# Computational Fluid Dynamics Course Project

Bao-Yuan Zhao 522010910016

May 25, 2025

## 1 Fundamental theory

### 1.1 Question 1

**The velocity vector of a steady incompressible flow field is given by:**

$$\boldsymbol{v} = (x+y)\boldsymbol{i} + (y+z)\boldsymbol{j} + 2(x-z)\boldsymbol{k}$$

**(a) Determine whether this flow field satisfies the continuity equation.**
For incompressible flow, the continuity equation is: $\nabla \cdot \boldsymbol{v} = 0$

$$\nabla \cdot \boldsymbol{v} = \frac{\partial}{\partial x}(x+y) + \frac{\partial}{\partial y}(y+z) + \frac{\partial}{\partial z}(2(x-z)) = 1 + 1 - 2 = 0$$

Since $\nabla \cdot \boldsymbol{v} = 0$, the flow field satisfies the continuity equation.
**(b) Assuming the viscosity coefficient $\mu$ is constant, calculate viscous stress tensor $\boldsymbol{\tau}$.**
The viscous stress tensor is defined as:

$$\boldsymbol{\tau} = \mu\left(\nabla \boldsymbol{v} + (\nabla \boldsymbol{v})^T\right)$$

Compute$\nabla \boldsymbol{v}$ :

$$\frac{\partial v_x}{\partial x} = \frac{\partial (x+y)}{\partial x} = 1, \quad \frac{\partial v_x}{\partial y} = \frac{\partial (x+y)}{\partial y} = 1, \quad \frac{\partial v_x}{\partial z} = \frac{\partial (x+y)}{\partial z} = 0$$
$$\frac{\partial v_y}{\partial x} = \frac{\partial (y+z)}{\partial x} = 0, \quad \frac{\partial v_y}{\partial y} = \frac{\partial (y+z)}{\partial y} = 1, \quad \frac{\partial v_y}{\partial z} = \frac{\partial (y+z)}{\partial z} = 1$$
$$\frac{\partial v_z}{\partial x} = \frac{\partial (2(x-z))}{\partial x} = 2, \quad \frac{\partial v_z}{\partial y} = \frac{\partial (2(x-z))}{\partial y} = 0, \quad \frac{\partial v_z}{\partial z} = \frac{\partial (2(x-z))}{\partial z} = -2$$
$$\boldsymbol{\tau} = \mu \begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & -4 \end{bmatrix}$$

**(c) The fluid density is denoted by $\rho$ and body forces are neglected. Please derive the pressure-gradient equation.**

To derive the pressure gradient equation for the given steady, incompressible flow with negligible body forces, the Navier-Stokes equationsequation simplifies to:

$$\nabla p = -\rho(\mathbf{v} \cdot \nabla \mathbf{v}) + \mu \nabla^2 \mathbf{v}$$

Step 1: Compute the convective term $(\mathbf{v} \cdot \nabla \mathbf{v})$

$$(\mathbf{v} \cdot \nabla v_x) = v_x \frac{\partial v_x}{\partial x} + v_y \frac{\partial v_x}{\partial y} + v_z \frac{\partial v_x}{\partial z} = (x+y)(1) + (y+z)(1) + 2(x-z)(0) = x + 2y + z$$

$$(\mathbf{v} \cdot \nabla v_y) = v_x \frac{\partial v_y}{\partial x} + v_y \frac{\partial v_y}{\partial y} + v_z \frac{\partial v_y}{\partial z} = (x+y)(0) + (y+z)(1) + 2(x-z)(1) = 2x + y - z$$

$$(\mathbf{v} \cdot \nabla v_z) = v_x \frac{\partial v_z}{\partial x} + v_y \frac{\partial v_z}{\partial y} + v_z \frac{\partial v_z}{\partial z} = (x+y)(2) + (y+z)(0) + 2(x-z)(-2) = -2x + 2y + 4z$$

Thus:

$$\mathbf{v} \cdot \nabla \mathbf{v} = (x + 2y + z)\mathbf{i} + (2x + y - z)\mathbf{j} + (-2x + 2y + 4z)\mathbf{k}$$

Step 2: Compute the viscous term $\mu \nabla^2 \mathbf{v}$
Since all velocity components are linear in $x, y, z$, their second derivatives vanish.

Step 3: Pressure gradient equation substitute into the simplified Navier-Stokes equation:

$$\nabla p = -\rho(\mathbf{v} \cdot \nabla \mathbf{v})$$

Final Answer:

$$\boxed{\begin{aligned} \frac{\partial p}{\partial x} &= -\rho(x + 2y + z), \\ \frac{\partial p}{\partial y} &= -\rho(2x + y - z), \\ \frac{\partial p}{\partial z} &= \rho(2x - 2y - 4z). \end{aligned}}$$

## 1.2 Question 2

The specific problem considered is the 1D convection equation with the following parameters:

- Convection velocity: $u = 1$
- Spatial grid spacing: $\Delta x = 1$
- Number of spatial grid points: $N_x = 100$
- Time step size: $\Delta t = 0.1$
- Number of time steps: $N_t = 1000$

### 1.2.1 Numerical Methods

The 1D convection equation is discretized using an explicit time-stepping approach. For a general finite difference scheme, the equation can be written as:

$$\frac{f_i^{n+1} - f_i^n}{\Delta t} + u \left( \frac{\partial f}{\partial x} \right)_i^n = 0$$

where $f_i^n$ denotes the value of $f$ at spatial point $i$ and time step $n$. The spatial derivative $\left( \frac{\partial f}{\partial x} \right)_i^n$ is approximated differently by each scheme.

**Central Difference Scheme (CDS)**
The central difference approximation for the spatial derivative is:

$$\left( \frac{\partial f}{\partial x} \right)_i^n \approx \frac{f_{i+1}^n - f_{i-1}^n}{2\Delta x}$$

Substituting this into the discretized equation gives the explicit CDS:

$$f_i^{n+1} = f_i^n - \frac{u\Delta t}{2\Delta x}(f_{i+1}^n - f_{i-1}^n)$$

This scheme is second-order accurate in space. However, for pure advection, it is known to be unconditionally unstable and typically produces significant oscillations (dispersion) when applied to hyperbolic problems.

**First-Order Upwind (FOU) Scheme**
The upwind scheme uses information from the direction of flow. Since $u = 1 > 0$, the flow is from left to right, so we use a backward difference for the spatial derivative:

$$\left( \frac{\partial f}{\partial x} \right)_i^n \approx \frac{f_i^n - f_{i-1}^n}{\Delta x}$$

The explicit FOU scheme is:

$$f_i^{n+1} = f_i^n - \frac{u\Delta t}{\Delta x}(f_i^n - f_{i-1}^n)$$

This scheme is first-order accurate in space. It is conditionally stable (requires $u\Delta t/\Delta x \leq 1$) but introduces significant numerical diffusion, which smears out sharp gradients.

**QUICK Scheme**

QUICK (Quadratic Upstream Interpolation for Convective Kinematics) is a second-order accurate upwind scheme. For $u > 0$, the scheme first interpolates the value at the upstream face of the control volume (e.g., at $x_{i-1/2}$) using a quadratic polynomial involving points $f_{i-2}^n$, $f_{i-1}^n$, and $f_i^n$. The interpolated face value, $f_{face}$, is given by:

$$f_{wface} = \frac{6}{8}f_{i-1}^n + \frac{3}{8}f_i^n - \frac{1}{8}f_{i-2}^n$$

$$f_{eface} = \frac{6}{8}f_i^n + \frac{3}{8}f_{i+1}^n - \frac{1}{8}f_{i-1}^n$$

Then, the update equation for $f_i^{n+1}$ is derived from a flux balance across the control volume, often expressed in a form that incorporates this interpolated face value:

$$f_i^{n+1} = f_i^n - \frac{u\Delta t}{\Delta x}(f_{eface}^n - f_{wface}^n)$$

This scheme is designed to be more accurate than FOU while minimizing numerical diffusion, but it can still exhibit oscillations near discontinuities, though generally less severe than CDS. It requires a stencil of four points, which needs careful handling of periodic boundary conditions.

