Lid Driven Cavity Using Fractional Step Algorithm

Subodh Chander*

Virginia Polytechnic Institute and State University, Blacksburg, VA, 24060, USA

I. Introduction

A rectangular or cubic container represents one of the most basic confined shapes for studying fluid motion. The simplest type of mechanical force that can be applied to a viscous fluid of uniform density, without altering the basic shape of the domain, is the tangential in-plane movement of one of its boundary walls. This type of system, where one of the solid walls of a cuboid moves tangentially to itself, is known as a lid-driven cavity.

II. Problem description

The problem was solved using steady state, incompressible Navier-Stokes equations. The 2-D cavity was modeled as a square box. The boundaries were defined as a Dirchlet boundary conditions, where sides and bottom boundaries were considered fixed, while the top was setup as a moving boundary with the non-dimensional velocity Ulid = 1.0. The Initial simulation was run using mesh size of 32x32, it was assumed that the flow is constant density, viscosity and temperature. The problem was analysed for Reynolds number 100, and 1000 using Fractional step algorithm. Finer mesh size of 64x64 and 128x128 were analysed as well. A Matlab code was written to solve the Navier Stokes equations with the boundary and initial conditions of the problem and compared with the results from [2]

III. Governing Equations and boundary conditions

The Navier Stokes equation for a 2-D incompressible flow can be written as:

Conservation of Mass

$$\nabla \cdot \vec{u^*} = 0 \tag{1}$$

*Graduate Student, Aerospace and Ocean Engineering, Address, and AIAA Member Grade.

Conservation of Momentum

$$\frac{\partial \vec{u^*}}{\partial t} + \nabla \cdot (\vec{u^*}\vec{u^*}) = -\frac{1}{\rho^*}\nabla p^* + \nu^* \nabla \cdot (\nabla \vec{u^*}) \tag{2}$$

In x and y components it can be written as:

$$\frac{\partial u^*}{\partial x} + \frac{\partial u^*}{\partial y} = 0 \tag{3}$$

$$\frac{\partial u^*}{\partial t} + \frac{\partial u^{*2}}{\partial x} + \frac{\partial u^* v^*}{\partial y} = -\frac{1}{\rho^*}\frac{\partial p^*}{\partial x} + \nu^*(\frac{\partial^2 u^*}{\partial x^2} + \frac{\partial^2 u^*}{\partial y^2}) \tag{4}$$

$$\frac{\partial v^*}{\partial t} + \frac{\partial v^* u^*}{\partial x} + \frac{\partial v^{*2}}{\partial y} = -\frac{1}{\rho^*}\frac{\partial p^*}{\partial x} + \nu^*(\frac{\partial^2 v^*}{\partial x^2} + \frac{\partial^2 v^*}{\partial y^2}) \tag{5}$$

where:

- u^* = x-velocity.
- v^* = y-velocity.
- p^* = Pressure.
- ρ^* = density.
- ν^* = kinematic viscosity.

The governing equations are non-dimensionalized for solving for numerical accuracy and stability. The non-dimensional variables are defined as x,y, u,v,p, ρ, ν, such that $x = \frac{x^*}{L_{ref}}$, $y = \frac{y^*}{L_{ref}}$, $u = \frac{u^*}{u_{ref}}$, $u = \frac{v^*}{u_{ref}}$, $\rho = \frac{\rho^*}{\rho_{ref}}$, $\nu = \frac{\nu^*}{\nu_{ref}}$

The governing equations in the non dimensional form can now be written as:

$$\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} = 0 \tag{6}$$

$$\frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} + \frac{\partial uv}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \tag{7}$$

$$\frac{\partial v}{\partial t} + \frac{\partial uv}{\partial x} + \frac{\partial v^2}{\partial y} = -\frac{\partial p}{\partial y} + \frac{1}{Re}(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) \tag{8}$$

where $Re = \frac{u_{ref} L_{ref}}{\nu_{ref}}$

A. Dirichlet Boundary Conditions

The non dimensional u and v velocity on the bottom, left and right boundary are set to zero. The top boundary is assigned a non dimensional velocity along x-axis , u = 1 and velocity along y axis is set as zero.

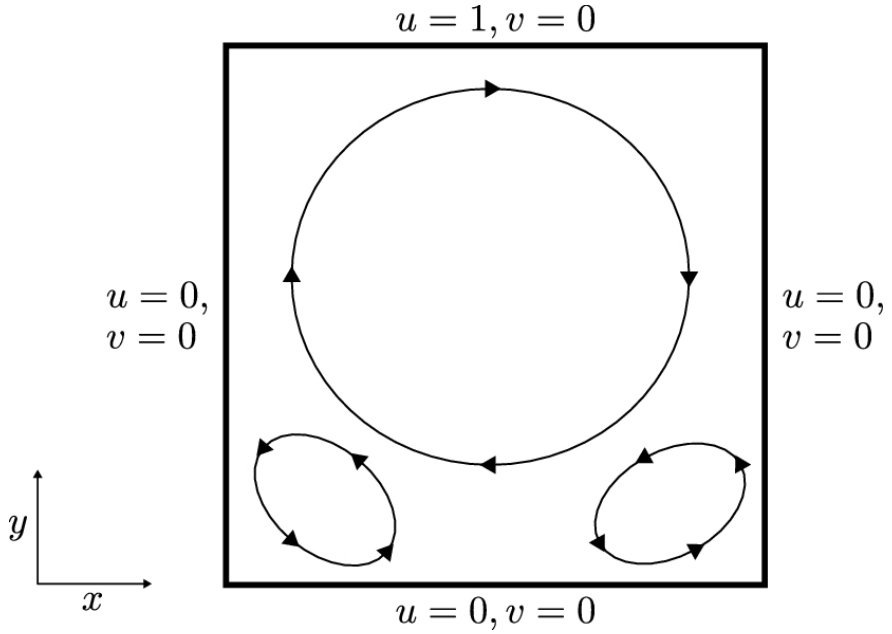

Figure 1. Boundary Conditions

IV. Numerical discretization

A. Governing Equations in Integral From

The continuity equation can be written as:

$$\nabla \cdot \vec{u} = 0 \tag{9}$$

For finite volume method, integrating the above equations over region R and applying divergence theorem:

$$\int_R \nabla \cdot \vec{u} \quad dv = \oint_{\delta R} \vec{u} \cdot \vec{n} \; ds = 0 \tag{10}$$

Integrating x-momentum equation :

$$\int_R \frac{\partial u}{\partial t} + \int_R (\frac{\partial u^2}{\partial x} + \frac{\partial uv}{\partial y}) = -\int_R \frac{\partial p}{\partial x} + \frac{1}{Re} \int_R (\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \tag{11}$$

or,

$$\int_R \frac{\partial u}{\partial t} \; dv + \int_R (\nabla \cdot u\vec{u}) \; dv = -\int_R \frac{\partial p}{\partial x} \; dv + \frac{1}{Re} \int_R \nabla \cdot (\nabla u) \; dv \tag{12}$$

Applying divergence theorem:

$$\frac{\partial}{\partial t} \int_R u \; dv + \oint_{\delta R} u(\vec{u} \cdot \vec{n}) \; ds = -\oint_{\delta R} p \; n_x \; ds + \frac{1}{Re} \oint_{\delta R} \nabla u \cdot \vec{n} \; ds \tag{13}$$

Similarly Y- momentum equation in integral form can be written:

$$\frac{\partial}{\partial t} \int_R v \; dv + \oint_{\delta R} v(\vec{v} \cdot \vec{n}) \; ds = -\oint_{\delta R} p \; n_y \; ds + \frac{1}{Re} \oint_{\delta R} \nabla v \cdot \vec{n} \; ds \tag{14}$$

B. Staggered Grid Discretization

The finite volume discretization of the problem is done using staggered grid.

1. *Continuity*

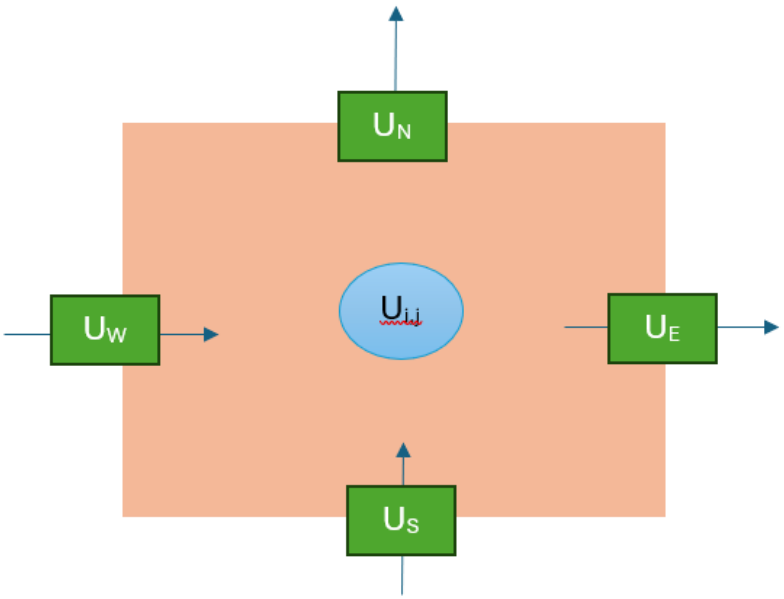

Figure 2. velocity

For a single cell the continuity equation can be written as:
$\oint_{\delta R} \vec{u} \cdot \vec{n}\ ds = 0$

$$\sum_{faces} (\vec{u} \cdot \vec{n})_{each face} ds_{face} = 0 \tag{15}$$

$$(u_e \delta y - u_w \delta y) + (u_n \delta x - u_s \delta x) = 0 \tag{16}$$

$$\frac{u_e - u_w}{dx} + \frac{u_n - u_s}{dy} = 0 \tag{17}$$

At t=n+1 time step, for velocity to be divergence free:

$$\frac{u_e^{n+1} - u_w^{n+1}}{dx} + \frac{u_n^{n+1} - u_s^{n+1}}{dy} = 0 \tag{18}$$

