# Solving the Navier-Stokes Equations: A Finite Element Approach

Matteo Cenzato, Giuseppe Pisante, Arianna Procaccio

February 2, 2024

# Contents

**1 Mathematical model** **3**

**2 Objective** **4**

**3 Domain of Computation** **5**

**4 C++ Implementation** **5**
4.1 Finite Element Setup . . . . . 6
4.2 Assemble of the system . . . . . 6
4.3 Assemble of the preconditioner . . . . . 7
4.3.1 SIMPLE preconditioner . . . . . 7
4.3.2 aSIMPLE preconditioner . . . . . 8
4.3.3 Results and performance . . . . . 8
4.4 Solver Algorithms . . . . . 10
4.5 Computation of drag and lift forces . . . . . 10
4.5.1 Overview . . . . . 10
4.5.2 Computational Process . . . . . 10

**5 Test Cases** **11**
5.1 2D Test Cases . . . . . 11
5.1.1 Test Case 2D-1 (Steady) . . . . . 11
5.1.2 Test Case 2D-2 (Unsteady) . . . . . 11
5.1.3 Test Case 2D-3 (Unsteady) . . . . . 14
5.2 3D Test Cases . . . . . 18
5.2.1 Test Cases unsteady 1 . . . . . 18
5.2.2 Test Cases unsteady 2 . . . . . 20

**6 Parallelization** **21**

# Introduction

This report aims to solve the unsteady, incompressible Navier-Stokes equations using the finite element method. The focus is to simulate the benchmark problem "flow past a cylinder" in two and three dimensions considering different values of the Reynolds number ($Re \leq 200$).

The report will discuss the obtained results, the methods employed, their stability and accuracy, and the algorithmic and computational aspects.

# Navier-Stokes Equations

Considering the set $\Omega \in \mathbb{R}^d$ (d = 2,3), the system of Navier-Stokes equations to be solved is given by:

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \nu \Delta \mathbf{u} + \nabla p & = \mathbf{f} \quad \text{in } \Omega, \\ \nabla \cdot \mathbf{u} & = 0 \quad \text{in } \Omega, \\ \mathbf{u} & = \mathbf{g} \quad \text{on } \Gamma_D \subset \partial\Omega, \\ \nu \nabla \mathbf{u} \cdot \mathbf{n} - p & = \mathbf{h} \quad \text{on } \Gamma_N = \partial\Omega \setminus \Gamma_D, \\ \mathbf{u}(t=0) & = \mathbf{u}_0 \quad \text{in } \Omega. \end{cases} \tag{1}$$

In this report, $\mathbf{u} \in \mathbb{R}^d$ will refer to the velocity vector field, $p \in \mathbb{R}$ to the pressure, $\nu$ to the kinematic viscosity, and $\mathbf{f}$, $\mathbf{g}$, and $\mathbf{h}$ are given functions. The boundary conditions are applied on $\Gamma_D$ and $\Gamma_N$, and $\mathbf{u}_0$ is the initial condition.

# 1 Mathematical model

Let $[H^1(\Omega)]^d = \{v \in [H^1(\Omega)]^d \,|\, v = 0 \text{ on } \Gamma_D\}$, and let $V$ denote the space $[H^1_D(\Omega)]^d$, and $Q$ the space $L^2(\Omega)$. Then, multiplying by $v \in V$ and integrating over $\Omega$, and similarly for $q \in Q$, we obtain the weak form as follows: find $(u, p) \in V \times Q$ such that

$$\begin{cases} \displaystyle\int_\Omega \frac{\partial u}{\partial t} v \, d\Omega + \int_\Omega \nu \nabla u : \nabla v \, d\Omega + \int_\Omega ((u \cdot \nabla)u) \cdot v \, d\Omega - \int_\Omega p \nabla \cdot v \, d\Omega \\ \displaystyle = \int_\Omega f_{\text{ext}} v \, d\Omega + \int_{\partial X} g_N v \, ds \quad \forall v \in V, \\ \displaystyle\int_\Omega q \nabla \cdot u \, d\Omega = 0 \quad \forall q \in Q. \end{cases} \tag{2}$$

The weak form is discretized in time using a semi-implicit treatment of the nonlinear convective term, and the time derivative is discretized using a first-order finite difference. The problem is then discretized in space using stable finite elements. Let $X_h^r$ be the space of finite elements defined by

$$X_h^r = \{v_h \in C^0(\Omega_h) \,:\, v_h|_K \in P_r, \forall K \in \mathcal{T}_h\},$$

where $P_r$ is the space of polynomials of degree less than or equal to $r$, and $\mathcal{T}_h$ is a regular triangulation forming an approximation $\mathcal{X}_h^r$ of $\Omega$. Then, the discrete spaces for velocity and pressure are denoted by $V_h = [\mathcal{X}_h^r]^d \cap V$ and $Q_h = \mathcal{X}_h^q \cap Q$ respectively. Let $u_{n,h} \in V_h$ and $p_{n,h} \in Q_h$ be

approximations of the solutions $u$ and $p$ at time $t_n$. We denote by $\theta_1, \ldots, \theta_{N_u}$ the finite element basis of $V_h$ and by $\phi_1, \ldots, \phi_{N_p}$ the basis of $Q_h$.

Then, we can write

$$u_{n,h}(x) = \sum_{i=1}^{N_u} u_{n,i}\theta_i(x), \quad p_{n,h}(x) = \sum_{i=1}^{N_p} p_{n,i}\phi_i(x).$$

We approximate $(u, p)$ at time $t_n$ by $u_{n,h}, p_{n,h} \in V_h \times Q_h$. Thanks to the Galerkin projection and by taking $v(x) = \theta_i$ and $q = \phi_j$, we obtain the discrete system

$$\begin{cases} \dfrac{1}{\Delta t}\displaystyle\int_{\Omega_h} u_h^{n+1}\theta_i \, dx + \int_{\Omega_h} \nabla u_h^{n+1} : \nabla\theta_i \, dx \\ \quad + \displaystyle\int_{\Omega_h} \left((u_h^n \cdot \nabla)u_h^{n+1}\right) \cdot \theta_i \, dx - \int_{\Omega_h} p_h^{n+1}\nabla \cdot \theta_i \, dx \\ = \displaystyle\int_{\Omega_h} f_{\text{ext}}(t_{n+1})\theta_i \, dx + \int_{\partial\Omega_h} g_N(t_{n+1})\theta_i \, ds \quad \forall i = 1, \ldots, N_u, \\ \displaystyle\int_{\Omega_h} \phi_j \nabla \cdot u_h^{n+1} \, dx = 0 \quad \forall j = 1, ...., N_p. \end{cases} \tag{3}$$

At each time step, these equations can be rewritten as a linear system of the form $Ax = b$ with:

$$\begin{bmatrix} F & B^T \\ -B & 0 \end{bmatrix} x = b$$

with

$$x = \begin{bmatrix} U_{n+1} \\ P_{n+1} \end{bmatrix}, b = \begin{bmatrix} G \\ 0 \end{bmatrix},$$

$F = \frac{1}{\Delta t}M + A + C(U_n)$, where $M$ is the fluid mass matrix, $A$ is the stiffness matrix, and $C(U_n)$ represents the linearized convection terms of the momentum equation. $B$ and $B^T$ are the discretized counterparts of the divergence operator and the gradient operator, respectively. $U_{n+1} = (u_1, \ldots, u_{N_u})^T$ is the vector of velocity unknowns at time $t = t_{n+1}$, and $P_{n+1} = (p_1, \ldots, p_{N_p})^T$ is the vector of pressure unknowns.

$G$ is a known vector depending on $U_n$, $f$, $g_D$, and $g_N$. This setting serves as a simple yet representative scenario to test preconditioners for the Navier–Stokes equations. The considered discretization in time and space is adequate for testing various preconditioners with different benchmark problems.

# 2 Objective

The primary objective of this study is to employ a Finite Element approach to solve the system of equations described earlier. The focus is on utilizing different preconditioners, namely SIMPLE and aSIMPLE. Our analysis will be conducted on a designated test mesh, obtained with gmsh through a .geo file. The key performance metrics for evaluation will be the lift and drag coefficients plotted over time for varying Reynolds numbers.

