Assignment 01

ME 670: Advanced Computational Fluid Dynamics

Lid Driven Cavity

Using Finite Volume Method with SIMPLE Algorithm

Nirmal S.
[234103107]
Computational Mechanics
MTech. Mechanical Engineering
IIT-Guwahati

Table of Contents

Problem Statement:

Consider the lid-drive cavity model problem shown in fig. 1. The square cavity is formed by three stationary walls (sides and bottom) and one moving (top) wall of length L. The cavity is very long along the z-direction. It is filled with a Newtonian fluid of density ρ and viscosity μ. The top lid/wall moves with a velocity $(u, v) = (U, 0)\ m/s$. The flow can be assumed two-dimensional, incompressible, steady, and isothermal. Solve the non-dimensional steady Navier-Stokes equations using the finite volume method on a staggered grid and SIMPLE scheme.

Use a 129×129 uniform and Cartesian finite volume grid. Use the Hybrid differencing scheme for solving the momentum equations. For convergence, use the $\|L\infty\|<10-5$ criterion. For the solution of the discretized equations, you can use the point Gauss-Seidel (GS) or ADI method.

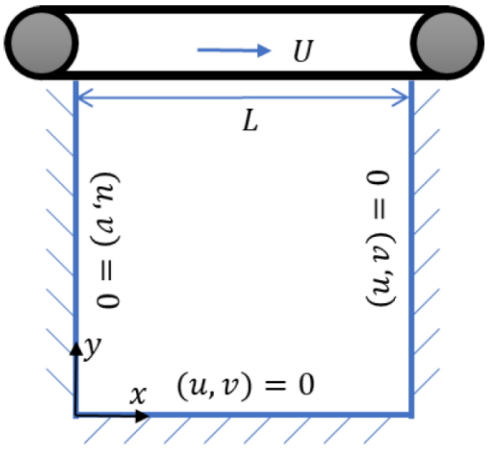


Figure 1: A Schematic of the lid-driven cavity problem

For Re $(=\rho U/L)$ = 100, 400 and 1000

1. Show contours for velocity magnitude and vorticity (ω). Also show streamline pattern by plotting the contours of stream function (ψ).
2. Plot the x-velocity profile at $x=L/2$, and x-velocity profile at $y=L/2$. Compare your results by plotting the values from Tables I and II in the paper by Ghia et al.
3. Compare the following results for the primary vortex given in Table V of Ghia et al. with your results: ψ_{min}, location (x, y) of ψ_{min} and the value of ω at the location of ψ_{min}.

Grid Details

A uniform, cartesian, staggered grid of size 129×129 is used to solve the problem. The control volume of the problem is given by the double red line.

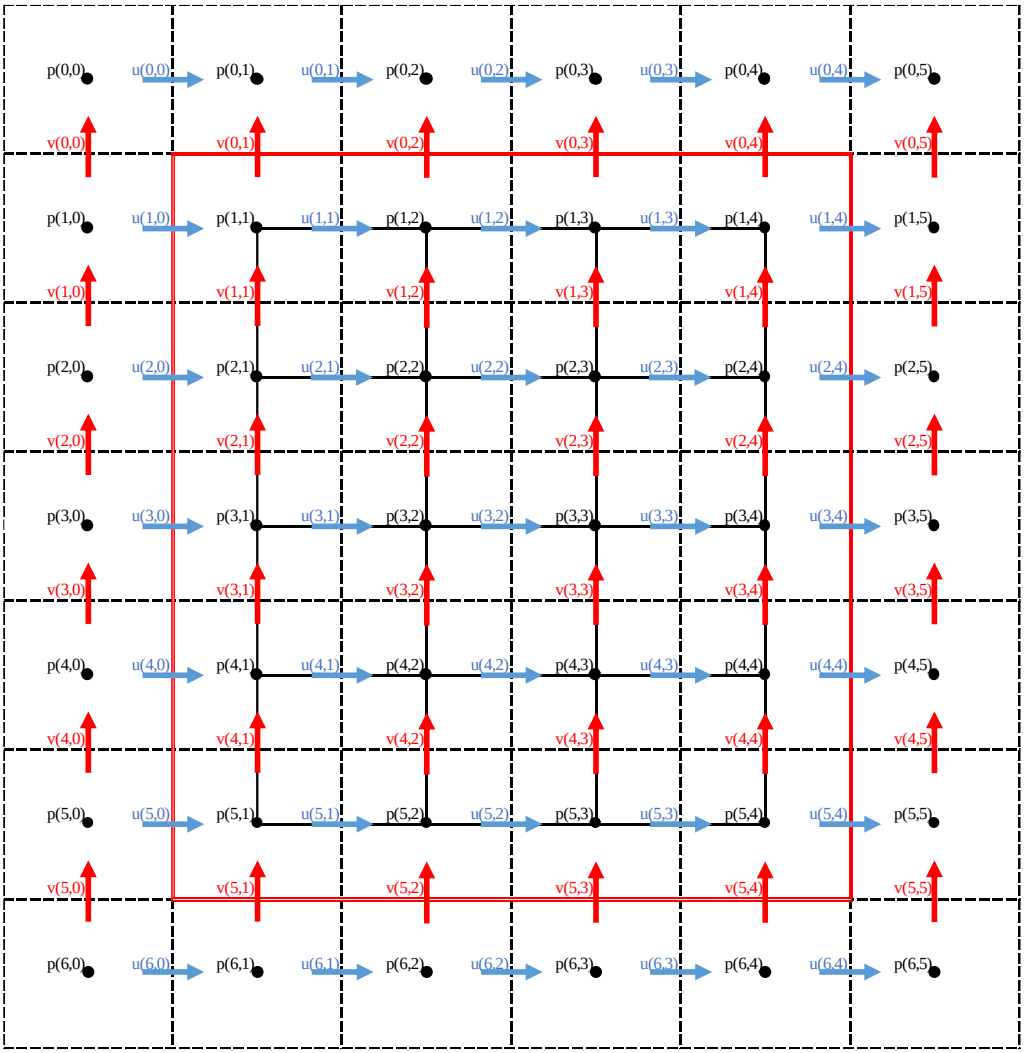


Figure 2: Staggered grid type used in problem. The grid here is shown for a 6 × 5 collocated grid with one extra layer of control volume cell around it.

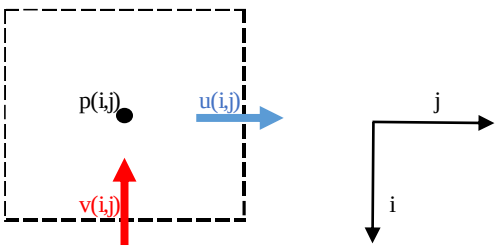


Figure 3: Control volume cell and the (i,j) notation of pressure, u-velocity and v-velocity. The direction of i and j is shown as well.

Discretised Equations

Navier Stokes Equation:

Continuity Equation:

$$\frac{\partial \rho}{\partial t} + \boldsymbol{\nabla} \cdot \rho \mathbf{u} = 0$$

Incompressibility condition:

$$\frac{\partial \rho}{\partial t} + \mathbf{u} \cdot \boldsymbol{\nabla} \rho = 0 \quad \text{or} \quad \boldsymbol{\nabla} \cdot \mathbf{u} = 0$$

Momentum Equation:

$$\frac{\partial \mathbf{u}}{\partial t} + \boldsymbol{\nabla} \cdot \mathbf{u}\mathbf{u} = -\frac{1}{\rho}\boldsymbol{\nabla} \mathrm{p} + \nu \nabla^2 \mathbf{u} + \boldsymbol{g}$$

Here, $\rho =$ *density of fluid*
$\mathbf{u} =$ velocity vector of fluid
$\boldsymbol{g} =$ gravity vector

Getting the momentum equation to non-dimensional form:

$$\frac{U^2}{L}\left[\frac{\partial\left(\frac{\mathbf{u}}{U}\right)}{\partial\left(\frac{t}{L/U}\right)} + L\boldsymbol{\nabla} \cdot \frac{\mathbf{u}}{U}\frac{\mathbf{u}}{U}\right] = \frac{U^2}{L}\left[-L\boldsymbol{\nabla}\frac{\mathrm{p}}{\rho U^2} + \frac{\nu}{\mathrm{UL}}(L\nabla)^2\frac{\mathbf{u}}{U} + \frac{\boldsymbol{g}L}{U^2}\right]$$

$$\frac{\partial \mathbf{u}^*}{\partial t^*} + \boldsymbol{\nabla}^* \cdot \mathbf{u}^*\mathbf{u}^* = -\boldsymbol{\nabla}^* \mathrm{p}^* + \frac{1}{Re}\nabla^{*2}\mathbf{u}^* + \frac{1}{Fr^2}\hat{\boldsymbol{g}}$$

