FOSSEE, IIT Bombay
Scilab Hackathon, 2025

A Scilab-Based CFD Solver for 2D Incompressible Flow Using the SIMPLE Algorithm

Pranjali Digambar Main[1]

[1]Dept. of Aerospace Engineering, Amity University Mumbai, Panvel, Maharashtra 410206

Abstract

Computational Fluid Dynamics (CFD) plays a crucial role in simulating and analysing fluid flow in engineering applications. This study presents the development and implementation of a two-dimensional CFD solver in Scilab 2025 using the Finite Volume Method (FVM) and the Semi-Implicit Method for Pressure-Linked Equations (SIMPLE) algorithm. The solver is designed for incompressible, laminar flow in a rectangular channel, discretized using a structured Cartesian grid. Governing equations, including the Navier-Stokes and continuity equations, are discretized using a central difference scheme for diffusion terms and an upwind scheme for convection terms. The iterative solution process is validated by monitoring the convergence of velocity and pressure residuals. Results indicate smooth convergence, demonstrating the accuracy and stability of the numerical scheme. Divergence is also found when parameters are changed or provided abrupt, thus indicating the physics logic implemented is accurate. Simulation data, including velocity, pressure, and residuals, are exported in CSV format for further analysis. The solver provides a cost-effective and open-source alternative for CFD simulations, making it accessible for research and educational purposes.

Keywords: Computational Fluid Dynamics, Finite Volume Method, SIMPLE Algorithm, Scilab, Incompressible Flow, Numerical Simulation

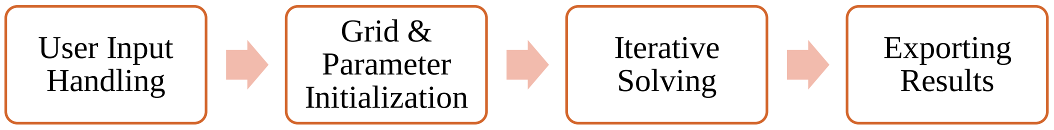


Fig 1 : Diagram of Solver Implementation

Chapter 1 : Introduction

CFD Workflow: Key Stages

Computational Fluid Dynamics (CFD) is a powerful numerical tool used to simulate and analyze fluid flow behavior across various engineering domains, including aerospace, mechanical, automotive, and biomedical applications. It provides an alternative to costly and time-consuming physical experiments by solving the governing equations of fluid motion, primarily the Navier-Stokes equations, along with additional transport equations for energy, turbulence, and chemical reactions when necessary. By leveraging advanced numerical techniques, CFD helps engineers optimize designs, improve efficiency, and gain deeper insights into complex flow phenomena.

Pre-processing: Geometry, Meshing, and Boundary Conditions

- The first step in any CFD simulation is pre-processing, which involves defining the problem and preparing it for numerical analysis. This stage begins with geometry creation, where a digital representation of the physical domain. Once the geometry is defined, the domain is discretized into smaller control volumes through a process called meshing or grid generation. The type of mesh used significantly impacts the accuracy and computational efficiency of the simulation. Structured meshes, with their uniform grid patterns, offer high precision but are often unsuitable for complex geometries. For the simplicity our solution will run on structured meshes. Finally, the boundary conditions and initial conditions are specified to define fluid interactions with its surroundings. These may include inlet velocity profiles, outlet pressure conditions, and no-slip conditions at solid walls, ensuring that the simulation accurately represents real-world behavior.

Solution Process: Governing Equations and Numerical Methods

After pre-processing, the solution process involves solving governing equations using discretization methods like FVM, FDM, or FEM, with FVM being the most common due to its conservation properties. For incompressible flows, velocity-pressure coupling is handled using algorithms like SIMPLE and PISO. Turbulence models such as k-ε, k-ω, and LES approximate chaotic fluid behavior. The solver iterates until convergence, monitored through residuals, while under-relaxation factors help maintain stability and prevent divergence. We'll continue with FVM as mode of Discretization and SIMPLE will be the algorithm.

Post-processing: Analysis and Validation of Results

After convergence, post-processing involves analyzing simulation results through velocity and pressure distributions, identifying flow features like vortices and boundary layer separation. Visualization tools generate contour plots and streamlines for better interpretation. Performance metrics coefficients evaluate design effectiveness. Finally, validation and verification compare results with experiments or industry software like OpenFOAM and ANSYS to ensure accuracy and real-world applicability.

The below figures help us in understanding physics of fluid dynamics in visualized manner. Fluid Flow is governed by Navier-Stokes condition. For simulation purposes, various terms of the same equation can be ignored like dynamic viscosity, etc for convergence, or faster convergence. A numerical model is convergent if and only if a sequence of model solutions with increasingly refined solution domains approaches a fixed value.

