USING MACHINE LEARNING FOR WALL FUNCTIONS INCLUDING PRESSURE GRADIENTS

Lars Davidson

NFFP8 E-WMLES, June 2024

- Machine learning (ML) is often a method where known data are used for teaching the algorithm to classify a set of data.

Machine Learning

- Machine learning (ML) is often a method where known data are used for teaching the algorithm to classify a set of data.
  - Photographs where the machine learning algorithm should recognize, e.g., traffic lights [4].

- Machine learning (ML) is often a method where known data are used for teaching the algorithm to classify a set of data.
  - Photographs where the machine learning algorithm should recognize, e.g., traffic lights [4].
  - ECG signals where the machine learning algorithm should recognize certain unhealthy conditions of the heart [2].

- Machine learning (ML) is often a method where known data are used for teaching the algorithm to classify a set of data.
  - Photographs where the machine learning algorithm should recognize, e.g., traffic lights [4].
  - ECG signals where the machine learning algorithm should recognize certain unhealthy conditions of the heart [2].
  - Detecting fraud for credit card payments [3].

- Machine learning (ML) is often a method where known data are used for teaching the algorithm to classify a set of data.
  - Photographs where the machine learning algorithm should recognize, e.g., traffic lights [4].
  - ECG signals where the machine learning algorithm should recognize certain unhealthy conditions of the heart [2].
  - Detecting fraud for credit card payments [3].
- In my case, input and output are numerical values.

- Machine learning (ML) is often a method where known data are used for teaching the algorithm to classify a set of data.
    - Photographs where the machine learning algorithm should recognize, e.g., traffic lights [4].
    - ECG signals where the machine learning algorithm should recognize certain unhealthy conditions of the heart [2].
    - Detecting fraud for credit card payments [3].
- In my case, input and output are numerical values.
- The ML will then be some form of regression method.

INITIAL WORK [1]

- Machine Learning (Neural Network) wall functions were developed
- Good results for channel flow placing the wall-adjacent cell at different locations
- Good results for developing boundary layer flow
- You can download my paper where I used `svr` here

$$\frac{\partial k}{\partial t}+\frac{\partial \bar{v}_j k}{\partial x_j}=\frac{\partial}{\partial x_j}\left[\left(\nu+\frac{\nu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_j}\right]+P_k-\psi\varepsilon,\quad \psi=\frac{L_t}{L_{hyb}},\quad L_t=\frac{k^{3/2}}{\varepsilon}$$

$$\frac{\partial \varepsilon}{\partial t}+\frac{\partial \bar{v}_j \varepsilon}{\partial x_j}=\frac{\partial}{\partial x_j}\left[\left(\nu+\frac{\nu_t}{\sigma_\varepsilon}\right)\frac{\partial \varepsilon}{\partial x_j}\right]+C_{\varepsilon 1}P_k\frac{\varepsilon}{k}-C_{\varepsilon 2}f_2\frac{\varepsilon^2}{k}$$

$$\nu_t=C_\mu f_\mu\frac{k^2}{\varepsilon},\quad P_k=\nu_t\left(\frac{\partial \bar{v}_i}{\partial x_j}+\frac{\partial \bar{v}_j}{\partial x_i}\right)\frac{\partial \bar{v}_i}{\partial x_j}, C_{\varepsilon 1}=1.5, C_{\varepsilon 2}=1.9, C_\mu=0.09, \sigma_k=1.4$$

The damping functions are defined as

$$f_2=\left[1-\exp\left(-\frac{y^*}{3.1}\right)\right]^2\left\{1-0.3\exp\left[-\left(\frac{R_t}{6.5}\right)^2\right]\right\},\quad y^*=\frac{u_\varepsilon d}{\nu}$$

$$f_\mu=\left[1-\exp\left(-\frac{y^*}{14}\right)\right]^2\left\{1+\frac{5}{R_t^{3/4}}\exp\left[-\left(\frac{R_t}{200}\right)^2\right]\right\},\quad u_\varepsilon=(\varepsilon\nu))^{1/4},\quad Re_t=\frac{k^2}{\nu\varepsilon}$$

where d denotes the distance to the wall.

