POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Navier-Stokes equations

Author: Giulio Enzo Donninelli, Alessia Rigoni, Vittorio Sironi and Alessandro Verrengia

Academic year: 2025-2026

---

1. Introduction

This project focuses on the numerical solution of the unsteady, incompressible Navier-Stokes equations to simulate laminar flow past a cylinder in 2D or 3D channels, adhering to the benchmarks defined by Schäfer et al. (1996).

The physical problem involves a fluid with kinematic viscosity $\nu = 10^{-3}\,\mathrm{m}^2/\mathrm{s}$ and density $\rho = 1.0\,\mathrm{kg}/\mathrm{m}^3$. The flow is characterized by the Reynolds number $\mathbb{R}e = \bar{U}D/\nu$, where D is the cylinder diameter and $\bar{U}$ is the mean velocity.

The primary objective is to accurately compute lift and drag coefficients for Reynolds numbers $\mathbb{R}e \leq 100$. These dimensionless quantities relate the hydrodynamic forces acting on the cylinder to the fluid inertia and are defined as:

$$C_D = \frac{2F_D}{\rho \bar{U}^2 A} \quad \text{and} \quad C_L = \frac{2F_L}{\rho \bar{U}^2 A}$$

where F_D and F_L denote the forces parallel and perpendicular to the flow direction, respectively, and A represents the cross-sectional length of the obstacle (corresponding to the diameter D in the 2D case and to $A = D \cdot H$ in the 3D case).

The solver is implemented in C++ using the `deal.II` library, utilizing distributed memory parallelism (MPI) and Trilinos linear algebra backends.

2. Mathematical model

2.1. Strong formulation

The original form of the Navier-Stokes equations for an incompressible viscous fluid in a domain $\Omega \subset \mathbb{R}^d$ $(d = 2, 3)$ is defined as follows:

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} - \nu\Delta\mathbf{u} + \nabla p = \mathbf{f} & \Omega, t > 0, \quad (1a) \\ \nabla \cdot \mathbf{u} = 0 & \Omega, t > 0, \quad (1b) \\ \mathbf{u} = \mathbf{g} & \Gamma_D, t > 0, \\ \nu\nabla\mathbf{u}\mathbf{n} - p\mathbf{n} = \mathbf{h} & \Gamma_N, t > 0, \\ \mathbf{u}(t=0) = \mathbf{u}_0 & \Omega \end{cases}$$

where $\mathbf{u}$ is the fluid velocity, p is the pressure, ν is the kinematic viscosity of the fluid, $\mathbf{f}$ are the external forces and $\mathbf{g}$ and $\mathbf{h}$ are the assigned functions.
In details, the Dirichlet and Neumann boundaries are respectively $\Gamma_D \subset \partial\Omega$ and $\Gamma_N = \partial\Omega \backslash \Gamma_D$.

2.2. Weak formulation

Let $L^2(\Omega)$ be the space of square-integrable functions and $H^1(\Omega)$ the Sobolev space. The function spaces for velocity and pressure are respectively:

$$V = \{\mathbf{v} \in [H^1(\Omega)]^d : \mathbf{v}|_{\Gamma_D} = \mathbf{0}\},$$
$$Q = L^2(\Omega)$$

where $H^1(\Omega)$ denotes the Sobolev space of functions that, along their first derivatives, are square-integrable (in the Lebesgue sense):

$$H^1(\Omega) = \{v \in L^2(\Omega), \nabla v \in L^2(\Omega)\}$$

The weak formulation is obtained multiplying the momentum equation (1a) by a test function $\mathbf{v} \in V$ and integrating over the domain Ω:

$$\int_\Omega \frac{\partial \mathbf{u}}{\partial t}\mathbf{v}\, d\Omega + \int_\Omega ((\mathbf{u}\cdot\nabla)\mathbf{u})\cdot\mathbf{v}\, d\Omega - \int_\Omega \nu\Delta\mathbf{u}\cdot\mathbf{v}\, d\Omega + \int_\Omega \nabla p\cdot\mathbf{v}\, d\Omega = \int_\Omega \mathbf{f}\cdot\mathbf{v}\, d\Omega$$

We can then apply the *First Green identity* to the diffusive term and integrate the pressure gradient by parts, obtaining the following results:

- for the diffusive term:

$$-\int_\Omega \nu\Delta\mathbf{u}\cdot\mathbf{v}\, d\Omega = \int_\Omega \nu\nabla\mathbf{u} : \nabla\mathbf{v}\, d\Omega - \int_{\partial\Omega} \nu(\nabla\mathbf{u}\cdot\mathbf{n})\mathbf{v}\, d\Gamma$$

- for the pressure gradient:

$$\int_\Omega \nabla p\cdot\mathbf{v}\, d\Omega = -\int_\Omega p(\nabla\cdot\mathbf{v})\, d\Omega + \int_{\partial\Omega} (p\mathbf{n})\cdot\mathbf{v}\, d\Gamma$$

Exploiting the boundary conditions on these two new formulations, the boundary integral $(\nu(\nabla\mathbf{u}\cdot\mathbf{n}) - p\mathbf{n})\cdot\mathbf{v}$ vanishes on Γ_D (since $\mathbf{v} = \mathbf{0}$) and equals $\mathbf{h}\cdot\mathbf{v}$ on Γ_N.

Thus, the momentum equation becomes:

$$\int_\Omega \frac{\partial \mathbf{u}}{\partial t}\mathbf{v}\, d\Omega + \int_\Omega (\mathbf{u}\cdot\nabla)\mathbf{u}\mathbf{v}\, d\Omega + \int_\Omega \nu\nabla\mathbf{u} : \nabla\mathbf{v}\, d\Omega - \int_\Omega p\nabla\cdot\mathbf{v}\, d\Omega = \int_\Omega \mathbf{f}\cdot\mathbf{v}\, d\Omega + \int_{\Gamma_N} \mathbf{h}\cdot\mathbf{v}\, d\Gamma$$

Similarly, the weak formulation for the continuity equation is obtained multiplying Equation (1b) by a test function $q \in Q$ and integrating over the domain Ω:

$$\int_\Omega (\nabla\cdot\mathbf{u})q\, d\Omega = 0$$

To write the problem in a more compact form, we can define the following bilinear and trilinear forms as:

$$a(\mathbf{u},\mathbf{v}) = \int_\Omega \nu\nabla\mathbf{u} : \nabla\mathbf{v}\, d\Omega,$$
$$b(\mathbf{v},q) = -\int_\Omega q(\nabla\cdot\mathbf{v})\, d\Omega,$$
$$c(\mathbf{w};\mathbf{u},\mathbf{v}) = \int_\Omega ((\mathbf{w}\cdot\nabla)\mathbf{u})\cdot\mathbf{v}\, d\Omega$$

while the linear functional can be defined as:

$$F(\mathbf{v}) = \int_\Omega \mathbf{f}\cdot\mathbf{v}\, d\Omega + \int_{\Gamma_N} \mathbf{h}\cdot\mathbf{v}\, d\Gamma.$$

The weak problem is then states as follows:

$$\begin{cases} \left(\frac{\partial \mathbf{u}}{\partial t}, \mathbf{v}\right) + a(\mathbf{u},\mathbf{v}) + c(\mathbf{u};\mathbf{u},\mathbf{v}) + b(\mathbf{v},p) = F(\mathbf{v}) & \text{(2a)} \\ b(\mathbf{u},q) = 0 & \text{(2b)} \end{cases}$$

where $(\cdot,\cdot)$ denotes the standard $L^2(\Omega)$ inner product.

2.3. Finite Element formulation

In order to discretize the problem in space, let introduce the space of finite elements:

$$X_h^r = \{\mathbf{v}_h \in C^0(\bar{\Omega}) | \mathbf{v}_{h|K} \in \mathbb{P}^r, \forall K \in \mathcal{T}_h\}$$

where $\mathbb{P}_r$ is the space of polynomials of degree less than or equal to r while $\mathcal{T}_h$ is a regular triangulation that forms an approximation Ω_h of Ω. The discrete spaces for velocity and pressure are then, respectively, denoted by:

$$V_h = \left(X_h^2\right)^d \cap V \quad (d = 2,3)$$
$$Q_h = X_h^1 \cap Q$$

To ensure numerical stability and satisfy the Inf-Sup condition, we employ Taylor-Hood finite element pairs. Specifically, we use for the velocity piecewise quadratic polynomials ($\mathbb{P}^2$) while for the pressure piecewise linear polynomials ($\mathbb{P}^1$).

The semi-discrete problem is obtained by restricting the variational formulation to the finite-dimensional subspaces $V_h \subset V$ and $Q_h \subset Q$. It reads as follows:
For each $t \in (0,T)$ find $(\mathbf{u}_h(t), p_h(t) \in V_h \times Q_h$ such that for all $(\mathbf{v}_h, q_h) \in V_h \times Q_h$, with $\mathbf{u}_h(0) = \mathbf{u}_{0,h}$:

$$\begin{cases} \left(\frac{\partial \mathbf{u}_h}{\partial t}, \mathbf{v}_h\right) + a(\mathbf{u}_h,\mathbf{v}_h) + c(\mathbf{u}_h;\mathbf{u}_h,\mathbf{v}_h) + b(\mathbf{v}_h,p_h) = F(\mathbf{v}_h) \\ b(\mathbf{u}_h,q_h) = 0 \end{cases}$$

By expressing the discrete velocity and pressure fields in terms of their respective bases, $\mathbf{u}_h(t) = \sum_{j=1}^{N_u} u_j(t)\varphi_j$ and $p_h(t) = \sum_{k=1}^{N_p} p_k(r)\psi_k$, the problem can be rewritten as a system of *Ordinary Differential Equations* (ODEs) as:

$$\begin{cases} \mathbf{M}\dot{\mathbf{U}} + \mathbf{A}\mathbf{U} + \mathbf{C}(\mathbf{U})\mathbf{U} + \mathbf{B}^T\mathbf{P} = \mathbf{F} \\ \mathbf{B}\mathbf{U} = \mathbf{0} \end{cases} \quad (3)$$

where U and P are the vectors of the degrees of freedom for velocity and pressure, respectively. In this notation:
- $\mathbf{M}$ is the mass matrix, $M_{ij} = (\varphi_j, \varphi_i)$;
- $\mathbf{A}$ is the stiffness matrix representing the viscous term, $A_{ij} = a(\phi_j, \phi_i)$;
- $\mathbf{C}(\mathbf{U})$ is the non-linear convection matrix, $C(U)_{ij} = c(\mathbf{u}_h; \phi_j, \phi_i)$;
- $\mathbf{B}$ is the divergence matrix, $B_{ki} = b(\phi_i, \psi_k)$.

2.4. Fully discrete formulation

To obtain the fully discrete formulation, we partition the time interval $[0,T]$ into N_T sub-intervals $[t^n, t^{n+1}]$ of constant width $\Delta t = T/N_T$ where $t^n = n\Delta t$ for $n = 0,\dots,N_T$.
Let $\mathbf{u}_h^n \in V_h$ and $p_h^n \in Q_h$ be the approximations of the solutions $\mathbf{u}$ and p at time t_n.