The coefficients, denoted as $C_D$ (drag coefficient) and $C_L$ (lift coefficient), are mathematically defined as:

$$C_D = \frac{F_D}{U^2 L}, \quad C_L = \frac{F_L}{U^2 L} \tag{4}$$

Here, $F_D$ and $F_L$ represent the forces acting on the cylindrical obstacle in the parallel and perpendicular directions to the fluid flow, respectively.The drag and lift forces are

$$F_D = \int_S \left( \rho\nu \frac{\partial v_t}{\partial n} n_y - P n_x \right) dS,$$
$$F_L = -\int_S \left( \rho\nu \frac{\partial v_t}{\partial n} n_x + P n_y \right) dS,$$

with the following notations: circle $S$, normal vector $n$ on $S$ with x-component $n_x$ and y-component $n_y$, tangential velocity $v_t$ on $S$, and tangent vector $t = (n_y, -n_x)$. The drag and lift coefficients are

$$c_D = \frac{2F_D}{\rho U^2 D},$$
$$c_L = \frac{2F_L}{\rho U^2 D}.$$

The parameters $U$ and $D$ stand for the reference flow velocity and the cross-sectional length of the obstacle. This investigation aims to provide a comprehensive understanding of the fluid dynamics using advanced numerical methods and preconditioners.

# 3 Domain of Computation

The domain of computation will be both 2D and 3D as shown in figure 1 and figure 2: For the 2D

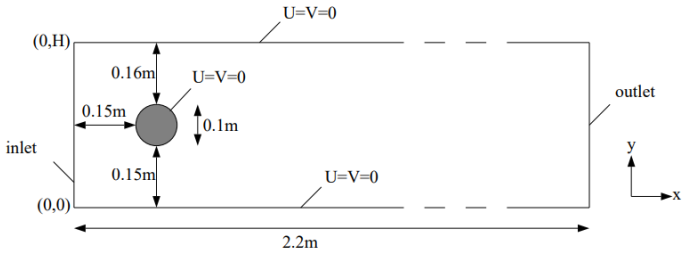



Figure 1: 2D Domain

test cases the flow around a cylinder with circular cross–section is considered. Some definitions are introduced to specify the values which have to be computed. $H = 0.41\,\mathrm{m}$ is the channel height and $D = 0.1\,\mathrm{m}$ is the cylinder diameter. The Reynolds number is defined by $Re = \frac{UD}{\nu}$ with the mean velocity $U(t) = \frac{2U(0,H/2,t)}{3}$.

# 4 C++ Implementation

The code exhibits a well-structured and object-oriented design, leveraging the capabilities of the Deal.ii library. The primary class, `NavierStokes`, encapsulates the entire simulation process, fos-

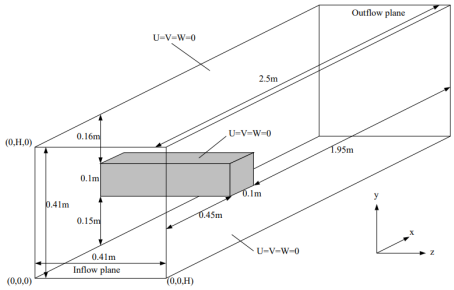



Figure 2: Geometry of 3D test cases with boundary conditions

tering modularity and ease of maintenance. The use of nested classes for the forcing term, boundary conditions, and initial conditions enhances the code's readability.

## 4.1 Finite Element Setup

The finite element setup is a crucial aspect of the code, initiated by the `setup` method. This method initializes the simulation by creating a mesh, defining finite element spaces, and setting up matrices and vectors for the linear system. The mesh is read from a .msh file, and the degrees of freedom for velocity and pressure are specified.

## 4.2 Assemble of the system

The assembly of the system in the Navier-Stokes solver involves the computation of various matrices and vectors that form the linear system of equations representing the discretized Navier-Stokes equations. The process is carried out in the `NavierStokes::assemble()` function. Let's break down the key steps:

- **Initialization:** The function initializes necessary parameters such as the number of degrees of freedom per cell (`dofs_per_cell`) and the number of quadrature points for the volume (`n_q`) and faces (`n_q_face`). It also sets up `FEValues` and `FEFaceValues` objects for accessing finite element values and face values.
- **Matrix and Vector Initialization:** Matrices (`cell_matrix`, `cell_pressure_mass_matrix`) and vectors (`cell_rhs`) are initialized for each cell. These represent the contributions of the cell to the global system matrix and right-hand side.
- **Iteration over Cells:** The assembly process iterates over all locally-owned cells in the finite element domain. For each cell:
  - **Finite Element Values:** Finite element values for velocity and pressure are extracted and stored.
  - **Integral Assembly:** The code performs the assembly of the Navier-Stokes system matrices and vectors. Here is a breakdown of the major components:
    - **Viscosity Term:** Computes the contribution of the viscosity term to the system matrix.

    - **Convective Term:** Calculates the convective term and adds it to the system matrix.
    - **Pressure Terms:** Handles pressure terms in the momentum and continuity equations.
    - **Pressure Mass Matrix:** Computes contributions to the pressure mass matrix.
    - **Forcing Terms:** Accounts for external forcing terms in the right-hand side vector.

    Integrals over the cell are computed for terms related to viscosity, pressure, and forcing. These contributions are added to the local cell matrices (`cell_matrix`, `cell_pressure_mass_matrix`) and vector (`cell_rhs`).
  - **Neumann Boundary Conditions:** Contributions from Neumann boundary conditions are incorporated into the right-hand side vector (`cell_rhs`).

- **Global Assembly:** The local contributions from each cell are added to the global system matrix (`system_matrix`), pressure mass matrix (`pressure_mass`), and right-hand side vector (`system_rhs`).
- **Dirichlet Boundary Conditions:** Dirichlet boundary conditions are enforced by setting prescribed values at specific degrees of freedom.
- **Compression:** The assembled system matrices and vectors are compressed to remove zero entries and optimize storage.

This assembly process ensures that the Navier-Stokes equations are appropriately discretized and ready for the solution phase.

## 4.3 Assemble of the preconditioner

### 4.3.1 SIMPLE preconditioner

The Semi-Implicit Method for Pressure Linked Equations (SIMPLE) is a numerical technique employed in fluid dynamics simulations. This method initially solves the momentum equation and subsequently updates the pressure field and velocity field to conserve mass through the continuity equation. The SIMPLE method can be interpreted as if it were associated with a preconditioner denoted as $P_{\text{SIMPLE}}$. This preconditioner is expressed as the product of two matrices:

$$P_{\text{SIMPLE}} = \begin{bmatrix} F & 0 \\ B & -\hat{S} \end{bmatrix} \begin{bmatrix} I & D^{-1}B^T \\ 0 & \alpha I \end{bmatrix},$$

where $D$ represents the diagonal of matrix $F$, and $\alpha \in (0, 1]$ is a parameter damping the pressure update. The matrix $\hat{S}$ is defined as:

$$\hat{S} = BD^{-1}B^T. \tag{5}$$

This preconditioner, encapsulated in the `PreconditionSIMPLE` class, aims to efficiently solve the linear system arising from the Navier-Stokes discretization.

The `initialize` method sets up the preconditioner by taking the sparse matrices `F`, `B`, and `B_t` as input. It constructs the inverse of the diagonal of `F` and initializes the matrices `S_tilde`, `D_inv`,

and preconditioners for `F` and `S_tilde`. The inverse of the diagonal is computed both as a vector and stored in the sparse matrix `D_inv`.

The `vmult` method applies the preconditioner to a given block vector. It employs iterative solvers, specifically GMRES (Generalized Minimal Residual) for the pressure-velocity coupling in `F` and CG (Conjugate Gradient) for solving the modified pressure equation in `S_tilde`. The overall process adheres to the SIMPLE algorithm, which iteratively updates velocity and pressure fields to satisfy the discretized Navier-Stokes equations.

