Zoomed-in view.



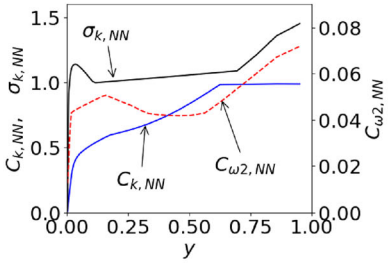
(a) Velocity.



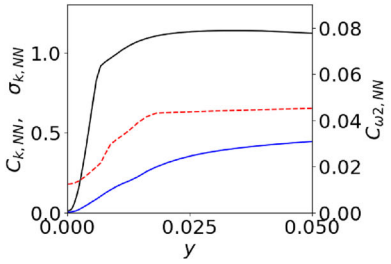
(b) Turbulent kinetic energy.



(c) Turbulent viscosity.



(d) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model.



(e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model. Zoomed-in view.

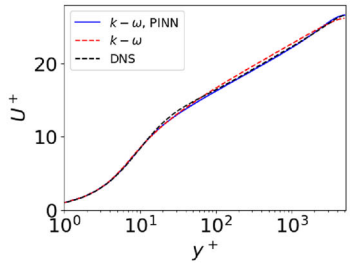
Figure 7. Fully-developed channel flow. $Re_\tau = 2000$. (a) Velocity. (b) Turbulent kinetic energy. (c) Turbulent viscosity. (d) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model and (e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model. Zoomed-in view.

made non-dimensional with half-channel width, δ, and friction velocity, u_τ. Quantities with superscript $+$ are made non-dimensional with kinematic viscosity, ν, and u_τ.

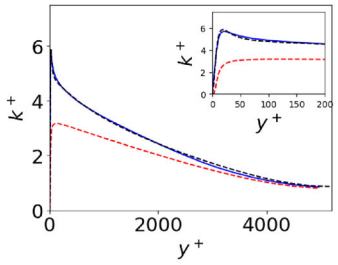
The predicted velocity, turbulent kinetic energy and turbulent viscosity using the Wilcox $k-\omega$ model and the $k-\omega$-PINN-NN model are compared with DNS in Figures 6, 7, 8 and 9. The velocity profiles are well predicted with both models except at the lowest Reynolds number, Figure 6(a), for which the $k-\omega$-PINN-NN model gives slightly too large values. The turbulent kinetic energy is much better predicted with the $k-\omega$-PINN-NN model. It can be seen that the turbulent viscosities predicted by the two turbulence models are similar for all Reynolds numbers. It may be recalled that this was indeed a requirement when developing the $k-\omega$-PINN-NN model in the previous section, see Equation (13). They are *very* similar in the inner part of the boundary layer ($y \lesssim 0.05$, $y \lesssim 0.1$, $y \lesssim 0.18$ and $y \lesssim 0.29$ for $Re_\tau = 550$, 2000, 5200 and $Re_\tau = 10,000$, respectively).

Figures 6(d), 7(d), 8(d) and 9(d) present $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ predicted by the NN model. They are very similar for all Reynolds numbers. The reason is that they were made functions of the input parameters τ_{tot}/u_τ^2 and $\nu_t/(yu_\tau)$; the former is independent of Reynolds number (see Equation (12)) and the latter is only weekly dependent on Reynolds number (see Figures 6(c), 7(c), 8(c), 9(c)). The reason for the over-predicted velocity profile in Figure 6(a) is probably that the input parameters should be modified at this low Reynolds number.

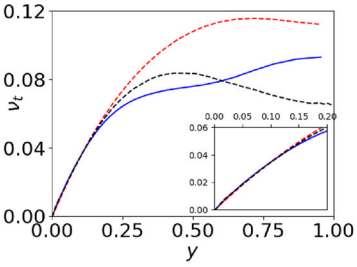
From the zoomed-in views in Figures 6(e), 7(e), 8(e) and 9(e) it is clearly seen that the extent of the near-wall region in which $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ diminishes as the Reynolds number increases. This mimics the effect of Reynolds number on the input parameter $\nu_t/(u_\tau y)$, see Figure 10.



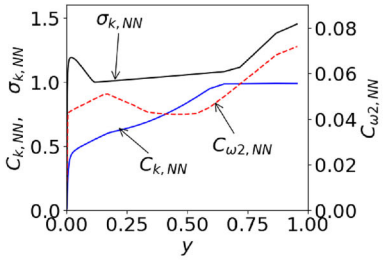
(a) Velocity.



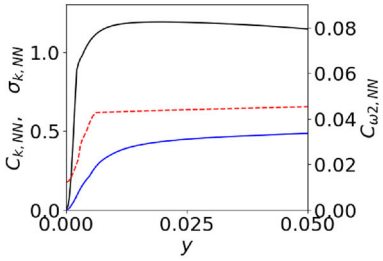
(b) Turbulent kinetic energy.



(c) Turbulent viscosity.



(d) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model.



(e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model. Zoomed-in view.

Figure 8. Fully-developed channel flow. $Re_\tau = 5200$. (a) Velocity. (b) Turbulent kinetic energy. (c) Turbulent viscosity. (d) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model and (e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model. Zoomed-in view.

It should be mentioned that $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ exhibit slow oscillations with respect to iteration number. Figure 11 shows how $\sigma_{k,NN}$ oscillates with respect to iteration number; the oscillation period is approximately 3000 iterations. This makes it difficult to converge the ω equation properly. The remedy is to introduce an exponentially weighted averaging operator acting on $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ over m iterations. The average of $\sigma_{k,NN}$, for example, is defined as [16, 17]

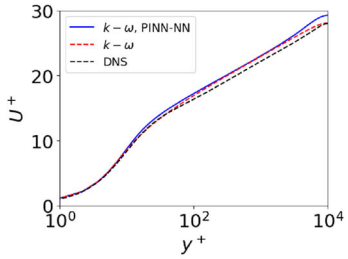
$$\langle \sigma_{k,NN} \rangle_T^n = a \langle \sigma_{k,NN} \rangle_T^{n-1} + (1-a)\sigma_{k,NN}^n, \quad a = \exp(-1/m) \tag{18}$$

$\langle \cdot \rangle_T$ indicates a time average over time, T, i.e.

