POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

PROJECT REPORT

Numerical Solution of Navier-Stokes Equations

NUMERICAL METHODS FOR PARTIAL DIFFERENTIAL EQUATIONS

Authors: Marzagora Bice, Tortorella Mara, Selis Riccardo

Academic year: 2023-2024

Abstract: This project focuses on simulating incompressible laminar flow around cylinders using the finite element method implemented using deal.II library coded in C++. Various preconditioners and MPI for parallelization are employed to improve performance and manage the extensive computational demands. This setup aims to produce predictions of fluid behavior, and measuring forces like drag and lift.

Contents

1. Mathematical formulation of the problem

1.1. Incompressible Unsteady Navier-Stokes Equations

The unsteady, incompressible Navier-Stokes equations describe the motion of a fluid, they form a system of nonlinear PDEs defined as follows:

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} - \nu \Delta \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p = \mathbf{f}, & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \\ \mathbf{u} = \mathbf{g}, & \text{on } \Gamma_D \\ \nu \dfrac{\partial \mathbf{u}}{\partial \mathbf{n}} - p\mathbf{n} = \mathbf{d}, & \text{on } \Gamma_N \\ \mathbf{u}|_{t=0} = \mathbf{u}_0, & \text{in } \Omega \end{cases} \tag{1}$$

$\mathbf{u} \in \mathbb{R}^d$ being the velocity vector field, $p \in \mathbb{R}$ the pressure, ν the kinematic viscosity, and $\mathbf{f}$ the forcing term which is a given function, same for $\mathbf{g}$, and $\mathbf{h}$. The boundary conditions are applied on Γ_D and Γ_N, and $\mathbf{u}_0$ is the initial condition. The term $(\mathbf{u} \cdot \nabla)\mathbf{u}$ describes the process of convective transport, and the condition $\nabla \cdot \mathbf{u} = 0$ guarantees that the fluid is incompressible.

1.2. Weak formulation

A weak formulation of problem (1) can be obtained by proceeding formally. Let us introduce the space functions:

$$V = [H^1_D(\Omega)]^d \quad \text{where} \quad H^1_D(\Omega) = \{f \in H^1(\Omega) : f = 0 \text{ on } \Gamma_D\} \tag{2}$$

$$Q = L^2(\Omega) \tag{3}$$

We multiply the first and second equation of (1) by a test function $v \in V$ and $q \in Q$ and integrating over the domain Ω, we obtain the weak form as follows:

find $(u(t), p(t)) \in V \times Q, u(0) = u_0, u = g$ on Γ_D, such that, for $t > 0$

$$\begin{cases} \displaystyle\int_\Omega \frac{\partial \mathbf{u}}{\partial t} \cdot \mathbf{v}\, d\Omega + \int_\Omega \nu \nabla \mathbf{u} : \nabla \mathbf{v}\, d\Omega + \int_\Omega [(\mathbf{u} \cdot \nabla)\mathbf{u}] \cdot \mathbf{v}\, d\Omega - \int_\Omega p \nabla \cdot \mathbf{v}\, d\Omega \\ \quad \displaystyle = \int_\Omega \mathbf{f}_{\text{ext}} \cdot \mathbf{v}\, d\Omega + \int_{\Gamma_N} \mathbf{d} \cdot \mathbf{v}\, ds \quad \forall \mathbf{v} \in \mathbf{V}, \\ \displaystyle\int_\Omega q \nabla \cdot \mathbf{u}\, d\Omega = 0 \quad \forall q \in Q \end{cases} \tag{4}$$

1.3. Galerkin Discretization

For the Galerkin method, we approximate the velocity and pressure using finite-dimensional subspaces. Let $\mathbf{u}_h \in \mathbf{V}^h$ and $p_h \in Q^h$ be the finite-dimensional approximations of $\mathbf{u}$ and p, respectively.
Let's approximate Navier-Stokes using **Taylor-Hood** finite elements, so the spaces are defined such as:

$$\mathbf{V}_h = \left\{ \mathbf{v}_h \in [C^0(\Omega)]^d,\ \mathbf{v}_h|_K \in (\mathbb{P}^{k+1})^d\ \forall K \in \mathcal{T}_h,\ \mathbf{v}_h|_{\Gamma_D} = \mathbf{0} \right\}$$
$$Q_h = \left\{ q_h \in C^0(\Omega),\ q_h|_K \in \mathbb{P}^k\ \forall K \in \mathcal{T}_h,\ \int_\Omega q_h = 0 \text{ if } \Gamma_D = \partial\Omega \right\}$$

Find $(\mathbf{u}_h, p_h) \in \mathbf{V}^h \times Q^h$ such that for all $(\mathbf{v}_h, q_h) \in \mathbf{V}^h \times Q^h$:

$$\begin{cases} \displaystyle\int_\Omega \frac{\partial \mathbf{u}_h}{\partial t} \cdot \mathbf{v}_h\, d\Omega + \nu \int_\Omega \nabla \mathbf{u}_h : \nabla \mathbf{v}_h\, d\Omega + \int_\Omega [(\mathbf{u}_h \cdot \nabla)\mathbf{u}_h] \cdot \mathbf{v}_h\, d\Omega \\ \qquad\qquad \displaystyle - \int_\Omega p_h (\nabla \cdot \mathbf{v}_h)\, d\Omega = \int_\Omega \mathbf{f} \cdot \mathbf{v}_h\, d\Omega + \int_{\Gamma_N} \mathbf{d} \cdot \mathbf{v}_h\, d\Gamma, \\ \displaystyle\int_\Omega q_h (\nabla \cdot \mathbf{u}_h)\, d\Omega = 0 \end{cases} \tag{5}$$

We approximate $\mathbf{u}_h$ and p_h using basis functions $\boldsymbol{\phi}_i$ and $\boldsymbol{\phi}_j$ for the velocity field, and $\boldsymbol{\psi}_i$ for the pressure field.

$$\mathbf{u}_h = \sum_{i=1}^{n_u} \mathbf{u}_i \phi_i, \quad p_h = \sum_{j=1}^{n_p} p_j \psi_j \tag{6}$$

Substituting the basis expansions into the discrete weak formulation, we obtain a system of algebraic equations. Let $\mathbf{U}$ and $\mathbf{P}$ be the vectors of coefficients for $\mathbf{u}_h$ and p_h respectively, the latter defines a nonlinear differential algebraic equation (DAE) system given by:

$$\begin{cases} M\dfrac{dU}{dt} + AU + N(U) + B^T P = G, \\ \hfill BU = 0, \end{cases} \tag{7}$$

where M denotes the mass matrix defined as:

$$M \in \mathbb{R}^{N_u \times N_u}, \quad M_{ij} = \int_\Omega \boldsymbol{\phi}_i \cdot \boldsymbol{\psi}_j \, d\Omega, \tag{8}$$

1.4. Time Discretization

We consider an **semi-implicit backward Euler scheme** for the time-discretization, in which the linear part of the equation is advanced implicitly, while nonlinear terms explicitly:

$$\begin{cases} \dfrac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} - \nu \Delta \mathbf{u}^{n+1} + (\mathbf{u}^n \cdot \nabla)\mathbf{u}^{n+1} + \nabla p^{n+1} = \mathbf{f}^{n+1}, \quad \text{in } \Omega \\ \hfill \nabla \cdot \mathbf{u}^{n+1} = 0 \end{cases} \tag{9}$$

The handling of nonlinear terms is critical in determining both the computational complexity and the stability of the solution scheme. Specifically, a semi-implicit method has been selected to mitigate the computational demands associated with a fully implicit nonlinear term. The main advantage of this approach, compared to the fully implicit method, is that it results in a linear system that needs to be solved at each time step. For further details on the computational cost and stability of this method, refer to section 1.5.

