B.Tech Project - CP303

Understanding Behaviour of Elastic Particles in Fluid

Submitted by

Akshat Chauhan (2021meb1265)

Kayitha Saran Yadav (2021meb1290)

Korupolu Deekshitha (2021meb1292)

Patel Mahant (2021meb1306)

Ujjawal Kumar (2021meb1331)

Under the supervision of

Dr. Navaneeth K. Marath

Department of Mechanical Engineering

Indian Institute of Technology, Ropar

Year of Submission: 2025

Abstract

This project focuses on calculating the hydrodynamic forces on elastic particles traversing through microfluidic channels using the Immersed Boundary Method (IBM). IBM is an effective computational technique that simulates the interaction between fluid flow(modeled on an Eulerian grid) and flexible structures (represented on a Lagrangian mesh).By applying this method, we aim to accurately model the forces acting on particles as they interact with fluid flows.The findings from this study will contribute to optimizing particle sorting and separation processes within microfluidic systems, with significant implications for applications in diagnostics, medical research, and industrial applications, such as bio-separation and lab-on-a-chip devices.

Contents

List of Figures

Nomenclature

β	Blockage ratio
μ	Dynamic Viscosity
ρ	Density
C_{D_f}	Friction drag
C_{D_P}	Pressure drag
E	Elastic Potential Energy
F	Force per unit area
f	Force Density on Eulerian Grid
k	Spring stiffness
M	Mass Density
P	Pressure
Re	Reynolds Number
X_M	Coordinates of master node
X_{SL}	Coordinates of slave node

1 Introduction

Microfluidic devices are essential tools in a variety of fields, especially in biological and medical diagnostics. They work by controlling and manipulating fluids at a very small scale, making it possible to separate, mix, or sort tiny particles like cells and proteins. One of their key uses is in separating healthy cells from diseased ones, such as in cancer diagnostics or isolating pathogens in a blood sample. As these cells or particles move through the tiny channels in these devices, they experience different hydrodynamic forces, which play a crucial role in how effectively they can be sorted. To optimize these devices, it is important to understand and quantify these forces. Microfluidic devices use different techniques to separate particles, including methods that leverage properties like particle size, shape, and electrical charge. Some of the techniques include deterministic lateral displacement, inertial focusing, dielectrophoresis, and acoustic separation. These methods are especially useful in medical diagnostics, where rapid and accurate sorting can make a significant difference[1]. When particles move through microchannels, they are subjected to several forces, such as drag, lift, and pressure forces. Understanding these forces and how they interact with the particles is essential for improving the design and efficiency of microfluidic devices. However, calculating these forces can be challenging due to the complex nature of fluid-structure interactions.

To address this complexity, we are using the Immersed Boundary Method (IBM), a computational technique originally developed by Dr. Charles Peskin in 1972 [2] to model how blood flows through heart valves.It has a unique ability to perform the entire simulation on a fixed cartesian grid, without explicitly creating a mesh. IBM has since evolved into a widely used tool for simulating the interactions between fluids and flexible structures. It has been applied in fields ranging from biology and bioengineering to materials science. IBM is particularly effective for systems where particles or structures are immersed in moving fluid, such as in microfluidic devices. The way IBM works is by combining two types of grids: one fixed grid for the fluid (Eulerian) and a flexible, moving grid for the structures (Lagrangian). This combination allows for accurate simulations of how the fluid affects the particles and vice versa. One of the best things about IBM is that it can handle large deformations of structures without needing to constantly remesh the grid, making it ideal for microfluidic applications where particle shapes and behaviors can vary greatly.

For this project,we implemented the Immersed Boundary Method (IBM) using ANSYS

(Dynamic Mesh) to simulate the movement of elastic particles in microchannels and calculate the hydrodynamic forces acting on them. IBM is particularly advantageous for such simulations, as it eliminates the need for body-fitted meshing, making it highly efficient for modeling fluid-structure interactions. The purpose of this study was to investigate the influence of these forces on particle behavior, which is essential for optimizing microfluidic device designs. By improving our understanding of particle transport and deformation in confined flows, we can enhance the efficiency of microfluidic systems used in biomedical and industrial applications.

We extend our research to analyze the crossflow of fluids past cylinders with circular cross-section in confined channels. This problem is of both fundamental and practical importance, as it provides insights into flow separation, wake formation, and drag forces in bounded environments. Our focus is on steady-state flow around a circular cylinder, examining the impact of the blockage ratio (β=H/D) on fluid dynamics, drag forces, and wake development. Using IBM within ANSYS, we efficiently simulate fluid-structure interactions without requiring complex meshing techniques. This study further explores the contributions of pressure drag and friction drag to the total drag coefficient, as well as the influence of Reynolds number on flow characteristics. The findings will provide valuable insights into the effect of confinement on flow behavior, which has implications for microfluidic design, biomedical applications, and engineering systems involving confined flows.

Section 2 provides a review of the literature on the development of the Immersed Boundary Method. Section 3 discusses the implementation of IBM and provides details on the open-source MATLAB code we are studying and using. The simulation conditions are also outlined. Finally, the corresponding results are presented and analyzed in Section 4.

2 Literature Review

Fluid-structure interaction (FSI) problems are crucial in both engineering and biological systems, where the interaction between fluids and elastic structures leads to complex, fully coupled dynamics. The Immersed boundary method, pioneered by Dr.Charles Peskin, is particularly well-suited to model such systems, with applications ranging from cardiovascular dynamics to aquatic locomotion and insect flight.The method employs a fixed Eulerian grid for the fluid domain and a moving Lagrangian framework for the structure. Forces exerted by the structure on the fluid are interpolated, enabling the simulation of interactions between flexible or rigid

bodies and the surrounding fluid with high accuracy. Subsequent developments have adapted IBM to a wide range of problems, including particle-laden flows and microfluidic applications. There are several high-performance implementations of the IB method, such as IBAMR (which

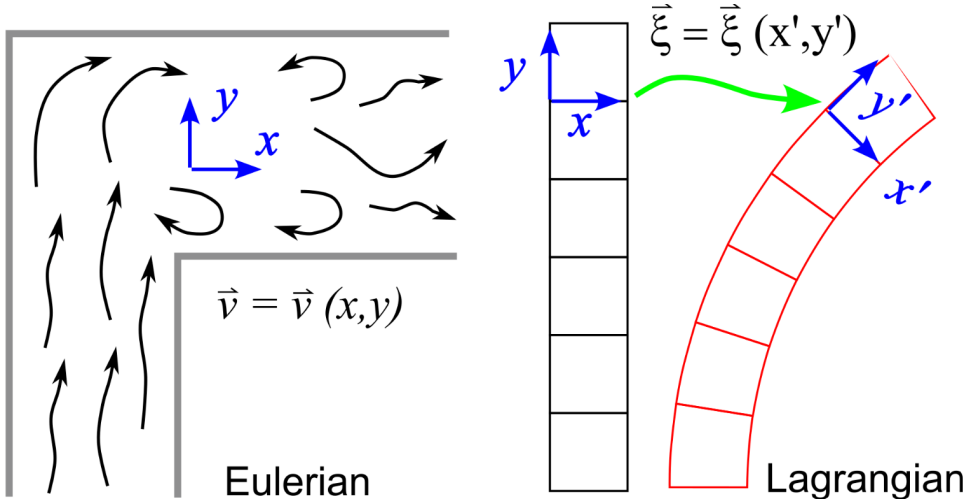

Figure 1: Difference between Lagrangian and Eulerian description

supports adaptive mesh refinement and parallelization in C++) and IBIS (implemented in FORTRAN). However, these require a deep understanding of lower-level programming languages and are computationally intensive, making them inaccessible to many researchers. Moreover, the installation and usage of these software packages can be complex due to dependencies on various libraries and high-performance computing requirements.

While there are other open-source IB codes, like matIB and pyIBM, they do not offer the range of fiber models, examples, or efficiency found in IB2d. Additionally, these earlier tools were often confined to specific problem domains or lacked broad applicability in biomechanics.

The IB2d package is positioned as a highly accessible tool for both experienced and novice users in the scientific community, providing full implementations in Python and MATLAB. IB2d simplifies the workflow for solving FSI problems and allows users to model a wide range of biomechanics problems, such as plant biomechanics, insect flight, muscle-fluid interactions, and physiological processes.

One of the key contributions of IB2d is its diverse array of fiber models, which provide significant flexibility in simulating the material properties of immersed structures. These models include Hookean and non-Hookean springs for elastic stretching, torsional springs for resistance to bending, and target points for enforcing motion or fixation at specific locations. Additionally, mass points allow the incorporation of inertia and gravity effects, while porous models simulate the permeability characteristics of biological tissues. Muscle models further enhance the framework by representing muscle dynamics through force-velocity and length-tension relationships, enabling detailed and realistic simulations of physiological behaviors.

The implementation of IBM ideology with ANSYS (Dynamic Mesh) represents a significant advancement in CFD simulations, offering improved efficiency and accuracy in modeling deformable structures in complex flow environments. By leveraging the versatile material representations of IBM and the adaptive meshing capabilities of ANSYS, researchers can achieve high-fidelity simulations applicable to a broad range of engineering and biomedical problems. Dynamic meshing is a widely used computational technique in Computational Fluid Dynamics (CFD) to simulate problems involving moving boundaries, deforming geometries, and fluid-structure interactions (FSI). Unlike static meshing, where the computational grid remains fixed, dynamic meshing enables the adaptation of the grid in response to changes in geometry, improving the accuracy of simulations without excessive computational costs.

Several studies have explored dynamic mesh techniques to handle complex simulations efficiently. Zhang et al. (2010) highlighted that dynamic meshing is essential for FSI problems, particularly in biomedical applications, aerodynamics, and microfluidics. The Spring-Based Smoothing method, where mesh nodes behave like interconnected springs, and the Laplacian Smoothing method, which redistributes node positions based on neighbors, are commonly used to improve mesh quality during deformation (Blom, 2000). In addition, local remeshing techniques are employed to refine or coarsen the mesh in highly deformed regions, ensuring numerical stability (Löhner, 2008).