The class includes a constant `alpha` with a value of 0.5, representing a scaling factor. Additionally, matrices `F`, `B_T`, and `B` represent discretized forms of the Navier-Stokes equations, while `S_tilde` and `D_inv` are auxiliary matrices. Preconditioners for `F` and `S_tilde` are instantiated as `preconditioner_F` and `preconditioner_S` respectively.

The method employs solver controls (`SolverControl`) to set maximum iterations (`maxiter`) and convergence tolerance (`tol`). The choice of GMRES and CG solvers enhances the stability and efficiency of the preconditioned iterative solution process.

#### 4.3.2 aSIMPLE preconditioner

The approximate SIMPLE (aSIMPLE) preconditioner is derived from the factorization of the SIMPLE preconditioner. This method starts with the factorized form of the SIMPLE preconditioner and replaces the inverses of the algebraic operators with suitable approximations. The aSIMPLE preconditioner is expressed as the product of several matrices:

$$P_{\text{aSIMPLE}} = \begin{bmatrix} D^{-1} & 0 \\ 0 & I \end{bmatrix} \cdot \begin{bmatrix} I & B^T \\ 0 & I \end{bmatrix} \cdot \begin{bmatrix} D & 0 \\ 0 & \frac{1}{\alpha} I \end{bmatrix} \cdot \begin{bmatrix} I & 0 \\ 0 & -\hat{S}^{-1} \end{bmatrix} \cdot \begin{bmatrix} I & 0 \\ -B & I \end{bmatrix} \cdot \begin{bmatrix} \hat{F}^{-1} & 0 \\ 0 & I \end{bmatrix}, \tag{6}$$

where $\hat{S}$ and $\hat{F}$ are approximations of the Schur complement and the factorized form of the SIMPLE preconditioner. The approximate SIMPLE method requires the construction of two approximations, denoted as $\hat{S}^{-1}$ and $\hat{F}^{-1}$. These approximations are typically defined based on suitable preconditioners for $\hat{S}^{-1}$ and $F$. In our implementation ILU preconditioners where chosen.

Regarding the c++ implementation, the `initialize` method sets up the preconditioner, taking sparse matrices `F`, `B`, and `B_t`, as well as the solution vector `sol_owned`, as input. The diagonal inverse and diagonal vectors (`diag_D_inv` and `diag_D`) are constructed from the inverse of the diagonal of `F`. The matrix `S` is computed, and preconditioners for `F` and `S` are initialized.

The `vmult` method applies the aSIMPLE preconditioner to a given block vector. It utilizes iterative solvers, specifically GMRES for solving the momentum equation in `F` and CG for solving the modified pressure equation in `S`.

The class includes constants such as `alpha` with a value of 0.5, representing a scaling factor. The matrices `F`, `B_T`, and `B` represent discretized forms of the Navier-Stokes equations. Matrices `S`, `diag_D`, and `diag_D_inv` serve as auxiliary structures in the preconditioning process.

Solver controls (`SolverControl`) are employed to set maximum iterations (`maxiter`) and convergence tolerance (`tol`). The use of GMRES and CG solvers enhances stability and computational efficiency during the iterative solution process.

#### 4.3.3 Results and performance

The ‘PreconditionaSIMPLE‘ preconditioner exhibits some improvements in construction speed compared to its predecessor, ‘PreconditionSIMPLE‘as shown in figure 3.

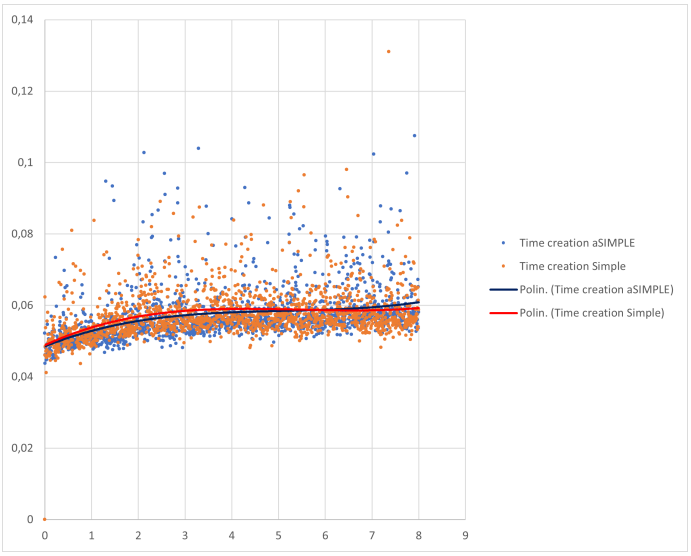



Figure 3: Time taken to initialize the preconditioner at each time step.

In addition, the time to solve each time step was compared in figure 4.


Figure 4: Time taken to solve the system at each time step.

Such image shows a slightly better performance of the SIMPLE preconditioner. This could be due to the fact that the SIMPLE preconditioner better approximates $A^{-1}$. $In fact, since the aSIMPLE is an approximation$
Finally, the number of iterations to solve the system at each time step are comparable.

## 4.4 Solver Algorithms

The code implements the time step solver for the Navier-Stokes equations using the GMRES iterative solver. Here's a breakdown of key components:

- **Solver Configuration:** Sets up the GMRES solver with a specified maximum number of iterations (`maxiter`) set to 100000 and a tolerance based on the L2 norm of the right-hand side (`tol`).
- **Preconditioner Initialization:** Initializes a custom preconditioner (commented-out options for other preconditioners are provided).
- **Solver Execution:** Solves the linear system using GMRES with the specified preconditioner.

The solution is updated for the next time step.

## 4.5 Computation of drag and lift forces

The presented C++ code captures the essence of force computation within a Navier-Stokes solver. Developed within the context of the deal.II library, this method, `compute_forces`, plays a pivotal role in understanding the interaction between fluid flow and boundaries related to the bluff body.

### 4.5.1 Overview

The primary goal of this method is to compute drag and lift forces exerted by the fluid on specified boundary faces. The solver leverages the Finite Element Method (FEM) for accurate modeling of fluid dynamics. The implementation utilizes the deal.II framework.

### 4.5.2 Computational Process

The method iterates over active cells, excluding non-locally owned cells. For each cell, finite element values are initialized, and relevant information, including velocity, pressure, and velocity gradients, is extracted. Special attention is given to boundary faces related to the bluff body, whose ID varies from the 2D and 3D implementation as shown in the code.

The drag and lift forces are computed by integrating fluid properties over the boundary faces. The method incorporates the normal and tangent vectors to accurately model the interaction between the fluid and the boundaries. The contributions from each quadrature point are accumulated in `local_drag` and `local_lift`. This is done to be able to execute such method in parallel, integrating the MPI parallelization by summing the local contributions accross different processes. Finally, the mean velocity, useful for the computation of the drag and lift coefficients is computed in the `InletVelocity` class through the `getMeanVelocity()` method.

# 5 Test Cases

## 5.1 2D Test Cases

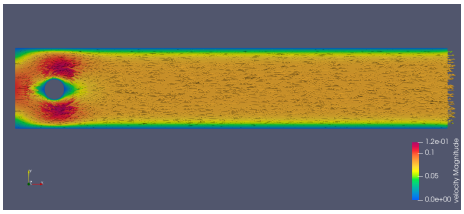
### 5.1.1 Test Case 2D-1 (Steady)

The inflow condition is given by:

$$U(0, y) = \frac{4U_m y(H - y)}{H^2}, \quad V = 0$$

with $U_m = 0.3\,\mathrm{m/s}$, yielding the Reynolds number $Re = 20$.

Pressure and velocity field over the fine mesh is shown in figure 5a.



(a) Velocity field

(b) Pressure field

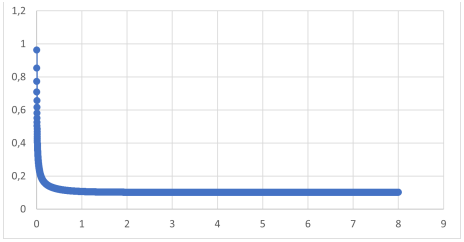
Velocity increases once flowing around the bluff body, thus keeping the mass flow rate constant. In addition, pressure decreases along the $x$-axis, accordingly with the energy dissipation due to viscous stresses.

