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: 2](#_Toc161522629)

[Grid Details 3](#_Toc161522630)

[Discretised Equations 4](#_Toc161522631)

[C-Code: SIMPLE Algorithm and Boundary Conditions 7](#_Toc161522632)

[*Grid Creation:* 7](#_Toc161522633)

[*Boundary Conditions:* 8](#_Toc161522634)

[*SIMPLE Algorithm:* 9](#_Toc161522635)

[Results 14](#_Toc161522636)

# 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 . 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∞‖<10−5 criterion. For the solution of the discretized equations, you can use the point Gauss-Seidel (GS) or ADI method.

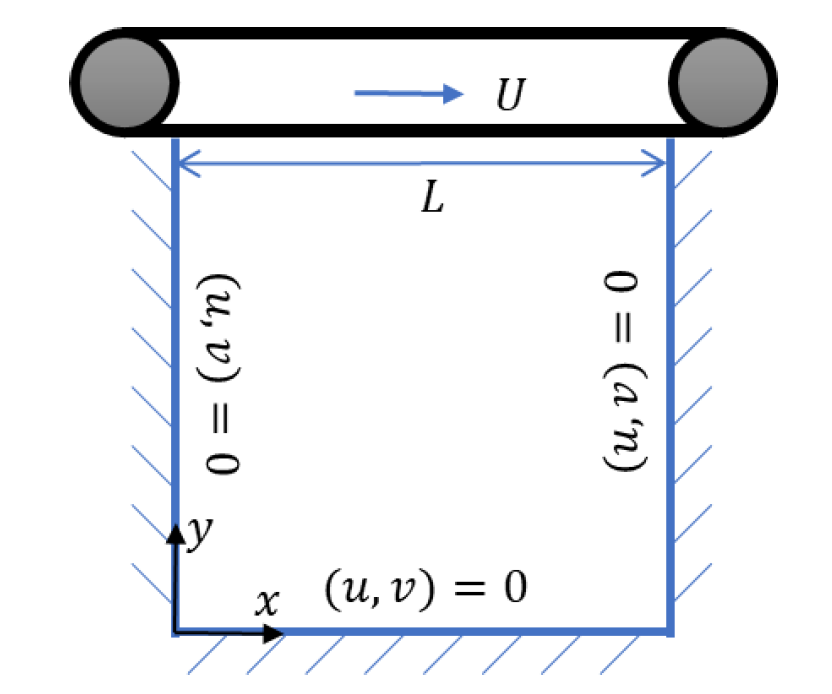


Figure 1: A Schematic of the lid-driven cavity problem

For Re (=𝜌𝑈/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: 𝜓min, location (𝑥, 𝑦) of 𝜓min and the value of 𝜔 at the location of 𝜓min.

# Grid Details

A uniform, cartesian, staggered grid of size 129129 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 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:*

*Incompressibility condition:*

*Momentum Equation:*

*Here,*

Getting the momentum equation to non-dimensional form:

*Here,*

In indicial notation it can be simplified to:

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

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

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

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

Integrating each term,

Where:

We get the x-momentum equation in form of:

From Hybrid Scheme, the coefficients are calculated as:

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

Integrating each term,

Where:

We get the y-momentum equation in form of:

From Hybrid Scheme, the coefficients are calculated as:



***Continuity Equation in 2D for p-control volume:***

Where:

# C-Code: SIMPLE Algorithm and Boundary Conditions

For simplicity, let us consider a grid of (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 here.

The u-velocity staggered grids are defined for here; from

The v-velocity staggered grids are defined for here; from

The pressure staggered grids are defined for here; from

// 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 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:

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:

***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:

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:

***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.

## *SIMPLE Algorithm:*

The algorithm is applied until the norm of error in u-velocity and v-velocity is less than

***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

From Hybrid Scheme, the coefficients are calculated as:

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

; for Gauss Seidel

; 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

From Hybrid Scheme, the coefficients are calculated as:

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

; for Gauss Seidel

; 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 and , we calculate the values of

Residual of continuity equation is calculated as:

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

***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:

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.

