AM5630: Foundations of Computational Fluid Dynamics

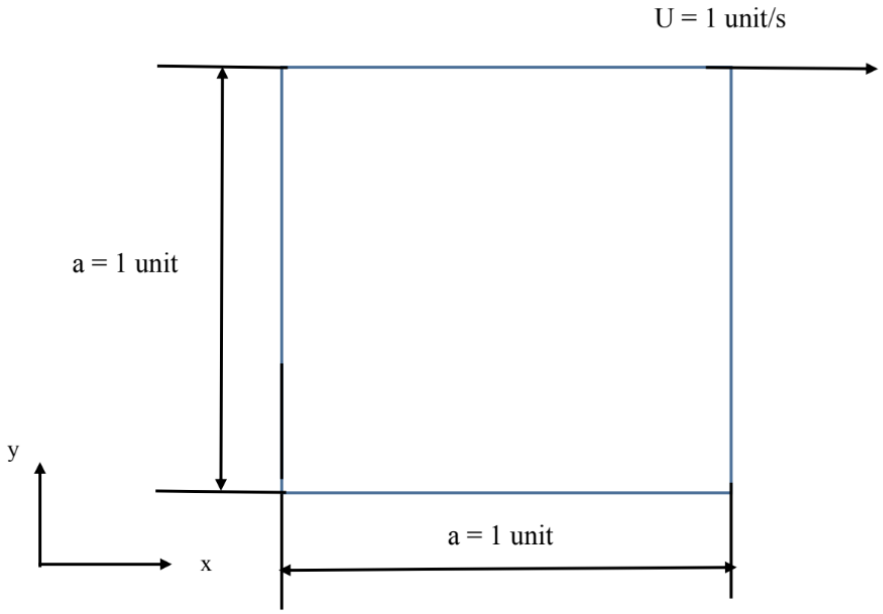
Computer Assignment - 3 (Lid Driven Cavity)

Jaswanth VG, MM21B031

19 November 2024

1 Problem Definition

The lid-driven cavity flow is a classic benchmark problem in computational fluid dynamics (CFD), used to validate numerical methods for incompressible flow. In this assignment, the steady-state flow and pressure distribution inside a 2D square cavity are analyzed at Re = 100, with the lid moving at a constant velocity of 1 unit/s. The vorticity-streamfunction formulation is employed, and results are compared with Ghia et al. (1982) for validation.



2 Governing Equations

Since the magnitudes of the dimensions of the cavity and the reference lid velocity are equal to unity in our case, we shall use the dimensional form of the Navier Stokes equations itself instead of converting them to the non-dimensional form since the variables are inherently normalized, and hence the results should be numerically identical. The 2-D steady state Navier Stokes equation in x and y directions without any source term (No external source in our problem) are:

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

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

The continuity equation is:

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

Since we are dealing with an incompressible fluid, the pressure values cannot be explicitly found. To tackle this issue, we shall use the **Streamfunction-Vorticity formulation**, where two new variables are introduced: the stream function (ψ) and vorticity (ω), both of which are defined in scalar form as follows:

$$u = \frac{\partial \psi}{\partial y}; v = -\frac{\partial \psi}{\partial y} \tag{4}$$

$$\omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{5}$$

The steady-state vorticity-transport equation is:

$$u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right) \tag{6}$$

The Streamfunction Poisson equation is:

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\omega \tag{7}$$

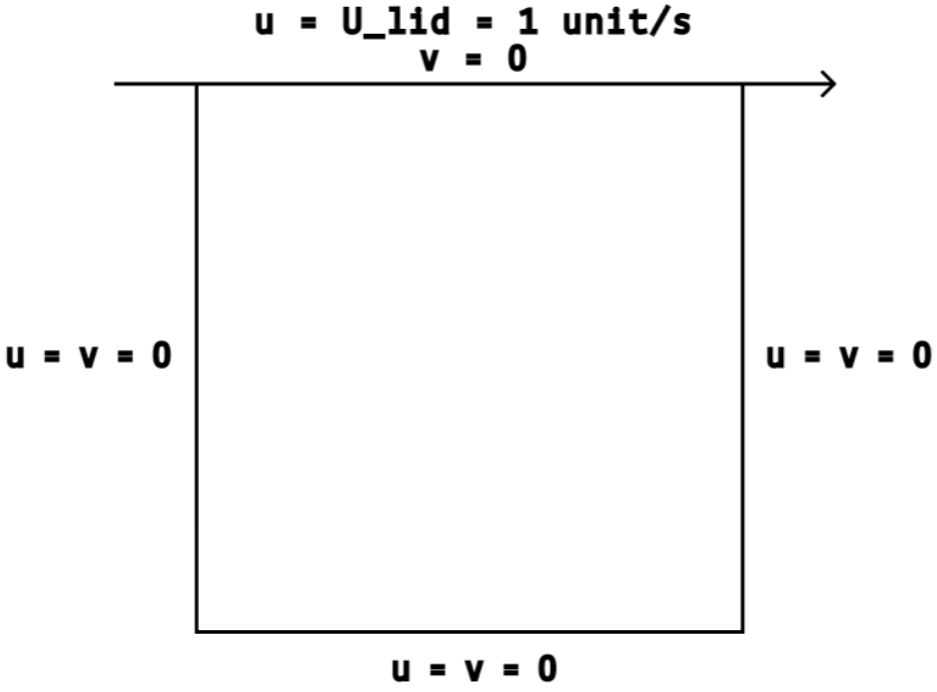
The pressure Poisson equation is:

$$\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} = 2\rho\left[\left(\frac{\partial^2 \psi}{\partial y^2}\right)\left(\frac{\partial^2 \psi}{\partial y^2}\right) - \left(\frac{\partial^2 \psi}{\partial x \partial y}\right)^2\right] \tag{8}$$

3 Boundary Conditions

3.1 u,v Velocities

Left, Bottom and Top walls: u = v = 0
Top wall: u = lid velocity = 1 unit/s, v = 0



3.2 Vorticity

At the left wall, using Taylor series expansion of streamfunction to find the vorticity at the left wall:

$$\psi_{i+1,j} = \psi_{i,j} + \Delta x\frac{\partial \psi}{\partial x} + \frac{(\Delta x)^2}{2!}\frac{\partial^2 \psi}{\partial x^2} + O(\Delta x^2) \tag{9}$$