$$L_{hyb} = f_d(1 + f_e)L_t + (1 - f_d)C_{DES}\Delta \tag{1}$$

where the Δ length scale is defined as

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

and $C_w = 0.15$, d_w is the distance to the wall; h_{wn} is the grid step in wall normal dir.

$$f_d = \max\left\{(1 - f_{dt}), f_B\right\}, \quad f_e = \max\left\{(f_{e1} - 1), 0\right\} f_{e2} \tag{2}$$

where the functions f_{dt} and f_B entering Eq. 2 are given by

$$f_{dt} = 1 - \tanh\left[(8r_{dt})^3\right], \quad f_B = \min\left\{2\exp\left(-9\alpha^2\right), 1\right\}, \quad \alpha = 0.25 - d_w/h_{max} \tag{3}$$

- The functions f_{e1} and f_{e2} in Eq. 2 read

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

and

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

where the functions f_t and f_l are given by

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

- The constants c_t and c_l given the same values as in the $k-\omega$ SST model, i.e. $c_t = 1.87$ and $c_l = 5$ [5]. The quantities r_{dt} (also entering Eq. 3) and r_{dl}, are defined as follows

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

- The IDDES model is implemented in the folder `hump-IDDES-ni-583-go4hybrid-mesh-STG-dt-0.002-GPU/` in pyCALC-LES.

$$
\begin{aligned}
y_P^+ &\quad : \quad \text{influence/inlet parameter} \\
P^+ = \nu(\partial \bar{p}/\partial x_1)/u_\tau^3 &\quad : \quad \text{influence/inlet parameter} \\
U^+ &\quad : \quad \text{output parameter} \\
u_\tau &\quad : \quad \bar{u}_P/U^+
\end{aligned}
$$

$$
\begin{aligned}
y_P^+ &: \quad \text{influence/inlet parameter}\\
P^+ = \nu(\partial\bar{p}/\partial x_1)/u_\tau^3 &: \quad \text{influence/inlet parameter}\\
U^+ &: \quad \text{output parameter}\\
u_\tau &: \quad \bar{u}_P/U^+
\end{aligned}
$$

$$
\begin{aligned}
\rho u_\tau^2 &: \quad \bar{u}\text{ equation}\\
C_\mu^{-1/2} u_\tau^2 &: \quad k\text{ equation}\\
\frac{u_\tau^3}{\kappa y} &: \quad \varepsilon\text{ equation}
\end{aligned}
$$

- Well resolved LES, $600 \times 150 \times 300$, $0.3 < \Delta y^+ < 22$, $\Delta z^+ = 11$, $\Delta x^+ = 22$

- Well resolved LES, $600 \times 150 \times 300$, $0.3 < \Delta y^+ < 22$, $\Delta z^+ = 11$, $\Delta x^+ = 22$
  - Bad idea!. There are two modeling errors: IDDES model and wall functions.

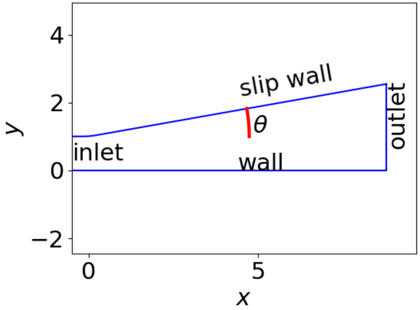
- Well resolved LES, $600 \times 150 \times 300$, $0.3 < \Delta y^+ < 22$, $\Delta z^+ = 11$, $\Delta x^+ = 22$
  - Bad idea!. There are two modeling errors: IDDES model and wall functions.
- Instead: create the target database by using the same IDDES model but on a low-Re mesh (i.e. $y^+ \leq 1$)

