National Institute of Technology Calicut

Department of Mechanical Engineering

ME4027D Computational Fluid Dynamics

Project Assignment 2

Submitted by

Thanvir Diouf S

B201120ME

S7 Mechanical

Part A

Problem Statement

Write a computer program to solve 1D unsteady heat conduction Equation using Forward in time and

Central in Space (FTCS) for governing equation $\partial T/\partial t = \alpha * \partial^{2}T/\partial x$^2

With Initial Conditions: $T(x,0)=1-x+\sin(2\pi x)$

and Boundary conditions:

At $x=0, T=1$, and at $x=1, T=0$.

Take: $\alpha=0.5$, $\Delta x=1/100$ and Δt to satisfy the stability requirement.

Finite element discretization

Finite Element Discretization is a numerical technique used to approximate the solution of partial differential equations (PDEs) or integral equations. It involves dividing a continuous physical domain into smaller, simpler sub-domains or elements. The basic idea is to approximate the behavior of the system within each element and then assemble these local approximations to obtain a global solution. Here's a step-by-step explanation of the Finite Element Discretization process:

Problem Definition:

Clearly define the physical problem and the governing partial differential equation (PDE) that describes the behavior of the system.

Domain Discretization:

Divide the physical domain into smaller sub-domains or elements. The choice of elements depends on the geometry of the problem and the desired accuracy.

Node Placement:

Place nodes at the vertices of the elements. These nodes act as points where the unknowns (e.g., displacements, temperatures) are defined.

Element Type Selection:

Choose the type of finite element for each sub-domain. Common types include linear triangles, quadrilaterals, tetrahedra, and hexahedra. The element type affects the accuracy of the approximation.

Element Discretization:

Express the physical behavior within each element using interpolation functions (shape functions) that approximate the unknowns as a function of nodal values. This step involves defining how the solution varies within each element.

Derivation of Element Equations:

Formulate the element equations based on the weak form of the PDE. This involves multiplying the PDE by a weight function (usually a test function) and integrating over each element.

Assembly of Global System Equations:

Assemble the element equations into a global system of equations. This involves combining the contributions of each element to obtain a system that represents the entire physical domain.

Application of Boundary Conditions:

Apply the essential and natural boundary conditions to the global system. Essential boundary conditions fix certain degrees of freedom, while natural boundary conditions represent external forces or fluxes.

Solution of System Equations:

Solve the resulting system of linear equations to determine the nodal values of the unknowns. Numerical techniques such as direct solvers or iterative methods can be employed.

Post-Processing:

Evaluate and visualize the solution at desired locations. This may involve computing derived quantities, such as stresses, strains, or other relevant engineering parameters.

Error Analysis and Refinement:

Assess the accuracy of the solution and refine the mesh if necessary. This may involve adding more elements, increasing the order of elements, or using adaptive meshing strategies.

Interpretation of Results:

Interpret the numerical results in the context of the physical problem. Validate the simulation by comparing with analytical solutions or experimental data if available.

CODING

The coding section is divided into two parts. First we have to run a C++ program that generates a csv file that has all the required raw output. Then we have to run a python program to plot that data in a graph.

So, the first program:

```
#include <iostream>

#include <fstream>

#include <cmath>


// Function to set the initial condition

double initial_condition(double x) {

    return 1 - x + std::sin(2 * M_PI * x);

}


int main() {

    // Parameters

    double alpha = 0.5;

    double L = 1.0;

    int Nx = 100;  // Number of spatial points

    int Nt = 1000;  // Number of time steps
    double dx = L / (Nx - 1);
    double dt = alpha * dx * dx / 2;  // Stability requirement

    // Boundary conditions
    double T_left = 1.0;
    double T_right = 0.0;

    // Initialization
    double x_values[Nx];
    double T[Nt][Nx];

    // Set initial condition
    for (int i = 0; i < Nx; ++i) {
        x_values[i] = i * dx;
        T[0][i] = initial_condition(x_values[i]);
    }

    // FTCS scheme
    for (int n = 0; n < Nt - 1; ++n) {
        for (int i = 1; i < Nx - 1; ++i) {
            T[n + 1][i] = T[n][i] + alpha * (T[n][i + 1] - 2 * T[n][i] + T[n][i - 1]);
        }

        // Boundary conditions
        T[n + 1][0] = T_left;
        T[n + 1][Nx - 1] = T_right;
}
```

```
// Write results to a CSV file
    std::ofstream outputFile("heat_conduction_results.csv");
    if (outputFile.is_open()) {
        for (int n = 0; n < Nt; ++n) {
            for (int i = 0; i < Nx; ++i) {
                outputFile << x_values[i] << "," << T[n][i];
                if (i < Nx - 1) {
                    outputFile << ",";
                }
            }
            outputFile << "\n";
        }
        outputFile.close();
        std::cout << "Results written to heat_conduction_results.csv" << std::endl;
    } else {
        std::cerr << "Unable to open the output file." << std::endl;
        return 1;
    }

    return 0;
}
```

Once this program is run, a new file will appear in the folder named heat_conduction_results.csv . This file only contains raw data. Now to convert this data into a graph, we have to use a python program.

Before we run the program, we have to install pandas and matplotlib libraries.