By storing velocity components directly at the control volume faces (where the fluxes are calculated), a staggered grid provides a more direct and physically accurate calculation of convective and viscous fluxes. This reduces interpolation errors that can occur on collocated grids, where velocity values at the faces must be interpolated from values at the centers, potentially introducing inaccuracies. In other words, staggering implies different control

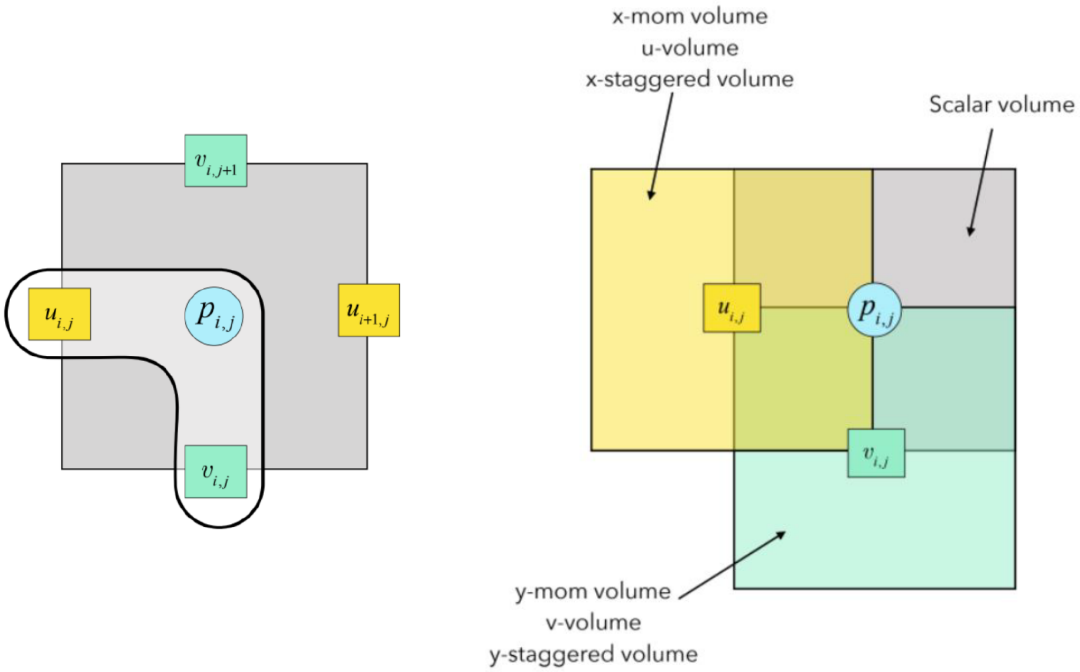

Figure 3. Staggering of Pressure and Velocity

volumes for different quantities. Scalar quantities like pressure are calculated at cell centre and the vector quantities such as velocity, momentum are calculated at the cell face.

Focusing on the u-staggered volume and calculating the velocity at face for x-momentum, see figure **??**.

$$
\begin{aligned}
u_{w-face} &= \frac{u_{i,j} + u_{i-1,j}}{2} \\
u_{e-face} &= \frac{u_{i,j} + u_{i+1,j}}{2} \\
u_{n-face} &= \frac{u_{i,j} + u_{i,j+1}}{2} \\
u_{s-face} &= \frac{u_{i,j} + u_{i,j-1}}{2} \\
v_{s-face} &= \frac{v_{i,j} + v_{i-1,j}}{2} \\
v_{n-face} &= \frac{v_{i,j+1} + v_{i-1,j+1}}{2}
\end{aligned}
\tag{19}
$$

Focusing on the v-staggered volume and calculating the velocity at face for y-momentum, see figure 1.

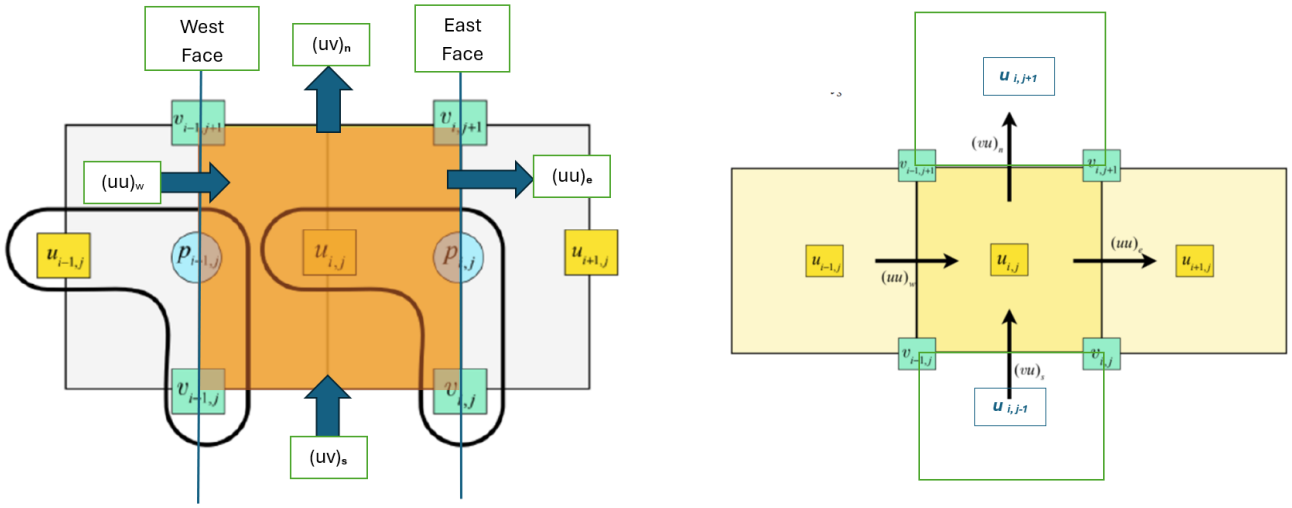

Figure 4. U-velocity control volume

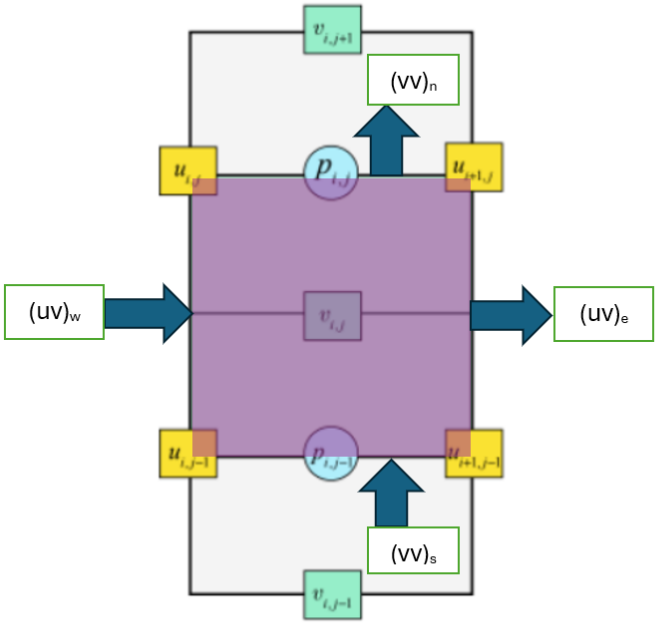

Figure 5. Y-staggered volume

$$
\begin{aligned}
u_{w-face} &= \frac{u_{i,j} + u_{i,j-1}}{2} \\
u_{e-face} &= \frac{u_{i+1,j} + u_{i+1,j-1}}{2} \\
v_{s-face} &= \frac{v_{i,j} + v_{i,j-1}}{2} \\
v_{n-face} &= \frac{v_{i,j} + v_{i,j+1}}{2} \\
v_{w-face} &= \frac{v_{i,j} + v_{i-1,j}}{2} \\
v_{e-face} &= \frac{v_{i,j} + v_{i+1,j}}{2}
\end{aligned} \tag{20}
$$

The pressure gradient on u-control volume can be written as:

$$
\oint_{\delta R} p \cdot n_x = (p_{i,j} - p_{i-1,j})\delta y \tag{21}
$$

similarly, The pressure gradient on v-control volume can be written as:

$$
\oint_{\delta R} p \cdot n_y = (p_{i,j} - p_{i,j-1})\delta x \tag{22}
$$

Diffusion Term u-Control Volume

$$
\begin{aligned}
\oint_{\delta R} \nabla u \cdot \vec{n}\ ds &= (\frac{\delta u}{\delta x}_e \delta y - \frac{\delta u}{\delta x}_w \delta y) + (\frac{\delta u}{\delta y}_n \delta x - \frac{\delta u}{\delta y}_s \delta x) \\
\frac{\delta u}{\delta x}_e &= \frac{u_{i+1,j} - u_{i,j}}{\delta x} \\
\frac{\delta u}{\delta x}_w &= \frac{u_{i,j} - u_{i-1,j}}{\delta x} \\
\frac{\delta u}{\delta y}_s &= \frac{u_{i,j} - u_{i,j-1}}{\delta y} \\
\frac{\delta u}{\delta y}_n &= \frac{u_{i,j+1} - u_{i,j}}{\delta y} \\
\oint_{\delta R} \nabla u \cdot \vec{n}\ ds &= \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\delta x}\ \delta y\ + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\delta y}\ \delta x
\end{aligned} \tag{23}
$$

Similarly for v-control Volume

Diffusion Term v-Control Volume

$$\oint_{\delta R} \nabla v \cdot \vec{n}\ ds = (\frac{\delta v}{\delta x}_e \delta y - \frac{\delta v}{\delta x}_w \delta y) + (\frac{\delta v}{\delta y}_n \delta x - \frac{\delta v}{\delta y}_s \delta x)$$

$$\begin{aligned} \frac{\delta v}{\delta x}_e &= \frac{v_{i+1,j} - v_{i,j}}{\delta x} \\ \frac{\delta v}{\delta x}_w &= \frac{v_{i,j} - v_{i-1,j}}{\delta x} \\ \frac{\delta v}{\delta y}_s &= \frac{v_{i,j} - v_{i,j-1}}{\delta y} \\ \frac{\delta v}{\delta y}_n &= \frac{v_{i,j+1} - v_{i,j}}{\delta y} \end{aligned} \tag{24}$$

$$\oint_{\delta R} \nabla u \cdot \vec{n}\ ds = \frac{v_{i-1,j} - 2v_{i,j} + v_{i+1,j}}{\delta x}\ \delta y\ + \frac{v_{i,j-1} - 2v_{i,j} + v_{i,j+1}}{\delta y}\ \delta x$$

V. Algorithm and solution procedure

Fractional step method is used for solving the Navier-Stokes equations. Fractional Step Method breaks down the Navier-Stokes equations into simpler sub-problems that can be solved sequentially. The momentum equations are solved first to predict an intermediate velocity field without considering the pressure term. The solution at this stage does not satisfy the continuity equation.

