Navier-Stokes Solver for Laminar Flow in 3D: CLL231 Project Report

Aditya Vishwakarma (2018CH10191)

This report outlines the process I followed to create a 3D Navier-Stokes solver for simulating incompressible, laminar flow within a channel. The project uses the Navier-Stokes equations and the Finite Volume Method (FVM) to discretize and solve the flow equations. The goal was to better understand both the underlying physics of fluid dynamics and the numerical techniques required to simulate it.

1. Introduction

Computational Fluid Dynamics (CFD) is crucial for analyzing fluid behavior. In this project, I focused on simulating laminar flow, where the fluid particles move smoothly in parallel layers. The Navier-Stokes equations are at the core of this simulation and describe the motion of fluid particles under various forces.

2. Theoretical Background

2.1 Navier-Stokes Equations

I worked with the incompressible form of the Navier-Stokes equations, which consist of two main components:

- **Momentum Equation:**

  $$\rho(\partial u/\partial t + u\cdot\nabla u) = -\nabla p + \mu\nabla^2 u + f$$

  Here, ρ represents fluid density, u is the velocity vector (u, v, w), and μ is the dynamic viscosity. The term f represents external forces, which I left out for simplicity in this project. The equation essentially balances inertia, pressure gradients, and viscous forces.

- **Continuity Equation:**

  $$\nabla\cdot u = 0$$

  This equation ensures mass conservation, a critical aspect of incompressible flow.

I used the SIMPLE algorithm for pressure-velocity coupling because the continuity equation doesn't directly solve for pressure, and SIMPLE is a well-known method to handle this by iteratively correcting pressure and velocity.

2.2 Finite Volume Method (FVM)

To implement the solver, I used the Finite Volume Method (FVM), which divides the computational domain into control volumes. By integrating the Navier-Stokes equations

over each control volume, I was able to discretize the problem.

Key steps I followed:

1. **Discretization**: I divided the 3D domain into a grid of control volumes.
2. **Integration**: I integrated the Navier-Stokes equations over each control volume.
3. **Interpolation**: I used central differencing to approximate fluxes at the control volume faces.

2.3 Data Structures

For the implementation, I designed the following data structures in C++ to handle the grid and fields:

- **Grid3D**: This class represents the grid in the 3D domain.

```
class Grid3D {
public:
    int nx, ny, nz; // Number of grid points in each direction
    double dx, dy, dz; // Grid spacing
};
```

- **VectorField3D**: This class stores the velocity components.

```
class VectorField3D {
public:
    Grid3D& grid;
    std::vector<double> u, v, w; // Velocity components
};
```

- **ScalarField3D**: This class stores the scalar fields like pressure.

```
class ScalarField3D {
public:
    Grid3D& grid;
    std::vector<double> p; // Pressure
};
```

2.4 Boundary Conditions

I applied boundary conditions to define the flow problem. The types I used include:

- **Dirichlet Conditions**: For example, I set `u = 0` at walls to enforce the no-slip condition.
- **Neumann Conditions**: I used these to specify gradients, like setting `∂p/∂n = 0` at the outlet.
- **Inlet/Outlet Conditions**: I defined a fixed velocity at the inlet and allowed outflow at the outlet.

3. Coding Implementation

3.1 Discretization

I started by discretizing the momentum and continuity equations using FVM. For example, I discretized the x-component of the momentum equation as:

```
ρ(u_i,j^(n+1) - u_i,j^n)/Δt + ... = - (p_i+1,j - p_i-1,j)/(2Δx) + ...
```

From there, I extended the equations to 3D by including the y and z components. The boundary conditions I defined earlier were incorporated into the discretization.

3.2 SIMPLE Algorithm

The SIMPLE algorithm played a central role in my solver. The steps I followed were:

1. **Initialization**: I provided initial guesses for the velocity and pressure fields.
2. **Momentum Prediction**: I solved the momentum equations using the initial pressure guess to predict the velocity.
3. **Pressure Correction**: I solved the pressure correction equation derived from the continuity equation.
4. **Velocity Correction**: Using the corrected pressure, I updated the velocity field.
5. **Update**: I updated the pressure and velocity fields.
6. **Iteration**: I repeated the above steps until convergence was reached.

3.3 Sparse Matrix Operations

Since the system of equations I dealt with is sparse, I used libraries like Eigen for efficient sparse matrix storage and operations. Using iterative solvers like Conjugate Gradient (CG) and Generalized Minimal Residual Method (GMRES) helped in solving the linear systems efficiently. Preconditioners like Incomplete LU (ILU) and Jacobi were essential for accelerating convergence.

4. Visualization

To visualize the results, I exported the data and used Paraview, which allowed me to plot velocity and pressure fields in 3D. Paraview was particularly useful for understanding how the flow developed over time and across the domain.

5. Testing and Validation

I validated my solver by comparing it to two well-known cases:

- **Poiseuille Flow**: I compared the velocity profile to the analytical solution for laminar flow in a channel and found good agreement.
- **Lid-Driven Cavity Flow**: I simulated the lid-driven cavity problem, which is a standard benchmark in CFD. The results were consistent with published data, confirming the accuracy of my solver.

6. Further Enhancements

While the solver works well for the current scope, there are several areas for improvement that I could explore in future work:

- **PISO Algorithm**: A more advanced pressure-velocity coupling algorithm that could improve convergence speed.
- **Higher-Order Discretization**: Instead of central differencing, I could explore higher-order schemes for better accuracy.
- **Complex Boundary Conditions**: Implementing more realistic boundary conditions could make the solver applicable to more complex scenarios.
- **Non-Uniform/Adaptive Grids**: This would allow for finer resolution in areas where it's needed, improving accuracy and efficiency.

For now, I've avoided advanced turbulence models like RANS or LES, as they are beyond the scope of this project.

This project has deepened my understanding of CFD, particularly in the areas of numerical methods and fluid dynamics. The implementation of the Navier-Stokes solver provided valuable experience in applying theoretical concepts to a practical problem, and I look forward to further refining and expanding this work in the future.