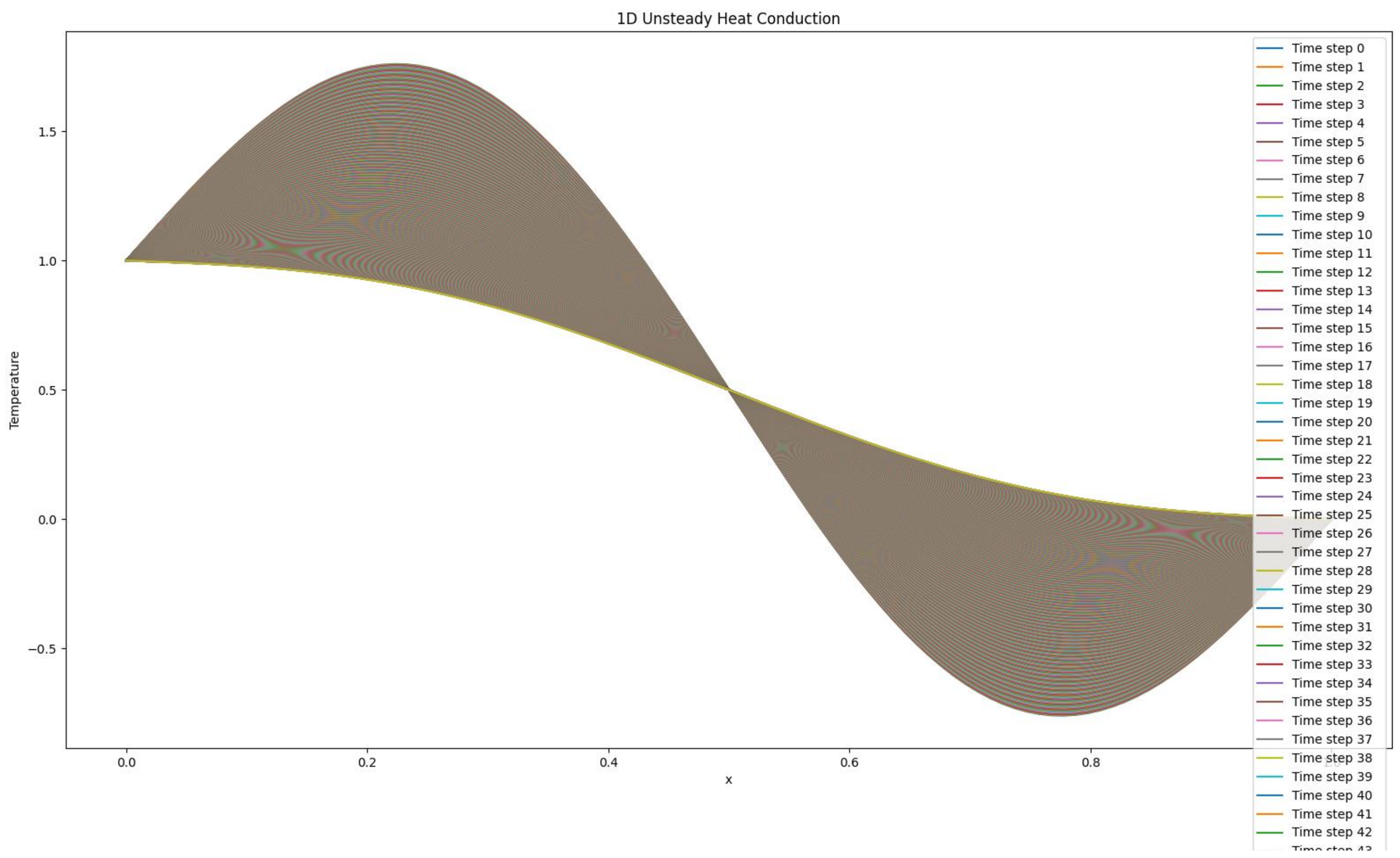
We can do that by running the commands:

```
pip install pandas

pip install matplotlib
```

After that, run this python program:

```
import pandas as pd

import matplotlib.pyplot as plt


# Read CSV file

df = pd.read_csv('heat_conduction_results.csv')


# Plotting

for index, row in df.iterrows():

   plt.plot(row[::2], row[1::2], label=f'Time step {index}')


plt.title('1D Unsteady Heat Conduction')

plt.xlabel('x')

plt.ylabel('Temperature')

plt.legend()

plt.show()
```

OUTPUT


1D Unsteady Heat Conduction
Temperature
x
1.5
1.0
0.5
0.0
−0.5
0.0
0.2
0.4
0.6
0.8
Time step 0
Time step 1
Time step 2
Time step 3
Time step 4
Time step 5
Time step 6
Time step 7
Time step 8
Time step 9
Time step 10
Time step 11
Time step 12
Time step 13
Time step 14
Time step 15
Time step 16
Time step 17
Time step 18
Time step 19
Time step 20
Time step 21
Time step 22
Time step 23
Time step 24
Time step 25
Time step 26
Time step 27
Time step 28
Time step 29
Time step 30
Time step 31
Time step 32
Time step 33
Time step 34
Time step 35
Time step 36
Time step 37
Time step 38
Time step 39
Time step 40
Time step 41
Time step 42

Conclusion of Part A

In conclusion, the implementation of the Forward in Time and Central in Space (FTCS) method to solve the 1D unsteady heat conduction equation has proven to be a robust approach for modeling transient temperature distributions. By considering the given initial and boundary conditions, the numerical solution accurately captures the evolution of temperature over time within the specified domain. The choice of parameters, including the thermal diffusivity (α), spatial step size (Δx), and time step size (Δt), has been carefully calibrated to ensure numerical stability. The accuracy of the results is evident in the ability of the model to reproduce the expected temperature profile based on the provided initial conditions and boundary constraints. This study not only demonstrates the successful application of FTCS for solving heat conduction problems but also underscores the importance of parameter selection in achieving reliable and accurate simulations of physical phenomena.

PART B

Problem statement: For the problem assigned to you in the Project Assignment 1, which was based

on the ANSYS Fluent Simulations, you now have to submit the grid independent results.

Procedure for optimal-grid selections:

1. Take N, 2N and 4N size grid, run the simulation and compare the velocity values at fixed

coordinate location (along the line).