To enforce incompressibility and correct the intermediate velocity field, a pressure correction equation (Poisson's equation) is derived from the continuity equation. Solving this Poisson equation for pressure allows updating the velocity field so that it satisfies the continuity equation.

The final step involves updating the velocity field using the newly computed pressure field. This ensures that the velocity field is divergence-free, satisfying the incompressibility condition.

The incompressible Navier-Stokes equation in tensor notations are:

$$\begin{aligned} \frac{\delta u_i}{\delta t} + \nabla \cdot (u_j\ u_i) &= -\nabla\ p + \frac{1}{Re}\nabla^2 u_i \\ \nabla \cdot u_i &= 0 \end{aligned} \tag{25}$$

These equations can be symbolically represented as:

$$\begin{aligned} \frac{\delta u_i}{\delta t} + D_j \cdot (u_j\ u_i) &= -G_i\ p + \frac{1}{Re} D_j \cdot (G_j\ u_i) \\ D_j \cdot u_i &= 0 \end{aligned} \tag{26}$$

$$\frac{\delta u_i}{\delta t} = -D_j \cdot (u_j\ u_i) - G_i\ p + \frac{1}{Re} D_j \cdot (G_j\ u_i) \tag{27}$$

$$\frac{\delta u_i}{\delta t} = -D_j \cdot (u_j\ u_i) - G_i\ p + \frac{1}{Re} D_j \cdot (G_j\ u_i) \tag{28}$$

Convective and diffusive terms can be combined as:

$$H_i = -D_j \cdot (u_j\ u_i) + \frac{1}{Re} D_j \cdot (G_j\ u_i) \tag{29}$$

Momentum equation can be written as:

$$\frac{\delta u_i}{\delta t} = H_i - G_i\ p \tag{30}$$

Integrating from t to t+Δt

$$\tilde{u_i} = u_i^n + \int_{t_n}^{t_n+\Delta t} H_i^n dt - \int_{t_n}^{t_n+\Delta t} G_i\ p\ dt \tag{31}$$

Predictor Step

Momentum equation is decoupled from pressure and the velocity calculated for intermediate step.

$$\tilde{u_i} = u_i^n + \int_{t_n}^{t_n+\Delta t} H_i^n dt \tag{32}$$

$\int H_i\ dt$ is approximated using second order Adams- Bashforth method.

$$\int_{t_n}^{t_n+\Delta t} H_i^n \approx \Delta t\ [\ \frac{3}{2}\ H_i^n - \frac{1}{2}\ H_i^{n-1}] \tag{33}$$

Pressure Correction In this step the pressure correction is applied to enforce incompressibility:

$$D_i\ \cdot (G_i\ p^{n+1}) = \frac{1}{\Delta t} D_i \cdot (\tilde{u}) \tag{34}$$

For 2d , Poisson equation can be written as:

$$\begin{aligned} \nabla^2\ p &= \frac{1}{\Delta t}\ \nabla \cdot\ \tilde{u} \\ (\frac{d^2p}{dx^2} + \frac{d^2p}{dy^2}) &= (\frac{1}{dt})(\frac{d\tilde{u}}{dx} + \frac{d\tilde{v}}{dy}) \end{aligned} \tag{35}$$

$LHS_{i,j} = \frac{(p_{i+1,j})-(2p_{i,j})+(p_{i-1,j})}{\Delta x^2} + \frac{(p_{i,j+1})-(2p_{i,j})+(p_{i,j-1})}{\Delta y^2}$
$RHS_{i,j} = \frac{1}{dt}(\frac{(u_{i+1,j})-(u_{i,j})}{dx} + \frac{(v_{i,j+1})-(v_{i,j})}{dy})$

Coefficients for the pressure calculation can be determined from discretized Laplacian term(LHS): $A_w = \frac{1}{\Delta x^2},\ A_e = \frac{1}{\Delta x^2}\ A_n = \frac{1}{\Delta y^2}\ A_s = \frac{1}{\Delta y^2}\ A_p = \frac{-2}{\Delta x^2 + \Delta y^2}$

Finally the intermediate velocity is updated with the pressure correction to determine the velocity at next time step.

$$u_i^{n+1} = \tilde{u} - \Delta t\ G_i\ p^{n+1} \tag{36}$$

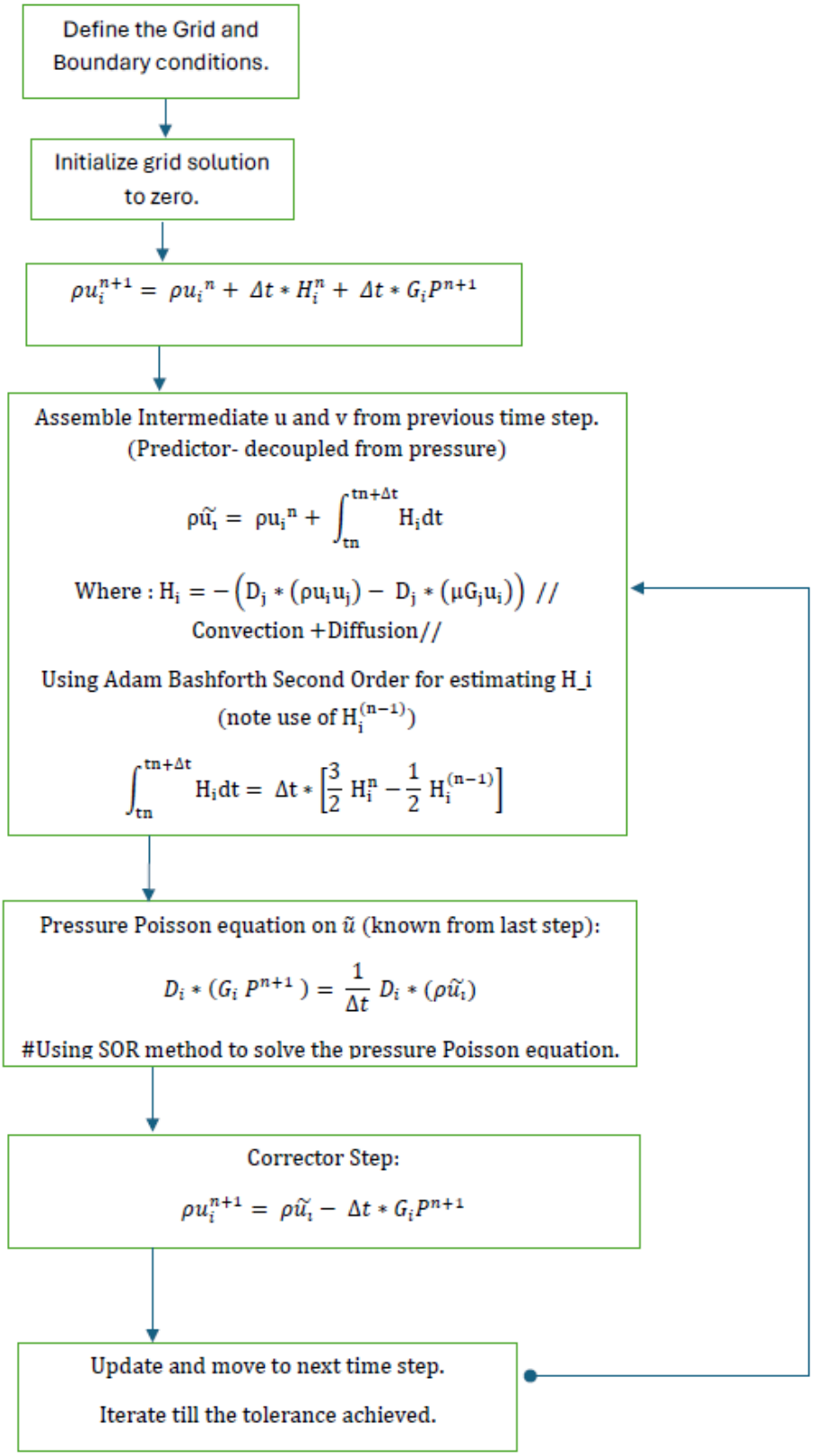

Figure 6. Fractional Step Method

VI. Boundary Conditions (Ghost Cells)

Left Wall :

