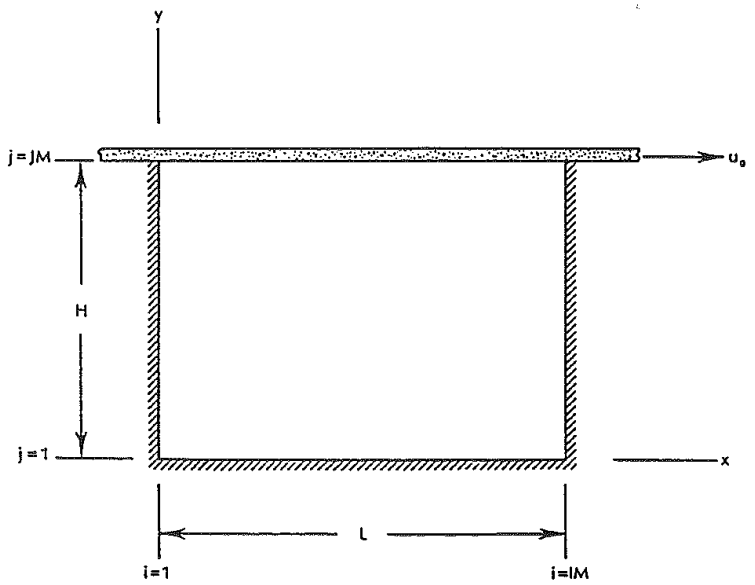
FLORIDA INSTITUTE OF TECHNOLOGY
MECHANICAL AND AEROSPACE ENGINEERING DEPARTMENT

**MAE 5150-E1: Computational Fluid Dynamics**
Fall 2017

# Max Le
# Final Project
Due December 11, 2017 @ 5:00pm

Consider the following figure:




The left, right, and bottom surfaces are stationary, while the top plate moves to the right with velocity $u = u_\infty$. This is known as *cavity-driven flow*, or simply *cavity flow*. For this project, you will write a code that will determine the laminar velocity field within the cavity. The equations to be solved are Eqns. (8-86, 8-87, and 8-88) in your textbook. Note that Eqn. (8.86) is the Poisson equation for pressure, and that

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

is called the *dilatation*.

These equations will be solved sequentially (not simultaneously) on a collocated mesh (not staggered). That is, for each time step, you will use Line Gauss-Seidel to iterate a solution to the pressure equation, followed by an explicit scheme to update the *x*- and *y*-components of velocity (which will use your pressure equation solution).

**Step 1: Write out on paper the numerical schemes for each equation.**

- Because the specific numerical schemes you must use are not in your book, you must develop them.
- Poisson equation for pressure (Eqn. 8-86): Use Line Gauss-Seidel. You used Line Gauss-Seidel in a previous assignment, so you should be able to adapt the equation from that assignment for use in this project. (Be mindful of the RHS term.) Use second-order central differencing for all second-order derivatives, and a first-order forward difference for the time derivative. See page 335 of your text as a guide for how to discretize the right-hand side, but keep in mind that the particular scheme on that page is for a staggered mesh. You will be best served if you write your numerical scheme to utilize Line Successive Over-Relaxation (LSOR). Your code will run much faster if you do.
- Dilitation: The pressure equation includes the dilatation, which means you must find its value at each mesh node. Use second-order central differencing and the definition above to find its value. To find the dilatation for a wall nodal point, use second-order forward or backwards differencing, as appropriate, instead.
- Momentum equations (Eqns. 8-87 and 8-88): Here, you will use a first-order forward difference for the time derivatives and explicit first-order upwinding for the convective derivatives. Continue to use central differencing for the second-order derivatives. See Eqns. (8-80 and 8-81) for a guide on how to discretize these equations (again keeping in mind that these equations are discretized for a staggered mesh), and Eqn. (8.96) and the surrounding text about how to incorporate upwinding. Remember, you do not know in advance whether the velocity will be positive or negative at a particular node.

**Step 2: Using the numerical schemes developed in Step 1, write a code to solve cavity-driven flow.**

- Your mesh should be 81 x 81 nodes (*IM* x *JM*) with $\Delta x = \Delta y = 0.00625$. The plate moves to the right with a non-dimensional velocity of $u = 1$. The Reynolds number is $Re = 1000$. For each time step, you should iterate your Line Gauss-Seidel solution for the pressure to an error less than *errormax* = 0.001.
- Assume the top and bottom surfaces are held at a constant (non-dimensional) pressure of 3350. For the left and right surfaces, assume $\partial p / \partial x = 0$. Properly embed boundary conditions accordingly.
- Run your code for 15,000 time steps with a $\Delta t = 0.003$ (non-dimensional).
- For the Line Gauss-Seidel, if you decide to use LSOR, the optimal value for the relaxation parameter is 1.315. This should keep the number of iterations down to approximately 95-120 each time step.