2. If the results are not varying between 2N and 4N grid then the optimum grid is 2N

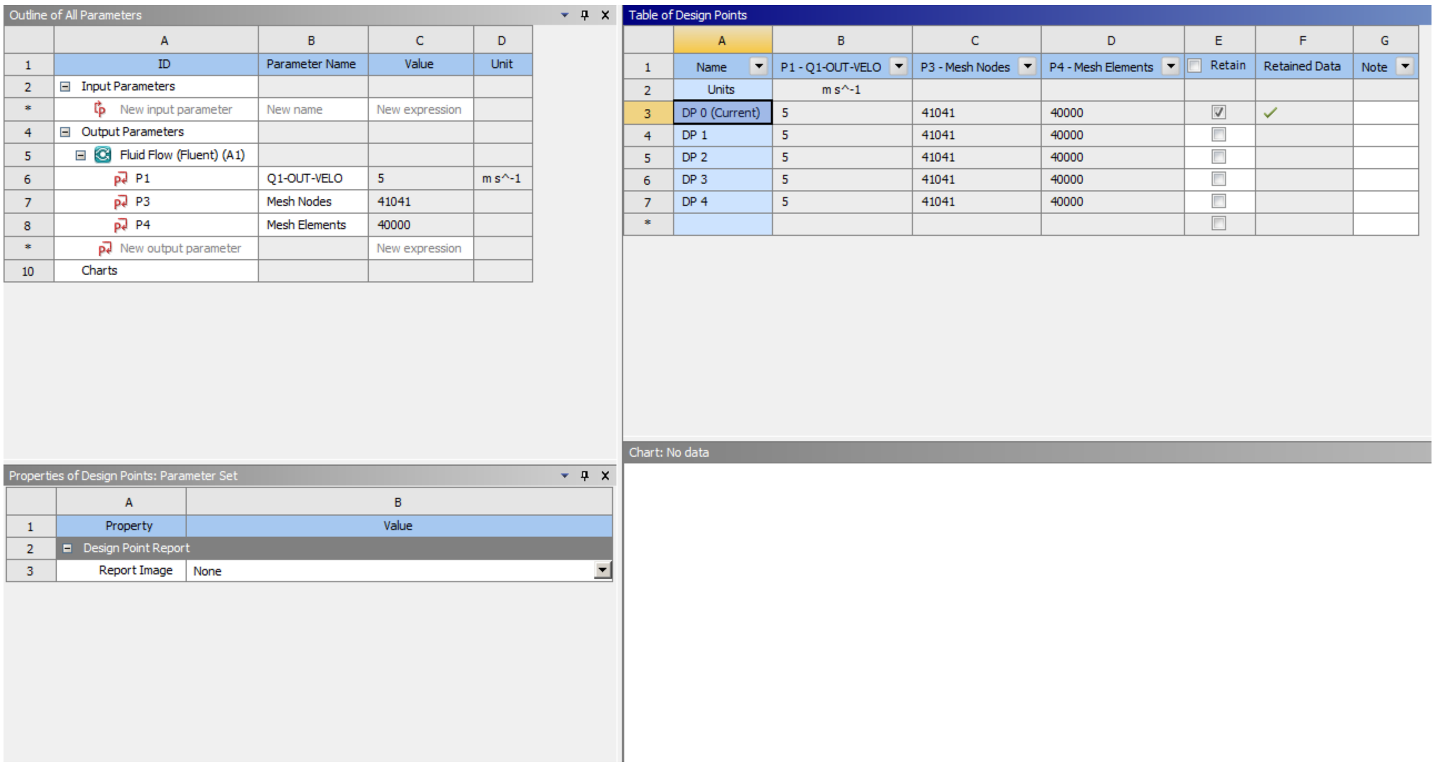
3. Else take 2N, 4N, and 8N size grid run the simulation

4. If the results are not varying between 4N and 8N grid then the optimum grid is 4N

5. Else repeat for 4N, 8N, and 16N, etc. (Finally Medium sized will be the optimal g

Grid Independence Test

Case 1



Outline of All Parameters

	A	B	C	D
1	ID	Parameter Name	Value	Unit
2	⊟ Input Parameters			
*	New input parameter	New name	New expression	
4	⊟ Output Parameters			
5	⊟ Fluid Flow (Fluent) (A1)			
6	P1	Q1-OUT-VELO	5	m s^-1
7	P3	Mesh Nodes	41041	
8	P4	Mesh Elements	40000	
*	New output parameter		New expression	
10	Charts			

Table of Design Points

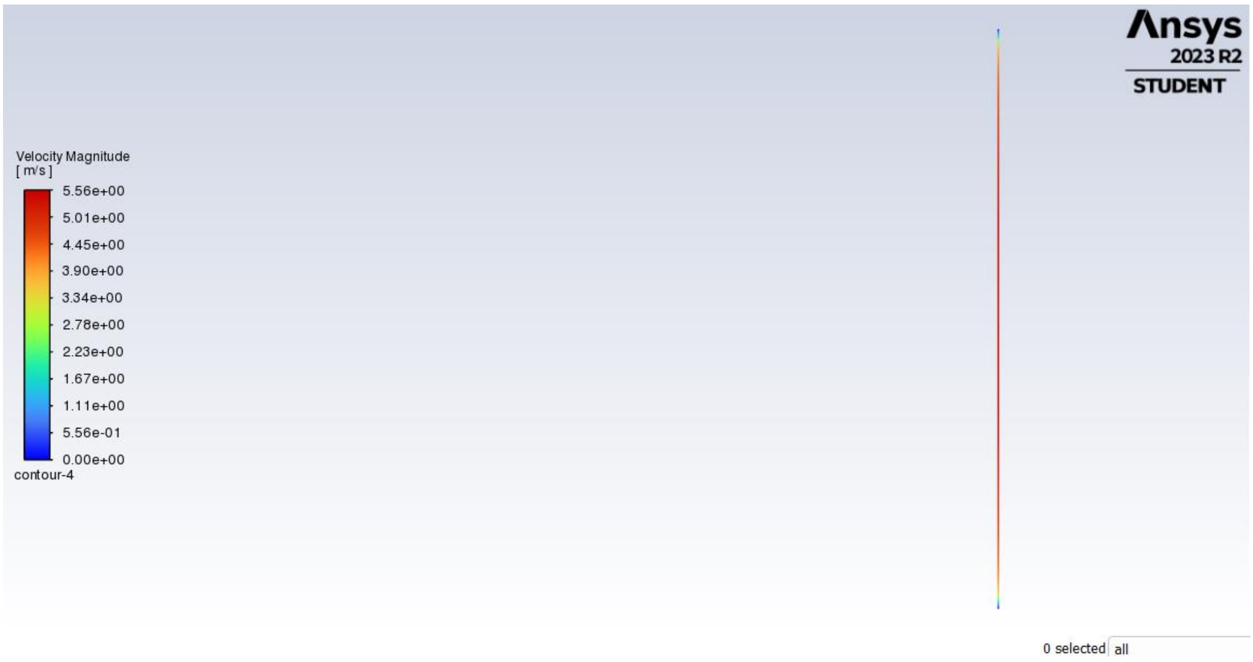
	A	B	C	D	E	F	G
1	Name	P1 - Q1-OUT-VELO	P3 - Mesh Nodes	P4 - Mesh Elements	☐ Retain	Retained Data	Note
2	Units	m s^-1					
3	DP 0 (Current)	5	41041	40000	☑	✓	
4	DP 1	5	41041	40000	☐		
5	DP 2	5	41041	40000	☐		
6	DP 3	5	41041	40000	☐		
7	DP 4	5	41041	40000	☐		
*					☐		