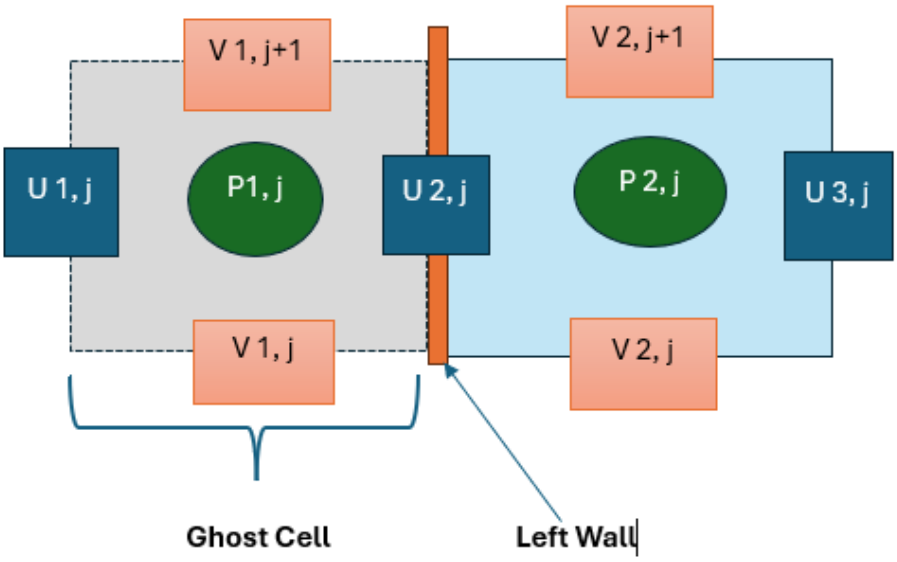

Figure 7. Left Boundary

u-velocity lies on the left wall face:

$$u_{2,j} = 0 \quad \forall\ j : 2\ to\ (end - 1) \tag{37}$$

v velocity at wall is zero but need interpolation because of ghost cell, therefore:

$$\frac{v_{1,j} + v_{2,j}}{2} = v_{wall} = 0 \quad \forall\ j : 1\ to\ (end - 1) \tag{38}$$

Right Wall : Similarily, u-velocity lies on the right wall face:

$$u_{end,j} = 0 \quad \forall\ j : 2\ to\ (end - 1) \tag{39}$$

v velocity at wall is zero but need interpolation because of ghost cell, therefore:

$$\frac{v_{end,j} + v_{end-1,j}}{2} = v_{wall} = 0 \quad \forall\ j : 1\ to\ (end - 1) \tag{40}$$

Top Wall :

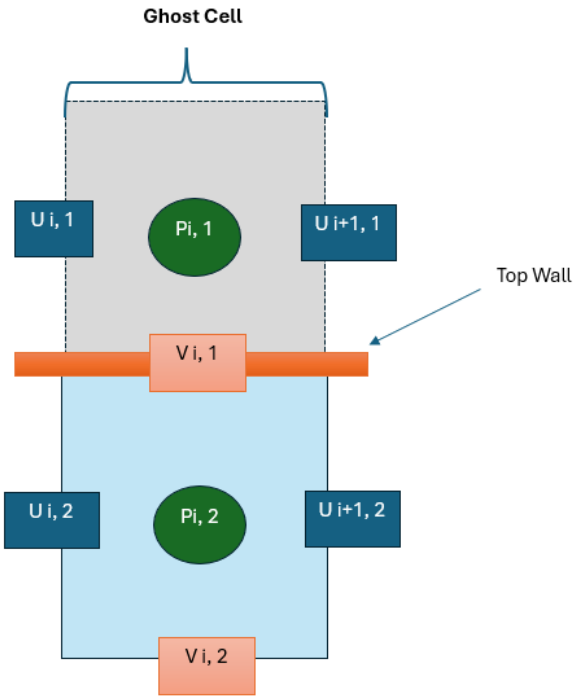

Figure 8. Top Boundary

u velocity at top wall is 1 but need interpolation because of ghost cell,therefore:

$$\frac{u_{i,1} + u_{end-i,2}}{2} = u_t = 1 \quad \forall\ j : 2\ to\ (end) \tag{41}$$

v-velocity lies on the top wall face and is zero:

$$v_{i,1} = 0 \quad \forall\ j : 2\ to\ (end - 1) \tag{42}$$

Bottom Wall : Similarly on the bottom wall: u velocity at bottom wall is 0 but need interpolation because of ghost cell,therefore:

$$\frac{u_{i,end} + u_{end-i,end-1}}{2} = 0 \quad \forall\ i : 2\ to\ (end) \tag{43}$$

v-velocity lies on the top wall face and is zero:

$$v_{i,end} = 0 \quad \forall\ i : 2\ to\ (end - 1) \tag{44}$$

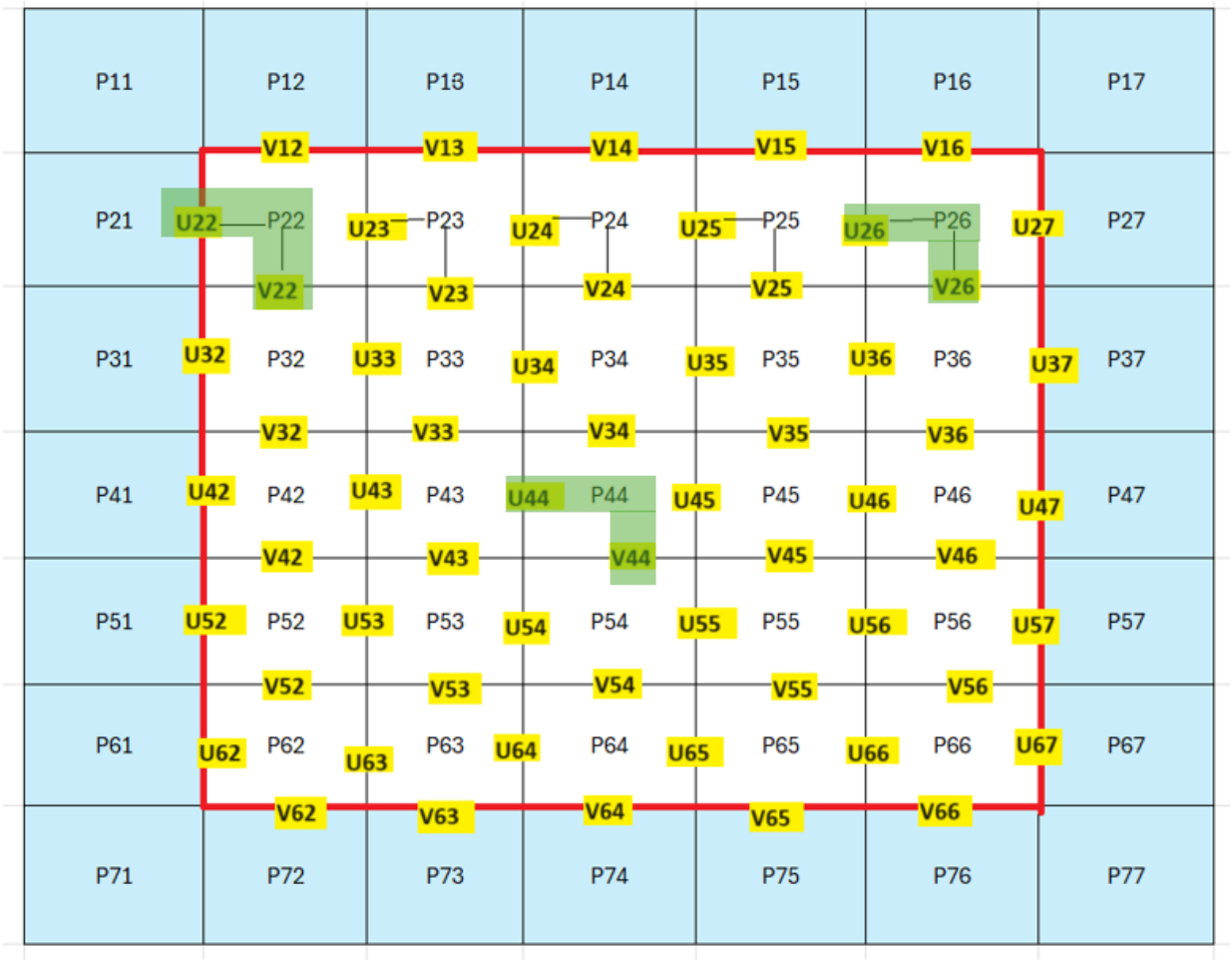

Figure 9. Staggered Grid

VII. Results

A. Part A

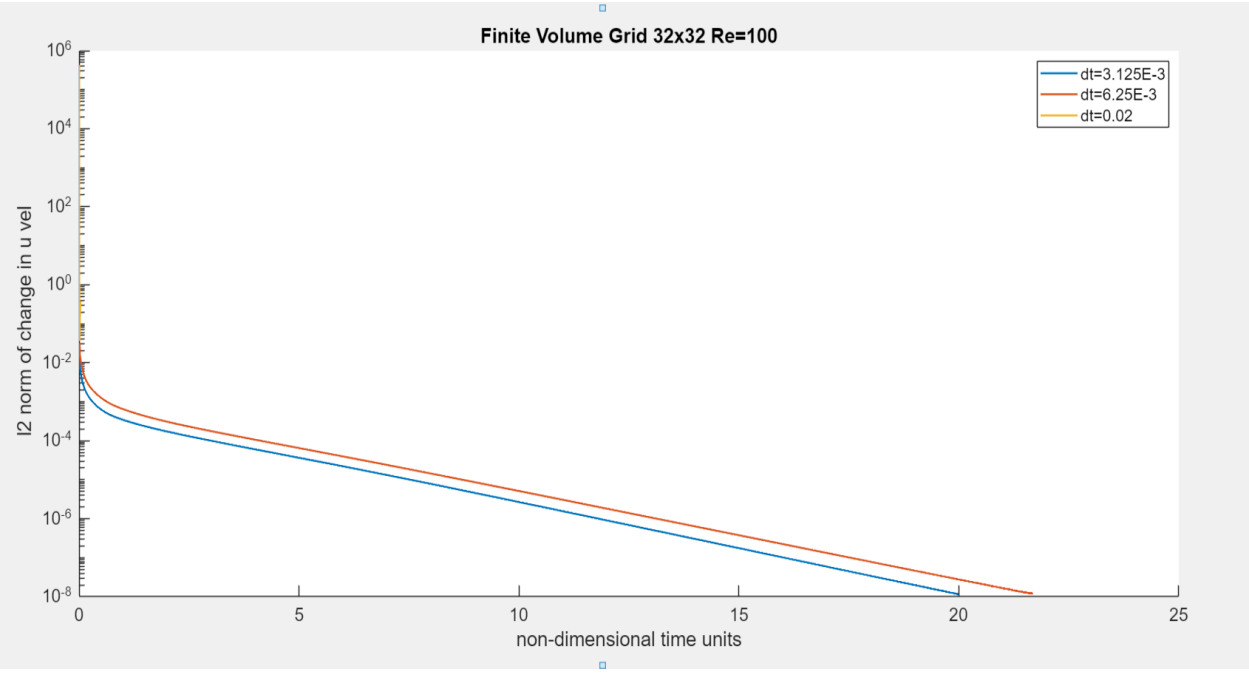

Figure 10. Grid 32x32 L2 Norm of Change in U velocity

For each of the three grid resolutions, it's seen that the required non-dimensional time units for convergence grow with an increase in the time step. Notably, on the 32x32 grid at a time step of 0.02, the solver diverge.