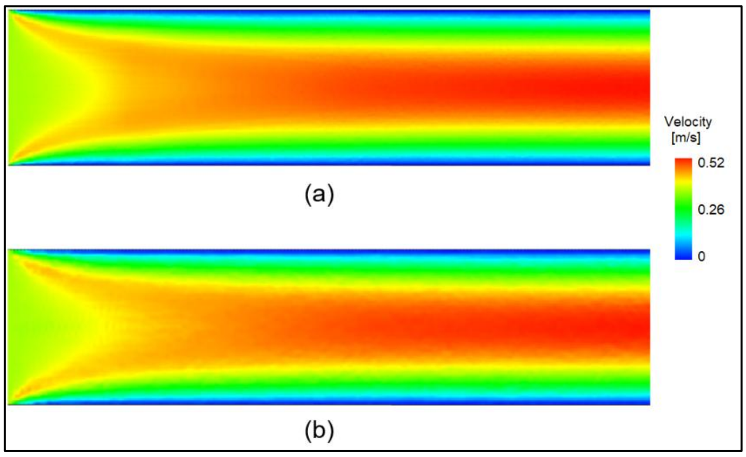


Fig 2 : 2D Pipe Laminar Flow Visualization of Velocity

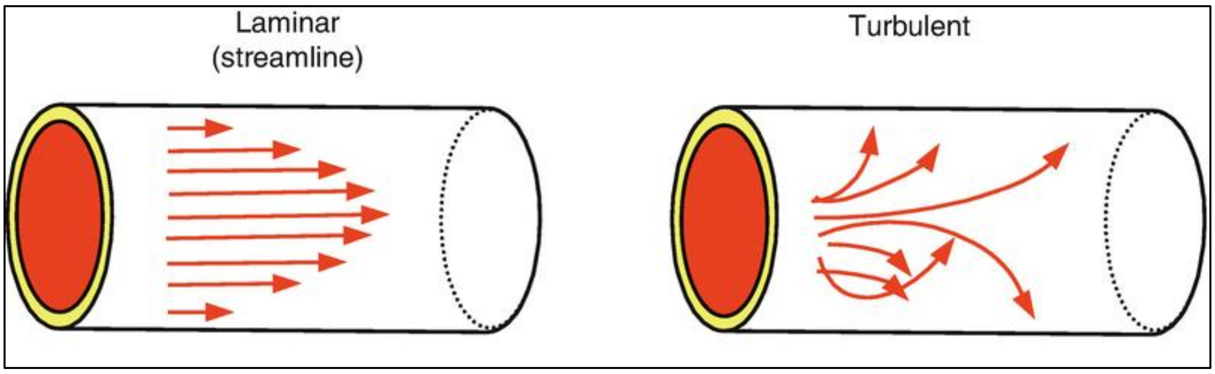


Fig 3 : Flow Types

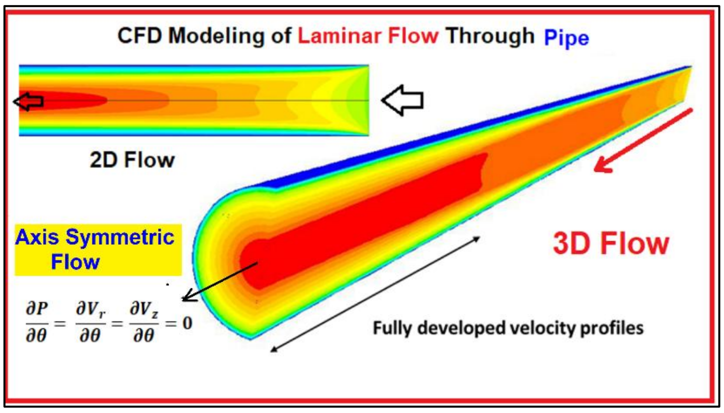

CFD Modeling of Laminar Flow Through Pipe
2D Flow
Axis Symmetric Flow
$\frac{\partial P}{\partial \theta} = \frac{\partial V_r}{\partial \theta} = \frac{\partial V_z}{\partial \theta} = 0$
3D Flow
Fully developed velocity profiles

Fig 4 : CFD Modelling of 3D Pipe

Chapter 2 : Problem Statement

Implementation of CFD Solver in Scilab

Building upon the foundational concepts of CFD, this section focuses on implementing a solver in Scilab for incompressible, steady-state flow using the Finite Volume Method (FVM) and the SIMPLE algorithm. Scilab, an open-source numerical computation platform, provides robust matrix operations and scripting flexibility, making it well-suited for CFD development.

Problem Definition and Discretization

The solver is designed to simulate 2D flow in a channel with prescribed inlet velocity and outlet pressure. The governing equations—the continuity and Navier-Stokes equations—are discretized using FVM. A structured grid is employed for simplicity, with control volumes centered around grid points. Central differencing is used for diffusion terms to maintain accuracy, while an upwind scheme ensures numerical stability for convection-dominated flows.

Staggered Grid:

In a staggered grid, velocity components are stored at the cell faces, while pressure and scalar quantities are stored at the cell centers. This arrangement reduces pressure-velocity decoupling and improves numerical stability. It is commonly used in structured grid solvers like SIMPLE and MAC methods.

Collocated Grid:

In a collocated grid, all flow variables (pressure, velocity, and temperature) are stored at the same location, typically the cell center. This simplifies data storage but can lead to numerical issues like pressure-velocity decoupling. Special interpolation techniques, such as the Rhie-Chow correction, are used to address these issues.

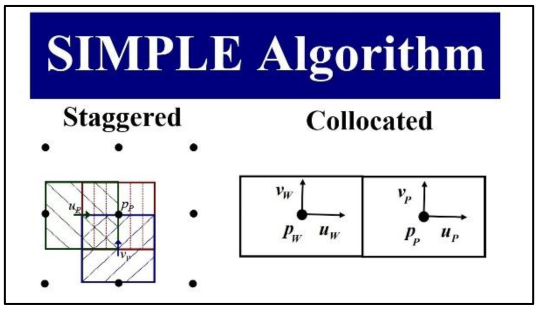


Fig 5a : SIMPLE Algorithm

The figure illustrates the difference between staggered and collocated grids in the SIMPLE algorithm for computational fluid dynamics (CFD). On the left, the staggered grid arrangement shows pressure stored at the cell center, while velocity components are stored at the faces of the control volume. This setup helps in avoiding pressure-velocity decoupling and improves numerical stability. On the right, the collocated grid stores all variables, including velocity and pressure at the same cell center, requiring special interpolation techniques to prevent oscillations in the solution.