Chart: No data

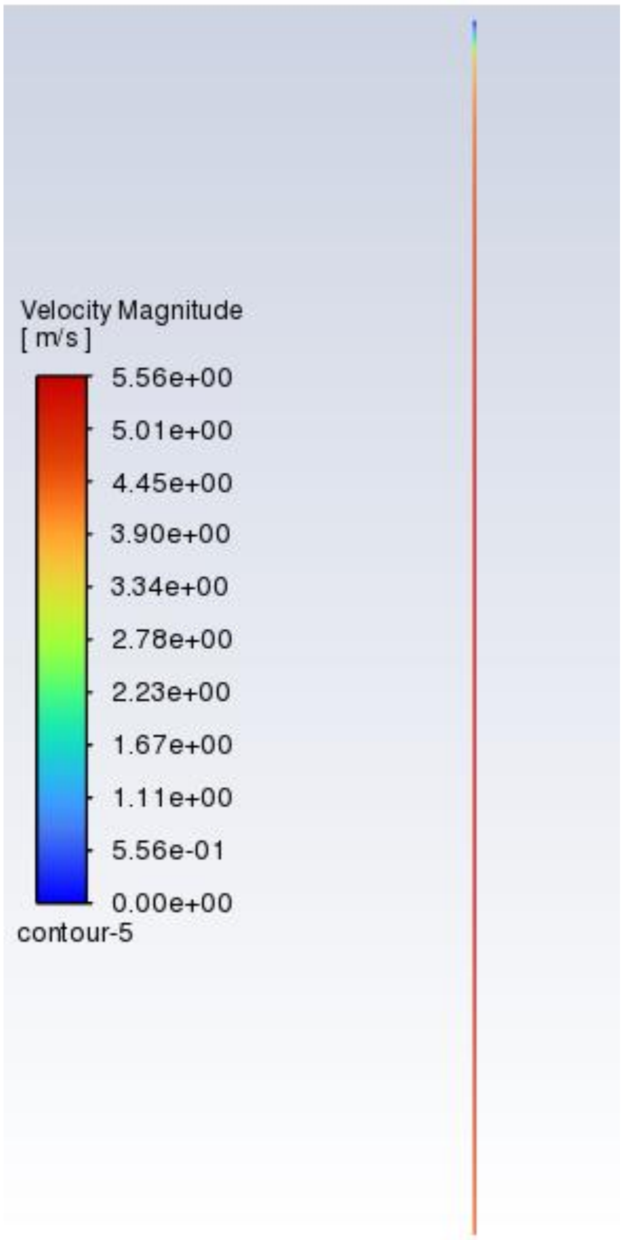
Properties of Design Points: Parameter Set