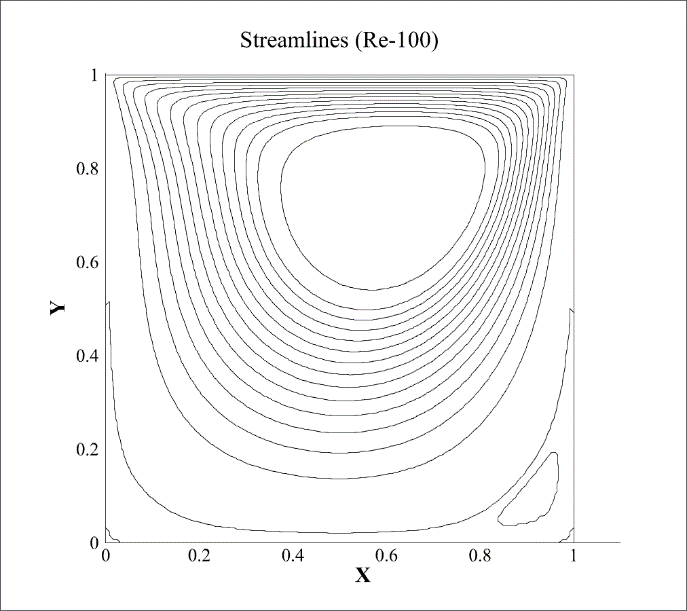
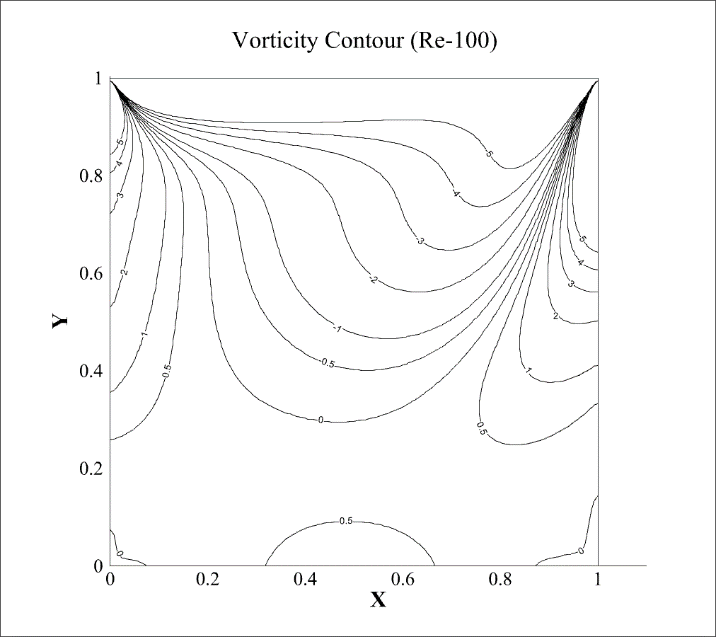
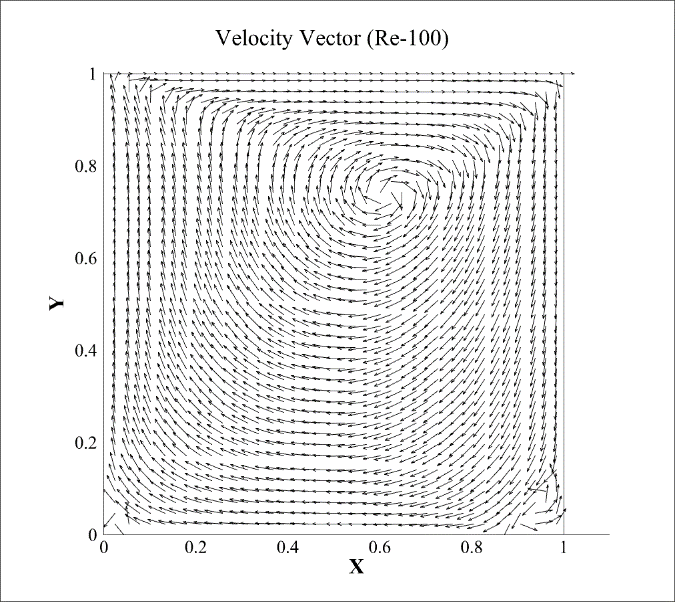
# Results