Numerical Implementation in Scilab

The solver follows a modular approach, breaking down the process into distinct functions:

- **Grid Generation:** Constructs a structured mesh with user-defined grid resolution.
- **Boundary Condition Enforcement:** Implements Dirichlet conditions for velocity and Neumann conditions for pressure where required.
- **Pressure-Velocity Coupling:** The SIMPLE algorithm iteratively corrects pressure and velocity fields to satisfy mass conservation.
- **Solver Execution:** Residuals are monitored, and under-relaxation factors are applied to enhance stability and ensure convergence.

Scilab's built-in solvers handle the matrix equations efficiently, and iterative loops refine the solution until the residuals drop below a predefined threshold.

Post-Processing and Validation

Since Scilab primarily supports text-based output, velocity and pressure distributions are exported for visualization in external tools like spreadsheets or Python-based libraries. Validation is performed by comparing the results with established CFD solvers such as ANSYS, ensuring accuracy and reliability. This implementation highlights the feasibility of using Scilab for CFD simulations, demonstrating its capabilities in solving real-world fluid dynamics problems. Future work could extend the solver to handle turbulence modeling, transient flows, and complex geometries.

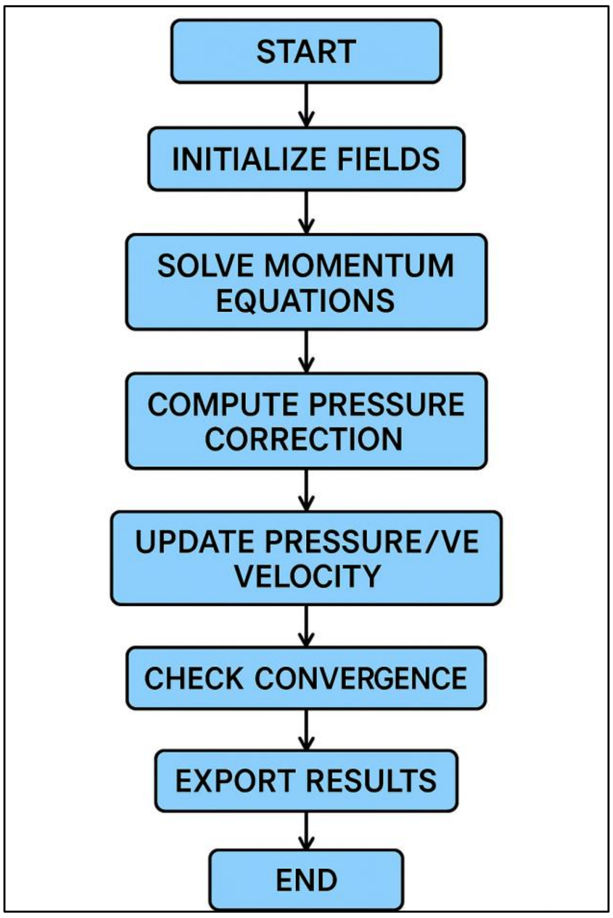


Fig 5b : SIMPLE Algorithm

Chapter 3 : Basic Concepts Related to Topic

Computational Fluid Dynamics (CFD) is a numerical approach to solving and analyzing problems involving fluid flow by applying the governing equations of motion—namely the Navier-Stokes equations and the continuity equation. These equations describe the conservation of mass and momentum in a fluid and are fundamental to understanding flow behavior.

In this project, the Finite Volume Method (FVM) is employed as the discretization technique. FVM is preferred in CFD for its conservation properties—it ensures that fluxes entering and exiting a control volume are balanced, which is critical in fluid mechanics. The computational domain is divided into structured Cartesian grid cells, and the governing equations are integrated over each cell.

To couple pressure and velocity in incompressible flows, we use the SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) algorithm. The SIMPLE algorithm is an iterative method that starts by solving momentum equations to obtain intermediate velocity fields, then solves a pressure correction equation to enforce mass conservation, and finally updates the pressure and velocity fields accordingly. This pressure-velocity coupling is essential because, in incompressible flows, the pressure does not have its own governing equation and must be inferred indirectly.

The central difference scheme is used for discretizing the diffusive terms, providing second-order accuracy, while the upwind scheme is used for convective terms to ensure numerical stability in advection-dominated flows.

The solver is based on a collocated grid layout, where all variables—pressure, velocity, and temperature—are stored at the same cell center. Although simpler to implement than a staggered grid, this arrangement can lead to pressure-velocity decoupling, which is mitigated using interpolation techniques such as the Rhie-Chow correction.

In summary, this project integrates key CFD concepts:

- Navier-Stokes and continuity equations for modeling fluid flow.
- Finite Volume Method for spatial discretization.
- SIMPLE algorithm for pressure-velocity coupling.
- Numerical schemes like central difference and upwind.
- Structured, collocated grid layout for domain representation.

These foundational principles enable the development of a robust, open-source CFD solver in Scilab.