The formal derivation of the linear system to be solved at each time step is as follows: Given a uniform time step Δt, let (u^n, p^n) represent an approximation of the exact solution $(u(t^n), p(t^n))$ at time $t^n = n\Delta t$. Let $\mathbf{u}_h^n \in V_h$ and $p_h^n \in Q_h$ be approximations of the solutions $\mathbf{u}$ and p at time t_n. Denote by $\varphi_1, \dots, \varphi_{N_u}$ the finite element basis of V_h and by $\phi_1, \dots, \phi_{N_p}$ the basis of Q_h. Then, the following can be written:

$$\mathbf{u}_h^n(\mathbf{x}) = \sum_{i=1}^{N_u} u_i^n \varphi_i(\mathbf{x}), \quad p_h^n(\mathbf{x}) = \sum_{i=1}^{N_p} p_i^n \phi_i(\mathbf{x}).$$

The approximation of $(\mathbf{u}, p)$ at time t_n is given by $(\mathbf{u}_h^n, p_h^n) \in V_h \times Q_h$. Thanks to the Galerkin projection of $V_h \times Q_h$, the discrete system is obtained as follows:

$$\begin{cases} \dfrac{1}{\Delta t}\displaystyle\int_{\Omega_h} \mathbf{u}_h^{n+1}\varphi_i \, d\Omega + \int_{\Omega_h} \nu \nabla \mathbf{u}_h^{n+1} : \nabla \varphi_i \, d\Omega \\ \quad + \displaystyle\int_{\Omega_h} [(\mathbf{u}_h^n \cdot \nabla)\mathbf{u}_h^{n+1}] \cdot \varphi_i \, d\Omega - \int_{\Omega_h} p_h^{n+1} \nabla \cdot \varphi_i \, d\Omega \\ = \displaystyle\int_{\Omega_h} \mathbf{f}_{\text{ext}}(t_{n+1}) \cdot \varphi_i \, d\Omega + \int_{\partial\Omega_h} \mathbf{d}(t_{n+1}) \cdot \varphi_i \, ds + \frac{1}{\Delta t}\int_{\Omega_h} \mathbf{u}_h^n \varphi_i \, d\Omega, \quad \forall i = 1, \dots, N_u, \\ \displaystyle\int_{\Omega_h} \phi_j \nabla \cdot \mathbf{u}_h^{n+1} \, d\Omega = 0, \hfill \forall j = 1, \dots, N_p. \end{cases} \tag{10}$$

At each time step, the linear system to be solved is given by:

$$\begin{bmatrix} \dfrac{M}{\Delta t} + A + N(\mathbf{U}^n) & B^T \\ -B & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ p^{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{G} \\ 0 \end{bmatrix} \tag{10}$$

Here, the matrices B and B^T represent the divergence and gradient operators, respectively. The right-hand side vector $\mathbf{G}$ accounts for external forces, boundary conditions, and values from the previous time step. Solving this system at each time step updates the velocity and pressure fields while ensuring the incompressibility constraint is maintained.

1.5. Considerations on Reynolds number

The Reynolds number is a dimensionless number used to compare fluid flow characteristics across different problems. It is defined as the ratio of inertial to viscous forces, given by the formula:

$$Re = \frac{|\mathbf{U}|L}{\nu}, \tag{11}$$

where L is the length of a channel where the fluid's flow is studied and $\mathbf{U}$ a representative fluid velocity. The Reynolds number measures the extent to which convection dominates over diffusion.
When $Re \ll 1$ the convective term $(\mathbf{u} \cdot \nabla)\mathbf{u}$ can be omitted, and the Navier-Stokes equations reduce to the so-called Stokes equations. On the other hand, if Re is large, problems may arise concerning uniqueness of the solution, the existence of stationary and stable solutions, the possible existence of strange attractors, the transition towards turbulent flows when fluctuations of flow velocity occur at very small spatial and temporal scales, their numerical approximation becomes very difficult, if not impossible.

1.6. Computational cost, stability and convergence rate

The convergence rates of the numerical methods in this formulation are determined by the spatial discretization and time-stepping schemes.
For the spatial discretization using the Taylor-Hood finite element pair $\mathbb{P}_{k+1}/\mathbb{P}_k$, the expected convergence rates are $\mathcal{O}(h^{k+1})$ for the velocity field and $\mathcal{O}(h^k)$ for the pressure field, where h is the mesh size. Specifically, this results in:

$$\|\nabla u(t) - \nabla u_h(t)\|_{L^2(\Omega)} \leq C_1(t)h^{k+1}, \quad \|p(t) - p_h(t)\|_{L^2(\Omega)} \leq C_2(t)h^k, \quad \forall t \in (0,T].$$

The computational cost of the semi-implicit method offers significant advantages: the semi-implicit method transforms the nonlinear convective term $(\mathbf{u} \cdot \nabla)\mathbf{u}$ into a linear form, $(\mathbf{u}^n \cdot \nabla)\mathbf{u}^{n+1}$, where $\mathbf{u}^n$ is known from the previous time step.
This results in a linear system at each time step, which is computationally less intensive than solving the nonlinear system required in the fully implicit approach, although the matrix to be solved depends on the known solution $\mathbf{u}^n$ and must be recomputed at each time step.
The resulting matrices are non-symmetric due to the nature of the convective term, which can be more complex to solve but still more efficient compared to handling nonlinear systems, typically solved using iterative methods like Newton's method. However, since these methods do not explicitly treat the nonlinear term, they are less accurate than the fully implicit method.

Regarding stability, the semi-implicit method is conditionally stable, the stability restriction on the time step takes the following form

$$\Delta t \leq C\frac{h}{\max_{\mathbf{x}\in\Omega} |\mathbf{u}^n(\mathbf{x})|}. \tag{12}$$

where C is a constant, h is the spatial discretization size, $\mathbf{u}^n(\mathbf{x})$ represents the velocity at time step n and position $\mathbf{x}$, and Ω denotes the computational domain.