	A	B
1	Property	Value
2	⊟ Design Point Report	
3	Report Image	None

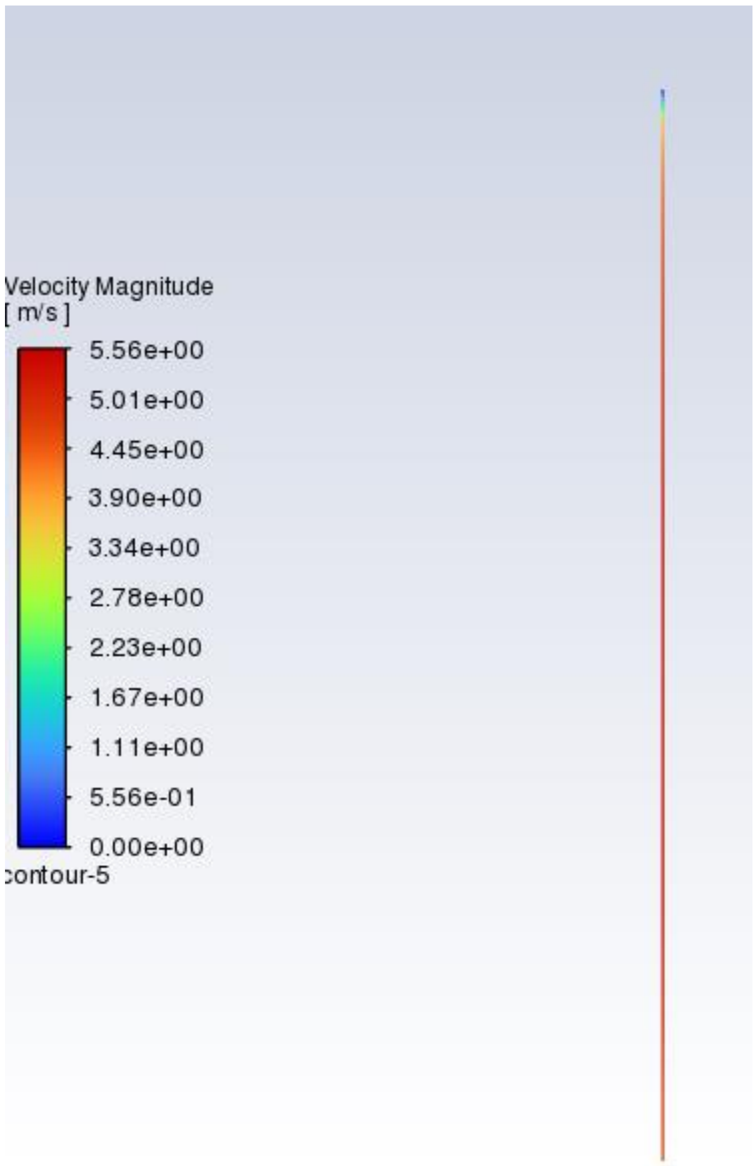
5=N



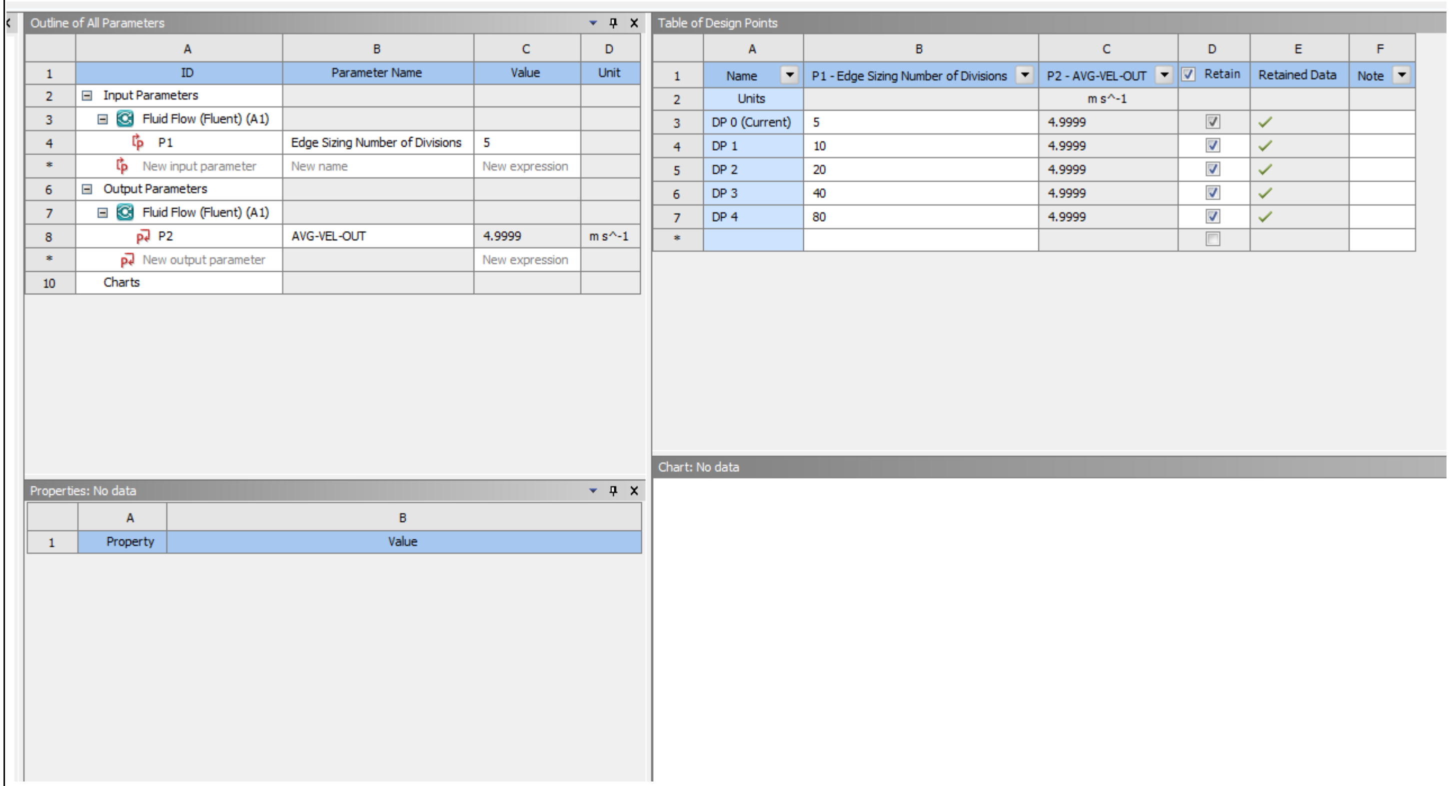
10=2N


Velocity Magnitude
[m/s]
5.56e+00
5.01e+00
4.45e+00
3.90e+00
3.34e+00
2.78e+00
2.23e+00
1.67e+00
1.11e+00
5.56e-01
0.00e+00
contour-5

20=4N


Velocity Magnitude
[m/s]
5.56e+00
5.01e+00
4.45e+00
3.90e+00
3.34e+00
2.78e+00
2.23e+00
1.67e+00
1.11e+00
5.56e-01
0.00e+00
contour-5

Case 2



Outline of All Parameters

	A	B	C	D
1	ID	Parameter Name	Value	Unit
2	Input Parameters			
3	Fluid Flow (Fluent) (A1)			
4	P1	Edge Sizing Number of Divisions	5	
*	New input parameter	New name	New expression	
6	Output Parameters			
7	Fluid Flow (Fluent) (A1)			
8	P2	AVG-VEL-OUT	4.9999	m s^-1
*	New output parameter		New expression	
10	Charts			