Chapter 4 : Flowchart

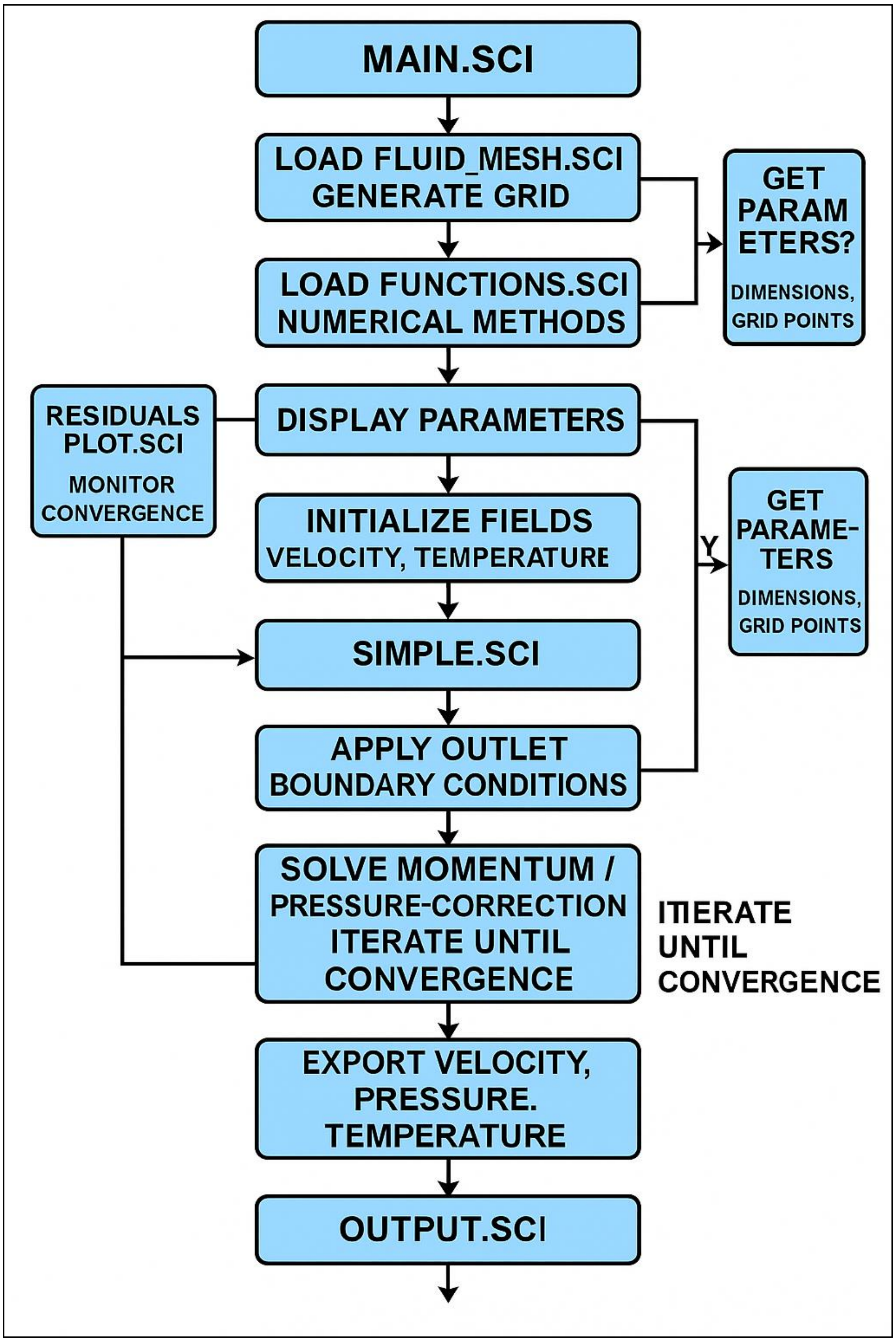

MAIN.SCI
LOAD FLUID_MESH.SCI GENERATE GRID
LOAD FUNCTIONS.SCI NUMERICAL METHODS
GET PARAM ETERS?
DIMENSIONS, GRID POINTS
RESIDUALS PLOT.SCI
MONITOR CONVERGENCE
DISPLAY PARAMETERS
INITIALIZE FIELDS VELOCITY, TEMPERATURE
Y
GET PARAME-TERS
DIMENSIONS, GRID POINTS
SIMPLE.SCI
APPLY OUTLET BOUNDARY CONDITIONS
SOLVE MOMENTUM / PRESSURE-CORRECTION ITERATE UNTIL CONVERGENCE
ITIERATE UNTIL CONVERGENCE
EXPORT VELOCITY, PRESSURE. TEMPERATURE
OUTPUT.SCI

Fig 5c : Flowchart

The main.sci script executes a sequence of Scilab files necessary for running the CFD simulation. It first loads fluid_mesh.sci, which generates the computational grid, ensuring a proper domain setup for the collocated scheme. Next, functions.sci is executed, containing numerical methods and helper functions used throughout the simulation. The residuals_plot.sci script is then called to monitor convergence by plotting residuals for velocity and pressure. The core solver, simple.sci, is executed next, implementing the SIMPLE algorithm to iteratively solve the Navier-Stokes equations while ensuring mass conservation. Finally, output.sci is run to process and visualize the simulation results, including velocity, pressure, and temperature distributions.

a) User Input for Default or Custom Values

This section of the code allows the user to either use default values or input their own custom parameters for a simulation. The user is first prompted to choose whether they want to use default settings or provide their own values. If they choose custom inputs, they are asked to enter values for various parameters related to the channel's dimensions and grid points. After selecting or entering values, the program displays the chosen parameters for confirmation. This setup ensures flexibility while giving the user control over the simulation's configuration.

b) Global Variables Used in Function

- ***Fw, Fe, Fn***, and ***Fs*** represent the convective mass flux at the west, east, north, and south faces of a computational cell, crucial for mass flow calculations across grid boundaries.
- ***DF*** is the diffusion coefficient that accounts for viscosity effects, modeling the diffusion of momentum or heat within the fluid.
- ***aW, aE, aS, aN***, and ***aP*** are coefficients for the west, east, south, north, and central terms in the finite volume method (FVM), used to discretize the governing equations.
- ***bP*** is the source term in the discretized equation, accounting for sources or sinks of the modeled quantity.
- ***dU*** and ***dV*** are velocity correction factors in the SIMPLE method, used for pressure-velocity coupling to ensure mass conservation.