- Well resolved LES, $600 \times 150 \times 300$, $0.3 < \Delta y^+ < 22$, $\Delta z^+ = 11$, $\Delta x^+ = 22$
  - Bad idea!. There are two modeling errors: IDDES model and wall functions.
- Instead: create the target database by using the same IDDES model but on a low-Re mesh (i.e. $y^+ \leq 1$)
- Goal: the NN wall function should perform as well as the low-Re IDDES

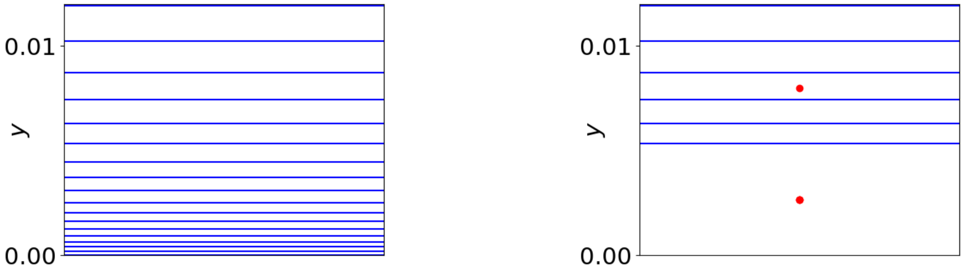
DIFFUSER, LES WITH WALE MODEL, PRESSURE GRADIENT

- Well resolved LES, $600 \times 150 \times 300$, $0.3 < \Delta y^+ < 22$, $\Delta z^+ = 11$, $\Delta x^+ = 22$
  - Bad idea!. There are two modeling errors: IDDES model and wall functions.
- Instead: create the target database by using the same IDDES model but on a low-Re mesh (i.e. $y^+ \leq 1$)
- Goal: the NN wall function should perform as well as the low-Re IDDES
- Diffuser angle: $10 \leq \theta \leq 15^o$

- Well resolved LES, $600 \times 150 \times 300$, $0.3 < \Delta y^+ < 22$, $\Delta z^+ = 11$, $\Delta x^+ = 22$
  - Bad idea!. There are two modeling errors: IDDES model and wall functions.
- Instead: create the target database by using the same IDDES model but on a low-Re mesh (i.e. $y^+ \leq 1$)
- Goal: the NN wall function should perform as well as the low-Re IDDES
- Diffuser angle: $10 \leq \theta < 15^o$



WALL FUNCTION MESH



(A) Low-Re number IDDES grid.

(B) Wall function grid. First cell is formed by merging 12 cells in the low-Re grid.

FIGURE: Different grids. —— : grid lines. •: cell center

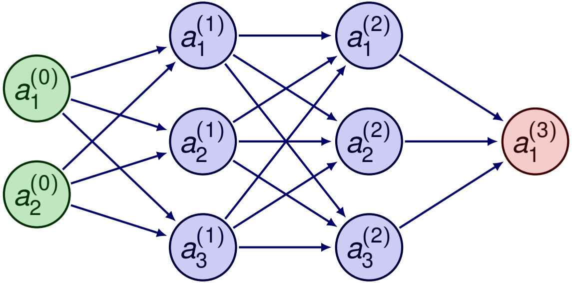
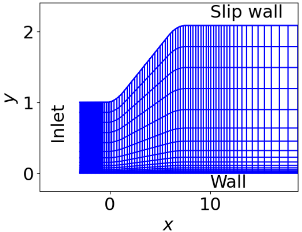
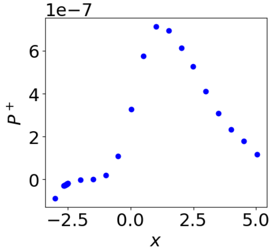


FIGURE: The Neural Network with two inputs variables, $a_1^{(0)} = y^+$ and $a_2^{(0)} = P^+$ and one output variable, $a_1^{(3)} = U^+$. From U^+ I get u_τ. There are three neurons in this figure; in the simulations I have 50.