In CFD software like ANSYS Fluent, dynamic meshing is implemented using three primary

methods:

Smoothing: Adjusts node positions to maintain mesh quality. Local Remeshing: Replaces distorted elements with refined or coarser mesh elements. Layering: Adds or removes mesh layers in structured grids to accommodate boundary motion. Studies by Mittal Iaccarino (2005) demonstrated the effectiveness of dynamic meshing for immersed boundary simulations, particularly in handling moving objects within a fluid domain without the need for body-fitted meshes. This has applications in microfluidic particle transport, oscillating structures, and biological tissue simulations.

Because of the compatibility issues with our software,we are working in Ansys using Dynamic meshing and UFD. Dynamic meshing and User-Defined Functions (UDFs) are crucial in computational fluid dynamics (CFD) for handling simulations involving moving boundaries and deforming structures. Dynamic meshing techniques allow the computational grid to adapt as the geometry moves, making it suitable for applications such as aerodynamics, fluid-structure interaction (FSI), and combustion systems. Various approaches to dynamic meshing include smoothing methods like Laplacian smoothing and the spring-based method, remeshing techniques that reconstruct the mesh when excessive distortion occurs, and the layering method used for structured meshes. Additionally, the Arbitrary Lagrangian-Eulerian (ALE) method provides a hybrid approach that accounts for fluid-structure interactions effectively. These methods have been widely applied in simulations of airfoil motion, arterial blood flow, and moving piston engines.

User-Defined Functions (UDFs) in CFD enable customization of solvers by allowing users to define complex physics beyond built-in solver capabilities. Written in the C programming language, UDFs help define boundary conditions, specify custom material properties, introduce source terms for transport equations, and control dynamic meshing. In the context of dynamic meshing, UDFs can be used to prescribe motion for rigid bodies, implement deforming mesh techniques, and enforce mesh quality controls to prevent excessive skewness. They also play a key role in applications such as defining temperature-dependent viscosity in non-Newtonian fluid simulations and incorporating adaptive mesh refinement (AMR) based on flow gradients. While UDFs provide greater flexibility in simulations, they come with challenges such as increased computational costs, potential numerical instabilities, and the need for efficient coding practices to optimize performance.

Combining dynamic meshing with UDFs enhances the ability of CFD solvers to tackle

complex problems across aerospace, biomedical engineering, and energy systems. While commercial software like ANSYS Fluent, OpenFOAM, and STAR-CCM+ offer built-in dynamic meshing capabilities, UDFs allow further customization, making them essential for specialized research and engineering applications. Future advancements in AI-driven mesh adaptation and hybrid meshing techniques could further improve the efficiency and accuracy of CFD simulations involving moving boundaries.

3 Analytical and Simulation Details

In this section, we discuss the details of the simulation[3]. We are trying to simulate the rigid circular cylinder inside a channel with a Poiseuille flow and calculate its drag force to validate our code.

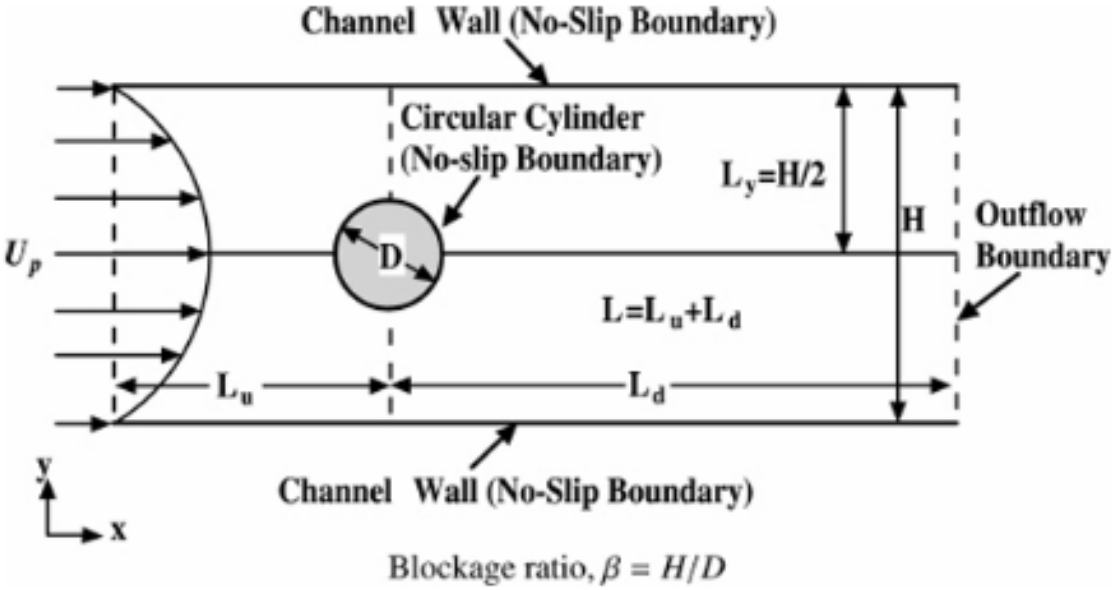

Figure 2: Geometry in consideration

We will first discuss the fiber models that are used. Then we will discuss the workflow of the simulation and the different functions and files used in the simulation.

3.1 What are Fiber Models?

The governing equation for fluid flow in the immersed boundary in Eulerian form is given as:

$$\rho\left(\frac{\partial \mathbf{u}(\mathbf{x},\mathbf{t})}{\partial \mathbf{t}} + \mathbf{u}(\mathbf{x},\mathbf{t})\cdot\nabla\mathbf{u}(\mathbf{x},\mathbf{t})\right) = -\nabla\mathbf{P}(\mathbf{x},\mathbf{t}) + \mu\nabla^2\mathbf{u}(\mathbf{x},\mathbf{t}) + \mathbf{f}(\mathbf{x},\mathbf{t}) \tag{1}$$

where, u(x,t) is the fluid velocity, P(x,t) is the pressure field, and f(x,t) is the force applied on the Eulerian grid by the immersed boundary. The immersed boundary method assumes periodic boundary conditions and a square fluid domain. The interaction equations between the fluid

and the immersed structure are given by:

$$\mathbf{f}(\mathbf{x},\mathbf{t}) = \int \mathbf{F}(\mathbf{r},\mathbf{t})\delta(\mathbf{x}-\mathbf{X}(\mathbf{r},\mathbf{t}))dr \tag{2}$$

$$\mathbf{U}(\mathbf{X}(\mathbf{r},\mathbf{t}),\mathbf{t}) = \frac{\partial \mathbf{X}(\mathbf{r},\mathbf{t})}{\partial \mathbf{t}} = \int \mathbf{u}(\mathbf{x},\mathbf{t})\delta(\mathbf{x}-\mathbf{X}(\mathbf{r},\mathbf{t}))dx \tag{3}$$

where, X(r,t) provides the Cartesian with coordinates at time t of the material point labeled by Lagrangian parameter r, F(r,t) is the force per unit area imposed onto the fluid by elastic deformations in the immersed structure as a function of the Lagrangian position, r, and time, t. The force density, F(r,t) is a function of the current immersed boundary's configuration. Various fiber models calculate the force density F(r,t) [4].

3.2 Fiber Models Used in Simulation

To calculate the force density F(r,t), we implement various fiber models to the Eulerian and Lagrangian points [5]. The following types of fiber models are used in the simulation:

1. Springs (Hookean)
2. Torsional springs (beams)
3. Target Points
4. Mass Points (with gravity)
5. Background Flow Profiles

We calculate the force density based on the potential energy due to the deformation. Once the total deformation energy has been calculated,

$$\mathbf{E}(\mathbf{X}(\mathbf{r},\mathbf{t}),\mathbf{t}) = \sum_{k=1}^{M} \mathbf{E}_{\mathbf{k}}(\mathbf{X}_{\mathbf{k},\mathbf{1}}, \mathbf{X}_{\mathbf{k},\mathbf{2}}, ..., \mathbf{X}_{\mathbf{k},\mathbf{M}}) \tag{4}$$

then the corresponding elastic forces are calculated by the derivative of the deformation energy, where the elastic deformation force at point c of fiber model k is calculated as

$$\mathbf{F}_{\mathbf{k},\mathbf{c}}(\mathbf{X}(\mathbf{r},\mathbf{t}),\mathbf{t}) = -\frac{\partial \mathbf{E}(X(r,t),t)}{\partial \mathbf{X}_{\mathbf{k},\mathbf{c}}} \tag{5}$$

Note that X contains all immersed boundary points coordinates, M is the number of fiber structures in the system, and M is the number of immersed boundary points in the fiber structure.

3.2.1 Springs(Hookean)

Resistance to stretching between successive Lagrangian points can be achieved by modeling the connections with Hookean springs of resting length R_L and spring stiffness. If the virtual spring displacement is below or beyond, the model will drive the system back towards a lower energy state. The elastic potential energy for a Hookean spring is given by:

$$\mathbf{E_{spring}} = \frac{1}{2} k_s (||X_{SL} - X_M|| - R_L)^2 \tag{6}$$

where X_M and X_{SL} are master and slave node coordinates respectively. The corresponding deformation forces is given by differentiation of the elastic energy:

$$\mathbf{F_{spring}} = k_s \left(1 - \frac{R_L}{||X_{SL} - X_M||}\right) \cdot \begin{pmatrix} x_{SL} - x_M \\ y_{SL} - y_M \end{pmatrix} \tag{7}$$

3.2.2 Torsional Springs

Resistance to bending between three successive Lagrangian points is modeled using a torsional spring connecting the three nodes. The model assumes a desired angle θ, a prescribed curvature $\mathbf{C}$ between the three Lagrangian points, with corresponding bending stiffness k_b. The corresponding bending energy is given as:

$$\mathbf{E_{bend}} = \frac{1}{2} k_b (\hat{\mathbf{z}} \cdot (X_R - X_M) \times (X_M - X_L) - \mathbf{C})^2 \tag{8}$$

where X_R, X_M , X_L , are right, left, and master/middle Lagrangian nodal coordinates. The penalty force is designed to drive any deviations in the angle between these links back toward a

lower energy state given by:

$$\mathbf{F_{bend}} = k_b \left[(x_R - x_M)(y_M - y_L) - (y_R - y_M)(x_M - x_L) - \mathbf{C} \right] \cdot \begin{pmatrix} (y_M - y_L) + (y_R - y_M) \\ -(x_R - x_M) - (x_M - x_L) \end{pmatrix} \tag{9}$$