c) Function Definitions

- **FVM_GS_ext_mesh** - This section defines the Gauss-Seidel solver for solving a finite volume method (FVM) problem on an external mesh, using the iterative approach.
- **FVM_pcorr** - This section sets up the pressure-correction equation coefficients for the finite volume method, using velocity components and adjusting for boundary conditions.
- **FVM_phi** - This section assembles the temperature (or scalar field) equation coefficients, accounting for advection, diffusion, and boundary conditions using hybrid upwinding.
- **FVM_u** - This function computes the x-momentum (u-velocity) equation coefficients, including convection fluxes, face values, and boundary conditions, and adjusts the pressure-velocity coupling.
- **FVM_v** - This function computes the y-momentum (v-velocity) equation coefficients, calculating convection fluxes, face values, and boundary conditions, and adjusts for pressure-velocity coupling.

d) Main Program - This section defines the geometry and grid for the simulation by calculating grid spacings in both x and y directions, creating grid locations for cell centers and velocity nodes, and defining the index ranges for the interior nodes used in the momentum and pressure equations.

e) **Fluid Properties and Other Parameters -** This section sets fluid properties, flow parameters, boundary conditions, and relaxation factors, either by default or user input, for the simulation.

f) **Iniialize Fields** - This section initializes the velocity, temperature, and pressure fields, along with other necessary variables for the simulation. It sets the inlet velocity, wall temperature, and the initial pressure distribution across the grid. Additionally, it defines the coefficients for momentum equations related to fluid viscosity. The temperature field is initialized at the inlet and the walls, and the pressure field is set with a linear drop to simulate fully developed flow.

g) **SIMPLE Algorithm Iterations -** This section implements the SIMPLE algorithm for solving momentum and pressure correction equations iteratively. It optionally includes solving the temperature equation within the loop, based on user input, updating the temperature field using thermal properties and heat flux. The loop continues until convergence is achieved, with progress displayed in each iteration.

   Apply Outlet Boundary Conditions: Ensure consistency at the boundaries (du/dx = 0, dv/dx = 0).
   Initialization : Store initial velocity and pressure values.
   Step 1a: Solve x-momentum equation for uStar (predicted u-velocity).
   Step 1b: Solve y-momentum equation for vStar (predicted v-velocity).
   Step 2: Solve pressure correction equation for pPrime.
   Step 3: Update pressure and velocity fields with pressure correction.
   Step 4: Check convergence; if residuals are small, stop iteration. If divergence occurs, stop iteration.

h) **Output Section -** This section handles the final output of the simulation by exporting the results to CSV and .txt files. It defines the path where the files will be stored and ensures the directory exists by creating it if necessary. The results, including residuals, pressure, velocity components (u and v), and temperature, are saved into separate CSV files. Once the export process is complete, a message confirming the successful export is displayed.

Chapter 5 : Software/Hardware Used

This project was developed and executed using the following software and hardware resources:

Software:

- **Operating System:** Windows 11 (64-bit)
- **Scilab Version:** Scilab 2025 (Open-source numerical computation platform)
- **Toolboxes/Modules:** No external toolboxes were used; the solver was built using native Scilab functions and scripts.

Scripts and Files Used:

- main.sci – master script to execute the full simulation sequence.
- fluid_mesh.sci – script for grid generation and domain discretization.
- functions.sci – includes helper functions and numerical schemes.
- simple.sci – core solver implementing the SIMPLE algorithm.
- residuals_plot.sci – script for monitoring convergence behavior.
- output.sci – script for exporting simulation results to .csv format.

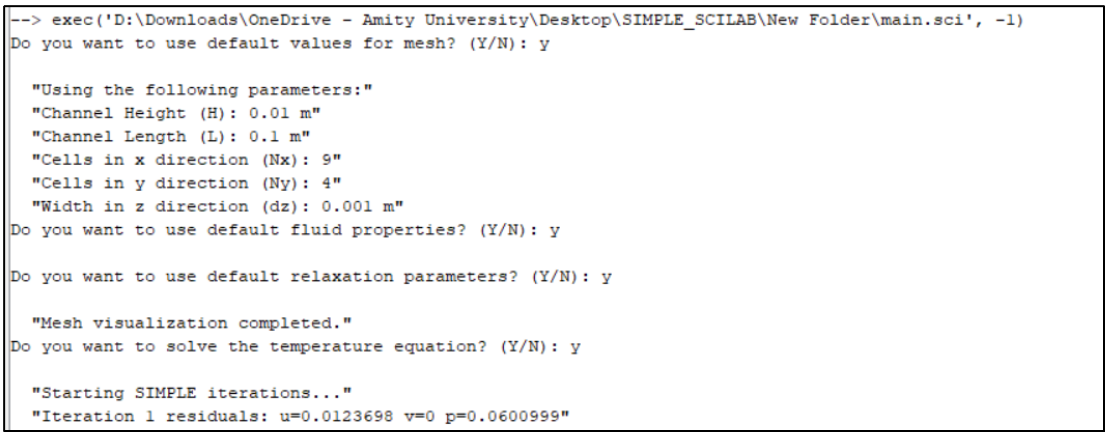
Hardware:

- **Processor:** Intel® Core™ i5-1135G7 @ 2.40GHz
- **RAM:** 16 GB DDR4
- **Storage:** 512 GB SSD
- **Display:** 15.6”, 1920x1080 resolution

No additional hardware such as GPUs or sensors was required for this project, as the computation was entirely numerical and performed on a standard personal laptop. The solver is lightweight and capable of running efficiently on low to mid-range machines, making it suitable for educational and research use in resource-constrained environments.

