Computational Fluid Dynamics MSc

Home assignment I.
Pre-study for a vortex flowmeter

Introduction

Your company gives you a task to make a preliminary study for designing a vortex flowmeter with the help of Computational Fluid Dynamics.

The working principle of a vortex flow meter: the fluid flows around a bluff body, downstream of the body a von Karman vortex street develops (periodical vortex shedding). The vortex shedding frequency can be measured and -with the help of the Strouhal number- the frequency can be calculated.

If you would like to learn more about the vortex flowmeter, please read https://mersz.hu/kiadvany/635/info/ (Fluid mechanics measurements lecture notes).

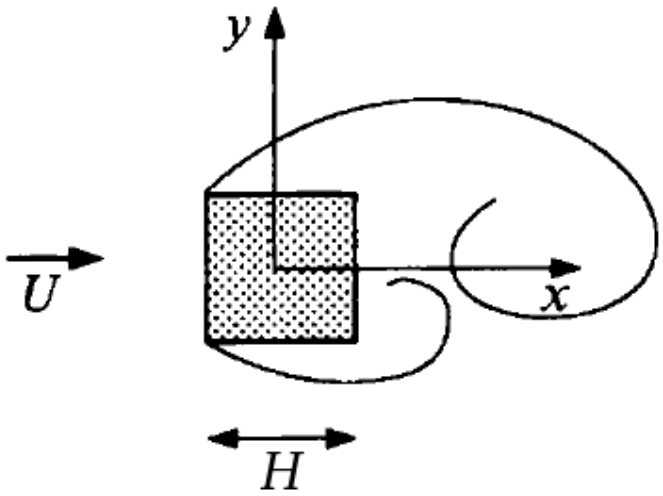


Figure 1. - Von Karman vortex street behind a bluff body

Task

To use CFD as a tool to design, you should be convinced that your software is capable of simulating the case with acceptable error. For validation, vortex shedding past a square cylinder measurements can be used (http://cfd.mace.manchester.ac.uk/ercoftac/doku.php?id=cases:case043).

- Make a 2D model of the validation case.
- Simulate with and without a turbulence model.
- Compare the frequency and the available velocity data.

- Estimate the discretization error of your simulation for the better performing model (laminar or turbulent) and estimate the Strouhal number for an infinitely fine mesh (Richardson extrapolation should be used, if possible).
- Give suggestions for the mesh (appropriate mesh size) and the preferable model (whether to use turbulence model or not).
- Write a technical report about your simulations.

Documentation

You should describe your investigation in your OWN words in the documentation. In the diagrams and figures, your OWN results should be presented. The document and the calculation must be uploaded to the Moodle system.

The documentation must contain the following parts:

- Aim of the simulation
- Introduction
    - Introduce the measurement
        - What was measured, where did it take place and by which equipment
        - Uncertainty of the measured variables
    - Refer to a few papers that made a simulation for this case
        - What type of mesh they used
        - What kind of (turbulence) model they used
- Describe the geometry (with figures)
- Describe the mesh (with figures)
- Describe the physical models
- Results of the simulation with different turbulence model
    - averaged velocity, pressure contour plots
    - time instantaneous velocity contours, where the von Karman Vortex Street can be seen
    - y+ values around the wall of the bluff body
    - graph from which the frequency can be read
    - comparison of velocity plots with the measurement results
- Estimation of discretization error and the Strouhal number to an infinitely fine mesh, if possible
    - comparison of contour plots from different meshes
    - comparison of velocity plot results of the different meshes and the measurement results
    - mesh dependency diagram for the Strouhal number
- Summary
- References

Data for validation

The following data must be used for the validation:

- Strouhal number
- Long-time-averaged data at these locations: $\frac{x}{D} = -0.5;\ 0; 0.5; 1;\ 1.25; 1.5; 2; 3;$

References

Vad, J (2008) – Advanced flow measurements - University lecture note, Műegyetem Kiadó, ISBN: 978 963 420 951 5

Budapest University of Technology and Economics
Faculty of Mechanical Engineering

Computational Fluid Dynamics
I. Homework

Gergő Kelle

November 12, 2023

Contents

1 Introduction

1.1 Aim of the simulation

The aim of the simulation is to replicate and analyze the turbulent near-wake flow around a square cylinder. This study aims to replicate ensemble-averaged flow statistics obtained from an experimental setup that employed two-component laser-Doppler measurements.
This experimental study, conducted using a two-component laser-Doppler velocimetry (LDV) system, provided ensemble-averaged statistics at a constant phase of the flow. The measurements were performed in a closed water channel, and the square cylinder had an edge length of 4 cm (thus $H = 40$ [mm]).

1.2 Reference experiment

As for sumarizing the reference experiment, the flow medium was water, and the experiment was conducted within a rectangular water channel measuring 390 [mm] in width and 560 [mm] in length. A square cylinder as a bluff body with an edge length of 4 cm ($H = 40$ [mm]) was introduced to the flowfield. LDV was employed to capture instantaneous point velocities, and a reference phase was defined based on the pressure signal. The reference velocity, denoted as U, was calculated with a relative error estimate of 2-3%, resulting in a value of approximately $U = 0.535\,[\mathrm{m/s}]$. The vortex shedding frequency was $f = 1.77$ Hz, with a very small margin of error at ± 0.05 Hz.

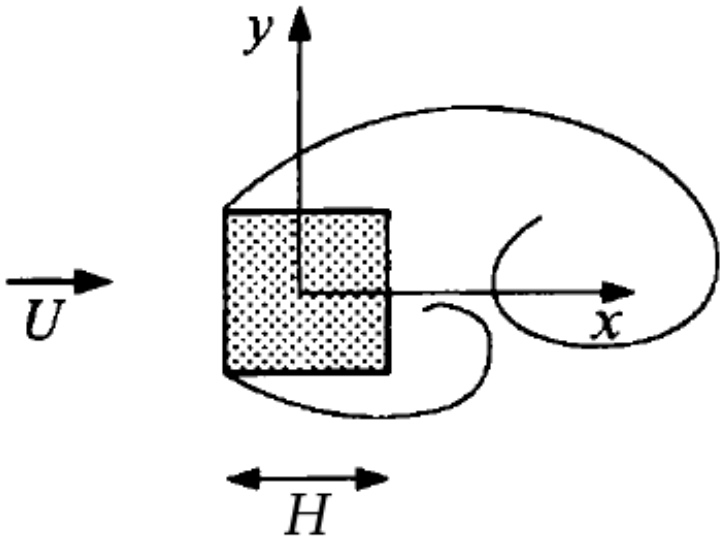


Figure 1: Your Caption Here

Additionally, the Reynolds number (Re) and Strouhal number (St) were derived from fundamental parameters. The Re was computed as 21400, and the Strouhal number (St) was found to be 0.132, with a minimal deviation of ± 0.004. These flow characteristics and turbulence properties forming the basis of my simulation.

Table 1: Summary of Reference Experiment Data

Parameter	Value	Uncertainty
Flow Medium	Water	-
Channel Dimensions	390 mm (width) x 560 mm (length)	-
Cylinder Edge Length (H)	40 mm	-
Reference Velocity (U)	0.535 m/s	2-3%
Vortex Shedding Frequency (f)	1.77 Hz	$\pm$ 0.05 Hz
Reynolds Number (Re)	21400	-
Strouhal Number (St)	0.132	$\pm$ 0.004

2 Pre-processing

2.1 Geometry

Prior to creating the geometry for the simulation, a comprehensive analysis of the preferred meshing strategy was conducted based on relevant literature research. This approach was influenced by a figure from reference [1].

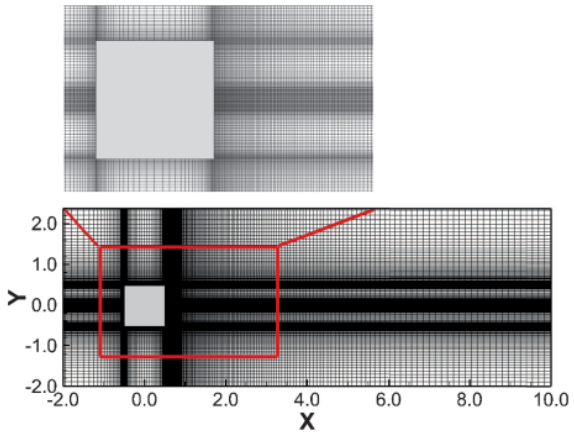


Figure 2: Illustration from Reference [1]

The geometric parameters used in creating the 2D geometry were derived from [2] and visually represented using information from [1]. These parameters are summarized in the following scientific table:

Table 2: Geometric Parameters

Parameter	Value
H	0.04 m
D	0.56 m
L_u	0.32 m
L_D	0.66 m

The information presented in the table is also illustrated in the figure from reference [1].