3.2.3 Target Points

Target points are used to describe the motion of Lagrangian points. Each Lagrangian point is associated with a virtual or target point. Target points are Lagrangian points that are anchored or have specific target positions. they both are connected by a spring with zero resting length. and the Elastic Energy associated is given by:

$$\mathbf{E_T}(\mathbf{X_M}) = \frac{1}{2} k_T (||X_M - X_M^T||)^2 \tag{10}$$

where k_T is the target point stiffness and X_M and X_M^T are the coordinates of the physical Lagrangian and virtual target points, respectively. The corresponding deformation force is given by the derivative of the elastic energy.

$$\mathbf{F_T} = -k_T \begin{pmatrix} x_M - x_M^T \\ y_M - y_M^T \end{pmatrix} \tag{11}$$

3.2.4 Massive Points

Massive points are artificial points and they do not interact with the fluid directly and they are virtual points. Y(r,t) gives the coordinates of the massive points with mass density M(r) that do not interact with the fluid. X(r,t) gives the coordinates of the massless points that interact with the fluid. The massive and massless points are connected by a stiff virtual spring. As the fluid moves it exerts some force on the massless point, so this point will move to a new location, but since the massless point is connected by a stiff spring i.e. $k_M >> 1$, which takes the massive point with it. The force acting on the massless point is transmitted to the massive point and it is given by:

$$\mathbf{F_M} = -k_M \left(Y(r,t) - X(r,t)\right) \tag{12}$$

This force will drive the position of the massive point according to the equation:

$$\mathbf{M(r)} \frac{\partial^2 (Y(r,t)}{\partial t^2} = -F_M - M(r) g \hat{e_2} \tag{13}$$

3.2.5 Background Flow Profiles

Although the computational domain is assumed to have periodic boundary conditions, we can induce a desired background flow profile by artificially adding a force term in the Naiver Stokes Equation 1. Essentially, the additional force will be a penalty-type term, which exerts a force onto a desired subset of the fluid grid, if the fluid velocity does not match the desired flow profile. Such a forcing term can take the form:

$$\mathbf{F_{arb}} = k_{arb} \left(u(r,t) - u_{flow}(r,t)\right) \tag{14}$$

where k_{arb} is the penalty strength coefficient and u_{flow} is the desired background velocity profile. In this simulation, a parabolic profile into the channel was generated in the form:

$$\mathbf{u_{flow}(x,t)} = U_{max} \left(1 - \left(\frac{0.5 - x}{R}\right)^2\right) \tag{15}$$

The y-component of the desired velocity was assumed to be 0.

3.3 Files Used in Simulation

In this subsection, we will describe the simulation workflow. We will be looking at the MATLAB files and Python codes which we will be using for the simulation. At present, we are analyzing the open-source codes available for IBM from [3].

3.3.1 Main2d and Input2d

For every IBM simulation, there is a main2d and input2d file associated with it. The input2d is a file where the user needs to select the parameters for the simulation i.e. the fluid parameters, temporal information, grid parameters, fiber model construction, how to save the data, etc. Once the user has selected the desired parameters and necessary flags in input2d, the simulation is then started by calling the main2d file. This script reads all the information from the input2d file and passes it to the IBM-Driver script. We will come back to the IBM Driver later.

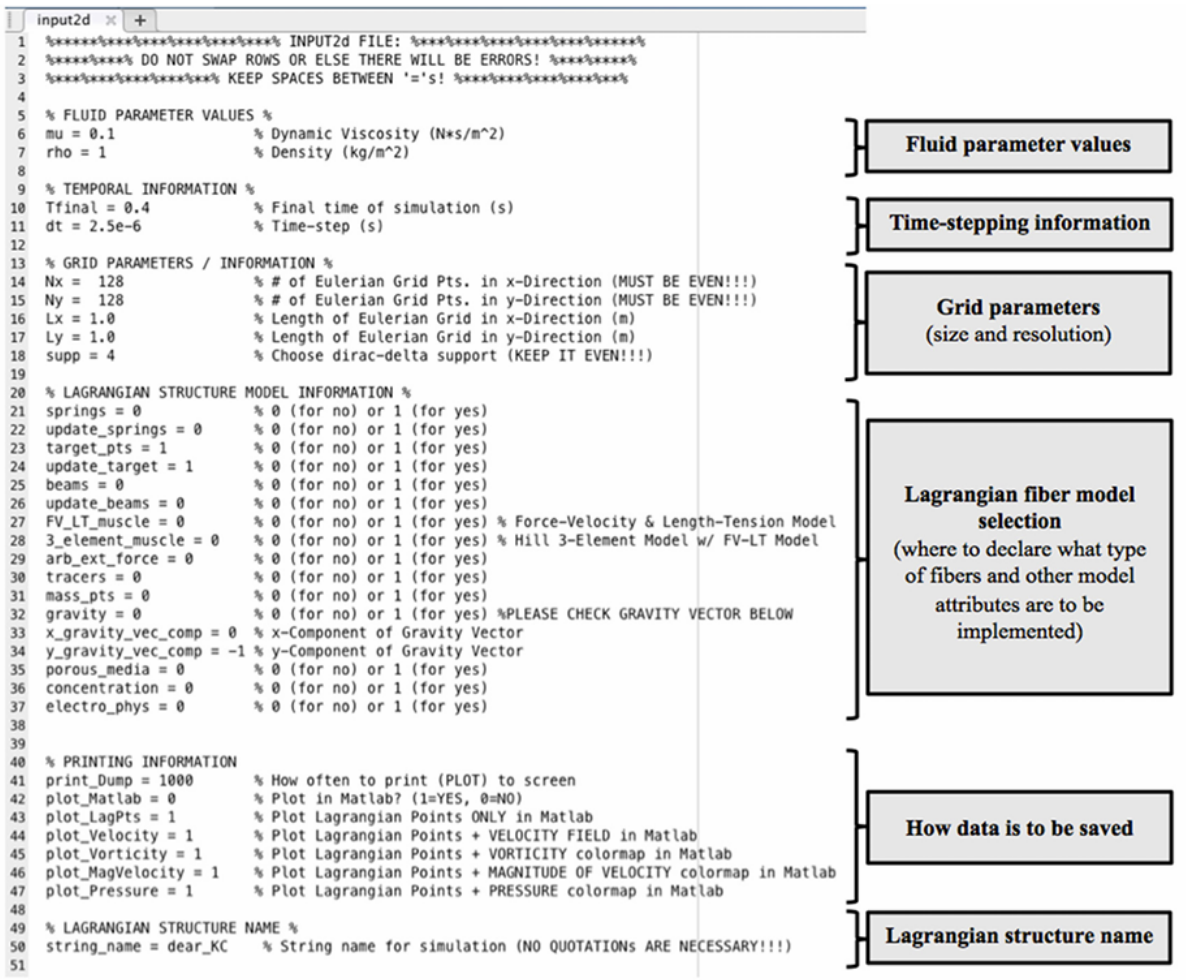

Figure 3: Input2d File Format

For our simulation of a cylinder in a vertical channel with an external flow against gravity, the fiber models which was selected were springs, target points, beams, external background forcing, and mass points with gravity in the negative y-direction. The fluid density $\rho = 1\frac{kg}{m^3}$ and viscosity $\mu = 0.01\frac{N.s}{m^2}$ were chosen. We will be observing the velocity contours and the motion of the Lagrangian points in the simulation.

3.3.2 Geometry Files

The first thing to do is to create a geometry for the simulation. The geometry file will read in the input parameters like grid spacing, and dimensions of the computational domain. From that, we define the dimensions of other structures used in the simulation. It will also read the fiber models that we have selected in the input2d file and according to that, it will create a *.vertex*, *.spring*, *.beam*, *.target*, *and .mass* files associated with the geometry of the simulation. These files represent the coordinates of the respective fiber model points and the stiffness of the spring used in the respective model.

The .vertex file lists all the initial Lagrangian coordinates in the domain.

The .spring file lists all the master and slave nodes for each linear spring, including their stiffness and resting length. If non-linear springs are used, it also shows the degree of non-linearity.

The .beam file lists the right, middle, and left Lagrangian indices of each beam and their associated stiffness and curvature.

The .target file lists all target point indices with the target point stiffness.

The .mass file lists the Lagrangian mass point indices with their associated mass and spring stiffness.

The computational domain in our case is 1 m x 1 m in dimension with a rigid cylinder of diameter 0.067 m inside the 0.9 m long channel with a diameter of 0.2m. The blockage ratio in our case is 3. 128 grid points were chosen in each direction. The final geometry after running the geometry file was obtained as shown:

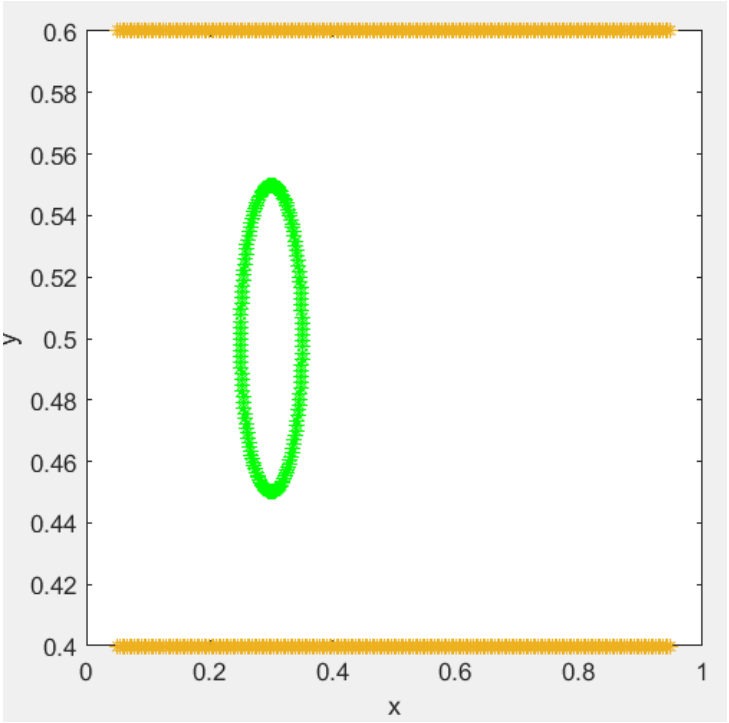

Figure 4: Geometry in consideration a 2d cylinder in an open channel

3.3.3 IBM Driver

The IBM Driver file is common for all the simulations. It takes the input from the input2d file and runs the selected functions for the fiber models the user has chosen in the input2d file. The IBM driver is like the core of all the simulations.