Table of Design Points

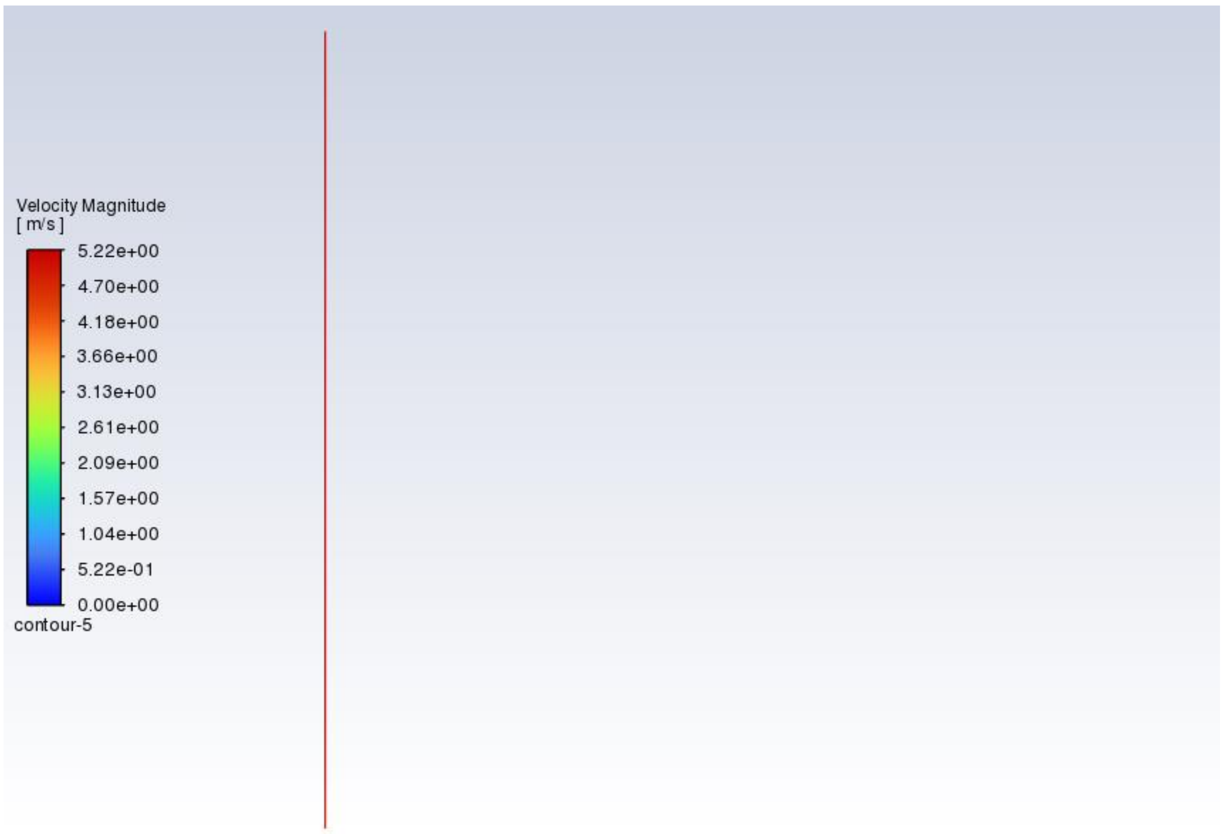
	A	B	C	D	E	F
1	Name	P1 - Edge Sizing Number of Divisions	P2 - AVG-VEL-OUT	✓ Retain	Retained Data	Note
2	Units		m s^-1			
3	DP 0 (Current)	5	4.9999	✓	✓	
4	DP 1	10	4.9999	✓	✓	
5	DP 2	20	4.9999	✓	✓	
6	DP 3	40	4.9999	✓	✓	
7	DP 4	80	4.9999	✓	✓	
*						