Chapter 6 : Procedure of Execution

The **main.sci** is executed and step-by-step explanation is highlighted below.



```
--> exec('D:\Downloads\OneDrive - Amity University\Desktop\SIMPLE_SCILAB\New Folder\main.sci', -1)
Do you want to use default values for mesh? (Y/N): y

  "Using the following parameters:"
  "Channel Height (H): 0.01 m"
  "Channel Length (L): 0.1 m"
  "Cells in x direction (Nx): 9"
  "Cells in y direction (Ny): 4"
  "Width in z direction (dz): 0.001 m"
Do you want to use default fluid properties? (Y/N): y

Do you want to use default relaxation parameters? (Y/N): y

  "Mesh visualization completed."
Do you want to solve the temperature equation? (Y/N): y

  "Starting SIMPLE iterations..."
  "Iteration 1 residuals: u=0.0123698 v=0 p=0.0600999"
```

Fig 6 : Output of Terminal / Console of mesh.sci

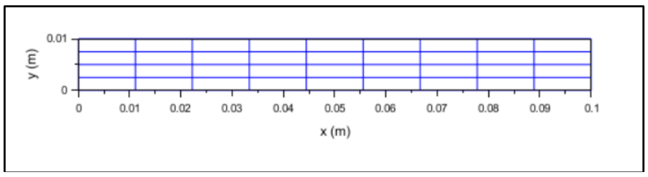
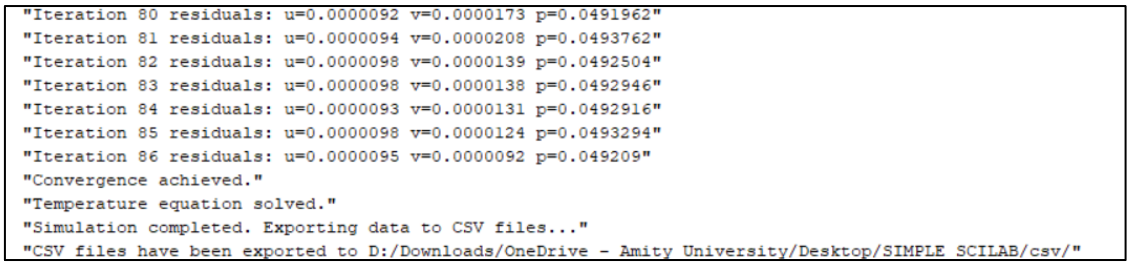


Fig 7 : Computational Mesh (default values)



```
"Iteration 80 residuals: u=0.0000092 v=0.0000173 p=0.0491962"
"Iteration 81 residuals: u=0.0000094 v=0.0000208 p=0.0493762"
"Iteration 82 residuals: u=0.0000098 v=0.0000139 p=0.0492504"
"Iteration 83 residuals: u=0.0000098 v=0.0000138 p=0.0492946"
"Iteration 84 residuals: u=0.0000093 v=0.0000131 p=0.0492916"
"Iteration 85 residuals: u=0.0000098 v=0.0000124 p=0.0493294"
"Iteration 86 residuals: u=0.0000095 v=0.0000092 p=0.049209"
"Convergence achieved."
"Temperature equation solved."
"Simulation completed. Exporting data to CSV files..."
"CSV files have been exported to D:/Downloads/OneDrive - Amity University/Desktop/SIMPLE_SCILAB/csv/"
```

Fig 8 : Final Output of Terminal / Console

Fig. 8 shows output achieved when default values are obtained. It also shows .csv files are successfully saved in the mentioned. The simulation is converged in 83 iterations only. Number of iterations vary for different input / user values. There is chance of simulation getting divergent. Divergence may occur due to factors like poor mesh resolution, incorrect boundary conditions, numerical instability, or inappropriate solver settings. Proper parameter selection and monitoring are key to achieving convergence and reliable results. Once the simulation converges, results are saved in .csv files for further analysis.

Chapter 7 : Results & Discussion

The solution process follows the SIMPLE algorithm. It starts by solving the u-momentum equation using hybrid differencing scheme where velocities are interpolated at faces (Fw, Fe, Fs, Fn) and pressure gradient acts as a source term. The equation is solved using Gauss-Seidel iteration to get uStar. Similarly, v-momentum is solved with interpolated velocities and pressure gradient as source term to get vStar. Then, the pressure correction equation is constructed using mass imbalance as the source term, with coefficients based on velocity corrections dU and dV, and zero pressure enforced at the outlet boundary. This produces pPrime which is used directly with uStar, vStar to get final corrected velocities. For temperature, it's solved after the pressure-velocity coupling, using hybrid differencing with convection terms interpolated at faces and appropriate wall boundary conditions. The entire process iterates until convergence is achieved based on the residuals of u, v and p equations.