We employ the θ-method for time discretization, which approximates the time derivative using a finite difference $\left(\frac{\partial \mathbf{u}(t)}{\partial t} \approx \frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t}\right)$ and evaluates the terms of the semi-discrete system (3) as a weighted average between the current time t^n and the next time t^{n+1}.

The fully discrete problem reads as follows:
Find $(\mathbf{u}_h^{n+1}, p_h^{n+1}) \in V_h \times Q_h$ such that:

$$\left(\frac{\mathbf{u}_h^{n+1}-\mathbf{u}_h^n}{\Delta t}, \mathbf{v}_h\right) + \theta\left[a(\mathbf{u}_h^{n+1},\mathbf{v}_h) + c(\mathbf{u}_h^{n+1};\mathbf{u}_h^{n+1},\mathbf{v}_h)\right] + (1-\theta)\left[a(\mathbf{u}_h^n,\mathbf{v}_h) + c(\mathbf{u}_h^n;\mathbf{u}_h^n,\mathbf{v}_h)\right] + b(\mathbf{v}_h, p_h^{n+1}) = \theta F^{n+1}(\mathbf{v}_h) + (1-\theta)F^n(\mathbf{v}_h)$$

and

$$b(\mathbf{u}_h^{n+1}, q_h) = 0$$

where $\theta = [0,1]$ is a parameter that defines the method used.

The three classical temporal schemes are:
- $\theta = 0$ (*forward Euler*), a first-order explicit scheme, that requires a restrictive constraint on Δt;
- $\theta = 1$ (*backward Euler*), a first-order implicit scheme, that is unconditionally stable;
- $\theta = 0.5$ (*Crank-Nicolson*), a second-order implicit scheme.

In this project, we have opted for the latter two methods due to their unconditional stability.

2.5. Linearization of the convective term

The fully discrete system (3) remains non-linear due to the convective term $c(\mathbf{u}_h^{n+1};\mathbf{u}_h^{n+1},\mathbf{v}_h)$. In order to solve this system at each time step t^{n+1}, we considered two linearization procedure: the iterative Newton's method (when the $\mathbb{R}e < 50$) and a semi-implicit linearized method (when $\mathbb{R}e \geq 50$).

2.5.1. Newton's Method

We define a nonlinear function whose root is a solution to the Navier-Stokes Equations by:

$$F(\mathbf{u}, p) = \begin{pmatrix} \frac{\mathbf{u}-\mathbf{u}^n}{\Delta t} - \nu\Delta\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \mathbf{f} \\ -\nabla\cdot\mathbf{u} \end{pmatrix} \quad (4)$$

This method treats the non-linearity by solving the root-finding problem $R(\mathbf{u}_h^{n+1}, p_h^{n+1}) = 0$, where R is the global non-linear residual of the discrete formulation. At each time step t^{n+1}, we introduce an iterative procedure (indexed by k) to find corrections $(\delta\mathbf{u}_h^k, \delta p_h^k)$ to the current guess $(\mathbf{u}_h^{n+1,k}, p_h^{n+1,k})$.

Linearizing the convective term around the current iterate $\mathbf{u}_h^k$, we obtain the Fréchet derivative which contributes to the Jacobian matrix. Denoting $\mathbf{x} = (\mathbf{u}, p)$, Newton's iteration on a vector function can be defined as:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - (\nabla F(\mathbf{x}^k))^{-1} F(\mathbf{x}^k)$$

Instead of evaluating the Jacobian matrix and taking its inverse, we consider the update term as a whole, $\delta\mathbf{x}^k = -(\nabla F(\mathbf{x}^k))^{-1} F(\mathbf{x}^k)$, where $\mathbf{x}^{k+1} = \mathbf{x}^k + \delta\mathbf{x}^k$. We find the update term by solving the system:

$$\nabla F(\mathbf{x}^k)\delta\mathbf{x}^k = -F(\mathbf{x}^k) \quad (5)$$

The left side of (5) represents the directional gradient of $F(\mathbf{x})$ along $\delta\mathbf{x}^k$ at $\mathbf{x}^k$. By definition, the directional gradient is given by:

$$\begin{aligned} &\nabla F(\mathbf{u}^k, p^k)(\delta\mathbf{u}^k, \delta p^k) \\ &\quad = \lim_{\epsilon\to 0} \frac{1}{\epsilon}(F(\mathbf{u}^k + \epsilon\delta\mathbf{u}^k, p^k + \epsilon\delta p^k) - F(\mathbf{u}^k, p^k)) \\ &\quad = \begin{pmatrix} \frac{\delta\mathbf{u}^k}{\Delta t} - \nu\Delta\delta\mathbf{u}^k + \mathbf{u}^k\cdot\nabla\delta\mathbf{u}^k + \delta\mathbf{u}^k\cdot\nabla\mathbf{u}^k + \nabla\delta p^k \\ -\nabla\cdot\delta\mathbf{u}^k \end{pmatrix} \end{aligned}$$

Therefore, we arrive at the linearized system:

$$\begin{cases} \frac{\delta\mathbf{u}^k}{\Delta t} - \nu\Delta\delta\mathbf{u}^k + \mathbf{u}^k\cdot\nabla\delta\mathbf{u}^k + \delta\mathbf{u}^k\cdot\nabla\mathbf{u}^k + \nabla\delta p^k = -F(\mathbf{x}^k) \\ -\nabla\cdot\delta\mathbf{u}^k = \nabla\cdot\mathbf{u}^k \end{cases}$$

The right-hand side of the second equation acts as a correction which leads the discrete solution of the velocity to be divergence-free along Newton's iteration. Now, Newton's iteration can be used to solve for the update terms:

Algorithm 1 Iterative Update for $\mathbf{u}$ and p

Initialization: Initial guess $\mathbf{u}_0$, p_0, tolerance τ.
$k \leftarrow 0$
repeat
  Linear solve to compute $\delta\mathbf{u}^k$ and δp^k.
  Update the approximation:
    $\mathbf{u}^{k+1} \leftarrow \mathbf{u}^k + \delta\mathbf{u}^k$
    $p^{k+1} \leftarrow p^k + \delta p^k$
  Check residual norm:
    $E^{k+1} \leftarrow \|F(\mathbf{u}^{k+1}, p^{k+1})\|$
  $k \leftarrow k + 1$
until $E^k \le \tau$

To enhance robustness, an adaptive damping parameter is applied to the update step.

2.5.2. Semi-Implicit Linearized Method

For time-dependent problems, it is possible to decouple the non-linearity from the implicit solve by linearizing the convective term using an extrapolated velocity field from previous time steps. This approach, often referred to as a semi-implicit or Oseen-type linearization, reduces the problem to a single linear system solve per time step, significantly reducing the computational cost compared to the Newton method.
The convective term is approximated as:

$$c(\mathbf{u}_h^{n+1}; \mathbf{u}_h^{n+1}, \mathbf{v}_h) \approx c(\mathbf{u}^*; \mathbf{u}_h^{n+1}, \mathbf{v}_h) \quad (6)$$

where $\mathbf{u}^*$ is an explicitly computed transport velocity. To preserve the second-order temporal accuracy of the Crank-Nicolson scheme, we employ a second-order extrapolation (Adams-Bashforth type) for $\mathbf{u}^*$:

$$\mathbf{u}^* = 2\mathbf{u}_h^n - \mathbf{u}_h^{n-1} \quad (7)$$

For the first time step, or when using the first-order Backward Euler scheme, the approximation simplifies to $\mathbf{u}^* = \mathbf{u}_h^n$. This method is conditionally stable but highly efficient for the range of Reynolds numbers investigated in this work ($\mathbb{R}e \le 100$).

2.5.3. Linearized system

Independently from which method we have chosen, at each time step these equations can be rewritten as a linear system of the form $\mathcal{A}\mathbf{x} = \mathbf{b}$ with:

$$\mathcal{A} = \begin{bmatrix} \mathbf{F} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \mathbf{U}^{n+1} \\ \mathbf{P}^{n+1} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \mathbf{G} \\ \mathbf{0} \end{bmatrix} \quad (8)$$

with $\mathbf{F} = \frac{1}{\Delta t}\mathbf{M} + \mathbf{A} + \mathbf{C}(\mathbf{U}^n)$, where $\mathbf{M}$ is the fluid mass matrix, $\mathbf{A}$ the stiffness matrix, and $\mathbf{C}(\mathbf{U}^n)$ the (linearized) convection terms of the momentum equation.
$\mathbf{B}$ and $\mathbf{B}^T$ are the discretized counterparts of the divergence operator and the gradient operator, respectively.
$\mathbf{U}^{n+1} = (u_1, \dots, u_{N_u})^T$ is the vector of the velocity unknowns at time $t = t^{n+1}$ and $\mathbf{P}^{n+1} = (p_1, \dots, p_{N_p})^T$ the one of the pressure unknowns.
$\mathbf{G}$ is a known vector depending on the discretized source force, on $\mathbf{U}^n$, and on $\mathbf{f}, \mathbf{g}$, and $\mathbf{h}$.

Remark. Formally when we use the Newton method the system we aim to solve is slightly different from the formulation in Equation (8), since we solve for the increments $\delta\mathbf{U}^k, \delta\mathbf{P}^k$.

2.6. Linear solver and preconditioning

To solve the system at each iteration, we employ the GMRES (Generalized Minimal Residual) method. Given the poor conditioning of the saddle-point matrix, an efficient block preconditioning strategy is mandatory. We utilize a block lower-triangular preconditioner $\mathcal{P}$ defined as:

$$\mathcal{P} = \begin{bmatrix} \hat{\mathbf{A}} & \mathbf{0} \\ \mathbf{B} & \hat{\mathbf{S}} \end{bmatrix} \quad (9)$$

where $\hat{\mathbf{A}}$ and $\hat{\mathbf{S}}$ are suitable approximations of the velocity block and the pressure Schur complement $\mathbf{S} = -\mathbf{B}\mathbf{A}^{-1}\mathbf{B}^T$, respectively.

2.6.1. Velocity block preconditioning

The velocity block $\mathbf{A}$ is non-symmetric due to the presence of the convective term. To provide a robust smoother that remains effective even at moderately high Reynolds numbers, we use an Incomplete LU (ILU) factorization for $\hat{\mathbf{A}}$. Specifically, the Trilinos `PreconditionILU` implementation is used to solve the velocity sub-system, providing a balance between memory consumption and approximation quality.

2.6.2. Cahouet-Chabard Schur complement approximation

The pressure block preconditioning is based on the Cahouet-Chabard approximation. This approach recognizes that the inverse of the Schur complement can be approximated by a combination of terms representing the mass-dominated and viscosity-dominated regimes:

$$\hat{\mathbf{S}}^{-1} \approx -\frac{\rho}{\Delta t}\mathbf{K}_p^{-1} - \theta\nu\mathbf{M}_p^{-1} \tag{10}$$

where $\mathbf{K}_p$ is the pressure Laplacian (stiffness) matrix and $\mathbf{M}_p$ is the pressure mass matrix. This strategy ensures that the GMRES convergence rate remains nearly independent of the mesh size h and the time step Δt. The components are handled as follows:

- for the pressure diffusion part ($\mathbf{K}_p$)), we utilize an Algebraic Multigrid (AMG) preconditioner, since the pressure Laplacian is a symmetric positive definite (SPD) elliptic operator. This allows for optimal scalability in parallel environments.
- for the pressure mass/reaction part ($\mathbf{M}_p$), that acts as the reaction part in the Schur complement approximation, we use the ILU factorization to ensure robust inversion.