It will read in the geometry files according to the fiber models selected. After reading the geometry files, it will initialize the simulation by creating the zero vectors of velocity, vorticity, pressure, and the forces on the Lagrangian grid. After initializing the simulation, the time stepping gets started. It will update the position of the Lagrangian points and find the corresponding forces that will act on the Eulerian grid due to the fluid-structure interaction.

After finding the forces on the Eulerian grid, the fluid solver will update the fluid velocity, pressure, and forces with the iterations. The Naiver-Stokes equation is solved using spectral methods in this simulation. After getting the final velocity, pressure, vorticity, and the positions

of the Lagrangian points, the IBM driver will check which results the user wants to plot. Accordingly, it will be plotted in MATLAB.

The IBM Driver uses many other functions that are defined specifically for a purpose like reading the geometry files, initializing the simulation, updating the velocity by solving Naiver-Stokes, updating the position of Lagrangian points, etc. These are also common for all the simulations.

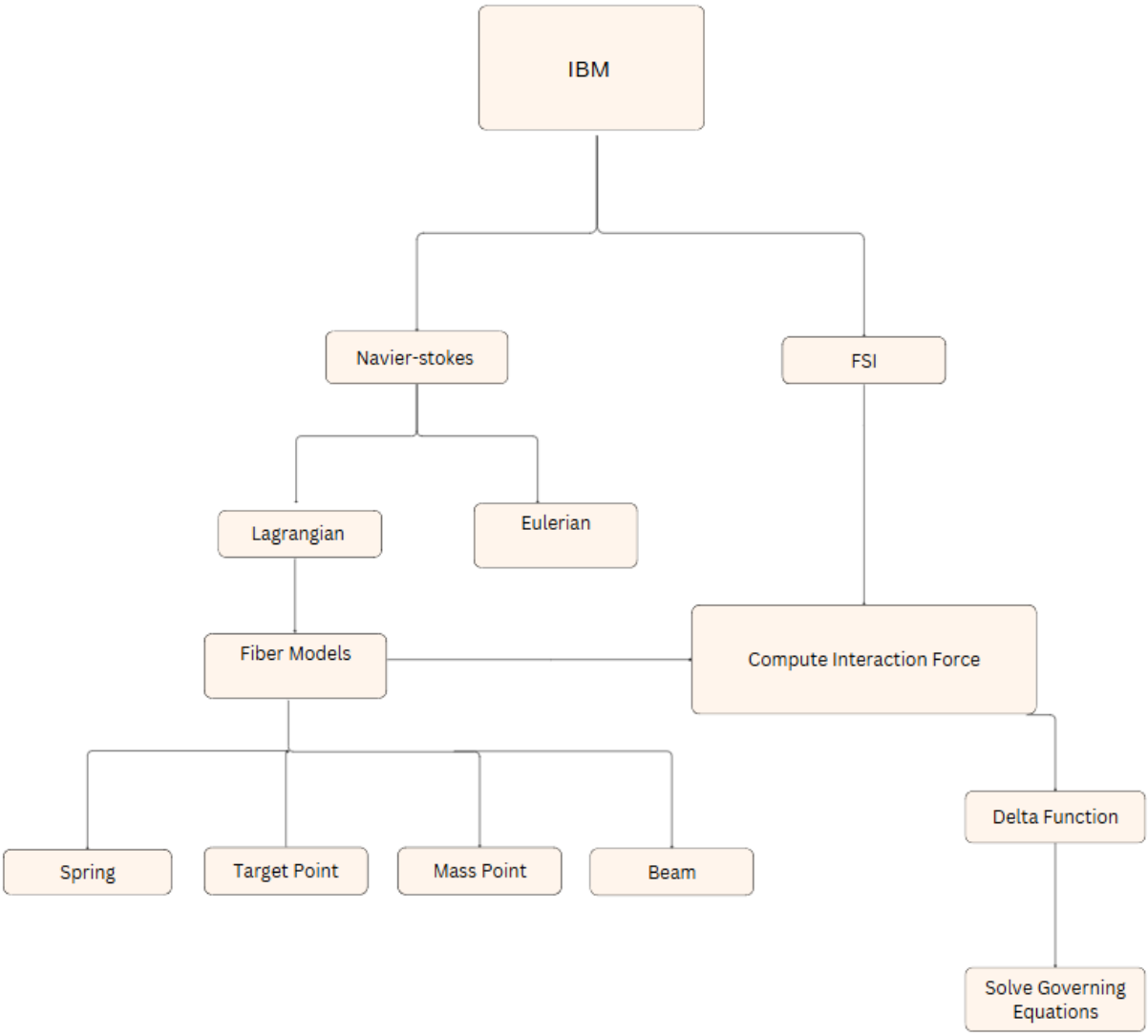

Figure 5: IBM workflow

4 Results and discussions

We ran the simulation for Re = 1 and Re = 5. The values of the drag coefficient from the MATLAB simulation did not match the value in the literature. So in order to find the cause of this error, we ran the simulation in ANSYS Fluent with the same conditions.

1. **Re = 5:**

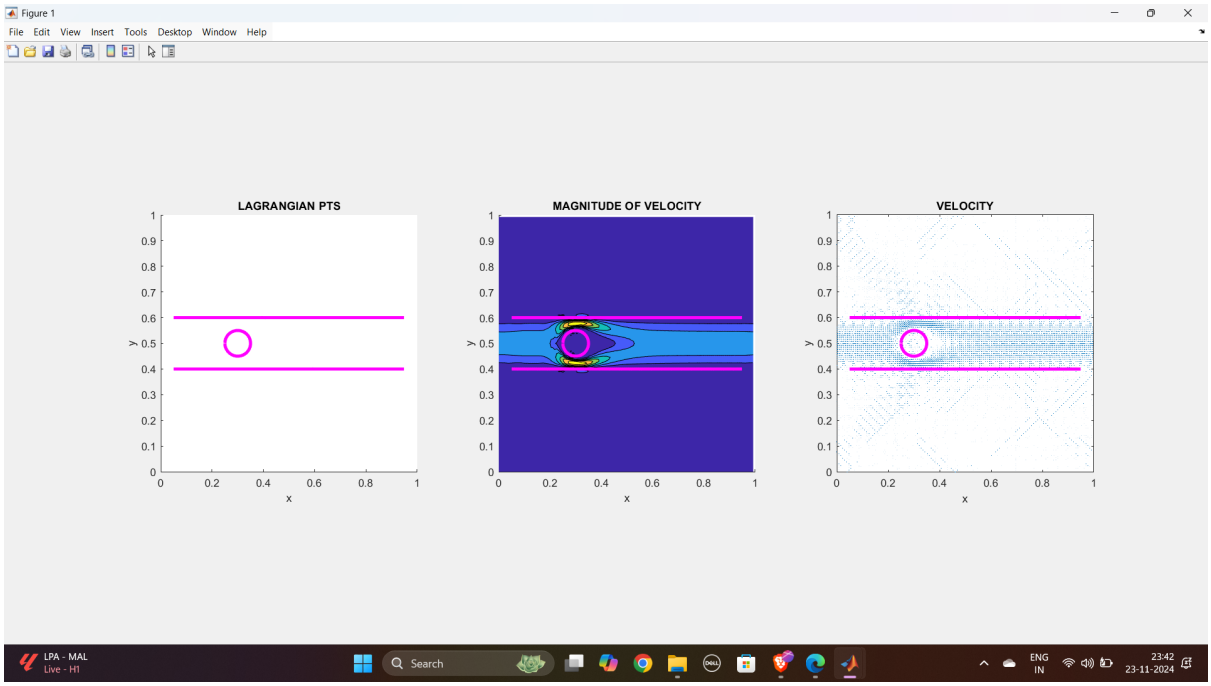

Figure 6: Flow over a cylinder at Re = 5, MATLAB

The simulation from MATLAB gave us insights into the position of Lagrangian points(left), velocity contours(middle), and velocity vectors (right). We were also able to calculate the drag force acting on the cylinder.