2. Implementation of the Navier-Stokes Solver using deal.II

The Navier-Stokes solver (both 2D and 3D) is implemented using C++ and deal.II Finite Element Library. The main steps of the implementation are encapsulated within the `NavierStokes<dim>` class and include:

2.1. Setup

The `setup()` function initializes all necessary data structures for the problem: first, the mesh is loaded from the file and partitioned for parallel processing so that the workload is distributed. Next, the finite element spaces for the velocity and pressure fields are initialized, the Degree of Freedom handler is set up to manage the distribution of degrees of freedom across the mesh. Finally, sparsity patterns for the system matrices are created.

2.2. Assembly of Constant Matrices

The `assemble_constant_matrices()` function assembles those parts of the system matrix that do not change over time, including:

- Viscosity Term
- Mass Matrix
- Pressure Term
- Preconditioner Matrices

2.3. Assembly Process

The `assemble()` function constructs the full system matrix and the corresponding right-hand side vector at each time step. The implementation was guided by the example provided in the deal.II library, particularly from step-35 [1]. the function iterates over all active cells, skipping those not owned by the current process. For each cell, the function re-initializes the `FEValues` object and computes the values and gradients of the velocity field at quadrature points, these values are used to form the nonlinear term, which is treated semi-implicitly, as discussed in Section 1.4, with options for handling its skew-symmetric form.

After assembling the system matrix and right-hand side vector, boundary conditions are incorporated. For Neumann boundary conditions, the function adds contributions from the boundary faces to the right-hand side vector. Dirichlet boundary conditions are applied to ensure the no-slip condition.

Finally, the assembled system matrix and right-hand side vector are compressed to finalize their setup, making them ready for the linear solver to compute the solution for the next time step.

A critical aspect of the Navier-Stokes equations is the treatment of the nonlinear term, which is a defining feature of these equations. Calculate that contribution involves iterating over quadrature points and degrees of freedom within each cell, then it computes contributions from $(\mathbf{u}\cdot\nabla)\mathbf{u}$ and $(\nabla\mathbf{u})^T\mathbf{u}$, then adds term to the local matrix (the skew-symmetric form of the non linear term):

$$\begin{aligned}\text{non\_linear\_contribution}(i,j) += 0.5\Big(&\\ &\text{scalar\_product}(\text{nonlinear\_term}, \text{fe\_values[velocity].value}(i,q)) +\\ &\text{scalar\_product}(\text{transpose\_term}, \text{fe\_values[velocity].value}(i,q))\Big)\\ &\times \text{fe\_values.JxW}(q)\end{aligned}$$

If `use_skew` is false, the the skew-symmetric form is not used and a standard semi-implicit treatment of the non-linear term is implemented:

$$\begin{aligned}\text{non\_linear\_contribution}(i,j) += \mathbf{u}_{\text{loc}}[q]\cdot&\\ &\text{fe\_values[velocity].gradient}(j,q)\cdot\\ &\text{fe\_values[velocity].value}(i,q)\cdot\\ &\text{fe\_values.JxW}(q)\end{aligned}$$

2.4. Linear Solver

The `solve_time_step()` function is responsible for solving the linear system at each time step. The iterative solver GMRES is used due to its effectiveness with non-symmetric matrices, and preconditioners are applied to improve the convergence rate of the GMRES solver. Different preconditioners can be used; see Section 4 for more details on their implementation.

At each time step, additional quantities of interest, such as drag and lift coefficients, are calculated in the `calculate_coefficients()` function.

2.5. Calculation of Drag and Lift Coefficients

The purpose of `calculate_coefficients()` is to evaluate the aerodynamic forces acting on the cylinder, specifically the drag force (F_D) and lift force (F_L), and then determine their corresponding dimensionless coefficients.

The main computational work involves iterating over all active cells in the domain and, for each boundary face with a specific boundary ID (the cylinder), the face values are reinitialized to compute the pressure and velocity gradient at quadrature points on the face. At each quadrature point, the function retrieves the normal vector (essential for projecting the stress tensor components in the direction normal to the surface), the pressure values, and velocity gradients from the solution vector. The pressure is used to construct a fluid pressure tensor, while the velocity gradient contributes to the viscous stress part of the stress tensor. The force vector at each quadrature point is calculated by combining the viscous stress and pressure contributions.

After that the drag and lift forces are accumulated by summing the respective components of the force vector over all quadrature points on all relevant boundary faces. The x-component of the force vector contributes to the drag force, while the y-component contributes to the lift force. The local contributions to the drag and lift forces from each process are summed across all processes using MPI reduction operations. These forces are then used to compute the dimensionless drag and lift coefficients by normalizing the forces with respect to the characteristic velocity and length of the cylinder.

3. Study case and results discussion

3.1. Geometry of the problem

The problem under consideration is the flow past a cylinder [3], focusing on both 2D and 3D cases, with Reynolds numbers (Re) varying up to 100. The primary objective is to compute and analyze the lift (C_L) and drag (C_D) coefficients over time for different Reynolds numbers.

In the 3D case, circular cross-sections of the cylinder are examined. The channel dimensions are $H = 0.41\,\text{m}$ and $W = 0.45\,\text{m}$, and the cylinder has a diameter and side length of $D = 0.1\,\text{m}$, with boundary conditions shown in Figure 11.

For the 2D case, the flow around a cylinder with a circular cross-section is considered. The domain has a height $H = 0.41\,\text{m}$ and a cylinder diameter $D = 0.1\,\text{m}$.
The problem geometry is represented in the following images:

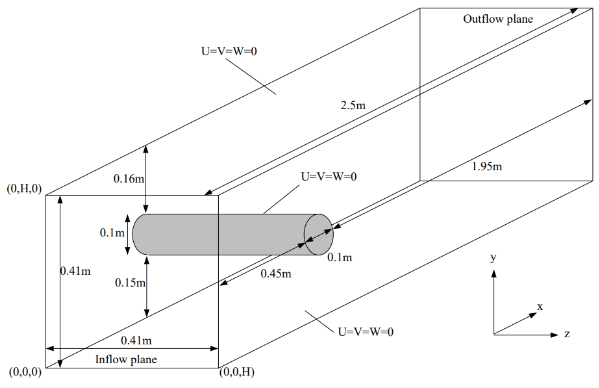


Figure 1: 3D case

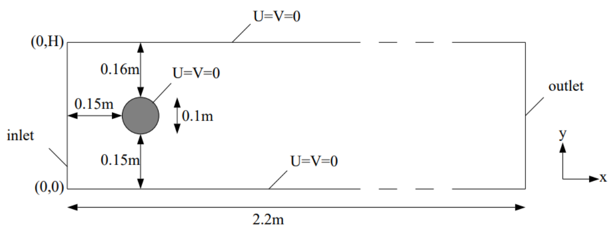


Figure 2: 2D case

The corresponding meshes are:

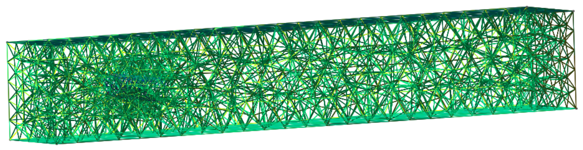


Figure 3: 3D coarse mesh

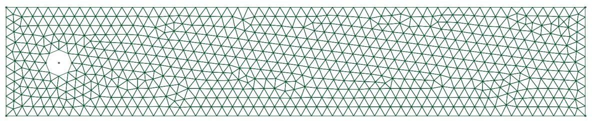


Figure 4: 2D coarse mesh

3.2. Reynolds number, drag and lift coefficients

The inflow velocity considered is the following:

$$U(0, y, t) = 4U_{\max}y(H - y)/H^2$$

where h is the domain height, which is 0.41 in this case, and $U_{\max}$ is the maximum inflow speed. No-slip boundary conditions are assumed on all other boundaries.