### 5.1.2 Test Case 2D-2 (Unsteady)

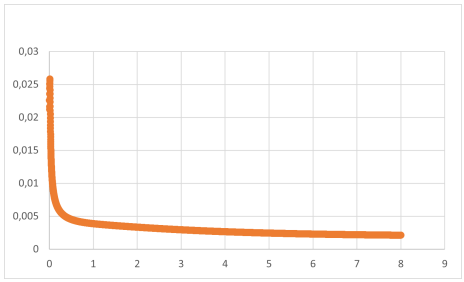
The inflow condition is given by:

$$U(0, y, t) = \frac{4U_m y(H - y)}{H^2}, \quad V = 0$$

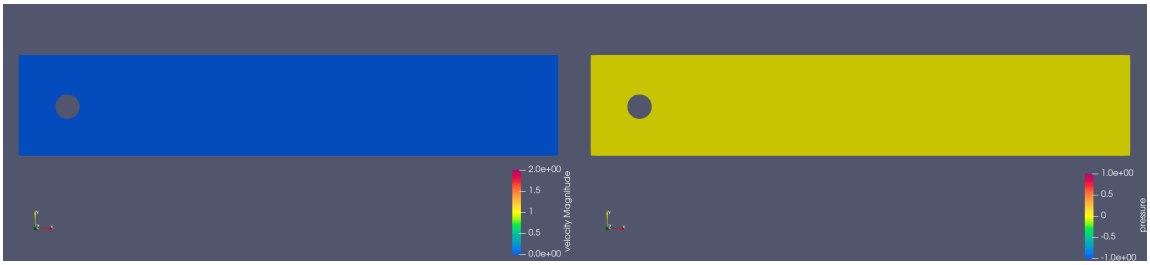
with $U_m = 1.5\,\mathrm{m/s}$, yielding the Reynolds number $Re = 100$. In addition, $c_D$ and $c_L$ are computed at each time step. In particular, the inlet velocity is kept constant throughout the whole period (0,T) and the initial velocity is null over the whole domain. The value of the lift coefficient tend to decrease over time, accordingly with the non-stationarity of the test as shown in figure 6a.Velocity and pressure transition over time are shown in the groups of figures below:



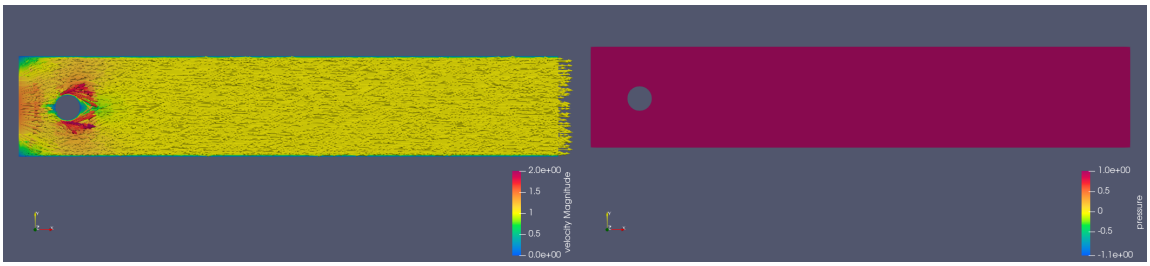

(a) Drag coefficient



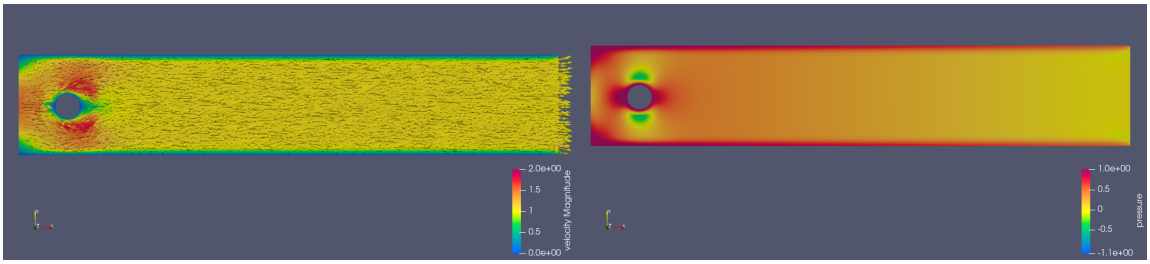

(b) Lift coefficient



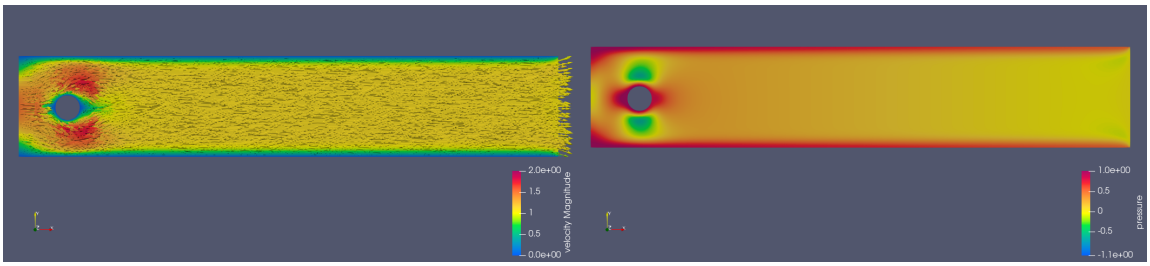

(a) Velocity at time t=0.001s (b) Pressure at time t=0.001s



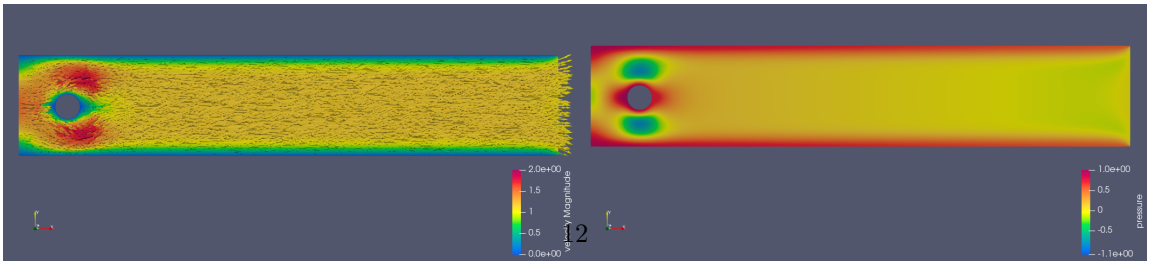

(c) Velocity at time t=0.010s (d) Pressure at time t=0.010s



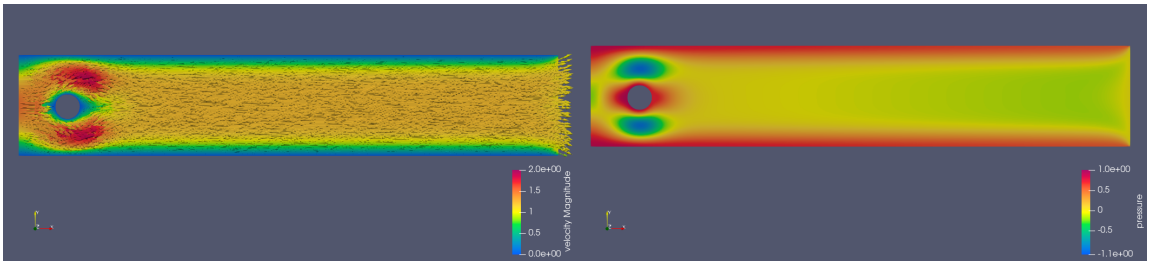

(e) Velocity at time t=0.150s (f) Pressure at time 0.15s



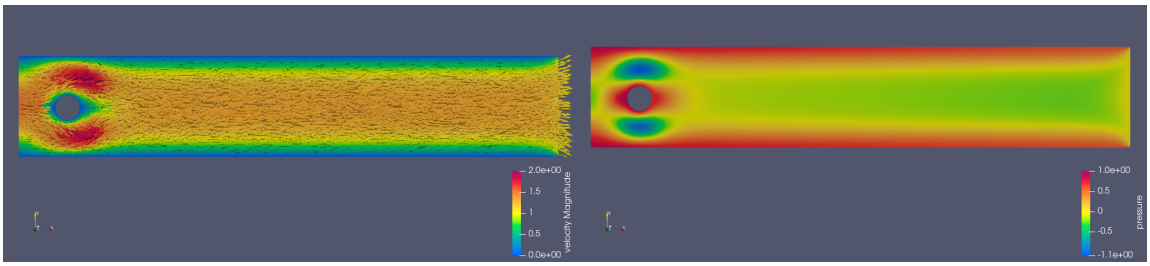

(g) Velocity at time t=0.3s (h) Pressure at time t=0.3s



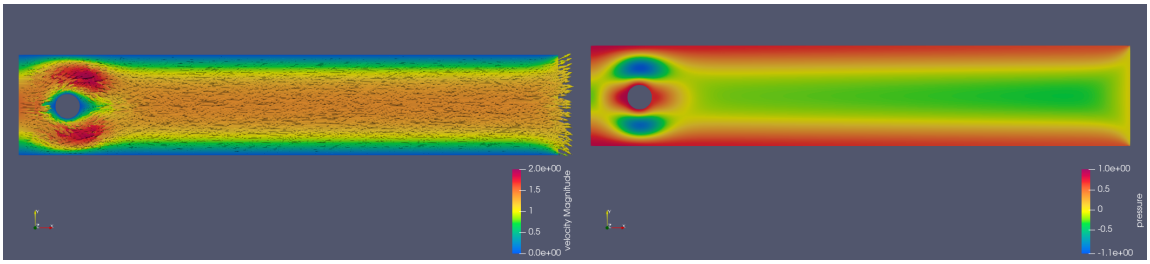

(i) Velocity at time t=0.6s (j) Pressure at time t=0.6s



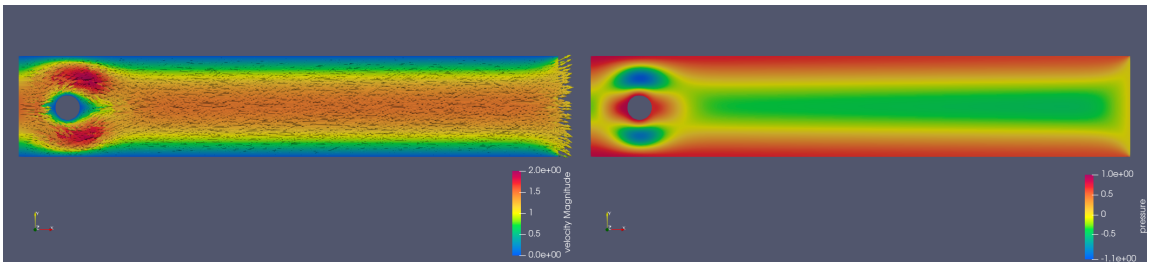

(a) Velocity at time t=1.2s (b) pressure at time t=1.2s



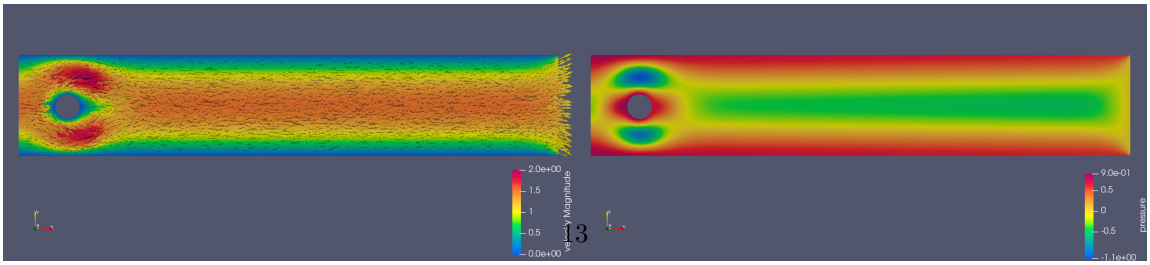

(c) Velocity at time t=2.0s (d) pressure at time t=1.5s




(e) velocity at time t=3.0s (f) pressure at time t=3.0s




(g) velocity at time t=5.5s (h) pressure at time t=5.5s




(i) velocity at time t=8.0s (j) pressure at time t=8.0s

This sequence of images shows the reason why this test cannot be considered stationary: the static initial condition combined with a high Reynolds number $Re$ implies a strong effect of the time derivative $\frac{\partial u}{\partial t}$ on the system matrix. Notably, the symmetry of the inlet velocity, bluff body, and boundary condition implies almost symmetric velocity and pressure fields with respect to the $x$-axis, implying a very small lift coefficient.

### 5.1.3 Test Case 2D-3 (Unsteady)

The inflow condition is given by:

$$U(0, y, t) = \frac{4U_m y(H - y) \sin\left(\frac{\pi t}{8}\right)}{H^2}, \quad V = 0$$

with $U_m = 1.5\,\mathrm{m/s}$, and the time interval is $0 \leq t \leq 8\,\mathrm{s}$. This gives a time-varying Reynolds number between $0 \leq Re(t) \leq 100$. The initial data ($t = 0$) are $U = V = P = 0$. The following quantities should be computed: drag coefficient $c_D$, lift coefficient $c_L$, and pressure difference $\Delta P$ as functions of time for $0 \leq t \leq 8\,\mathrm{s}$, maximum drag coefficient $c_{D_{\max}}$, maximum lift coefficient $c_{L_{\max}}$, and pressure difference $\Delta P(t)$ at $t = 8\,\mathrm{s}$.

#### Results

The experiments for this problem were carried out on a fine mesh using the aSIMPLE preconditioner. The following visualizations showcase the consistent and relatively lower time required for preconditioner initialization and the time taken to solve the system at each time step.

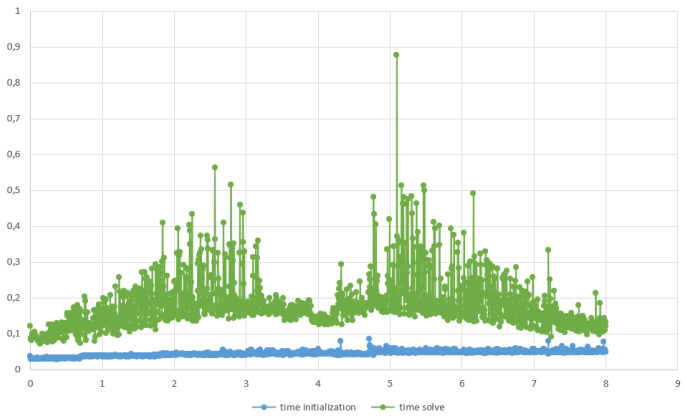
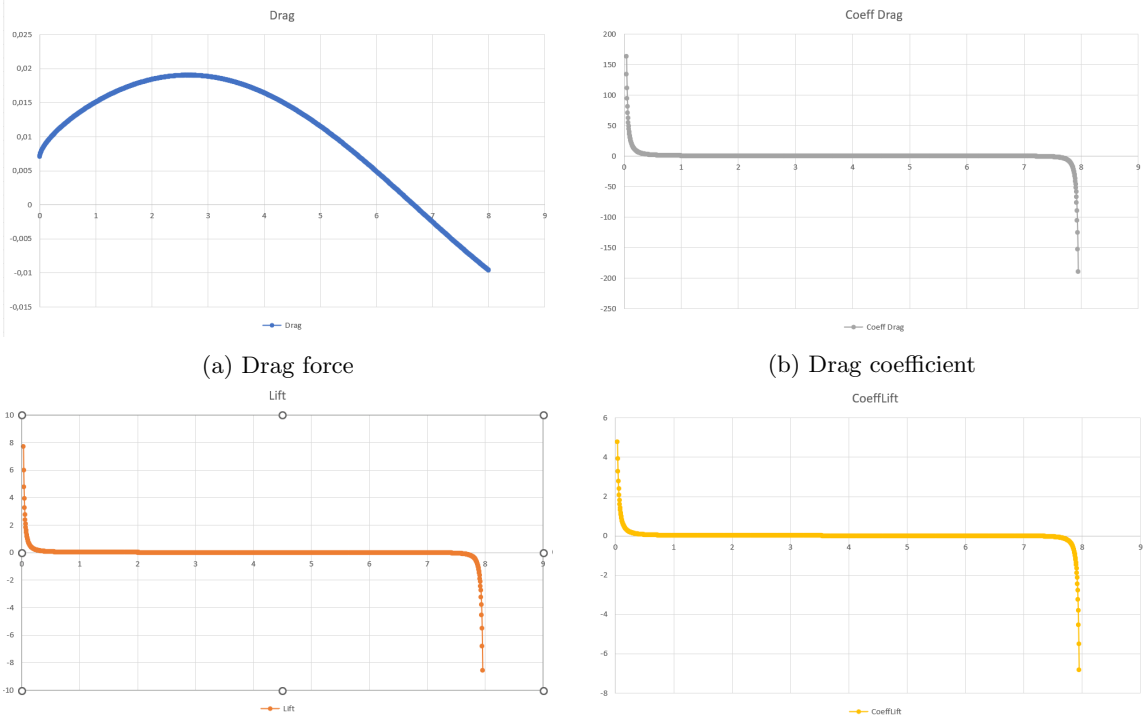