### 1.2.2 Results and Discussion

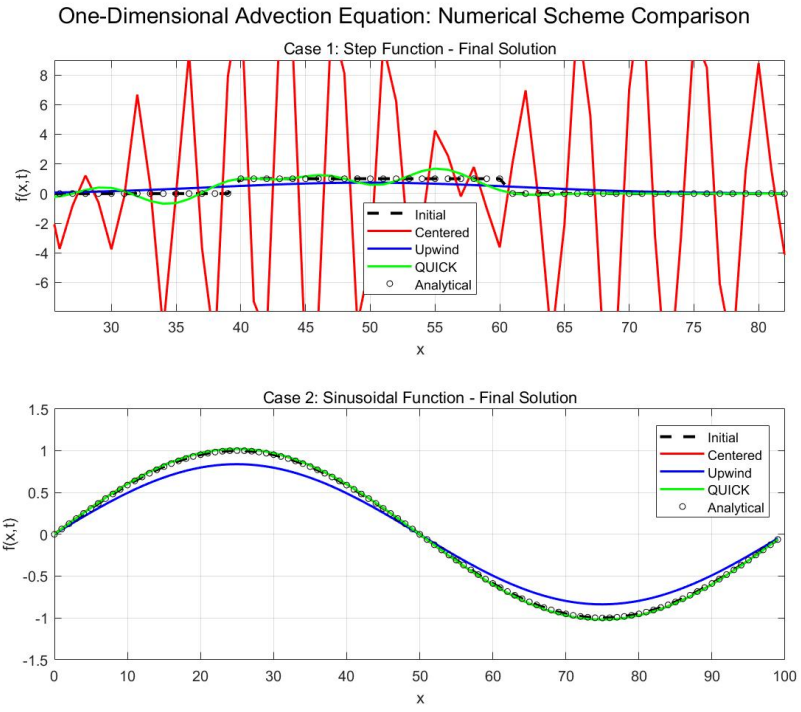


Figure 1: Comparison of numerical solutions at final time.

Table 1 shows the $L_2$ errors for different schemes and grid resolutions.

| Case | CDS $L_2$ Error | FOU $L_2$ Error | QUICK $L_2$ Error |
|---|---|---|---|
| 1 | 6.425346(Unstable) | 0.221924 | 0.233686 |
| 2 | 0.014396 | 0.115094 | 0.014044 |

Table 1: $L_2$ Errors for Two different cases with Initial Condition at $t = 100$.

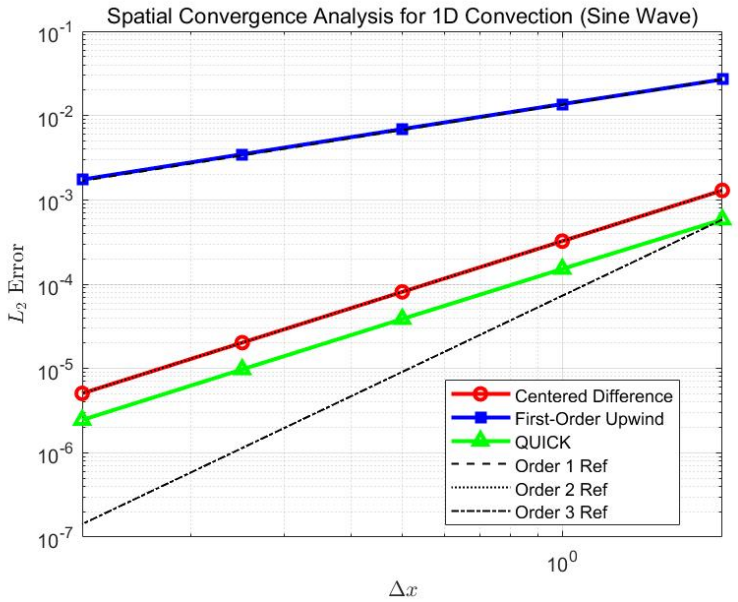


Figure 2: Observed convergence rates for Sine Wave Initial Condition

**Initial Condition 1: Square Wave**

The square wave is a challenging test case due to its sharp discontinuities.

- **Central Difference Scheme (CDS):** As expected, CDS exhibits significant oscillatory behavior (dispersion) around the discontinuities. It struggles to maintain the sharp profile, and the oscillations grow over time, indicating its instability for this type of problem.
- **First-Order Upwind (FOU) Scheme:** FOU introduces considerable numerical diffusion. The sharp edges of the square wave are significantly smeared out, and the wave loses its crispness as it propagates. While stable, its accuracy for discontinuous profiles is limited.
- **QUICK Scheme:** QUICK performs better than FOU in terms of reducing numerical diffusion, maintaining a sharper profile. However, it still produces some oscillations (overshoots and undershoots) near the discontinuities, a common characteristic of higher-order schemes when applied to non-smooth solutions.

**Initial Condition 2: Sine Wave**

- **Central Difference Scheme (CDS):** For the sine wave, CDS has small error for continuous function as it is a second-order method.
- **First-Order Upwind (FOU) Scheme:** FOU accurately propagates the sine wave, but with noticeable amplitude damping due to numerical diffusion. The $L_2$ error decreases as $N_x$ increases, and the observed convergence rate should be close to 1, consistent with its first-order accuracy.
- **QUICK Scheme:** QUICK provides a significantly more accurate solution for the sine wave compared to FOU. It exhibits less amplitude damping and phase error. The $L_2$ error is considerably lower than FOU.It offers a good balance between accuracy and stability for advection problems

## 1.3 Question 3

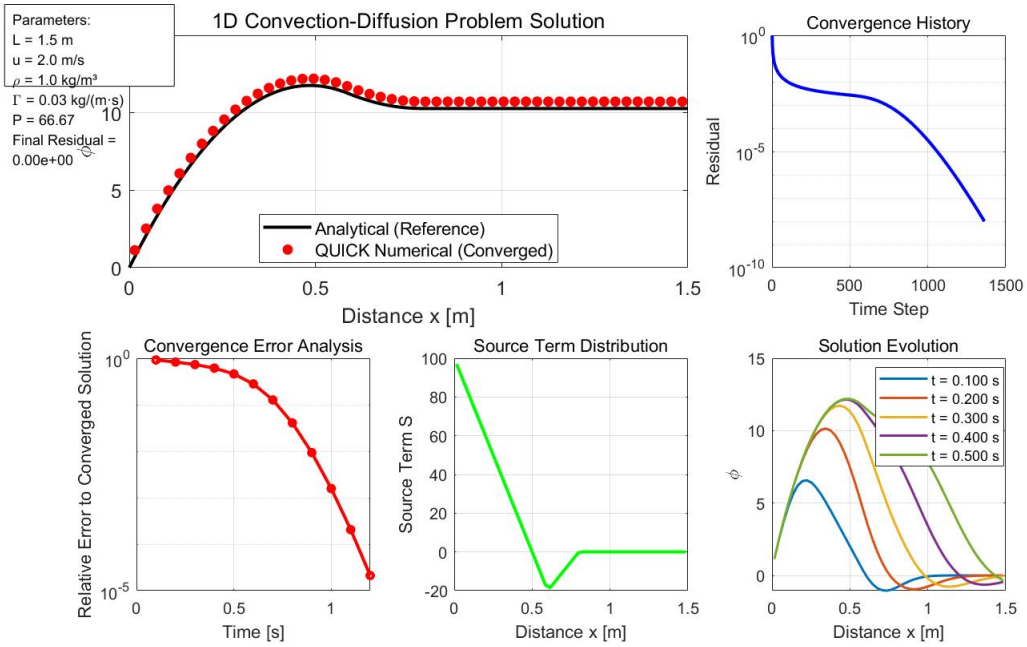


Figure 3: Question 3 result

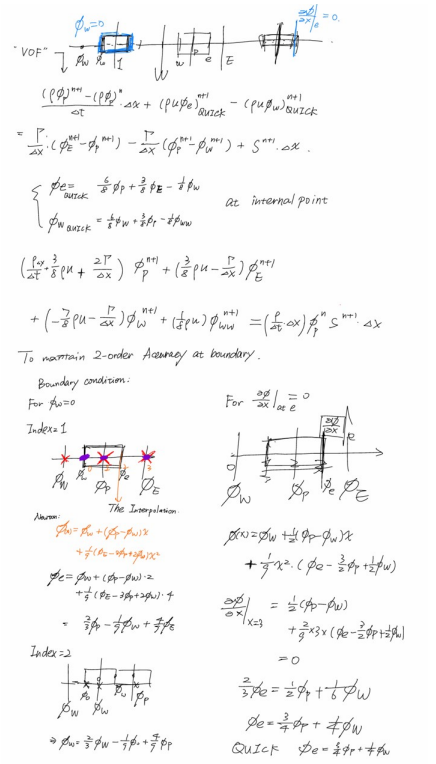


Figure 4: Detailed QUICK Algorithm and Boundary condition

## 1.4 Question 4

Consider a two-dimensional steady flow as shown in 7. Given: $u_w = 50$, $v_s = 20$, $p_N = 0$, $p_E = 10$. The governing discrete momentum equations for the east and north faces are:

$$u_e = p_p - p_E \tag{1}$$
$$v_n = 0.7(p_p - p_N) \tag{2}$$

where $p_p$ is the center cell pressure, $u_e$ is the east face velocity, and $v_n$ is the north face velocity. The objective is to solve for $u_e$, $v_n$, and $p_p$ using the SIMPLE algorithm.

The SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) algorithm follows these steps:

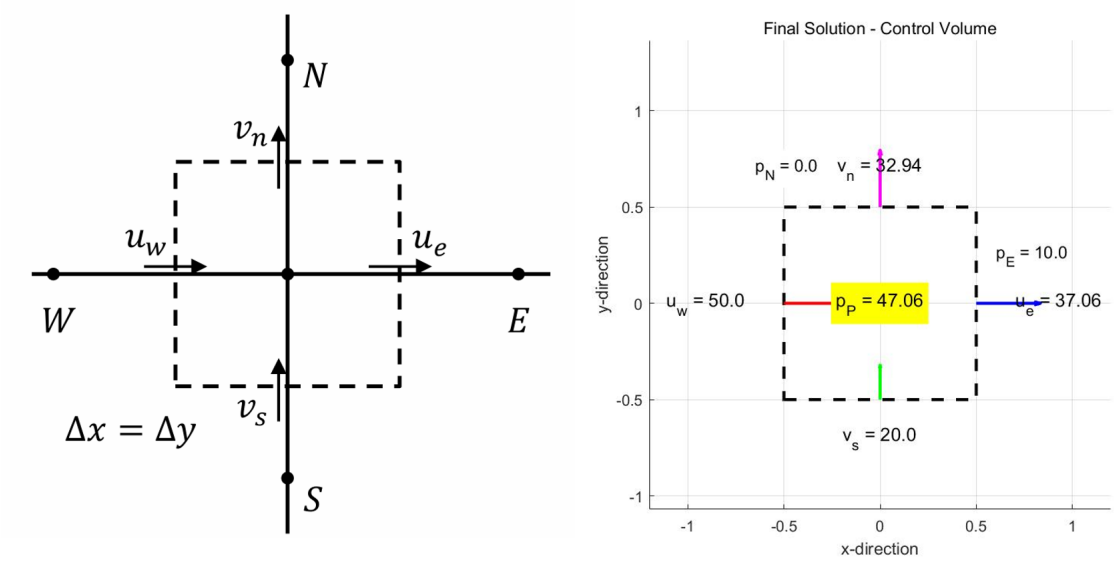


Figure 5: Question 4 and result

**Step 1: Initial Pressure Guess** Start with an initial guess for the center pressure:

$$p_p^* = 0 \quad \text{(initial guess)} \tag{3}$$

**Step 2: Calculate Velocities from Momentum Equations** Using the guessed pressure $p_p^*$, calculate the velocities:

$$u_e^* = p_p^* - p_E = -10 \tag{4}$$
$$v_n^* = 0.7(p_p^* - p_N) = 0 \tag{5}$$

**Step 4: Pressure Correction Equation**
For the corrected velocities:

$$u_e = u_e^* + u_e' \tag{6}$$
$$v_n = v_n^* + v_n' \tag{7}$$

From the momentum equations, the velocity corrections are:

$$u_e' = p_p' - p_E' = p_p' \quad (\text{since } p_E \text{ is fixed, } p_E' = 0) \tag{8}$$
$$v_n' = 0.7(p_p' - p_N') = 0.7 \cdot p_p' \quad (\text{since } p_N \text{ is fixed, } p_N' = 0) \tag{9}$$

where $p_p'$ is the pressure correction.

Applying mass conservation to the corrected velocities:

$$(u_e^* + u_e') - u_w + (v_n^* + v_n') - v_s = 0 \tag{10}$$
$$(u_e^* + p_p') - u_w + (v_n^* + 0.7p_p') - v_s = 0 \tag{11}$$
$$1.7p_p' = u_w - u_e^* + v_s - v_n^* \tag{12}$$

Substituting the values:

$$1.7p_p' = 50 - (-10) + 20 - 0 \tag{13}$$
$$p_p' = \frac{80}{1.7} = 47.059 \tag{14}$$

**Step 5: Update Pressure**

$$p_p^{new} = p_p^* + \alpha \cdot p_p' \tag{15}$$
$$= 47.059 \tag{16}$$

**Step 6: Calculate Corrected Velocities**

$$u_e = u_e^* + u_e' = u_e^* + p_p' = -10 + 47.059 = 37.059 \tag{17}$$
$$v_n = v_n^* + v_n' = v_n^* + 0.7p_p' = 0 + 0.7 \times 47.059 = 32.941 \tag{18}$$

The corrected velocities satisfy mass conservation exactly.

The SIMPLE algorithm converges in exactly one iteration for this problem due to the momentum equations (1) and (2) are linear in pressure. This linearity means that the pressure correction exactly accounts for the velocity changes needed to satisfy continuity conservation.

## 1.5 Question 5

Given equation:

$$x_{\xi\xi} = P \tag{19}$$

Boundary conditions:

$$x(0) = 0, \quad \xi(0) = 0 \tag{20}$$
$$x(2) = 2, \quad \xi(1) = 1 \tag{21}$$

When $P$ is constant, we can solve this by integrating twice:

$$x_{\xi\xi} = P \tag{22}$$
$$x_\xi = P\xi + C_1 \tag{23}$$
$$x = \frac{P}{2}\xi^2 + C_1\xi + C_0 \tag{24}$$

where $C_0$ and $C_1$ are integration constants that can be determined from the boundary conditions.

$$x(\xi) = \frac{P}{2}\xi^2 + \left(2 - \frac{P}{2}\right)\xi \tag{25}$$

When $P = 4$, the analytical solution becomes: $x(\xi) = 2\xi^2$. When $P = -4$, the analytical solution becomes: $x(\xi) = -2\xi^2 + 4\xi$.

Figure 9 shows the $\xi$-$x$ relationship curves for both cases.

This distribution pattern means that when $P > 0$, the grid concentrates more nodes at the left end of the computational domain, which is suitable for problems with large gradient changes at the left end.

This distribution pattern means that when $P < 0$, the grid concentrates more nodes at the right end of the computational domain, which is suitable for problems with large gradient changes at the right end.