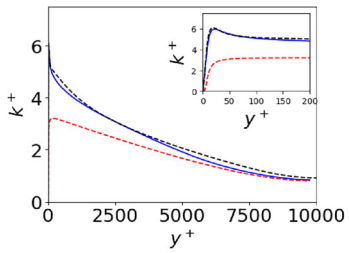
$$\langle \phi(t) \rangle_T = \frac{1}{T} \int_{-\infty}^{t} \phi(\tau) \exp(-(t-\tau)/T) d\tau \Rightarrow$$
$$\langle \phi \rangle_T^n \equiv \langle \phi \rangle_T = a \langle \phi \rangle_T^{n-1} + (1-a)\phi^n,$$

where $a = 1/(1 + \Delta t/T)$ and n denotes the timestep number.

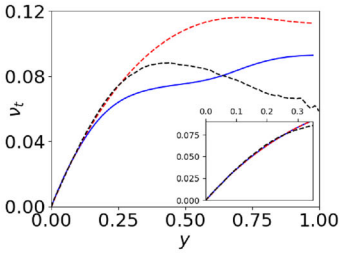
Taking guidance from Figure 11 we set $m = 3000$. For the other three Reynolds numbers $m = 500$. The problem of slow variations of $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ does not occur in the flat-plat boundary layer or the hill flow. This issue is probably related to the strong elliptic character of fully-developed channel flow, i.e. the transport takes place only through diffusion.



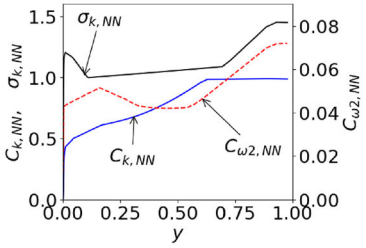
(a) Velocity.



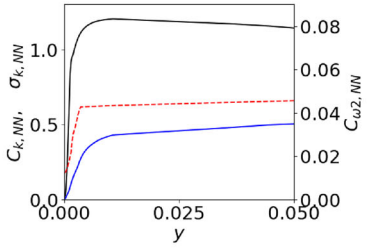
(b) Turbulent kinetic energy.



(c) Turbulent viscosity.



(d) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model.



(e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model. Zoomed-in view.

Figure 9. Fully-developed channel flow. $Re_\tau = 10,000$. (a) Velocity. (b) Turbulent kinetic energy. (c) Turbulent viscosity. (d) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model and (e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model. Zoomed-in view.

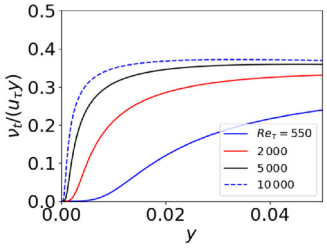


Figure 10. Fully-developed channel flow. Near-wall behaviour of the input paramenter $\nu_t/(u_\tau y)$.

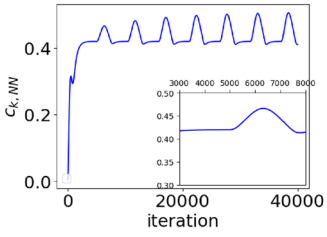
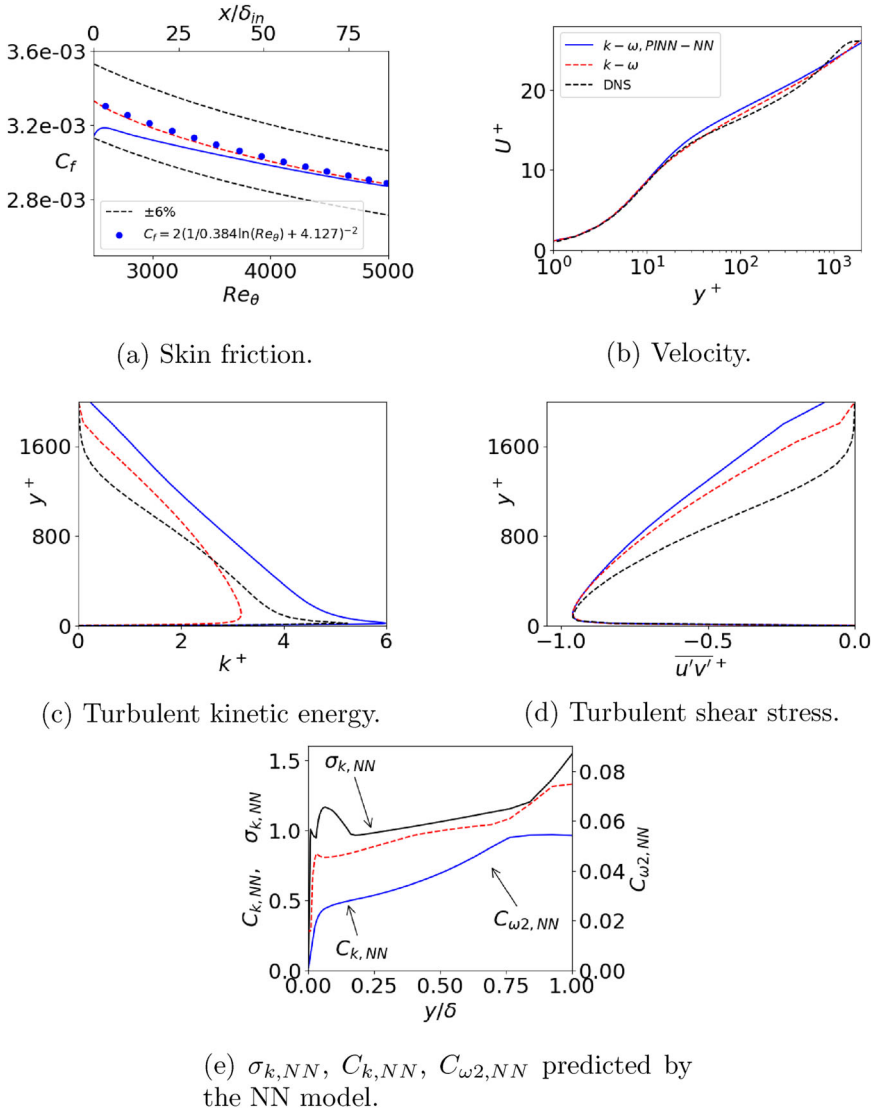


Figure 11. Fully-developed channel flow. $Re_\tau = 10,000$. $C_{k,NN}$ vs. iteration at $y^+ = 100$. $C_{k,NN}$ has been filtered over 500 iterations.