Parameters taken to solve for different Reynold’s number

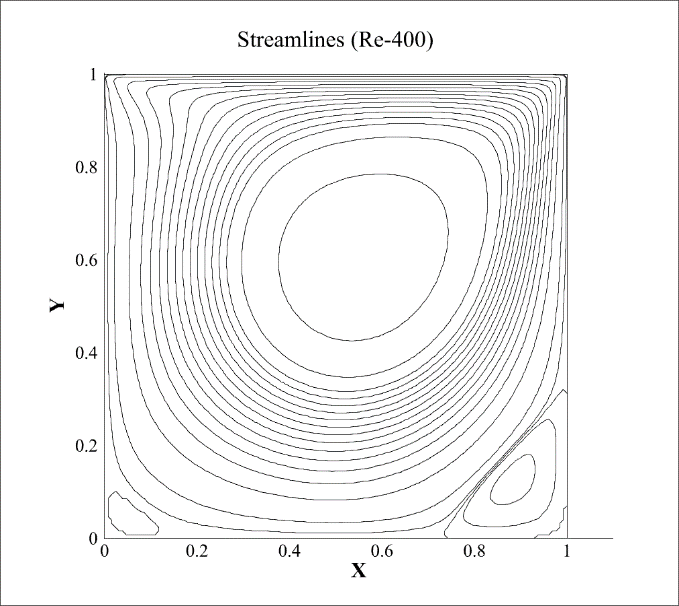
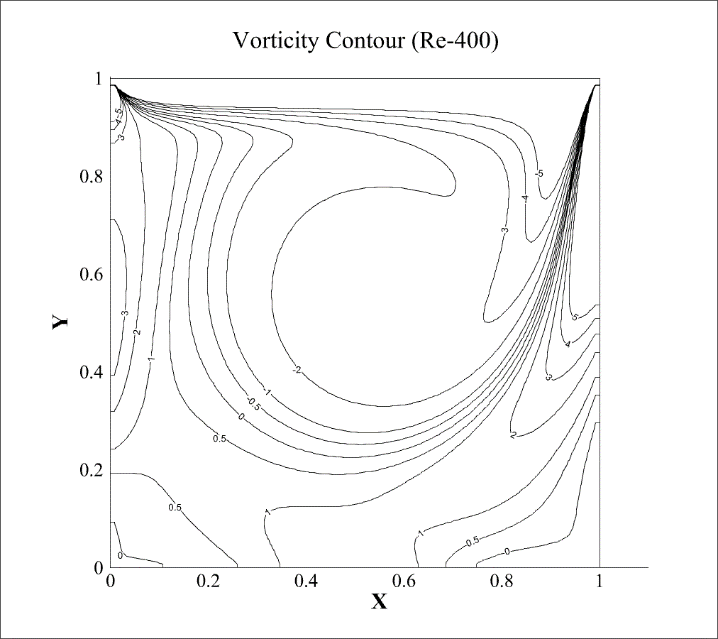
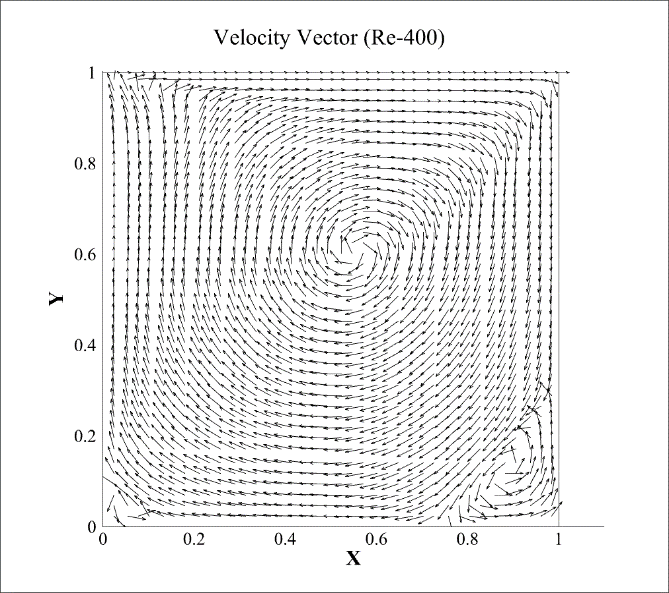
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Re** | **Pressure Relaxation factor ()** | **Velocity Relaxation factor ()** | **Solution Method** | **Converged?** |
| 100 | 0.5 | 0.5 | Gauss Seidel | Yes |
| 400 | 0.8 | 0.8 | Gauss Seidel | Yes |
| 1000 | 1.0 | 1.0 | Jacobi | No |