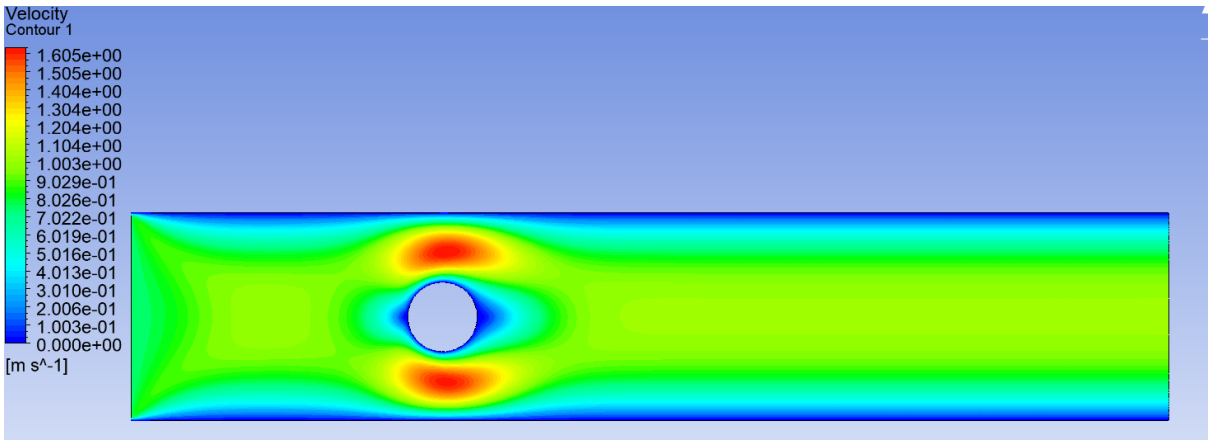

Figure 7: Velocity contour of fully developed poiseuille flow for Re=5, ANSYS

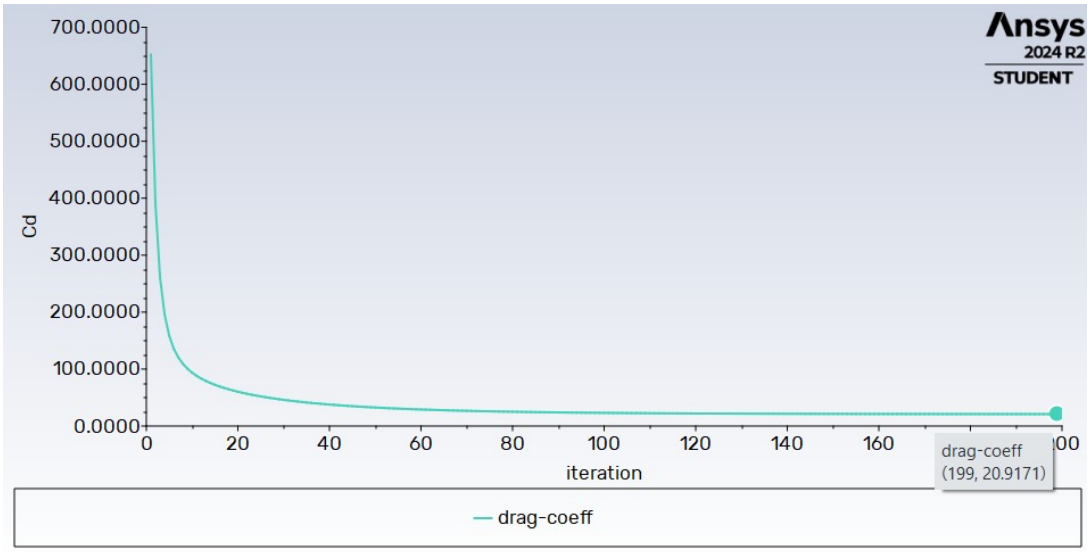

Figure 8: Drag coefficient convergence at Re =5

The graph shows the convergence behavior of the drag coefficient (Cd) as a function of the number of iterations during the simulation. At Reynolds number (Re) = 5, the drag coefficient initially exhibits large fluctuations but stabilizes as the solution converges with increasing iterations. The final value of Cd is approximately 21, indicating a steady-state solution has been achieved.

2. Re = 1:

The simulations were performed in MATLAB and ANSYS Fluent for Re = 1

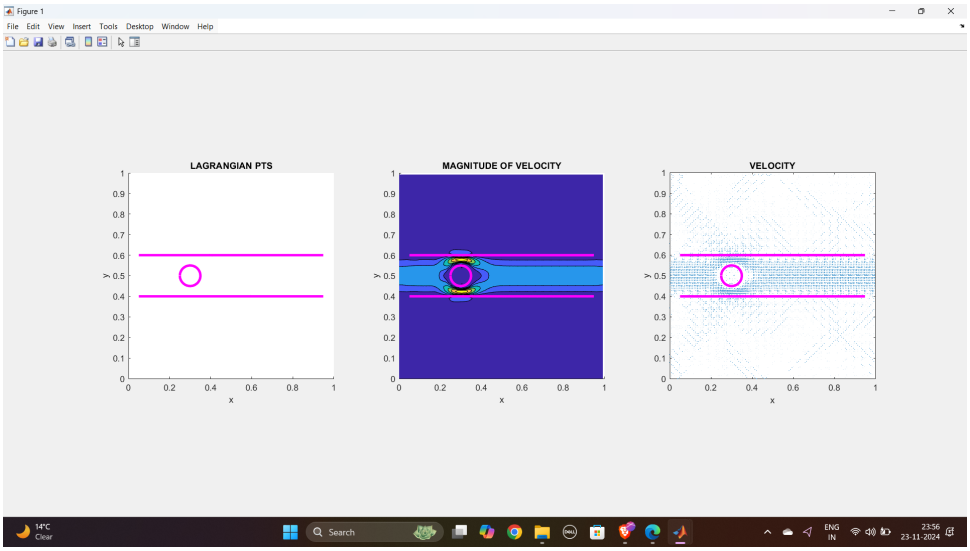

Figure 9: Flow over cylinder at Re = 1, MATLAB

The simulation from MATLAB gave us insights into the position of Lagrangian points(left), velocity contours(middle), and velocity vectors (right). We were also able to calculate the drag force acting on the cylinder.

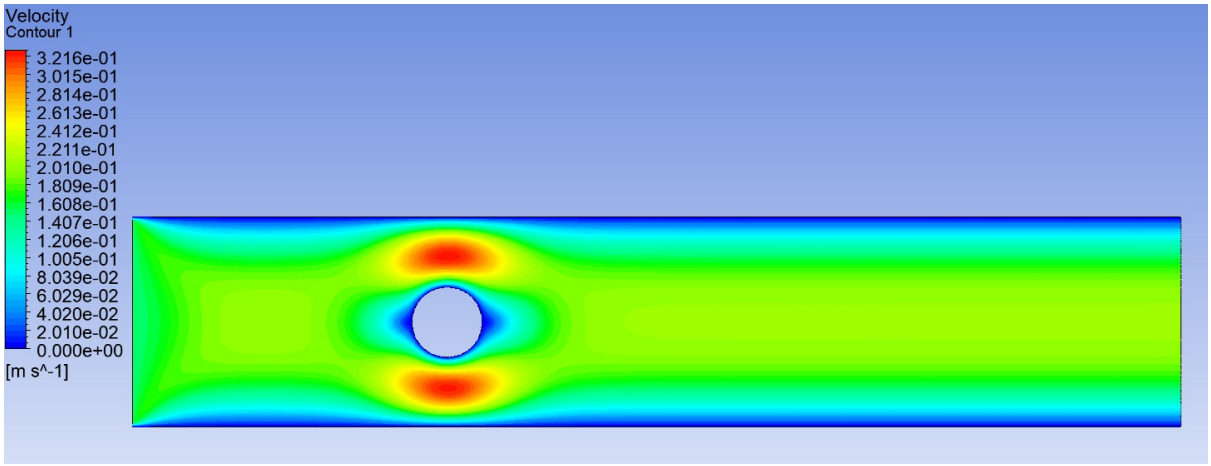

Figure 10: Velocity contour of fully developed poiseuille flow for Re=1, ANSYS

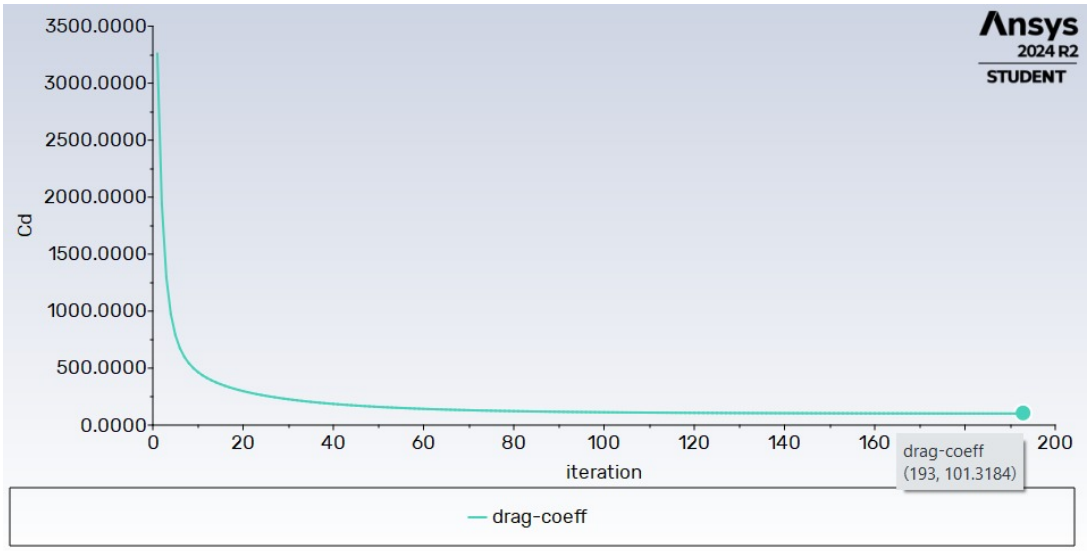

Figure 11: Drag coefficient convergence at Re =1

The graph shows the convergence behavior of the drag coefficient (Cd) as a function of the number of iterations during the simulation. At Reynolds number (Re) = 1, the Cd initially exhibits large fluctuations but stabilizes as the solution converges with increasing iterations. The final value of Cd is approximately 101.3184, indicating a steady-state solution has been achieved.

After running the simulation in Ansys fluent, velocity contour was plotted for Re = 5 , 1 and Drag coefficient was caluclated Grid Independence test was also performed for different grid size and the result obtained was same for the both tests.

The values obtained from the MATLAB simulations and ANSYS simulations were compared to the values from the literature[6].

Drag coefficient	Literature	MATLAB	ANSYS
Re = 1	117	20	101
Re = 5	23	4.95	21

Table 1: Drag coefficient comparison

As seen from the above table, the values obtained from ANSYS and literature values are closer to each other than the values obtained from MATLAB. This means that the MATLAB code had some error due to which the results were not matching.

Semester-8 Work:

Our Aim is to find and compare drag coefficient on an ellipse at different orientations i.e., at θ = 90 degrees and θ = 40 degrees respectively. And the dimensions of the ellipse: minor axis = 2b = 30mm and major axis = 2a = 150mm. The ellipse is placed inside a rectangular channel of length = 1m and height = 0.1m. The flow is Laminar and Re = 100. The properties of fluid is as follows: density = 1, viscosity = 0.01.

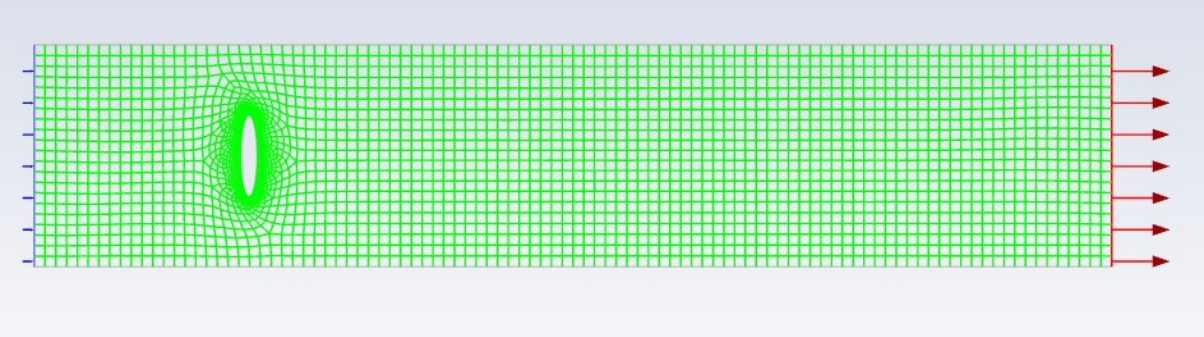

Figure 12: Drag coefficient around an ellipse at 90° at Re =100

At θ = 90 degrees After running the simulation we found the drag coefficient as 4.622. Grid independence test was also performed for different grid sizes namely element size of 0.01 and 0.02 and the results obtained was almost same for the both tests.