Since we have u $= \frac{\partial \psi}{\partial y} = 0$ and v $= -\frac{\partial \psi}{\partial x} = 0$ along the left wall, equation (9):

$$\omega = -\left(\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial}{\partial y}\left(\frac{\partial \psi}{\partial y}\right)\right) = -\left(\frac{\partial^2 \psi}{\partial x^2}\right) \tag{10}$$

Hence, using equation (9), we get:

$$\psi_{i+1,j} = \psi_{i,j} - \frac{(\Delta x)^2}{2!}\omega_{i,j}\Big|_{left} \tag{11}$$

$$\omega_{1,j}\Big|_{left} = \frac{-2}{(\Delta x)^2}\left(\psi_{2,j} - \psi_{1,j}\right) \tag{12}$$

Equation (14) is the Boundary condition for vorticity along the left wall. Similarly, we can derive the vorticity boundary conditions for the right, bottom and top walls, and they are as follows:

$$\omega_{imax,j}\Big|_{right} = \frac{2}{(\Delta x)^2}\left(\psi_{imax,j} - \psi_{imax-1,j}\right) \tag{13}$$

$$\omega_{i,1}\Big|_{bottom} = \frac{-2}{(\Delta y)^2}\left(\psi_{i,2} - \psi_{i,1}\right) \tag{14}$$

$$\omega_{i,jmax}\Big|_{top} = \frac{2}{(\Delta y)^2}\left(\psi_{i,jmax} - \psi_{i,jmax-1} - (\Delta y.u\_lid)\right) \tag{15}$$

3.3 Streamfunction

Since we have u $= \frac{\partial \psi}{\partial y} = 0$ along the left and right walls and v $= -\frac{\partial \psi}{\partial x} = 0$ along the top and bottom walls, the stream function can be taken to be a constant along the walls, say equal to 0 in our case. Hence, $\psi = 0$ along all the walls.

3.4 Tabulation of Boundary Conditions

Boundary	**u, v Velocities**	**Vorticity, ω**	**Streamfunction, ψ**
Left Wall	$u = 0, v = 0$	$\omega = -\frac{2(\psi_{2,j} - \psi_{1,j})}{\Delta x^2}$	$\psi = 0$
Right Wall	$u = 0, v = 0$	$\omega = \frac{2(\psi_{imax,j} - \psi_{imax-1,j})}{\Delta x^2}$	$\psi = 0$
Bottom Wall	$u = 0, v = 0$	$\omega = -\frac{2(\psi_{i,2} - \psi_{i,1})}{\Delta y^2}$	$\psi = 0$
Top Wall	$u = 1, v = 0$	$\omega = \frac{2(\psi_{i,jmax} - \psi_{i,jmax-1} - u_{\text{lid}} \cdot \Delta y)}{\Delta y^2}$	$\psi = 0$

Table 1: Boundary conditions for velocity, vorticity, and stream function.

4 Numerical Formulation

All the below discretizations are done for a uniform grid, i.e., $\Delta x = \Delta y$ throughout our domain. We shall use the Finite Difference Method (FDM) to discretize equations (6) and (9) according to the central difference scheme:

$$u = \frac{\psi_{i,j+1} - \psi_{i,j-1}}{2\Delta y}; v = \frac{\psi_{i-1,j} - \psi_{i+1,j}}{2\Delta x} \tag{16}$$

$$\left(\frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{(\Delta x)^2} + \frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{(\Delta y)^2}\right) = -\omega_{i,j} \tag{17}$$

Using the Line Successive Over Relaxation Scheme (LSOR), where β is the relaxation parameter, and taking $\Delta x = \Delta y$, we get:

$$\beta.\psi^{k+1}_{i-1,j} - 4\psi^{k+1}_{i,j} + \beta.\psi^{k+1}_{i,j} = -4(1-\beta)\psi^{k}_{i,j} - \beta(\psi^{k+1}_{i,j-1} + \psi^{k}_{i,j+1} + (\Delta x)^2\omega^{k+1}_{i,j}) \tag{18}$$

The above streamfunction Poisson equation can be solved by the Thomas algorithm (also known as the Tridiagonal Matrix Algorithm (TDMA)).
Now, coming to the vorticity transport equation (8), we shall use the order upwind scheme, which is known to capture the flow physics better than the central difference scheme. According to the First order upwind scheme:

$$u\frac{\partial\omega}{\partial x} = u_{i,j}\left[\frac{\omega_{i,j}-\omega_{i-1,j}}{\Delta x}\right] \qquad u_{i,j} >= 0 \tag{19}$$

$$u\frac{\partial\omega}{\partial x} = u_{i,j}\left[\frac{\omega_{i+1,j}-\omega_{i,j}}{\Delta x}\right] \qquad u_{i,j} < 0 \tag{20}$$

$$v\frac{\partial\omega}{\partial x} = v_{i,j}\left[\frac{\omega_{i,j}-\omega_{i,j-1}}{\Delta x}\right] \qquad v_{i,j} >= 0 \tag{21}$$

$$v\frac{\partial\omega}{\partial x} = v_{i,j}\left[\frac{\omega_{i,j+1}-\omega_{i,j}}{\Delta x}\right] \qquad v_{i,j} < 0 \tag{22}$$

The simplified discretized equations of the vorticity transport equations under each of the four cases of flow direction using the Point Gauss-Seidel method are given below:
When $u_{i,j} >= 0$ and $v_{i,j} >= 0$:

$$\omega_{i,j}^{k+1} = \frac{\left\{\frac{\nu}{\Delta x}\left[w_{i+1,j}^{k} + w_{i-1,j}^{k+1} + w_{i,j+1}^{k} + w_{i,j-1}^{k+1}\right] + (u_{i,j}.\omega_{i-1,j}^{k+1} + v_{i,j}.\omega_{i,j-1}^{k+1})\right\}}{\left[u_{i,j} + v_{i,j} + \frac{4\nu}{\Delta x}\right]} \tag{23}$$

When $u_{i,j} < 0$ and $v_{i,j} >= 0$:

$$\omega_{i,j}^{k+1} = \frac{\left\{\frac{\nu}{\Delta x}\left[w_{i+1,j}^{k} + w_{i-1,j}^{k+1} + w_{i,j+1}^{k} + w_{i,j-1}^{k+1}\right] + (-u_{i,j}.\omega_{i+1,j}^{k+1} + v_{i,j}.\omega_{i,j-1}^{k+1})\right\}}{\left[-u_{i,j} + v_{i,j} + \frac{4\nu}{\Delta x}\right]} \tag{24}$$