(a) Skin friction. (b) Velocity.

(c) Turbulent kinetic energy. (d) Turbulent shear stress.

(e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model.

Figure 12. Flat-plate boundary layer. Profiles at $Re_\theta = 4500$. DNS [13]. (a) Skin friction. (b) Velocity. (c) Turbulent kinetic energy. (d) Turbulent shear stress and (e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model.

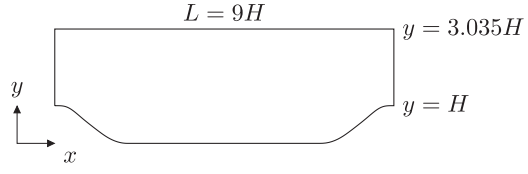
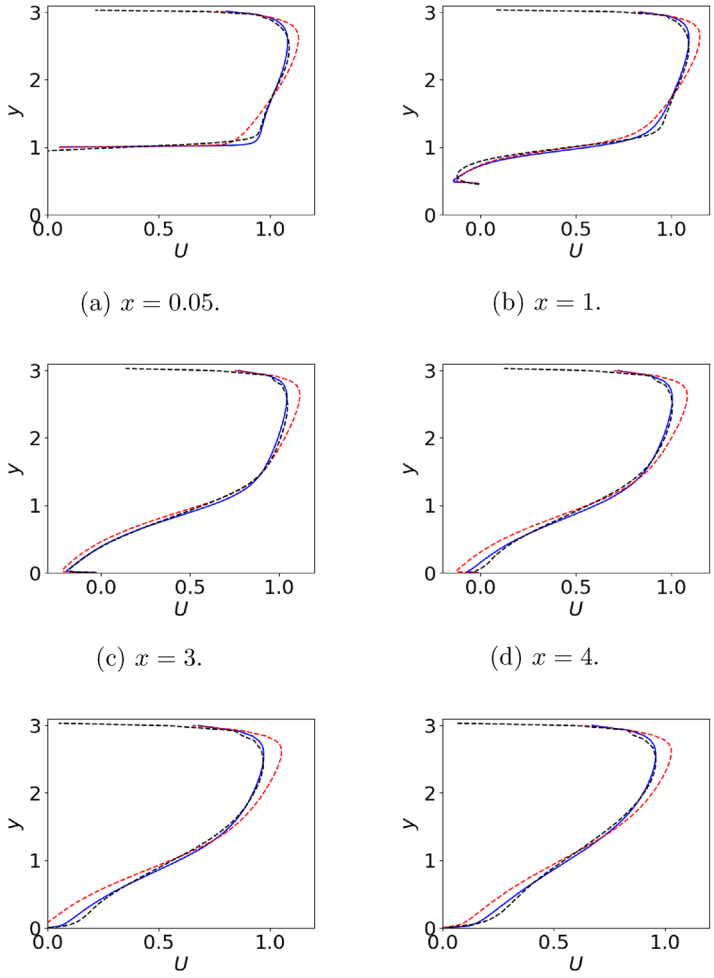


Figure 13. The geometry of the hill.

5.2. Flat-plat boundary layer flow

This flow is also simulated using **pyCALC-RANS**. The Reynolds number at the inlet is $Re_\theta = 2550$. The mean profiles are taken from a pre-cursor 2D RANS simulation. The grid has 150 × 90 cells (x, y). The domain size is $92\delta_{in} \times 20\delta_{in}$ where δ_{in} denotes the boundary-layer thickness at the inlet. The grid is stretched in the wall-normal direction by 10% up to $y = 6.7\delta_{in}$ where the cell size is $\Delta_y = 0.6\delta_{in}$; it is stretched by 1% in the streamwise direction.

Figure 12 presents comparisons of predictions by the $k-\omega$-PINN-NN model and the Wilcox $k-\omega$ with DNS. It can be seen that the skin friction and the velocity profile are very good predicted by both turbulence



(a) $x = 0.05$. (b) $x = 1$.

(c) $x = 3$. (d) $x = 4$.

(e) $x = 5$. (f) $x = 7$.

Figure 14. Hill flow. Velocity. ——: $k - \omega$-PINN-NN; – –: standard $k - \omega$; ▬ ▬: DNS [15]. (a) $x = 0.05$. (b) $x = 1$. (c) $x = 3$. (d) $x = 4$. (e) $x = 5$ and (f) $x = 7$.

models. The peak of the shear stress is also well predicted by both models but the magnitude of the shear stress is slightly too small in the outer region. However, the predicted peak of the turbulent kinetic energy by the $k - \omega$-PINN-NN model is too large but its form is much better than that predicted with the Wilcox $k - \omega$ model. The predicted $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ (see Figure 12(e)) are similar to those for the channel flows.