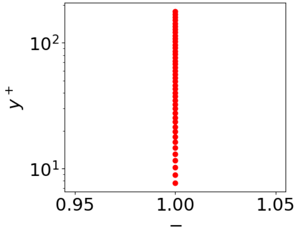
TRAINING TIME-AVERAGED DATA FOR NN, 10^o



(A) Low-Re IDDES grid.

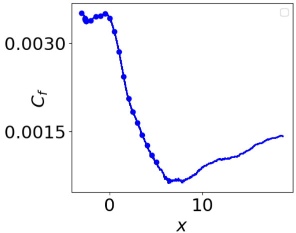


(B) P^+. Sampling points.

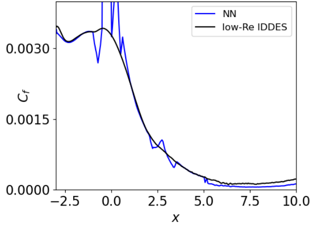


(C) 43 Sampling points in y.

- Input y^+ and P^+ from low-Re IDDES at $y^+ \simeq 35$
- I use NN to predict U^+ (u_τ)



(D) C_f. Sampling points.

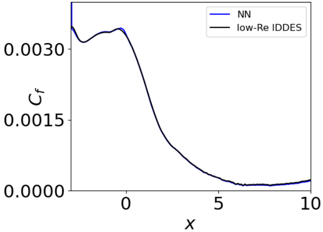


(E) Predicted C_f using NN (no CFD)

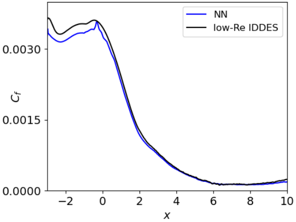
ADD INPUT PARAMETERS

- Use P^+ and $Re = \bar{u}d_w/\nu$ as input parameters (d_w: wall distance)
- Use P^+ and $\frac{\partial \bar{v}_1}{\partial x_2}\nu/U_B^2$ as input parameters

- Input y^+, P^+, Re (or $\frac{\partial \bar{v}_1^*}{\partial x_2}$) from low-Re IDDES at $y^+ \simeq 35$
- I use NN to predict U^+ (u_τ)



(A) Predicted C_f. Re added input.



(B) Predicted C_f. $\frac{\partial \bar{v}_1}{\partial x_2}$ added input.

FIGURE: Evaluating two different sets of input parameters (no CFD).

- Wall functions based on Neural Network (NN) or Reichardt wall functions
- Wall functions based Reichardt's law

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

  is solved using the Newton-Raphson method `scipy.optimize.newton` in Python.
- Turbulence model: IDDES based on the AKN low-Re $k-\varepsilon$ model
- Instantaneous inlet b.c. from pre-cursor channel IDDES
- Grids; $600\times70\times96$, $600\times70\times48$ or $300\times70\times48$ (low-Re IDDES grid: $600\times90\times96$)

- The input pressure gradient reads

$$P^+ = \nu \frac{\partial \bar{p}/\partial x_1}{u_\tau^3}$$

TIME AVERAGING P^+, y^+, Re AND $\frac{\partial \bar{v}_1}{\partial x_2}$

- The input pressure gradient reads

$$P^+ = \nu \frac{\partial \bar{p}/\partial x_1}{u_\tau^3}$$

- Both $\partial \bar{p}/\partial x_1$ and u_τ^3 are very unsteady; P^+ can be very large when u_τ gets small

TIME AVERAGING P^+, y^+, Re AND $\frac{\partial \bar{v}_1}{\partial x_2}$

- The input pressure gradient reads

$$P^+ = \nu \frac{\partial \bar{p}/\partial x_1}{u_\tau^3}$$

- Both $\partial \bar{p}/\partial x_1$ and u_τ^3 are very unsteady; P^+ can be very large when u_τ gets small
- I always limit the input variable to min/max of training data