When $u_{i,j} >= 0$ and $v_{i,j} < 0$:

$$\omega_{i,j}^{k+1} = \frac{\left\{\frac{\nu}{\Delta x}\left[w_{i+1,j}^{k} + w_{i-1,j}^{k+1} + w_{i,j+1}^{k} + w_{i,j-1}^{k+1}\right] + (u_{i,j}.\omega_{i-1,j}^{k+1} - v_{i,j}.\omega_{i,j+1}^{k+1})\right\}}{\left[u_{i,j} - v_{i,j} + \frac{4\nu}{\Delta x}\right]} \tag{25}$$

When $u_{i,j} < 0$ and $v_{i,j} < 0$:

$$\omega_{i,j}^{k+1} = \frac{\left\{\frac{\nu}{\Delta x}\left[w_{i+1,j}^{k} + w_{i-1,j}^{k+1} + w_{i,j+1}^{k} + w_{i,j-1}^{k+1}\right] + (-u_{i,j}.\omega_{i+1,j}^{k+1} - v_{i,j}.\omega_{i,j+1}^{k+1})\right\}}{\left[-u_{i,j} - v_{i,j} + \frac{4\nu}{\Delta x}\right]} \tag{26}$$

Discretizing the Pressure Poisson equation (10) using the central difference method gives us:

$$\left(\frac{P_{i+1,j}-2P_{i,j}+P_{i-1,j}}{(\Delta x)^2} + \frac{P_{i,j+1}-2P_{i,j}+P_{i,j-1}}{(\Delta y)^2}\right) = \frac{2\rho}{(\Delta x)^2}\left[\left(\frac{\psi_{i+1,j}-2\psi_{i,j}+\psi_{i-1,j}}{(\Delta x)^2}\right)\left(\frac{\psi_{i,j+1}-2\psi_{i,j}+\psi_{i,j-1}}{(\Delta y)^2}\right)\right] - \frac{2\rho}{(\Delta x)^2}\left[\left(\frac{\psi_{i+1,j+1}-\psi_{i-1,j+1}-\psi_{i+1,j-1}-\psi_{i-1,j-1}}{\Delta x\Delta y}\right)^2\right] \tag{27}$$

Simplifying the above equation to use the Point Gauss-Seidel Method gives us:

$$P_{i,j}^{k+1} = \frac{1}{4}\left(P_{i-1,j}^{k+1} + P_{i+1,j}^{k} + P_{i,j-1}^{k+1} + P_{i,j+1}^{k}\right) - \frac{\rho}{2(\Delta x)^2}\left[(\psi_{i+1,j+1}^{k+1} - \psi_{i-1,j+1}^{k+1} - \psi_{i+1,j-1}^{k+1} - \psi_{i-1,j-1}^{k+1})^2\right] - \frac{\rho}{2(\Delta x)^2}\left[(\psi_{i+1,j}^{k+1} - 2\psi_{i,j}^{k+1} + \psi_{i-1,j}^{k+1})(\psi_{i,j+1}^{k+1} - 2\psi_{i,j}^{k+1} + \psi_{i,j-1}^{k+1})\right] \tag{28}$$