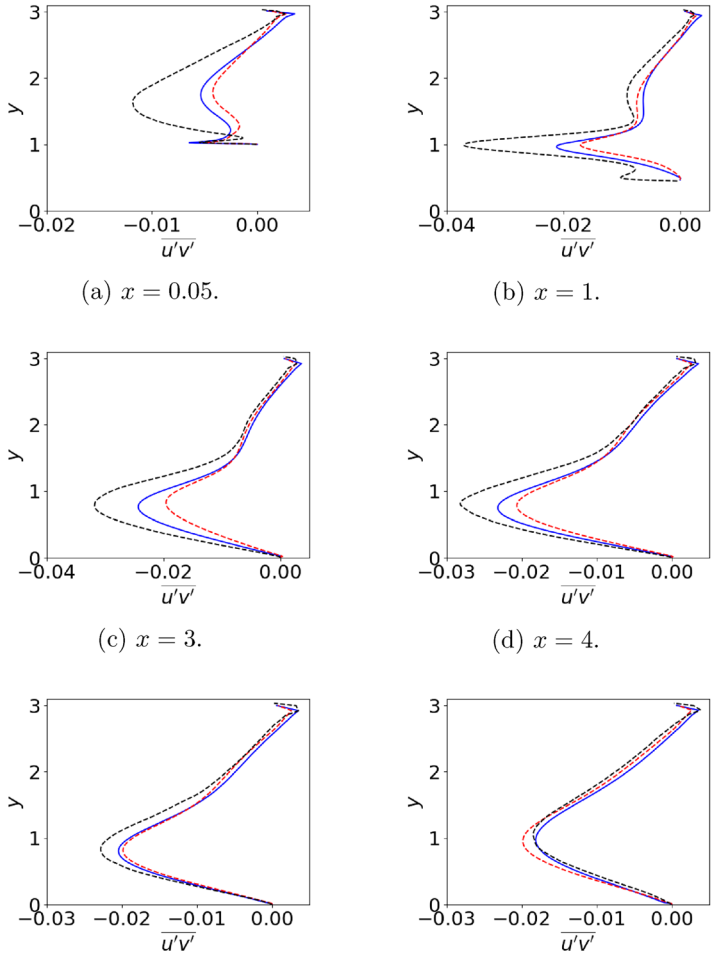
5.3. Hill flow

The final test case is the flow over a two-dimensional periodic hill, see Figure 13. This flow is simulated using **pyCALC-LES**. The size of the domain is $9H \times 3.035H \times H$ in the streamwise (x) and the wall-normal (y) direction (z), respectively. The Reynolds number is $Re = 10{,}600$ based on the hill height and the bulk velocity U_b at the top of the hill. Periodic boundary conditions are used in the x direction. Slip conditions are prescribed in the z direction.

Unsteady simulations are carried out marching to steady-state conditions. The time step is set to 0.01 which gives a maximum CFL of 0.4. Two global iterations (solving $\bar{u}$, $\bar{v}$, $\bar{w}$, k and ω) are carried out each time step. The flow is solved over 40, 000 time steps which corresponds to approximately 44 through-flows; the flow variables are time-averaged over the last 100 time steps.

A wall function is used at the upper wall for the hill flow. It is based on Reichardt's law

$$\frac{\bar{u}_P}{u_\tau} \equiv U_P^+$$



(a) $x = 0.05$. (b) $x = 1$. (c) $x = 3$. (d) $x = 4$. (e) $x = 5$. (f) $x = 7$.

Figure 15. Hill flow. Turbulent shear stresses. ——: $k-\omega$-PINN-NN; – –: standard $k-\omega$; ■ ■: DNS [15]. (a) $x = 0.05$. (b) $x = 1$. (c) $x = 3$. (d) $x = 4$. (e) $x = 5$ and (f) $x = 7$.

$$= \frac{1}{\kappa}\ln(1-0.4y_P^+) + 7.8\left[1-\exp\left(-y_P^+/11\right) - (y_P^+/11)\exp\left(-y_P^+/3\right)\right] \tag{19}$$

The friction velocity is then obtained by re-arranging Equation (19) and solving the implicit equation

$$u_\tau - \bar{u}_P\left\{\ln(1-0.4y_P^+)/\kappa + 7.8\left[1-\exp\left(-y_P^+/11\right) - (y_P^+/11)\exp\left(-y_P^+/3\right)\right]\right\}^{-1} = 0 \tag{20}$$

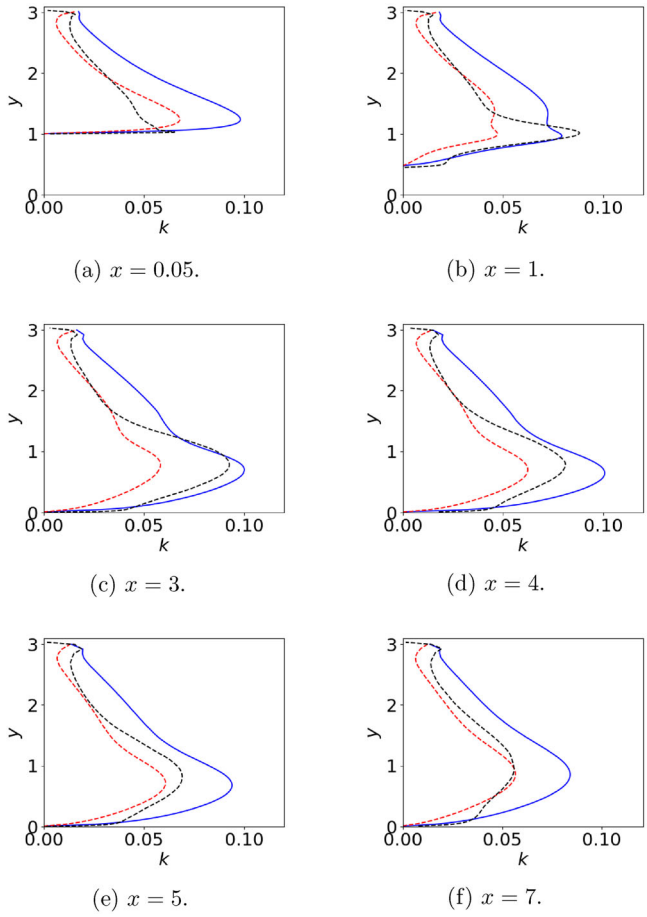
using the Newton–Raphson method `scipy.optimize.newton` in Python. $\bar{u}_P$ and $y_P^+ = u_\tau y/\nu$ denote the wall-parallel velocity and non-dimensional wall distance, respectively, at the first wall-adjacent cells. The boundary condition for the wall-parallel velocity is the shear stress boundary condition, ρu_τ^2. The turbulent kinetic energy, k, and its specific dissipation, ω, are set at the wall-adjacent cells centres as

$$C_\mu^{-1/2}u_\tau^2 \;:\; k \text{ equation}$$
$$\frac{u_\tau}{\kappa y} \;:\; \omega \text{ equation}$$

where $\kappa = 0.4$.