1. Contours and Velocity Magnitude plots:

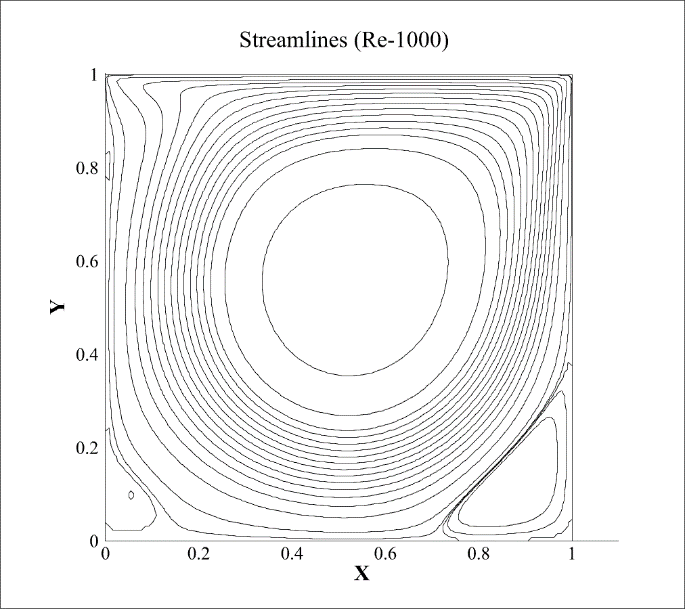
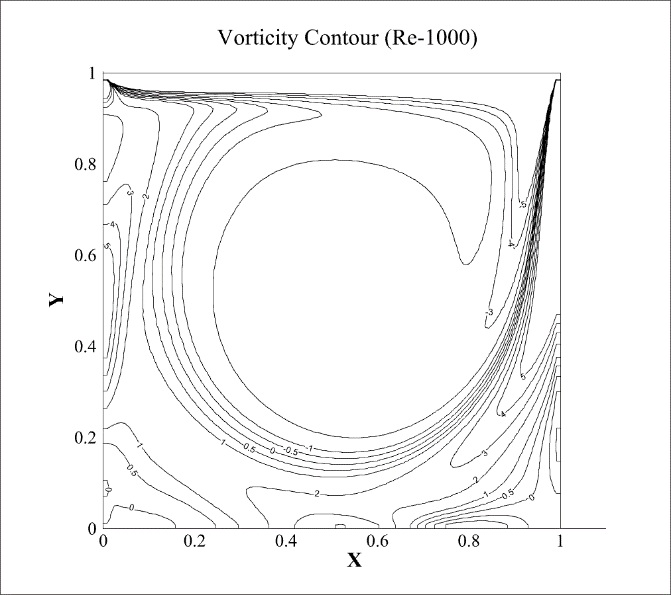
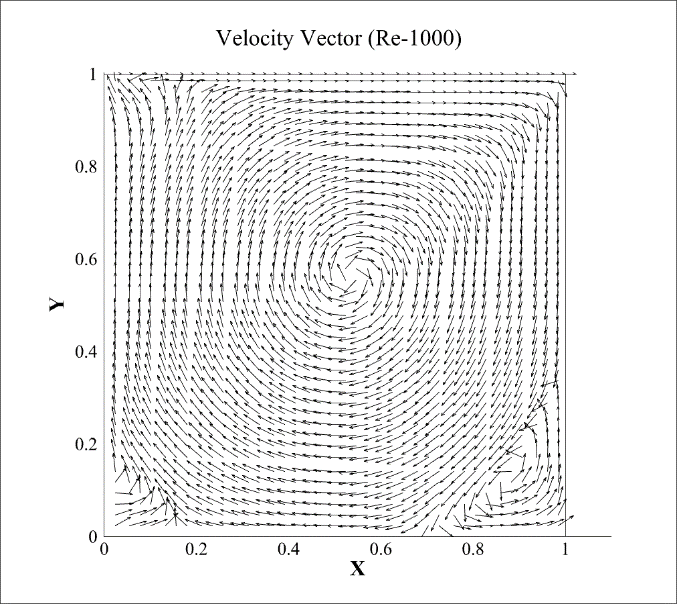
***Re 100***