Here, $x^* = \frac{x}{L}$, $\boldsymbol{\nabla}^* = L\boldsymbol{\nabla}$, $\mathbf{u}^* = \frac{\mathbf{u}}{\boldsymbol{U}}$, $t^* = \frac{t}{L/U}$,
$p^* = \frac{\mathrm{p}}{\rho U^2}, Fr = \sqrt{\frac{U^2}{gL}}, Re = \frac{UL}{\nu}$

In indicial notation it can be simplified to:

$$\frac{\partial \mathbf{u}_i^*}{\partial t^*} + \frac{\partial}{\partial x_j^*}\left(\mathbf{u}_i^*\mathbf{u}_j^*\right) = -\frac{\partial p^*}{\partial x_i^*} + \frac{1}{Re}\frac{\partial^2 \mathbf{u}_i^*}{\partial x_i^* \partial x_j^*} - \frac{1}{Fr^2}\widehat{\boldsymbol{g}_i}$$

Considering ***steady state*** and ***without influence of gravity***, the equation becomes:

$$\frac{\partial}{\partial x_j^*}\left(\mathbf{u}_i^*\mathbf{u}_j^*\right) = -\frac{\partial p^*}{\partial x_i^*} + \frac{1}{Re}\frac{\partial^2 \mathbf{u}_i^*}{\partial x_i^* \partial x_j^*}$$

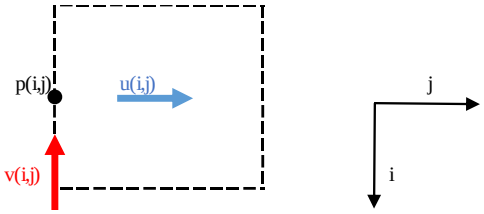
For **simplicity**, let us rewrite [ ]* as [ ] making the above equation as:

$$\frac{\partial}{\partial x_j}\left(\mathbf{u}_i\mathbf{u}_j\right) = -\frac{\partial p}{\partial x_i} + \frac{1}{Re}\frac{\partial^2 \mathbf{u}_i}{\partial x_i \partial x_j}$$

For the finite volume we consider, the above equation can be integrated as

$$\iiint_V \boldsymbol{\nabla} \cdot \mathbf{u}\mathbf{u}\, dV = -\iiint_V \boldsymbol{\nabla} \mathrm{p}\, dV + \frac{1}{Re}\iiint_V \nabla^2 \mathbf{u}\, dV$$

X-Momentum Equation in 2D for u-control volume:



Integrating each term,

$$\int_s^n \int_w^e u \frac{\partial u}{\partial x} dx\, dy = F_e u_e - F_w u_w$$

$$\int_s^n \int_w^e v \frac{\partial u}{\partial y} dx\, dy = F_n u_n - F_s u_s$$

$$\int_s^n \int_w^e -\frac{\partial p}{\partial x} dx\, dy = -(p_e - p_w) A_e$$

$$\int_s^n \int_w^e \frac{1}{Re} \frac{\partial^2 u}{\partial x^2}\, dx\, dy = \frac{1}{Re}\left(\frac{\partial u}{\partial x}|_e - \frac{\partial u}{\partial x}|_w\right) A_e$$

$$\int_s^n \int_w^e \frac{1}{Re} \frac{\partial^2 u}{\partial y^2}\, dx\, dy = \frac{1}{Re}\left(\frac{\partial u}{\partial y}|_n - \frac{\partial u}{\partial y}|_s\right) A_n$$

Where:

$$F_e = \left[\frac{u_i^{j+1} + u_i^j}{2}\right] \Delta y \times 1; \quad F_w = \left[\frac{u_i^j + u_i^{j-1}}{2}\right] \Delta y \times 1$$

$$F_n = \left[\frac{v_{i-1}^{j+1} + v_{i-1}^j}{2}\right] \Delta x \times 1; \quad F_s = \left[\frac{v_i^{j+1} + v_i^j}{2}\right] \Delta x \times 1$$

$$D_e = D_w = \frac{1}{Re} \frac{\Delta y \times 1}{\Delta x}; \quad D_n = D_s = \frac{1}{Re} \frac{\Delta x \times 1}{\Delta y}$$

We get the x-momentum equation in form of:

$$a_P u_P = \sum_{nb} a_{nb} u_{nb} - (p_e - p_w) A_e$$

From Hybrid Scheme, the coefficients are calculated as:

$$a_E = \max\left[-F_e, D_e - \frac{F_e}{2}, o\right]; \quad a_W = \max\left[F_w, D_w + \frac{F_w}{2}, o\right];$$

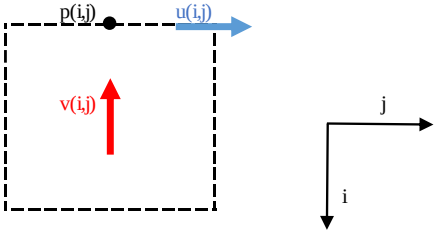
$$a_N = \max\left[-F_n, D_n - \frac{F_n}{2}, o\right]; \quad a_S = \max\left[F_s, D_s + \frac{F_s}{2}, o\right];$$

$$a_P = a_E + a_W + a_N + a_S + (F_e - F_w + F_n - F_s);$$

$$A_e = \Delta y \times 1; \quad A_n = \Delta x \times 1$$

$$p_e = p_i^{j+1}; \quad p_w = p_i^j$$

Y-Momentum Equation in 2D for v-control volume:



Integrating each term,

$$\int_s^n \int_w^e u \frac{\partial v}{\partial x} dx\, dy = F_e v_e - F_w v_w$$

$$\int_s^n \int_w^e v \frac{\partial v}{\partial y} dx\, dy = F_n v_n - F_s v_s$$

$$\int_s^n \int_w^e -\frac{\partial p}{\partial y} dx\, dy = -(p_n - p_s) A_n$$

$$\int_s^n \int_w^e \frac{1}{Re} \frac{\partial^2 v}{\partial x^2}\, dx\, dy = \frac{1}{Re}\left(\frac{\partial v}{\partial x}|_e - \frac{\partial v}{\partial x}|_w\right) A_e$$

$$\int_s^n \int_w^e \frac{1}{Re} \frac{\partial^2 v}{\partial y^2}\, dx\, dy = \frac{1}{Re}\left(\frac{\partial v}{\partial y}|_n - \frac{\partial v}{\partial y}|_s\right) A_n$$

Where:

$$F_e = \left[\frac{u_{i+1}^j + u_i^j}{2}\right] \Delta y \times 1; \quad F_w = \left[\frac{u_{i+1}^{j-1} + u_i^{j-1}}{2}\right] \Delta y \times 1$$

$$F_n = \left[\frac{v_i^j + v_{i-1}^j}{2}\right] \Delta x \times 1; \quad F_s = \left[\frac{v_{i+1}^j + v_i^j}{2}\right] \Delta x \times 1$$

$$D_e = D_w = \frac{1}{Re} \frac{\Delta y \times 1}{\Delta x}; \quad D_n = D_s = \frac{1}{Re} \frac{\Delta x \times 1}{\Delta y}$$

We get the y-momentum equation in form of:

$$a_P v_P = \sum_{nb} a_{nb} v_{nb} - (p_n - p_s) A_n$$

From Hybrid Scheme, the coefficients are calculated as:

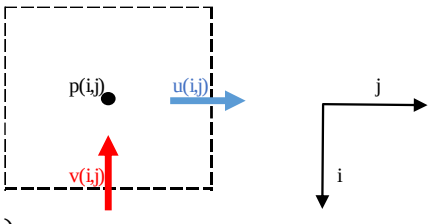
$$a_E = \max\left[-F_e, D_e - \frac{F_e}{2}, o\right]; \quad a_W = \max\left[F_w, D_w + \frac{F_w}{2}, o\right];$$

$$a_N = \max\left[-F_n, D_n - \frac{F_n}{2}, o\right]; \quad a_S = \max\left[F_s, D_s + \frac{F_s}{2}, o\right];$$

$$a_P = a_E + a_W + a_N + a_S + (F_e - F_w + F_n - F_s);$$

$$A_e = \Delta y \times 1 \quad A_n = \Delta x \times 1$$

Continuity Equation in 2D for p-control volume:



$$\nabla \cdot \mathbf{u} = 0$$

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

$$\int_s^n \int_w^e \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) dx\, dy = (u_e A_e - u_w A_w) + (v_n A_n - v_s A_s)$$

Where:

$$u_e = u_i^j; \quad u_w = u_i^{j-1}; \quad v_n = v_{i-1}^j; \quad v_s = v_i^j$$

$$A_e = \Delta y \times 1; \quad A_n = \Delta x \times 1$$

C-Code: SIMPLE Algorithm and Boundary Conditions

For simplicity, let us consider a grid of 5×4 (as shows in *Figure 2*) to show the working of the code. A layer of control volume cell is considered outside the boundary to solve the equation.