The Reynolds number Re can be calculated as:

$$Re = \frac{UL}{\nu}. \tag{5.8}$$

with U being the average inflow velocity, for the 2D case it is:

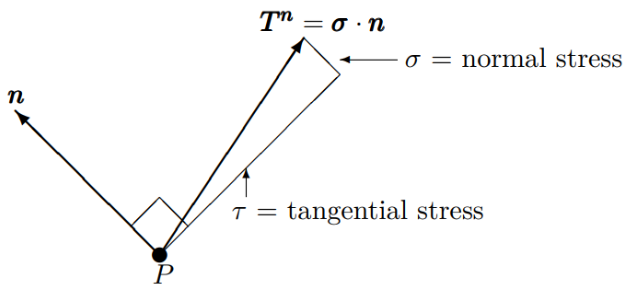
$$U = \frac{1}{H}\int_0^H \frac{4U_{\max}y(H-y)}{H^2}\,dy = \frac{4U_{\max}}{H^3}\left(\frac{Hy^2}{2} - \frac{y^3}{3}\right)\Bigg|_0^H = \frac{2}{3}U_{\max}.$$

for the 3D case it is:

$$U = \frac{4}{9} U_{\max}.$$

The drag and lift forces experienced by the cylinder are influenced by the normal and tangential stresses on its surface. As illustrated in Figure 5, the normal stress vector at a point on the circle is computed using the formula:

$$\mathbf{T}^n = \langle \tau, \sigma \rangle = \sigma \cdot \mathbf{n},$$



$\boldsymbol{T^n} = \boldsymbol{\sigma} \cdot \boldsymbol{n}$
σ = normal stress
$\boldsymbol{n}$
τ = tangential stress
P

Figure 5: Normal and tangential stresses at a point P on the circle.

In this expression, τ denotes the tangential component of the stress vector $\mathbf{T}^n$, while σ represents the normal component of the stress vector, acting perpendicular to the surface. The vector $\mathbf{n}$ is the unit normal vector at a point on the circle. The stress tensor σ is given by:

$$\sigma = \nu \nabla \mathbf{u} - p\mathbf{I},$$

where ν is the dynamic viscosity, $\nabla \mathbf{u}$ is the gradient of the velocity field, p is the pressure, and $\mathbf{I}$ is the identity matrix. The total drag force (F_D) and lift force (F_L) on the circle are calculated by integrating the normal stress vector around the entire circumference of the circle:

$$\langle F_D, F_L \rangle = \oint_S \mathbf{T}^n \, ds,$$

where S denotes the surface of the circle. Using these forces, the dimensionless lift and drag coefficients are determined as:

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

This project aims to analyze both steady and unsteady flows to explain the important quantities being considered.

3.3. Simulations

This section discusses the results obtained by varying the Reynolds number, considering values up to Re = 50.

The following figure shows the solution of the velocity field with Re=20, applying glyph filter at time 2.0s.

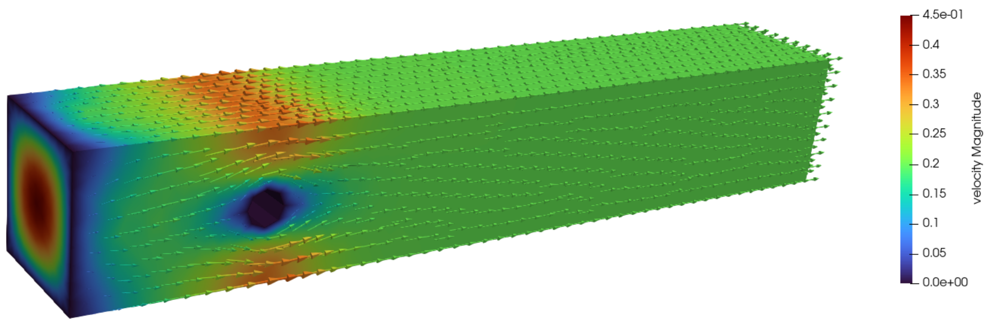


Figure 6: Glyph filter

In this image the Slice filter has been applied:

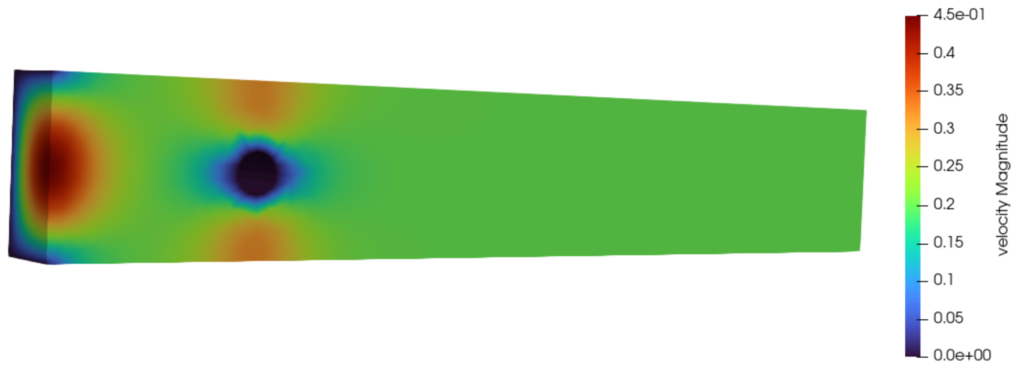
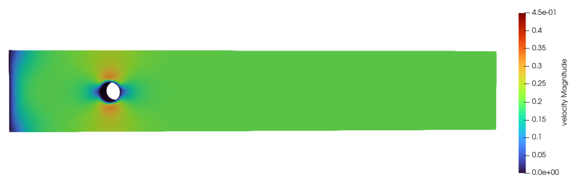


Figure 7: Slice at t=2.0

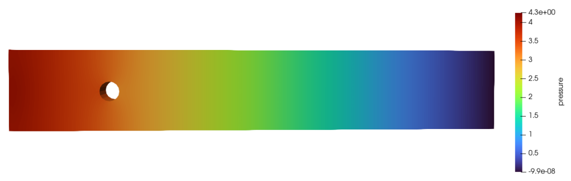
This provides an overview of the simulations. Detailed analyses and discussions of these simulations will be presented below.