The integration of these techniques within the `deal.II` framework, leveraging the Trilinos linear algebra library, allows the solver to handle complex 3D simulations with efficient MPI distribution.

2.7. Stabilization techniques

For the 3D benchmark cases and high-Reynolds number unsteady simulations using the linearized semi-implicit solver, the standard Galerkin formulation is augmented with stabilization terms to control spurious oscillations and improve mass conservation. We employ a combined Streamline Upwind/Petrov-Galerkin (SUPG) and Grad-Div stabilization strategy.

2.7.1. Streamline Upwind/Petrov-Galerkin

The SUPG method is implemented specifically within the linearized solver context. It modifies the test function space by adding a perturbation proportional to the streamline derivative of the test function. The stabilized weak form adds the following residual-based term to the formulation over each element K:

$$S_{\text{SUPG}} = \sum_{K\in\mathcal{T}_h} \left(\mathcal{R}_{\text{lin}}(\mathbf{u}_h^{n+1}, p_h^{n+1}), \tau(\mathbf{u}^*\cdot\nabla\mathbf{v}_h)\right)_K \tag{11}$$

where $\mathbf{u}^*$ is the extrapolated transport velocity. The linearized momentum residual $\mathcal{R}_{\text{lin}}$ included in the stabilization (11) is:

$$\mathcal{R}_{\text{lin}}(\mathbf{u}_h, p_h) \approx \frac{\mathbf{u}_h}{\Delta t} + \mathbf{u}^*\cdot\nabla\mathbf{u}_h + \nabla p_h - \mathbf{f} - \frac{\mathbf{u}_h^n}{\Delta t}$$

Note that the viscous term is neglected in the stabilization residual as is common for linear elements or convection-dominated flows.
The stabilization parameter τ_K is computed elementwise to balance the effects of the time step, local convection, and diffusion:

$$\tau_K = \left[\left(\frac{2}{\Delta t}\right)^2 + \left(\frac{2\|\mathbf{u}^*\|_2}{h_K}\right)^2 + \left(\frac{4\nu}{h_K^2}\right)^2\right]^{-\frac{1}{2}}$$

This specific definition ensures that the method transitions correctly between diffusion-dominated, convection-dominated, and transient-dominated regimes. The inclusion of the pressure gradient ∇p_h in the residual $\mathcal{R}_{\text{lin}}$ also provides a Pressure-Stabilizing Petrov-Galerkin (PSPG) effect, relaxing the compatibility constraints on the velocity-pressure spaces.

2.7.2. Grad-Div stabilization

To enforce the incompressibility constraint more strictly and improve the conditioning of the linear system, we add a Grad-Div penalization term. This term penalizes the divergence of the velocity field in the weak form:

$$S_{\text{GD}} = \sum_{K\in\mathcal{T}_h}\int_K \gamma(\nabla\cdot\mathbf{u}_h^{n+1})(\nabla\cdot\mathbf{v}_h)\,d\Omega$$

Based on numerical experiments and the implemented configuration, the stabilization parameter is set to $\gamma = 0.1$. This term is crucial for 3D simulations where mass conservation errors can accumulate more significantly than in 2D.

3. Force coefficients and benchmark configurations

3.1. Computation of lift and drag coefficients

When a body is immersed in a moving fluid, it experiences a resultant force due to the interaction between the fluid and the solid surface. In fluid dynamics, this force is decomposed into two orthogonal components relative to the direction of the flow velocity:

- the **drag force**, denoted as F_D, is defined as the component of the resultant force acting *parallel* to the direction of the uniform flow. This force represents the resistance the fluid offers to the motion of the body.
  Physically, it arises from two distinct contributions acting on the surface elements:
  - the *pressure drag*, that results from the distribution of normal pressure p over the body surface;
  - the *friction drag*, that results from the tangential shear stresses τ_w caused by the viscosity of the fluid acting along the surface;
- the **lift force**, denoted as F_L, is defined as the component of the resultant force acting *perpendicular* to the direction of the uniform flow.
  Lift is generated only when there is an asymmetry in the flow field: for a perfectly symmetric body aligned with the flow, the lift force is theoretically zero due to the symmetry of the pressure distribution.

Mathematically, the resultant force $\mathbf{F}$ acting on the body surface S is obtained by integrating the fluid stress tensor $\boldsymbol{\sigma}$ over the surface of the body, according to the formula:

$$\mathbf{F} = \int_S \mathbf{t}\, dS = \int_S \boldsymbol{\sigma} \cdot \mathbf{n}\, ds$$

For a Newtonian incompressible fluid, the stress tensor is defined as:

$$\boldsymbol{\sigma} = -p\mathbf{I} + \rho\nu(\nabla\mathbf{u} + (\nabla\mathbf{u})^T) \tag{12}$$

where $\mathbf{I}$ is the identity matrix, p is the pressure, ρ is the fluid density and ν is the kinematic viscosity.

Following the benchmark definitions provided by Schäfer and Turek, the drag and lift forces are computed by projecting the stress vector onto the Cartesian axes.
Let $\mathbf{n} = (n_x, n_y)$ be the outward unit normal vector to the surface S, and let $\mathbf{t} = (n_y, -n_x)$ be the tangential vector, which component on the surface is denoted by v_t. In the 2D configuration, we have that:

- the *drag force* F_D corresponds to the force component in the flow direction:

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

  Here, the term $-pn_x$ represents the contribution of pressure forces (pressure drag), while the term proportional to viscosity represents the friction forces exerted by the fluid on the body surface.

- the *lift force* F_L corresponds to the force component perpendicular to the flow direction (y-axis):

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

This logic remains the same in the 3D configuration, adapting the components to the rotated coordinate system used in the numerical implementation.
In the present 3D simulation, the cylinder axis is aligned with x, the flow is directed along z, and the lift is directed along y. Let $\mathbf{n} = (n_x, n_y, n_z)$ be the outward unit normal vector. By analogy with the 2D case (mapping the flow direction $x \to z$), we have:

- the *drag force* F_D which corresponds to the force component in the flow direction (in this case along the z-axes):

$$F_D = \int_S \left(\rho\nu \frac{\partial v_t}{\partial n} n_y - p n_z\right) dS$$

- the *lift force* F_L corresponds to the force component perpendicular to the flow direction (y-axis):

$$F_L = -\int_S \left(\rho\nu \frac{\partial v_t}{\partial n} n_z + p n_y\right) dS$$

While the original benchmark defines the flow along the x-axis for both 2D and 3D cases, in the present 3D numerical implementation the domain is oriented such that the flow aligns with the z-axis.
Consequently, for the 3D results presented here, the components along x in the drag formula are substituted with the components along z. The lift force remains along the y-axis, while the side force (now along x) is theoretically negligible due to symmetry.

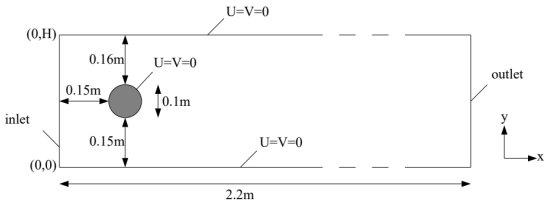

Figure 1: Geometry of 2D test cases with boundary conditions.

From the computed forces, the dimensionless drag and lift coefficients serve as primary validation metrics for the numerical solver, allowing for a quantitative comparison with the standard benchmark results provided by Schäfer and Turek.
The coefficients are defined as:

$$C_D = \frac{2F_D}{\rho \bar{U}^2 A} \quad \text{and} \quad C_L = \frac{2F_L}{\rho \bar{U}^2 A} \tag{13}$$

where $\bar{U}$ is the mean inflow velocity and A is the reference area (the diameter D for 2D cases and $D \times H$ for 3D cases).

3.2. Benchmark test cases

To validate the numerical solver, the simulations are configured according to the standard benchmark problems defined by Schäfer et al. (1996).
These proposes a set of 2D and 3D laminar flow test cases around a cylinder, designed to evaluate the accuracy and efficiency of different numerical approaches for the incompressible Navier-Stokes equations.

For all test cases, the fluid is assumed to be an incompressible Newtonian fluid. The fluid properties are kept constant, with a density of $\rho = 1.0$ kg/m^3 and a kinematic viscosity of $\nu = 10^{-3}$ m^2/s. The Reynolds number is defined based on the mean inflow velocity $\bar{U}$ and the cylinder diameter D as $\mathbb{R}e = \frac{\bar{U}D}{\nu}$.

3.2.1. 2D test cases

In the two-dimensional configuration, the flow passes around a cylinder with a circular cross-section, as we can seen in Figure 1.
The computational domain is a channel of height $H = 0.41$ m, and the cylinder has a diameter of $D = 0.1$ m. The cylinder is positioned slightly asymmetrically with respect to the horizontal centerline of the channel (0.15 m vs 0.16 m) to encourage the development of the von Kármán vortex street.
The mean velocity in 2D is given by $\bar{U} = \frac{2}{3}U(0, H/2, t)$.

Three distinct 2D test cases are considered:

- the test case 2D-1 (steady) with an inflow condition time-independent of:

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

  In this case the maximum velocity is equal to $U_m = 0.3$ m/s while the Reynolds number is $\mathbb{R}e = 20$;

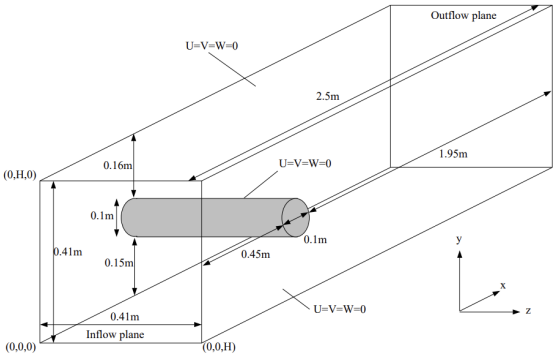

Figure 2: Configuration and boundary conditions for flow around a cylinder with circular cross–section.

- the test case 2D-2 (unsteady) with an higher Reynolds number of $\mathbb{R}e = 100$ and an increased velocity of $U_m = 1.5$ m/s. The inflow profile remains constant in time and equals to the test case 2D-1;
- the test case 2D-3 (unsteady) with an inlet velocity that varies over time, modulated by a sinusoidal function:

$$U(0, y, t) = \frac{4U_m y(H - y)\sin(\pi t/8)}{H^2} \text{ and } V = 0$$

for $0 \leq t \leq 8$ s. The maximum inlet velocity is $U_m = 1.5$ m/s, which means the instantaneous Reynolds number fluctuates smoothly in the range $0 \leq Re(t) \leq 100$.