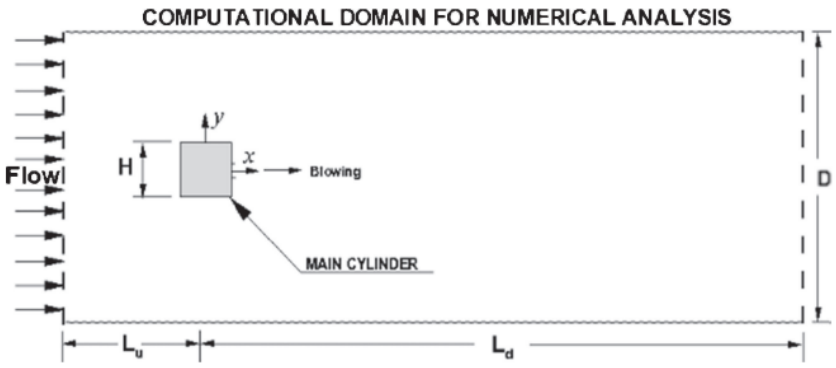


Figure 3: Illustration from Reference [1]

2.2 Discretization - Meshing

The meshing process for simulating the flow around the vortex square cylinder was carried out with a focus on three distinct mesh resolutions: coarse, normal, and fine. This approach allowed for a comprehensive analysis of the impact of mesh density on the simulation results.

Table 3: Mesh Statistics for Different Resolutions

Mesh Resolution	Nodes	Elements
Coarse Mesh	2215	2100
Normal Mesh	4340	4175
Fine Mesh	12532	12240

The meshing strategy involved utilizing edge sizing with the right bias type to create a finer mesh near the edges. This was achieved by setting higher bias factors in the meshing parameters. Additionally, the MultiZone Quad/Tri method was employed to generate a structured mesh, optimizing grid quality and simulation accuracy while keeping the skewness value at an acceptable level.
The mesh structures for each resolution are illustrated in the figures below:

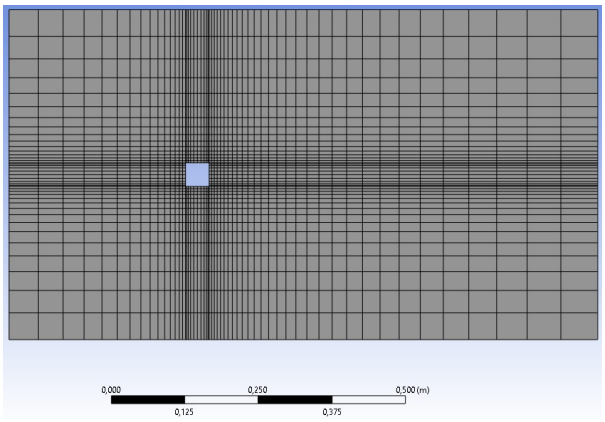


Figure 4: Coarse Mesh (Mesh A)

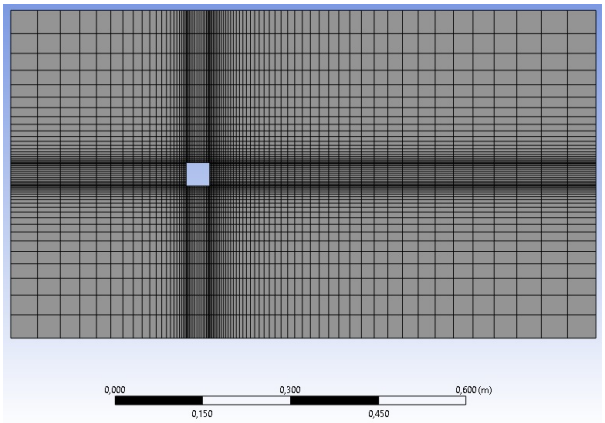


Figure 5: Normal Mesh (Mesh B)

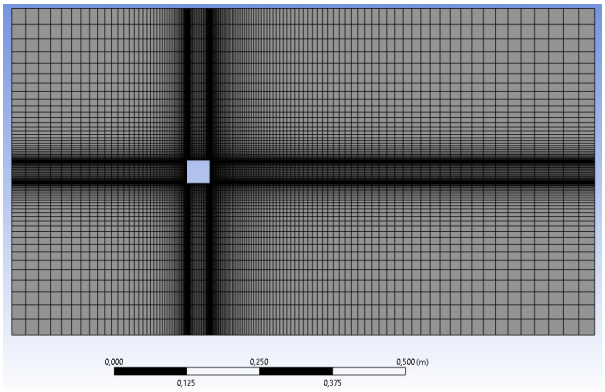


Figure 6: Fine Mesh (Mesh C)

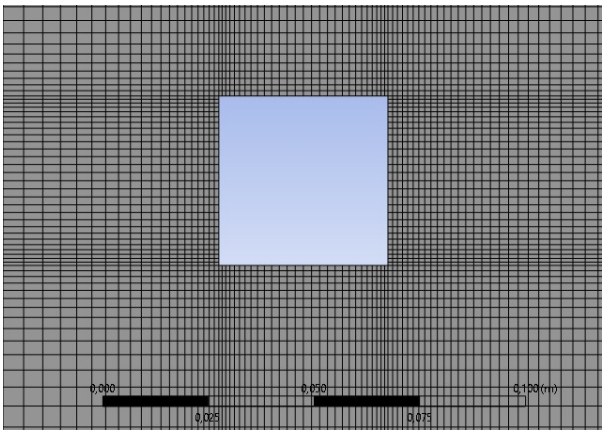


Figure 7: Fine Mesh (Mesh C) - Close-up

2.3 Defining the Properties of the Model

Before conducting the simulations, a series of pre-processing steps were taken to configure the model properties and boundary conditions. The domain was segmented into symmetry (top and bottom), velocity inlet, pressure outlet, wall, and fluid zones. Fluent's automatic boundary condition selection was utilized, ensuring that each zone was appropriately defined for the subsequent simulations.

Two distinct simulation types were conducted, employing different mesh densities: one for laminar flow analysis and another for k-ε turbulence modeling, which considers turbulence effects in the flow. To account for turbulence, the Enhanced Wall Treatment was applied to the boundaries. This treatment enhances wall boundary layer predictions.
For the turbulence parameters at the velocity inlet and pressure outlet, the hydraulic diameter value associated with the bluff body was used. In accordance with reference [2], the velocity at the inlet was set to $U = 0.535$ m/s, representing the known free-stream velocity.

In preparation for post-processing, line elements were introduced at specific locations, with coordinates at $x/H = -0.5, 0, 0.5, 1, 1.25, 1.5, 2, 3$. The origin of the coordinate system was positioned in the left corner of the computational domain. The simulation post-processing was adjusted to compensate for this offset.
Additionally, calculation activities were defined to save *ASCII* files for every second time step, focusing on the lift coefficient and x-velocity along the created line elements. These calculations were performed for both the coarse and fine mesh resolutions. The time step size for the simulations was set to 0.005 seconds, accumulating a total of 2400 time steps. Consequently, the simulation duration extended to 12 seconds to ensure the establishment of a periodic state.

3 Post-processing

Most of the post-processing tasks were carried out with using *Python* code based on gathering *ASCII*, *.out*, and *.h5* files. This task was an intresting job to do in order to keep my coding

Figure 8: Line Elements ("surfaces") for Post-processing

knowledge up-to-date. The whole code were written using Jupyter Notebook and it is available in the 6. section.