Re = 20

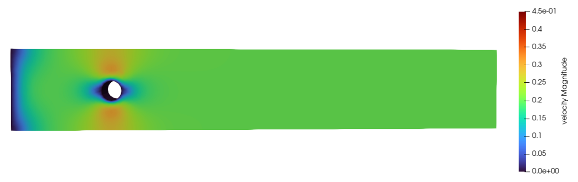
These are the results with Reynold Number 20 at time 0.125s and 2.0s.



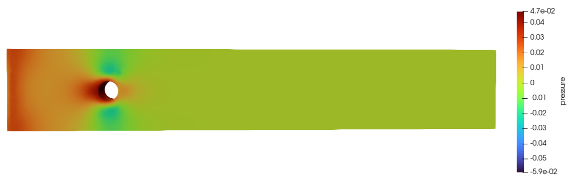
Velocity Magnitude at time t = 0.0 s



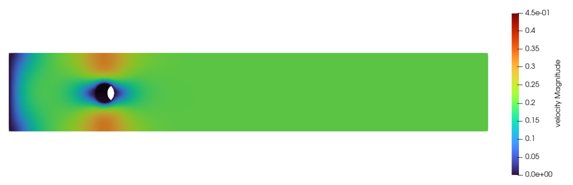
Pressure at time t = 0.0 s



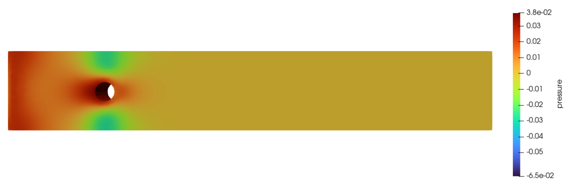
Velocity Magnitude at time t = 0.5 s



Pressure at time t = 0.5 s



Velocity Magnitude at time t = 2.0 s



Pressure at time t = 2.0 s

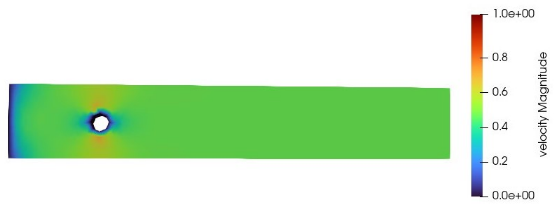
Figure 8: Time evolution with $Re = 20$, 3D

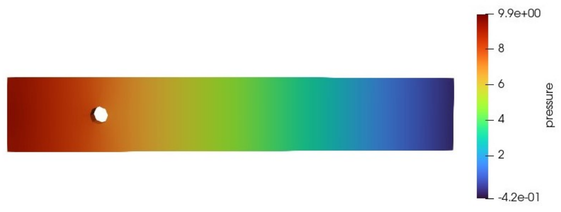
As shown in the results, the pressure and velocity distribution remain stable over time, which is a characteristic

of laminar flow. In laminar flow, the fluid moves in parallel layers with no disruption between them, resulting in smooth and predictable behavior. This stability in the pressure and velocity fields suggests that the flow is not transitioning to turbulence, which would typically be indicated by fluctuations and irregularities in these distributions.
At 0.125 seconds, the initial conditions set the stage for the flow to develop. By 2.0 seconds, the flow has reached a steady state, maintaining its laminar characteristics. The uniformity in the velocity distribution indicates that the flow velocity is consistent across different points in the flow field, and the pressure distribution shows a gradual and smooth change, reinforcing the laminar nature of the flow.

Re = 50

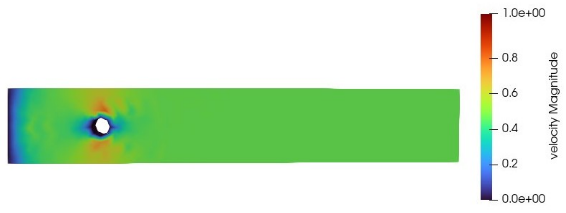
These are the results with Reynold Number 50 from time 0.0s to 2.0s

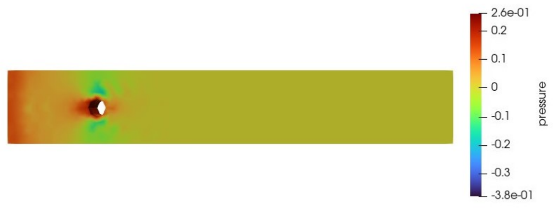




Velocity Magnitude at time t = 0.0 s

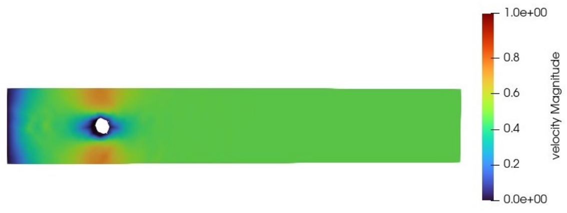
Pressure at time t = 0.0 s

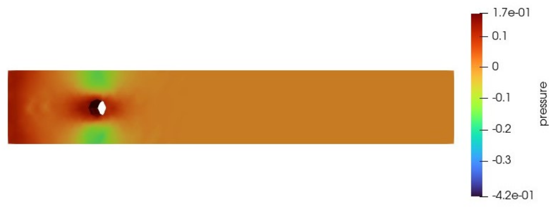




Velocity Magnitude at time t = 0.375 s

Pressure at time t = 0.375 s





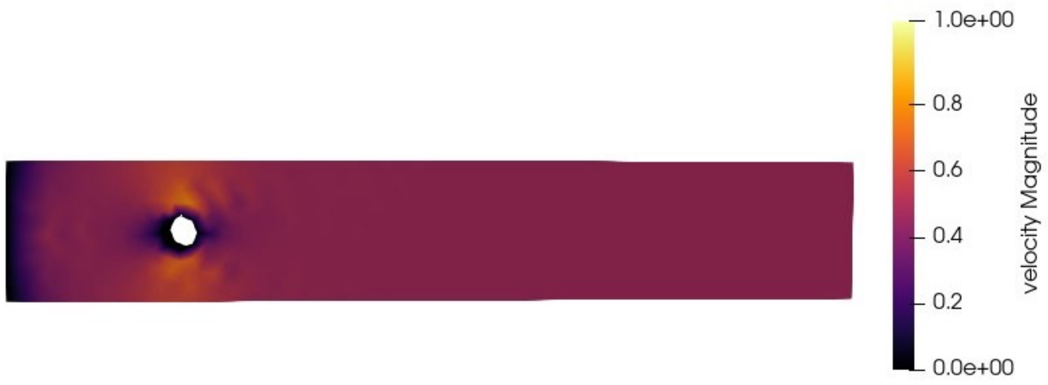
Velocity Magnitude at time t = 2.0 s

Pressure at time t = 2.0 s

Figure 9: Time evolution with $Re = 50$, 3D

The Reynolds number Re = 50 suggests that the flow is laminar but approaching transitional. At this value, vortices may start to form and become more noticeable: the image shows vortex structures forming due to the geometry and flow conditions.

Given this observation, it is reasonable to conclude that at higher Reynolds numbers, these phenomena may become more frequent and pronounced. As the Reynolds number increases, the flow tends to become more unstable and transitional, eventually leading to turbulence. In such regimes, vortex shedding can occur more regularly and with greater intensity, leading to a complex and chaotic flow pattern.