The bulk velocity is kept constant by adjusting β in Equation (9) at each time step and iteration. The coefficient in Equation (9) is computed as

$$\beta^n = \beta^{n-1} + 0.001(U_T - 2U_b^{n-1} + U_b^n) \tag{21}$$



(a) $x = 0.05$. (b) $x = 1$. (c) $x = 3$. (d) $x = 4$. (e) $x = 5$. (f) $x = 7$.

Figure 16. Hill flow. Turbulent kinetic energy. ——: $k-\omega$-PINN-NN; – –: standard $k-\omega$; ▬ ▬: DNS [15]. (a) $x = 0.05$. (b) $x = 1$. (c) $x = 3$. (d) $x = 4$. (e) $x = 5$ and (f) $x = 7$.

where $U_T = 1$ is the target bulk velocity at the hill crest and n is the time step number.

The predictions are compared with DNS [15]. The grid is the same as in the DNS in the x–y plane, i.e. 196 × 128 ($x \times y$) cells. Two cells are used in the z direction. All quantities are normalised using the height of the hill (H) and the bulk velocity (U_b) at the crest of the hill.

Figure 14 compare the predicted velocity profiles with DNS and as can be seen the agreement with the $k-\omega$-PINN-NN model is much better than with the standard $k-\omega$ model. The shear stresses are slightly better predicted by the $k-\omega$-PINN-NN model model than by the standard $k-\omega$ model (which is logical as the velocity profiles are better predicted by the former model). However, it is seen that both models predict much too small magnitude of Reynolds shear stresses in the outer region near and downstream of the crest of the hill (Figure 15(a,b)). The reason may be that there is a large-scale flapping motion near the crest in the DNS that is not captured in steady RANS .

Figure 16 show the predicted turbulent kinetic energy profiles. As can be seen the predicted k with the $k-\omega$-PINN-NN model is in general too large; the agreement for the standard $k-\omega$ model is actually better for $x \geq 5$. It should be mentioned that also the steady full Reynolds-stress closures gives inaccurate turbulent kinetic energy [18] (much too small values).

Figure 17 presents a zoomed-in view of the predicted $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ by the NN model. It can be seen that for $y - y_{wall} \gtrsim 0.03$ they take a constant value at all six x locations. Gray-scale plots in Figure 18(a–c) confirm that they are constant in almost the entire domain. Away from the wall, $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ take the values 1.42, 0.42 and 0.043, respectively (in 88% of the cells in the entire domain we find that $1.42 \leq \sigma_{k,NN} \leq 1.43$, $0.42 \leq C_{k,NN} \leq 0.42$ and $0.043 \leq C_{\omega 2,NN} \leq 0.044$). The reason is that the input parameters to the NN model are limited to the minimum and maximum values during the training process of the NN model, see lines 42–49 in Listing 4 in the Appendix. The maximum values of input parameters τ_{tot}/u_τ^2 and