3.2.2. 3D test cases

The three-dimensional test cases extend the geometry into a rectangular channel with both height and width equal to $H = 0.41$ m while the side length and diameter of the cylinder are $D = 0.1$ m[1]. The geometry is described in Figure 2.
The characteristic velocity is $\bar{U}(t) = \frac{4}{9}W(H/2, H/2, 0, t)$ where W is the component of the velocity along z[2] and the Reynolds number is defined by $\mathbb{R}e = \frac{\bar{U}D}{\nu}$.
Analogous to the 2D problem, the 3D circular cylinder test cases include:

- the test case 3D-1Z (steady) that regards a steady flow at $\mathbb{R}e = 20$. The spatial inflow profile is a 3D paraboloid:

$$\begin{cases} W(x, y, 0) = \frac{16U_m xy(H-x)(H-y)}{H^4} \\ U = V = 0 \end{cases}$$

with $U_m = 0.45$ m/s, yielding the Reynolds number $\mathbb{R}e = 20$;
- the test case 3D-2Z (unsteady) with a greater Reynolds number of $\mathbb{R}e = 100$, $U_m = 2.25$ m/s and an inflow condition equals to:

$$\begin{cases} W(x, y, 0, t) = \frac{16U_m xy(H-x)(H-y)}{H^4} \\ U = V = 0 \end{cases}$$

The initial data ($t = t_0$) are arbitrary for fully developed flow;
- the test case 3D-3Z (unsteady) where the 3D parabolic inlet profile is multiplied by a time-dependent sinusoidal term $\sin(\pi t/8)$:

$$\begin{cases} W(x, y, 0, t) = \frac{16U_m xy(H-x)(H-y)\sin(\pi t/8)}{H^4} \\ U = V = 0 \end{cases}$$

with $U_m = 2.25$ m/s. The time interval is $0 \leq t \leq 8$ s, that yields a time-varying Reynolds number between $0 \leq \mathbb{R}e(t) \leq 100$. The initial data ($t = 0$) are $U = V = P = 0$.

4. Implementation methodologies

The whole code-base has been structured in four different files:

- the file `NavierStokes.hpp` defines the solver's architecture, parameters and public API;
- the file `NavierStokes.cpp` contains the concrete implementation of the numerical methods, from setup to time-marching output;
- the file `TestCases.hpp` holds the physical and numerical parameters needed to simulate the Schäfer-Turek benchmarks;
- the file `main.cpp` is responsible for initializing the MPI environment, running the selected simulation.

4.1. NavierStokes.hpp

The header file `NavierStokes.hpp` contains the declarations for the physical parameters, the finite element data structures, the linear algebra objects and the block preconditioners required to solve the Navier-Stokes equations in parallel.

First of all, to ensure the modularity of the solver, we have imposed global enumerations in order to select the `TimeScheme` (backward Euler or Crank-Nicolson. in order to ensure stability) and the `NonLinearMethod` (Newton or linearized basing on the Reynolds number that we are considering).

The solver then utilizes a series of specialized classes derived from the `Function<dim>` to manage the physics of the problem.
The `InletVelocity` class implements the spatial flow profile required by the benchmark: it adapts between the 2D and the 3D distribution, with the optimal temporal sinusoidal modulation $\sin(\pi t/8)$ when the time-dependent flag is active (for the test case 2D-3 and 3D-3Z).
We also add some helper classes like `ZeroDirichletBC`, `ForcingTerm` and `InitialCondition`, that provide default implementations respectively for homogeneous constraints, unforced system or stationary fluid state.

In order to manage the different test cases, we implement the `BenchmarkTestCase` structure that packages all runtime metadata, including mesh file paths, polynomial degrees for velocity and pressure, the Reynolds number ($\mathbb{R}e$), the maximum velocity (U_m) and pointers to the appropriate boundary functions.

[1] We have chosen to study only the case with the circular cross-section and not the square cross-section one.
[2] This is different from Schäfer et al. (1996) since, as we have reported in the previous paragraph, we are considering the channel flow with the z-axis

Given the saddle-point nature of the Navier-Stokes equations, the `PreconditionBlockTriangular` class implements a Cahouet–Chabard Schur complement approximation discussed in Section 2.6.
It utilizes Trilinos wrappers to apply an incomplete LU (ILU) factorization to the velocity block and the pressure mass matrix ($\mathbf{M}_p$), while an Algebraic Multigrid (AMG) preconditioner is applied to the pressure Laplacian ($\mathbf{K}_p$). The approximation is dynamically weighted by the fluid density ρ, kinematic viscosity ν, time step Δt, and the θ parameter of the time-stepping scheme as shown in Equation (10). It ensure mesh-independent convergence by addressing both mass-dominated and viscosity-dominated flow regimes.

The main `NavierStokes<dim>` class manages the full simulation, with the `setup()`, `run()` and `output()` phases, as we can see in Figure 3.
In details, it initializes a `FESystem` that enforces the Taylor-Hood element formulation, necessary to satisfy the discrete Inf-Sup condition. Using `TrilinosWrappers`, the class declares `BlockSparseMatrix` and `MPI::BlockVector` objects, with the aim of separating the velocity and pressure degrees of freedom, which is mandatory for the block preconditioning strategy.
The class also declares separate routines for the nonlinear approaches (using `assemble_newton_system` and `solve_newton_system`) and the semi-implicit approaches (using `assemble_linearized_system` and `solve_linear_system`), allowing the algorithm to use the chosen methodology. To facilitate the second-order extrapolation of the convective velocity (as described in Equation (7)), multiple solution vectors are maintained (`solution_old`, `solution_old_old` and `current_solution`).

In addition to this, in this class, methods such as `compute_lift_drag` and `compute_pressure_difference` are declared to calculate the objective metrics of the Schäfer-Turek benchmark at every time step.

4.2. `NavierStokes.cpp`

The `NavierStokes.cpp` file contains the core computational logic of the solver. The implementation is divided into six main operational phases: setup, system assembly, solver execution, post-processing, output and run.

4.2.1. `setup()` phase

The `setup()` method initializes as the numerical foundation of the solver, transforming the continuous problem into a discrete algebraic structure.

First of all, the solver implements a multi-stage mesh ingestion process to support fully distributed triangulations. Since large-scale distributed meshes often cannot be read directly from standard formats, the routine first reads the grid into a serial triangulation, partitions it across the available MPI ranks and then constructs the `parallel::fullydistributed` mesh from these descriptions.
To ensure compatibility with Gmsh files, we converted the `$ParametricNodes` header to standard `$Nodes`,

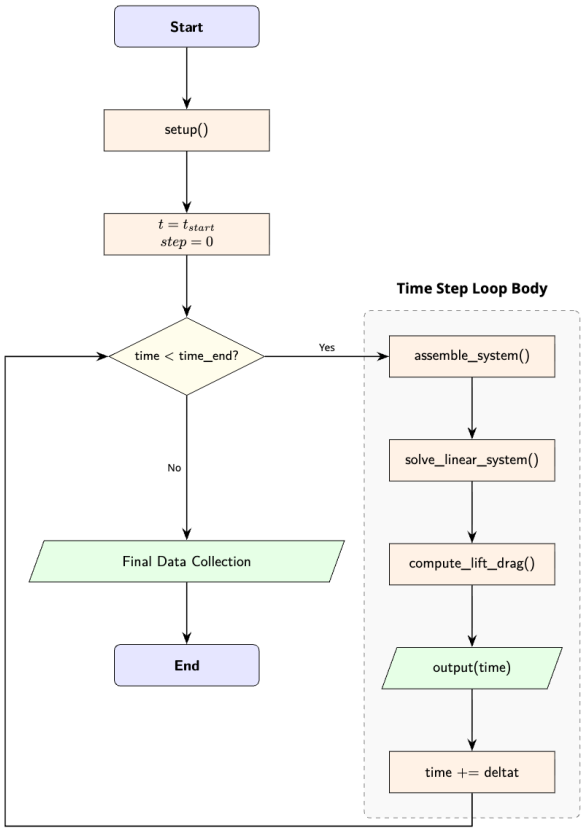

Figure 3: Algorithmic structure of the solver, highlighting the initialization, assembly and linear system solution phases per time step.

allowing the coordinates to be parsed safely.
Furthermore, we included a geometric safety check: if the expected boundary IDs (inlet, outlet, walls and cylinder) are missing or incomplete locally, the solver reassigns them based on the spatial coordinates of the face centers.

Once the geometry is established, the solver derives the kinematic viscosity from the Reynolds number and characteristic velocity parameters, following the instructions provided in Section 3.2.
The `DoFHandler` then distributes the finite element spaces across the mesh. In details, it organizes the degrees of freedom into distinct blocks for velocity and pressure.

To manage distributed-memory parallelism, the solver distinguishes between locally owned and locally relevant degrees of freedom. The owned sets are used for linear algebra ownership during the solve phase, while the relevant sets include *ghost* DoFs necessary for assembly and accessing solution values on adjacent cell partitions.

For the Newton path, the solver builds a set of homogeneous Dirichlet constraints on the velocity components for the inlet, channel walls and cylinder surface. Since the Newton solver is designed to compute the incremental update $\delta\mathbf{u}$ rather than the full velocity field, these constraints must be zero to ensure that the non-homogeneous physical values (such as the parabolic

inlet profile) already applied to the current solution guess are strictly preserved throughout the iterative process.

Finally, the global sparsity pattern is generated and distributed across ranks: without it, off-process matrix entries would be silently dropped during the matrix compression phase, leading to a corrupted Jacobian and solver failure.

4.2.2. The assemble system phase

In this specific phase, we need to consider two mutually exclusive assembling methods, as described in the Section 2.5. Depending on the kind of test case we are running, we use either the Newton's method or the semi-implicit linearization method. For this reason we have build one method each.

Assembling Newton System:
The `assemble_newton_system` function implements the construction of the Jacobian matrix and the residual vector required for the nonlinear iteration loop. It employs a θ-weighted temporal discretization, allowing for the consistent treatment of terms in both Backward Euler and Crank-Nicolson schemes.

Iterating over all the active cells within the computational mesh, the solver extracts the values for the velocities, the pressures and their respective spatial gradients and Laplacians. It does this for both the current Newton iteration's guess and the one obtained from the previous time step.

For every degree of freedom within that specific cell, the solver computes the residual and the Jacobian matrix, according to the theory discussed in Section 2.5.1.
The residual (right-hand side) is defined as:

$$
\begin{aligned}
-F(\cdot) = \\
\int_\Omega \Bigg\{ & -\frac{\mathbf{u}_k - \mathbf{u}^n}{\Delta t} \cdot \mathbf{v} \\
& - \theta \Big[\nu \nabla \mathbf{u}_k : \nabla \mathbf{v} + ((\mathbf{u}_k \cdot \nabla)\mathbf{u}_k) \cdot \mathbf{v} \Big] \\
& - (1-\theta) \Big[\nu \nabla \mathbf{u}^n : \nabla \mathbf{v} + ((\mathbf{u}^n \cdot \nabla)\mathbf{u}^n) \cdot \mathbf{v} \Big] \\
& + p_k (\nabla \cdot \mathbf{v}) + q(\nabla \cdot \mathbf{u}_k) \\
& + \Big[\theta \mathbf{f}^{n+1} + (1-\theta)\mathbf{f}^n \Big] \cdot \mathbf{v} \Bigg\} d\Omega
\end{aligned}
$$

The Jacobian matrix is computed in the following way:

$$
\begin{aligned}
J(\cdot) = \int_\Omega \Bigg\{ & \frac{1}{\Delta t} \delta \mathbf{u} \cdot \mathbf{v} \\
& + \theta \nu \nabla \delta \mathbf{u} : \nabla \mathbf{v} \\
& + \theta \Big[(\mathbf{u}_k \cdot \nabla)\delta \mathbf{u} + (\delta \mathbf{u} \cdot \nabla)\mathbf{u}_k \Big] \cdot \mathbf{v} \\
& - \delta p (\nabla \cdot \mathbf{v}) \\
& - q(\nabla \cdot \delta \mathbf{u}) \Bigg\} d\Omega
\end{aligned}
$$

where $(\mathbf{u}_k \cdot \nabla)\delta\mathbf{u} + (\delta\mathbf{u} \cdot \nabla)\mathbf{u}_k$ represents the linearization of the non-linear convective term $(\mathbf{u} \cdot \nabla)\mathbf{u}$.