Detail of vortex structures forming at time t = 0.375 s

The image above is identical to the one used for analyzing velocity magnitude at time t=0.375, but with an adjusted color palette to more clearly illustrate the forming vortex structure.

3.4. Instability of the solution for high Reynolds number

For high Reynolds numbers and, consequently, velocities higher than v = 2.0, the solution becomes unsteady and the solvers blows up even with smaller time steps and finer meshes. This is due to the fact that the flow becomes turbulent and the semi-implicit method not being unconditionally stable. Alternative approaches, such as fully implicit methods or advanced stabilization techniques, may be necessary to achieve stable and accurate results. These methods, while more computationally intensive, offer the stability needed to handle the increased nonlinearity and turbulence associated with high Reynolds number flows.

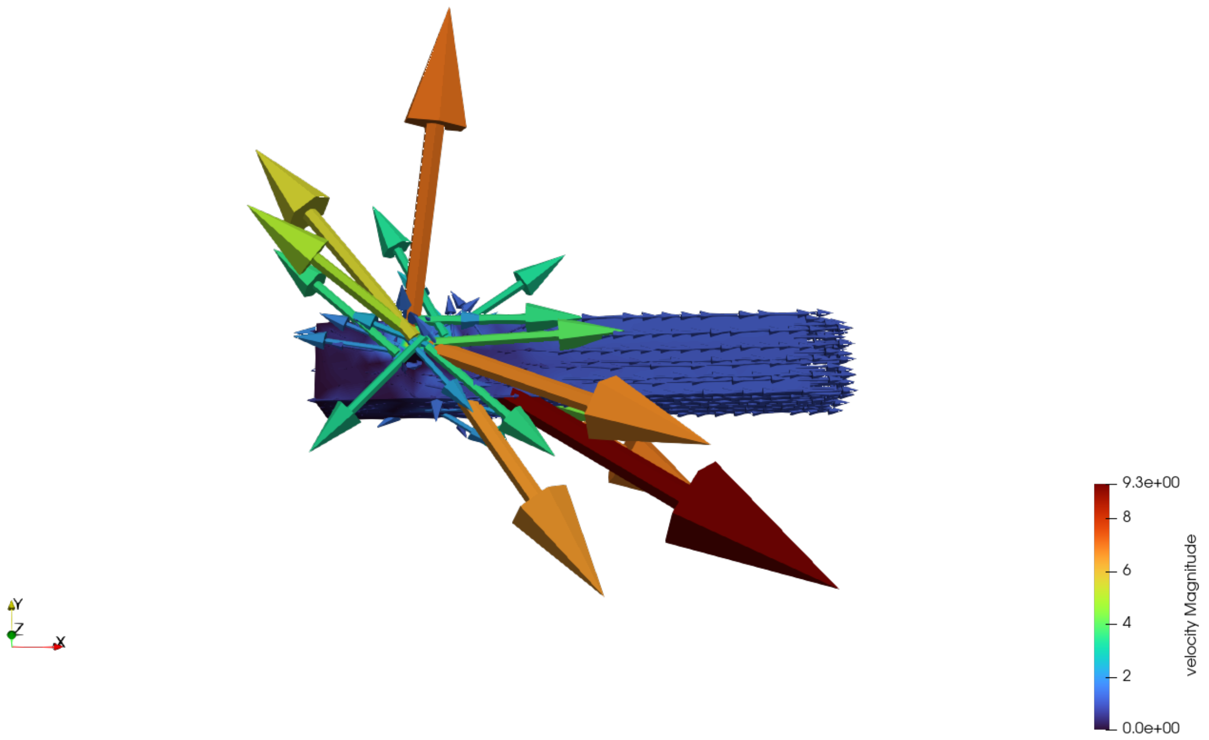


Figure 10: View of blown up solver for Re = 100

3.5. Use of skew-symmetric form

Several mathematical representations of the convective term exist, each providing specific advantages depending on the application context. To enhance the understanding of nonlinear behavior in numerical simulations, the skew-symmetric form of the convective term is also analyzed. This form is given by:

$$\frac{1}{2}((\mathbf{u} \cdot \nabla)\mathbf{u} + (\nabla \mathbf{u})^T \mathbf{u})$$

the advantages of employing this formulation are that it aids in conserving energy within the numerical system and it mitigates the risk of numerical oscillations.

the left image represents the results using the standard form of the convective term, while the right image shows the results using the skew-symmetric form. indeed, these two images are quite different. The justification lies in the fact that using a different representation of the non linear term can affect the velocity distribution and other parameters of the flow.

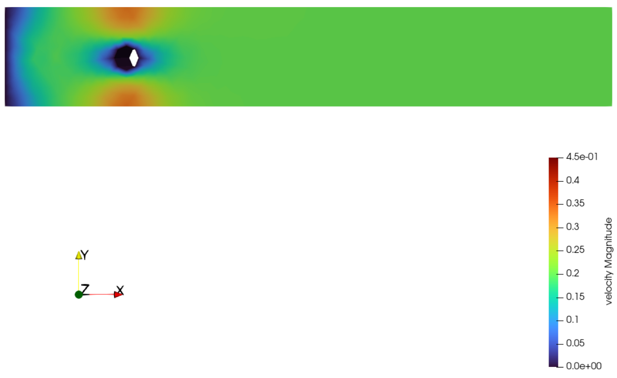


Figure 11: Without skew symmetric form (3D)

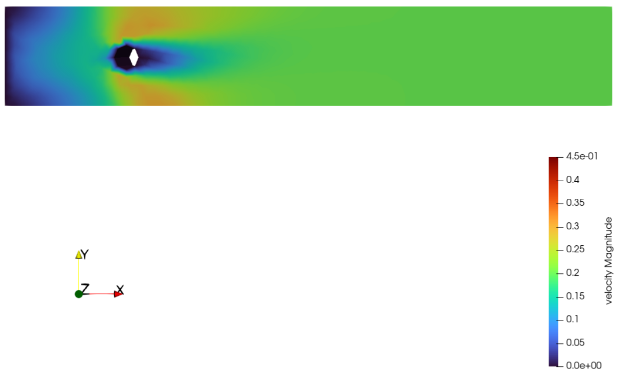


Figure 12: With skew symmetric form (3D)

3.6. Drag and lift coefficients results

The figures below show the time evolution of drag and lift coefficients for three different Reynolds numbers (Re = 5, Re = 20, and Re = 50. With velocities v = 0.1, v = 0.45 and v = 1.05, respectively) in a flow past a cylinder scenario, using the 0.1 mesh, viscosity $\nu = 0.001$ and $\Delta t = 0.125$.