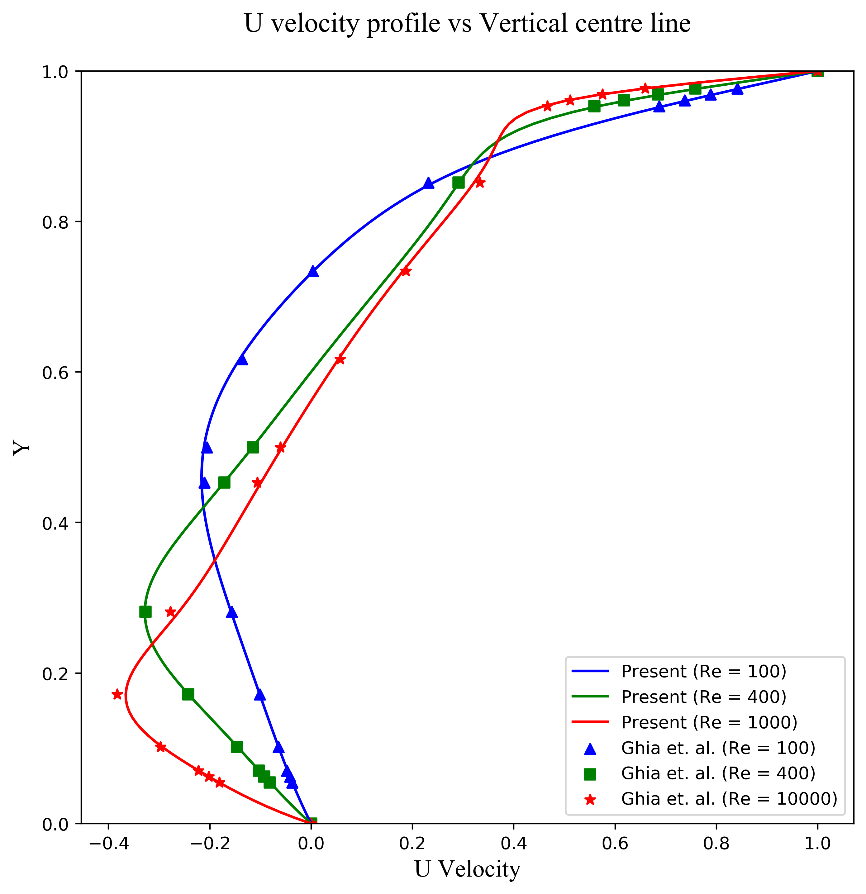
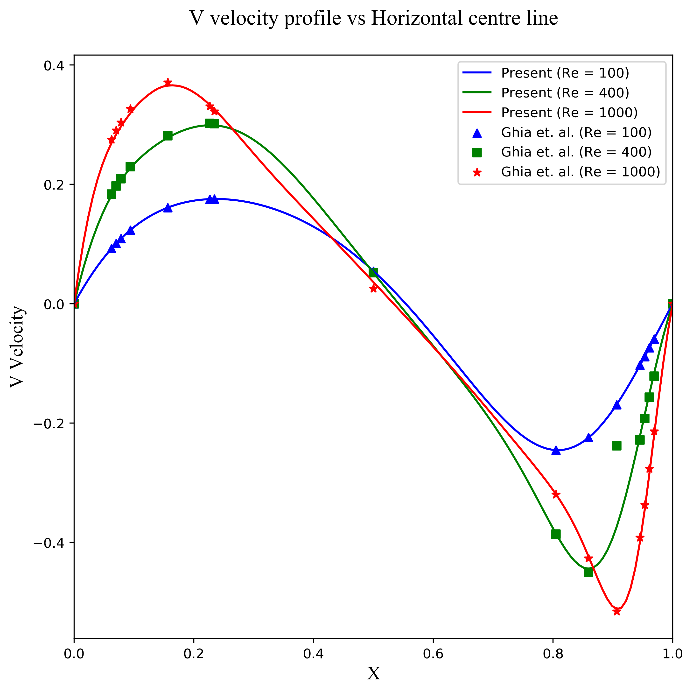
***Re 400***



***Re 1000***



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

1. **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