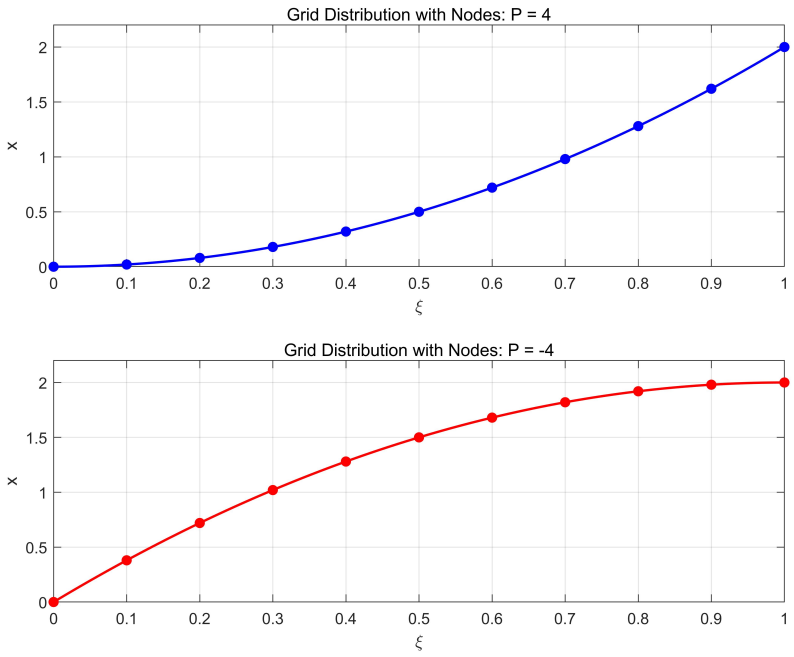


Figure 6: $\xi$-$x$ Relationship Curves for $P = 4$ and $P = -4$ Cases

# 2 SIMPLE Algorithm Analysis for 2D Lid-Driven Cavity Flow

## 2.1 Governing Equations

The two-dimensional incompressible Navier-Stokes equations in Cartesian coordinates are:
**Continuity equation:**

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{26}$$

**x-momentum equation:**

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{27}$$

**y-momentum equation:**

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{28}$$

For steady-state analysis, the time derivative terms are neglected.
The lid-driven cavity problem is defined with the following boundary conditions:

- **Top wall (moving lid):** $u = U_0$, $v = 0$
- **Bottom wall:** $u = 0$, $v = 0$
- **Left wall:** $u = 0$, $v = 0$
- **Right wall:** $u = 0$, $v = 0$

## 2.2 Definition of Reynolds number

The Reynolds number characterizing the flow is:

$$Re = \frac{\rho U_0 L}{\mu} = \frac{U_0 L}{\nu} \tag{29}$$

where $L$ is the characteristic length (cavity width) and $\nu$ is the kinematic viscosity. In our simulation, we fix the $L$ and $U_0$ as 1, and change the kinematic viscosity $\nu$ to change the Reynolds number.

## 2.3 Staggered Grid and Ghost Point Method

Here we employ the staggered-grid configuration. The stream-wise and wall-normal velocity components are stored on the cell faces and pressure quantities at the cell center. Explicit, second-order finite differences on a staggered grid are used to approximate (x, y)-derivatives. The convective terms are calculated using a fully conservative divergence form.

The staggered-grid cell configuration and its indexes are sketched below.

- Pressure: stored at cell centers $(i, j)$
- u: stored at east faces $(i, j + 1/2)$
- v: stored at north faces $(i + 1/2, j)$

In addition, ghost point method is added to extend the computational domain by adding fictitious cells (ghost cells) outside the physical boundaries. These ghost cells allow for easy implementation of various boundary condition types. **Moreover, the ghost point method helps to maintain second-order accuracy at the boundaries.**

```
%% Grid setup
dx = L / nx;      % Grid spacing in x direction
dy = D / ny;      % Grid spacing in y direction

% Initialize velocity arrays
u = zeros(ny + 2, nx + 1);    % x-velocity includes 2 ghost layer top/bottom
v = zeros(ny + 1, nx + 2);    % y-velocity includes 2 ghost layer left/right
p = zeros(ny, nx);            % pressure
```

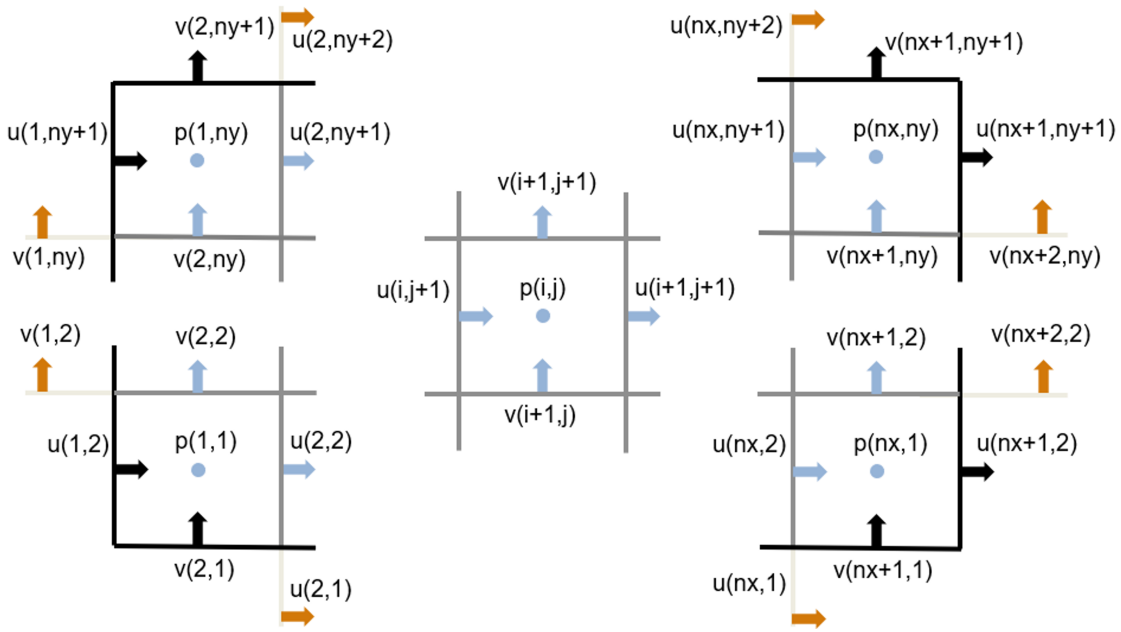


Figure 7: Stagger mesh and ghost point for grid setting

## 2.4 SIMPLE Algorithm Overview

The SIMPLE algorithm follows these steps:

The pseudo-code for the core outer iteration is as follows; we will now explain each step in sequence.

**Algorithm 1** SIMPLE Algorithm

1: Initialize velocity and pressure fields
2: **while** the continuity not converged **do**
3: Solve discretized momentum equations for $u^*$ and $v^*$
4: Solve pressure correction equation for $p'$ (Gauss-Seidel method)
5: Update pressure using under relaxation: $p^{n+1} = p^n + \alpha_p p'$
6: Correct velocities: $u^{n+1} = u^* + d_u \nabla p'$, $v^{n+1} = v^* + d_v \nabla p'$
7: Apply boundary conditions
8: **end while**

```
%%Initialization

while ~converged && iteration < maxIter
    iteration = iteration + 1;

    % Store old velocities for convergence check
    u_old = u;
    v_old = v;

    % Apply boundary conditions
    u = applyBoundaryConditions(u, v, U0, 'u');
    v = applyBoundaryConditions(u, v, U0, 'v');

    %% Step 1: Solve Momentum Equations
    [u_star, ae_u, aw_u, an_u, as_u, ap_u] = solveUMomentum(u, v, p, rho, mu, dx, dy, nx, ny);
    [v_star, ae_v, aw_v, an_v, as_v, ap_v] = solveVMomentum(u, v, p, rho, mu, dx, dy, nx, ny);

    %% Step 2: Calculate Mass Imbalance
    b = calculateMassImbalance(u_star, v_star, rho, dx, dy, nx, ny);

    % Track convergence metrics
    [avg_mass_imbalance, initial_continuity_residual] = ...
        trackConvergence(b, iteration, initial_continuity_residual, nx, ny);

    % Step 3: Solve pressure correction equation
    [ae_p, aw_p, an_p, as_p, ap_p] = setupPressureCorrectionCoeffs(ap_u, ap_v, rho, dx, dy, nx, ny);
    pc = solvePressureCorrection(ae_p, aw_p, an_p, as_p, ap_p, b, nx, ny);

    %% Step 4: Correct Pressure and Velocities
    p = correctPressure(p, pc, alpha_p, nx, ny);
    u = correctUVelocity(u, u_star, pc, ap_u, dy, nx, ny);
    v = correctVVelocity(v, v_star, pc, ap_v, dx, nx, ny);

    %% Check Convergence
    [res_u, res_v] = calculateMomentumResiduals(u, v, u_old, v_old);

    % Store residuals
    continuity_residual_hist(iteration) = avg_mass_imbalance;
    u_residual_hist(iteration) = res_u;
    v_residual_hist(iteration) = res_v;

    % Check for convergence
    if avg_mass_imbalance < continuity_tolerance
        converged = true;
        printConvergenceInfo(iteration, avg_mass_imbalance, res_u, res_v, true);
    end
end
%% Post-processing and Visualization
```