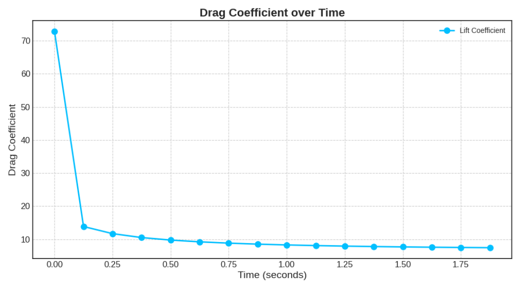


Figure 13: Drag coefficient for Re = 5

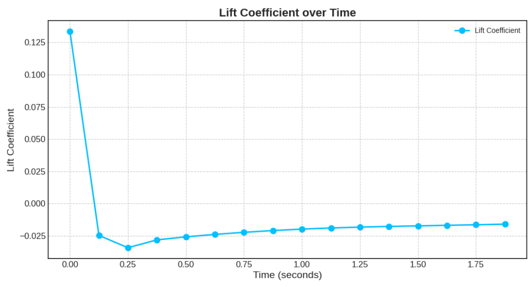


Figure 14: Lift coefficient for Re = 5

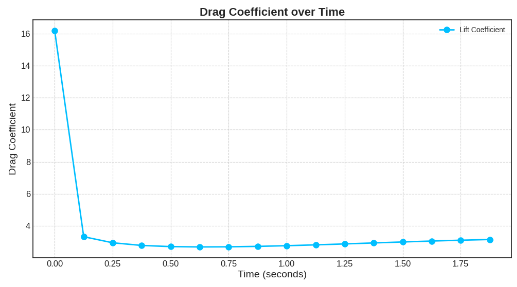


Figure 15: Drag coefficient for Re = 20

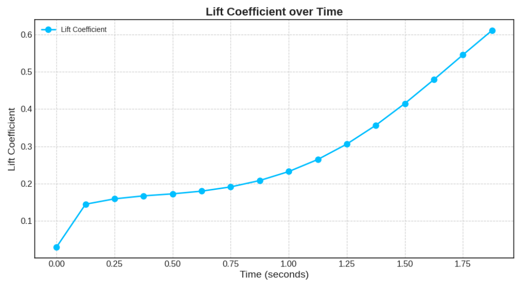


Figure 16: Lift coefficient for Re = 20

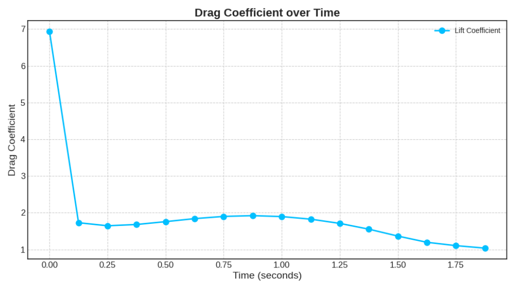


Figure 17: Drag coefficient for Re = 50

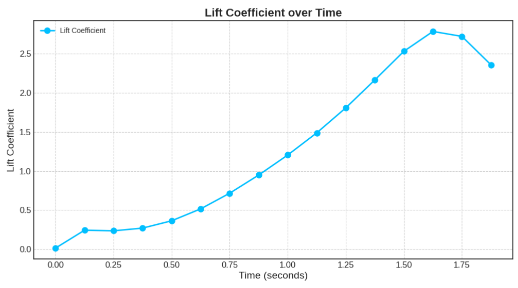


Figure 18: Lift coefficient for Re = 50

Reynolds Number 5 (Re = 5):

- *Drag Coefficient (Figure 13)*: The drag coefficient starts very high at the initial moment and then sharply decreases within a short period. It stabilizes at a much lower value after about 0.25 seconds.
- *Lift Coefficient (Figure 14)*: The lift coefficient initially shows a high positive value, quickly drops, and then fluctuates around zero

Reynolds Number 20 (Re = 20):

- *Drag Coefficient (Figure 15)*: Similarly, the drag coefficient starts high and decreases quickly within a short time, stabilizing at a lower value around 0.25 seconds.
- *Lift Coefficient (Figure 16)*: The lift coefficient starts at a low value and increases steadily over time, indicating a growing lift force as time progresses.

Reynolds Number 50 (Re = 50):

- *Drag Coefficient (Figure 17)*: The drag coefficient starts high and decreases quickly but does not stabilize as completely as in lower Reynolds numbers. There is a gradual decline over the observed period.
- *Lift Coefficient (Figure 18)*: The lift coefficient starts at zero and increases steadily, peaking around 1.5 seconds before showing a slight decline, indicating the presence of more complex flow dynamics such as vortex shedding and transient lift forces.

These observations are consistent with theoretical expectations for flow past a cylinder governed by the Navier-Stokes equations, demonstrating how the Reynolds number influences the stability and symmetry of the flow [3].

4. Preconditioners

In computational fluid dynamics, solving the the uncompressible Navier-Stokes equations poses significant challenges for numerical solution techniques due to their nonlinear nature, particularly when employing iterative solvers. In this context, preconditioning techniques are fundamental to enhance convergence rates and computational efficiency. By modifying the system matrices to mitigate stiffness and ill-conditioning, preconditioners aim to facilitate the convergence of iterative solvers, thereby accelerating the solution process, enabling more effective and robust simulations of fluid flow phenomena. For this reason, it is important to choose efficient, optimal and scalable preconditioners.

4.1. SIMPLE preconditioner

The Semi-Implicit Method for Pressure Linked Equations (SIMPLE) is an iterative method which first solves the momentum equation and then updates the pressure field and the velocity field to conserve the mass by using the continuity equation. The method can be reinterpreted as if it were associated to a preconditioner :

$$P_{SIMPLE} = \begin{bmatrix} F & 0 \\ B & -\tilde{S} \end{bmatrix} \begin{bmatrix} I & D^{-1}B^T \\ 0 & \alpha I \end{bmatrix} \tag{13}$$

where D is the diagonal of F, α is a parameter that damps the pressure update, and $\tilde{S} = BD^{-1}B^T$.
The implementation of the SIMPLE preconditioner involves creating the inverse of the diagonal of F and constructing the Schur complement approximation matrix $\tilde{S}$.
In the vmult method, the preconditioner solves $F \cdot \text{sol}_u = \text{src}_u$, computes the residual $\text{inter\_sol} = B \cdot \text{sol}_u - \text{src}_p$, and then solves $\tilde{S} \cdot \text{sol}_p = \text{inter\_sol}$ using GMRES. Finally, the residual dst_u is computed as $\text{sol}_u - D^{-1} \cdot B^T \cdot \text{sol}_p$ and adjusted accordingly. This detailed implementation ensures that the SIMPLE preconditioner is effectively applied, enhancing the solution process.
SIMPLE is efficient especially when the problem matrix is diagonally dominant. In particular, this is true in our case if Δt is small enough. However the efficiency deteriorates as Δt increases; in this case $\tilde{S}$ is a poor approximation of the exact Schur complement S of the approximation of the Schur complement.