3.1 Vortex Shedding Frequency Analysis

The vortex shedding frequency is a critical parameter in characterizing the flow behavior. The dominant vortex shedding frequency was determined for each simulation, representing a crucial aspect of the flow dynamics. To visualize these results, the corresponding figures are provided below:

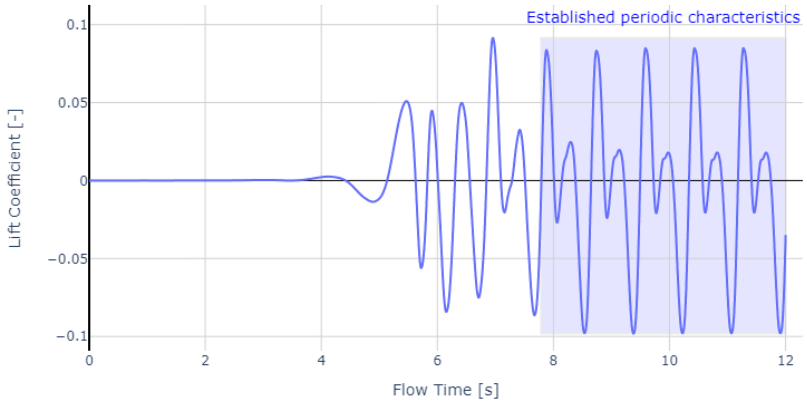


Figure 9: Laminar Vortex Shedding Frequency

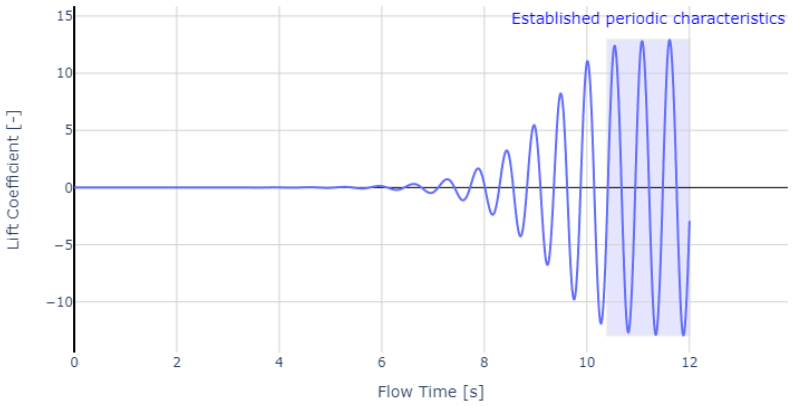


Figure 10: Turbulent Vortex Shedding Frequency (Coarse Mesh)

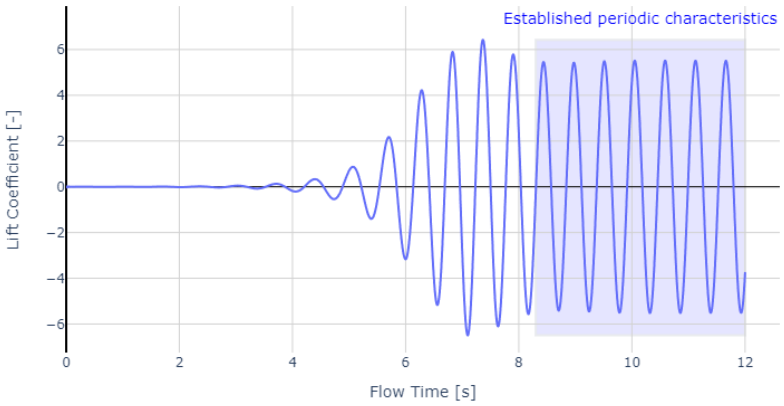


Figure 11: Turbulent Vortex Shedding Frequency (Fine Mesh)

Based on these frequency values and visual representations I was able to determine the beginning of the established peroidic characteristics.The results of the frequency analysis for different simulations are presented in the table below:

Table 4: Vortex Shedding Frequency Analysis

Simulation Type	**Initial time (t_0)**	**Frequency (f)**
Laminar	7.79 s	1.18 Hz
Turbulent (Coarse Mesh)	10.4 s	1.87 Hz
Turbulent (Fine Mesh)	8.3 s	1.85 Hz

Based on the results tubulent model with fine mesh gave us the best solution, thus for further calculations I will rather focus on turbulent model. According to [1] The shedding frequency, f, was estimated as 1.77 ± 0.05 Hz from a direct analysis of the low-pass-filtered pressure signal with

the peak-finding algorithm used in the phase-sorting procedure. Comparing the results to $f_{t,fine}$ is almost within the error range so for later calculation I will consider it's result as the most accurate.

3.2 Velocity and Pressure

3.2.1 Laminar Vortex Shedding

For the laminar simulation, a positive amplitude time step within the established periodic characteristics range was identified. At this specific time step, the velocity contour plot was generated, illustrating only its u component (Figure 12). Additionally, the pressure contour plot was produced to visualize the pressure distribution at the identified time step (Figure 13).

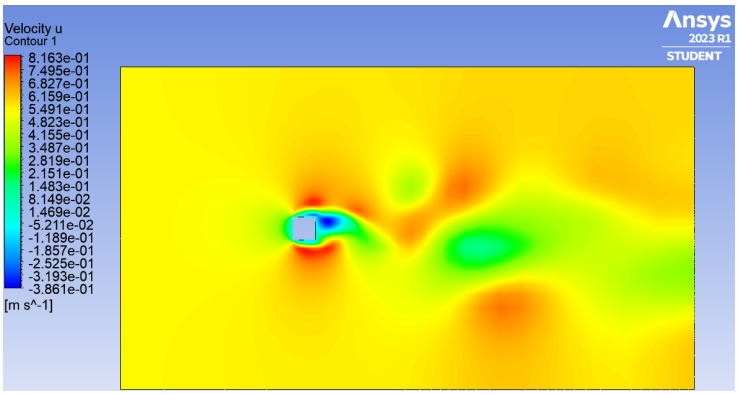


Figure 12: Velocity Contour Plot for Laminar Vortex Shedding

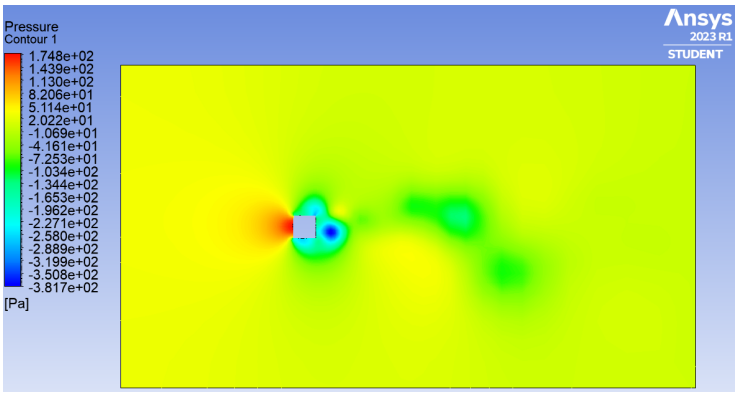
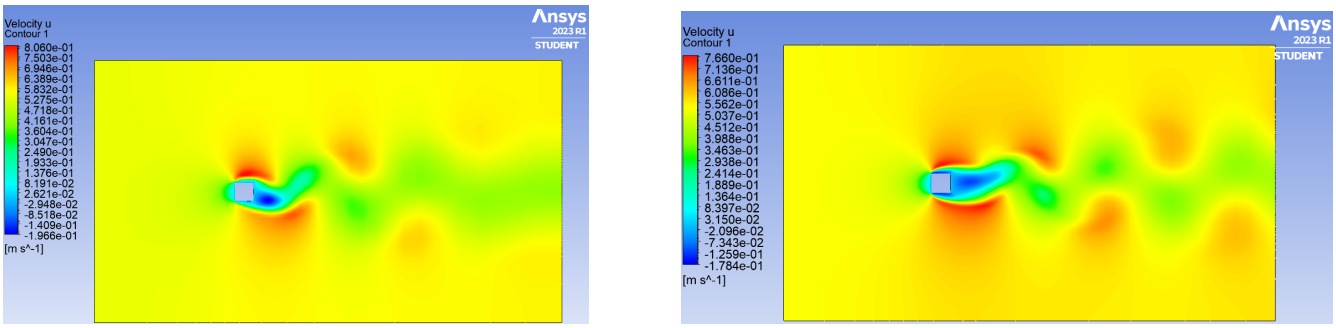


Figure 13: Pressure Contour Plot for Laminar Vortex Shedding

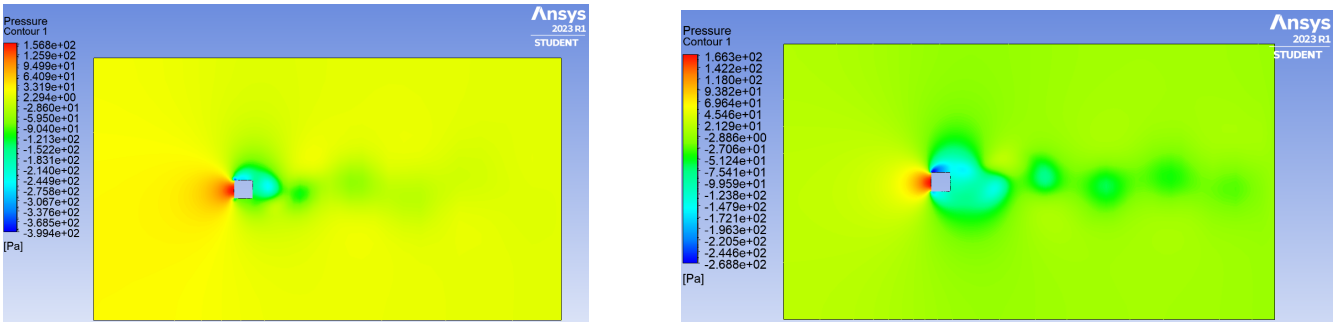
3.2.2 Turbulent Vortex Shedding

For the turbulent simulation, positive amplitude time steps within the established periodic characteristics range were determined for both coarse and fine meshes. At these time steps, the velocity contour plots, emphasizing the u component, were generated. Additionally, the pressure contour plots were created to visualize the pressure distribution during turbulent vortex shedding.
As for comparison i found streamline representation the most effective way which is shown in Figure 16.