TIME AVERAGING P^+, y^+, Re AND $\frac{\partial \bar{v}_1}{\partial x_2}$

- The input pressure gradient reads

$$P^+ = \nu \frac{\partial \bar{p}/\partial x_1}{u_\tau^3}$$

- Both $\partial \bar{p}/\partial x_1$ and u_τ^3 are very unsteady; P^+ can be very large when u_τ gets small
- I always limit the input variable to min/max of training data
- Instead I use $\langle P^+ \rangle = \nu \frac{\langle \partial \bar{p}/\partial x_1 \rangle}{U_B^3}$ (running average)

TIME AVERAGING P^+, y^+, Re AND $\frac{\partial \bar v_1}{\partial x_2}$

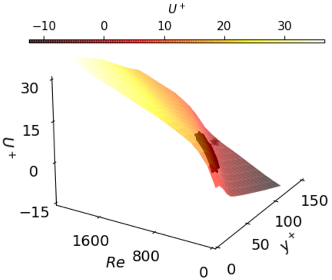
- The input pressure gradient reads

$$P^+ = \nu \frac{\partial \bar p/\partial x_1}{u_\tau^3}$$

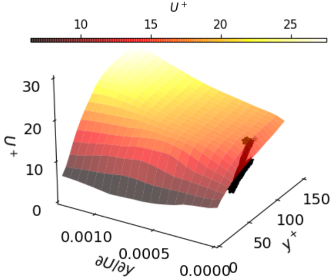
- Both $\partial \bar p/\partial x_1$ and u_τ^3 are very unsteady; P^+ can be very large when u_τ gets small
- I always limit the input variable to min/max of training data
- Instead I use $\langle P^+\rangle = \nu \frac{\langle \partial \bar p/\partial x_1\rangle}{U_B^3}$ (running average)
- I also time average the input parameters y^+ and Re (or $\frac{\partial \bar v_1^*}{\partial x_2} = \frac{\partial \bar v_1}{\partial x_2}\nu/U_B^2$)

Check parameter space

- Left: Input y^+, P^+, $\frac{\partial \bar{v}_1^*}{\partial x_2}$. May give negative U^+ although it was trained on $U^+ > 0$
- Right: Input y^+, P^+, Re. Give always $U^+ > 0$



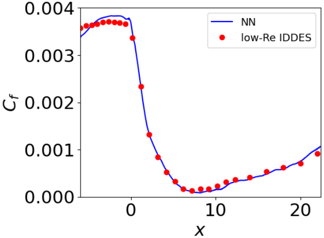
(A) Input y^+, P^+, $\frac{\partial \bar{v}_1^*}{\partial x_2}$.



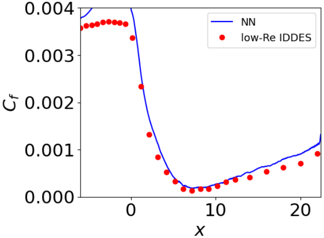
(B) Input y^+, P^+, Re.

FIGURE: Markers: NN predictions in the diffuser, $\alpha = 10^o$. $P^+ = 0.05(P^+_{max} - P^+_{min})$.

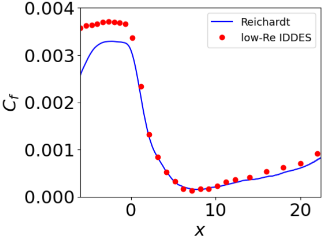
- NN wall function trained on the same diffuser., $\alpha = 10^o$ (this is the first attempt)
- database: low-IDDES on $600 \times 90 \times 96$ mesh
- wall function mesh: $600 \times 70 \times 96$ mesh



(A) NN wall functions. Input y^+, P^+, Re.