Grid Creation:

```
// *** Calculations and 2D array creation ***
double dx = x_length/x_no_divisions; // Division length
double dy = y_length/y_no_divisions;
int n = x_no_divisions + 1;  // No. of points
int m = y_no_divisions + 1;
```

For this example, we take `x_no_divisions = 4` and `y_no_divisions = 5`. Hence the values for `m` and `n` becomes: `m = 6; n = 5.`

We define the final collocated grid with the dimension as required $m \times n = 6 \times 5$ here.
The u-velocity staggered grids are defined for $(m+1) \times n = 7 \times 5$ here; from $u_{i=0}^{j=0} \rightarrow u_6^4$
The v-velocity staggered grids are defined for $m \times (n+1) = 6 \times 6$ here; from $v_{i=0}^{j=0} \rightarrow v_5^5$
The pressure staggered grids are defined for $(m+1) \times (n+1) = 7 \times 6$ here; from $p_{i=0}^{j=0} \rightarrow p_6^5$

```
// Final Collocated Variables
double u_final[m][n], v_final[m][n], p_final[m][n], stream_final[m][n],
       vorticity_final[m][n];

// Staggered Grid
double u[m+1][n], u_star[m+1][n], d_e[m+1][n],
       v[m][n+1], v_star[m][n+1], d_n[m][n+1],
       p[m+1][n+1], p_star[m+1][n+1],
       pc[m+1][n+1], b[m+1][n+1];
```

`u_final[m][n]:` Final u values on collocated grid
`u[m+1][n]:` u values on staggered grid
`u_star[m+1][n]:` Intermediate values of u during SIMPLE algorithm
`d_e[m+1][n]:` Value of A_e/a_P stored for each point
Similar variables for v_final, v, v_star and d_n[m][n+1]
`p_final[m][n]:` Final p values on collocated grid
`p[m+1][n+1]:` p values on staggered grid
`p_star[m+1][n+1]:` Intermediate values of p during SIMPLE algorithm
`p_c[m+1][n+1]:` Pressure Correction values
`b[m+1][n+1]:` To store the continuity equation error at each point

```
// *** Initialisation ***
initialise_Values(m, n, u, u_star, d_e, v, v_star, d_n, p, p_star,
                  pc, b, u_initialise, v_initialise, p_initialise);
```

Initialise all values of the grid to a predetermined value based on the input provided.
Ex: Here we have kept all values to be initialised as zero.

Boundary Conditions:

Boundary condition of u-velocity:

```
void u_Boundary_Conditions(int m, int n, double u[m+1][n], double u_left_value, double u_right_value,
                           double u_top_value, double u_bottom_value){
    for(int i=0; i<m+1; i++){
             u[i][0] = u_left_value;
             u[i][n-1] = u_right_value;
             }
    for(int j=0; j<n; j++){
             u[0][j] = u_top_value*2 - u[1][j];
             u[m][j] = u_bottom_value*2 - u[m-1][j];
             }
    return;
    }
```

Since the left and right most values of staggered u lie on the actual control volume boundary, we can maintain the u values of left and right as:

$$u_i^0 = u_{left}, \quad u_i^{n-1} = u_{right} \quad \forall\, i \in [0, m];$$

For the top and bottom, since the control volume boundary lie between two of the points, we consider the average of the two as the boundary value:

$$\frac{(u_0^j + u_1^j)}{2} = u_{top} \quad \Rightarrow u_0^j = 2 \times u_{top} - u_1^j \quad \forall\, j \in [0, n-1];$$

$$\frac{(u_m^j + u_{m-1}^j)}{2} = u_{bottom} \quad \Rightarrow u_m^j = 2 \times u_{bottom} - u_{m-1}^j \quad \forall\, j \in [0, n-1];$$

Boundary condition of v-velocity:

```
void v_Boundary_Conditions(int m, int n, double v[m][n+1], double v_left_value, double v_right_value,
                           double v_top_value, double v_bottom_value){
    for(int i=0; i<m; i++){
             v[i][0] = v_left_value*2 - v[i][1];
             v[i][n] = v_right_value*2 - v[i][n-1];
    }
    for(int j=0; j<n+1; j++){
             v[0][j] = v_top_value;
             v[m-1][j] = v_bottom_value;
    }
    return;
}
```

Since the top and bottom most values of staggered v lie on the actual control volume boundary, we can maintain the v values of top and bottom as:

$$v_0^j = v_{top}, \quad v_{m-1}^j = v_{bottom} \quad \forall\, j \in [0, n];$$

For the top and bottom, since the control volume boundary lie between two of the points, we consider the average of the two as the boundary value:

$$\frac{(v_i^0 + v_i^1)}{2} = v_{left} \quad \Rightarrow v_i^0 = 2 \times v_{left} - v_i^1 \quad \forall\, i \in [0, m-1];$$

$$\frac{(v_i^n + v_i^{n-1})}{2} = v_{right} \quad \Rightarrow v_i^n = 2 \times v_{right} - v_i^{n-1} \quad \forall\, i \in [0, m-1];$$

Boundary condition of pressure:

```
void p_Boundary_Conditions(int m, int n, double p[m+1][n+1]){
    for(int i=0; i<m+1; i++){
              p[i][0] = p[i][1];
              p[i][n] = p[i][n-1];
    }
    for(int j=0; j<n+1; j++){
              p[0][j] = p[1][j];
              p[m][j] = p[m-1][j];
    }
    return;
}
```

For pressure, we consider the first derivative of pressure along x-axis is zero on left and right boundaries and first derivate of it along y-axis is zero on the top and bottom.

$$p_i^0 = p_i^1, \quad p_i^n = p_i^{n-1} \quad \forall\, i \,\epsilon\, [0, m];$$

$$p_0^j = p_1^j, \quad p_m^j = p_{m-1}^j \quad \forall\, j \,\epsilon\, [0, n];$$

SIMPLE Algorithm:

The algorithm is applied until the L_2 norm of error in u-velocity and v-velocity is less than $\varepsilon = 10^{-5}$

X - Momentum Equation: on the u-control volume cell

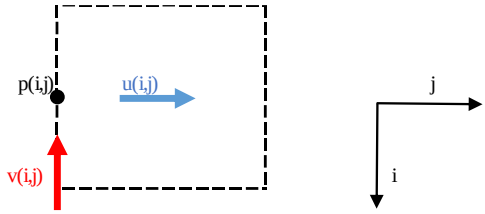
```
// 1.1 X-Momentum Equation Interior:
for(int i=1; i<m; i++){
    for(int j=1; j<n-1; j++){
              Fe = (u[i][j+1] + u[i][j])/2.0*dy*1;
              Fw = (u[i][j] + u[i][j-1])/2.0*dy*1;
              Fn = (v[i-1][j+1] + v[i-1][j])/2.0*dx*1;
              Fs = (v[i][j+1] + v[i][j])/2.0*dx*1;

              De = (1/Re)*(dy*1/dx);
              Dw = (1/Re)*(dy*1/dx);
              Dn = (1/Re)*(dx*1/dy);
              Ds = (1/Re)*(dx*1/dy);

              aE = max(-Fe, De - Fe/2.0, 0);
              aW = max( Fw, Dw + Fw/2.0, 0);
              aN = max(-Fn, Dn - Fn/2.0, 0);
              aS = max( Fs, Ds + Fs/2.0, 0);

              aP = aE + aW + aN + aS + Fe - Fw + Fn - Fs;

              d_e[i][j] = dy*1/aP;

              if(method =='G') u_star[i][j] = (aE*u_star[i][j+1] + aW*u_star[i][j-1] + aN*u_star[i-1][j] +
aS*u_star[i+1][j])/aP - d_e[i][j]*(p[i][j+1] - p[i][j]);
              if(method =='J') u_star[i][j] = (aE*u[i][j+1] + aW*u[i][j-1] + aN*u[i-1][j] + aS*u[i+1][j])/aP -
d_e[i][j]*(p[i][j+1] - p[i][j]);

    }
}


// 1.2 u-Boundary Conditions
u_Boundary_Conditions(m, n, u_star, u_left_value, u_right_value, u_top_value,
u_bottom_value);
```

For the interior points, we first calculate the coefficients

$$F_e = \left[\frac{u_i^{j+1} + u_i^j}{2}\right]\Delta y \times 1; \quad F_w = \left[\frac{u_i^j + u_i^{j-1}}{2}\right]\Delta y \times 1$$

$$F_n = \left[\frac{v_{i-1}^{j+1} + v_{i-1}^j}{2}\right]\Delta x \times 1; \quad F_s = \left[\frac{v_i^{j+1} + v_i^j}{2}\right]\Delta x \times 1$$

$$D_e = D_w = \frac{1}{Re}\frac{\Delta y \times 1}{\Delta x}; \quad D_n = D_s = \frac{1}{Re}\frac{\Delta x \times 1}{\Delta y}$$