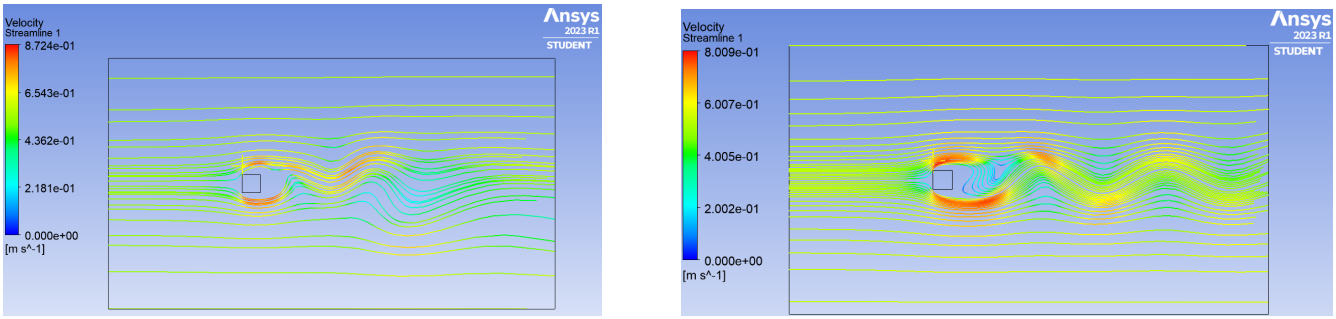
(a) Coarse Mesh (b) Fine Mesh

Figure 14: Velocity Contour Plots for Turbulent Vortex Shedding



(a) Coarse Mesh (b) Fine Mesh

Figure 15: Pressure Contour Plots for Turbulent Vortex Shedding



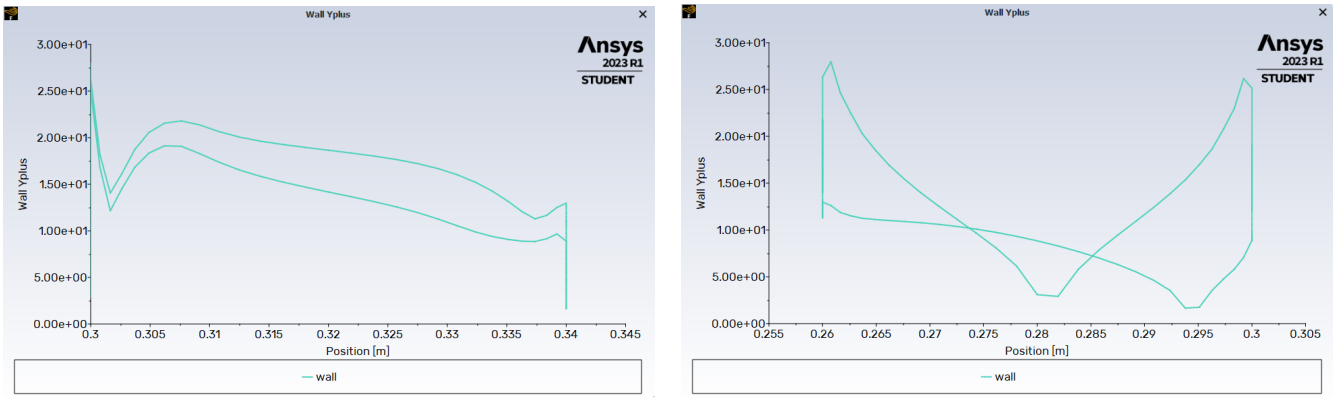
(a) Laminar (b) Turbulent

Figure 16: Streamline plots for velocity with different mathematical models

3.3 y^+ value

In CFD, y^+ is a dimensionless parameter representing the distance of the first computational grid point from the wall normalized by the molecular viscous length scale. It is essential for accurate modeling of near-wall turbulent flows, with low y^+ values indicating proper grid resolution within the viscous sublayer.

The y^+ values for the top, bottom, left, and right walls are visualized in Figures 17a and 17b.



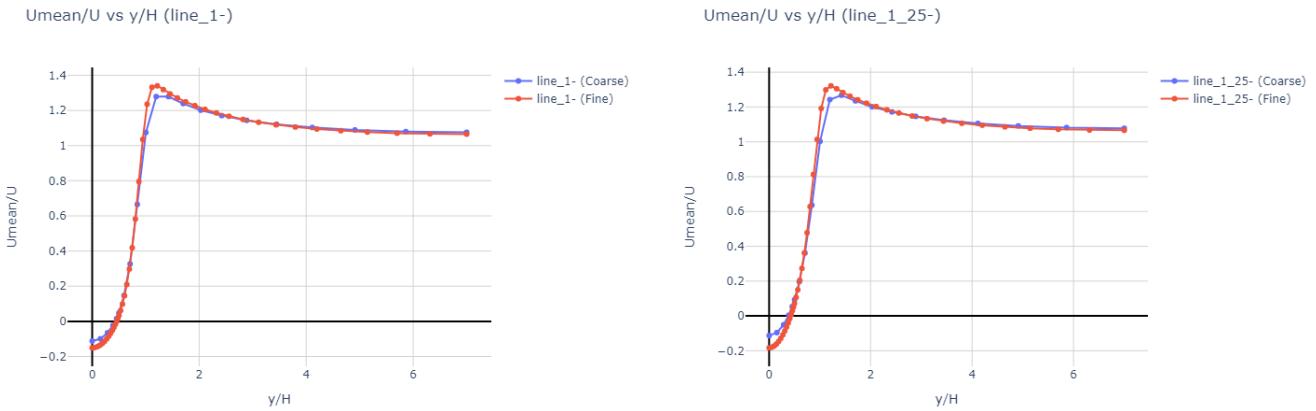
(a) Top and Bottom Walls (b) Left and Right Walls

Figure 17: y^+ Distributions on Different Walls

3.4 Comparison

As for comparison I've used the available experimental data from [2] and post-process every file that I written out along the simulations for every second timestep and for every line element that are placed. This resulted in a post-processing task with $\approx$ 20000 ASCII file.

> *i: The solution and the coding can be viewed at the end of the document appendix.*