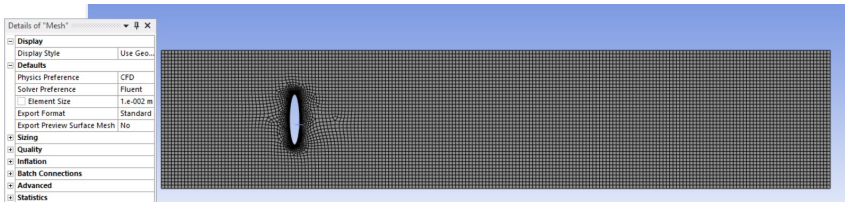

Figure 13: Geometry for an ellipse at 90° with mesh size 0.01

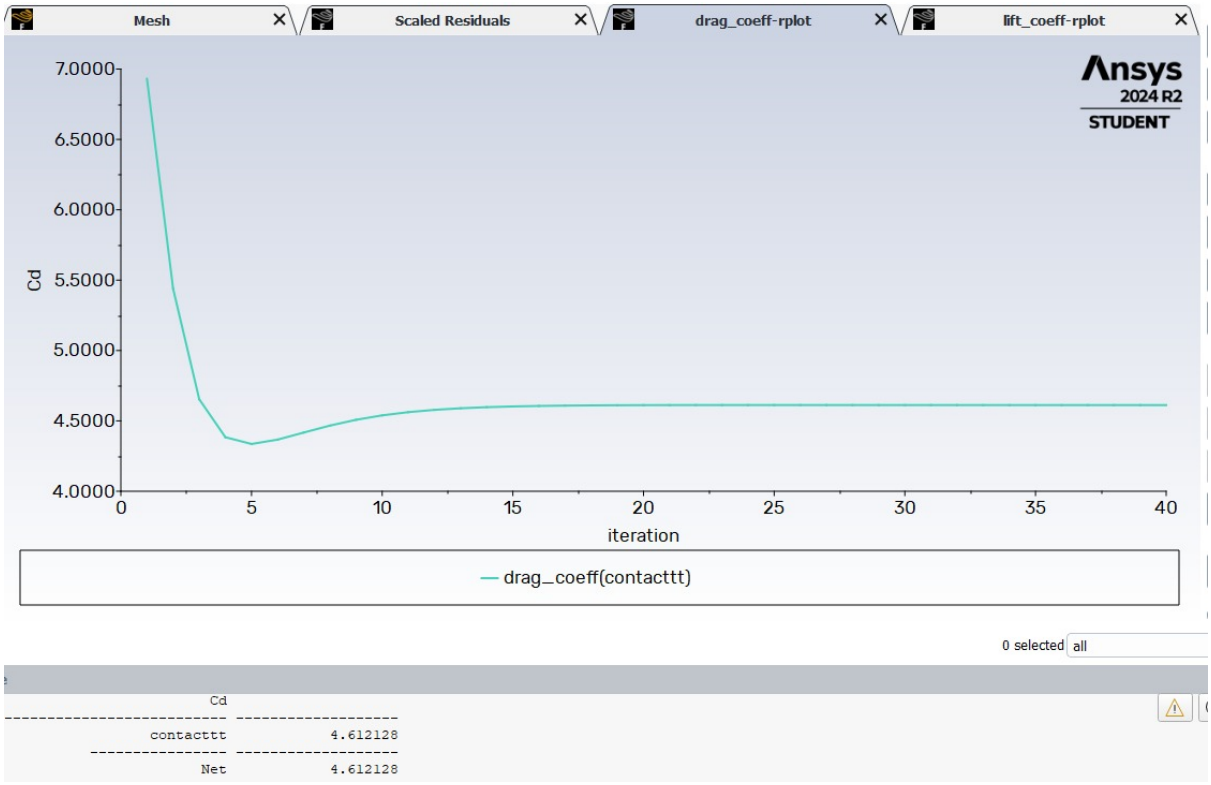

Figure 14: Drag around an ellipse at 90° at Re =100

We tried the same simulation for mesh size of 0.02

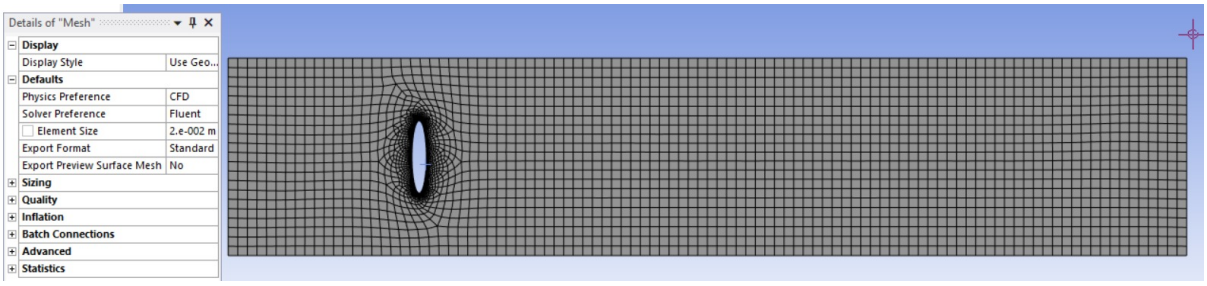

Figure 15: Geometry of an ellipse at 90° with mesh size 0.02

The graph shows the convergence behavior of the drag coefficient (Cd) as a function of the number of iterations during the simulation. At Reynolds number (Re) = 100, the Cd initially exhibits large fluctuations but stabilizes as the solution converges with increasing iterations. The final value of Cd is approximately 4.612, indicating a steady-state solution has been achieved.

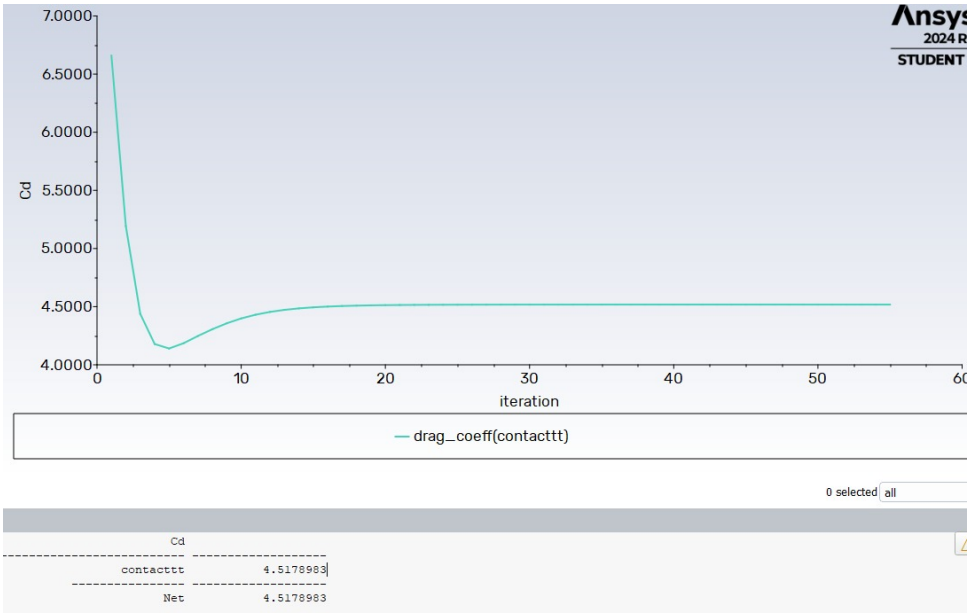

Figure 16: Drag around an ellipse at 90° at Re =100 with mesh size 0.02

The below graph shows the convergence behavior of the drag coefficient at the element size of 0.02 m length. The drag coefficient found out to be 4.517 which is almost same as at an element size 0.01.

At θ = 40 degrees After running the simulation we found the drag coefficient as 2.41. Grid independence test was also performed for different grid sizes namely at element sizes of 0.01 and 0.02 and the results obtained was almost same for the both tests.