4.2. Yosida preconditioner

The Yosida preconditioner is inspired by the Yosida method, which approximates nonsmooth functions through a sequence of smooth (continuously differentiable) functions. This method uses a parameter (often denoted as λ) that controls the smoothness of these approximations, enabling effective handling of nonsmooth problems.
The Schur complement is approximated using $\tilde{S} = \Delta t B M_u^{-1} B^T$, where M_u denotes the mass matrix associated to the velocity shape functions. The smaller Δt the better the approximation since, as Δt tends to zero, the mass matrix M_u becomes the dominating term in F.
The definition of the Yosida preconditioner can be written as follows,

$$P_Y = \begin{bmatrix} F & 0 \\ B & -\tilde{S} \end{bmatrix} \begin{bmatrix} I & F^{-1}B^T \\ 0 & I \end{bmatrix} \tag{14}$$

where M_u^{-1} is replaced by the inverse of the diagonal D of M_u such that we can form explicitly $\tilde{S}$ as in SIMPLE.
The initialize method sets up the preconditioner by saving references to the input matrices and computing the inverse diagonal of the mass matrix associated to the velocity shape functions, creating the matrix $\tilde{S}$ and initializing the ILU preconditioners.
The vmult method applies the preconditioners to solve a linear system: it solves first $F \cdot \text{sol1}_u = \text{src}_u$ and then $\tilde{S} \cdot \text{sol1}_p = B \cdot \text{sol1}_u - \text{src}_p$, both using GMRES with ILU preconditioners. Lastly we combine the results $\text{dst}_u = \text{sol1}_u - F^{-1}B^T \cdot \text{sol1}_p$.
The efficiency of the Yosida preconditioner can deteriorate as Δt increases. When Δt is large, the approximation of $\tilde{S}$ becomes less accurate, and the dominance of the mass matrix and the dominance of mass matrix M_u in F diminishes, leading to decreased performance.

4.3. Preconditioners for HPC

Preconditioners are crucial in optimizing high-performance computing by enhancing computational code execution. They must be meticulously designed with specific properties for optimal performance, as highlighted by [2].
First, preconditioners need **strong scalability** to maintain performance as problem sizes grow, especially in finite element codes with large meshes. Second, they should be **independent of mesh size**, crucial for handling finer meshes without increasing iterations or computational costs. Another key property is **optimality**,

ensuring that preconditioned iterations remain bounded and that the computational cost scales linearly with the matrix dimension N. Finally, preconditioners must be **robust**, performing well regardless of the physical parameters in partial differential equations, such as varying Reynolds numbers in the Navier-Stokes equations. Adhering to these principles enhances the efficiency and reliability of high-performance computing, enabling more complex computational challenges to be tackled effectively.

Approximate SIMPLE Preconditioner

The aSIMPLE preconditioner extends the Semi-Implicit Method for Pressure Linked Equations (SIMPLE) to improve the efficiency of solving linear systems arising from fluid dynamics simulations.

$$P^{-1}_{\text{aSIMPLE}} = \begin{bmatrix} D^{-1} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & -B^T \\ 0 & I \end{bmatrix} \begin{bmatrix} D & 0 \\ 0 & \frac{1}{\alpha} I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \hat{S}^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -B & I \end{bmatrix} \begin{bmatrix} \hat{F}^{-1} & 0 \\ 0 & I \end{bmatrix}$$

To address the fact that as Δt increases, the efficiency of the SIMPLE preconditioner deteriorates, the aSIMPLE preconditioner uses an Incomplete LU factorization to better approximate the inverse of F and S, thus enhancing the preconditioner's effectiveness even for larger Δt.
In the initialize function, the class sets up the required matrices and vectors, computes the diagonal inverses, and forms the Schur complement matrix S. It then initializes ILU preconditioners for both F and S.
The vmult function applies the preconditioner by first solving for the momentum component and then updating the modified pressure component, both using GMRES.
This systematic approach helps in accelerating the convergence of the iterative solvers, especially when the problem is not strongly diagonally dominant.

4.4. Discussion results

The following table shows the performance of the solver using the preconditioners implemented, showing the total amount of time (in seconds) required to execute the solver as well as the the number of iterations required using only 1 process.

Method	Time (seconds)	Iterations
SIMPLE	4.790718	170
aSIMPLE	4.040719	139
YOSIDA	10.790526	201

Table 1: Performance of different methods on mesh 0.1

As we can see, aSIMPLE and SIMPLE perform pratically in the same way, with aSIMPLE showing better performances in terms of iterations and time with respect to SIMPLE, while both outperforming the Yosida preconditioner in the number of iterations as well as the amount of time required to execute the solver. This results are in agreement with the study proposed in [2].
These tests are performed on the coarser mesh using only one process. These results are confirmed by performing the same tests in parallel (four processes) on finer meshes, where the Yosida preconditioner takes twice the amount of time with respect to the SIMPLE and aSIMPLE methods.

Method	Time (seconds)	Iterations
SIMPLE	10.00447	139
aSIMPLE	11.39612	147
YOSIDA	22.58623	135

Table 2: Performance of different methods on mesh 0.05

Method	Time (seconds)	Iterations
SIMPLE	9.97	139
aSIMPLE	11.48633	147
YOSIDA	21.41855	135

Table 3: Performance of different methods on mesh 0.0125

5. Computation Time Analysis

This section evaluates the impact of utilizing multiple MPI processes for computing the solution. The following tables shows the time taken using different numbers of MPI processes and different meshes:

# MPI processes	Time (seconds)
1	22.0199
4	11.1516
8	10.0424

Table 4: time taken using 0.1 mesh, total time 2s and deltat 0.125s

When all computations are handled by one process, there is no overhead from inter-process communication but the process must handle the entire computational load. Using 4 MPI processes reduces the computation time to 11.1516 seconds, nearly halving the time required compared to a single process. Notice that increasing the number of MPI processes to eight further reduces the computation time to 10.0424 seconds. While this is an improvement over four processes, the reduction in time is less pronounced compared to the jump from one to four processes. This could suggests increased communication overhead. Using this setup the speed up using more than 4 processors is nearly 2.
While using 0.025 mesh with deltat 0.125 and aSIMPLE preconditioner:

# MPI processes	Time (seconds)
1	2344.77
2	1613.84
4	1155.45
8	1054.91

Table 5: time taken using 0.025 mesh, total time 2s and deltat 0.125s

Also in this case, the difference using more than 4 processes is negligible, using 8 processes the computation time only decreases from 1155.4s to 1054.91 seconds. This suggests, like in the previous case, that communication overhead and synchronization between the larger number of processes are beginning to offset the gains from parallelization.

References

[1] dealii.org. Code gallery: Time dependent navier stokes. `https://www.dealii.org/current/doxygen/deal.II/code_gallery_time_dependent_navier_stokes.html`, 2024.

[2] Quarteroni A. Deparis S., Grandperrin G. Parallel preconditioners for the unsteady navier-stokes equations and applications to hemodynamics simulations. *Elsevisier*, pages 253–273, 2013.

[3] Durst F. Krause E. Rannache R. Schäfer M., Turek S. Benchmark computations of laminar flow around a cylinder. *Vieweg+Teubner*, pages 547–566, 1996.