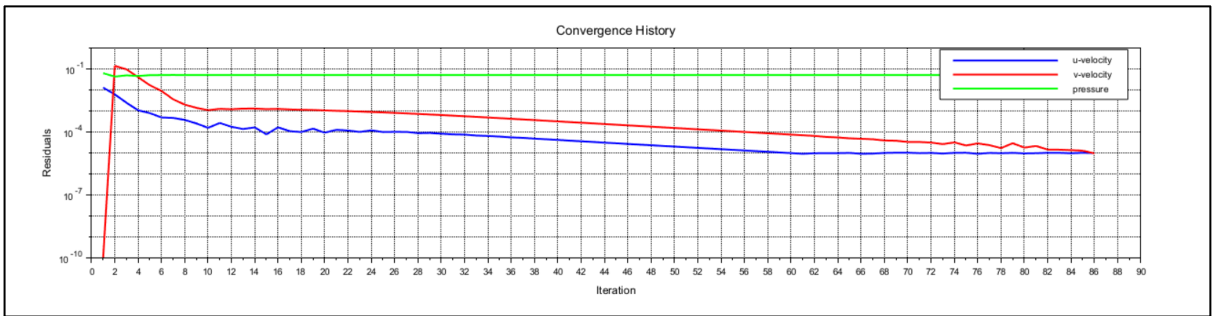


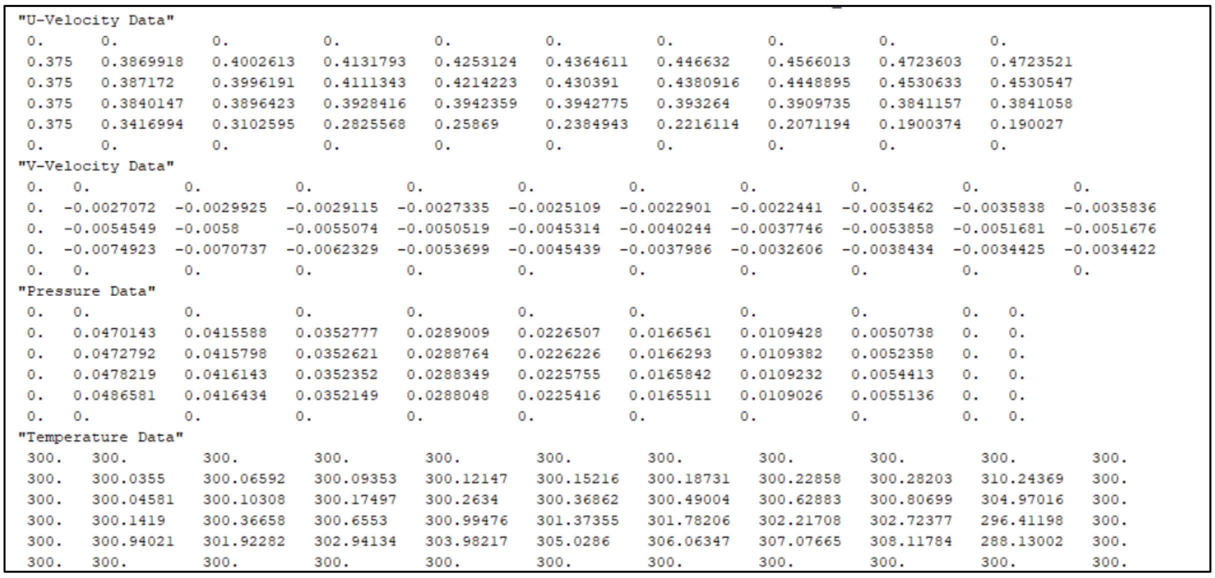
Fig 10 : Convergence Plot

From Fig. 10, the convergence history plot represents the evolution of residuals over multiple iterations for U-velocity (blue), V-velocity (red), and pressure (green). Initially, the velocity residuals are high, but they drop sharply within the first few iterations, indicating rapid error reduction in the early stages of the solution process. As the iterations progress, the rate of decrease slows down, and the residuals exhibit a gradual decline, suggesting that the solution is approaching a stable state. The pressure residual, represented by the green line, remains nearly constant throughout the iterations, indicating that pressure corrections are not changing significantly. The overall trend of the velocity residuals suggests that the solver is converging toward a steady-state solution.

We are transposing the data to align it correctly with the computational grid, ensuring that each value corresponds to the appropriate cell location. This helps in accurate visualization and interpretation of flow variables within the 4×9 mesh.

The given **U-Velocity Data** in Fig. 11 represents the velocity distribution in an axisymmetric pipe flow, where only half of the domain (below the centerline) is solved due to symmetry. The velocity profile varies in the radial direction, with values decreasing as they approach the bottom boundary, indicating the no-slip condition at the pipe wall. The highest velocity occurs near the centerline, which aligns with the expected parabolic velocity distribution in fully developed laminar or transitional flow regimes. The transposed data ensures correct alignment with computational cells, preserving the spatial representation of velocity variations. Since the number of computational cells is very low, the velocity gradients are not well-resolved, leading to higher velocity values compared to a finer mesh. The figure presents numerical simulation data for a pipe flow problem, showing computed values of U-Velocity, V-Velocity, Pressure, and Temperature across a structured computational grid. Each dataset is displayed in a tabular format, representing flow field variations over the mesh.