Figure 9: Time taken to initialize the preconditioner and solve the system at each time step.

Moving on, the subsequent images illustrate the forces of drag and lift, along with their coefficients, observed over the course of time steps.

Such results, particularly from $t = 4s$ onward, may be influenced by the fact that the inlet velocity undergoes a gradual decrease, indicating an overall deceleration of the flow throughout the domain. In contrast to the time interval $(0 - 4]s$, during which the pressure decreases along the x-axis, the pressure exhibits an increasing trend along the x-axis in the later time period.




(a) Drag force

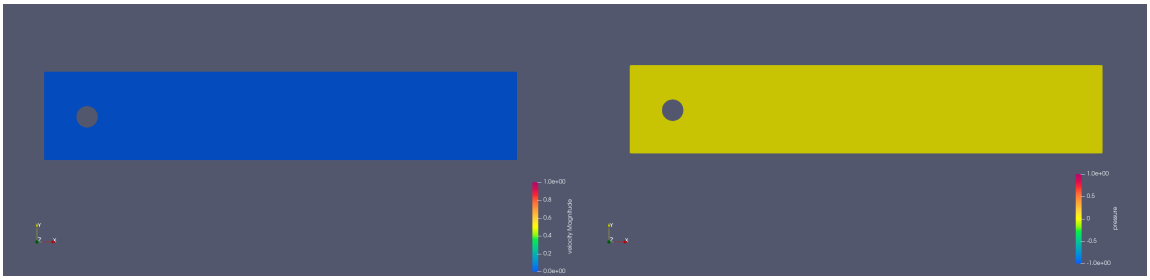
(b) Drag coefficient

(c) Lift force

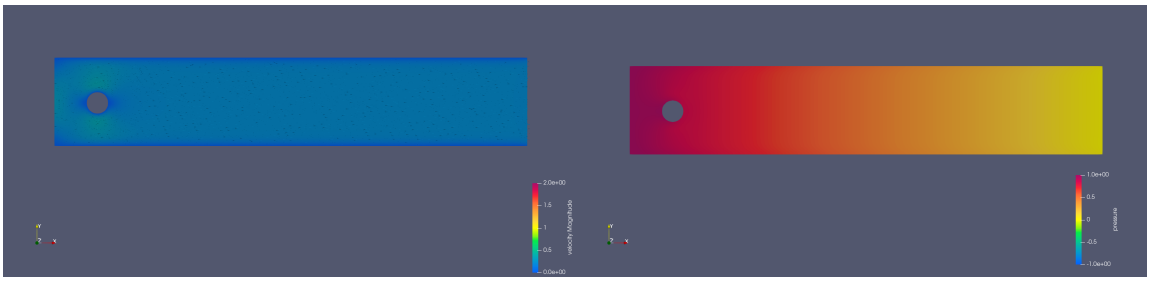
(d) Lift coefficient

Figure 10: Forces of drag and lift over time steps for different cases.

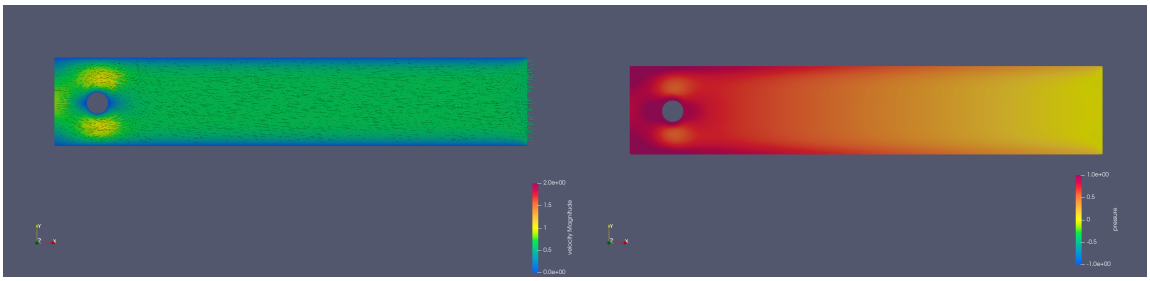
Finally, Velocity and pressure transition over time are shown in the groups of figures below:



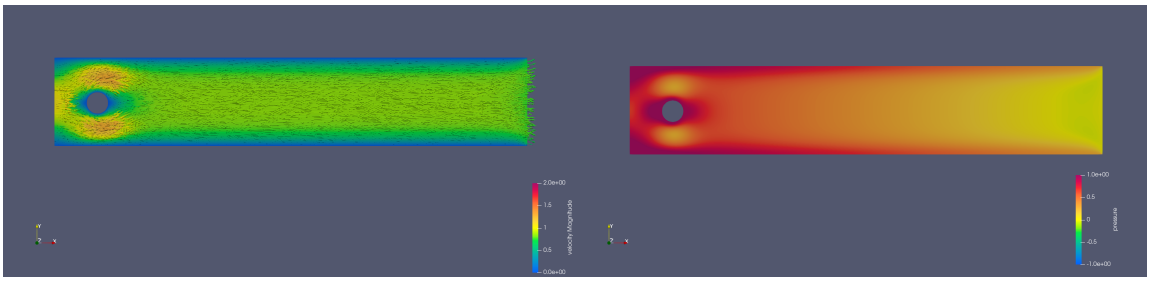
(a) Velocity at time t=0s (b) Pressure at time t=0s



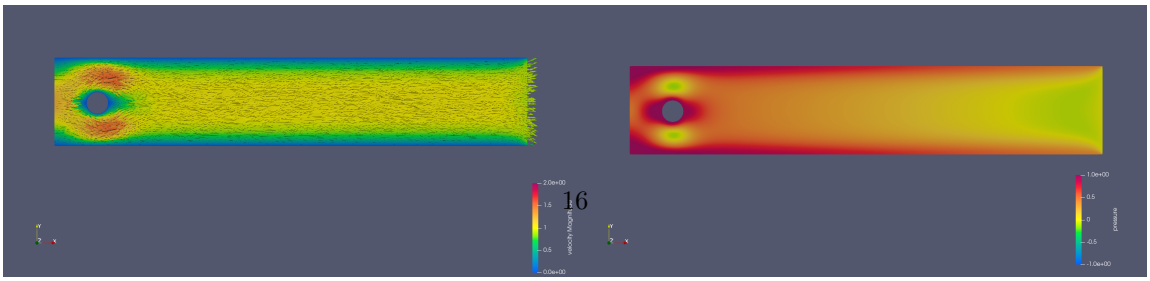
(c) Velocity at time t=0.5s (d) Pressure at time 0.5s