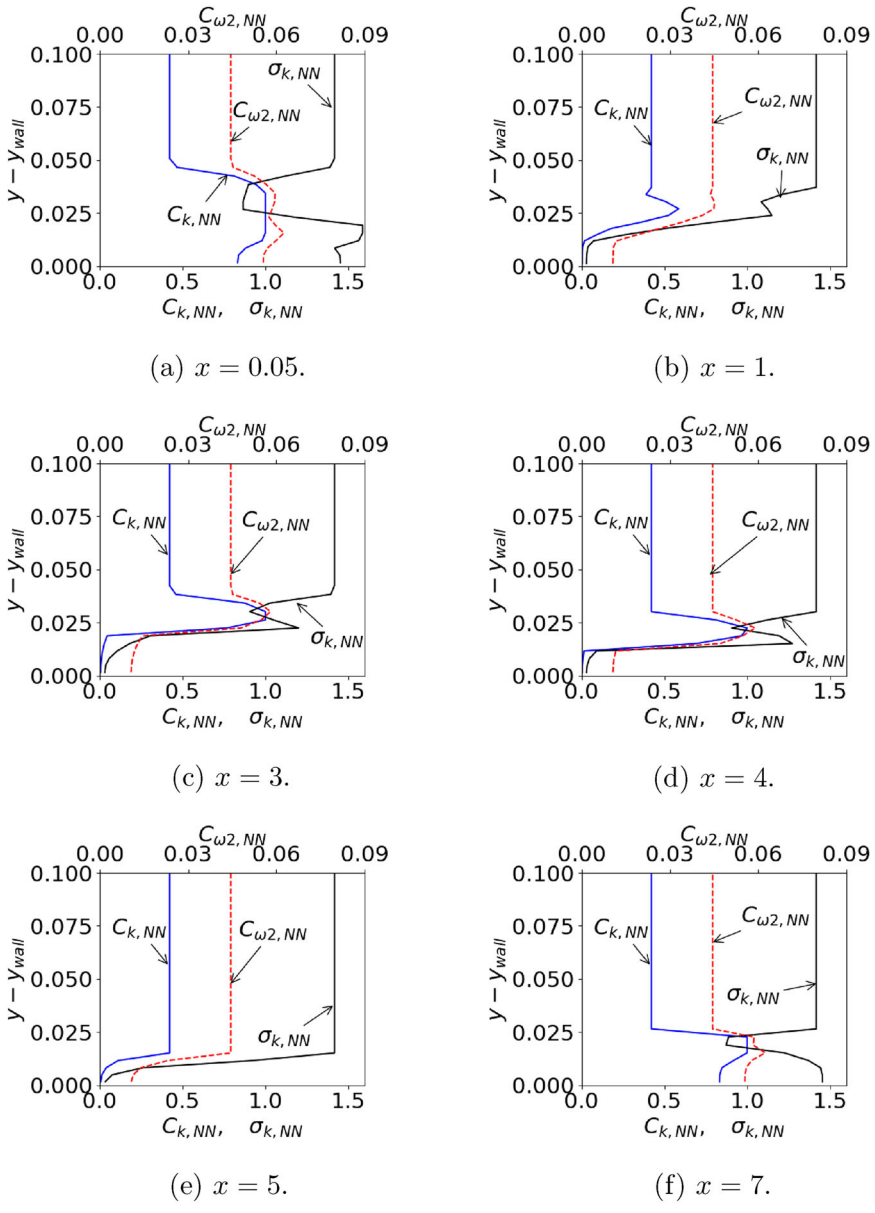


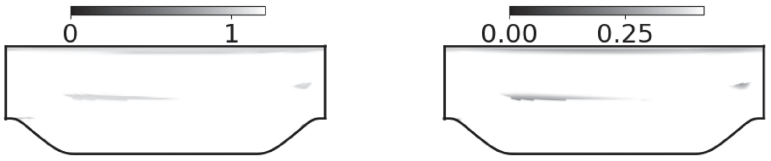
Figure 17. Hill flow. Turbulent Prandtl number, $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ predicted by the NN model. Zoomed-in view. (a) $x = 0.05$. (b) $x = 1$. (c) $x = 3$. (d) $x = 4$. (e) $x = 5$ and (f) $x = 7$.

$\nu_t/(yu_\tau)$ in the NN model are 0.995 and 0.37, respectively. The question now arises: what happens if we use these constant values ($\sigma_{k,NN}, C_{k,NN}, C_{\omega 2,NN} = 1.42, 0.42, 0.043$) in the entire domain? The answer is that the predicted profiles in Figures 14–16 are very similar (not shown). However, the $k-\omega$-PINN-NN model with these constants fails in the channel flows and the flat-plate boundary layer flow; for example, the centreline velocity in channel flow at $Re_\tau = 10,000$ and the skin friction in the flat-plate boundary layer are 10% too low and 17% too high at $Re_\theta = 4500$, respectively (not shown).

Figure 18(d) presents the ratio of $\nu_{tot,NN}$ to $\nu_{tot,k-\omega}$. The peak value is 2.3 and it is smaller than one in a few points near the lower wall. Integrated over the entire domain, the ratio is 1.26.

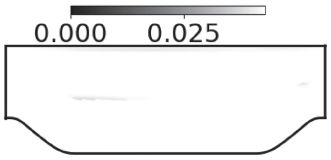
6. Conclusions

The present work proposes a methodology for using PINN and NN for improving turbulence models. A new $k-\omega$ turbulence model, $k-\omega$-PINN-NN, is presented. The k equation is turned into an ODE for the turbulent viscosity, $\nu_{t,PINN}$, by taking k, P^k and ε from DNS. Then the turbulent Prandtl number in the k

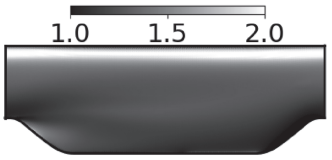


(a) $\sigma_{k,NN}$. Maximum value of $\sigma_{k,NN}$ in color bar is 1.21.

(b) $c_{k,NN}$. Maximum value of $C_{k,NN}$ in color bar is 0.41.



(c) $c_{\omega 2,NN}$. Maximum value of $C_{\omega 2,NN}$ in color bar is 0.043.



(d) Ratio of $\nu_{tot,NN}$ to $\nu_{tot,k-\omega}$. Maximum value of the ratio in color bar is 2.0.

Figure 18. Hill flow. Gray-scale plots of $\sigma_{k,NN}$, $c_{k,NN}$, $c_{\omega 2,NN}$ and ratio of total viscosities. (a) $\sigma_{k,NN}$. Maximum value of $\sigma_{k,NN}$ in colour bar is 1.21. (b) $c_{k,NN}$. Maximum value of $C_{k,NN}$ in colour bar is 0.41. (c) $c_{\omega 2,NN}$. Maximum value of $C_{\omega 2,NN}$ in colour bar is 0.043 and (d) Ratio of $\nu_{tot,NN}$ to $\nu_{tot,k-\omega}$. Maximum value of the ratio in colour bar is 2.0.

equation is given by $\sigma_{k,PINN} = \nu_t/\nu_{t,PINN}$. In order to keep the turbulent viscosity, ν_t, the same as in the standard $k-\omega$ model, two new functions are introduced: one, $C_{k,PINN}$, in front of the dissipation term in the k equation and another, $C_{\omega 2,PINN}$, in front of the destruction term in the ω equation. They are both obtained from the k and ω equations using DNS data.

Next, three NN models were created for predicting the turbulent Prandtl number, $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$. Two input parameters, τ_{tot}/u_τ^2 and $\nu_t/(yu_\tau)$, were used. The difference between $\sigma_{k,PINN}$ and $\sigma_{k,NN}$ is that the former is obtained from $\nu_{t,PINN}$ (which is predicted by PINN) whereas the latter is predicted by the NN model using $\sigma_{k,PINN}$ as the target. The differences between $C_{k,PINN}$ (or $C_{\omega 2,PINN}$) and $C_{k,NN}$ (or $C_{\omega 2,NN}$) is similar: the former (subscript *PINN*) are obtained via PINN and DNS data whereas the latter (subscript *NN*) are predicted by the NN models using the former as targets.

The new $k-\omega$-PINN-NN is shown to predict good skin friction, velocity and k profiles in flat-plate boundary layer flow and fully-developed channel flow. It is also found to give very good results in the periodic hill flow. For the hill flow the NN models predict constant values of $\sigma_{k,NN} = 1.42$, $C_{k,NN} = 0.42$ and $C_{\omega 2,NN} = 0.043$ at $y - y_{wall} \gtrsim 0.03$. The reason is that the input parameters in the NN model are not allowed to exceed the values that were using when training the NN models. An additional simulation of the hill flow was carried out using these constant values for $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$. The predicted results were virtually the same. But when these constant values were used for the channel flow and flat-plate boundary layer, the agreement with DNS was very poor.