### 2.4.1 Boundary condition

The boundary condition for velocity and pressure is given as follow.

```
    % u-velocity boundary conditions
    % Bottom wall (y=0): No-slip, u=0. Ghost cell u(1,j) mirrors u(2,j)
```

```
u(1, :) = -u(2, :);
% Top wall (y=D): Moving lid, u=U0. Ghost cell u(ny+2,j) (above y=D)
u(ny + 2, :) = 2*U0 - u(ny + 1, :);
% Left wall (x=0): No-penetration, u=0. u(i,1) is on the boundary
u(:, 1) = 0;
% Right wall (x=L): No-penetration, u=0. u(i,nx+1) is on the boundary
u(:, nx + 1) = 0;

% v-velocity boundary conditions
% Bottom wall (y=0): No-penetration, v=0. v(1,j) is on the boundary
v(1, :) = 0;
% Top wall (y=D): No-penetration, v=0. v(ny+1,j) is on the boundary
v(ny + 1, :) = 0;
% Left wall (x=0): No-slip, v=0. Ghost cell v(i,1) mirrors v(i,2)
v(:, 1) = -v(:, 2);
% Right wall (x=L): No-slip, v=0. Ghost cell v(i,nx+2) mirrors v(i,nx+1)
v(:, nx + 2) = -v(:, nx + 1);
```

The coefficients for the pressure correction equation ($ae_p, aw_p, an_p, as_p, ap_p$) are modified for cells adjacent to the boundaries as well.

For instance, if a pressure cell is at the bottom boundary (i.e., j == nx), its $ae_p$ coefficient (representing the south neighbor) is set to zero because there's no south neighbor to influence the pressure correction within the internal domain.

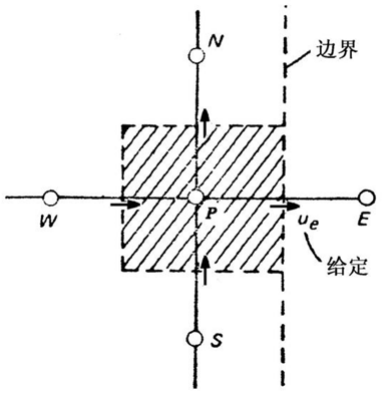


Figure 8: boundary condition

### 2.4.2 Finite Volume Discretization

The momentum equations are discretized using the finite volume method. For a u-momentum control volume:

• **Viscous terms**

For $u$ / $v$ the terms are defined as at the $u$ / $v$ -control cell center.

$$Du(i,j) = \frac{u(i+1,j) - 2u(i,j) + u(i-1,j)}{dx^2} + \frac{u(i,j+1) - 2u(i,j) + u(i,j-1)}{dy^2}$$

$$Dv(i,j) = \frac{v(i+1,j) - 2v(i,j) + v(i-1,j)}{dx^2} + \frac{v(i,j+1) - 2v(i,j) + v(i,j-1)}{dy^2}$$

• **Pressure gradient**

Similarly with above, 2nd order.

$$\partial_x p = \frac{p(i,j) - p(i-1,j)}{dx} \quad \partial_y p = \frac{p(i,j) - p(i,j-1)}{dy}$$

The pressure gradient needs to be defined at $u$ / $v$ inside the domain, which means that the boundary condition for pressure is not required.

• **Convection terms**

For $u$ / $v$ the terms using a upwind scheme are defined as:

$$Cu(i,j) = u_{avg,e}\frac{u(i,j+1)-u(i,j)}{dx} + u_{avg,w}\frac{u(i,j)-u(i,j-1)}{dx} + v_{avg,n}\frac{u(i+1,j)-u(i,j)}{dy} + v_{avg,s}\frac{u(i,j)-u(i-1,j)}{dy}$$

where $u_{avg,e}$, $u_{avg,w}$ are average $u$-velocities across east and west faces, and $v_{avg,n}$, $v_{avg,s}$ are average $v$-velocities across north and south faces of the $u$-momentum control volume, typically approximated as:

$$u_{avg,e} = max\{\frac{u(i,j)+u(i,j+1)}{2}, 0\}$$
$$u_{avg,w} = -max\{-\frac{u(i,j-1)+u(i,j)}{2}, 0\}$$
$$v_{avg,n} = max\{\frac{v(i,j)+v(i,j+1)}{2}, 0\}$$
$$v_{avg,s} = -max\{-\frac{v(i-1,j)+v(i-1,j+1)}{2}, 0\}$$

Note: These are simplified forms of convection terms. In practice, various schemes (e.g., Upwind, QUICK, Hybrid) are used to handle convection more robustly, especially at higher Reynolds numbers. The coefficients in the MATLAB code 'Fe', 'Fw', 'Fn', 'Fs' represent these fluxes.

```
% Coefficients for u-momentum equation (for INTERIOR u-faces)
for i = 2:ny + 1 % Iterate over y-indices of u-points
    for j = 2:nx % Iterate over x-indices of u-points
        % Convection terms (mass fluxes through faces)
        Fe = rho * dy * 0.5 * (u_old(i,j+1) + u_old(i,j));     % East face
        Fw = rho * dy * 0.5 * (u_old(i,j) + u_old(i,j-1));     % West face
        Fn = rho * dx * 0.5 * (v_old(i,j) + v_old(i,j+1));     % North face
        Fs = rho * dx * 0.5 * (v_old(i-1,j) + v_old(i-1,j+1)); % South face

        % Diffusion terms
        De = mu * dy / dx; Dw = mu * dy / dx;
        Dn = mu * dx / dy; Ds = mu * dx / dy;

        % Upwind differencing scheme coefficients
        aw_u(i,j) = Dw + max(0, Fw);   % West neighbor coefficient
        ae_u(i,j) = De + max(0, -Fe);  % East neighbor coefficient
        as_u(i,j) = Ds + max(0, Fs);   % South neighbor coefficient
        an_u(i,j) = Dn + max(0, -Fn);  % North neighbor coefficient

        % Central coefficient
        ap_u(i,j) = aw_u(i,j) + ae_u(i,j) + as_u(i,j) + an_u(i,j);

        % Pressure gradient source term
        Su = -(p(i-1,j) - p(i-1,j-1)) * dy;

        % Solve for predicted u-velocity
        u_star(i,j) = (aw_u(i,j)*u(i,j-1) + ae_u(i,j)*u(i,j+1) + ...
                       as_u(i,j)*u(i-1,j) + an_u(i,j)*u(i+1,j) + Su) / ap_u(i,j);
    end
end
```

### 2.4.3 Pressure Correction Equation and Velocity and Pressure Correction

```
% Set coefficients for pressure correction equation
for i = 1:ny
    for j = 1:nx
        % Handle boundary conditions for corner and edge cells
        if i == 1 && j == 1
            as_p(i,j) = 0; aw_p(i,j) = 0;
            ae_p(i,j) = rho * dy^2 / ap_u(i+1,j+1);
            an_p(i,j) = rho * dx^2 / ap_v(i+1,j+1);
        % ... (additional boundary conditions)
        else
```

```
            ae_p(i,j) = rho * dy^2 / ap_u(i+1,j+1);
            aw_p(i,j) = rho * dy^2 / ap_u(i+1,j);
            an_p(i,j) = rho * dx^2 / ap_v(i+1,j+1);
            as_p(i,j) = rho * dx^2 / ap_v(i,j+1);
        end
        ap_p(i,j) = ae_p(i,j) + aw_p(i,j) + an_p(i,j) + as_p(i,j);
    end
end
```

The pressure correction equation is solved using the Gauss-Seidel iterative method with convergence tolerance of $10^{-8}$.