(e) Velocity at time t=1.5s (f) Pressure at time t=1.5s

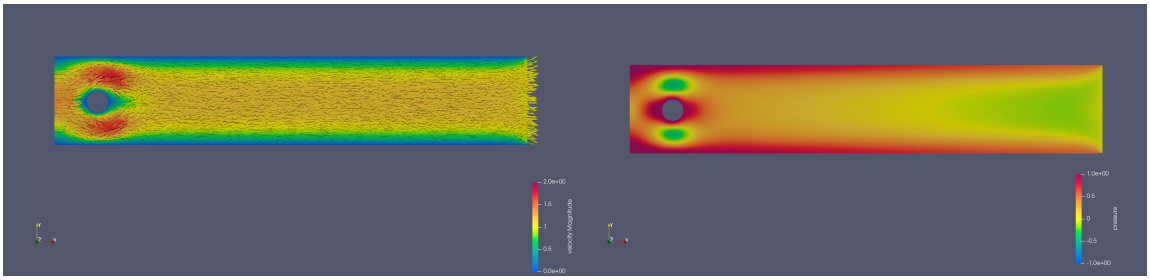


(g) Velocity at time $t = 2.0s$ (h) Pressure at time $t = 2.0s$



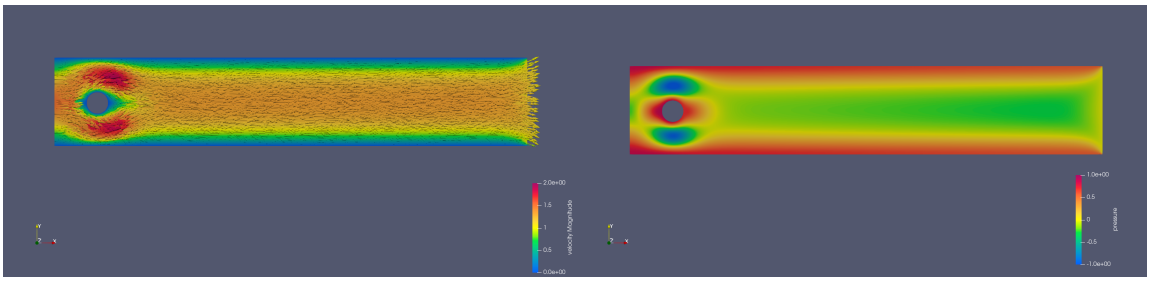
(i) Velocity at time t=2.5s (j) pressure at time t=2.5s

Figure 11: Velocity and pressure field with accelerating fluid



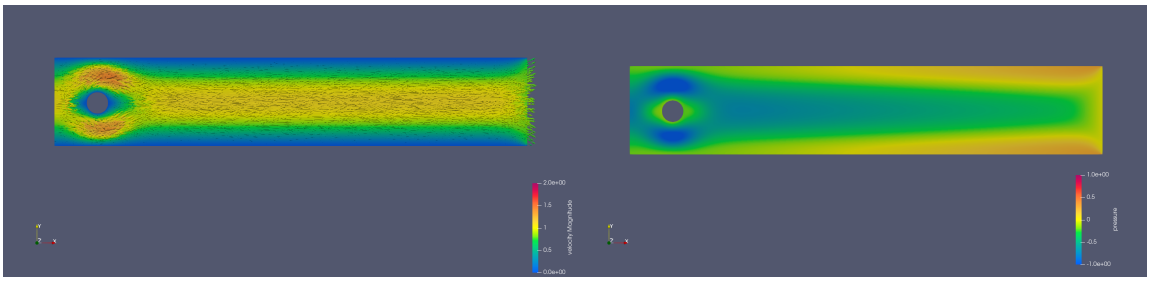
(a) Velocity at time t=3.0s

(b) pressure at time t=3.0s



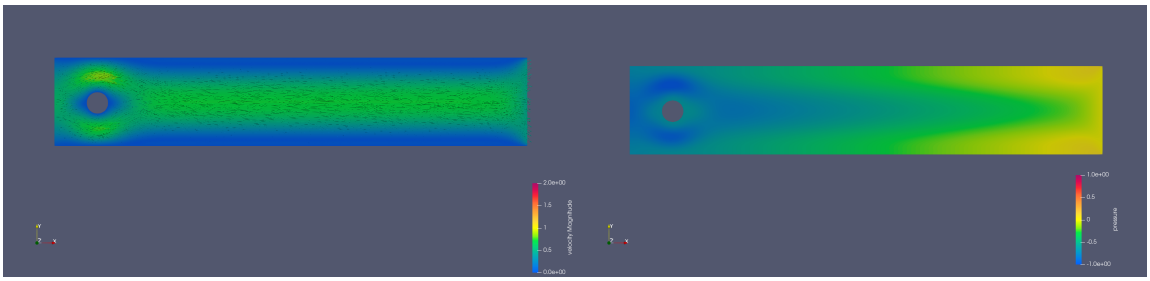
(c) Velocity at time t=4.0s

(d) pressure at time t=4.05s



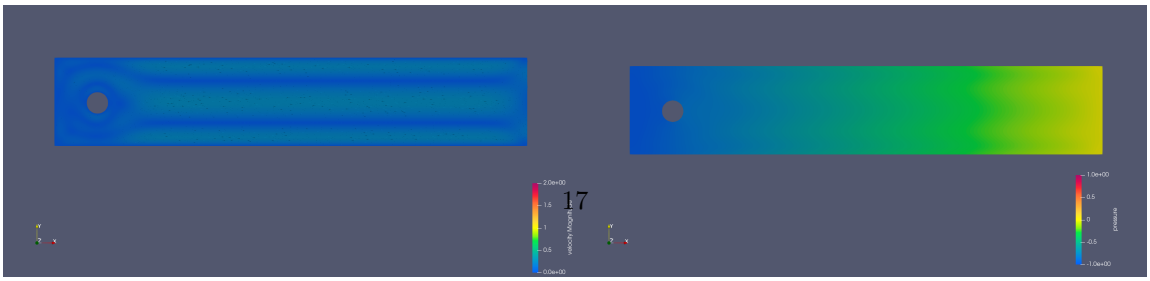
(e) velocity at time t=6.0s

(f) pressure at time t=6.0s



(g) velocity at time t=7.0s

(h) pressure at time t=7.0s



(i) velocity at time t=8.0s

(j) pressure at time t=8.0s

## 5.2 3D Test Cases

For the 3D test cases, the definitions above remain the same except for the tangent vector, which becomes $t = (n_y, -n_x, 0)$.

### 5.2.1 Test Cases unsteady 1

The inflow condition is given by:

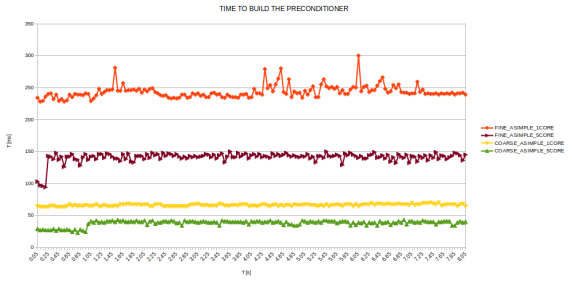
$$U(0, y, z) = \frac{16U_m yz(H - y)(H - z)}{H^4}, \quad V = W = 0$$

with $U_m = 0.45\,\mathrm{m/s}$, yielding the Reynolds number $Re = 20$.

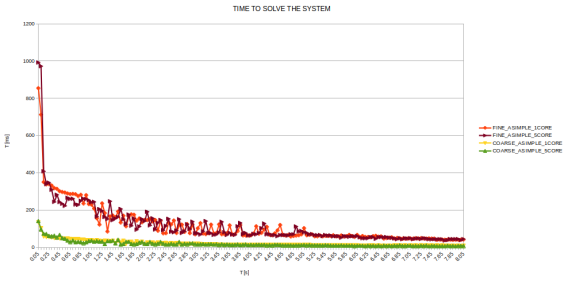
#### Results

To solve this problem, we exclusively used the aSIMPLE preconditioner with $\Delta t = 0.05$ in a time interval $0 \leq t \leq 8$.