One way to further develop the $k-\omega$-PINN-NN model could be to replace the NN models with symbolic regression models. For example, the turbulent Prandtl number, $\sigma_{k,NN}$, can be replaced with $\sigma_{k,pySR}$ which is an algebraic equation. Figure 19 shows

$$\sigma_{k,pySR} = 0.469x_0 + \frac{0.574 + \frac{1}{49.3+\left(x_0x_1^2-0.362\right)^{-1}}}{x_0 + 0.246 + 0.0516x_1/x_0} \tag{22}$$

where $x_0 = \nu_t/(yu_\tau)$ and $x_1 = \tau_{tot}/u_\tau^2$. Equation (22) was found using Python symbolic regression (pySR). In Figure 19 it is compared with $\sigma_{k,PINN}$. One advantage of using Equation (22) instead of the NN model for $\sigma_{k,NN}$ is that Equation (22) can conveniently be used in commercial CFD codes in which it may be difficult to import Pytorch NN models.

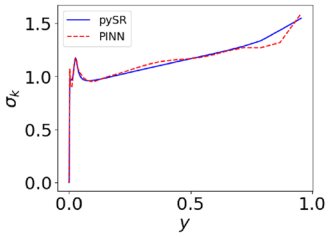


Figure 19. σ_k predicted with pySR.

Acknowledgments

This work was financed by Chalmers University of Technology.

Author contributions

CRediT: **Lars Davidson:** Methodology

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by VINNOVA [2023-01569].

References

[1] Yazdani S, Tahani M. Data-driven discovery of turbulent flow equations using physics-informed neural networks. Phys Fluids. 2024;36(3):035107. doi: 10.1063/5.0190138
[2] Luo S, Vellakal M, Koric S, et al. Parameter identification of RANS turbulence model using physics-embedded neural network. In: Jagode H, Anzt H, Juckeland G, Ltaief H, editors. High performance computing. Cham: Springer International Publishing; 2020. p. 137–149. doi: 10.1007/978-3-030-59851-8
[3] Thakur S, Esmaili E, Libring S, et al. Inverse resolution of spatially varying diffusion coefficient using physics-informed neural networks. Phys Fluids. 2024;36(8):081915. doi: 10.1063/5.0207453
[4] Wilcox DC. Reassessment of the scale-determining equation. AIAA J. 1988;26(11):1299–1310. doi: 10.2514/3.10041
[5] Lee M, Moser RD. Direct numerical simulation of turbulent channel flow up to $Re_\tau \approx 5200$. J Fluid Mech. 2015;774:395–415. doi: 10.1017/jfm.2015.268
[6] Davidson L. pyCALC-RANS: a 2D Python code for RANS. In: Division of fluid dynamics. Gothenburg: Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology; 2021. Available from: https://www.cfd-sweden.se/lada/pyCALC-RANS.html
[7] van Leer B. Towards the ultimate conservative difference scheme. v. A second-order sequel to Godunov's method. J Comput Phys. 1979;32(1):101–136. doi: 10.1016/0021-9991(79)90145-1
[8] Rhie CM, Chow WL. Numerical study of the turbulent flow past an airfoil with trailing edge separation. AIAA J. 1983;21:1525–1532. doi: 10.2514/3.8284
[9] Davidson L. pyCALC-LES: a python code for DNS, LES and hybrid LES-RANS. In: Division of fluid dynamics. Gothenburg: Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology; 2021. Available from: https://www.cfd-sweden.se/lada/postscript_files/py-calc-les.pdf
[10] Davidson L. Using physical informed neural network (PINN) and neural network (NN) to improve a $k-\omega$ turbulence model: python CFD code and PINN script. In: Division of fluid dynamics. Gothenburg: Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology; 2025. Available from: https://www.cfd-sweden.se/lada/Using-Physical-Informed-Neural-Network-PINN-and-NN-improve-a-k-omega-turbulence-model.html
[11] Davidson L. Using physical informed neural network (PINN) to improve a $k-\omega$ turbulence model. In: 15th International ERCOFTAC Symposium on Engineering Turbulence Modelling and Measurements (ETMM15), Dubrovnik on 22-24 September; 2025. Available from: https://www.cfd-sweden.se/lada/Using-Physical-Informed-Neural-Network-PINN-improve-a-k-omega-turbulence-model.html
[12] Davidson L. Fluid mechanics, turbulent flow and turbulence modeling. eBook. Gothenburg: Division of Fluid Dynamics, Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology; 2021. Available from: https://www.cfd-sweden.se/lada/postscript_files/solids-and-fluids_turbulent-flow_turbulence-modelling.pdf

[13] Sillero JA, Jimenez J, Moser RD. One-point statistics for turbulent wall-bounded flows at Reynolds numbers up to $\delta^+ \simeq 2000$. Phys Fluids. 2014;25:105102. doi: 10.1063/1.4823831
[14] Tucker PG. Assessment of geometric multilevel convergence robustness and a wall distance method for flows with multiple internal boundaries. Appl Math Model. 1998;22(4):293–311. doi: 10.1016/S0307-904X(98)10007-0
[15] Froehlich J, Mellen C, Rodi W, et al. Highly-resolved large eddy simulations of separated flow in a channel with streamwise periodic constrictions. J Fluid Mech. 2005;526:19–66. doi: 10.1017/S0022112004002812
[16] Xiao H, Jenny P. A consistent dual-mesh framework for hybrid LES/RANS modeling. J Comput Phys. 2012;231:1848–1865. doi: 10.1016/j.jcp.2011.11.009
[17] Davidson L. Non-zonal detached eddy simulation coupled with a steady RANS solver in the wall region. Int J Heat Fluid Flow. 2021;92:108880. doi: 10.1016/j.ijheatfluidflow.2021.108880
[18] Jakirlic S, Maduta R. Extending the bounds of 'steady' rans closures: toward an instability-sensitive reynolds stress model. Int J Heat Fluid Flow. 2015;51:175–194. Theme special issue celebrating the 75th birthdays of Brian Launder and Kemo Hanjalic.doi: 10.1016/j.ijheatfluidflow.2014.09.003.
[19] Davidson L. Understanding autograd and neural network in Pytorch. In: Division of fluid dynamics. Gothenburg: Dept. of Mechanic Engineering, Chalmers University of Technology; 2026. Available from: https://www.cfd-sweden.se/lada/postscript_files/understanding-autograd-and-neural-network-in-pytorch.pdf

Appendices

Appendix 1. Physics informed NN (PINN)

Let's create a simple NN that finds a damping function in, e.g. the $k-\varepsilon$ model, $Y \equiv f$, as a function of $X \equiv y^+$, see Figure A1. It has one input ($X = a_1^{(0)}$), one hidden layer with two neurons ($a_1^{(1)}, a_2^{(1)}$) and one output ($y_{pred} = a_1^{(2)}$). The target is the correct $Y = f_{DNS}$ taken from DNS. In Listing 1 in Appendix 2, you find the line labelled `Connection 0-1` which connects $a_1^{(0)}$ and $(a_1^{(1)}, a_2^{(1)})$ and the line labelled `Connection 1-2` which connects $(a_1^{(1)}, a_2^{(1)})$ and $a_1^{(2)}$.