Properties: No data

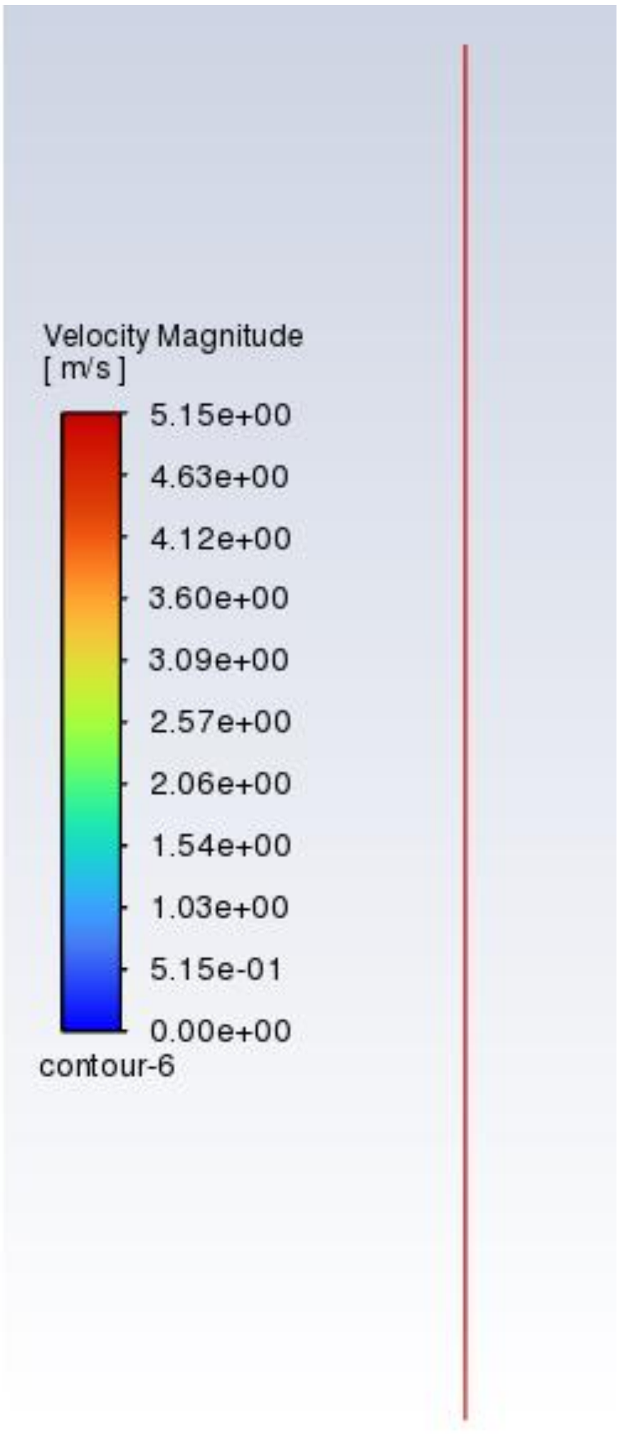
	A	B
1	Property	Value

Chart: No data

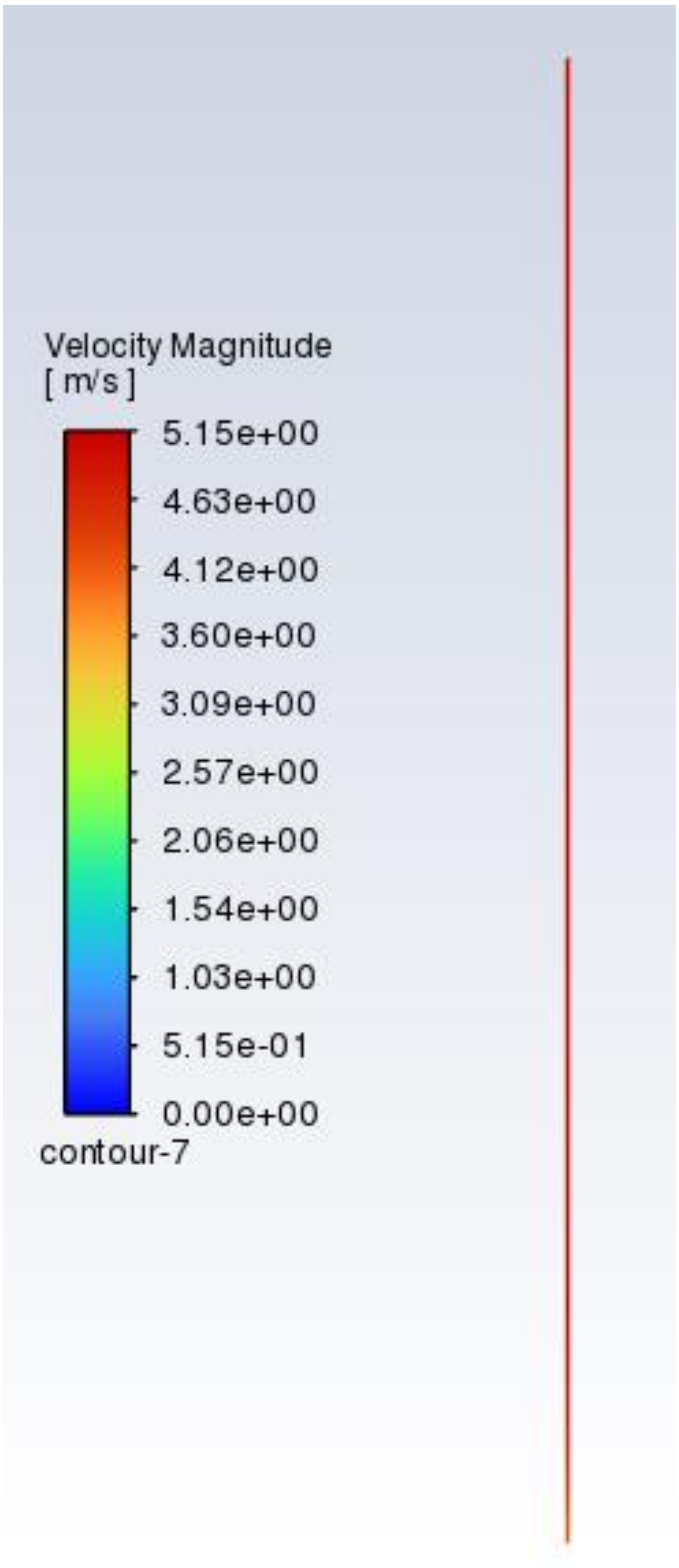
5=N

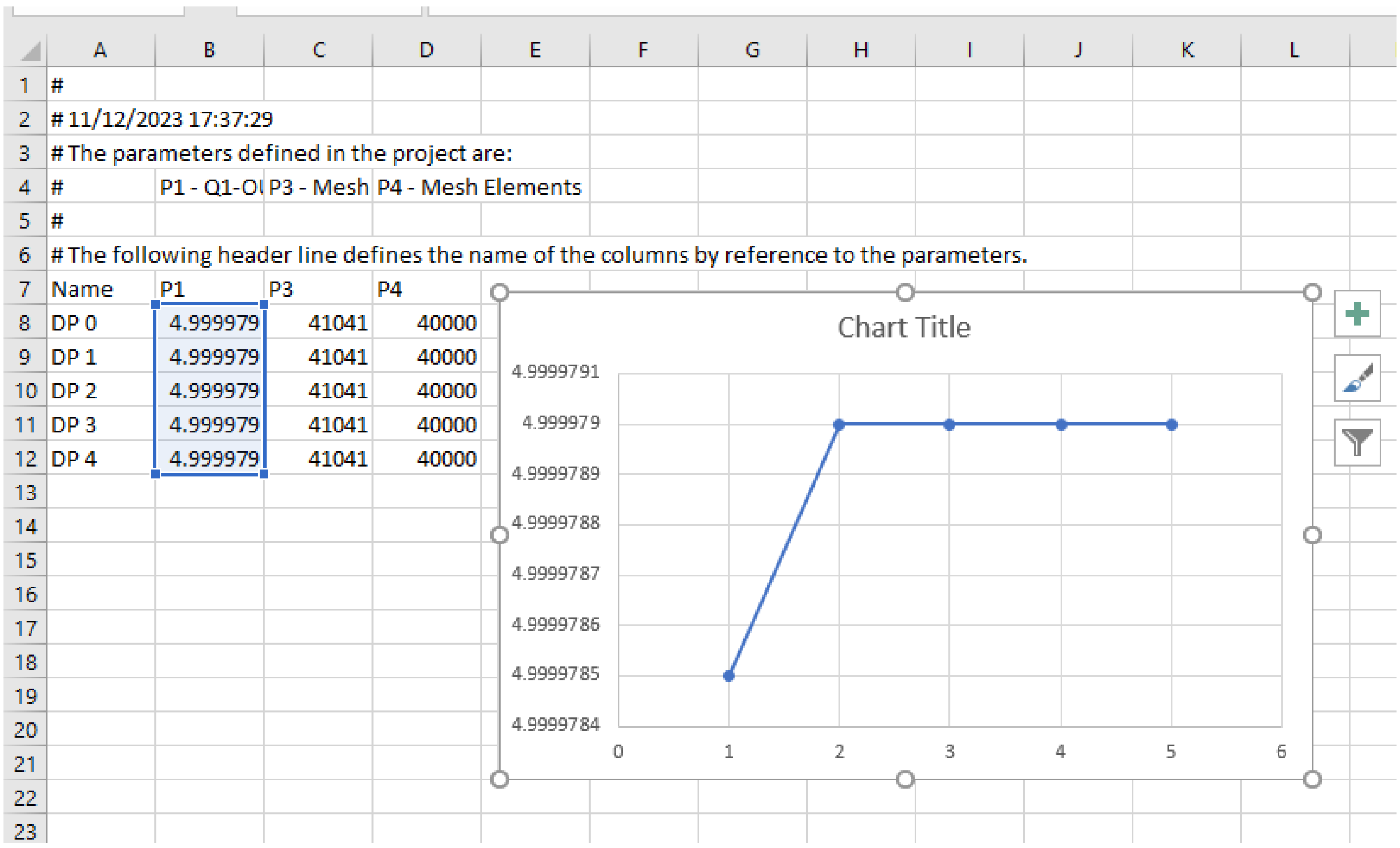


10=2N


Velocity Magnitude
[m/s]
5.15e+00
4.63e+00
4.12e+00
3.60e+00
3.09e+00
2.57e+00
2.06e+00
1.54e+00
1.03e+00
5.15e-01
0.00e+00
contour-6