From Hybrid Scheme, the coefficients are calculated as:

$$a_E = \max\left[-F_e, D_e - \frac{F_e}{2}, o\right]; \quad a_W = \max\left[F_w, D_w + \frac{F_w}{2}, o\right];$$

$$a_N = \max\left[-F_n, D_n - \frac{F_n}{2}, o\right]; \quad a_S = \max\left[F_s, D_s + \frac{F_s}{2}, o\right];$$

$$a_P = a_E + a_W + a_N + a_S + (F_e - F_w + F_n - F_s);$$

$$A_e = \Delta y \times 1; \quad A_n = \Delta x \times 1$$

$$de_i^j = A_e / {a_P}_i^j$$

Based on the selection whether to solve using Gauss-Siedel or Jacobi method, we have two equations:

$$u_i^{*\,j} = (\textstyle\sum_{nb} a_{nb} u_{nb}^*)/a_P - de\left(p_i^{j+1} - p_i^j\right); \quad \text{for Gauss Seidel}$$

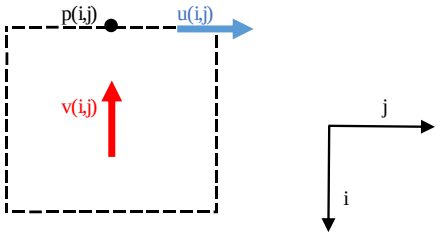
$$u_i^{*\,j} = (\textstyle\sum_{nb} a_{nb}\, u_{nb})/a_P - de\left(p_i^{j+1} - p_i^j\right); \quad \text{for Jacobi}$$

Y - Momentum Equation: on the v-control volume cell

```
// 1.3 Y-Momentum Equation Interior:
for(int i=1; i<m-1; i++){
    for(int j=1; j<n; j++){
            Fe = (u[i+1][j] + u[i][j])/2.0*dy*1;
            Fw = (u[i+1][j-1] + u[i][j-1])/2.0*dy*1;
            Fn = (v[i][j] + v[i-1][j])/2.0*dx*1;
            Fs = (v[i+1][j] + v[i][j])/2.0*dx*1;

            De = (1/Re)*(dy*1/dx);
            Dw = (1/Re)*(dy*1/dx);
            Dn = (1/Re)*(dx*1/dy);
            Ds = (1/Re)*(dx*1/dy);

            aE = max(-Fe, De - Fe/2.0, 0);
            aW = max( Fw, Dw + Fw/2.0, 0);
            aN = max(-Fn, Dn - Fn/2.0, 0);
            aS = max( Fs, Ds + Fs/2.0, 0);

            aP = aE + aW + aN + aS + Fe - Fw + Fn - Fs;

            d_n[i][j] = dx*1/aP;

            if(method =='G') v_star[i][j] = (aE*v_star[i][j+1] + aW*v_star[i][j-1] + aN*v_star[i-1][j] +
aS*v_star[i+1][j])/aP - d_n[i][j]*(p[i][j] - p[i+1][j]);
            if(method =='J') v_star[i][j] = (aE*v[i][j+1] + aW*v[i][j-1] + aN*v[i-1][j] + aS*v[i+1][j])/aP -
d_n[i][j]*(p[i][j] - p[i+1][j]);
    }
}
```

```
// 1.4 v-Boundary Conditions
v_Boundary_Conditions(m, n, v_star, v_left_value, v_right_value, v_top_value, v_bottom_value);
```

For the interior points, we first calculate the coefficients



$$F_e = \left[\frac{u_{i+1}^{j} + u_i^{j}}{2}\right]\Delta y \times 1; \quad F_w = \left[\frac{u_{i+1}^{j-1} + u_i^{j-1}}{2}\right]\Delta y \times 1$$

$$F_n = \left[\frac{v_i^{j} + v_{i-1}^{j}}{2}\right]\Delta x \times 1; \quad F_s = \left[\frac{v_{i+1}^{j} + v_i^{j}}{2}\right]\Delta x \times 1$$

$$D_e = D_w = \frac{1}{Re}\frac{\Delta y \times 1}{\Delta x}; \quad D_n = D_s = \frac{1}{Re}\frac{\Delta x \times 1}{\Delta y}$$

From Hybrid Scheme, the coefficients are calculated as:

$$a_E = \max\left[-F_e, D_e - \frac{F_e}{2}, o\right]; \quad a_W = \max\left[F_w, D_w + \frac{F_w}{2}, o\right];$$

$$a_N = \max\left[-F_n, D_n - \frac{F_n}{2}, o\right]; \quad a_S = \max\left[F_s, D_s + \frac{F_s}{2}, o\right];$$

$$a_P = a_E + a_W + a_N + a_S + (F_e - F_w + F_n - F_s);$$

$$A_e = \Delta y \times 1 \quad A_n = \Delta x \times 1$$

$$dn_i^j = A_n / {a_P}_i^j$$

Based on the selection whether to solve using Gauss-Siedel or Jacobi method, we have two equations:

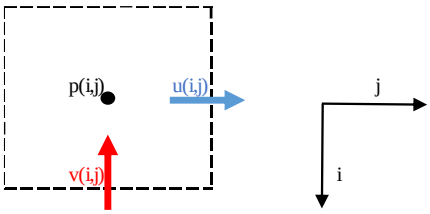
$${v_i^*}^j = (\textstyle\sum_{nb} a_{nb} v_{nb}^*)/a_P - dn\left(p_i^j - p_{i+1}^j\right); \quad \text{for Gauss Seidel}$$

$${v_i^*}^j = (\textstyle\sum_{nb} a_{nb}\, v_{nb})/a_P - dn\left(p_i^j - p_{i+1}^j\right); \quad \text{for Jacobi}$$

Pressure Correction Equation: on the p-control volume cell

```
// 2.1 Initialising pressure correction array to zero
for(int i=0; i<m+1; i++){
    for(int j=0; j<n+1; j++){
              pc[i][j] = 0.0;
    }
}

// 2.2 Pressure Correction Interior
for(int i=1; i<m; i++){
    for(int j=1; j<n; j++){
              aE = d_e[i][j]*dy*1;
              aW = d_e[i][j-1]*dy*1;
              aN = d_n[i-1][j]*dx*1;
              aS = d_n[i][j]*dx*1;

              aP = aE + aW + aN + aS;

              b[i][j] =  (u_star[i][j] - u_star[i][j-1])*dy*1 + (v_star[i-1][j] - v_star[i][j])*dx*1;

              pc[i][j] = (aE*pc[i][j+1] + aW*pc[i][j-1] + aN*pc[i-1][j] + aS*pc[i+1][j] - b[i][j])/aP;

    }
}
```

From the calculate values of de and dn, we calculate the values of a_{nb}



$$a_E = de_i^j \times A_e; \quad a_W = de_i^{j-1} \times A_w;$$
$$a_N = dn_{i-1}^j \times A_n; \quad a_S = dn_i^j \times A_s;$$
$$a_P = a_E + a_W + a_N + a_S;$$

Residual of continuity equation is calculated as:

$$b_i^j = (u_i^{*j} - u_i^{*j-1})A_e + (v_{i-1}^{*j} - v_i^{*j})A_n$$

The pressure correction term is updated using Gauss Seidel method as:

$$pc_i^j = (\sum_{nb} a_{nb} pc_{nb} - b_i^j)/a_P$$

Correcting Pressure and Velocity values

```
error_u = 0.0;
error_v = 0.0;

// 3.1 Correcting Pressure Field
for(int i=1; i<m; i++){
    for(int j=1; j<n; j++){
              p[i][j] = p[i][j] + pressure_alpha*pc[i][j];
    }
}

// 3.2 p-Boundary Conditions
p_Boundary_Conditions(m, n, p);


// 3.3 Correcting u-velocity
for(int i=1; i<m; i++){
    for(int j=1; j<n-1; j++){
              error_u += pow(u[i][j] - (u_star[i][j] - vel_alpha*d_e[i][j]*(pc[i][j+1] - pc[i][j])), 2);
              u[i][j] = u_star[i][j] - vel_alpha*d_e[i][j]*(pc[i][j+1] - pc[i][j]);
              u_star[i][j] = u[i][j];
    }
}

// 3.4 u-Boundary Conditions
u_Boundary_Conditions(m, n, u, u_left_value, u_right_value, u_top_value, u_bottom_value);
u_Boundary_Conditions(m, n, u_star, u_left_value, u_right_value, u_top_value, u_bottom_value);

// 3.5 Correcting v-velocity
for(int i=1; i<m-1; i++){
    for(int j=1; j<n; j++){
              error_v += pow(v[i][j] - (v_star[i][j] - vel_alpha*d_n[i][j]*(pc[i][j] - pc[i+1][j])), 2);
              v[i][j] = v_star[i][j] - vel_alpha*d_n[i][j]*(pc[i][j] - pc[i+1][j]);
              v_star[i][j] = v[i][j];
    }
}

// 3.6 v-Boundary Conditions
v_Boundary_Conditions(m, n, v, v_left_value, v_right_value, v_top_value, v_bottom_value);
v_Boundary_Conditions(m, n, v_star, v_left_value, v_right_value, v_top_value, v_bottom_value);
```