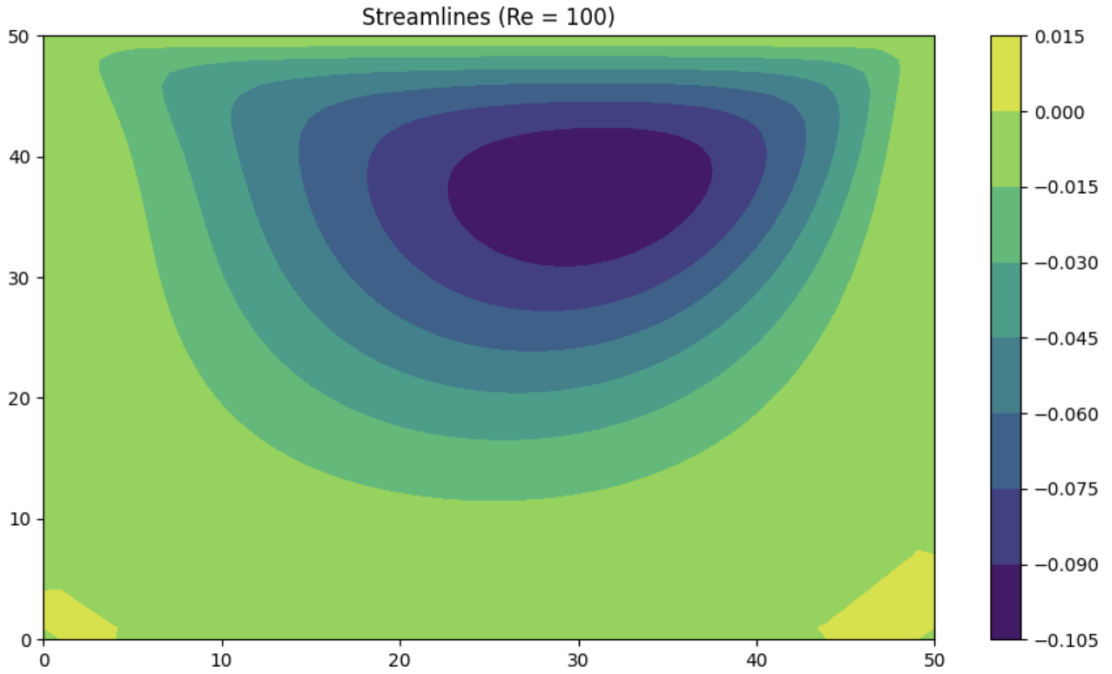
5 Pseudocode

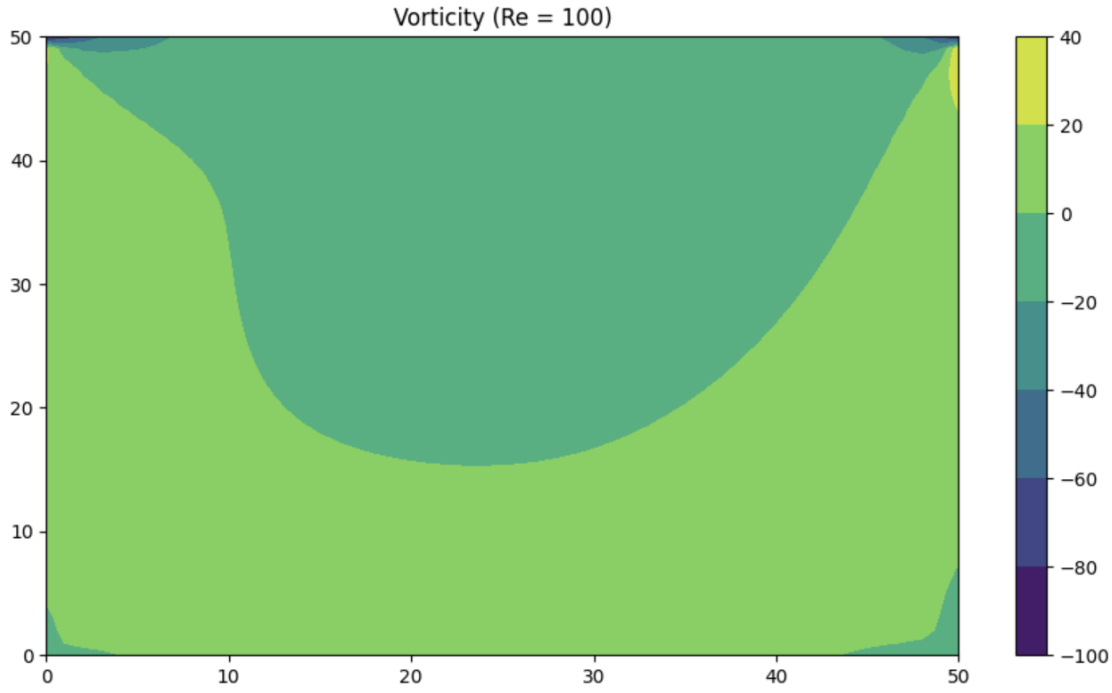
Algorithm 1 Lid-Driven Cavity Flow Simulation Using Vorticity-Streamfunction Method

Initialize Parameters:
$N \leftarrow 51$ ▷ Grid size
$\Delta x \leftarrow \frac{1}{N-1}$, $\Delta y \leftarrow \Delta x$
$Re \leftarrow 100.0$, $U_{\text{lid}} \leftarrow 1.0$
$\mu \leftarrow \frac{U_{\text{lid}} \cdot L}{Re}$ ▷ Dynamic viscosity
$\rho \leftarrow 1.164$ ▷ Density of air at 30°C
Initialize Variables:
$W_{\text{old}}, P_{\text{old}}, \Psi_{\text{old}}, U, V \leftarrow 0$
Set $U[-1, :] \leftarrow 1.0$ ▷ Top lid velocity
$error \leftarrow 1.0$ ▷ Convergence criteria
while $error \geq 10^{-3}$ **do** ▷ Iterate until convergence
  Step 1: Solve Vorticity Transport Equation
  for $i = 1$ to $N-2$ **do**
    for $j = 1$ to $N-2$ **do**
      if $U[i,j] \geq 0$ and $V[i,j] \geq 0$ **then**
        Update $W_{\text{new}}[i,j]$ using upwind scheme for positive U, V
      else if $U[i,j] < 0$ and $V[i,j] \geq 0$ **then**
        Update $W_{\text{new}}[i,j]$ using upwind scheme for negative U
      else if $U[i,j] \geq 0$ and $V[i,j] < 0$ **then**
        Update $W_{\text{new}}[i,j]$ using upwind scheme for negative V
      else
        Update $W_{\text{new}}[i,j]$ using upwind scheme for negative U, V
      end if
    end for
  end for
  Step 2: Solve Poisson Equation for Streamfunction (Ψ)
  for $i = 1$ to $N-2$ **do**
    Initialize a, b, c, d for TDMA solver
    Update source term d using W_{new}
    Solve tridiagonal system using TDMA solver
    Update $\Psi_{\text{new}}[i, :]$
  end for
  Step 3: Compute Velocities (U, V)
  for $i = 1$ to $N-2$ **do**
    for $j = 1$ to $N-2$ **do**
      $U[i,j] \leftarrow \frac{\Psi_{\text{new}}[i+1,j] - \Psi_{\text{new}}[i-1,j]}{2 \cdot \Delta y}$
      $V[i,j] \leftarrow \frac{\Psi_{\text{new}}[i,j-1] - \Psi_{\text{new}}[i,j+1]}{2 \cdot \Delta x}$
    end for
  end for
  Step 4: Apply Boundary Conditions for Vorticity (W_{new})
  $W_{\text{new}}[:, 0] \leftarrow -2.0 \cdot (\Psi_{\text{new}}[:, 1] - \Psi_{\text{new}}[:, 0])/(\Delta x^2)$ ▷ Left wall
  $W_{\text{new}}[:, -1] \leftarrow 2.0 \cdot (\Psi_{\text{new}}[:, -1] - \Psi_{\text{new}}[:, -2])/(\Delta x^2)$ ▷ Right wall
  $W_{\text{new}}[0, :] \leftarrow -2.0 \cdot (\Psi_{\text{new}}[1, :] - \Psi_{\text{new}}[0, :])/(\Delta y^2)$ ▷ Bottom wall
  $W_{\text{new}}[-1, :] \leftarrow \frac{2.0}{\Delta y^2} \cdot (\Psi_{\text{new}}[-1, :] - \Psi_{\text{new}}[-2, :] - U_{\text{lid}} \cdot \Delta y)$ ▷ Top lid
  Step 5: Solve Pressure Poisson Equation
  for $i = 1$ to $N-2$ **do**
    for $j = 1$ to $N-2$ **do**
      Update $P_{\text{new}}[i,j]$ using finite difference approximation
    end for
  end for
  Step 6: Check Convergence
  $error \leftarrow \sum_{i=1}^{N-2} \sum_{j=1}^{N-2} |\Psi_{\text{new}}[i,j] - \Psi_{\text{old}}[i,j]|$
  Step 7: Update Variables
  $W_{\text{old}} \leftarrow W_{\text{new}}, \Psi_{\text{old}} \leftarrow \Psi_{\text{new}}, P_{\text{old}} \leftarrow P_{\text{new}}$
end while
Verify Continuity Equation:
$continuity \leftarrow \sum_{i=1}^{N-2} \sum_{j=1}^{N-2} \left| \frac{U[i,j+1] - U[i,j-1]}{2 \cdot \Delta x} + \frac{V[i+1,j] - V[i-1,j]}{2 \cdot \Delta y} \right|$
Post-Processing:
Plot streamlines, vorticity, pressure, and velocity profiles.