Schneider et al. [3], have demonstrated that for convection dominated flows, stability requirements are more demanding than those defined by the linear CFL condition. It was found that higher-order methods, including Adams-Bashforth and Runge-Kutta, are constrained by a non-linear CFL-like condition, $\delta t \leq C(\frac{\delta x}{U})^{\alpha}$; where C is between 0 and 1 and alpha determined by the scheme [1].

Second Order Adam Bashforth:

$\delta t <= 2^{2/3}\ C^{1/3}\ (\frac{\delta X}{U})^{(4/3)}$

For $32 * 32$ grid , max $C = 1$ for Adam Bashforth scheme gives $\delta t = 0.0156$, which is lower than the time step 0.02 therefore the solver diverges.

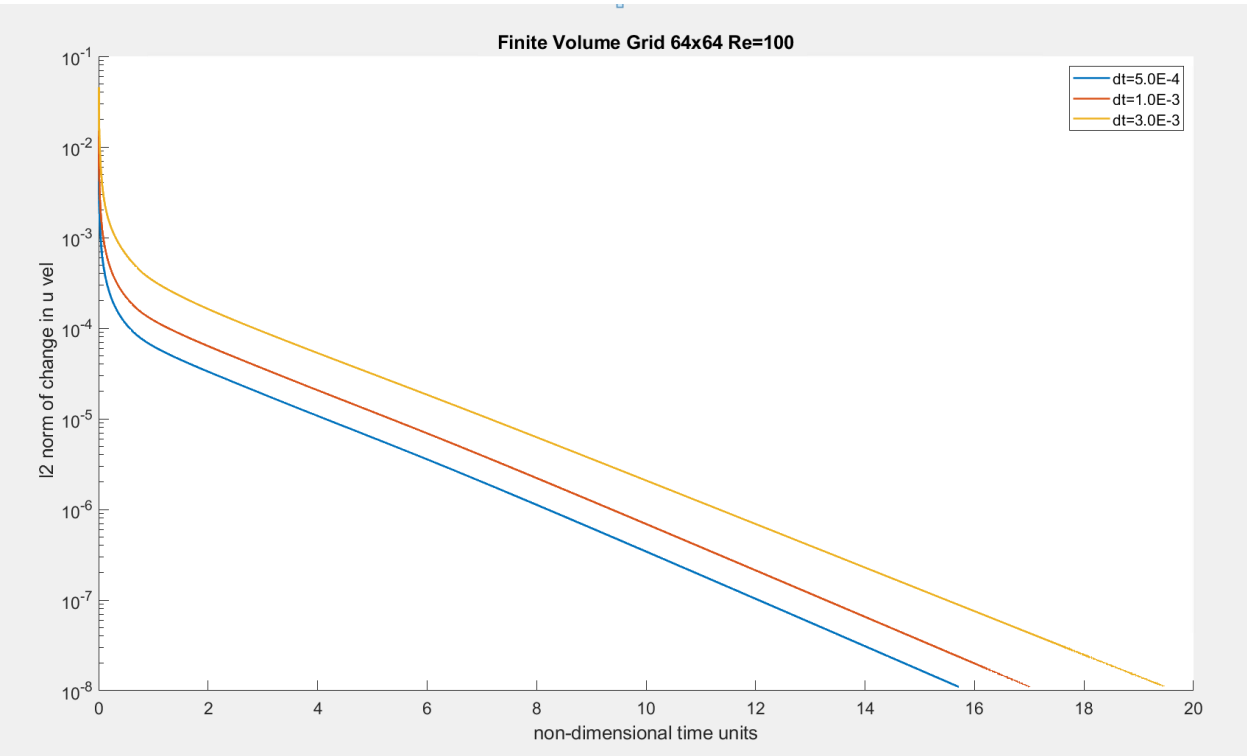

Figure 11. Grid 64x64 L2 Norm of Change in U velocity

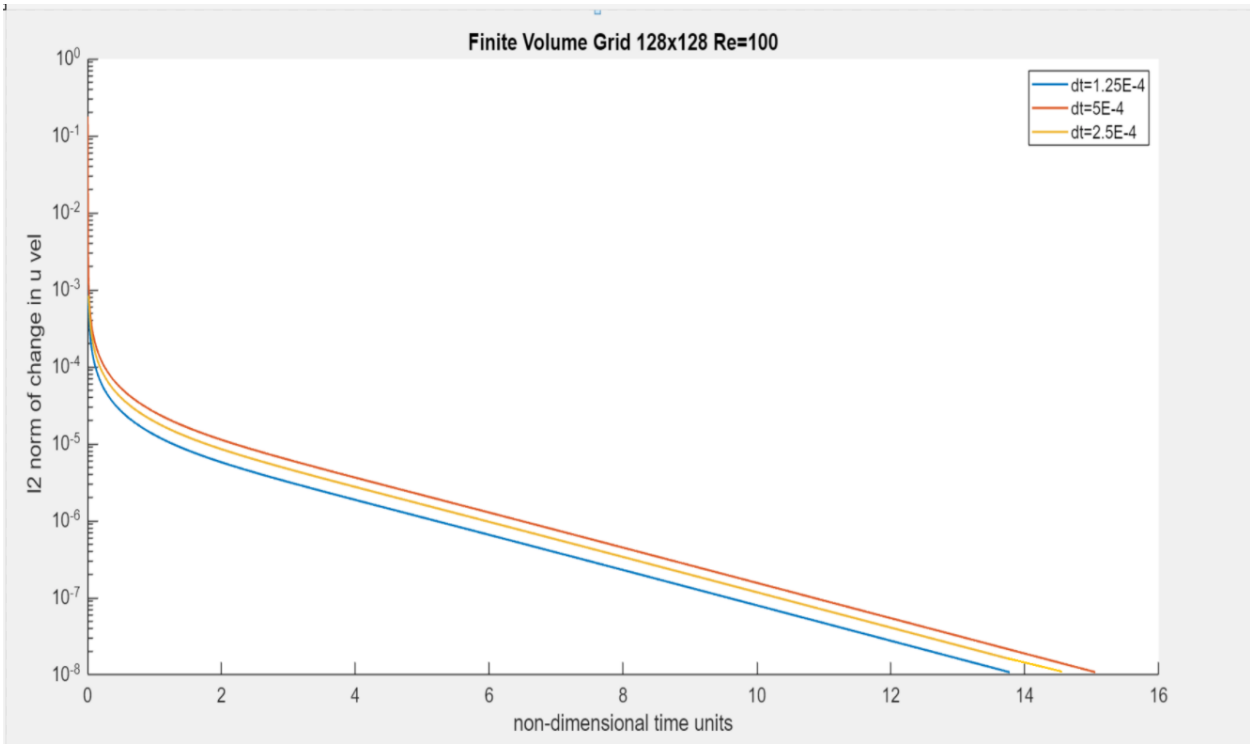

Figure 12. Grid 128x128 L2 Norm of Change in U velocity

1. Velocity Comparison Re=100

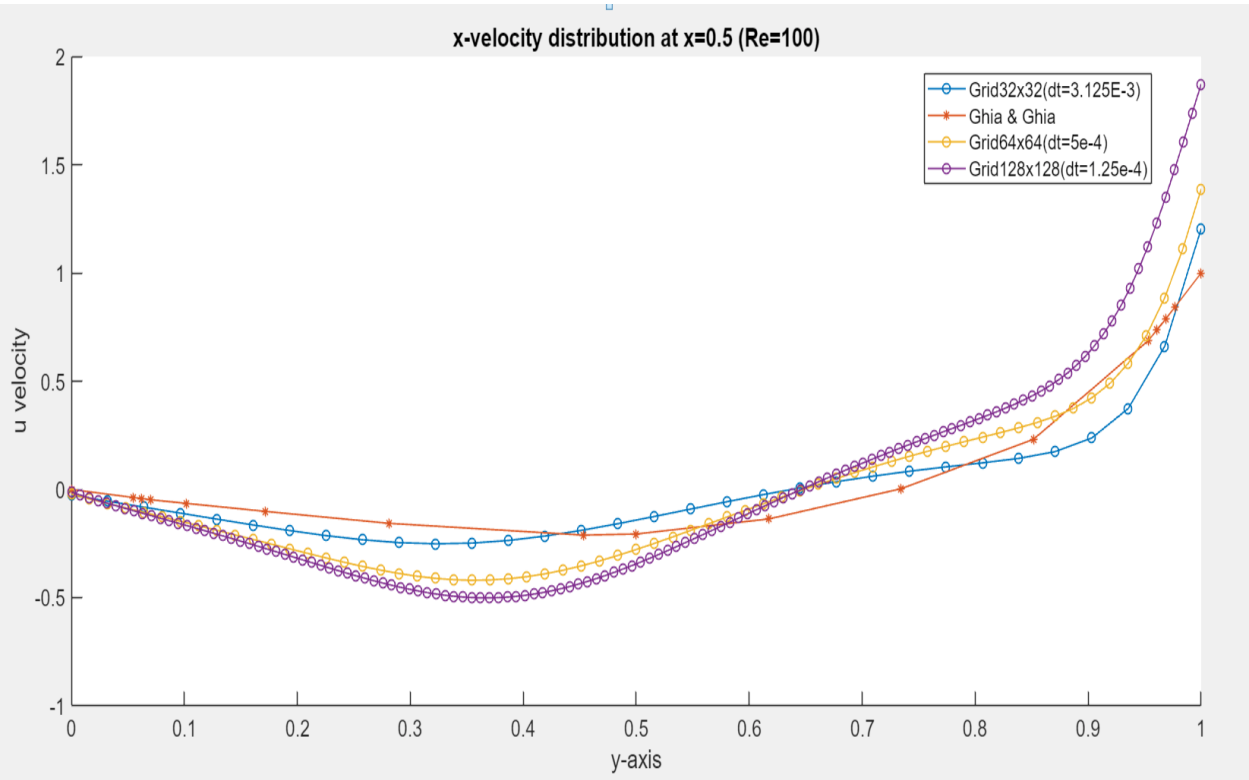

Figure 13. u velocity comparison

U velocity results are comparable to Ghia and Ghia study with some deviations. V-velocity plots show oscillation.