During this step, the solver also constructs isolated pressure matrices that will be used later to accelerate the linear solver.

After the local cell integration is complete, the solver identifies the global indices for the cell's degrees of freedom. It then maps and adds the local contributions into the global matrices and vectors, ensuring that physical boundary conditions are strictly enforced.

Assembling semi-implicit System:
The `assemble_linearized_system` function constructs a single linear system per time step to avoid the computational overhead of full Newton iterations.
Like the Newton variant, it employs a θ-weighted scheme but differs fundamentally by linearizing the convective term through an extrapolated transport velocity ($\mathbf{u}^*$) rather than an iterative Jacobian, as described in details in Section 2.5.2.

Local assembly within each cell begins with the construction of non-homogeneous `system_constraints`. Unlike the homogeneous updates used in Newton's method, these constraints directly incorporate the physical values from the `inlet_velocity` function and homogeneous conditions at the walls and cylinder boundaries.

Then the solver initializes the parallel loop over the cells owned by the local processor. For each cell, it extracts the solution from the previous time step ($\mathbf{u}^n$) and, if available, the one from the step before that ($\mathbf{u}^{n-1}$).
During the loop on the quadrature points, it calculates $\mathbf{u}^*$:

- if we are in the first step, second step or we are using backward Euler method, it is computed as $\mathbf{u}^* = \mathbf{u}^n$ (first-order extrapolation);
- otherwise, it is computed as $\mathbf{u}^* = 2\mathbf{u}^n - \mathbf{u}^{n-1}$ (according to the Equation (7)).
  If the extrapolated velocity grows more than 20% compared to the previous step, the solver returns to the first-order extrapolation to prevent numerical blow-ups.

The right-hand side is computed according to the following equation:

$$
\begin{aligned}
F(\mathbf{v}) = \int_\Omega \Bigg\{ & \frac{1}{\Delta t}\mathbf{u}^n \cdot \mathbf{v} \\
& -(1-\theta)\Big[\nu \nabla \mathbf{u}^n : \nabla \mathbf{v} \\
& +((\mathbf{u}^n \cdot \nabla)\mathbf{u}^n) \cdot \mathbf{v} \Big] \\
& + \Big[\theta \mathbf{f}^{n+1} + (1-\theta)\mathbf{f}^n \Big] \cdot \mathbf{v} \Bigg\} d\Omega
\end{aligned}
$$

while the system matrix uses the semi-implicit lineariza-

tion:

$$A(\mathbf{u}, p; \mathbf{v}, q) = \int_\Omega \Big\{ \frac{1}{\Delta t}\mathbf{u}\cdot\mathbf{v} + \theta\nu\nabla\mathbf{u} : \nabla\mathbf{v} + \theta\Big((\mathbf{u}^*\cdot\nabla)\mathbf{u}\Big)\cdot\mathbf{v} - p(\nabla\cdot\mathbf{v}) - q(\nabla\cdot\mathbf{u})\Big\} d\Omega$$

In addition of this, we included the local scaling parameter in order to use the SUPG and Grad-Div stabilization techniques described in Section 2.7. This was implemented to maintain stability with high-Reynolds number.

4.2.3. The solver execution phase

In this phase, as the one before, we consider two different solving methods that depend on the method used to solve the linearization problem.

Solving Newton System:
The `solve_newton_system()` method handles the algebraic resolution of the linear system generated during each Newton iteration to find the update vector $\delta\mathbf{u}$.

First, it calculates the L^2 norm of the `system_rhs`, that represents the stopping criterion used to determine convergence. The `SolverControl` is configured with a maximum of 500 iterations and with a relative tolerance of 10^{-2} times the initial residual norm.
This means the linear solver terminates when the internal residual is reduced to 1% of the initial residual for that specific Newton iteration.

To handle the ill-conditioning of the saddle-point system, we used the `PreconditionBlockTriangular` preconditioner, that follows the Cahouet-Chabard approach, described in the Section 2.6.

The linear system is solved using the Generalized Minimal Residual (GMRES) method, with a restart value of 150 to maintain efficiency and stability. The solver evaluates the `system_matrix` and the `preconditioner` to compute the `newton_update` vector (which corresponds to the increments $\delta\mathbf{u}^k$ and δp^k).

Then, we applied the `newtonconstraints` to the solution to ensure that the homogeneous Dirichlet boundary conditions are strictly enforced on the update vector.

Solving semi-implicit System:
The `solve_linear_system()` method performs the algebraic resolution of the system generated during the assemble phase.
Similar to the Newton approach, the solver utilizes the `SolverGMRES` algorithm with a restart value of 150 and a `SolverControl` object configured with a relative tolerance of 10^{-2}.
As in the previous case, we used the `PreconditionBlockTriangular` class.

The distinction with the Newton's method solver is that we wrapped the `solver.solve` in a `try-catch` block. If GMRES fails to converge, the function catches the `SolverControl::NoConvergence` exception, issues a warning and flags the step as unconverged while retaining the best available approximation. This allows the simulation to proceed with that best available approximation.

Another key difference is the application of the `system_constraints` to the resulting `solution_owned` vector. Unlike the homogeneous constraints used in the Newton method, these constraints impose the non-homogeneous Dirichlet boundary conditions. Such as the inlet velocity profile in order to ensure that the final discrete field is physically consistent.

4.2.4. The post-processing phase

In the post-processing phase, we initialize two different methods to compute two different quantitaties: the pressure difference and the drag and lift coefficients.

Computing the pressure difference:
The function `compute_pressure_difference` calculates the pressure drop between the upstream (front) and downstream (back) points of the cylinder. We considered the evaluation coordinates described in Schäfer et al. (1996):

- for the 2D configurations $p_{\text{front}} = (0.15, 0.2)$ and $p_{\text{end}} = (0.25, 0.2)$;
- for the 3D configurations $p_{\text{front}} = (0.205, 0.2, 0.40)$ and $p_{\text{end}} = (0.205, 0.2, 0.50)$.

The solver uses a lambda function that attempts to read the local pressure via `VectorToools::point_value`. These local results are then aggregated using a global reduction (`Utilities::MPI::sum`). The final pressure is averaged across the ranks that successfully located the point to prevent double-counting at partition boundaries.

Computing the drag and lift coefficients:
The `compute_lift_drag` function is responsible for calculating the forces exerted by the fluid and deriving their dimensionless coefficients, as described in details in the Section 3.1.

The function iterates exclusively over locally owned cells, isolating the ones that lie on the `cilynder_boundary_id`. Only for these a `FEFaceValues` object evaluates the pressure p, the velocity gradient tensor $\nabla\mathbf{u}$, the outward unit normal vector $\mathbf{n}$ and the quadrature weights JxW.
The solver, then, constructs the fluid stress tensor, as described in Equation (12), and computes the localized force vector exerted on the cylinder as $-\boldsymbol{\sigma}\cdot\mathbf{n}$.
These local traction contributions are numerically integrated across the face and accumulated into directional force variables (F_x, F_y and, if we are in 3D configuration, F_z). Then a global reduction is done to aggregate the total force vector over the entire mesh.

These forces are then normalized to calculate the drag and lift coefficients, according to the Equation (13).
Consistent with the domain orientation discussed in Section 3.1, the drag force is mapped to F_x for 2D domains

and F_z for 3D domains, while the lift force is consistently extracted from the y-component.

4.2.5. The `output()` method

The `output()` method is responsible for exporting the simulation results at discrete time steps, formatting the data for post-processing and visualization in Paraview.

The `current_solution` vector is partitioned in two parts: the first d components are treated as a d-dimensional vector field, called "velocity", while the final component is treated as a distinct scalar field, called "pressure".

The spatial representation of the data is generated via the `build_patches(*mapping)` function, which projects the finite element solution onto the graphical elements. Finally, the method `write_vtu_with_pvtu_record` writes the output to disk.

4.2.6. The `run()` method

The `run()` method is needed to merging together the setup phase, the time-stepping loop, the numerical solvers and the output phase.

First of all, the method calls the `setup()` to configure the distributed mesh and algebraic structures. The initial conditions are interpolated into the discrete velocity and pressure vectors.

At the beginning of the `while (time < T)` loop, needed to advance the physical time by Δt at each iteration, we implemented a numerical stabilization: if the Crank-Nicolson scheme ($\theta = 0.5$) is selected, the solver temporarily overrides the parameter to Backward Euler ($\theta = 1.0$) exclusively for the first time step. This choice is motivated by the zero initial conditions; using $\theta = 0.5$ at the first step would lead to a discontinuity in the function, potentially resulting in numerical instability or excessively large gradient values.

Then, we have implemented two different solution strategies based on the `nonlinear_method`:

- for the Newton's method, the boundary conditions are applied to the current solution and a `while` loop is employed to minimize the error. To keep this solver stable, it uses an adaptive damping strategy: if the error grows, the solver takes smaller steps.
  If the GMRES linear solver fails, the code simply goes back to the previous state and tries again with a much smaller damping factor;
- for the semi-implicit linear method, it automatically adjusts the time step and saves the system's previous state. It first tries to solve the system using the current time step, Δt, but, if the GMRES solver fails to converge, it reloads the saved state and cuts the time step in half (allowing up to 4 reductions). If all these attempts fail, the solver uses a safer, first-order Backward Euler method as a last resort to push past the numerical issues.

At the end of a successful time step, the solution vectors are shifted forward in time. The solver computes the benchmark metrics and used the `output()` method to write the results.

4.3. `TestCases.hpp`

The `TestCases.hpp` defines the specific physical and numerical parameters required to accurately replicate the established 2D and 3D test cases.

First of all, the file implements the `BenchmarkInletVelocity` class, which prescribes the exact parabolic inflow profile for the 2D and 3D configurations, adding, for the 2D-3 and 3D-3Z cases, the time-dependent modulation $\sin(\pi t/8)$.
We also implemented a smooth temporal ramp (a half-cosine function) to prevent numerical divergence in the linear solvers during the start-up phase of impulsively started unsteady cases, specifically 2D-2 and 3D-2Z.

We have then implemented several functions in the `BenchmarkTestCase` structure, each of them providing the setup for the specific test cases, defining:

- the physical constants, such as the Reynolds number $\mathbb{R}e$ and the maximum velocity U_m;
- the finite element polynomial degrees for velocity and pressure;
- the linearized method and the time method chosen for that simulation;
- the optimal time step Δt.