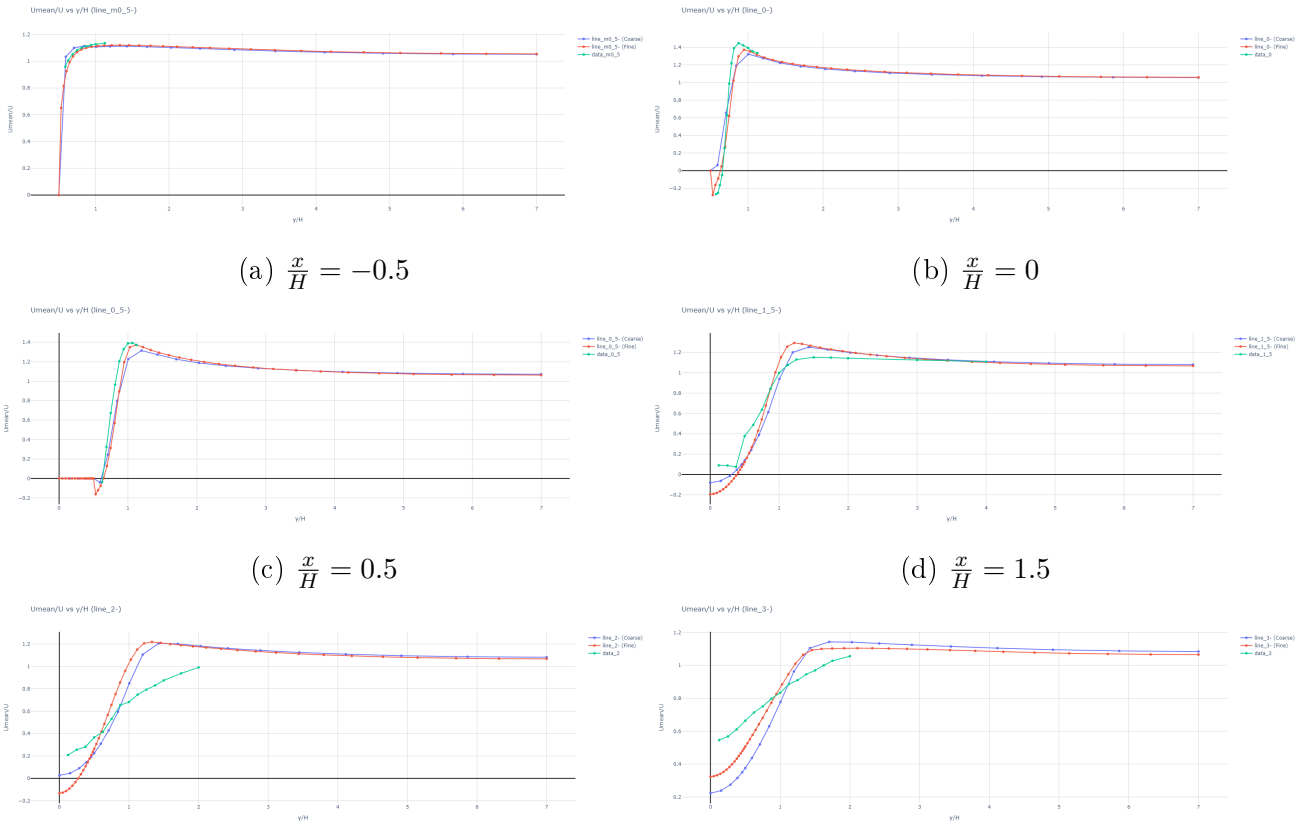


(a) $\frac{x}{H} = 1$ (b) $\frac{x}{H} = 1.25$

Figure 18: Resulting $\frac{U_{mean}}{U}$ values as a function of $\frac{y}{H}$

Since i didn't find experimental data I plotted only the resulting data for the coarse and the fine mesh for the turbulent model as Figure 18 shows. The red line and markers stands for the fine mesh, and the blue line and markers stands for the coarse mesh.

As for the rest of the line element data I was able to compare my simulation results with the experimental results which is showed in Figure 19. On that Figure the green line and markers shows the experimental data.



(a) $\frac{x}{H} = -0.5$ (b) $\frac{x}{H} = 0$

(c) $\frac{x}{H} = 0.5$ (d) $\frac{x}{H} = 1.5$

(e) $\frac{x}{H} = 2$ (f) $\frac{x}{H} = 3$

Figure 19: Resulting $\frac{U_{mean}}{U}$ values as a function of $\frac{y}{H}$ for different $\frac{x}{H}$

3.5 Estimation of discretization error

In this section I will check the mesh dependancy of the Strouhal number and use Richardson extrapolation method described in [3] to calculate the value for infinitely fine mesh. Since we are working with finite element methods infinitely fine mesh could never exist thus we use mathermatical extrapolation methods to predict the value towhich the solution converge.

$$St_i = \frac{f \cdot H}{U_{mean}} \tag{1}$$

$$St_\infty = St_3 + \frac{St_3 - St_2}{\left(\frac{h_3}{h_2}\right) - 1} = 0.13822 \tag{2}$$

Where

Strouhal number	:	St_i
Vortex shedding frequency	:	f_i
Characteristic length	:	H
Mesh size (elements)	:	h_i

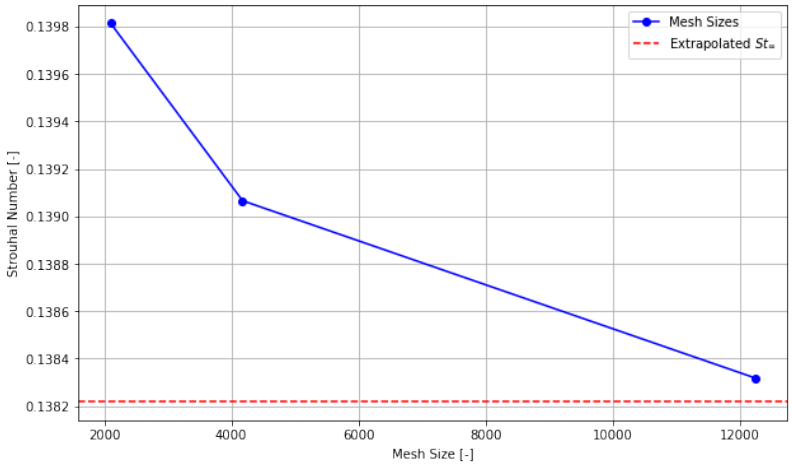


Figure 20: Mesh Dependency of Strouhal Number

Furthermore it is important to point out that the fine mesh turbulent model simulation gave a strouhal number of 0.13831 while the experiment worked with a Strouhal number of 0.132 ± 0.004. My result is slightly over the reference error interval (same with the frequency value), but close enough to call the simulation similar to the experiment.

4 Sumarry

In summary I denoted that the fine mesh with turbulent k - ε model describe the most accurate way the experiment while as Figure 18a - 19 shows the coarse and fine mesh had very little difference speaking of results while the fine mesh $6x$ more elements.

As for reflection to the aim of the simulation the replication of the experiment can be called as an achived result since the simulation results showed similarities to the experimental data given by [2].

This task was a great challange to do since I've decided to do almost every post-processing task with *Python* importing the resulting *.out* and *ASCII* files.

5 Reference

References

[1] Saha, Arun K. and Shrivastava, Ankit (Year). *Suppression of Vortex Shedding around a Square Cylinder using Blowing. Sadhana - Academy Proceedings in Engineering Sciences*. URL: https://www.ias.ac.in/article/fulltext/sadh/040/03/0769-0785.

[2] Lyn, D. A. and Rodi, W. *Vortex Shedding Past Square Cylinder Experiment*. URL: http://cfd.mace.manchester.ac.uk/ercoftac/doku.php?id=cases:case043.

[3] NASA Glenn Research Center. *Spatial Convergence Tutorial*. URL: https://www.grc.nasa.gov/www/wind/valid/tutorial/spatconv.html.

6 Appendix

CFD_I_HW

November 10, 2023

0.1 Calculating vortex shredding frequency

0.1.1 Plots

[17]:
```
##Coarse mesh - laminar

import pandas as pd
import plotly.express as px

# Define the full path to the "lif-force-1-rfile.out" file
data_file_path = r'C:\Users\Kelle Gergő\Untitled␣
↪Folder\result2\laminar\lift-force-2-rfile.out'

# Read the data from the file
data = pd.read_csv(data_file_path, delim_whitespace=True)

# Create a line plot using Plotly
fig = px.line(data, x='flow-time', y='lif-force-1')
fig.update_xaxes(title_text="Flow Time [s]", showgrid=True,␣
↪gridcolor="lightgrey", zeroline=True, zerolinecolor="black", zerolinewidth=2)
fig.update_yaxes(title_text="Lift Coefficient [-]", showgrid=True,␣
↪gridcolor="lightgrey", zeroline=True, zerolinecolor="black", zerolinewidth=1)

# Change the background to white
fig.update_layout(
    paper_bgcolor="white",
    plot_bgcolor="white"
)

# Add a low-opacity purple background
fig.add_shape(
    type="rect",
    x0=7.79,  # Start of the background
    x1=data['flow-time'].max(),  # End of the background (you can adjust this)
    y0=data['lif-force-1'].min(),
    y1=data['lif-force-1'].max(),
    fillcolor="blue",
    opacity=0.1,
```

```
    layer="below",
)

# Add text "SAMPLE" in the center vertically
fig.add_annotation(
    x=7.79 + (data['flow-time'].max() - 7.79) / 2,
    y=(data['lif-force-1'].max()*1.15),
    text="Established periodic characteristics",
    showarrow=False,
    font=dict(size=14, color="blue"),
)


fig.update_layout(
    width=1600/2,
    height=900/2
)
# Show the Plotly plot
fig.show()
fig.write_image('plots/laminar_lift_coefficient.png')
```