The first and last rows in each dataset contain zero values, which are redundant in the context of mapping to computational cells. These zeros represent boundary conditions or extrapolated values outside the main region of interest. Since only the internal computational cells are significant for final iterations, these redundant rows should be ignored to ensure accurate representation of flow properties.



```
"U-Velocity Data"
 0.      0.          0.          0.          0.          0.          0.          0.          0.          0.
 0.375   0.3869918   0.4002613   0.4131793   0.4253124   0.4364611   0.446632    0.4566013   0.4723603   0.4723521
 0.375   0.387172    0.3996191   0.4111343   0.4214223   0.430391    0.4380916   0.4448895   0.4530633   0.4530547
 0.375   0.3840147   0.3896423   0.3928416   0.3942359   0.3942775   0.393264    0.3909735   0.3841157   0.3841058
 0.375   0.3416994   0.3102595   0.2825568   0.25869     0.2384943   0.2216114   0.2071194   0.1900374   0.190027
 0.      0.          0.          0.          0.          0.          0.          0.          0.          0.
"V-Velocity Data"
 0.   0.          0.          0.          0.          0.          0.          0.          0.          0.          0.
 0.  -0.0027072  -0.0029925  -0.0029115  -0.0027335  -0.0025109  -0.0022901  -0.0022441  -0.0035462  -0.0035838  -0.0035836
 0.  -0.0054549  -0.0058     -0.0055074  -0.0050519  -0.0045314  -0.0040244  -0.0037746  -0.0053858  -0.0051681  -0.0051676
 0.  -0.0074923  -0.0070737  -0.0062329  -0.0053699  -0.0045439  -0.0037986  -0.0032606  -0.0038434  -0.0034425  -0.0034422
 0.   0.          0.          0.          0.          0.          0.          0.          0.          0.          0.
"Pressure Data"
 0.   0.          0.          0.          0.          0.          0.          0.          0.          0.   0.
 0.   0.0470143   0.0415588   0.0352777   0.0289009   0.0226507   0.0166561   0.0109428   0.0050738   0.   0.
 0.   0.0472792   0.0415798   0.0352621   0.0288764   0.0226226   0.0166293   0.0109382   0.0052358   0.   0.
 0.   0.0478219   0.0416143   0.0352352   0.0288349   0.0225755   0.0165842   0.0109232   0.0054413   0.   0.
 0.   0.0486581   0.0416434   0.0352149   0.0288048   0.0225416   0.0165511   0.0109026   0.0055136   0.   0.
 0.   0.          0.          0.          0.          0.          0.          0.          0.          0.   0.
"Temperature Data"
 300.   300.        300.        300.        300.        300.        300.        300.        300.        300.        300.
 300.   300.0355    300.06592   300.09353   300.12147   300.15216   300.18731   300.22858   300.28203   310.24369   300.
 300.   300.04581   300.10308   300.17497   300.2634    300.36862   300.49004   300.62883   300.80699   304.97016   300.
 300.   300.1419    300.36658   300.6553    300.99476   301.37355   301.78206   302.21708   302.72377   296.41198   300.
 300.   300.94021   301.92282   302.94134   303.98217   305.0286    306.06347   307.07665   308.11784   288.13002   300.
 300.   300.        300.        300.        300.        300.        300.        300.        300.        300.        300.
```

Fig 11 : Final Data

The **V-Velocity Data** represents the vertical velocity component in the axisymmetric pipe flow, showing small negative values that indicate the influence of radial momentum exchange. Since the primary flow direction is axial, the magnitude of v-velocity remains much lower than u-velocity, with values approaching zero at the boundaries due to the imposed no-slip condition. These small variations help in capturing the effect of pressure-driven flow and secondary motion. The transposed data ensures that each value corresponds accurately to its respective computational cell, maintaining spatial accuracy in velocity mapping.

The **Pressure Data** illustrates the pressure field within the computational domain, showing a gradual decrease due to frictional effects and energy dissipation along the pipe length. Higher values near the inlet and lower values downstream indicate the expected pressure gradient that drives the flow. The structured variation ensures continuity and convergence in the numerical scheme. Since the number of computational cells is limited, the pressure gradient may not be well-refined, leading to a more abrupt transition. The transposed format ensures that pressure values are correctly mapped to their corresponding grid locations for accurate pressure field representation.

The **Temperature Data** depicts the thermal distribution within the flow, reflecting the combined effects of conduction and convection. Temperature values increase as energy is transferred from the heated pipe walls to the fluid, leading to higher temperatures near the boundaries and a more uniform distribution in the core region due to convective mixing. The presence of a structured mesh influences the resolution of thermal gradients, meaning that a finer mesh would capture more detailed temperature variations. The transposed data allows for proper alignment with the computational cells, ensuring that thermal gradients and variations are mapped correctly within the final solution.

This study successfully developed and implemented a 2D Computational Fluid Dynamics (CFD) solver in Scilab for incompressible, laminar flow using the Finite Volume Method (FVM) and the SIMPLE algorithm. The solver was designed to handle structured Cartesian grids and employed a collocated scheme for variable storage. The governing equations—Navier-Stokes and continuity—were discretized using a central difference scheme for diffusion terms and an upwind scheme for convection terms.

The solver's performance was validated by monitoring residuals of velocity and pressure, ensuring convergence and numerical stability. Simulation results showed smooth convergence in most cases, with divergence occurring under inappropriate parameter selection, highlighting the accuracy of the implemented numerical scheme. The solver successfully exported key flow field variables—velocity, pressure, and residuals—in CSV format for further analysis.

Visualization of results, including convergence plots and velocity distributions, confirmed the expected flow behavior in a rectangular channel. The velocity profile demonstrated the expected parabolic shape, while pressure corrections aligned with theoretical predictions. The study demonstrates the feasibility of using Scilab as a cost-effective and open-source alternative for CFD simulations.

Future work can extend the solver's capabilities to include turbulence modeling, transient flows, and complex geometries. Additionally, improving grid resolution and implementing adaptive meshing techniques could enhance accuracy and computational efficiency. This implementation serves as a foundation for further research and educational applications in fluid dynamics.

The future work under same domain can be done by :

a) Implementation of PISO, PIMPLE, SIMPLEC algorithms
b) Running Unstructured Mesh / Grid
c) GUI Based Visualization of Results
d) Modelling Turbulence

Chapter 8 : References

1) Khawaja, H., & Moatamedi, M. (2018). Semi-Implicit Method for Pressure-Linked Equations (SIMPLE) – solution in MATLAB®. The International Journal of Multiphysics, 12(4), 313-326. https://doi.org/10.21152/1750-9548.12.4.313

2) Patankar, S.V. and Spalding, D.B. (1972) A Calculation Procedure for Heat, Mass and Momentum Transfer in Three-Dimensional Parabolic Flows. International Journal of Heat and Mass Transfer, 15, 1787-1806. http://dx.doi.org/10.1016/0017-9310(72)90054-3

3) Lamsal, Abish. (2023). Analyzing Pipe Flow Scenarios using Computational Fluid Dynamics (CFD).