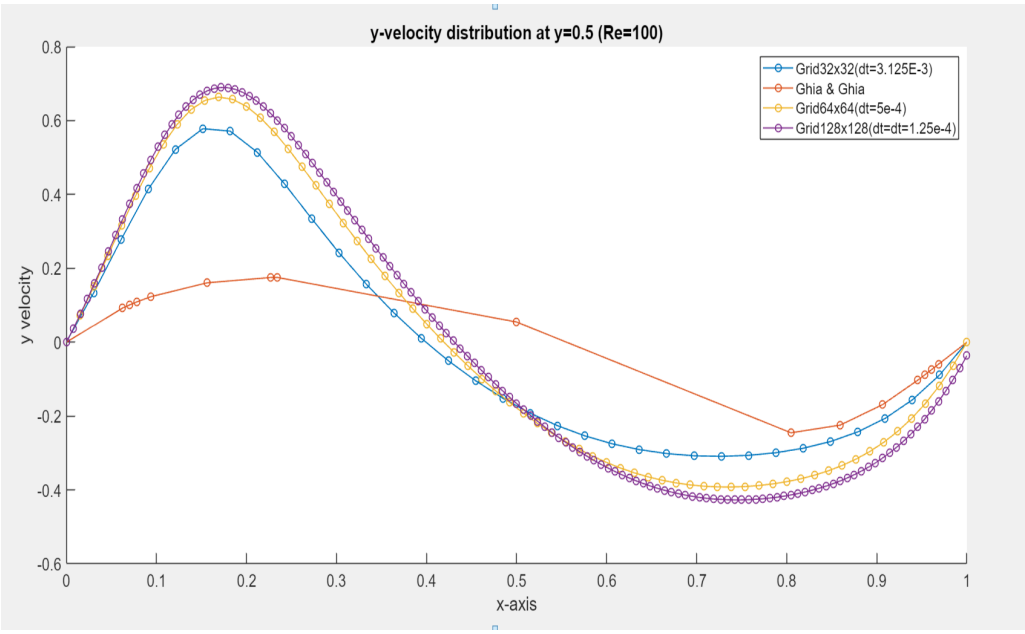

Figure 14. v velocity comparison

2. CPU Time

The rate of increase in CPU time for the finer grid with smaller time step is significantly steeper compared to the other grids. This indicates that finer spatial resolutions (smaller dx) and smaller time steps (dt) significantly increase computational effort. This is because smaller dx and dt values generally lead to better accuracy but require more iterations and more computational resources to resolve the physics adequately. This suggests that while finer grids may provide better detail and potentially more accurate results, they come at a significant cost in terms of CPU time, especially at higher resolutions.

Approximate Power Law

Using matlab power law relationship between grid sixe and CPU time was determined. $y = a * x^b$

where y= CPU time and X is the grid resolution.

$Fit\ for\ CPU\ Time\ Min : y = 6.411e-03 * x^{3.258}$

Fit for CPU Time Intermediate: y = 2.482e-03 * $x^{3.267}$

Fit for CPU Time Max: y = 7.564e-02 * $x^{2.099}$

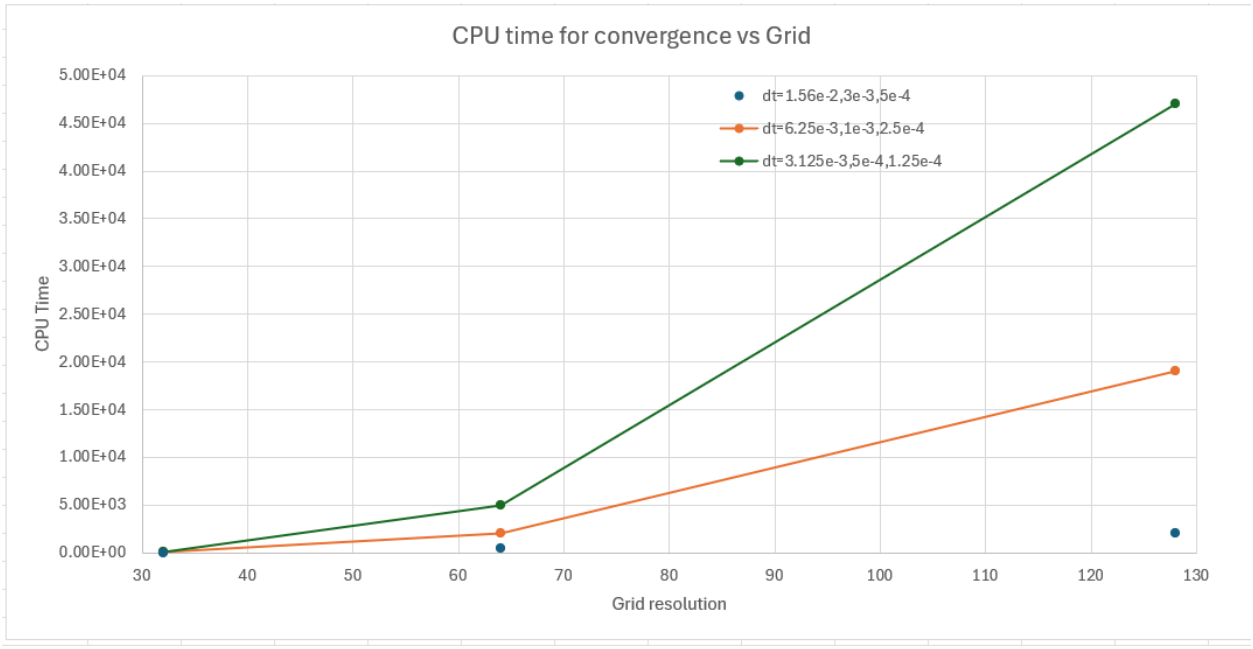

Figure 15. CPU Time

VIII. Part B

1. *Largest Permissible time Grid 128x128*

Table 1. Stability Conditions and Maximum Time Steps

Stability Condition	Max dt (Re=100)	Max dt (Re=1000)
CFL	7.80×10^{-3}	7.80×10^{-3}
2D Neumann	1.53×10^{-3}	1.53×10^{-2}
NonLinear CFL	2.50×10^{-3}	2.50×10^{-3}

$$CFL = CFLNo. * \frac{\delta x}{U_t}$$
$$2DNeumann = \frac{1}{4} * \frac{\delta x^2}{\nu}$$

$$Non - Linear\ CFL : \delta t\ <= 2^{\frac{2}{3}}\ C^{\frac{1}{3}}\ (\frac{\delta x}{U})^{\frac{4}{3}}\ \ [3]$$

Max time step used is $2.5e-3$. In high Reynolds number flows, the convective (or inertial) forces significantly outweigh the viscous forces (diffusive), diffusion generally tend to smooth out the solution but for convection dominated problem, it can take a long time for the system to reach a steady state, therefore non-linear CFL condition is appropriate in such situations.

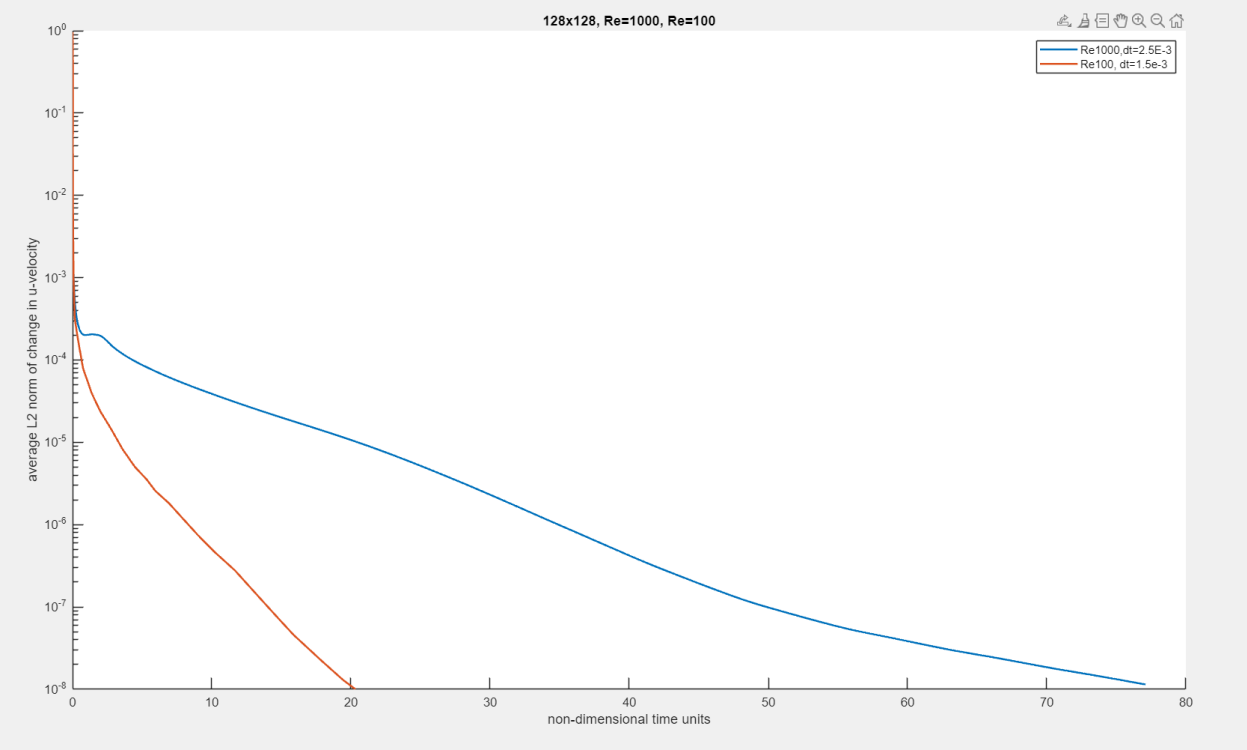

Figure 16. Grid 128x128 Re=100 and Re1000 time comparison

The time required for convergence at Re=1000 is almost 3-4 times the time for Re=100. As Re=1000 is a convection dominant problem, and takes longer based on the stability.