The values of pressure and velocities are updated with the formula:

$$p_i^j = p_i^j + \alpha_p \, pc$$

$$u_i^j = u_i^{*j} - \alpha_u \, de_i^j \left(pc_i^{j+1} - pc_i^j\right)$$
$$v_i^j = v_i^{*j} - \alpha_v \, dn_i^j \left(pc_i^{j+1} - pc_i^j\right)$$

Once the values have converged, we calculate the collocated grid's values for pressure and velocities with the code:

```
void calculate_Collocated_Grid(int m, int n, double u[m+1][n], double v[m][n+1], double p[m+1][n+1], double
u_final[m][n], double v_final[m][n], double p_final[m][n]){
    for(int i=0; i<m; i++){
            for(int j=0; j<n; j++){
                    u_final[i][j] = (u[i][j] + u[i+1][j])/2.0;
                    v_final[i][j] = (v[i][j] + v[i][j+1])/2.0;
                    p_final[i][j] = (p[i][j] + p[i+1][j] + p[i][j+1] + p[i+1][j+1])/4.0;
            }
    }
}
```

The values of Stream function and Vorticity is calculated from the collocated grid velocities and then the results are shown in the next section.

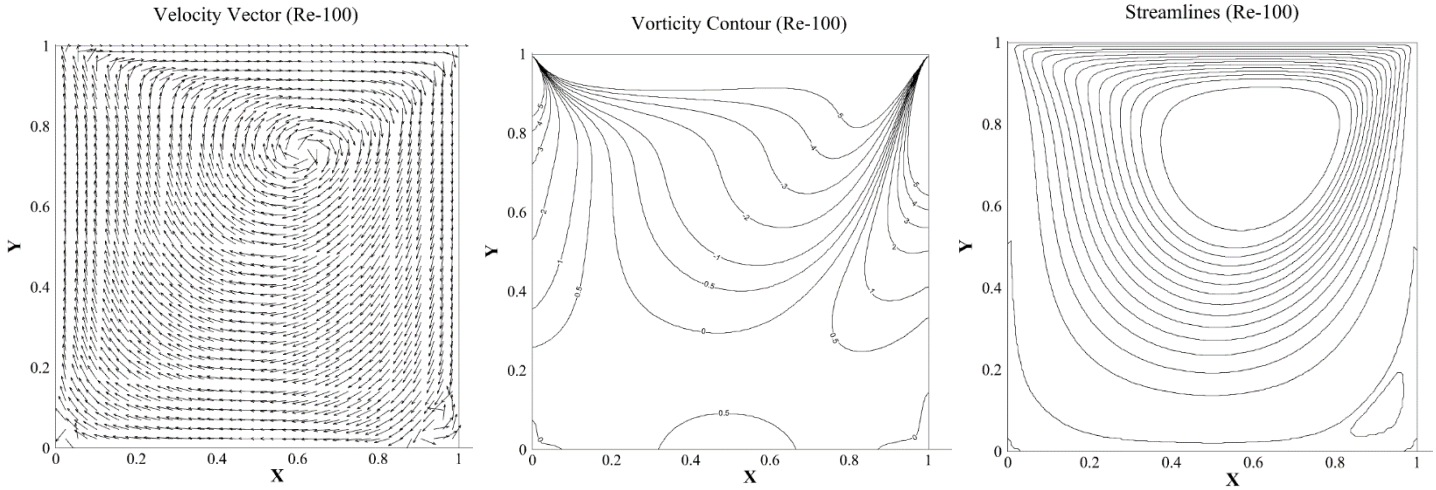
Results

Parameters taken to solve for different Reynold's number

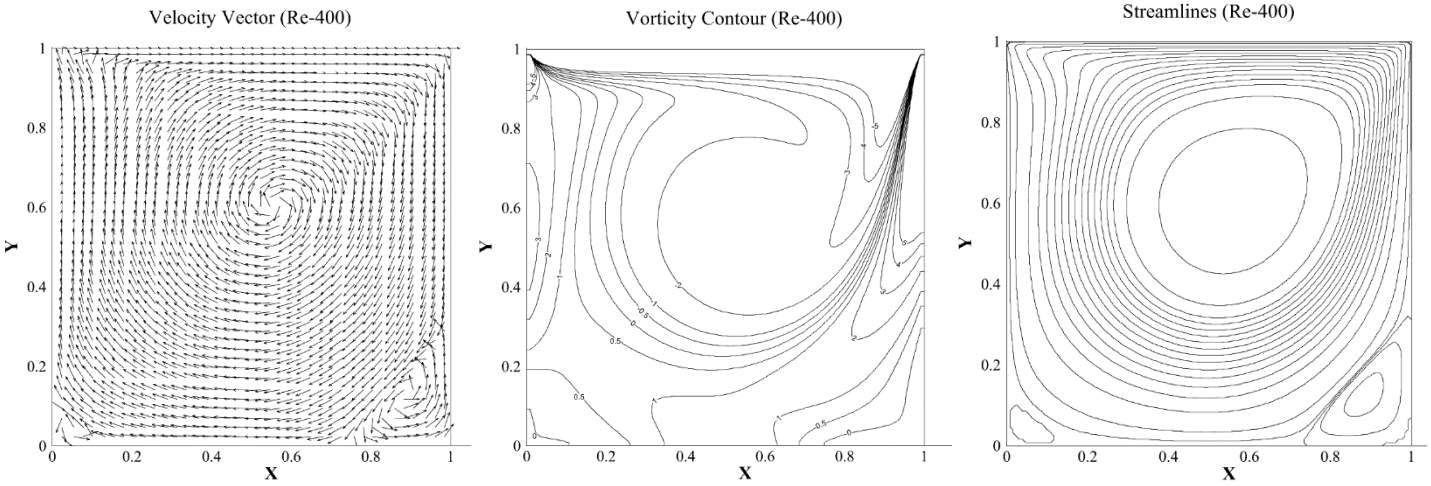
Re	Pressure Relaxation factor (α_p)	Velocity Relaxation factor (α_v)	Solution Method	Converged?
100	0.5	0.5	Gauss Seidel	Yes
400	0.8	0.8	Gauss Seidel	Yes
1000	1.0	1.0	Jacobi	No