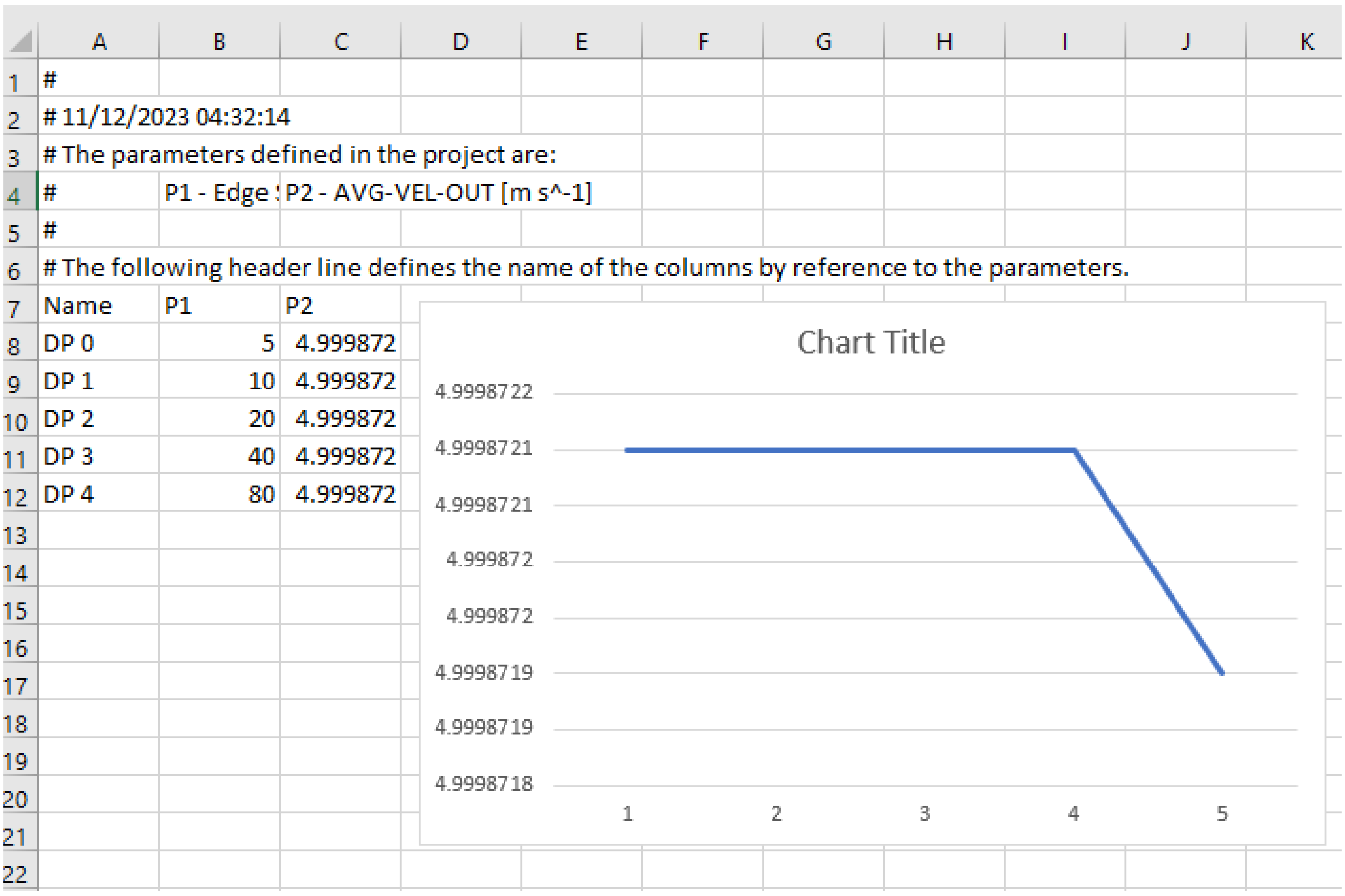
20=4N


Velocity Magnitude
[m/s]
5.15e+00
4.63e+00
4.12e+00
3.60e+00
3.09e+00
2.57e+00
2.06e+00
1.54e+00
1.03e+00
5.15e-01
0.00e+00
contour-7



	A	B	C	D
1	#			
2	# 11/12/2023 17:37:29			
3	# The parameters defined in the project are:			
4	#	P1 - Q1-O	P3 - Mesh	P4 - Mesh Elements
5	#			
6	# The following header line defines the name of the columns by reference to the parameters.			
7	Name	P1	P3	P4
8	DP 0	4.999979	41041	40000
9	DP 1	4.999979	41041	40000
10	DP 2	4.999979	41041	40000
11	DP 3	4.999979	41041	40000
12	DP 4	4.999979	41041	40000

Case1



	A	B	C
1	#		
2	# 11/12/2023 04:32:14		
3	# The parameters defined in the project are:		
4	#	P1 - Edge	P2 - AVG-VEL-OUT [m s^-1]
5	#		
6	# The following header line defines the name of the columns by reference to the parameters.		
7	Name	P1	P2
8	DP 0	5	4.999872
9	DP 1	10	4.999872
10	DP 2	20	4.999872
11	DP 3	40	4.999872
12	DP 4	80	4.999872

Case 2

Conclusion of Part B

The grid independence study conducted for the assigned problem in Project Assignment 1, which focused on ANSYS Fluent simulations, has yielded crucial insights into the optimal mesh size required for accurate and reliable results. Employing a systematic approach of comparing simulations on grids of sizes N, 2N, 4N, 8N, and so forth, the goal was to identify the point at which the solution became independent of further grid refinement. This iterative process allowed for the careful evaluation of velocity values at fixed coordinate locations along the simulation domain.

This comprehensive grid independence study not only enhances the credibility of the ANSYS Fluent simulations but also provides a valuable framework for future simulations, ensuring computational efficiency without compromising accuracy. The findings underscore the importance of judicious mesh refinement in achieving reliable numerical simulations for fluid dynamics problems.