2. Grid 128x128 Re=1000 with dt_{max}

We see a good agreement between the results for 128x128 grid at Re=1000.

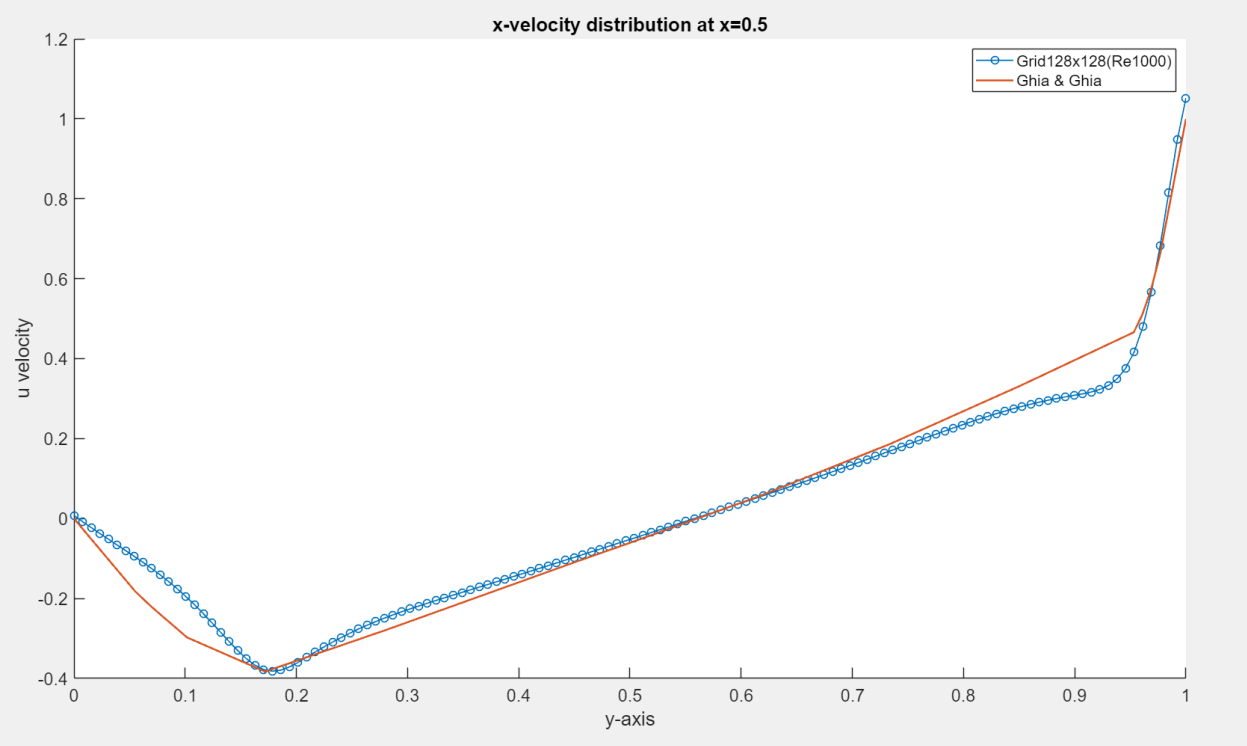

Figure 17. Grid 128x128 Re=1000 comparison

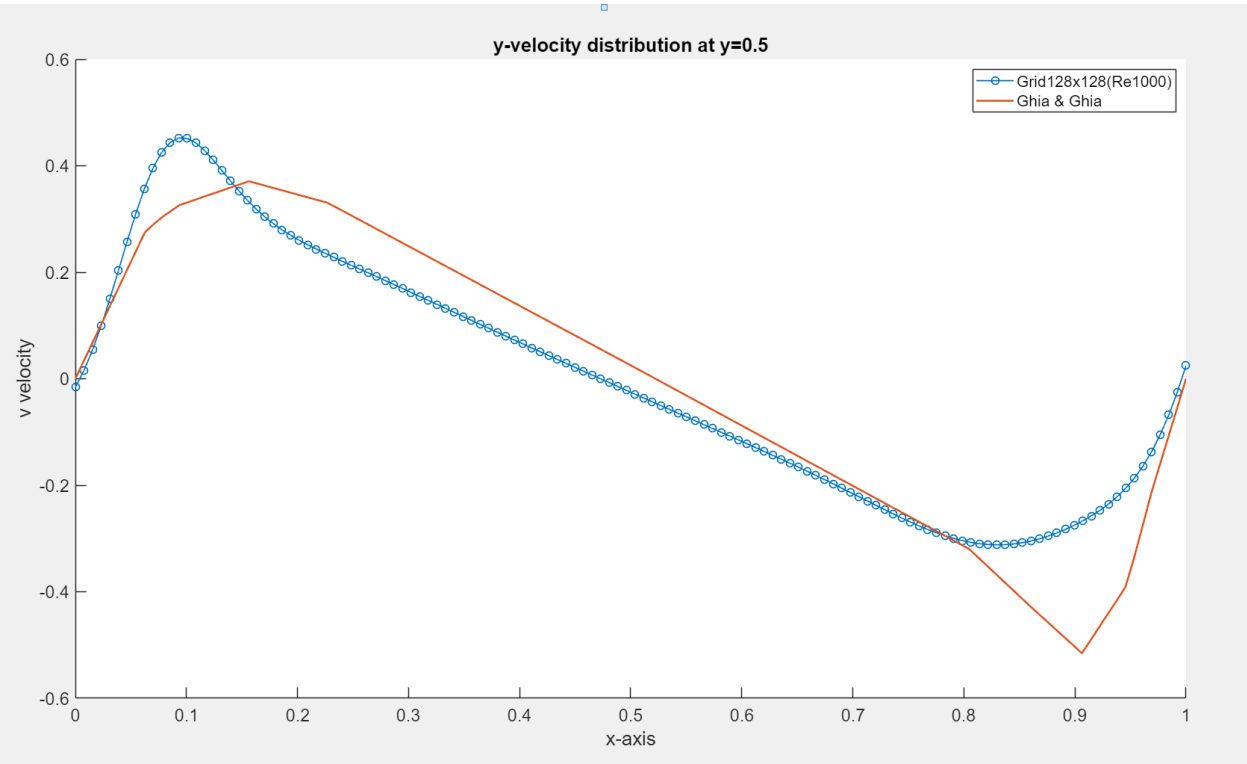

Figure 18. Grid 128x128 Re=1000 v velocity comparison

IX. Part C: Summary and Conclusions

In this excersice to solve the Navier-Stokes equations numerically, it was found that the solution is really influenced by stability conditions beyond the well-known CFL and Neumann criteria. The problem gets worse as the Reynolds number goes up because the viscosity isn't strong enough to stabilize the flow. As a result, it takes longer for the solver to converge when dealing with higher Reynolds numbers. This shows that detailed study of von Neumann stability analysis for specific problems is needed, instead of relying on the usual stability criteria. In this exercise, pressure correction is solved using Successive Over Relaxation (SOR) method, multi grid or more sophisticated methods like conjugate gradient can help accelerate reaching the steady state. We could also use dynamic time stepping to accelerate the solver. In terms of exploiting the hardware, writing the code for parallel computing and solving on ARC cluster could have been a good approach.

References

[1] Erwan Deriaz. "Stability conditions for the numerical solution of convection-dominated problems with skew-symmetric discretizations". In: *SIAM Journal on Numerical Analysis* 50.3 (2012), pp. 1058–1085.

[2] UKNG Ghia, Kirti N Ghia, and CT Shin. "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method". In: *Journal of computational physics* 48.3 (1982), pp. 387–411.

[3] Kai Schneider, Dmitry Kolomenskiy, and Erwan Deriaz. "Is the CFL condition sufficient? Some remarks". In: *The Courant–Friedrichs–Lewy (CFL) Condition: 80 Years After Its Discovery* (2013), pp. 139–146.