4.4. `main.cpp`

The `main.cpp` file handles environment setup, benchmark configuration and execute the solver.
In details, the user defines the path to the computational mesh and instantiates a scenario using factory functions from the `TestCases` namespace.
This process creates an instance of the `NavierStokes` solver template and initiates the simulation by calling the `run()` method.

5. Computational aspects

Prior to utilizing our personal hardware, we conducted preliminary tests on the cluster provided by the MOX department at Politecnico di Milano. However, these simulations yielded a lower speedup compared to our local machines. Consequently we opted to use our specific hardware to ensure optimal performance.

5.1. Hardware specifications

The numerical simulations were executed on a workstation equipped with an AMD Ryzen 5 7600X 6-Core Processor (clocked at 4.70 GHz) paired with 32.0 GB of system memory (4800 MT/s).

5.2. 2D benchmark performance

The two-dimensional test cases used the following computational costs across all tested regimes:

- the test 2D-1 (with steady flow and $\mathbb{R}e = 20$) was executed with a time step of $\Delta t = 0.1$ seconds, with a simulation that was extremely stable. The computational cost averaged approximately 13.5 seconds per iteration, characterized by a highly consistent execution times with a variance of less than one seconds;

- in the test 2D-2 (with unsteady flow, constant inlet and a $\mathbb{R}e = 100$), we used a reduced time step of $\Delta t = 0.01$ seconds. The system maintained high stability, averaging 3.2 seconds with minimal variance (≤ 0.3 seconds);

- the last test 2D-3 (characterized by unsteady flow, time-varying inlet and $0 \leq \mathbb{R}e \leq 100$) was executed with $\Delta t = 0.01$ seconds. Also in this case, the solver is stable, with an average of 3.0 seconds per iteration. The variance slightly increased to approximately 1 second, reflecting the dynamic nature of the time-dependent sinusoidal boundary conditions.

eIt is important to justify the choice of the time steps. For the first case (2D-1), we used a larger time step of $\Delta t = 0.1$ seconds because the flow safely converges to a stationary state and the unconditionally stable Backward Euler scheme is employed. This allows us to efficiently accelerate the transition to the steady-state without stability issues.
Conversely, for the other two cases (2D-2 and 2D-3), we used a significantly reduced time step of $\Delta t = 0.01$ seconds. This choice is required both physically, to accurately capture the high-frequency dynamics of the vortex shedding, and numerically, because the semi-implicit Linearized solver relies on an explicit second-order extrapolation for the convective velocity, introducing a strict stability restriction.

5.3. 3D benchmark performance and challenges

About the three-dimensional domain, this introduced spikes in computational complexity:

- the test 3D-1Z (with steady flow and $\mathbb{R}e = 20$) provides highly stable. The robust convergence properties allow for a significantly finer spatial discretization compared to the unsteady subsequent 3D tests. Using $\Delta t = 0.01$, the wall time is approximately 10 seconds per iteration;

- the test 3D-2Z (characterized with an unsteady flow, a constant inlet and $\mathbb{R}e = 100$) contains some computational complexity.
  During the initial transient phase, the impulsive start from the inlet velocity introduces severe instabilities, causing the linear solver to struggle with convergence. In order to manage this, we deliberately restricted the maximum number of Krylov solver iterations. This forced the system to trigger in our robust fallback mechanism (as, for example, the adaptive time-stepping or 1st-order Backward Euler overrides) during these critical early stages.
  Once the initial transient subsides, the semi-implicit (linearized) formulation and its history-based preconditioner perform optimally, stabilizing the system. The execution time averages 12 seconds per iteration, with occasional peaks up to 40 seconds, likely driven by strong convective fluxes that temporarily ill-condition the system matrices. The baseline time step remains $\Delta t = 0.01$.

  Remark. Attempting this test on a more complex mesh strictly requires a reduced time step of $\Delta t = 0.0025$, which escalated the computational cost to approximately 95 seconds per iteration.
  This specific run was aborted as it saturated the 6-core processor's hardware limits. Executing such high-resolution 3D simulations to completion necessitates an HPC environment with a higher core count;

- the final test 3D-3Z (with unsteady flow, time-varying inlet e $\mathbb{R}e = 100$) has a performance profile very similar to the previous one. Also the overall stability and computational cost are similar, though the wall time per iteration exhibits slightly higher variability due to the continuously fluctuating sinusoidal inflow conditions.

6. Numerical results

6.1. Drag and lift analysis for the 2D configuration

Following the definitions of the benchmark test cases provided in Section 3.2.1, the numerical results for the two-dimensional configurations are evaluated by analyzing the temporal evolution of the drag coefficients (C_D) and the lift coefficients (C_L), computed as stated in the Equation (13).
The plots in Figure 4 for the drag coefficient and in Figure 5 for the lift coefficient illustrate the responses of the system under all the steady and time-varying inflow conditions studied.

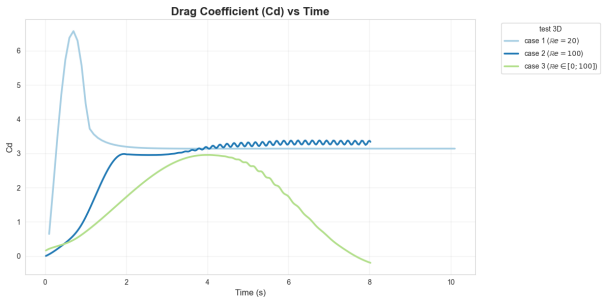

Figure 4: Temporal evolution of the drag coefficient (C_D) for the three 2D benchmark test cases.

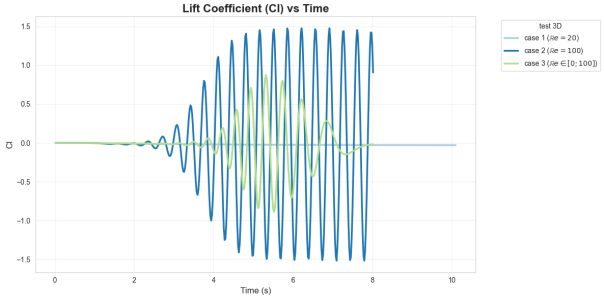

Figure 5: Temporal evolution of the lift coefficient (C_L) for the three 2D benchmark test cases.

6.1.1. Test 2D-1

Starting with the numerical results of the steady test case 2D-1, the simulation accurately captures the transition of the fluid from an initial state of rest to a fully developed, steady laminar flow.

In the Figure 6, 7 and 8 we have the rendered velocity magnitude extracted, respectively, at $t = 1$, $t = 4$ and $t = 8$.

Figure 6: Rendered velocity for the test case 2D-1 at time $t = 1.0$.

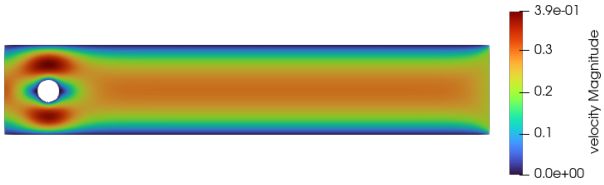

Figure 7: Rendered velocity for the test case 2D-1 at time $t = 4.0$.

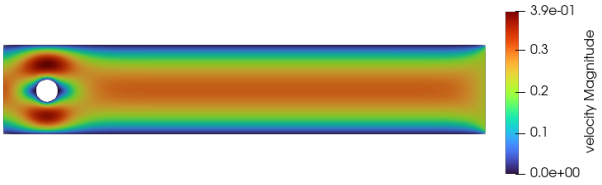

Figure 8: Rendered velocity for the test case 2D-1 at time $t = 8.0$.

As observed in the velocity renders, the flow accelerates smoothly around the cylinder.
The maximum velocity peaks at approximately 0.39 m/s (as indicated by the color scales) in the narrowed cross-sections above and below the obstacle. By $t = 4$ s, the velocity field has already reached a steady configuration, which is fully maintained at $t = 8$ s without any visible variations.

We can see that the wake behind the cylinder remains elongated and symmetric. At the Reynolds number of $\mathbb{R}e = 20$, the flow remains strictly within the stationary laminar regime.
Under these conditions, the viscous forces are dominant enough to suppress flow instabilities, meaning the critical threshold for the von Kármán vortex street has not been reached and no vortex shedding is observed[3].

[3] Tipycally, unsteady vortex shedding begins at a Reynolds number of approximately 40. Up to this, the boundary layer separates and two counterrotating vortices form in the wake.

This stable convergence is confirmed by the analysis of the aerodynamic force coefficients.
The drag coefficient for 2D-1 test case (light blue line in Figure 4) exhibits an initial spike due to the impulsive start and the rapid acceleration of the fluid from rest, before smoothly and rapidly converging to a steady-state value of approximately 3.1.

On the other hand, the lift coefficient, shows no periodic oscillations and, while remaining at a mathematically very low order of magnitude, does not completely vanish: it settles to a constant negative value.
This non-zero lift is a direct physical consequence of the slight geometric asymmetry in the cylinder's placement within the channel (0.15 m from the bottom wall versus 0.16 m from the top wall as we can see in Figure 1), which structurally prevents a perfectly symmetric pressure distribution around the obstacle.

6.1.2. Test 2D-2

Shifting the analysis to the unsteady test case 2D-2, we analyze the case with a Reynolds number equals to $\mathbb{R}e = 100$.
As in the previous case, we reported the velocity at three different times: $t = 1.0$ (Figure 9), $t = 3.6$ (Figure 10) and $t = 7.0$ (Figure 11).

Figure 9: Rendered velocity for the test case 2D-2 at time $t = 1.0$.

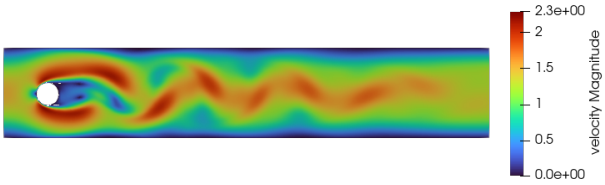

Figure 10: Rendered velocity for the test case 2D-2 at time $t = 3.6$

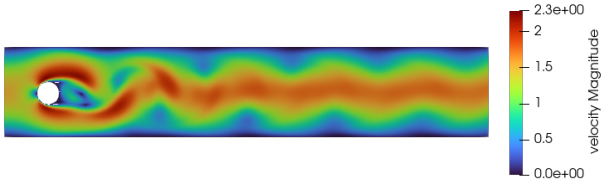

Figure 11: Rendered velocity for the test case 2D-2 at time $t = 7.0$.

The velocity magnitude fields illustrate the flow's progression and its instability at this Reynolds number. Starting from a uniform zero-velocity field, the fluid

accelerates around the cylinder forming initially a symmetric and elongated wake (Figure 9).
Then, since $\mathbb{R} = 100$ is above the critical threshold defined before, the geometric asymmetry of the channel acts as a perturbation. This triggers the periodic detachment of alternating vortices from the top and bottom surfaces of the cylinder, generating a fully developed von Kármán vortex street, as visible in Figures 10 and 11.
Peak velocities reach approximately 2.3 m/s in the narrow passages bypassing the cylinder.