[15]:
```
import pandas as pd
import plotly.express as px

# Define the full path to the "lif-force-1-rfile.out" file
data_file_path = r'C:\Users\Kelle Gergő\Untitled␣
 ↪Folder\result2\turbulent_fine\lif-force-1-rfile.out'

# Read the data from the file
data = pd.read_csv(data_file_path, delim_whitespace=True)

# Create a line plot using Plotly
fig = px.line(data, x='flow-time', y='lif-force-1')
fig.update_xaxes(title_text="Flow Time [s]", showgrid=True,␣
 ↪gridcolor="lightgrey", zeroline=True, zerolinecolor="black", zerolinewidth=2)
fig.update_yaxes(title_text="Lift Coefficient [-]", showgrid=True,␣
 ↪gridcolor="lightgrey", zeroline=True, zerolinecolor="black", zerolinewidth=1)

# Change the background to white
fig.update_layout(
    paper_bgcolor="white",
    plot_bgcolor="white"
)

# Add a low-opacity blue background
fig.add_shape(
    type="rect",
```

```
    x0=8.3,  # Start of the background
    x1=data['flow-time'].max(),  # End of the background (you can adjust this)
    y0=data['lif-force-1'].min(),
    y1=data['lif-force-1'].max(),
    fillcolor="blue",
    opacity=0.1,
    layer="below",
)

# Add text "Established periodic characteristics" in the center vertically
fig.add_annotation(
    x=8.3 + (data['flow-time'].max() - 8.3) / 2,
    y=(data['lif-force-1'].max()*1.15),
    text="Established periodic characteristics",
    showarrow=False,
    font=dict(size=14, color="blue"),
)

# Show the Plotly plot
fig.update_layout(
    width=1600/2,
    height=900/2
)
fig.show()
fig.write_image('plots/turb_lift_coef_coarse.png')
```

[16]:
```
##Coarse mesh

import pandas as pd
import plotly.express as px

# Define the full path to the "lif-force-1-rfile.out" file
data_file_path = r'C:\Users\Kelle Gergő\Untitled␣
 ↪Folder\result2\turbulent\lif-force-1-rfile.out'

# Read the data from the file
data = pd.read_csv(data_file_path, delim_whitespace=True)

# Create a line plot using Plotly
fig = px.line(data, x='flow-time', y='lif-force-1')
fig.update_xaxes(title_text="Flow Time [s]", showgrid=True,␣
 ↪gridcolor="lightgrey", zeroline=True, zerolinecolor="black", zerolinewidth=2)
fig.update_yaxes(title_text="Lift Coefficient [-]", showgrid=True,␣
 ↪gridcolor="lightgrey", zeroline=True, zerolinecolor="black", zerolinewidth=1)

# Change the background to white
fig.update_layout(
```

```
    paper_bgcolor="white",
    plot_bgcolor="white"
)

# Add a low-opacity purple background
fig.add_shape(
    type="rect",
    x0=10.4,  # Start of the background
    x1=data['flow-time'].max(),  # End of the background (you can adjust this)
    y0=data['lif-force-1'].min(),
    y1=data['lif-force-1'].max(),
    fillcolor="blue",
    opacity=0.1,
    layer="below",
)

# Add text "SAMPLE" in the center vertically
fig.add_annotation(
    x=10.4 + (data['flow-time'].max() - 10.4) / 2,
    y=(data['lif-force-1'].max()*1.15),
    text="Established periodic characteristics",
    showarrow=False,
    font=dict(size=14, color="blue"),
)

# Show the Plotly plot
fig.update_layout(
    width=1600/2,
    height=900/2
)
fig.show()
fig.write_image('plots/turb_lift_coef_fine.png')
```

0.1.2 Frequency calculation

[4]:
```
import pandas as pd
## Laminar - Coarse mesh

# Define the full path to the "lif-force-1-rfile.out" file
data_file_path1 = r'C:\Users\Kelle Gergő\Untitled␣
 ↪Folder\result2\laminar\lift-force-2-rfile.out'

# Read the data from the file
data1 = pd.read_csv(data_file_path1, delim_whitespace=True)

# Filter data after flow time > 8.3
filtered_data1 = data1[data1['flow-time'] > 7.79]
```

```
# Find local maxima in the lif-force-1 column
local_maxima1 = filtered_data1['lif-force-1'][
    (filtered_data1['lif-force-1'] > filtered_data1['lif-force-1'].shift(1)) &
    (filtered_data1['lif-force-1'] > filtered_data1['lif-force-1'].shift(-1))
]

# Check if there are at least two local maxima
if len(local_maxima1) >= 2:
    # Calculate the time difference between the two local maxima
    time_difference1 = (local_maxima1.index[6] - local_maxima1.index[4])*0.005

    # Calculate the frequency as the inverse of the time difference
    frequency1 = 1 / time_difference1

    print(f"Vortex shedding frequency (Laminar - Coarse mesh) t > 7.79:"+'\n'+␣
 ↪f"{frequency1:.2f}" + "Hz")
else:
    print("Not enough local maxima found.")


## Turbulent - Coarse mesh

# Define the full path to the "lif-force-1-rfile.out" file
data_file_path2 = r'C:\Users\Kelle Gergő\Untitled␣
 ↪Folder\result2\turbulent\lif-force-1-rfile.out'

# Read the data from the file
data2 = pd.read_csv(data_file_path2, delim_whitespace=True)

# Filter data after flow time > 8.3
filtered_data2 = data2[data2['flow-time'] > 10.4]

# Find local maxima in the lif-force-1 column
local_maxima2 = filtered_data2['lif-force-1'][
    (filtered_data2['lif-force-1'] > filtered_data2['lif-force-1'].shift(1)) &
    (filtered_data2['lif-force-1'] > filtered_data2['lif-force-1'].shift(-1))
]

# Check if there are at least two local maxima
if len(local_maxima2) >= 2:
    # Calculate the time difference between the two local maxima
    time_difference2 = (local_maxima2.index[1] - local_maxima2.index[0])*0.005

    # Calculate the frequency as the inverse of the time difference
    frequency2 = 1 / time_difference2
```

```
    print("Vortex shedding frequency (Turbulent - Coarse mesh) t > 10.4:"+'\n'+␣
 ↪f"{frequency2:.2f}" + "Hz")
else:
    print("Not enough local maxima found.")

## Turbulent - Fine mesh


# Define the full path to the "lif-force-1-rfile.out" file
data_file_path3 = r'C:\Users\Kelle Gergő\Untitled␣
 ↪Folder\result2\turbulent_fine\lif-force-1-rfile.out'

# Read the data from the file
data3 = pd.read_csv(data_file_path3, delim_whitespace=True)

# Filter data after flow time > 8.3
filtered_data3 = data3[data3['flow-time'] > 8.3]

# Find local maxima in the lif-force-1 column
local_maxima3 = filtered_data3['lif-force-1'][
    (filtered_data3['lif-force-1'] > filtered_data3['lif-force-1'].shift(1)) &
    (filtered_data3['lif-force-1'] > filtered_data3['lif-force-1'].shift(-1))
]

# Check if there are at least two local maxima
if len(local_maxima3) >= 2:
    # Calculate the time difference between the two local maxima
    time_difference3 = (local_maxima3.index[5] - local_maxima3.index[4])*0.005

    # Calculate the frequency as the inverse of the time difference
    frequency3 = 1 / time_difference3

    print("Vortex shedding frequency (Turbulent - Fine mesh) t > 8.3"+'\n'+␣
 ↪f"{frequency3:.2f}" + "Hz")
else:
    print("Not enough local maxima found.")
```

```
Vortex shedding frequency (Laminar - Coarse mesh) t > 7.79:
1.18Hz
Vortex shedding frequency (Turbulent - Coarse mesh) t > 10.4:
1.87Hz
Vortex shedding frequency (Turbulent - Fine mesh) t > 8.3
1.85Hz
```