1. Contours and Velocity Magnitude plots:

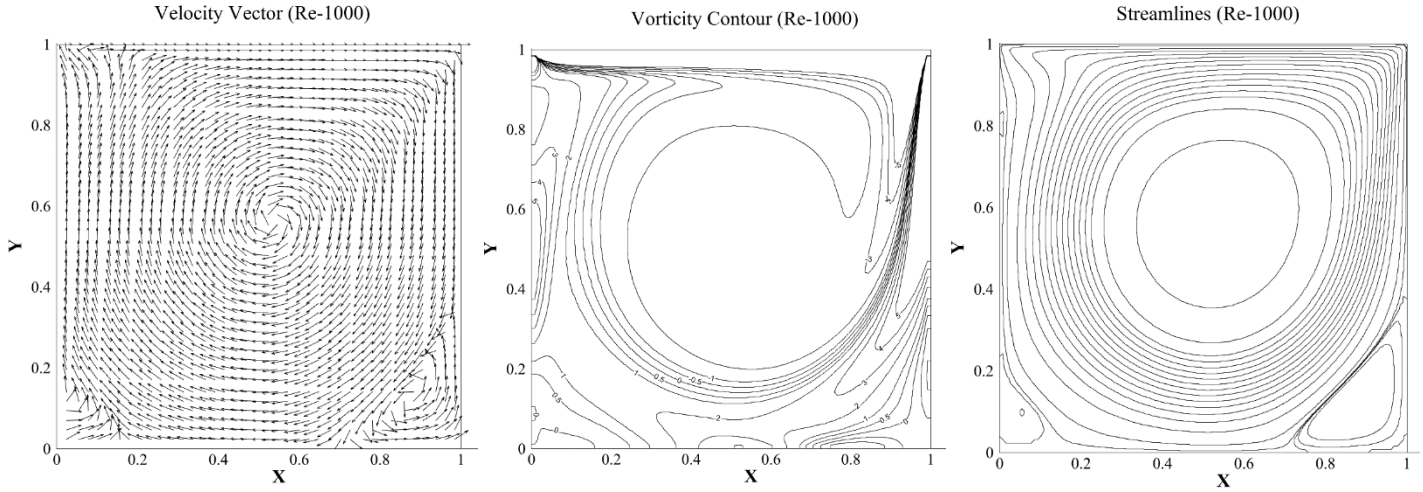
Re 100



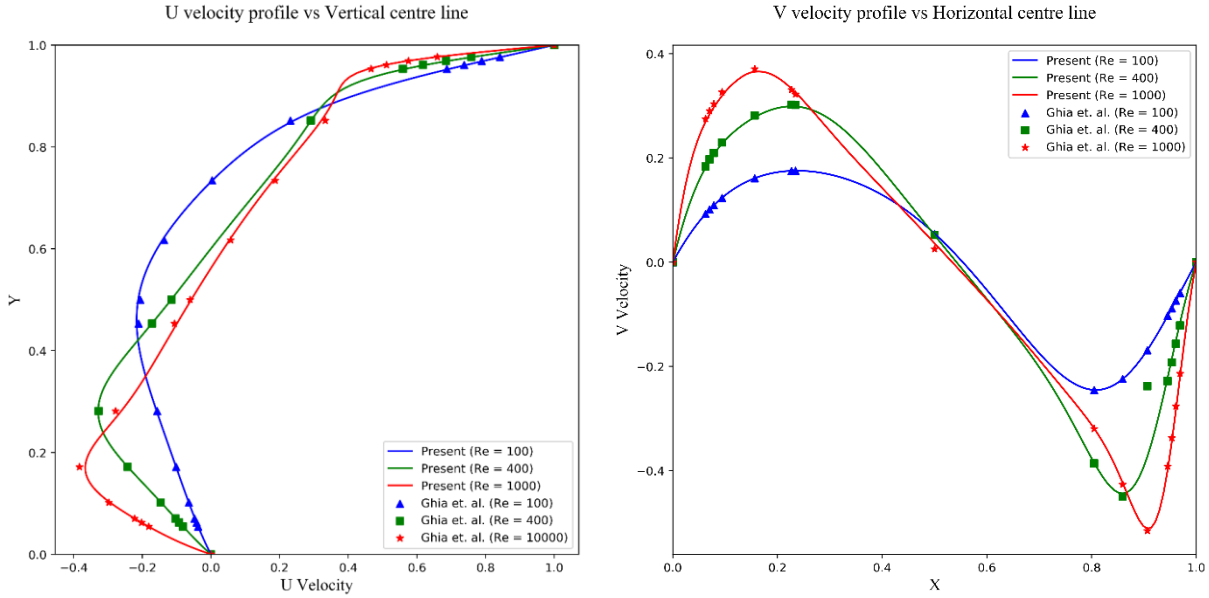
Re 400



Re 1000



2. U velocity profile at vertical midsection and V velocity profile at horizontal midsection.



3. **Re-100:** Stream: -0.1033, x_val:0.617, y_val:0.742, vorticity: -3.1670
   Re-400: Stream: -0.1132, x_val:0.555, y_val:0.617, vorticity: -2.2867
   Re-1000: Stream: -0.1152, x_val:0.539, y_val:0.570, vorticity: -1.9967

Assignment 02

ME 670: Advanced Computational Fluid Dynamics

Multi-Grid Methods

V-Cycle and Full-Multigrid Method

Nirmal S.
[234103107]
Computational Mechanics
MTech. Mechanical Engineering
IIT-Guwahati

Table of Contents

Problem Statement:

1. Develop a V-cycle program for the one-dimensional model problem $-u''(x)+\sigma u(x)=f$, with homogenous boundary conditions, solved using finite difference method. Write a function/subroutine for each individual component of the algorithm as follows.

(a) Given an approximation array v, a right-side array f, and a level number $1\leq l\leq L$ (smallest level number corresponds to the finest grid), write separate subroutines that will carry out v number of weighted Jacobi or Gauss-Seidel sweeps on level l. Keep them in a file named "relaxation_methods" for example: relaxation_methods.F90
(b) Given an array f and a level number $1\leq l\leq L-1$, write a subroutine that will carry out full weighting between level l and level $l+1$. Keep it in a file named "restriction_methods".
(c) Given an array v and a level number $2\leq l\leq L$, write a subroutine that will carry out linear interpolation between level l and level $l-1$. Keep it in a file named "prolongation_methods".
(d) Write a subroutine named 'V_cylce' that carries out a single V-cycle by calling the three preceding subroutines. The V-cycle should be able to start from a given level l. Keep it in a file named "MG_methods".
(e) Write a main program that initializes the data arrays and calls V-cycle subroutine. For testing, for fixed k, take $f(x)=C\sin(k\pi x)$ on the interval $0\leq x\leq 1$, where C is a constant. Then the exact solution to model problem is

$$u(x) = \frac{C}{\pi^2 k^2 + \sigma}\sin(k\pi x)$$

(f) Write another subroutine that computes 2-norm of error and residual. Keep it in a file named "postprocessing_methods". You can also keep files writing subroutines in this file.
(g) Take $n=512$, $\omega=2/3$, $\nu 1=\nu 2=2$, $\sigma=1$ and $C=\pi 2k2+\sigma$. For $k=1$ and 10, apply basic iterative methods and V-cycle iterations till the residual 2-norm is greater than $10-6$. Plot the residual norm against the number of iterations for all three methods on the same figure.

2. Using the V-cycle subroutine, write a full multigrid (FMG) subroutine named 'FMG'. The FMG-cycle should start from a given level l. Keep it in the file named "MG_methods". Verify the solver using the problem statement given in Q1. Report the 2-norm of the residual after applying one FMG-cycle to the test problem in Q1. Afterwards, apply V-cycle to solution approximation till the residual 2-norm is greater than 10^{-6}. Report the number of V-cycle iterations.

Grid Details

A 1D grid is considered, which stores the values of *u, v, f, v_temp* which can be split in levels as shown in the figure below:

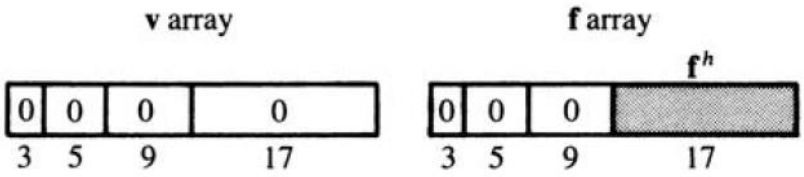


Figure 1: Image of the arrays to store the values (here the finest grid is considered with 17 grid points).

If the number of grid points are 2^l+1: (for 512 divisions, there are 513 grid points and l=9)

Then the total length of the array: $(2^1 + 1) + (2^2 + 1) + \cdots + (2^l + 1) = (2^{l+1} - 2) + l$

Number of grids in level l: $2^l + 1$

For a given number of divisions, number of levels *Levels*: $\log_2(no\_of\_div + 1)$

Discretised Equations

The 1D model problem:

$$-u''(x) + \sigma u(x) = f$$

Discretising it in Finite Difference Method:

$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} + \sigma u_i = f_i$$

C-Code: Multi Grid Methods

Main Code:

The main function will run either V-Cycle or Full-MultiGrid based on the user input. The function can be checked from the program file *"A02_234103107.c"*.

Functions:

Functions to calculate the starting point of the level, and the number of cells in the array at that level:

```
int index_level(int Levels, int lvl){return pow(2, Levels-lvl+1) - 2 + (Levels-lvl);}

int num_grid(int Levels, int lvl){return pow(2,Levels-lvl+1) + 1;}
```

Function to perform relaxation method, based on the level *lvl* at which it has to be performed, on array *v* and *f*, and based on the *method* which is *J* for Jacobi and *G* for Gauss Seidel.

```
void relaxation_methods(int M, int lvl, int Levels, double v[M], double f[M], double sigma, int nu,
                        char method, double J_weight, double h){

        int index_lvl = index_level(Levels, lvl);
        int m = index_lvl + num_grid(Levels, lvl);
        double h2 = pow(h,2);

        if(method == 'J') {
                int n = pow(2,Levels-lvl+1) + 1;
                double v_new[n];
                for(int i=0; i<n; i++) v_new[i] = v[index_lvl+i];

                for(int iter=0; iter<nu; iter++){
                        for(int i=1; i<n-1; i++){
                                v_new[i] = (1-J_weight)*v_new[i] + J_weight*(h2*f[index_lvl+i] +
v[index_lvl+i-1] + v[index_lvl+i+1])/(2+sigma*h2);
                        }
                        for(int i=0; i<n; i++)  v[index_lvl+i] = v_new[i];
                }
        }

        if(method == 'G') {
                for(int iter=0; iter<nu; iter++){
                        for(int i=index_lvl+1; i<m-1; i++){
                                v[i] = (h2*f[i] + v[i-1] + v[i+1])/(2+sigma*h2);
                        }
                }
        }

        return;
}
```