Once the pressure correction is obtained, the pressure field is updated with under-relaxation:

```
% Correct pressure with under-relaxation
for i = 1:ny
    for j = 1:nx
        p(i,j) = p(i,j) + alpha_p * pc(i,j);
    end
end
```

The velocity field is then corrected using the pressure correction gradients:

```
% Correct u-velocity
for i = 2:ny + 1
    for j = 2:nx
        u(i,j) = u_star(i,j) + dy * (pc(i-1,j-1) - pc(i-1,j)) / ap_u(i,j);
    end
end

% Correct v-velocity
for i = 2:ny
    for j = 2:nx + 1
        v(i,j) = v_star(i,j) + dx * (pc(i-1,j-1) - pc(i,j-1)) / ap_v(i,j);
    end
end
```

### 2.4.4 Convergence Criteria

The simulation uses a continuity-based convergence criterion, monitoring the average mass imbalance across all cells:

```
% Calculate convergence metrics
max_mass_imbalance = max(abs(b(:)));
total_mass_imbalance = sum(abs(b(:)));
num_interior_cells = ny * nx;
avg_mass_imbalance = total_mass_imbalance / num_interior_cells;

% Check convergence
if avg_mass_imbalance < continuity_tolerance
    converged = true;
    fprintf('\nConverged after %d iterations!\n', iteration);
end
```

## 2.5 Results and Analysis

The numerical simulations were conducted for three different Reynolds numbers (Re = 400, 1000, and 7500) using a grid resolution of 128 points in each direction. The computational results are compared with the benchmark data from Ghia et al. (1982) to validate the accuracy of the implemented numerical scheme.

### 2.5.1 Velocity Profile Comparison

Table 2 presents a quantitative comparison of the v-velocity component along the horizontal centerline passing through the geometric center of the cavity. The results show reasonable agreement with

the reference data, particularly for the lower Reynolds numbers. Some discrepancies are observed, especially at higher Reynolds numbers, which can be attributed to the grid resolution limitations and the complexity of the flow physics at higher Re values.

Table 2: v-velocity ($v/U_0$) along horizontal centerline through geometric center of cavity

| $y/D$ | Grid Point | Ghia et al. (1982) | | | Current Study | | |
|---|---|---|---|---|---|---|---|
| | | Re = 400 | Re = 1000 | Re = 7500 | Re = 400 | Re = 1000 | Re = 7500 |
| 1.0000 | 129 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 0.9688 | 125 | -0.12146 | -0.21388 | -0.53858 | -0.1878 | -0.2773 | -0.3820 |
| 0.9609 | 124 | -0.15663 | -0.27669 | -0.55216 | -0.2406 | -0.3363 | -0.3542 |
| 0.9531 | 123 | -0.19254 | -0.33714 | -0.52347 | -0.2406 | -0.3782 | -0.3128 |
| 0.9453 | 122 | -0.22847 | -0.39188 | -0.48590 | -0.3313 | -0.4026 | -0.2711 |
| 0.9063 | 117 | -0.23827 | -0.51550 | -0.41050 | -0.4227 | -0.3543 | -0.1581 |
| 0.8594 | 111 | -0.44993 | -0.42665 | -0.36213 | -0.3557 | -0.2241 | -0.1264 |
| 0.8047 | 104 | -0.38598 | -0.31966 | -0.30448 | -0.2384 | -0.1393 | -0.0991 |
| 0.5000 | 65 | 0.05186 | 0.02526 | 0.00824 | 0.1225 | 0.00865 | 0.0197 |
| 0.2344 | 31 | 0.30174 | 0.32235 | 0.27348 | 0.1545 | 0.1381 | 0.1091 |
| 0.2266 | 30 | 0.32203 | 0.33075 | 0.28117 | 0.1536 | 0.1374 | 0.1083 |
| 0.1563 | 21 | 0.28124 | 0.37095 | 0.35060 | 0.1466 | 0.1315 | 0.1013 |
| 0.0938 | 13 | 0.22965 | 0.32627 | 0.41824 | 0.1362 | 0.1270 | 0.0966 |
| 0.0781 | 11 | 0.20920 | 0.30353 | 0.43564 | 0.1300 | 0.1246 | 0.0957 |
| 0.0703 | 10 | 0.19713 | 0.29012 | 0.44030 | 0.1256 | 0.1227 | 0.0953 |
| 0.0625 | 9 | 0.18360 | 0.27485 | 0.43979 | 0.1256 | 0.1199 | 0.0949 |
| 0.0000 | 1 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |

### 2.5.2 Convergence Analysis

Figure 9 demonstrates the convergence behavior of the numerical solution for all three Reynolds numbers. The convergence curves show that the solution reaches steady-state conditions within reasonable computational time. The convergence rate decreases with increasing Reynolds number.

It is worth noting that at high Reynolds number($Re = 7500$), the SIMPLE algorithm is no longer convergent as the time progressing, so we take the minimal residual error for analysis. However, the result cannot be fully trusted, so we recalculate the high Reynolds number condition using OpenFOAM in the following section.

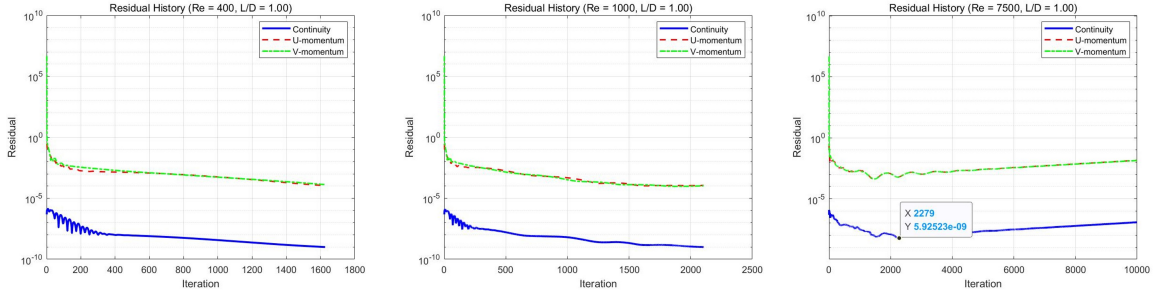


Figure 9: Convergence Curves for $Re = 400, 1000, 7500$

### 2.5.3 Flow Field Visualization

The streamline patterns shown in Figure 10 reveal the characteristic flow structures within the lid-driven cavity for different Reynolds numbers. At Re = 400, the flow exhibits a single primary recirculation zone with relatively smooth streamlines. As the Reynolds number increases to 1000, the flow becomes more complex with the appearance of secondary vortices in the corners of the cavity. At Re

= 7500, the flow field demonstrates highly complex behavior with multiple recirculation zones and enhanced mixing characteristics, indicating the transition towards turbulent flow regimes.