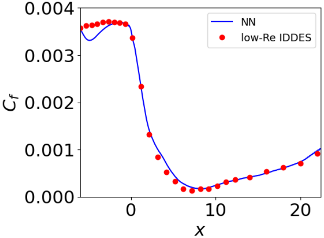


(B) NN wall functions. Input y^+, P^+, $\frac{\partial \bar{v}_1^*}{\partial x_2}$.

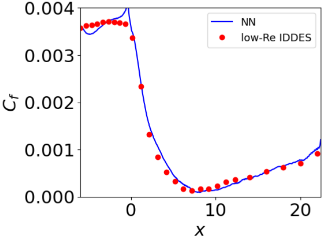


(C) Reichardt wall functions.

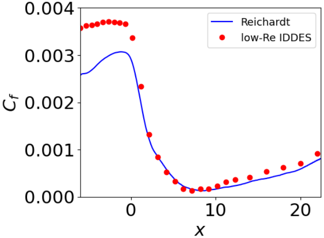
- NN wall function trained on the same diffuser., $\alpha = 10^o$ (this is the first attempt)
- database: low-IDDES on $600 \times 90 \times 96$ mesh
- wall function mesh: $600 \times 70 \times 48$ mesh



(A) NN wall functions. Input y^+, P^+, Re.



(B) NN wall functions. Input y^+, P^+, $\frac{\partial \bar{v}_1^*}{\partial x_2}$.



(C) Reichardt's wall function.

PYTHON SCRIPTS FOR y^+, P^+, Re

- creating the NN wall model
- part of CFD code **pyCALC-LES** where the NN wall model is used

NN ON RECIRCULATING FLOW. SWITCH TO KDTREE

- I tried NN wall function at $\alpha = 15^o$ (with recirculation).
- It does not work (diverges)
- Instead, I start with KDtree (look-up table)
- Input and output: U^+ and y^+.
- u_τ can be taken from U^+ or y^+

- The NN wall-function for flows including pressure gradients has been presented

- The NN wall-function for flows including pressure gradients has been presented
  - I need to add one input parameter in addition of y^+ and U^+: *Re* or $\partial \bar{u}/\partial y$.

Conclusions

- The NN wall-function for flows including pressure gradients has been presented
  - I need to add one input parameter in addition of y^+ and U^+: *Re* or $\partial \bar{u}/\partial y$.
  - Works fine as long as the flow is attached; diverges when backflow

- The NN wall-function for flows including pressure gradients has been presented
  - I need to add one input parameter in addition of y^+ and U^+: *Re* or $\partial \bar{u}/\partial y$.
  - Works fine as long as the flow is attached; diverges when backflow
- KDtree works much better

- The NN wall-function for flows including pressure gradients has been presented
  - I need to add one input parameter in addition of y^+ and U^+: *Re* or $\partial\bar{u}/\partial y$.
  - Works fine as long as the flow is attached; diverges when backflow
- KDtree works much better
- Python scripts and my paper *Hybrid LES/RANS for flows including separation: A new wall function using Machine Learning based on binary search trees* can be downloaded here

References

[1] L. Davidson. Using machine learning for formulating new wall functions for Detached Eddy Simulation. In *14th International ERCOFTAC Symposium on Engineering Turbulence Modelling and Measurements (ETMM14), Barcelona/Digital, Spain 6–8 September*, 2023.

[2] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas Schön. *Machine Learning: A First Course for Engineers and Scientists*. Cambridge University Press, 2022.

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

[4] Sudarshana S Rao and Santosh R Desai. Machine learning based traffic light detection and ir sensor based proximity sensing for autonomous cars. In *Proceedings of the International Conference on IoT Based Control Networks & Intelligent Systems – ICICNIS*, 2021.

[5] M. L. Shur, P. R. Spalart, M. Kh. Strelets, and A. K. Travin. A hybrid RANS-LES approach with delayed-DES and wall-modelled LES capabilities. *International Journal of Heat and Fluid Flow*, 29:1638–1649, 2008.