0.2 Comparing the results to Long-time-averaged mean velocity

0.2.1 Data

[8]:
```
import pandas as pd
from io import StringIO

data_m0_5_str = """ x/H    y/H    Umean/U  Ufluc/U skewn. flatn.    fff
  -0.5000  0.5500   0.875   0.093   0.263   2.194   1.000
  -0.5000  0.5875   0.959   0.097   0.267   2.217   1.000
  -0.5000  0.6250   1.006   0.101   0.198   2.745   1.000
  -0.5000  0.6875   1.052   0.097   0.211   2.082   1.003
  -0.5000  0.7500   1.084   0.093   0.171   2.313   1.003
  -0.5000  0.8125   1.101   0.090   0.124   2.355   1.003
  -0.5000  0.8750   1.112   0.084   0.108   2.512   1.003
  -0.5000  0.9375   1.121   0.079   0.062   2.609   1.003
  -0.5000  1.0000   1.127   0.075  -0.001   2.767   1.003
  -0.5000  1.1250   1.135   0.065   0.104   2.078   1.003
"""

data_0_str = """ x/H    y/H    Umean/U  Ufluc/U skewn. flatn.    fff
   0.0000  0.5500  -0.293   0.166  -0.307   4.219   0.030
   0.0000  0.5750  -0.267   0.176  -0.089   3.811   0.053
   0.0000  0.6000  -0.254   0.179  -0.020   4.292   0.062
   0.0000  0.6250  -0.166   0.232   0.629   4.555   0.198
   0.0000  0.6550  -0.050   0.305   0.762   4.252   0.376
   0.0000  0.6875   0.254   0.456   0.530   3.094   0.685
   0.0000  0.7200   0.630   0.553   0.228   2.419   0.879
   0.0000  0.7500   0.987   0.540  -0.213   2.333   0.965
   0.0000  0.7800   1.219   0.447  -0.711   2.924   0.992
   0.0000  0.8125   1.389   0.314  -1.223   5.012   0.998
   0.0000  0.8750   1.447   0.155  -0.639   5.042   1.003
   0.0000  0.9375   1.422   0.120   0.034   2.072   1.003
   0.0000  1.0000   1.393   0.108   0.030   1.978   1.003
   0.0000  1.0625   1.353   0.095   0.004   1.995   1.003
   0.0000  1.1250   1.336   0.088  -0.005   2.086   1.000
"""

data_0_5_str = """ x/H    y/H    Umean/U  Ufluc/U skewn. flatn.    fff
   0.5000  0.5750  -0.189   0.295   0.864   4.695   0.210
   0.5000  0.6250  -0.037   0.407   0.987   3.695   0.360
   0.5000  0.6875   0.325   0.589   0.451   2.138   0.621
   0.5000  0.7500   0.673   0.634  -0.082   1.798   0.803
   0.5000  0.8125   0.964   0.564  -0.571   2.256   0.927
   0.5000  0.8750   1.204   0.430  -1.224   3.933   0.985
   0.5000  0.9375   1.329   0.297  -1.873   7.170   0.996
   0.5000  1.0000   1.389   0.183  -2.609  15.307   1.002
   0.5000  1.0625   1.393   0.120  -1.724  14.546   1.000
```

```
   0.5000  1.1250   1.372   0.099  -1.481  18.692   1.003
"""

data_1_5_str = """x/H     y/H   Umean/U  Vmean/U  u'/U    v'/U   u'v'/U^2  ␣
 ↪fff_u  fff_v
   1.5000   0.0000   0.082  -0.032   0.393   0.815  1.74e-02   0.61   0.48
   1.5000   0.1250   0.090  -0.092   0.406   0.796 -3.05e-02   0.61   0.45
   1.5000   0.2500   0.088  -0.050   0.387   0.837  6.85e-03   0.63   0.48
   1.5000   0.3750   0.075  -0.069   0.393   0.826  2.78e-03   0.60   0.47
   1.5000   0.5000   0.378  -0.226   0.501   0.669 -1.63e-01   0.75   0.36
   1.5000   0.6250   0.488  -0.254   0.525   0.606 -1.70e-01   0.80   0.34
   1.5000   0.7500   0.636  -0.256   0.493   0.514 -1.44e-01   0.87   0.29
   1.5000   0.8750   0.845  -0.252   0.434   0.389 -1.02e-01   0.94   0.22
   1.5000   1.0000   0.998  -0.232   0.350   0.295 -5.49e-02   0.98   0.17
   1.5000   1.1250   1.075  -0.187   0.286   0.243 -2.86e-02   0.99   0.17
   1.5000   1.2500   1.129  -0.153   0.232   0.204 -1.11e-02   1.00   0.21
   1.5000   1.5000   1.151  -0.097   0.151   0.155  9.85e-05   1.00   0.28
   1.5000   1.7500   1.148  -0.069   0.110   0.114  8.66e-04   1.00   0.29
   1.5000   2.0000   1.142  -0.054   0.084   0.090  5.62e-04   1.00   0.30
   1.5000   3.0000   1.125  -0.002   0.039   0.045  1.01e-04   1.01   0.47
   1.5000   4.0000   1.108  -0.002   0.022   0.034 -1.57e-05   1.01   0.49
"""

data_2_str = """x/H     y/H   Umean/U  Vmean/U  u'/U    v'/U   u'v'/U^2   fff_u␣
 ↪ fff_v
   2.0000   0.0000   0.178  -0.761   0.393   0.417 -3.77e-02   0.67   0.06
   2.0000   0.1250   0.209  -0.764   0.385   0.383 -3.74e-02   0.72   0.05
   2.0000   0.2500   0.256  -0.720   0.406   0.381 -6.67e-02   0.72   0.06
   2.0000   0.3750   0.282  -0.686   0.400   0.320 -6.78e-02   0.76   0.05
   2.0000   0.5000   0.366  -0.594   0.374   0.338 -6.50e-02   0.81   0.06
   2.0000   0.6250   0.415  -0.538   0.383   0.282 -5.52e-02   0.85   0.06
   2.0000   0.7500   0.531  -0.450   0.333   0.269 -4.86e-02   0.91   0.07
   2.0000   0.8750   0.654  -0.374   0.277   0.200 -2.99e-02   0.96   0.06
   2.0000   1.0000   0.682  -0.293   0.243   0.189 -2.15e-02   0.98   0.05
   2.0000   1.1250   0.748  -0.198   0.247   0.193 -2.35e-02   0.98   0.11
   2.0000   1.2500   0.793  -0.163   0.176   0.127 -1.18e-02   0.99   0.09
   2.0000   1.3750   0.830  -0.101   0.174   0.138 -1.54e-02   0.99   0.14
   2.0000   1.5000   0.875  -0.075   0.135   0.103 -7.16e-03   1.00   0.17
   2.0000   1.7500   0.938  -0.022   0.080   0.082 -3.08e-03   1.00   0.35
   2.0000   2.0000   0.991  -0.021   0.058   0.054 -1.24e-03   1.00   0.35
"""

data_3_str = """x/H     y/H   Umean/U  Vmean/U  u'/U    v'/U   u'v'/U^2   fff_u␣
 ↪ fff_v
   3.0000   0.0000   0.550  -0.062   0.284   0.710  5.42e-03   0.96   0.47
   3.0000   0.1250   0.546  -0.056   0.290   0.705 -8.04e-03   0.95   0.47
   3.0000   0.2500   0.568  -0.058   0.293   0.682 -2.10e-02   0.96   0.46
```

```
   3.0000   0.3750   0.611  -0.054   0.292   0.656 -3.34e-02   0.97   0.45
   3.0000   0.5000   0.664  -0.064   0.290   0.628 -4.33e-02   0.98   0.44
   3.0000   0.6250   0.714  -0.062   0.293   0.594 -5.00e-02   0.99   0.42
   3.0000   0.7500   0.751  -0.045   0.301   0.570 -5.03e-02   0.99   0.42
   3.0000   0.8750   0.798  -0.041   0.295   0.521 -5.21e-02   0.99   0.41
   3.0000   1.0000   0.834  -0.030   0.295   0.482 -5.14e-02   1.00   0.40
   3.0000   1.1250   0.886  -0.049   0.293   0.430 -4.68e-02   0.99   0.36
   3.0000   1.2500   0.910  -0.028   0.284   0.400 -4.40e-02   1.00   0.36
   3.0000   1.3750   0.946  -0.034   0.269   0.355 -3.88e-02   1.00   0.33
   3.0000   1.5000   0.970  -0.026   0.260   0.316 -3.24e-02   1.00   0.33
   3.0000   1.6250   1.000  -0.024   0.234   0.277 -2.51e-02   1.00   0.34
   3.0000   1.7500   1.028  -0.021   0.206   0.234 -1.51e-02   1.00   0.36
   3.0000   2.0000   1.056  -0.015   0.161   0.174 -6.67e-03   1.00   0.40
"""



data_m0_5 = pd.read_csv(StringIO(data_m0_5_str), delim_whitespace=True)
data_0 = pd.read_csv(StringIO(data_0_str), delim_whitespace=True)
data_0_5 = pd.read_csv(StringIO(data_0_5_str), delim_whitespace=True)
data_1_5 = pd.read_csv(StringIO(data_1_5_str), delim_whitespace=True)
data_2 = pd.read_csv(StringIO(data_2_str), delim_whitespace=True)
data_3 = pd.read_csv(StringIO(data_3_str), delim_whitespace=True)

vector_m0_5_column_2 = data_m0_5.iloc[1:, 1].values
vector_m0_5_column_3 = data_m0_5.iloc[1:, 2].values

vector_0_column_2 = data_0.iloc[1:, 1].values
vector_0_column_3 = data_0.iloc[1:, 2].values

vector_0_5_column_2 = data_0_5.iloc[1:, 1].values
vector_0_5_column_3 = data_0_5.iloc[1:, 2].values

vector_1_5_column_2 = data_1_5.iloc[1:, 1].values
vector_1_5_column_3 = data_1_5.iloc[1:, 2].values

vector_2_column_2 = data_2.iloc[1:, 1].values
vector_2_column_3 = data_2.iloc[1:, 2].values

vector_3_column_2 = data_3.iloc[1:, 1].values
vector_3_column_3 = data_3.iloc[1:, 2].values
```