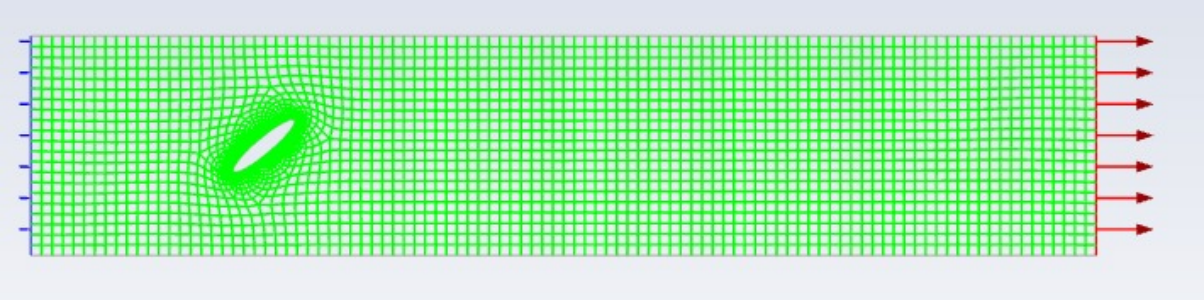

Figure 17: Boundary conditions for an ellipse at 40°

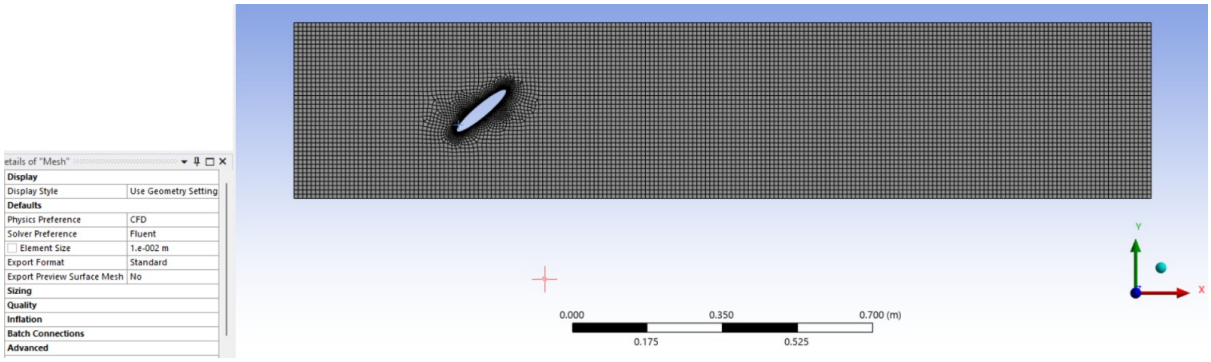

Figure 18: Drag around an ellipse at 40° at Re =100 with mesh size 0.01

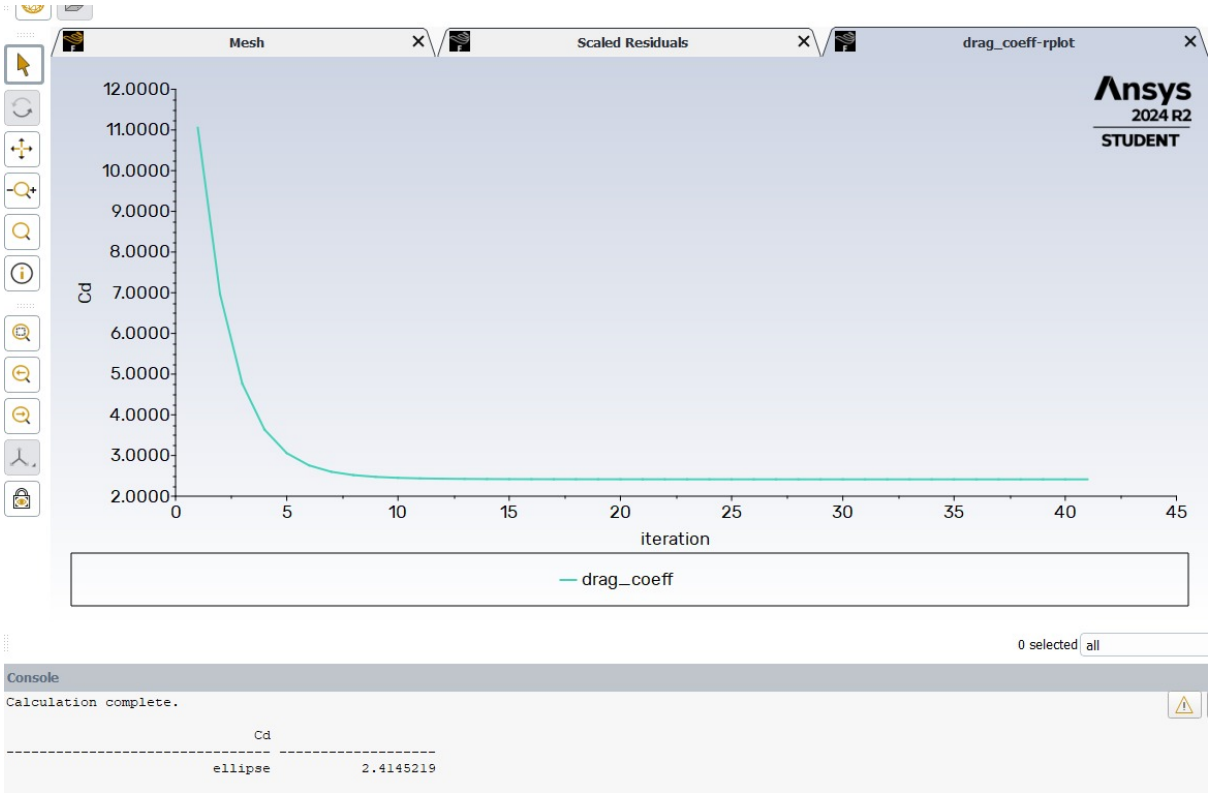

Figure 19: Drag around an ellipse at 40° at Re =100 with mesh size 0.01

The graph shows the convergence behavior of the drag coefficient (Cd) as a function of the number of iterations during the simulation. At Reynolds number (Re) = 100, the Cd initially exhibits large fluctuations but stabilizes as the solution converges with increasing iterations. The final value of Cd is approximately 2.41, indicating a steady-state solution has been achieved.

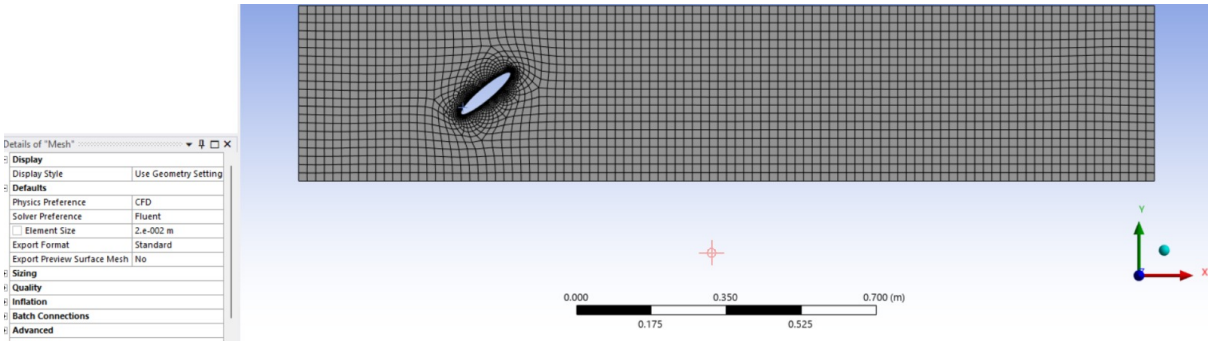

Figure 20: Drag around an ellipse at 40° at Re =100 with mesh size 0.02

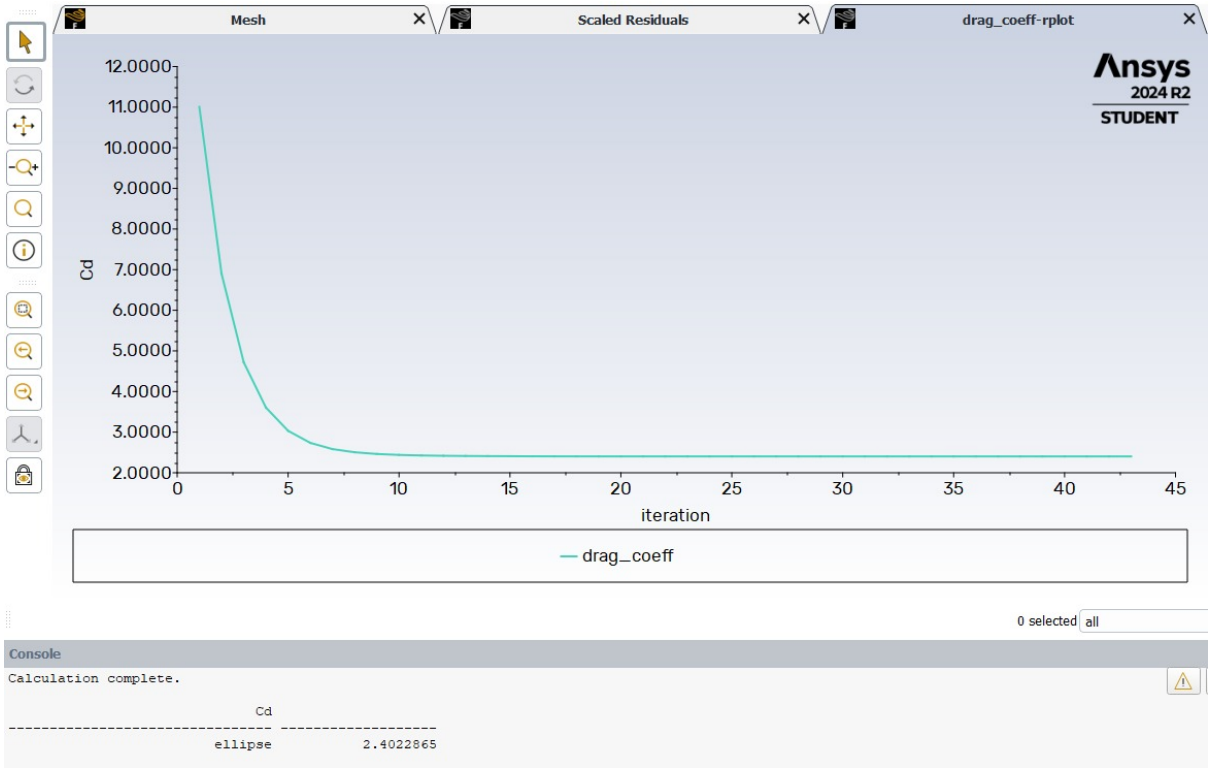

Figure 21: Drag around an ellipse at 40° at Re =100 with mesh size 0.02

The graph shows the convergence behavior of the drag coefficient at the element size of 0.02 m length. The drag coefficient found out to be 2.40 which is almost same as at element size 0.01.

Angle	Literature	ANSYS
$\theta = 90$	5.27	4.612
$\theta = 40$	2.41	2.618

Table 2: Drag coefficient comparison for elliptical particle

To validate the ANSYS results, we compared the drag coefficients from our results to the literature[7], which was acceptable to move forward with the problem. Moving forward, we simulated the translating ellipse with a constant velocity in a channel to calculate the drag forces acting on it.