Function to perform *Full Weighting* restriction method to *f* at level *lvl* and store it in the next level.

```
void restriction_methods(int M, int lvl, int Levels, double f[M]){

        int index_lvl = index_level(Levels, lvl);
        int index_lvl_nxt = index_level(Levels, lvl+1);
        int n = num_grid(Levels, lvl);

        for(int i=1; i<n/2; i++){
                f[index_lvl_nxt+i] = 0.25*(f[index_lvl+2*i-1] + 2*f[index_lvl+2*i] +
                                           f[index_lvl+2*i+1]);
        }
        return;
}
```

Function to perform *Linear Interpolation* prolongation method to *v* at level *lvl* and store it in the previous level

```
void prolongation_methods(int M, int lvl, int Levels, double v[M], double v_temp[M]){

    void v_Boundary_Conditions(int m, int n, double v[m][n+1], double v_left_value, double v_right_value,
                               double v_top_value, double v_bottom_value){
        for(int i=0; i<m; i++){
                v[i][0] = v_left_value*2 - v[i][1];
                v[i][n] = v_right_value*2 - v[i][n-1];
        }
        for(int j=0; j<n+1; j++){
                v[0][j] = v_top_value;
                v[m-1][j] = v_bottom_value;
        }
        return;
    }
}
```

Function to calculate error and residual at level *lvl* using *u, v,*and *f* and store it in *error* and *res*.

```
void post_processing_methods(int M, int lvl, int Levels, double h, double sigma, double f[M], double
v[M], double u[M], double *error, double *res){

        int index_lvl = index_level(Levels, lvl);
        int n = num_grid(Levels, lvl);
        double h2 = pow(h,2);

        for(int i=index_lvl+1; i<index_lvl+n-1; i++){
                *res += pow(f[i] - 1/h2*(- v[i-1] - v[i+1] + (2+sigma*h2)*v[i]), 2);
                *error += pow(u[i] - v[i], 2);
        }
        *res = sqrt(*res);
        *error = sqrt(*error);

        return;
}
```

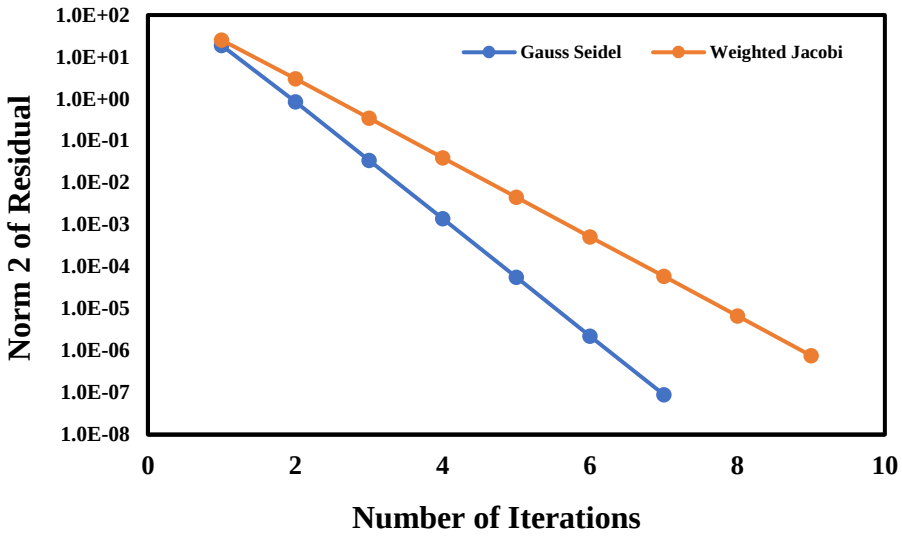
Function to run *V-Cycle*

```
void v_cycle(int M, int Levels, int lvl, double v[M], double f[M], double residual[M], double u[M],
             double C, double k, double sigma, int nu1, int nu2, double h, char method, double J_weight){

        double v_temp[M];  // temp variable used in V-cycle while going up the levels
        for(int i=0; i<M; i++) v_temp[i] = 0;
        for(int i=0; i<index_level(Levels, 1); i++) v[i] = 0;

        int index_lvl = index_level(Levels, lvl);
        int n = num_grid(Levels, lvl);
        double h1, h2;

        // *** Initialising 'f' (RHS) array and exact solution 'u' ***
        for(int i=index_lvl; i<index_lvl+n; i++) f[i] = C*sin((float)k*M_PI*(i-index_lvl)*h);
        for(int i=index_lvl; i<index_lvl+n; i++) u[i] = C/(pow(M_PI,2)*pow(k,2) +
                                                           sigma)*sin((float)k*M_PI*(i-index_lvl)*h);


        // *** Moving towards coarser grids ***
        for(int lvl_i=lvl; lvl_i<Levels; lvl_i++){

                h1 = h*pow(2,lvl_i-1);
                h2 = pow(h1,2);

                // Applying relaxation method to next level
                relaxation_methods(M, lvl_i, Levels, v, f, sigma, nu1, method, J_weight, h1);

                // Calculating the residual
                index_lvl = index_level(Levels, lvl_i);
                n = num_grid(Levels, lvl_i);
                for(int i=index_lvl+1; i<index_lvl+n-1; i++)    residual[i] = f[i] - 1/h2*(- v[i-1] -
                                                                                  v[i+1] + (2+sigma*h2)*v[i]);

                // Taking residual to next level
                restriction_methods(M, lvl_i, Levels, residual);
                index_lvl = index_level(Levels, lvl_i+1);
                n = num_grid(Levels, lvl_i+1);

                // Storing residual back into 'f' at the next level
                for(int i=index_lvl+1; i<index_lvl+n-1; i++)  f[i] = residual[i];

        }


        // *** Solve the equation at the coarsest grid ***
        h1 = h*pow(2,Levels-1);
        relaxation_methods(M, Levels, Levels, v, f, sigma, nu1, method, J_weight, h1);


        // *** Moving towards finer grids ***
        for(int lvl_i=Levels; lvl_i>lvl; lvl_i--){

                // Interpolate to the previous level
                prolongation_methods(M, lvl_i, Levels, v, v_temp);

                // Recalculate the 'v' value with the interpolated value
                index_lvl = index_level(Levels, lvl_i-1);
                n = num_grid(Levels, lvl_i-1);
                for(int i=index_lvl+1; i<index_lvl+n-1; i++)  v[i] += v_temp[i];

                // Relaxing the equation at the previous level
                h1 = h*pow(2,lvl_i-2);
                relaxation_methods(M, lvl_i-1, Levels, v, f, sigma, nu2, method, J_weight, h1);

        }

        return;
}
```

Results

1. Plotting residual norm against number of iterations:

For $k = 1$:



For $k = 10$:

2. Report of residual after FMG and number of V-cycles needed for until residual norm is below epsilon

K	Relaxation Method	Residual after one FMG cycle.	Number of V-Cycles required further
1	Gauss Seidel	18.8597	6
1	Jacobi	25.3194	8
10	Gauss Seidel	1843.9372	7
10	Jacobi	2133.6977	9

Assignment 03

ME 670: Advanced Computational Fluid Dynamics

Conjugate Gradient

and Preconditioned Conjugate Gradient

Nirmal S.
[234103107]
Computational Mechanics
MTech. Mechanical Engineering
IIT-Guwahati

Table of Contents

Problem Statement:

1. Consider 2D conduction problem governed by equation

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

On a unit square domain with homogamous Dirichlet boundary conditions i.e. $T=0$ at the left, right and bottom boundaries. The top wall (non-dimensional) temperature is 1 unit. The analytical solution of the problem is:

$$T(x,y) = \frac{2}{\pi}\sum_{n=1}^{\infty} \frac{(-1)^{n+1}+1}{n} \sin(n\pi x) \frac{\sinh(n\pi y)}{\sinh(n\pi)}$$

Discretize the governing equation using finite volume method. The FV mesh is uniform in both the directions with $N_i=N_j=128$ finite volumes in x- and y-directions, respectively. The FV cells are numbered in lexicographic ordering by lines of constant i. Show the final expression of the discrete set of equations in the form $Ax=b$.

Solve the system of equations using conjugate gradient method, preconditioned conjugate gradient method with Jacobi, ILU and SIP preconditioners. Use residual 2-norm falling below 10^{-6} as convergence criteria.

a. Compare the iterations vs residual 2-norm plot for all four iterative methods.
b. Compare the temperature contours of the analytical and numerical solutions obtained from iterative methods.
c. Compare the temperature variation with y along mid-vertical plane x = 0.5 obtained from the analytical and numerical solutions.
d. Compare the temperature variation with x along mid-vertical plane y = 0.5 obtained from the analytical and numerical solutions.
e. For $N_i=N_j=4$, tabulate the values of diagonals of L and U matrices coming from ILU and SIP factorization.

n	L_W^n	L_S^n	L_P^n	U_N^n	U_E^n
1					
.					
16					

Grid Details

A 1D grid is considered using lexicographic ordering in the form from top left, along the top row, and down to the next line and so on, as shown in the figure below.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Figure 1: A sample grid showcasing a 5x5 grid and its lexicographic ordering considered in the code.