0.2.2 Comparison

[9]:
```
import os
import pandas as pd
import plotly.graph_objects as go

# Define the list of file prefixes
file_prefixes = ['line_m0_5-', 'line_0-', 'line_0_5-', 'line_1-', 'line_1_25-',␣
↪'line_1_5-', 'line_2-', 'line_3-']

# Directory where your coarse and fine data files are located
data_files_dir_coarse = r'C:\Users\Kelle Gergő\Untitled␣
↪Folder\result2\turbulent'
data_files_dir_fine = r'C:\Users\Kelle Gergő\Untitled␣
↪Folder\result2\turbulent_fine'

# Initialize empty DataFrames to store the data for each file type
final_dfs_coarse = {}
final_dfs_fine = {}

# Loop through each file prefix
for prefix in file_prefixes:
    final_df_coarse = pd.DataFrame(columns=['nodenumber', 'x-coordinate',␣
↪'y-coordinate', 'x-velocity'])
    final_df_fine = pd.DataFrame(columns=['nodenumber', 'x-coordinate',␣
↪'y-coordinate', 'x-velocity'])

    # Loop through the desired range of file numbers
    for file_number in range(2, 2401, 2):
        filename_coarse = f'{prefix}{file_number:04d}'
        file_path_coarse = os.path.join(data_files_dir_coarse, filename_coarse)

        filename_fine = f'{prefix}{file_number:04d}'
        file_path_fine = os.path.join(data_files_dir_fine, filename_fine)

        if os.path.isfile(file_path_coarse):
            with open(file_path_coarse, 'r') as file:
                lines = file.readlines()
                nodenumber = []
                x_coordinate = []
                y_coordinate = []
                x_velocity = []

                # Skip the header line
                data_lines = lines[1:]

                for line in data_lines:
```

```
                    parts = line.split()
                    if len(parts) == 4:
                        nodenumber.append(int(parts[0]))
                        x_coordinate.append(float(parts[1]))
                        y_coordinate.append(float(parts[2]))
                        x_velocity.append(float(parts[3]))

                df_coarse = pd.DataFrame({'nodenumber': nodenumber,␣
↪'x-coordinate': x_coordinate, 'y-coordinate': y_coordinate, 'x-velocity':␣
↪x_velocity})
                final_df_coarse = pd.concat([final_df_coarse, df_coarse],␣
↪ignore_index=True)

        if os.path.isfile(file_path_fine):
            with open(file_path_fine, 'r') as file:
                lines = file.readlines()
                nodenumber = []
                x_coordinate = []
                y_coordinate = []
                x_velocity = []

                # Skip the header line
                data_lines = lines[1:]

                for line in data_lines:
                    parts = line.split()
                    if len(parts) == 4:
                        nodenumber.append(int(parts[0]))
                        x_coordinate.append(float(parts[1]))
                        y_coordinate.append(float(parts[2]))
                        x_velocity.append(float(parts[3]))

                df_fine = pd.DataFrame({'nodenumber': nodenumber,␣
↪'x-coordinate': x_coordinate, 'y-coordinate': y_coordinate, 'x-velocity':␣
↪x_velocity})
                final_df_fine = pd.concat([final_df_fine, df_fine],␣
↪ignore_index=True)

    final_dfs_coarse[prefix] = final_df_coarse
    final_dfs_fine[prefix] = final_df_fine

# Create a list to store the combined figures
combined_figures = []

# Create separate sets of average_y_divided_by_H values for coarse and fine␣
↪meshes
H = 0.04
```

```
for prefix in file_prefixes:
    final_df_coarse = final_dfs_coarse[prefix]
    final_df_fine = final_dfs_fine[prefix]

    if prefix in ['line_0_5-', 'line_1-', 'line_1_25-', 'line_1_5-', 'line_2-',␣
↪'line_3-']:
        offset = 0.28  # Adjust the offset for specific prefixes
    else:
        offset = 0.28

    average_y_coordinate_coarse = final_df_coarse.
↪groupby('nodenumber')['y-coordinate'].mean() - offset
    average_y_divided_by_H_coarse = average_y_coordinate_coarse / H

    average_y_coordinate_fine = final_df_fine.
↪groupby('nodenumber')['y-coordinate'].mean() - offset
    average_y_divided_by_H_fine = average_y_coordinate_fine / H

    mean_vel_experiment = 0.535
    average_x_velocity_coarse = final_df_coarse.loc[12464:, 'x-velocity'].
↪groupby(final_df_coarse['nodenumber']).mean()
    average_x_velocity_fine = final_df_fine.loc[3500:, 'x-velocity'].
↪groupby(final_df_fine['nodenumber']).mean()

    # Create a figure for the current file type
    fig = go.Figure(layout=dict(
        plot_bgcolor="white",
        xaxis=dict(showgrid=True, gridcolor="lightgrey", zerolinecolor="black"),
        yaxis=dict(showgrid=True, gridcolor="lightgrey", zerolinecolor="black")
    ))

    # Add a scatter plot with markers and lines for the current file type␣
↪(Coarse)
    fig.add_trace(go.Scatter(x=average_y_divided_by_H_coarse,␣
↪y=average_x_velocity_coarse / mean_vel_experiment,
                             mode='lines+markers', name=f'{prefix} (Coarse)'))

    # Add a scatter plot with markers and lines for the current file type (Fine)
    fig.add_trace(go.Scatter(x=average_y_divided_by_H_fine,␣
↪y=average_x_velocity_fine / mean_vel_experiment,
                             mode='lines+markers', name=f'{prefix} (Fine)'))

    # Add the vector data to the plot for data_m0_5
    if prefix == 'line_m0_5-':
        fig.add_trace(go.Scatter(x=vector_m0_5_column_2, y=vector_m0_5_column_3,
```

```
                                mode='lines+markers', name='data_m0_5'))

    # Add the vector data to the plot for data_0
    if prefix == 'line_0-':
        fig.add_trace(go.Scatter(x=vector_0_column_2, y=vector_0_column_3,
                                mode='lines+markers', name='data_0'))

    # Add the vector data to the plot for data_0_5
    if prefix == 'line_0_5-':
        fig.add_trace(go.Scatter(x=vector_0_5_column_2, y=vector_0_5_column_3,
                                mode='lines+markers', name='data_0_5'))

    # Add the vector data to the plot for data_1_5
    if prefix == 'line_1_5-':
        fig.add_trace(go.Scatter(x=vector_1_5_column_2, y=vector_1_5_column_3,
                                mode='lines+markers', name='data_1_5'))

    # Add the vector data to the plot for data_2
    if prefix == 'line_2-':
        fig.add_trace(go.Scatter(x=vector_2_column_2, y=vector_2_column_3,
                                mode='lines+markers', name='data_2'))

    # Add the vector data to the plot for data_3
    if prefix == 'line_3-':
        fig.add_trace(go.Scatter(x=vector_3_column_2, y=vector_3_column_3,
                                mode='lines+markers', name='data_3'))


    # Set the title and axis labels
    fig.update_layout(
        title=f'Umean/U vs y/H ({prefix})',
        xaxis_title='y/H',
        yaxis_title='Umean/U',
        showlegend=True  # Add a legend
    )

    # Add the current figure to the list of combined figures
    combined_figures.append(fig)

# Show each combined Plotly plot
for fig in combined_figures:
    fig.show()

# Create a "plots" folder if it doesn't exist
plots_folder = 'plots'
if not os.path.exists(plots_folder):
    os.mkdir(plots_folder)
```

```
# Loop through the combined figures and save them as .png files
for i, fig in enumerate(combined_figures):
    filename = os.path.join(plots_folder, f'plot_{i + 1}.png')
    fig.write_image(filename)
    print(f'Plot saved as {filename}')
```

```
Plot saved as plots\plot_1.png
Plot saved as plots\plot_2.png
Plot saved as plots\plot_3.png
Plot saved as plots\plot_4.png
Plot saved as plots\plot_5.png
Plot saved as plots\plot_6.png
Plot saved as plots\plot_7.png
Plot saved as plots\plot_8.png
```

[ ]: