

USING MACHINE LEARNING FOR WALL FUNCTIONS INCLUDING PRESSURE GRADIENTS

Lars Davidson

LESisMORE, Kickoff, Sept 2024

[Download paper and Python scripts](#)

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

- 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 [11].

- 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 [11].
 - **ECG signals** where the machine learning algorithm should recognize certain unhealthy conditions of the heart [9].

- 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 [11].
 - **ECG signals** where the machine learning algorithm should recognize certain unhealthy conditions of the heart [9].
 - **Detecting fraud** for credit card payments [10].

- 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 [11].
 - **ECG signals** where the machine learning algorithm should recognize certain unhealthy conditions of the heart [9].
 - **Detecting fraud** for credit card payments [10].
- 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 [11].
 - **ECG signals** where the machine learning algorithm should recognize certain unhealthy conditions of the heart [9].
 - **Detecting fraud** for credit card payments [10].
- In my case, input and output are **numerical** values.
- The ML will then be some form of **regression method**.

INITIAL WORK [6]

- 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
- Training the Neural Network with steady or instantaneous data: **same results**
- Training nearest neighbor (Python's `scipy.spatial.KDTree`) with instantaneous data: **same results**

- **KDTree** will be used for finding y^+ .
- It is essentially a fast look-up table
- There will be two sets of data points.
 - One is the target data set, i.e. low-Re IDDES ($\mathbf{X} = [U_{target}^+, y_{target}^+]$)
 - The other one is the wall-function IDDES ($\mathbf{x} = [U_{CFD}^+, y_{CFD}^+]$)
- **KDTree** computes the distance between the vectors as

$$\mathbf{d}_s = \mathbf{X}_i - \mathbf{x}_j \quad (1)$$

for all samples i and j and finds the k nearest neighbors for each j .

THE NUMERICAL METHOD

THE NUMERICAL METHOD

- The **Python** finite volume code **pyCALC-LES** [5] is used.

THE NUMERICAL METHOD

- The **Python** finite volume code **pyCALC-LES** [5] is used.
- Fully vectorized (i.e. no `for` loops).

THE NUMERICAL METHOD

- The **Python** finite volume code **pyCALC-LES** [5] is used.
- Fully vectorized (i.e. no `for` loops).
- Fractional step. For velocities, second-order central differencing in space and Crank-Nicolson in time.

THE NUMERICAL METHOD

- The **Python** finite volume code **pyCALC-LES** [5] is used.
- Fully vectorized (i.e. no `for` loops).
- Fractional step. For velocities, second-order central differencing in space and Crank-Nicolson in time.
- For k and ε , hybrid central/upwind scheme

THE NUMERICAL METHOD

- The **Python** finite volume code **pyCALC-LES** [5] is used.
- Fully vectorized (i.e. no `for` loops).
- Fractional step. For velocities, second-order central differencing in space and Crank-Nicolson in time.
- For k and ε , hybrid central/upwind scheme
- The discretized equations are solved with **Python sparse matrix solvers**.

THE NUMERICAL METHOD

- The **Python** finite volume code **pyCALC-LES** [5] is used.
- Fully vectorized (i.e. no `for` loops).
- Fractional step. For velocities, second-order central differencing in space and Crank-Nicolson in time.
- For k and ε , hybrid central/upwind scheme
- The discretized equations are solved with **Python sparse matrix solvers**.
- It runs either on the CPU or the GPU (the GPU is up to **70 times** faster)

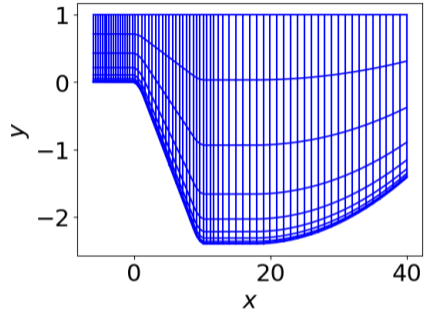
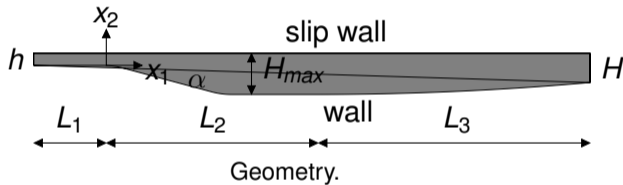
THE NUMERICAL METHOD

- The **Python** finite volume code **pyCALC-LES** [5] is used.
- Fully vectorized (i.e. no `for` loops).
- Fractional step. For velocities, second-order central differencing in space and Crank-Nicolson in time.
- For k and ε , hybrid central/upwind scheme
- The discretized equations are solved with **Python sparse matrix solvers**.
- It runs either on the CPU or the GPU (the GPU is up to **70 times** faster)
- On the GPU, the Algebraic Multigrid solver in AMGX is used; it **very** fast.

THE NUMERICAL METHOD

- The **Python** finite volume code **pyCALC-LES** [5] is used.
- Fully vectorized (i.e. no `for` loops).
- Fractional step. For velocities, second-order central differencing in space and Crank-Nicolson in time.
- For k and ε , hybrid central/upwind scheme
- The discretized equations are solved with **Python sparse matrix solvers**.
- It runs either on the CPU or the GPU (the GPU is up to **70 times** faster)
- On the GPU, the Algebraic Multigrid solver in AMGX is used; it **very** fast.
- `cupy` is used to switch from CPU to GPU (`import cupy`)

CREATE TARGET DATABASE 1: DIFFUSER

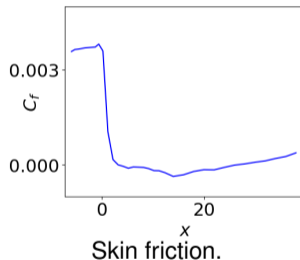
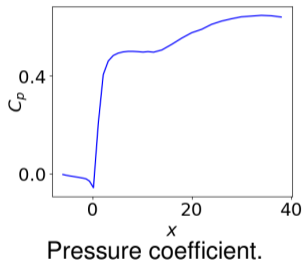


Grid, $x - y$ plane (not to scale). 700×90 cells. Every 10^{th} grid line is shown.

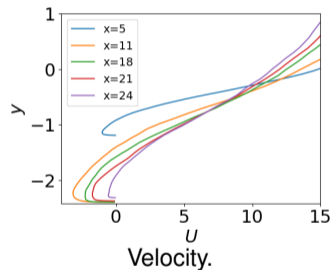
Diffuser, $\alpha = 15^\circ$.

TARGET DATABASE: RESULTS

- $700 \times 90 \times 96$. $k - \varepsilon$ IDDES.
- Inlet b.c. from pre-cursor IDDES channel flow at $Re_\tau = 5\,200$.

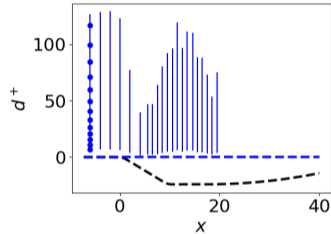


*

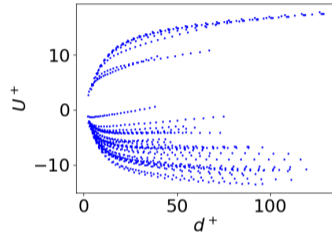


Diffuser flow. Target data base.

TARGET DATABASE FOR **KDTREE** . BASELINE: $K = 5$ (FIVE NBRS)



Data points of y^+ vs. x .



Scatter plot of U^+ and y^+ .

Diffuser flow. The target database consists of time-averaged 41 profiles of U^+ vs. y^+ with 26 points in each profile. d is the wall distance. Every second x line and y point are shown.

INPUT/OUTPUT IN THE **KD**TREE .

$$\begin{aligned} y_P^+ &: \quad \text{inlet and outlet parameter} \\ U^+ &: \quad \text{inlet and output parameter} \\ u_\tau &: \quad y_P^+ \nu / y_P \end{aligned}$$

INPUT/OUTPUT IN THE **KD**TREE .

y_P^+ : inlet and outlet parameter

U^+ : inlet and output parameter

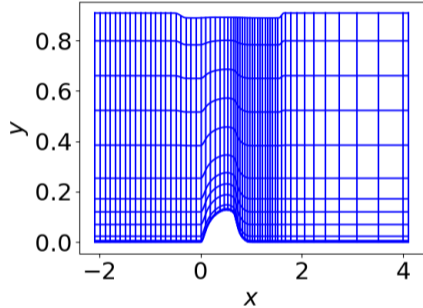
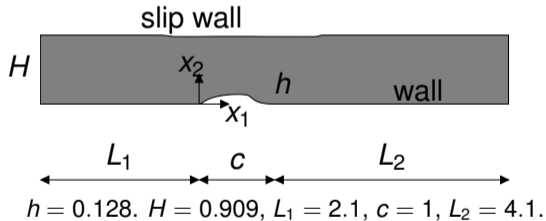
u_τ : $y_P^+ \nu / y_P$

ρu_τ^2 : \bar{u} equation

$C_\mu^{-1/2} u_\tau^2$: k equation

$\frac{u_\tau^3}{\kappa y}$: ε equation

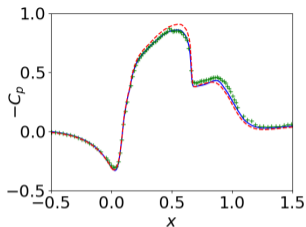
CREATE TARGET DATABASE 2: HUMPH



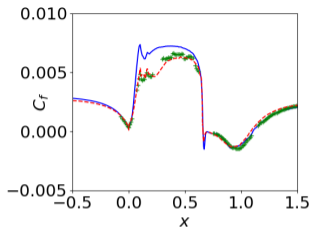
Grid. $582 \times 128 \times 64$ cells. Every 10^{th} .

Hump flow.

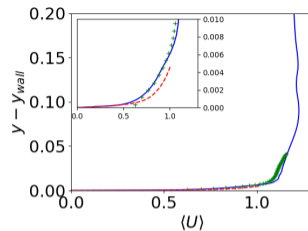
TARGET DATABASE 2: RESULTS



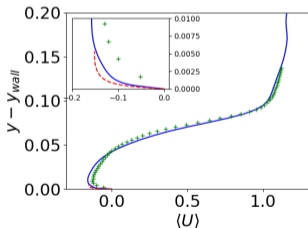
Pressure coefficient.



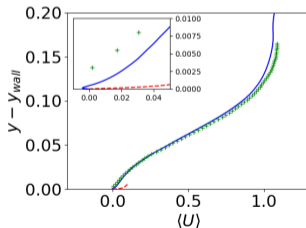
Friction coefficient.



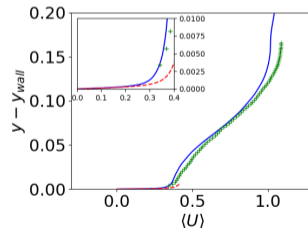
Velocity at $x = 0.65$.



Velocity at $x = 0.80$.



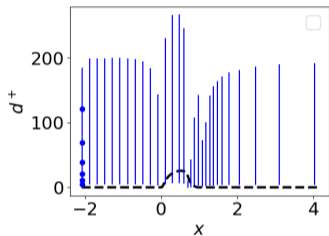
Velocity at $x = 1.10$.



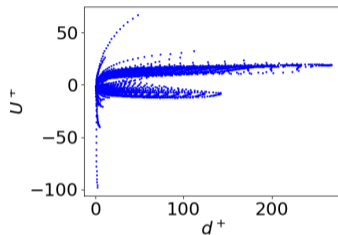
Velocity at $x = 1.30$.

Hump flow, low-Re IDDES. +: experiments [8, 7].

TARGET DATABASE FOR **KDTREE** . BASELINE: $K = 1$ (ONE NBR).



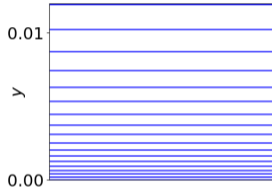
Data points of y^+ vs. x .



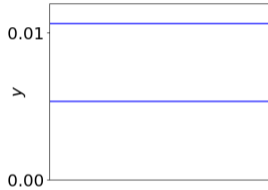
Scatter plot of U^+ and y^+ .

Hump flow. d is the wall distance. The target database consists of time-averaged 582 profiles (all grid lines) of U^+ vs. y^+ with 24 points in each profile. Every 20th x line and every 4th y point are shown.

NEW WALL FUNCTION GRID STRATEGY



Low-Re number grid.



Wall function grid.



New wall function grid.

Different grids. — : grid lines.

DIFFUSER FLOW, WALL FUNCTIONS: SETUP

- Wall functions based on **KDTree** or Reichardt wall functions
- Wall functions based Reichardt's law

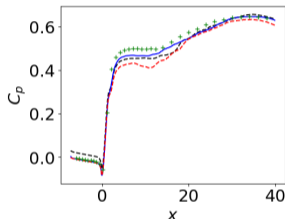
$$\frac{\bar{u}_P}{u_\tau} \equiv U^+ = \frac{1}{\kappa} \ln(1 - 0.4y^+) + 7.8 \left[1 - \exp(-y^+/11) - (y^+/11) \exp(-y^+/3) \right]$$

is solved using the Newton-Raphson method `scipy.optimize.newton` in Python.

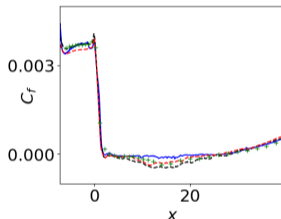
- Turbulence model: IDDES based on the AKN low-Re $k - \varepsilon$ model
- Instantaneous inlet b.c. from pre-cursor channel IDDES using **KDTree** wall functions
- Grid: $462 \times 70 \times 48$ (low-Re IDDES grid: $600 \times 90 \times 96$)

RESULTS, DIFFUSER FLOW, $\alpha = 15^\circ$

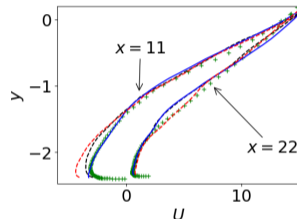
- $468 \times 70 \times 48$ cells (every 2^{nd} in x and z)



Pressure coefficient.



Skin coefficient.

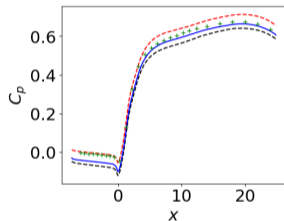


Velocity profiles.

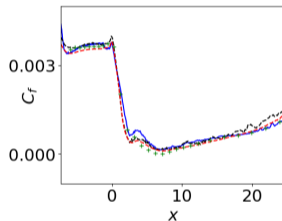
Diffuser flow, $\alpha = 15^\circ$. — : **KDTree** using hump flow data; - - - : **KDTree** using diffuser flow data; - - - : Reichardt's law; + : low-Re IDDES.

RESULTS, DIFFUSER FLOW, $\alpha = 10^0$

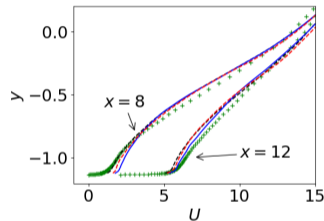
- $387 \times 70 \times 48$ cells (every 2^{nd} in x and z)



Pressure coefficient.



Skin coefficient.



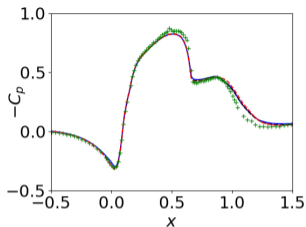
Velocity profiles.

Diffuser flow, $\alpha = 10^0$. — : **KDTree** using hump flow data; - - : **KDTree** using diffuser flow data; - - : Reichardt's law; + : low-Re IDDES.

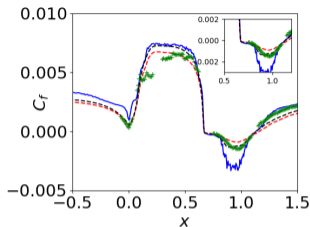
HUMP FLOW, WALL FUNCTIONS: SETUP

- The Reynolds number is $Re_c = 936\,000$. Spanwise extent is $z_{max} = 0.2$.
- The mesh has $291 \times 106 \times 64/32$ cells $[x, y, z]$ (low-Re IDDES $582 \times 106 \times 64$)
- Inlet b.c.
 - Mean from 2D RANS
 - Inlet turbulence: fluctuation from STG
 - Inlet k and ε : 2D RANS plus commutation term in k eq. [3, 1] (Model 3)
- Comparison with
 - Experiments [8, 7]

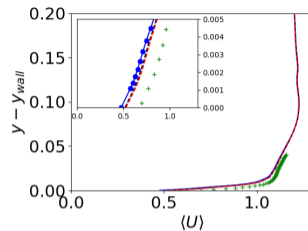
RESULTS, HUMP FLOW. $583 \times 106 \times 64$ CELLS.



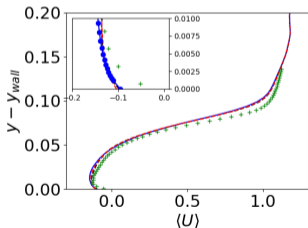
Pressure coefficient.



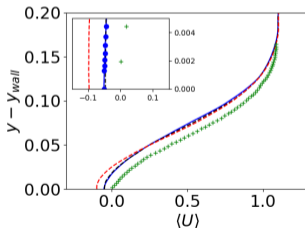
Friction coefficient.



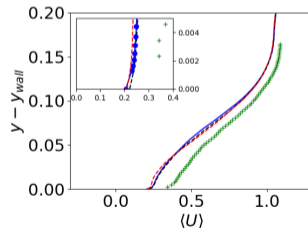
$x = 0.65$.



$x = 0.80$.



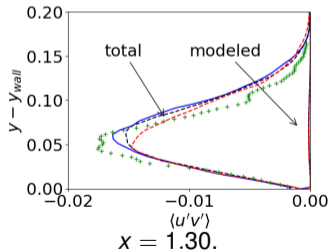
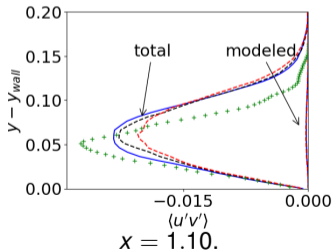
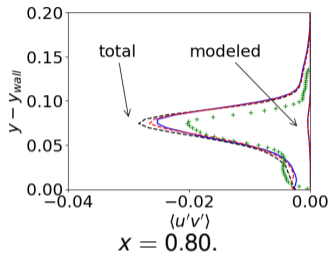
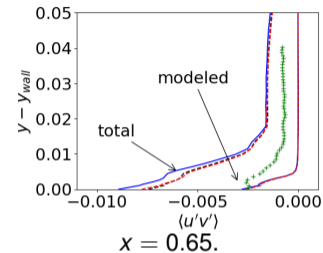
$x = 1.10$.



$x = 1.30$.

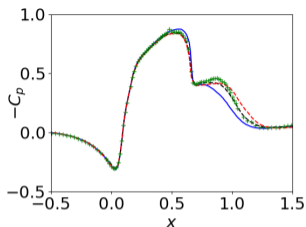
— : KDTree hump data; --- : KDTree diffuser data; - - : Reichardt's law; + : exp.

RESULTS, HUMP FLOW. $583 \times 106 \times 64$ CELLS. SHEAR STRESSES

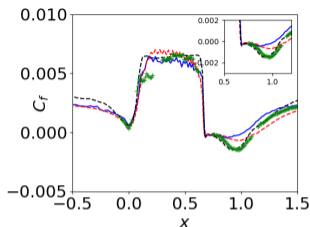


— : KDTree hump data; --- : KDTree diffuser data; - - : Reichardt's law; + : exp.

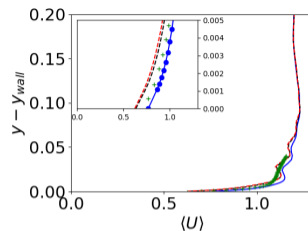
RESULTS, HUMP FLOW. $291 \times 106 \times 32$ CELLS.



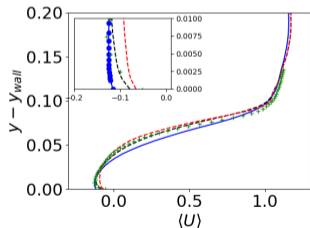
Pressure coefficient.



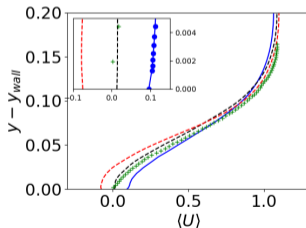
Friction coefficient.



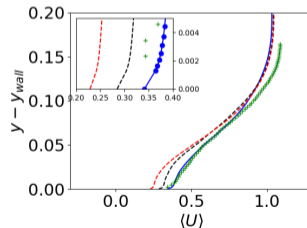
$x = 0.65$.



$x = 0.80$.



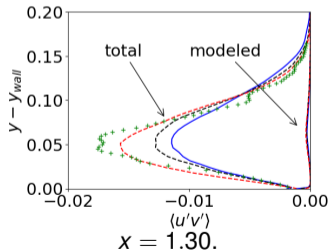
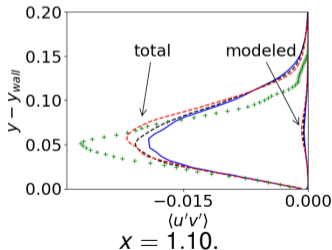
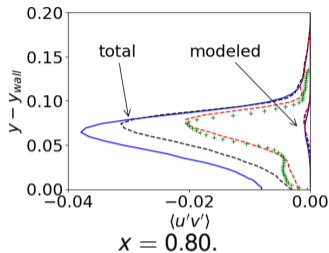
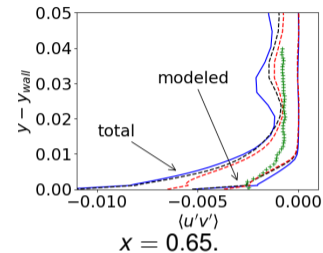
$x = 1.10$.



$x = 1.30$.

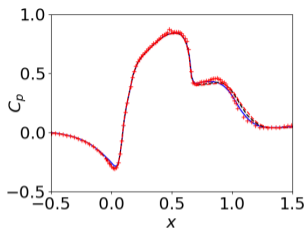
— : KDTree hump data; --- : KDTree diffuser data; - - : Reichardt's law; + : exp.

RESULTS, HUMP FLOW. $291 \times 106 \times 32$ CELLS. SHEAR STRESSES

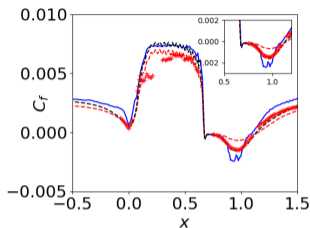


— : KDTree hump data; --- : KDTree diffuser data; - - : Reichardt's law; + : exp.

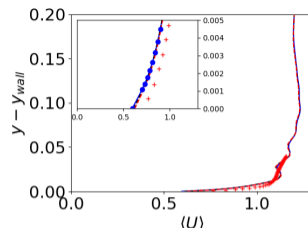
RESULTS, HUMP FLOW. $291 \times 106 \times 32$ CELLS, $K = 5$.



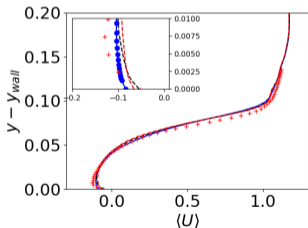
Pressure coefficient.



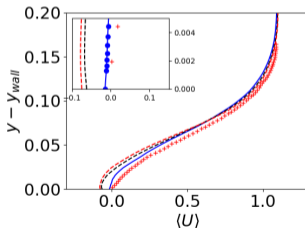
Friction coefficient.



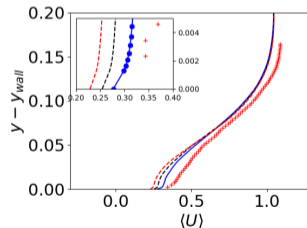
$x = 0.65$.



$x = 0.80$.



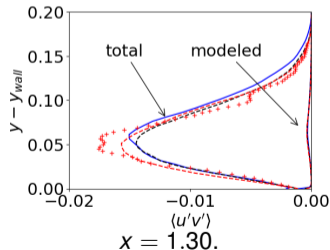
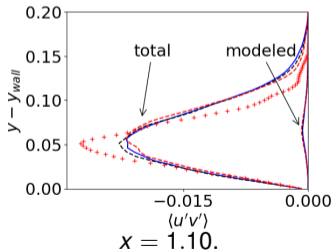
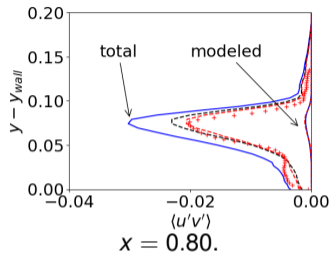
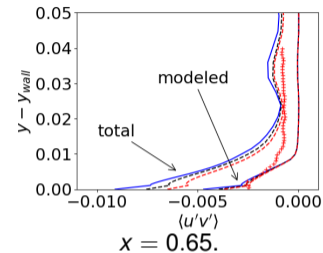
$x = 1.10$.



$x = 1.30$.

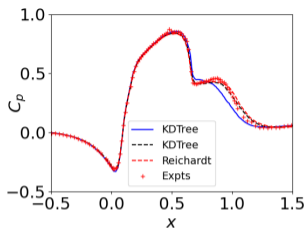
— : KDTree hump data; --- : KDTree diffuser data; - - : Reichardt's law; + : exp.

HUMP FLOW. $291 \times 106 \times 32$ CELLS. SHEAR STRESSES, $K = 5$.

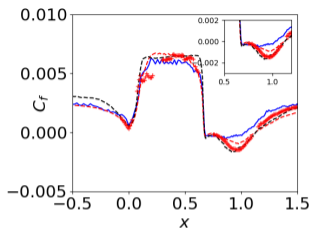


— : KDTree hump data; --- : KDTree diffuser data; - - : Reichardt's law; + : exp.

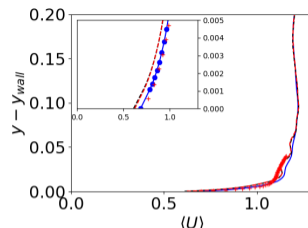
RESULTS, HUMP FLOW. $291 \times 106 \times 16$ CELLS. VELOCITY



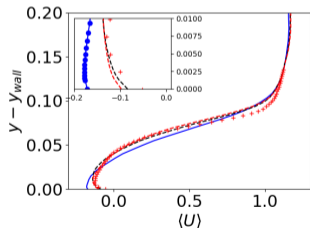
Pressure coefficient.



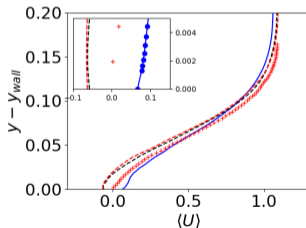
Friction coefficient.



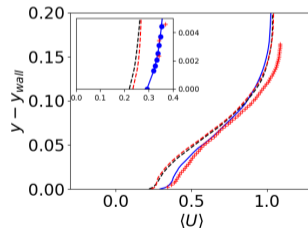
$x = 0.65$.



$x = 0.80$.



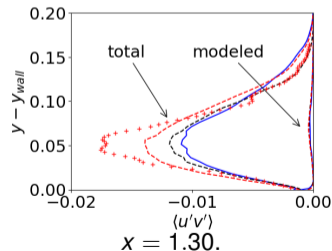
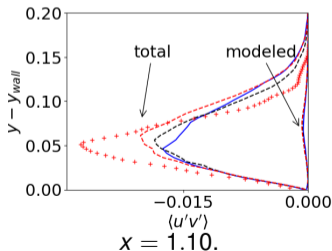
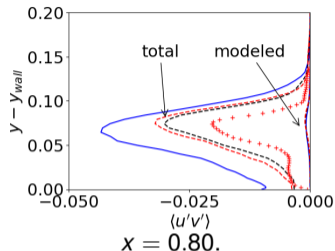
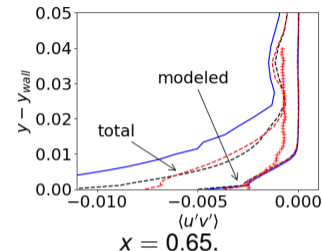
$x = 1.10$.



$x = 1.30$.

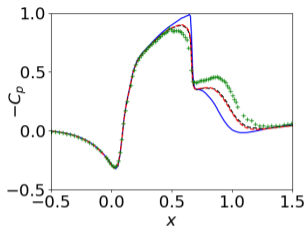
— : KDTree hump data; --- : KDTree diffuser data; --- : Reichardt's law; + : exp.

RESULTS, HUMP FLOW. $291 \times 106 \times 16$ CELLS. SHEAR STRESSES

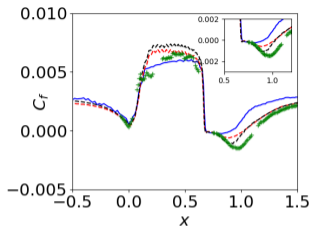


— : KDTree hump data; --- : KDTree diffuser data; - - : Reichardt's law; + : exp.

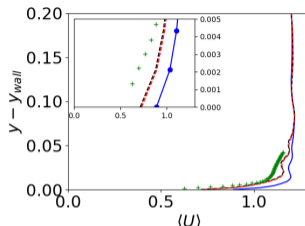
RESULTS, HUMP FLOW. STANDARD WALL FUNCTION MESH, $N_y = 80$



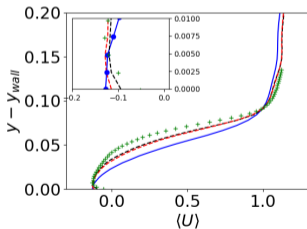
Pressure coefficient.



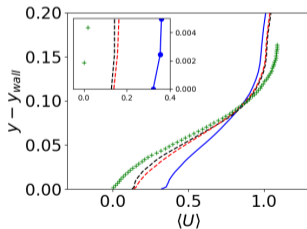
Friction coefficient.



$x = 0.65$.



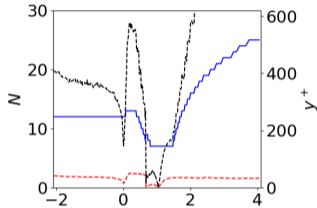
$x = 0.80$.



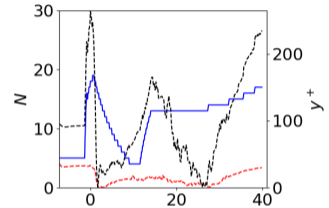
$x = 1.10$.

— : KDTree hump data; --- : KDTree diffuser data; - - : Reichardt's law; + : exp.

URANS/LES INTERFACE.



Hump flow.



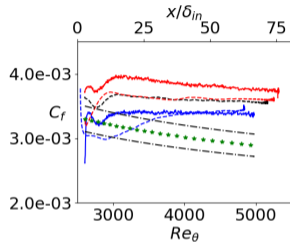
Diffuser flow.

— : Number of cells in the URANS region (left y axis); - - : y^+ of wall-adjacent cells (right y axis).

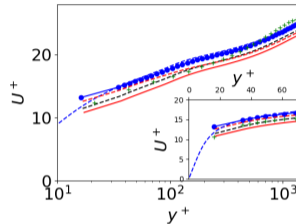
BOUNDARY LAYER FLOW.

- Inlet b.c. taken from a pre-cursor $k - \omega$ simulation at $Re_\theta \simeq 2\,500$
- Grid: $550 \times 90 \times 64$
- Domain: $63 \times 4.6 \times 3.2$.
- Inlet boundary layer thickness: $\delta_{in} = 0.86$
- Inlet k and ε : 2D RANS plus commutation term in k eq. [4, 1].
- Synthetic fluctuations [12, 2] are superimposed on the mean flow

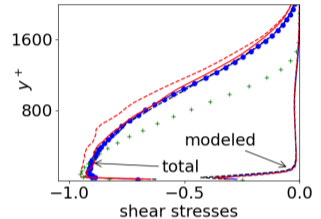
BOUNDARY LAYER FLOW. RESULTS. 3rd CELL.



(A) Friction coefficient.



(B) Mean velocity.



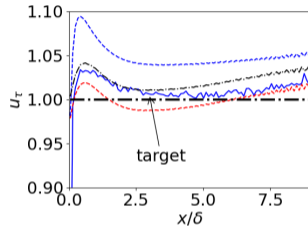
(C) Shear stresses.

u_τ is computed by using U^+ and y^+ at the 4th cell. Velocity and shear stresses are shown at $Re_\theta = 4\,000$. — : **KDTree**, hump flow --- : **KDTree**, diffuser flow data, $K = 5$; — : **KDTree**, diffuser flow data, $K = 1$; - - : Reichardt's wall function; •: cell centers; - - : low-Re IDDES; *: $C_f = 2(1/0.384 \ln(Re_\theta) + 4.127)^{-2}$; - · - : $\pm 6\%$; +: DNS.

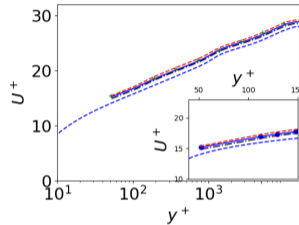
CHANNEL FLOW.

- $Re_\tau = 16\,000$, Inlet-outlet
- Grid: $96 \times 32 \times 32$
- Domain: $9 \times 2 \times 1.6$
- Inlet k and ε : 2D RANS plus commutation term in k eq. [4, 1].
- Synthetic fluctuations [12, 2] are superimposed on the mean flow

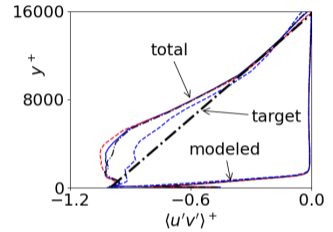
CHANNEL FLOW. RESULTS.



(A) Friction velocity.



(B) Mean velocity.



(C) Shear stress.

Velocity and shear stress are shown at $x/\delta = 6$. — : **KDTree** , hump flow - - : **KDTree** , diffuser flow; - - : low-Re IDDES; — : **KDTree** , hump flow , $K = 5$; - - : Reichardt's wall function; •: cell centers; +: Reichardt's law

CONCLUSIONS

- A new wall function based on **KDTree** (look-up table) has been presented

CONCLUSIONS

- A new wall function based on **KDTree** (look-up table) has been presented
 - Two sets of target data are evaluated: diffuser flow ($\alpha = 15^\circ$) and hump flow.

CONCLUSIONS

- A new wall function based on **KDTree** (look-up table) has been presented
 - Two sets of target data are evaluated: diffuser flow ($\alpha = 15^\circ$) and hump flow.
 - Four flows are used as test cases: diffuser flow ($\alpha = 15^\circ$ and $\alpha + 10^\circ$), hump flow, boundary layer flow and channel flow

CONCLUSIONS

- A new wall function based on **KDTree** (look-up table) has been presented
 - Two sets of target data are evaluated: diffuser flow ($\alpha = 15^\circ$) and hump flow.
 - Four flows are used as test cases: diffuser flow ($\alpha = 15^\circ$ and $\alpha + 10^\circ$), hump flow, boundary layer flow and channel flow
- The diffuser target data set gives in general better results: it's a much simpler, cleaner flow than the hump flow

CONCLUSIONS

- A new wall function based on **KDTree** (look-up table) has been presented
 - Two sets of target data are evaluated: diffuser flow ($\alpha = 15^\circ$) and hump flow.
 - Four flows are used as test cases: diffuser flow ($\alpha = 15^\circ$ and $\alpha + 10^\circ$), hump flow, boundary layer flow and channel flow
- The diffuser target data set gives in general better results: it's a much simpler, cleaner flow than the hump flow
- Much more target data (many more x profiles) are needed in the hump flow than in the diffuser flow


CONCLUSIONS

- A new wall function based on **KDTree** (look-up table) has been presented
 - Two sets of target data are evaluated: diffuser flow ($\alpha = 15^\circ$) and hump flow.
 - Four flows are used as test cases: diffuser flow ($\alpha = 15^\circ$ and $\alpha + 10^\circ$), hump flow, boundary layer flow and channel flow
- The diffuser target data set gives in general better results: it's a much simpler, cleaner flow than the hump flow
- Much more target data (many more x profiles) are needed in the hump flow than in the diffuser flow
- You can download Python scripts [here](#)

REFERENCES

- [1] S. Arvidson, L. Davidson, and S.-H. Peng. Interface methods for grey-area mitigation in turbulence-resolving hybrid RANS-LES. *International Journal of Heat and Fluid Flow*, 73:236–257, 2018.
- [2] M. Carlsson, L. Davidson, S.-H. Peng, and S. Arvidson. Investigation of turbulence injection methods in large eddy simulation using a compressible flow solver. In *AIAA Science and Technology Forum and Exposition, AIAA SciTech Forum*, 2022.
- [3] L. Davidson. Zonal PANS: evaluation of different treatments of the RANS-LES interface. *Journal of Turbulence*, 17(3):274–307, 2016.
- [4] L. Davidson. Two-equation hybrid RANS-LES models: A novel way to treat k and ω at inlets and at embedded interfaces. *Journal of Turbulence*, 18(4):291–315, 2017.
- [5] L. Davidson. pyCALC-LES: a Python code for DNS, LES and Hybrid LES-RANS. Division of Fluid Dynamics, Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, 2021.

REFERENCES

- [6] 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*.
- [7] D. Greenblatt, K. B. Paschal, C.-S. Yao, and J. Harris. A separation control CFD validation test case Part 1: Zero efflux oscillatory blowing. AIAA-2005-0485, 2005.
- [8] D. Greenblatt, K. B. Paschal, C.-S. Yao, J. Harris, N. W. Schaeffler, and A. E. Washburn. A separation control CFD validation test case. Part 1: Baseline & steady suction. AIAA-2004-2220, 2004.
- [9] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas Schön. *Machine Learning: A First Course for Engineers and Scientists*. Cambridge University Press, 2022.

REFERENCES

- [10] 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.
- [11] 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.
- [12] M. Shur, P.R. Spalart, M.K. Strelets, and A.K. Travin. Synthetic turbulence generators for RANS-LES interfaces in zonal simulations of aerodynamic and aeroacoustic problems. *Flow, Turbulence and Combustion*, 93:69–92, 2014.