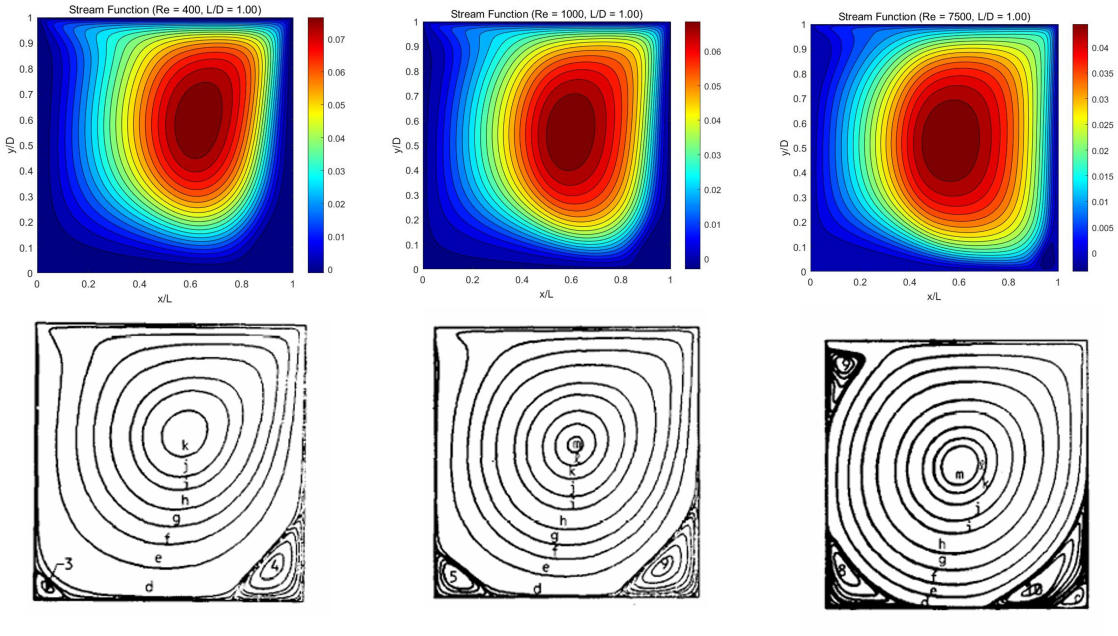


Figure 10: Streamline patterns for $Re = 400, 1000, 7500$

The velocity field visualization in Figure 11 provides additional insights into the flow dynamics. The velocity magnitude contours clearly show the development of boundary layers along the walls and the formation of high-velocity regions near the moving lid. The velocity gradients become steeper with increasing Reynolds number, particularly near the walls, which is consistent with the theoretical expectations for viscous flow behavior.

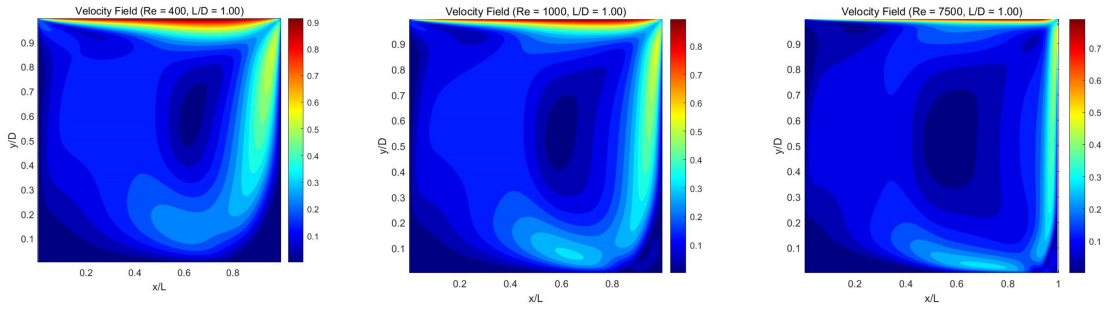


Figure 11: Velocity field magnitude contours for $Re = 400, 1000, 7500$

## 2.6 Reslut for different L/D

As for different L/D ration, Streamline patterns, Velocity field distribution and Centerline velocity profiles are display in following part.