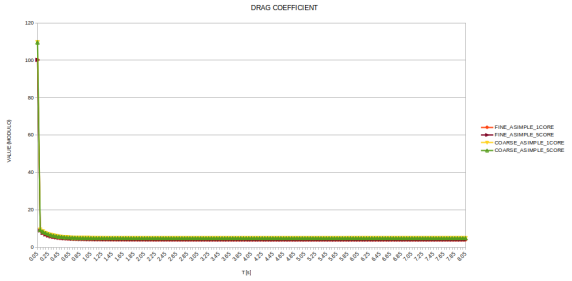
For each time step, we present the time taken to construct the preconditioner, the time taken to solve the problem, and both the drag and lift coefficients. The parallel implementation utilized 5 physical cores, and the tests were conducted on both coarse and fine meshes.



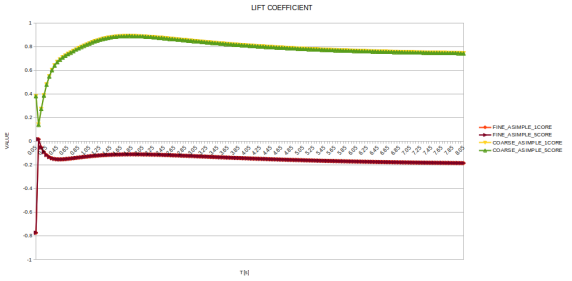

(a) Time taken to initialize the preconditioner at each time step.




(b) Drag coefficient over time.




(c) Drag coefficient over time.




(d) Lift coefficient over time.

Observations indicate that the time required to build the preconditioner remains consistent with a notable difference between single and multicore. The time to solve the problem varies significantly

instead, it becomes even smaller then the preconditioner. This variation can be attributed to the steady nature of the problem, where the solution stabilizes over time, necessitating fewer iterations for problem resolution.

No substantial differences were observed elsewhere; the drag and lift coefficients over time remained relatively consistent regardless of the mesh granularity.

We can observe also the difference in the time taken to solve the problem between the preconditioned and non preconditioned system in a coarse mesh:

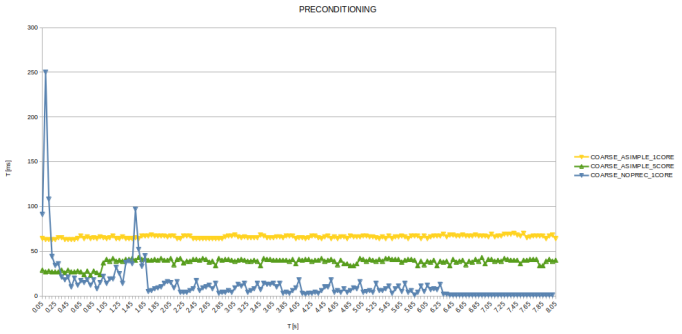
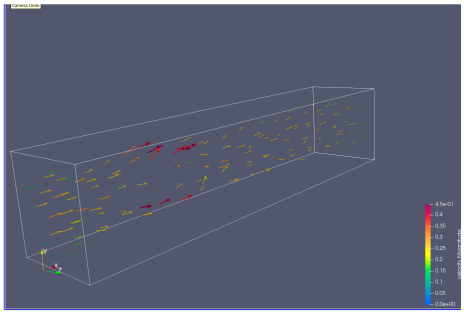


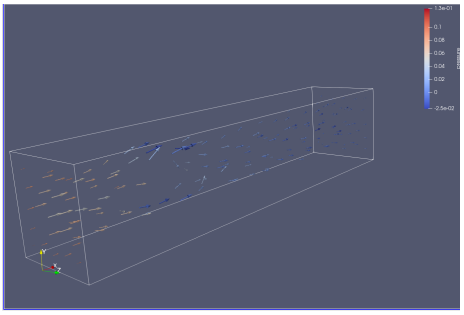

Figure 14: Differences between aSIMPLE and no preconditioning.

The interesting result is that for a coarse mesh the time taken to solve the problem is even lower then the preconditioned system. But there are 2 main problems with this statement: first, the time at $t = 0.05s$ (first step) is not shown because the actual time taken is $20s$ . Second, the system with no preconditioner is solved using a higher tollerance, because the norm of the residual in this case stabilizes around 0.001 after a few iterations and it does not go lower.

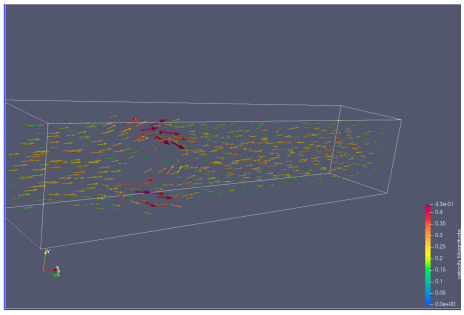
Below are some figures for reference taken at $t = 4.0s$ of the 3D solution for both a coarse and fine meshes.



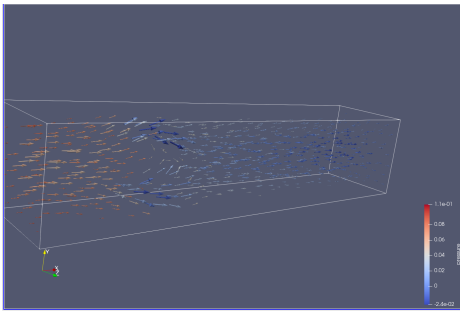
(a) Velocity field in a coarse mesh.



(b) Pressure field in a coarse mesh.



(c) Velocity field in a fine mesh.



(d) Pressure field in a fine mesh.

### 5.2.2 Test Cases unsteady 2

For the unsteady problems described in the reference paper as test case 3, no results of convergence were shown usign both a fine and coarse mesh, no matter the timestep used.

After more or less $t = 0.2s$ the solution would blow up, as shown in the figure below.

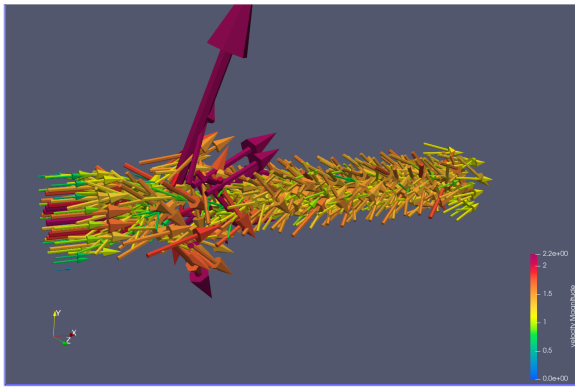


Figure 16: Velocity in an unsteady case at $t = 0.1s$ in a fine mesh. After this, the problem starts to diverge.

# 6 Parallelization

## Results

We conducted multiple runs of Test Case 2D-1 with $T = 2$, $\delta t = 0.01$ and the preconditioner aSIMPLE using different processor counts to assess the impact of parallelization on the code's performance. The results are as follows:

**6 processors** : The solution time is 76.4115 seconds, featuring an average preconditioner initialization time of 0.04 seconds and approximately 0.30 seconds for the initial time steps (around 15 GMRES iterations). Notably, the solution time significantly reduces towards the end. Without the preconditioner, the entire system requires about 149.708 seconds to solve. Notably, in the absence of the preconditioner, approximately 28979 iterations are performed in the initial steps, affirming the effectiveness of the preconditioner.

**2 processors** : In this configuration, the solution time is 160.326 seconds. During each time step, approximately 0.1 seconds are spent initializing the preconditioner, and an average of 0.4 seconds is required to solve the problem — still achieved in around 15 iterations.

**1 processor** : Operating the code in serial mode leads to a significant performance decrease, with the total time required to solve the system being 262.893 seconds. The initialization time for the preconditioner increases to around 0.135 seconds at every time step, and an average of 0.6 seconds is needed to solve the system.

# References


[1] Schäfer, S., Turek, F., Durst, F., Krause, E., and Rannacher, R., *Benchmark Computations of Laminar Flow Around a Cylinder*, Vieweg+Teubner Verlag, Wiesbaden, 1996.

[2] Deparis, S., Grandperrin, G., and Quarteroni, A., *Parallel Preconditioners for the Unsteady Navier–Stokes Equations and Applications to Hemodynamics Simulations*, Computers & Fluids, 92:253–273, 2014.