This dynamic behavior is strictly reflected in the force coefficients. Looking at the dark blue line in the graphs, the drag coefficient in Figure 4 initially rises and stabilizes near 3.0 during the symmetric phase. As the vortex shedding begins (around $t \approx 3$ s), the drag starts to oscillate, eventually settling into a steady periodic oscillation around a mean value of approximately 3.2.

The lift coefficient in Figure 5, initially close to zero, experiences an exponential growth in amplitude as the instability amplifies, finally reaching a stable limit cycle. The lift oscillates symmetrically with a considerable amplitude (between roughly $+1.5$ and -1.5)[4].

6.1.3. Test 2D-3

Concluding the analysis with the case 2D-3, the simulation captures the hydrodynamics of a flow driven by a time-varying inlet velocity.
The rendered velocity are shown in Figure 12 (at time $t = 1.0$), Figure 13 (at time $t = 3.6$) and in Figure 14 (at time $t = 5.8$).

Figure 12: Rendered velocity for the test case 2D-3 at time $t = 1.0$.

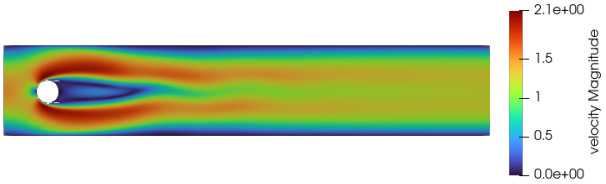

Figure 13: Rendered velocity for the test case 2D-3 at time $t = 3.6$

At the initial phase with $t = 1.0$ s (Figure 12), the flow is in its initial acceleration phase, due to the sinusoidal modulation of the inlet velocity profile $\sin(\pi t/8)$. The instantaneous Reynolds number is below the critical threshold, resulting in a symmetric and attached wake behind the cylinder.

[4]This trend is consistent with the physics of the Schäfer et al. (1996), which reports a reference maximum lift coefficient $C_{L,max} \approx 1.0$ for the 2D-2 configuration.

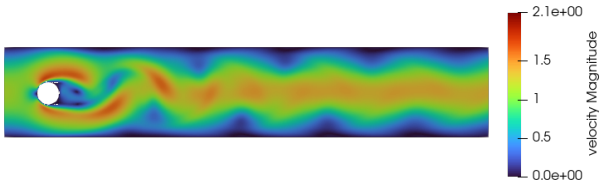

Figure 14: Rendered velocity for the test case 2D-3 at time $t = 5.8$.

As the simulation progresses towards $t = 3.6$ s (Figure 13), the inlet velocity approaches its maximum peak (which occurs at $t = 4.0$ s, where $Re \approx 100$). The wake elongates and the velocity in the narrow passages above and below the cylinder reaches its maximum magnitude. Even if the Reynolds number is high enough to trigger the instability, the vortex shedding has not fully developed yet.
At $t = 5.8$ s (Figure 14), the inlet velocity is in its decelerating phase. The accumulated flow instability, combined with the geometric asymmetry of the channel, has triggered the alternating detachment of vortices, forming a clear developed von Kármán vortex street.

This behaviour is reflected in the green curves in Figure 4 and 5.
About the drag coefficient, we can see that it follows the macroscopic acceleration and deceleration of the flow. It rises from zero, reaching a peak of 3.0 around $t = 4.0$ s, corresponding, as said few lines above, to the maximum flow inertia. During the decelerating phase, the drag gradually decreases, exhibiting small oscillations, as a consequence of the vortex shedding phenomenon.
On the other hand, the lift coefficient remains virtually zero during the initial acceleration phase. Once the Reynolds number arrives at $\mathbb{R}e \approx 100$, the period instability begins. The amplitude of the lift oscillations grows rapidly, peaking between $t = 5.0$ s and $t = 6.0$ s (with maximum values around ± 0.9). Then, it decreases as the velocity reaches zero towards $t = 8.0$ s.

To validate our results, we compared the temporal evolution of the coefficients with the results obtained by recent literature, utilizing the same benchmark defined by Schäfer et al. (1996).
The trends observed in our simulations are in agreement with the finding reported by de Frutos et al. (2019). Indeed, we can see that the drag coefficient closely follows the inlet velocity, while the lift oscillations exhibit the expected delay, growth and eventual dampening.
This confirmed the accuracy of our numerical methods and stabilization strategies in dynamic regimes.

6.2. Drag and lift analysis for the 3D configuration

We conclude our analysis considering the numerical results for the three-dimensional configurations, described in 3.2.2.
The plots in Figure 15 and in Figure 16 represent the temporal evolution of the drag coefficients and the lift coefficients for the test cases 3D-1Z, 3D-2Z and 3D-3Z.

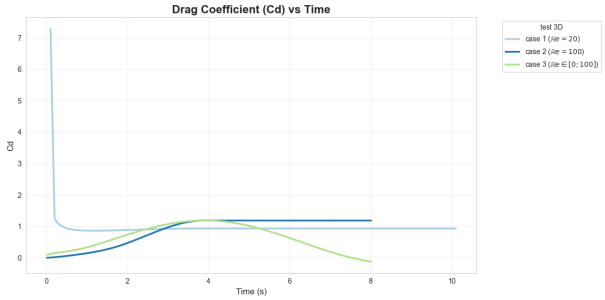

Figure 15: Temporal evolution of the drag coefficient (C_D) for the three 3D benchmark test cases.

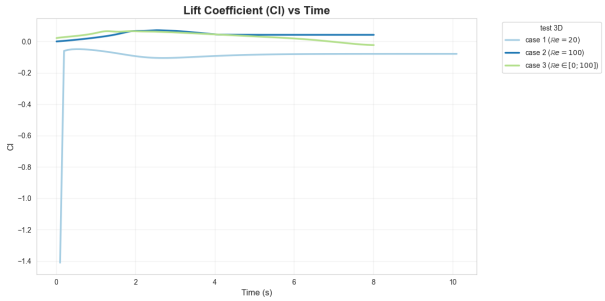

Figure 16: Temporal evolution of the lift coefficient (C_L) for the three 3D benchmark test cases.

6.2.1. Test 3D-1Z

We begin our analysis for the 3D configuration with the steady test case 3D-1Z.
In the Figure 17, 18 and 19 we have the rendered velocity magnitude extracted, respectively, at $t = 1.0$, $t = 4.0$ and $t = 8.0$.

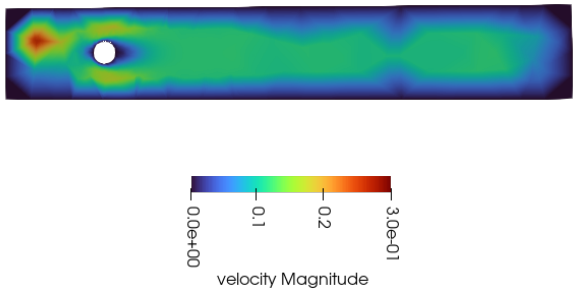

Figure 17: Rendered velocity for the test case 3D-1Z at time $t = 1.0$.

As observed in the rendered velocity magnitude fields, the flow starts from rest and smoothly accelerates around the cylinder, with the wake that is still developing in Figure 17 at time $t = 1.0$.
By $t = 4.0$ (Figure 18), the fluid has reached a fully developed steady state, which is perfectly maintained at $t = 8.0$ (Figure 19) without any visible variations. The slight asymmetry observed in the wake is a direct physical consequence of the cylinder placement, as defined in the benchmark configuration (Figure 2).

Probably the velocity asymmetry between the upward and downward flow is caused by the grid being too

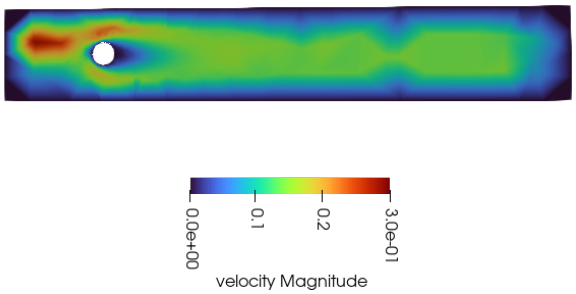

Figure 18: Rendered velocity for the test case 3D-1Z at time $t = 4.0$.

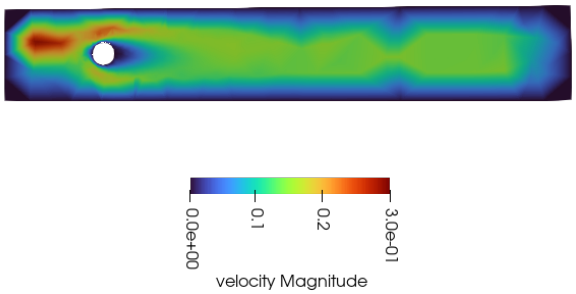

Figure 19: Rendered velocity for the test case 3D-1Z at time $t = 8.0$.

coarse, meaning that the subdomains are not sufficiently refined. This issue can be observed in all 3D benchmark cases.

Similar to the 2D-1 case, since the Reynolds number is strictly $\mathbb{R}e = 20$, the flow remains within the stationary laminar regime, avoiding the formation of a von Kármán vortex street. The maximum velocity peaks around 0.30 m/s in the narrowed cross-sections bypassing the cylinder.

This stable behavior is quantitatively confirmed by the temporal evolution of the force coefficients. Looking at the respective curves Figure 15 and Figure 16, the drag and the lift coefficients exhibit an initial transient phase due to the start-up conditions, but rapidly and smoothly converge to a constant steady-state value.

6.2.2. Test 3D-2Z

Continuing the analysis to the unsteady test case 3D-3Z, we analyze the case with a Reynolds number equals to $\mathbb{R}e = 100$.
Following the previous methodology, the velocity field is examined at four distinct instances: $t = 0.5$, $t = 1.8$, $t = 3.5$ and $t = 8.0$ as illustrated in Figures 20, 21, 22 and 23 respectively.

At the earliest stage in Figure 20, the fluid is beginning to accelerate. Then, as the inlet velocity ramps up (Figure 21), the wake starts elongate in a way that seems symmetric.
By $t = 3.5$ s (represented in Figure 22), nearing the end of the velocity ramp, the wake is distinctly elongated and the velocity bypasses the cylinder reaching the peak of 2.8 m/s.

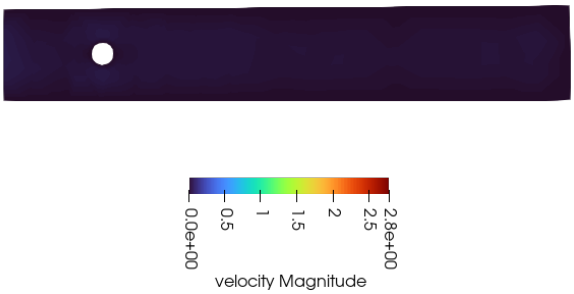

Figure 20: Rendered velocity for the test case 3D-2Z at time $t = 0.5$.