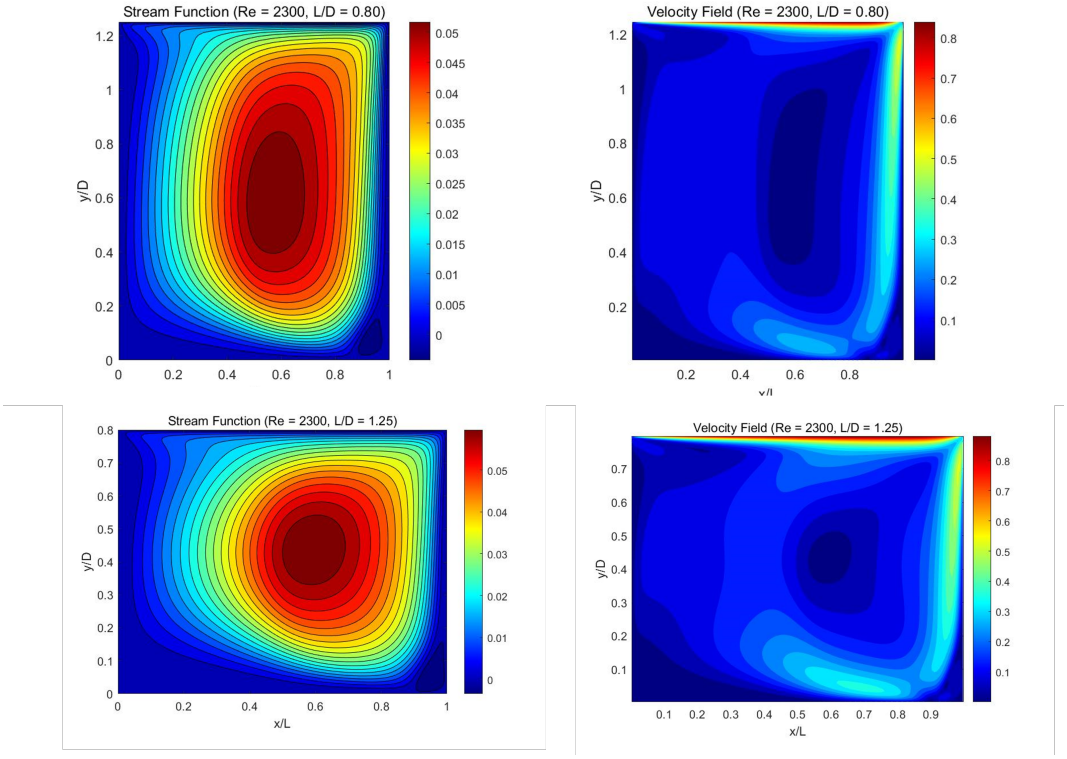


Figure 12: Velocity field and Stream line for $Re = 2300$ with $L/D = 0.8, 1.25$

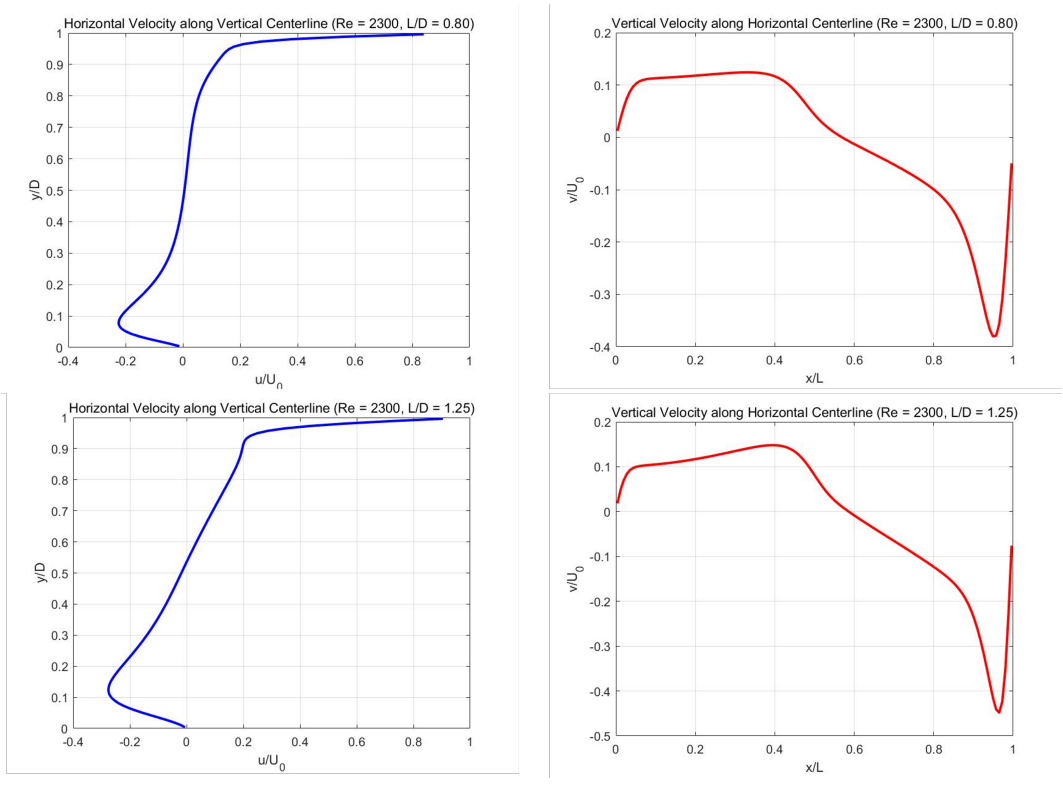


Figure 13: Central velocity for $Re = 2300$ with $L/D = 0.8, 1.25$

# 3 Advanced algorithm implementation and case analysis

Due to the non-convergence of the self-written MATLAB algorithm at high Reynolds numbers, this study employs the PisoFoam Solver in OpenFOAM for numerical simulation. The non-uniform grid will be provided in the next file, while the uniform grid is prepared with the same number of cells for comparison.

## 3.1 Mesh Configuration

Two mesh configurations were implemented:

1. **Uniform Mesh**: Equally spaced grid points in all three directions
2. **Non-uniform Mesh**: Graded mesh with refined regions near boundaries

The non-uniform mesh employs a grading strategy as shown in the blockMeshDict configuration (provided in separate file), featuring:

- X-direction grading: (0.2 0.4 5), (0.6 0.2 1), (0.2 0.4 0.2) for three segments
- Y-direction grading: (0.2 0.4 5), (0.6 0.2 1), (0.2 0.4 0.2) for three segments
- Z-direction grading: Uniform (grading = 1)

Both meshes maintain identical total cell counts to ensure fair computational comparison as shown in fig.14.

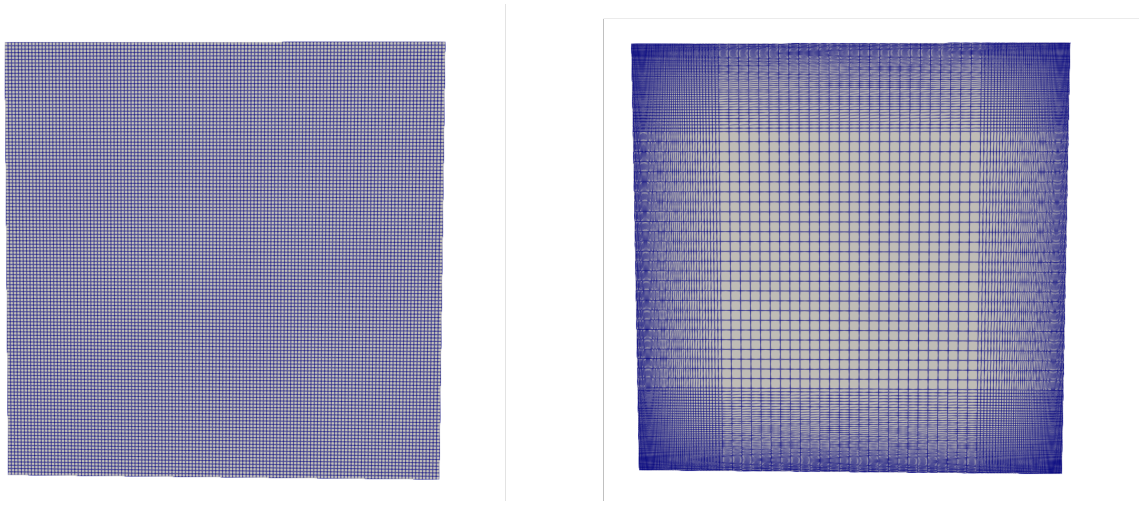

Figure 14: Uniform mesh vs. non-uniform mesh

## 3.2 Computational Cost Analysis and Convergence Performance

After running the same number of outer iterations, the non-uniform mesh required significantly more CPU time compared to the uniform mesh as show in table3. This increased computational cost stems may from **Matrix conditioning**. The resulting coefficient matrices exhibit poorer conditioning due to mesh nonuniformity, requiring more computational effort per linear solver iteration.

Table 3: Computational Efficiency Comparison after 1000 times iteration

| Mesh Type | CPU Time (s) | U-Convergence |
|---|---|---|
| Uniform | 27.28s | 6.7192e-06 |
| Non-uniform | 47.92s | 3.2967e-07 |

Despite the higher per-iteration cost, the non-uniform mesh demonstrated superior convergence characteristics:

- **Faster residual reduction**: The non-uniform mesh achieved lower residual levels more rapidly due to enhanced resolution of critical flow regions
- **Boundary layer capture**: Refined cells near cavity walls better resolve steep velocity gradients, leading to more physically accurate solutions and improved numerical stability

The non-uniform mesh presents a classic computational trade-off: higher per-iteration cost but faster convergence. For high Reynolds number flows where boundary layer resolution is critical, the improved convergence rate often compensates for the increased computational overhead, particularly for steady-state solutions where total iteration count becomes the determining factor for overall efficiency.

## 3.3 Flow Characteristics Analysis at Different Reynolds Numbers

To analyze flow characteristics at different Reynolds numbers, two-dimensional lid-driven cavity simulations were performed using OpenFOAM's PisoFOAM solver for higher Reynolds numbers and self-written MATLAB code for low Renolds number. Post-processing and visualization were conducted using Paraview to examine velocity fields, streamlines, and vortex structures.

### 3.3.1 Low Reynolds Number (Re = 400)

At Re = 400, the flow exhibits classical characteristics:

- Single primary recirculation zone occupying most of the cavity
- Steady-state solution achieved rapidly
- Maximum velocity occurs near the center of the primary vortex

### 3.3.2 Moderate Reynolds Number (Re = 1000)

As Reynolds number increases to 1000:

- **Secondary corner vortices emerge**: Small counter-rotating vortices appear in the bottom corners
- **Primary vortex shifts**: The center of the main recirculation zone moves toward the geometric center

### 3.3.3 High Reynolds Number (Re = 7500)

At higher Reynolds numbers:

- **Multiple vortex structures**: Additional tertiary vortices develop along cavity walls
- **Boundary layer thinning**: Wall boundary layers become significantly thinner, requiring higher mesh resolution
- **Increased circulation**: Higher velocity magnitudes and stronger vorticity concentrations

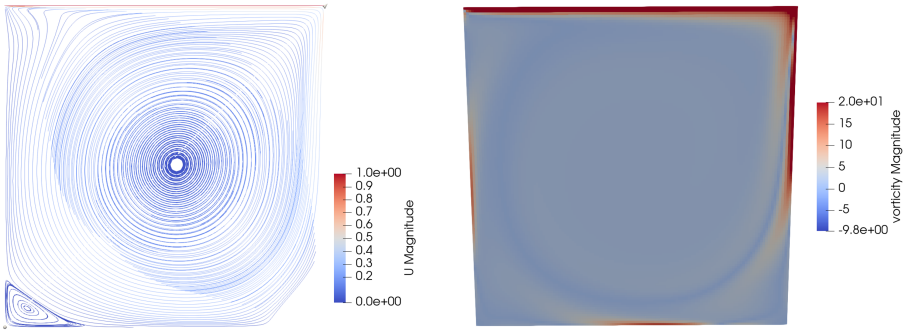


Figure 15: Streamline and vorticity field for $Re = 7500$ in paraview

# References

[1] Ghia, U., Ghia, K.N., and Shin, C.T. (1982). High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of Computational Physics*, 48(3), 387-411.

[2] Versteeg, H.K. and Malalasekera, W. (2007). *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education Limited.

[3] Ferziger, J.H. and Perić, M. (2002). *Computational Methods for Fluid Dynamics*. Springer-Verlag.

[4] OpenFOAM Foundation. (2023). *OpenFOAM User Guide*.