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

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

The expressions in the curly parenthesis of s are the arguments of the actuator, e.g. x in `relu = max(0,x)`. The Python code is given in Listing 2 in Appendix 2.

The `loss.backward()` command computes the gradients of the loss, L, with respect to the weights, biases and activators, (i.e. $\partial L/\partial w_1, \partial L/\partial b_1, \partial L/\partial s \ldots$) in order to get new improved $w_1, b_1, w_2, \ldots$

For in-depth understanding of autograd and NN, the reader is referred to [19] and the references therein.

Appendix 2. Simple python scripts

```
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

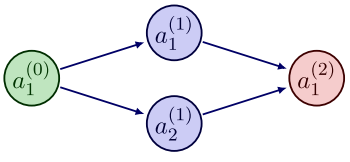


Figure A1. Schematic of a simple NN. One hidden layer with two neurons denoted by circles. The vectors between the neurons represent connections.

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

Listing 2. Simple NN. Prediction and backward step

```
class MyNet(nn.Module):

 def __init__(self):
  super(MyNet, self).__init__()
  self.layer_1 = nn.Linear(1, 30)
  self.layer_2 = nn.Linear(30, 30)
  self.layer_3 = nn.Linear(30, 30)
  self.layer_4 = nn.Linear(30, 30)
  self.layer_5 = nn.Linear(30, 1)

 def forward(self, x):
  x = torch.nn.functional.sigmoid(self.layer_1(x))
  x = torch.nn.functional.sigmoid(self.layer_2(x))
  x = torch.nn.functional.sigmoid(self.layer_3(x))
  x = torch.nn.functional.sigmoid(self.layer_4(x))
  x = torch.nn.functional.sigmoid(self.layer_5(x))

  return x

def ODE(y, nut):
 nut_y = grad(nut, y, torch.ones\
 (y.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. nut_0 = 0 at wall; nut_1 = nut_DNS at centreline
 BC_loss = (nut[0] - nut_0) ** 2 + (nut[-1] - nut_1) ** 2
 return ODE_loss, BC_loss
nut = model(x)
loss_ODE, loss_bc = ODE(y,nut)
L = loss_ODE+100*loss_bc
L.backward()
```

Listing 3. Python code for PINN.

```
def modify_PINN(prand_k_ML, c_k_ML, c_omega_2_ML):
# load packages etc
.
.
.
 if itstep == 0 and iter == 0:
# load data c_k
  NN_c_k = torch.load('NN-model.pth',weights_only=False)
  scaler_vist_over_y = load('model-scaler-vist_over_yin')
  scaler_uv = load('model-scaler-uv_tot.bin')

  name='min-max.txt'
  uv_min,uv_max,vist_over_y_min,vist_over_y_max = xp.loadtxt(name)

# load NN models for prand_k, c_omega_2
.
.
.

# compute ustar, south wall
 ustar_s=(abs(u2d[:,0])*viscos/dist2d[:,0])**0.5
#make it 2D
 ustar_s=xp.repeat(ustar_s[:,None], repeats=nj, axis=1)
# compute ustar, north wall (from wall functions)
 ustar_n=cmu**0.25*k2d[:,-1]**0.5
#make it 2D
 ustar_n=xp.repeat(ustar_n[:,None], repeats=nj, axis=1)
 ywall_s=0.5*(y2d[0:-1,0]+y2d[1:,0])
 dist_s=yp2d-ywall_s
# dist2d = distance to nearest wall
 ustar = xp.where(dist_s > dist2d[:,0],ustar_n,ustar_s)
```

```
# strain-rate tensor
 s11 = dudx
 s12 = 0.5**(dudy+dvdx)
 s21 = s12
 s22 = dvdy
 ss = (2*(s11**2+s12**2+s21**2+s22**2))**0.5
 uv = vis2d*ss/ustar**2
 vist_over_y = (vis2d - viscos)/dist2d/ustar

# limit min/max
# set limits on uv
 uv=xp.minimum(uv,uv_max)
 uv=xp.maximum(uv,uv_min)

# set limits on vist_over_y
 vist_over_y=xp.minimum(vist_over_y,vist_over_y_max)
 vist_over_y=xp.maximum(vist_over_y,vist_over_y_min)

# give input parameters to NN model
 vist_over_y = vist_over_y.reshape(-1,1)
 uv= uv.reshape(-1,1)
 X=xp.zeros((len(uv),2))
 X[:,0] = scaler_vist_over_y.transform(vist_over_y)[:,0]
 X[:,1] = scaler_uv.transform(uv)[:,0]
 X_tensor = torch.tensor(X, dtype=torch.float32)

# predict prand_k
 prand_k_pred = NN_prand_k(X_tensor)
# transform from tensor to numpy
 prand_k_ML = prand_k_pred.detach().numpy()[:,0]
 prand_k_ML = xp.reshape(prand_k_ML,(ni,nj,nk))

# set limits om prand_k_ML
 prand_k_ML=np.minimum(prand_k_ML,prand_k_ML_max)
 prand_k_ML=np.maximum(prand_k_ML,prand_k_ML_min)

# predict c_K_ML and c_omega_2_ML
.
.
.
 return prand_k_ML, c_k_ML, c_omega_2_ML
```

Listing 4. NN model in the CFD code (hill flow).