6 Results and Discussion

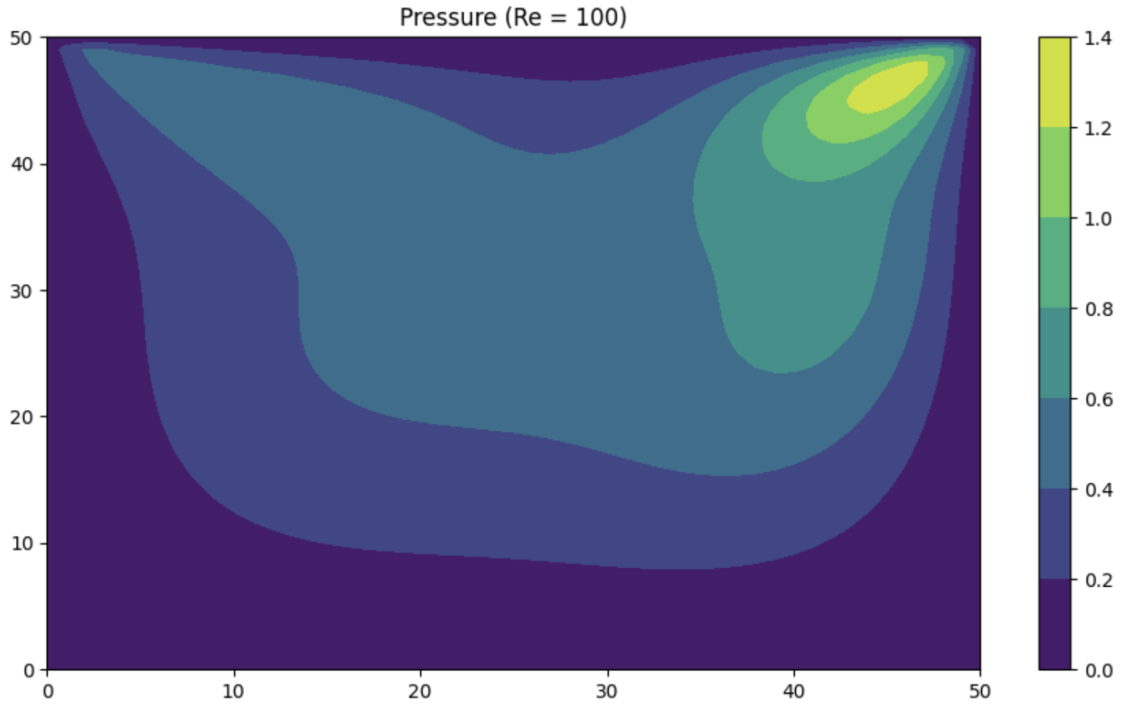
A Python script was created to implement the above algorithm. Square matrices of size NxN are created with rows and columns representing the length of the 2D plate along the X and Y directions, respectively. Upon running the script using a grid size of 0.02 units (N=51), the streamline, vorticity, pressure, u-velocity and v-velocity contours are obtained using the contourf() function in matplotlib, as shown:

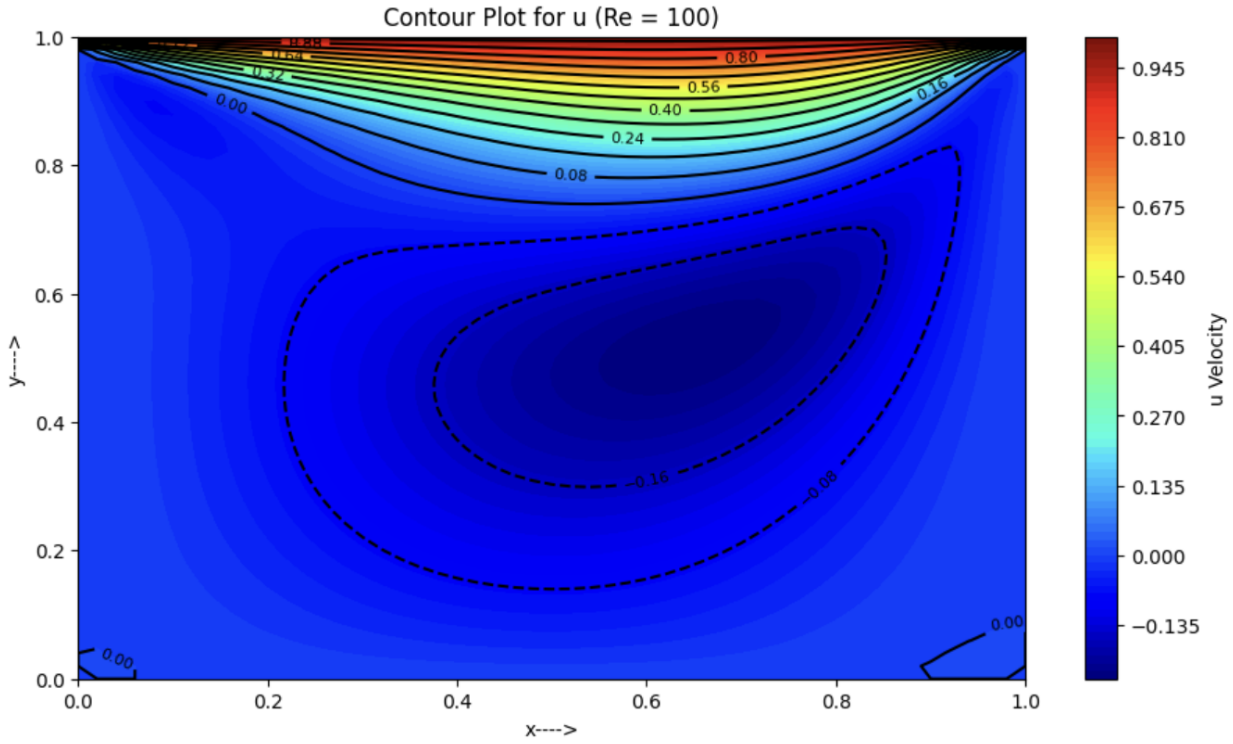




Remarks:

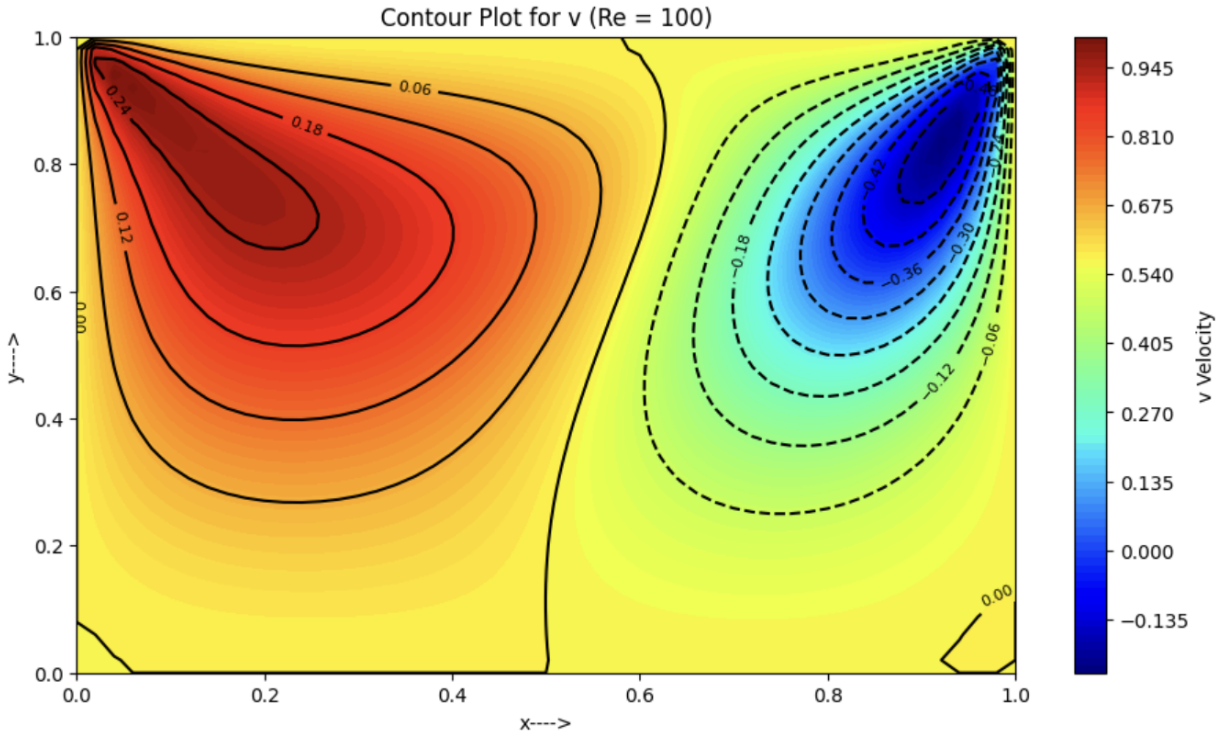
- Streamlines reveal a single primary vortex at Re = 100, consistent with laminar flow characteristics.
- Vorticity is concentrated near the corners of the moving lid due to high shear.





Remarks:

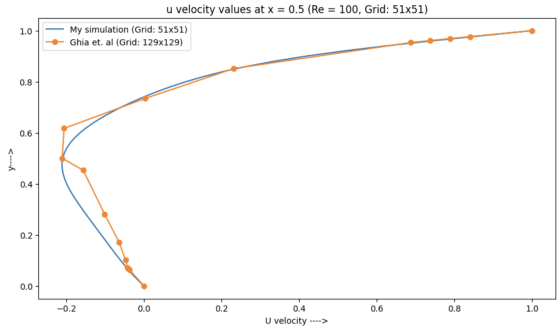
- The pressure contours show symmetry about the vertical centerline, with the high-pressure zone concentrated near the upper-left corner due to lid-driven flow dynamics and a corresponding low-pressure zone near the upper-right corner.
- The u-velocity contours show an increase near the top moving lid and a gradual reduction towards the stationary boundaries, reflecting the transfer of horizontal momentum from the lid to the cavity.



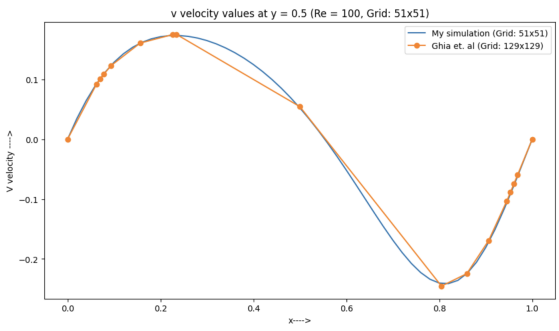
Remarks:

- The v-velocity contours reveal peak values near the upper corners of the cavity due to the lid's induced shear stress and stationary wall constraints. Recirculation zones are observed in the central region of the cavity.

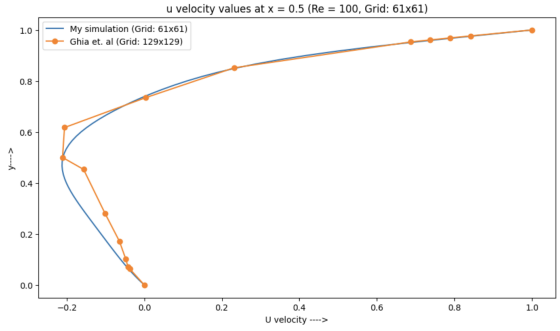
Now let us compare the u-velocity and v-velocity values at their respective geometric centres of x=0.5 units and y=0.5 units, respectively, with the data obtained from Ghia et. al, where the same has been plotted using a 129x129 Grid compared to four different grid sizes that were used individually to get the following plots:



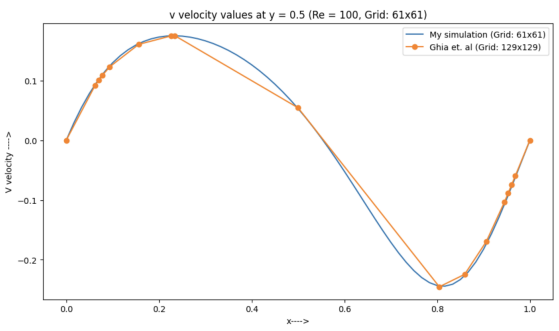
(a) Variation of u-Velocity: 51x51 Grid



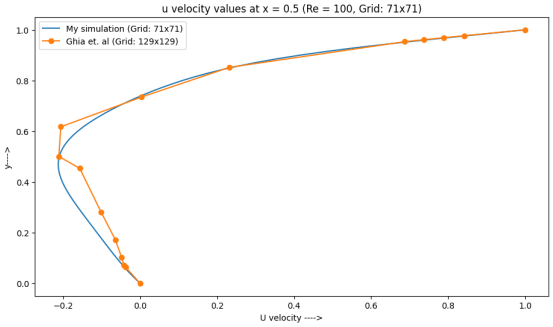
(b) Variation of v-Velocity: 51x51 Grid



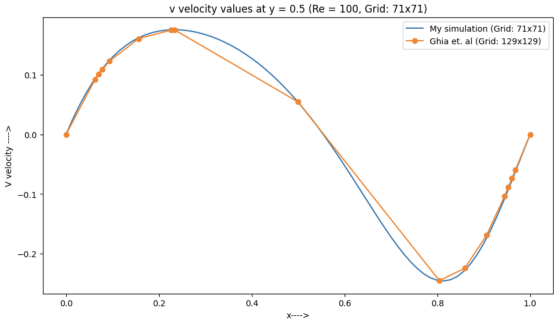
(c) Variation of u-Velocity: 61x61 Grid



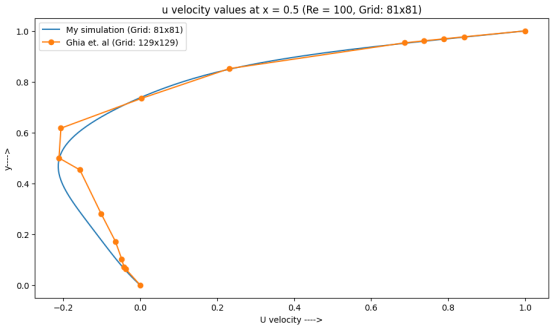
(d) Variation of v-Velocity: 61x61 Grid



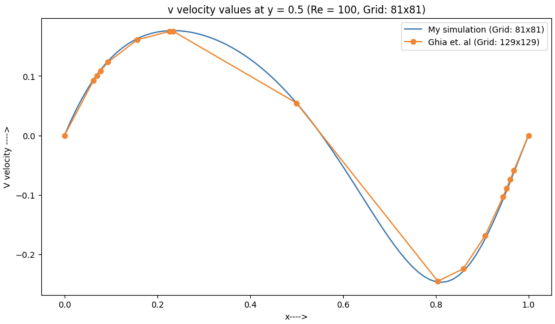
(e) Variation of u-Velocity: 71x71 Grid



(f) Variation of v-Velocity: 71x71 Grid



(g) Variation of u-Velocity: 81x81 Grid



(h) Variation of v-Velocity: 81x81 Grid

The velocity profiles along the vertical and horizontal centerlines show excellent agreement with Ghia et al. (1982) and align well with maximum deviations of less than 5% near the boundary layers, validating the numerical approach employed in this study.

7 Appendix

7.1 Python Script:

```python
import numpy as np
import matplotlib.pyplot as plt

#Define necessaary constants
N = 51 #Gridsize
del_x = 1.0 / (N-1)
del_y = del_x
Re = 100.0 #Given
U_lid = 1.0
L = 1.0
Mu = (U_lid * L) / Re
rho = 1.164 #density of air at 30deg.C

# Function for thomas algorithm
def tdma_solve(a,b,c,d):
    n = len(a)
    bprime = b.copy()
    dprime = d.copy()
    for i in range(1,n+1):
        factor = a[i-1] / bprime[i-1]
        bprime[i] -= factor*c[i-1]
        dprime[i] -= factor*dprime[i-1]

    nx = len(d)
    x = np.zeros(nx)
    x[-1] = dprime[-1] / bprime[-1]
    for i in range(nx-2,-1,-1):
        x[i] = (dprime[i] - c[i]*x[i+1]) / bprime[i]
    return x

#Initialize the variables u,v,P,vorticity(W),Streamfunction(Psi)
W_old = np.zeros([N,N])
W_new = W_old.copy()
P_old = np.zeros([N,N])
P_new = P_old.copy()
Psi_old = np.zeros([N,N])
Psi_new = Psi_old.copy()
U = np.zeros([N,N])
V = np.zeros([N,N])
U[-1, :] = 1.0  #lid velocity
```

```python

error = 1.0 #to get into the loop
while (error >= 1e-3):  #iterate till converged
    #Solve vorticity-transport equation:
    for i in range(1,N-1):
        for j in range(1,N-1):
            #Using first order upwind scheme:
            if(U[i][j] >= 0 and V[i][j] >= 0):
                W_new[i][j] = ((Mu/del_x)*(W_old[i][j+1] + W_new[i][j-1] + W_old[i+1][j] + W_new[i-1][j])
                               + U[i][j]*W_new[i][j-1] + V[i][j]*W_new[i-1][j])/(U[i][j] + V[i][j] + (4*Mu
                                   /del_x))
                continue
            elif(U[i][j] < 0 and V[i][j] >= 0):
                W_new[i][j] = ((Mu/del_x)*(W_old[i][j+1] + W_new[i][j-1] + W_old[i+1][j] + W_new[i-1][j])
                               - U[i][j]*W_new[i][j+1] + V[i][j]*W_new[i-1][j])/(V[i][j] - U[i][j] + (4*Mu
                                   /del_x))
            elif(U[i][j] >= 0 and V[i][j] < 0):
                W_new[i][j] = ((Mu/del_x)*(W_old[i][j+1] + W_new[i][j-1] + W_old[i+1][j] + W_new[i-1][j])
                               + U[i][j]*W_new[i][j-1] - V[i][j]*W_new[i+1][j])/(U[i][j] - V[i][j] + (4*Mu
                                   /del_x))
            elif(U[i][j] < 0 and V[i][j] < 0):
                W_new[i][j] = ((Mu/del_x)*(W_old[i][j+1] + W_new[i][j-1] + W_old[i+1][j] + W_new[i-1][j])
                               - U[i][j]*W_new[i][j+1] - V[i][j]*W_new[i+1][j])/((4*Mu/del_x) -U[i][j] -
                                   V[i][j])

    #Solve Poisson equation
    omega = 1.11 #Over relaxation parameter
    for i in range(1,N-1):
        a = np.ones(N-3) * 1.0 * omega
        b = np.ones(N-2) * -4.0
        c = np.ones(N-3) * 1.0 * omega
        d = np.zeros(N-2)
        for j in range(1,N-1):
            d[j-1] = -4.0*(1-omega)*Psi_old[i][j] - omega*(Psi_old[i+1][j] + Psi_new[i-1][j] + W_new[i][j]
                *(del_x**2))
        new_values = tdma_solve(a,b,c,d)
        Psi_new[i, 1:N-1] = new_values

    #Find U and V using Psi:
    for i in range(1,N-1):
```

```
        for j in range(1,N-1):
            U[i][j] = (Psi_new[i+1][j] - Psi_new[i-1][j])/(2*del_y)
            V[i][j] = (Psi_new[i][j-1] - Psi_new[i][j+1])/(2*del_x)

    #Implement Boundary Conditions for Vorticity
    W_new[:, 0] = -2.0 * (Psi_new[:,1] - Psi_new[:, 0]) / (del_x**2) #Left wall BC
    W_new[:, -1] = 2.0 * (Psi_new[:,-1] - Psi_new[:, -2]) / (del_x**2) #Right wall BC
    W_new[0, :] = -2.0 * (Psi_new[1, :] - Psi_new[0,:]) / (del_y**2) #Bottom wall BC
    W_new[-1, :] = (2.0/(del_y**2)) * (Psi_new[-1,:] - Psi_new[-2,:] - U_lid*del_y) # Top lid BC

    #Solve the pressure poisson equation
    for i in range(1,N-1):
        for j in range(1,N-1):
            P_new[i][j] = 0.25*(P_old[i][j+1] + P_old[i+1][j] + P_new[i-1][j] + P_new[i][j-1])
            - (rho/(2*del_x*del_x))*((Psi_new[i][j+1] - 2*Psi_new[i][j] + Psi_new[i][j-1])*(Psi_new[i+1][j] -2
                *Psi_new[i][j]
            + Psi_new[i-1][j]) - (Psi_new[i+1][j+1] - Psi_new[i+1][j-1] - Psi_new[i-1][j+1] + Psi_new[i-1][j
                -1])**2)

    #Find value of convergence:
    error = 0
    for i in range(1,N-1):
        for j in range(1,N-1):
            error += abs(Psi_new[i][j] - Psi_old[i][j])

    if(error == 0.0): #To bypass the initial error = 0.0 value that we get from initilizing all the variables
        to zero.
        error = 1.0

    W_old = W_new.copy()
    Psi_old = Psi_new.copy()
    P_old = P_new.copy()

#Verify if the continuity equation is satisfied
continuity = 0
for i in range(1,N-1):
    for j in range(1,N-1):
        continuity += abs((U[i][j+1] - U[i][j-1])/(2*del_x) + (V[i+1][j] - V[i-1][j])/(2*del_y))
```

```
print('Continuity error: ',continuity)
#Post-Processing
U_var = []
V_var = []
Grid = []
half = (N-1)//2
for i in range(N):
    U_var.append(U[i][half])
    V_var.append(V[half][i])
    Grid.append(i*del_x)

plt.figure(figsize=(11,6))
plt.contourf(Psi_new)
plt.colorbar()
plt.title('Streamlines (Re = 100)')
plt.show()

plt.figure(figsize=(11,6))
plt.contourf(W_new)
plt.colorbar()
plt.title('Vorticity (Re = 100)')

plt.show()
plt.figure(figsize=(11,6))
plt.contourf(P_new)
plt.colorbar()
plt.title('Pressure (Re = 100)')
plt.show()

plt.figure(figsize=(11,6))
contour_plot_u = plt.contourf(Grid, Grid, U, cmap='jet', levels=100)
plt.colorbar(contour_plot_u, label='u Velocity')
contour_lines_u = plt.contour(Grid, Grid, U, colors='black', levels=15)
plt.clabel(contour_lines_u, inline=1, fontsize=8)
plt.title("Contour Plot for u (Re = 100)")
plt.xlabel("x---->")
plt.ylabel("y---->")

plt.figure(figsize=(11,6))
plt.plot(U_var,Grid)
```

```
plt.xlabel('U velocity ---->')
plt.ylabel('y---->')
plt.title(f'u velocity values at x = 0.5 (Re = 100, Grid: {N}x{N})')
# Data for U-velocity from ghia et. al
y_u = np.array([1.0000, 0.9766, 0.9688, 0.9609, 0.9531, 0.8516, 0.7344, 0.6172, 0.5000,
                0.4531, 0.2813, 0.1719, 0.1016, 0.0703, 0.0625, 0.0000])
u_velocity = np.array([1.00000, 0.84123, 0.78871, 0.73722, 0.68717, 0.23151, 0.00332, -0.20581,
                       -0.21090, -0.15662, -0.10150, -0.06434, -0.04775, -0.04192, -0.03717, 0.00000])
plt.plot(u_velocity,y_u, marker='o')
plt.legend([f'My simulation (Grid: {N}x{N})','Ghia et. al (Grid: 129x129)'])
plt.show()

plt.figure(figsize=(11,6))
contour_plot_v = plt.contourf(Grid, Grid, V, cmap='jet', levels=100)
plt.colorbar(contour_plot_u, label='v Velocity')
contour_lines_v = plt.contour(Grid, Grid, V, colors='black', levels=15)
plt.clabel(contour_lines_v, inline=1, fontsize=8)
plt.title("Contour Plot for v (Re = 100)")
plt.xlabel("x---->")
plt.ylabel("y---->")

plt.figure(figsize=(11,6))
plt.plot(Grid,V_var)
plt.xlabel('x---->')
plt.ylabel('V velocity ---->')
plt.title(f'v velocity values at y = 0.5 (Re = 100, Grid: {N}x{N})')
# Data for V-velocity from Ghia et. al
x_v = np.array([1.0000, 0.9688, 0.9609, 0.9531, 0.9453, 0.9063, 0.8594, 0.8047, 0.5000,
                0.2344, 0.2266, 0.1563, 0.0938, 0.0781, 0.0703, 0.0625, 0.0000])
v_velocity = np.array([0.00000, -0.05906, -0.07391, -0.08864, -0.10313, -0.16914, -0.22445, -0.24533,
                       0.05454, 0.17527, 0.17507, 0.16077, 0.12317, 0.10890, 0.10091, 0.09233, 0.00000])
plt.plot(x_v,v_velocity, marker='o')
plt.legend([f'My simulation (Grid: {N}x{N})','Ghia et. al (Grid: 129x129)'])
plt.show()
```