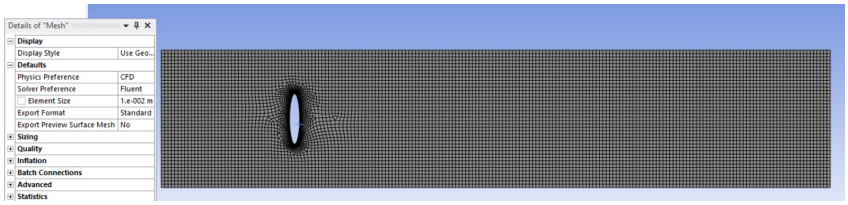

Figure 22: Geometry for an translating ellipse at 90° with mesh size 0.01

Using the following User-Defined Functions(UDFs) for the inlet velocity and the constant velocity of the elliptical particle, we made the particle move with a constant velocity.

```
#include "udf.h"
DEFINE_PROFILE(inlet_x_velocity, thread, position)
{
real x[ND_ND]; /* this will hold the position vector */
real y, h;
face_t f;
h = 0.41; /* inlet height in m */
begin_f_loop(f,thread)
{
F_CENTROID(x, f, thread);
y = 2.*((x[1]-0.326)-0.5*h)/h; /* non-dimensional y coordinate */
F_PROFILE(f, thread, position) = 0.5*(1.0-y*y);
}
end_f_loop(f, thread)
}
```

Figure 23: UDF for inlet velocity

```
#include "udf.h"

DEFINE_CG_MOTION(moving_cell, dt, vel, omega, time, dtime)
{
    /* Set constant velocity */
    vel[0] = 0.1;  /* Constant x-velocity */
    vel[1] = 0.0;  /* No movement in y-direction */

    /* No rotational motion */
    omega[2] = 0.0;

    /* Debug message */
    Message("Time: %f, Velocity: [%f, %f]\n", time, vel[0], vel[1]);
}
```

Figure 24: UDF for ellpise's constant velocity

We ran the simulation for 100 time steps with a time step size of 0.01 s and found the drag coefficient to be 1.587 and the drag force of 0.119 N acting on the particle.

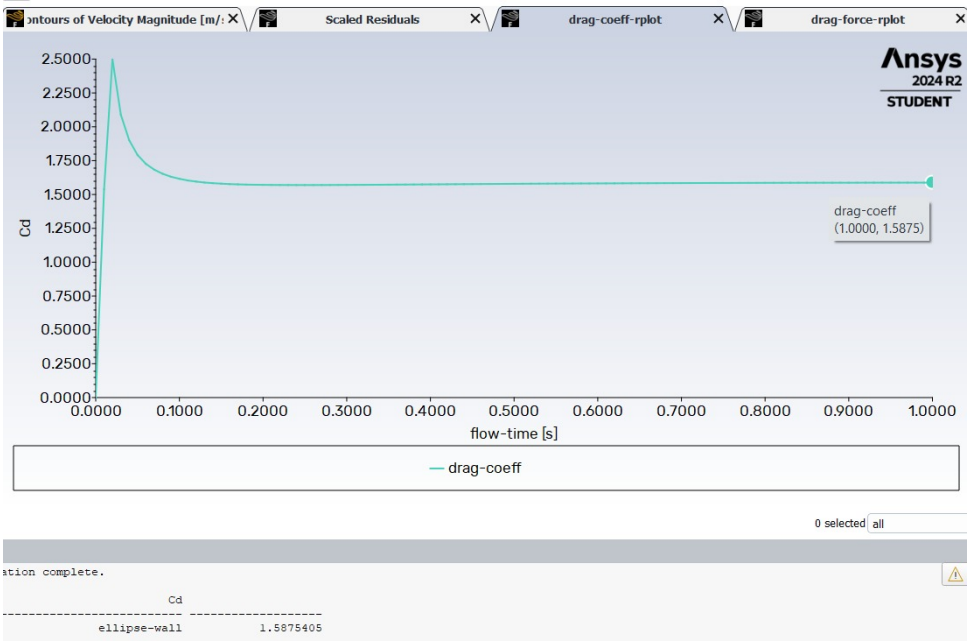

Figure 25: Drag-coefficient for moving ellipse

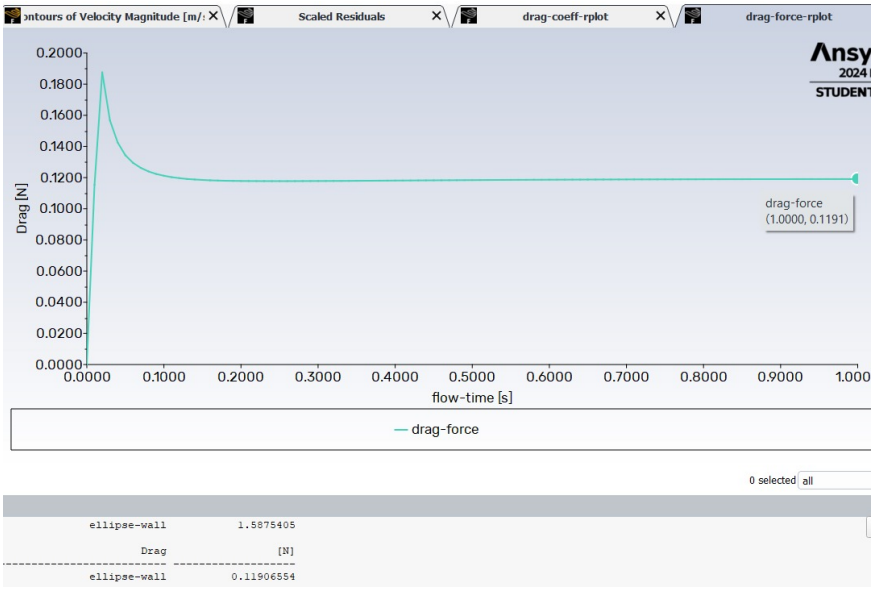

Figure 26: Drag-force for moving ellipse

We observed the flow field around the elliptical particle for a flow time of 1 s.

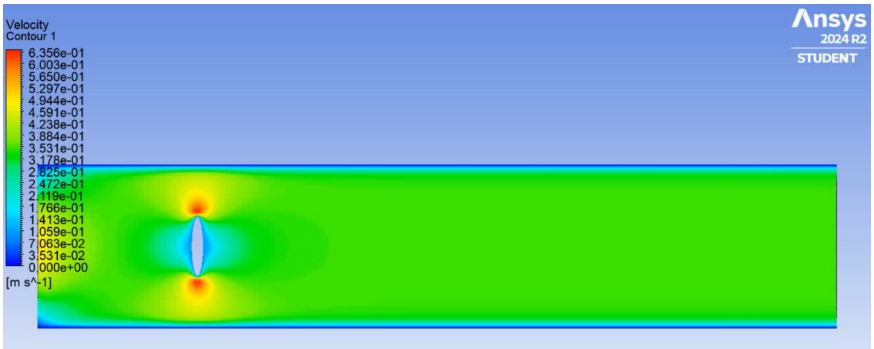

Figure 27: Initial position and the flow field for elliptical particle

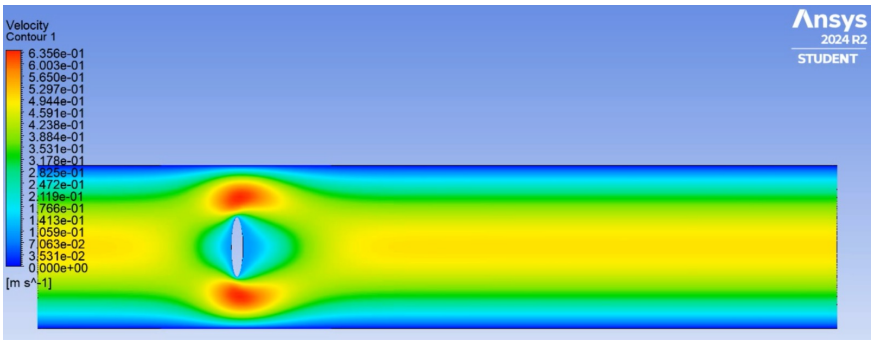

Figure 28: Position and the flow field for elliptical particle at t=1 sec

5 Conclusion

We initially understood the Immersed Boundary Method (IBM) and implemented it in MATLAB to simulate the motion of a mass point within a confined channel. To validate the code, we attempted to modify it to compute the drag force on a stationary rigid 2D cylinder. However, the results obtained from the MATLAB simulation did not match the values reported in the literature.

To investigate the cause for the error, we performed a similar simulation in ANSYS, which produced highly accurate results. This comparison helped us identify the source of the error in the MATLAB implementation. The issue found out to be the usage of periodic boundary conditions in the MATLAB code. These boundary conditions likely prevented the flow from fully developing before reaching the cylinder, which is a critical requirement for obtaining accurate drag force values. In contrast, the ANSYS simulation successfully achieved a steady-state flow and ensured the flow was fully developed before interacting with the cylinder, thereby yielding reliable and accurate results.

In this semester, we started to simulate the fluid structure interaction in Ansys Fluent. And we found a research paper on drag coefficients and forces over an ellipse at different orientations. And then we tried to compare our Results with the paper. First we evaluated the drag force and coefficient of a stationary ellipse and compared and validated with the literature. The values we got were very close to the paper. Then we simulated a moving ellipse inside a moving fluid confined in a rectangular channel and evaluated the drag coefficient and drag force.

6 Future Work

For the next phase of the project, We will try to rotate and translate the ellipse simultaneously and evaluate the hydrodynamic forces using user-defined functions. Then we will try to track the path of a particle with a user-defined function, which will calculate the force acting on the particle and based on these forces, we will analyze how these particle moves.

References

[1] Jun Zhang, Sheng Yan, Dan Yuan, Gursel Alici, Nam-Trung Nguyen, Majid Ebrahimi Warkiani, and Weihua Li. Fundamentals and applications of inertial microfluidics: A review. *Lab on a Chip*, 16(1):10–34, 2016.

[2] Charles S Peskin. Flow patterns around heart valves: a numerical method. *Journal of computational physics*, 10(2):252–271, 1972.

[3] nickabattista. Ib2d, 2024. Accessed: 2024-09-18.

[4] Nicholas A Battista, W Christopher Strickland, and Laura A Miller. Ib2d: a python and matlab implementation of the immersed boundary method. *Bioinspiration & biomimetics*, 12(3):036003, 2017.

[5] Nicholas A Battista, W Christopher Strickland, Aaron Barrett, and Laura A Miller. Ib2d reloaded: A more powerful python and matlab implementation of the immersed boundary method. *Mathematical Methods in the Applied Sciences*, 41(18):8455–8480, 2018.

[6] Ram Prakash Bharti, RP Chhabra, and V Eswaran. Two-dimensional steady poiseuille flow of power-law fluids across a circular cylinder in a plane confined channel: wall effects and drag coefficients. *Industrial & engineering chemistry research*, 46(11):3820–3840, 2007.

[7] M Taeibi-Rahni, V Esfahanian, and M Salari. Investigation of flow around a confined elliptical cylinder, using lattice boltzmann method. *Middle-East Journal of Scientific Research*, 15(1):08–13, 2013.