Appendix: MATLAB Code

```
%  Lid Driven Cavity Parameters
nx = 128;
ny = 128;
nu = 0.01; %  viscosity
lx = 1;
ly = 1;
dx = lx/nx;
dy = ly/ny;
%% Boundary conditions
uT = 1.0;
uB = 0.0;
vL = 0.0;
vR = 0.0;
% Reynolds no.
Re = uT*lx/nu;
% dt based on linear advection-diffusion
dt_max = min([dx/uT , 0.25*dx*dx/nu]);
%% CFL condition
% dt = dt_max;
dt = 5e-4;%Added
CFL_no = uT*dt/dx;


%% Staggereg Grid

u = zeros(ny+2,nx+2); %preallocating
v = zeros(ny+2,nx+2); %preallocating

%setting boundary conditions
% Set bcs on u velocity grid
    % left wall
    u(2:end-1,2) = 0.0;
    % right wall
    u(2:end-1,end) = 0.0;
    % top wall -ghost cell
    u(1,2:end-1) = 2.0*uT - u(2,2:end-1);
    % bottom wall- ghost cell
    u(end,2:end-1) = 2.0*uB - u(end-1,2:end-1);

% Set bcs on v velocity grid
    % top wall
    v(1,2:end-1) = 0.0;
    % bottom wall
    v(end-1,2:end-1) = 0.0;
    % left wall-ghost cell
    v(1:end-1,1) = 2.0*vL - v(1:end-1,2);
    % right wall-ghost cell
    v(2:end-1,end) = 2.0*vR - v(2:end-1,end-1);

%%
%For calculating L2 error norm
```

```
u_old = u;
v_old = v;

delta_u = zeros(ny+2,nx+2);
delta_v = zeros(ny+2,nx+2);

l2_norm_u = [];
l2_norm_v = [];

error_u = 1;
error_v = 1;
%%

%preallocating intermidate velocity (before pressure correction)
u_tilde = zeros(ny+2,nx+2);
v_tilde = zeros(ny+2,nx+2);

%preallocating convection and diffusion for saving n-1 time step values
%for Adam Bashforth Method
temp_Xcon = zeros(ny+2,nx+2);
temp_Xdiff = zeros(ny+2,nx+2);
temp_Ycon = zeros(ny+2,nx+2);
temp_Ydiff = zeros(ny+2,nx+2);

%preallocating pressure
p = zeros(ny+2,nx+2);

%preallocating convection and diffusion term
con = zeros(1,1);
diff = zeros(1,1);

%% Assemble pressure coefficients using RHS of Poission Equation
%∇2 P =
Ae = 1/dx^2*ones(ny+2,nx+2);
Aw = 1/dx^2*ones(ny+2,nx+2);
An = 1/dy^2*ones(ny+2,nx+2);
As = 1/dy^2*ones(ny+2,nx+2);
% set the left wall coefficients
Aw(2:end-1,2) = 0.0;
% set the right wall coefficients
Ae(2:end-1,end-1) = 0.0;
% set the top wall coefficients
An(2,2:end-1) = 0.0;
% set the bottom wall coefficients
As(end-1,2:end-1) = 0.0;
Ap = -(Aw + Ae + An + As);

%% Time integration
tStart = cputime;
n = 1;
while error_u > 1e-8 && error_v > 1e-8


%% Assemble x-mom first for u_tilde -  interior points
    for i = 3:nx+1
```

```
        for j = 2:ny+1
            ue = 0.5*(u(j,i)+u(j,i+1));
            uw = 0.5*(u(j,i)+u(j,i-1));
            un = 0.5*(u(j,i)+u(j+1,i));
            us = 0.5*(u(j,i)+u(j-1,i));
            vn = 0.5*(v(j+1,i-1)+v(j+1,i));
            vs = 0.5*(v(j,i)+v(j,i-1));


            con(j,i) = -(ue*ue - uw*uw)/dx - (un*vn - us*vs)/dy; %Convection
            diff(j,i) = (1/Re)*((u(j,i+1) - 2*u(j,i) + u(j,i-1))/dx^2 + ...
                        (u(j+1,i) - 2*u(j,i) + u(j-1,i))/dy^2) ; %Diffusion
            %Hi = Convection + Diffusion
            % u_tilde  ≅ u_i^n+ ∫H_i dt
            %∫H_i dt= Δt*[(3/2 * (H_i^n)) - (1/2 * H_i^(n-1))]  Adam Bashforth

            u_tilde(j,i) = u(j,i) + dt*(((3/2)*(con(j,i) + diff(j,i))) -
((1/2)*(temp_Xcon(j,i) + temp_Xdiff(j,i)))); %  Adams Bashforth

            temp_Xcon(j,i) = con(j,i); %saving convection term for (n-1) time
            temp_Xdiff(j,i) = diff(j,i);%saving convection term for (n-1) time
        end
    end

    %% Assemble y-mom first for v_tilde - do interior points only
    for i = 2:nx+1
        for j = 2:ny
            ue = 0.5*(u(j,i+1) + u(j-1,i+1));
            uw = 0.5*(u(j-1,i) + u(j,i));
            ve = 0.5*(v(j,i) + v(j,i+1));
            vw = 0.5*(v(j,i) + v(j,i-1));
            vn = 0.5*(v(j,i) + v(j+1,i));
            vs = 0.5*(v(j,i) + v(j-1,i));

            con(j,i) = -(ve*ue - vw*uw)/dx - (vn*vn - vs*vs)/dy;
            diff(j,i) = (1/Re)*((v(j,i+1) - 2*v(j,i) + v(j,i-1))/dx^2 + ...
                        (v(j+1,i) - 2*v(j,i) + v(j-1,i))/dy^2) ;

            %Hi = Convection + Diffusion
            % u_tilde  ≅ u_i^n+ ∫H_i dt
            %∫H_i dt= Δt*[(3/2 * (H_i^n)) - (1/2 * H_i^(n-1))]  Adam Bashforth

            v_tilde(j,i) = v(j,i) + dt/2*(3*(con(j,i) + diff(j,i)) - (temp_Ycon(j,i)
+ temp_Ydiff(j,i))); % 2nd order Adams Bashforth

            temp_Ycon(j,i) = con(j,i); %saving convection term for (n-1) time
            temp_Ydiff(j,i) = diff(j,i); %saving convection term for (n-1) time
        end
    end

    %Pressure Correction
    %D_i*(G_i  P^(n+1))=  1/Δt  D_i* (u tilde_i ) ~
    %% rhs: prhs = 1/Δt  D_i* (u tilde_i ) =  1/dt * (du_tilde/dx + dv_tilde/dy)
    divut = zeros(ny+2,nx+2);
```

```
    divut(2:end-1,2:end-1) = (u_tilde(2:end-1,3:end) - u_tilde(2:end-1,2:end-1))/dx +
(v_tilde(2:end-1,2:end-1) - v_tilde(1:end-2,2:end-1))/dy;
    prhs = divut/dt;

    %% Solve pressure poisson equation
    p = sor_solver(p,Ae,Aw,An,As,Ap,prhs,nx,ny);

    % Final time advance - do interior only
    % u = ut - dt*dpdx
    u(2:end-1,3:end-1) = u_tilde(2:end-1,3:end-1) - dt*(p(2:end-1,3:end-1)-p(2:end-
1,2:end-2))/dx; % (n+1)th time step
    v(2:end-2,2:end-1) = v_tilde(2:end-2,2:end-1) - dt*(p(3:end-1,2:end-1)-p(2:end-
2,2:end-1))/dy; % (n+1)th time step
    delta_u = u - u_old;
    delta_v = v - v_old;


%Calculating and Appending L2_error_norms
    l2_norm_u = [l2_norm_u,sqrt(sum(delta_u.^2,"all")/(nx*ny))];
    l2_norm_v = [l2_norm_v,sqrt(sum(delta_v.^2,"all")/(nx*ny))];
    error_u = l2_norm_u(end);
    error_v = l2_norm_v(end);
    fprintf('\t%d \t%f \t%f\n',n,l2_norm_u,l2_norm_v);
    n = n+1;

    %Update U_old for new iteration
    u_old = u;
    v_old = v;
end
time = dt*linspace(2,n,n-1);
figure(1)
% xlim([0,80])
% ylim([1e-8,1])
% subplot(2,1,1)
title('Re = 100  on a 32*32 finite volume grid')
semilogy(time,l2_norm_u,'^');
xlabel('non-dimensional time units')
ylabel('average l2 norm of u vel')
hold on

tEnd = cputime - tStart;
%%
%Load Ghia&Ghia data
T = readtable('GhiaData.csv');
x_coord = T.x_coord;
y_coord = T.y_coord;
u_ghia_Re100 = T.u_ghia_Re100;
v_ghia_Re100 = T.v_ghia_Re100;

%%

figure(2)
title('u velocity along vertical line through geometric centre of cavity')
plot(y_coord,u_ghia_Re100)
hold on
```

```
plot(linspace(0,1,66),u(end:-1:1,34),'--')
% % hold on
% plot(linspace(0,1,130),u(end:-1:1,66),'o')
% % hold on
 % plot(linspace(0,1,34),u(end:-1:1,18),'d')

figure(3)
plot(x_coord,v_ghia_Re100)
hold on
plot(linspace(0,1,66),v(33,end:-1:1),'--')


%% Save data
% save('G32Dt1.mat', 'time', 'l2_norm_u', 'u', 'v',
"x_coord","y_coord","u_ghia_Re100","v_ghia_Re100");
%save('G32Dt2.mat', 'time', 'l2_norm_u', 'u', 'v',
"x_coord","y_coord","u_ghia_Re100","v_ghia_Re100");
% save('G32Dt3.mat', 'time', 'l2_norm_u', 'u', 'v',
"x_coord","y_coord","u_ghia_Re100","v_ghia_Re100");

save('G128Dt3.mat', 'time', 'l2_norm_u', 'u', 'v',
"x_coord","y_coord","u_ghia_Re100","v_ghia_Re100","tEnd");

%% SOR solver function
function phi = sor_solver(phi_0,Ae,Aw,An,As,Ap,rhs_vec,nx,ny)
   w = 2/(1+sin(pi/(nx+1)));
   phi = phi_0;
   r = zeros(nx,ny);
   residual = 1;
%    error = 1;
   tol = 1e-5;
%    residue_history = [];
%    error_history = [];
%    phi = zeros(nx+2,ny+2);
   while residual > tol
       for i = 2:nx+1
           for j = 2:ny+1
               phi(j,i) = w*((rhs_vec(j,i) - Ae(j,i)*phi(j,i+1) - Aw(j,i)*phi(j,i-1)
- ...
                   An(j,i)*phi(j+1,i) - As(j,i)*phi(j-1,i))/Ap(j,i)) + (1-
w)*phi(j,i);
           end
       end
       residual = 0;
       for i = 2:nx+1
           for j = 2:ny+1
               r(j,i) = rhs_vec(j,i) - Ae(j,i)*phi(j,i+1) - Aw(j,i)*phi(j,i-1) - ...
                   An(j,i)*phi(j+1,i) - As(j,i)*phi(j-1,i) - Ap(j,i)*phi(j,i);
               residual = residual + r(j,i)*r(j,i);
           end
       end
       residual = sqrt(residual/(nx*ny));

   end
end
```