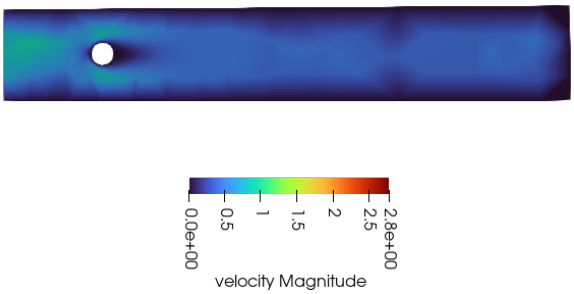

Figure 21: Rendered velocity for the test case 3D-2Z at time $t = 1.8$

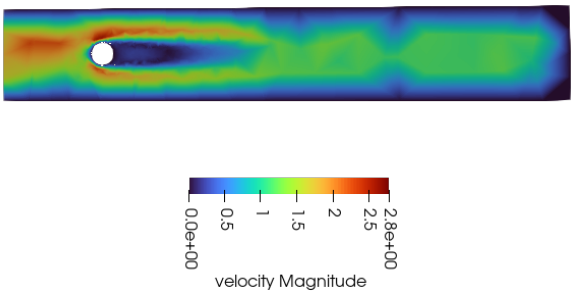

Figure 22: Rendered velocity for the test case 3D-2Z at time $t = 3.5$.

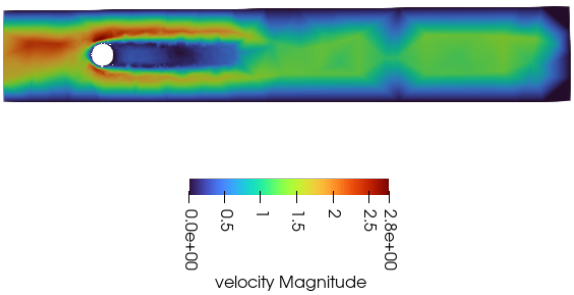

Figure 23: Rendered velocity for the test case 3D-2Z at time $t = 8.0$.

At the final simulated time of $t = 8.0$ s (Figure 23), the flow has reached its maximum velocity and fully developed.

While in the 2D configuration, a Reynolds number of $\mathbb{R}e = 100$ was sufficient to trigger a fully developed von Kármán vortex street, the 3D-2Z results exhibit a stationary and stable wake. We thought that this behavior could be attributed to two main factors:

- due to hardware limitations, the 3D simulation was relatively coarse spatial discretization;
- the stabilizing effect of the 3D geometry inherently suppresses the instability at this specific Reynolds number.
  This exact difficulty is acknowledged in the reference benchmark. As explicitly stated by Schäfer et al. (1996), "*for the nonstationary 3D problems a higher Reynolds number should be considered, since in the present case ($\mathbb{R}e = 100$) the problem may be particularly hard as the flow tends to become almost stationary*". The benchmark authors themselves noted that achieving reliable reference solutions for the 3D-2Z test case was problematic, validating the stationary nature of our result.

The temporal evolution of the drag coefficient (in dark blue in Figure 15) reflects the behavior of the velocity. Indeed, the drag coefficient increases smoothly from zero, until it reaches its maximum around $t = 4.0$. After this point, instead of entering a periodic oscillatory state, the drag plateaus and remains perfectly constant at a value of approximately 1.2.
About the lift coefficient, the dark blue line in Figure 16, shows a slight initial adjustment but quickly dampens and stabilizes practically at zero. The complete absence of sinusoidal oscillations confirms that no periodic vortex shedding is occurring.

6.2.3. Test 3D-3Z

Concluding the analysis with the 3D-3Z case, the rendered velocity are shown in Figure 24 (at time $t = 0.5$), Figure 25 (at time $t = 1.8$), Figure 26 (at time $t = 2.9$), Figure 27 (at time $t = 6.0$) and Figure 28 (at time $t = 8.0$).

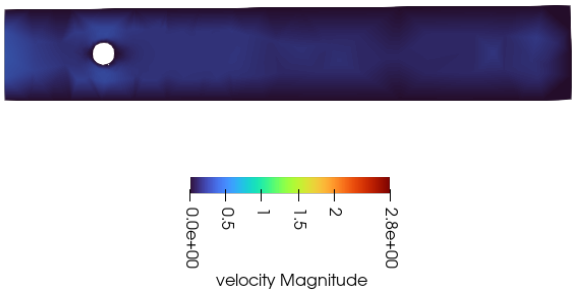

Figure 24: Rendered velocity for the test case 3D-3Z at time $t = 0.5$.

At $t = 0.5$, the overall velocity magnitude is extremely low and the fluid is only marginally disturbed by the presence of the obstacle, with no visible wake formed yet.
Then, during the accelerating phase (as shown in Figure 25 and Figure 26), the wake behind the cylinder gradually elongates, with the maximum velocity bypassing the obstacle progressively increases.
During the decelerating phase ($t = 6.0$ s in Figure 27), the overall kinetic energy begins to drop as the inlet

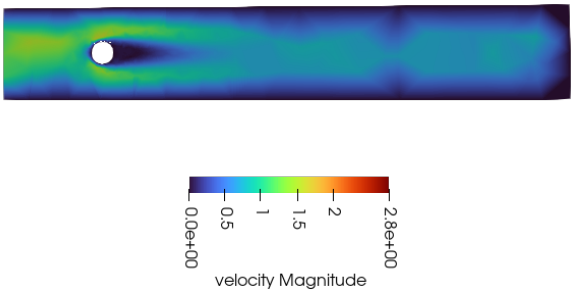

Figure 25: Rendered velocity for the test case 3D-3Z at time $t = 1.8$

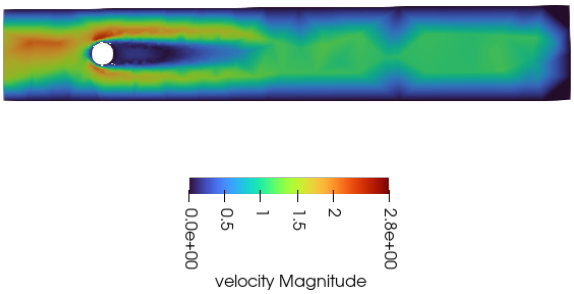

Figure 26: Rendered velocity for the test case 3D-3Z at time $t = 2.9$.

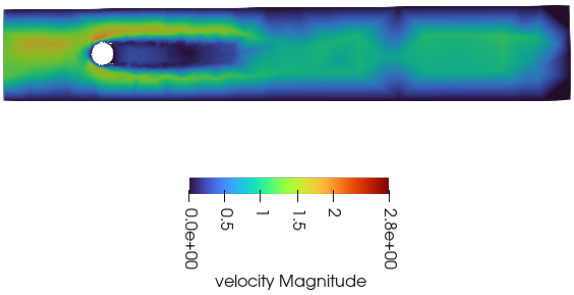

Figure 27: Rendered velocity for the test case 3D-3Z at time $t = 6.0$

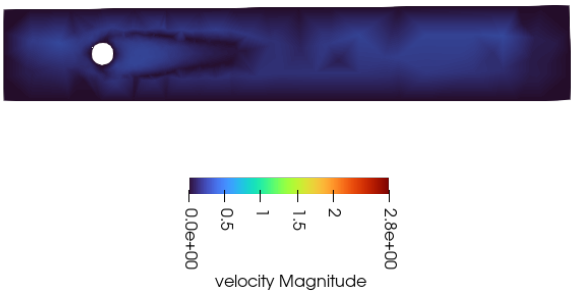

Figure 28: Rendered velocity for the test case 3D-3Z at time $t = 8.0$.

velocity decreases.
At the end of the simulation ($t = 8.0$ s), the fluid velocity returns toward zero and the wake dissipates, as can be seen in Figure 28.

Unlike the corresponding 2D-3 configuration, where the instability during the decelaration phase triggered a von Kármán vortex street, in this case the wake remains symmetric and attached to the cylinder.
As said in the previous section about the test case 3D-2Z, this is probably caused by the spatial discretization and the general difficulty for the 3D cases, cited by Schäfer et al. (1996).

We can see that the light green curve in the drag plot (Figure 15) smoothly rises from zero, closely tracking the sinusoidal modulation of the inlet velocity, and peaks at exactly $t = 4.0$ (where the flow reaches $\mathbb{R}e = 100$) with a value of approximately 1.2. Afterward, it smoothly descends back toward zero, exhibiting absolutely no high-frequency oscillations[5].
The lift coefficient (light green line in the Figure 16) remains virtually zero throughout the entire simulation, with a minimal non-oscillating deviation observed as a consequence of the slight geometric asymmetry of the cylinder's vertical placement.

7. Conclusion

This project successfully implemented a robust parallel finite element solver for the incompressible Navier-Stokes equations, effectively simulating laminar flow past a cylinder in accordance with the standard 2D and 3D benchmarks defined by Schäfer et al. (1996).

For the two-dimensional configurations, the solver accurately captured both the steady-state flow at $\mathbb{R}e = 20$ and the fully developed von Kármán vortex street at $\mathbb{R}e = 100$, producing drag and lift coefficients that align with existing literature.

In the three-dimensional cases, the steady regime was successfully reproduced; however, due to limited computational resources, the numerical results lacked perfect symmetry. While this could be mitigated by a finer mesh, the associated computational cost would grow exponentially.
The unsteady configurations at $\mathbb{R}e = 100$ exhibited a stable and stationary wake without periodic vortex shedding. This behavior is attributed to the inherent stabilizing effect of the 3D geometry and the numerical dissipation introduced by the necessarily coarse spatial discretization, a recognized challenge that was also noted by the original benchmark authors.

References

Alessandro Bottaro. Meccanica dei Continui – Capitolo 11: Equazioni di Navier-Stokes. `http://www.dicat.unige.it/bottaro/cm_files/Chapter_11.pdf`, 2020. Lecture Notes.

Javier de Frutos, Bosco García-Archilla, and Julia Novo. Fully discrete approximations to the time-dependent Navier-Stokes equations with a projection method in time and grad-div stabilization. *Journal of Scientific Computing*, 80(2):1330–1368, 2019. doi: 10.1007/s10915-019-00980-9.

Simone Deparis, Gwenol Grandperrin, and Alfio Quar-

[5] While the accelerating phase is very similar to what happens in the 2D-3 test case, the absence of oscillations in the deceleraing phase is caused by the absence of the von Kármán vortex street.

teroni. Parallel preconditioners for the unsteady Navier-Stokes equations and applications to hemodynamics simulations. *Computers & Fluids*, 92:253–273, 2014. doi: 10.1016/j.compfluid.2013.10.033.

Joel Guerrero. Flow past a cylinder: From laminar to turbulent flow. Wolf Dynamics Tutorial, 2020. OpenFOAM Introductory Training.

Bärbel Janssen and Georg Kanschat. The deal.II tutorial: Step-57 – Navier-Stokes equations. `https://dealii.org/current/doxygen/deal.II/step_57.html`, 2024. Accessed: 2024-05-22.

Yasuki Nakayama. *Introduction to Fluid Mechanics*. Butterworth-Heinemann, 2nd edition, 2018. ISBN 978-0-08-102437-9. doi: 10.1016/B978-0-08-102437-9.00009-7.

Michael Schäfer, Stefan Turek, Franz Durst, Egon Krause, and Rolf Rannacher. Benchmark computations of laminar flow around a cylinder. In *Flow Simulation with High-Performance Computers II*, volume 52 of *Notes on Numerical Fluid Mechanics (NNFM)*, pages 547–566. Vieweg+Teubner Verlag, Wiesbaden, 1996. doi: 10.1007/978-3-322-89849-4_39.