Discretised Equations

The 2D heat conduction problem:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

Discretising it in Finite Volume Method:

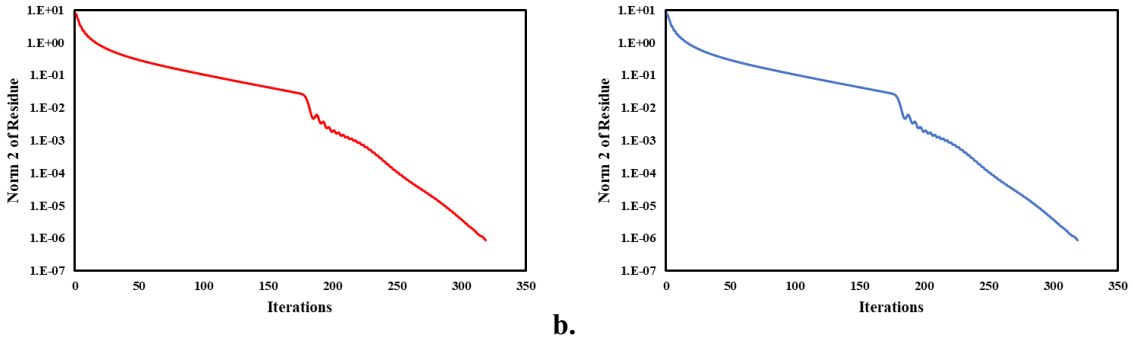
$$\beta^2 T_{i,j-1} + T_{i-1,j} - 2(1+\beta^2)T_{i,j} + T_{i+1,j} + \beta^2 T_{i,j+1} = 0$$

Method solved in C.

Plotting in Excel and Python.

Results

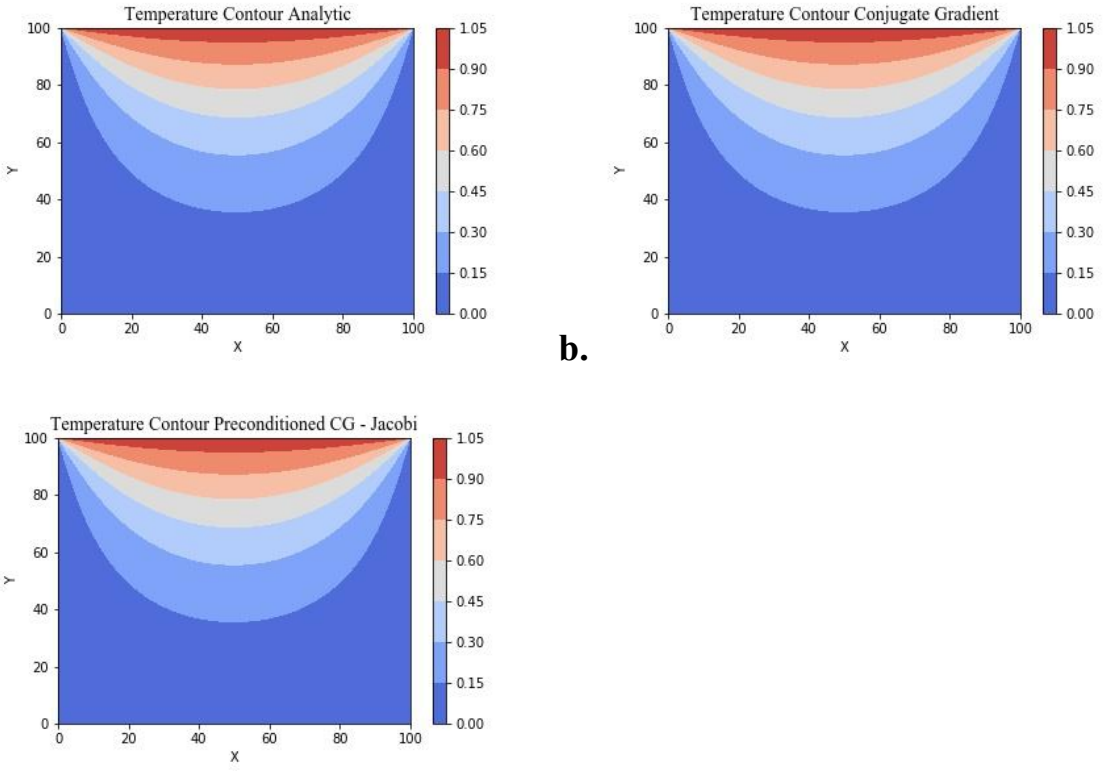
1. Plotting residual norm against number of iterations:



a. **b.**

Figure 2: Norm2 of residual vs iteration of (a) Conjugate Gradient Method, (b) Preconditioned CG – Jacobi Method.

2. Comparing Temperature Contours

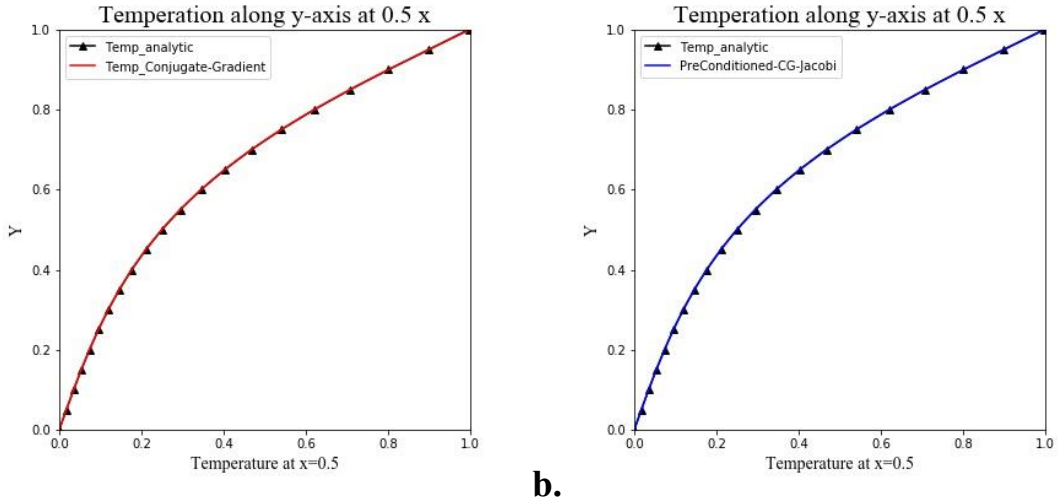


a. **b.**

c.

Figure 3: Temperature contours of (a) Analytic calculation of the solution, (b) Conjugate Gradient Method, (c) Preconditioned CG – Jacobi Method.

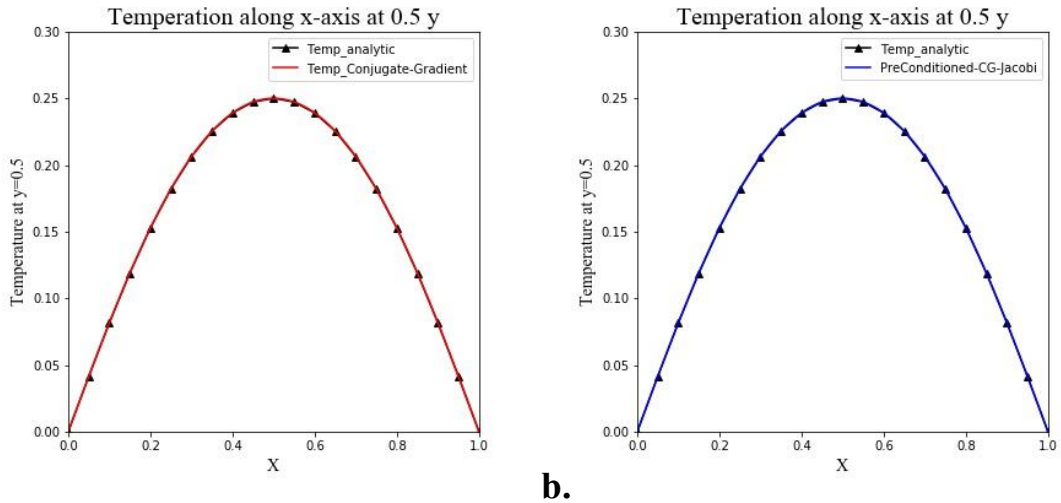
3. Temperature variation along y-axis at horizontal midplane



a. **b.**

Figure 4: Temperature along y-axis at horizontal midplane of (a) Conjugate Gradient Method, (b) Preconditioned CG – Jacobi Method.

4. Temperature variation along x-axis at vertical midplane



a. **b.**

Figure 5: Temperature along x-axis at vertical midplane of (a) Conjugate Gradient Method, (b) Preconditioned CG – Jacobi Method.