**Step 3: Create a streamline plot of the flow.**

- Once you have your solution, use MATLAB or other plotting software to create a streamline plot of the flow. To do this, you will need to load *x*, *y*, *u*, and *v* variable data into the plotting software.

- If you use MATLAB, the following will be of use:

  *streamline* (*x*, *y*, *u*, *v*, *SX*, *SY*, [*step-size*, *max vertices*])

  where *x*, *y*, *u*, *v* are 81 x 81 matrices. *SX* and *SY* are matrices of data points of where to start each streamline. To determine these, I would recommend the following commands:

  *sx*=[0.0:0.05:0.5];
  *sy*=[0.0:0.05:0.5];
  [*SX*, *SY*] = *meshgrid* (*sx*, *sy*);

  *Step-size* is the length of each segment along the streamline, and *max vertices* is the number of segments. I would recommend *step-size* = 0.1 and *max vertices* = 2500. You may vary any of these parameters to obtain the best plot.

- If you do not use MATLAB for the plot, use your best judgment.
- Ensure your axes are scaled equally and have labels and titles. Your plot should also be titled. Use all proper plotting techniques. You will be graded on these.
- A vector plot may also be turned in if done legibly. By "legibly" I mean in such a manner that the arrows can be easily seen, but do not significantly overrun each other, and the primary flow structures can be viewed. Including a *proper* vector plot, though not required, will be viewed favorably in determining the project grade.

**Step 4: To complete the project you must turn in:**

- The numerical schemes you developed in Step 1, neatly written and formatted or (preferably) typed.
- The streamline plot and (optionally) the vector plot. Hardcopies only.
- A hardcopy of your code.
- An electronic copy of your code submitted via Canvas.

The project grade will depend upon:

- Developing the correct numerical schemes
- Obtaining the correct simulation results
- Using all proper numerical techniques including
  - correct implementation of Line Gauss-Seidel for the pressure equation
  - correctly embedding the appropriate boundary conditions
  - proper use of pressure solution in the momentum equations
- Showing correct results on plot(s) using all proper plotting techniques

- Turning in all required materials as described above

## Developing Numerical Schemes for a Lid Driven Cavity Flow

## 1. Governing Equations

Our governing equations for this simulation are as follow:

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

*Equation 1- Continuity Equation*

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

*Equation 2- X momentum*

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

*Equation 3- X momentum*

From these equations, we need additional equations to solve for the pressure. Taking derivatives of both the X and Y momentum and adding them up results in the following:

$$\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} = -\frac{\partial D}{\partial t} - \frac{\partial^2}{\partial x^2}(u^2) - 2\frac{\partial^2}{\partial x \partial y}(uv) - \frac{\partial^2}{\partial y^2}(v^2) + \frac{1}{Re}\left(\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2}\right)$$

*Equation 3- Pressure Poisson Equation*

where D is the dilatation to ensure continuity and is given by

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

*Equation 4- Dilatation*

## 2. Scheme for the Pressure Poisson Equation

Discretizing both sides as follow:

$$\frac{P_{i+1,j}^{n+1} - 2P_{i,j}^{n+1} + P_{i-1,j}^{n+1}}{\Delta x^2} + \frac{P_{i,j+1}^{n} - 2P_{i,j}^{n} + P_{i,j-1}^{n+1}}{\Delta y^2} = RHS_{ij}^{n}$$

Where our RHS (right hand side) expression is:

$$RHS_{ij}^{n} - \frac{D_{ij}^{n+1} - D_{ij}^{n}}{\Delta t} - \frac{(u^2)_{i+1,j}^{n} - 2(u^2)_{i,j}^{n} + (u^2)_{i-1,j}^{n}}{\Delta x^2} - \frac{(v^2)_{i,j+1}^{n} - 2(v^2)_{i,j}^{n} + (v^2)_{i,j-1}^{n}}{\Delta y^2} + \frac{1}{Re}\left(\frac{D_{i+1,j}^{n} - 2D_{ij}^{n} + D_{i-1,j}^{n}}{\Delta x^2} + \frac{D_{i,j+1}^{n} - 2D_{ij}^{n} + D_{i,j-1}^{n}}{\Delta y^2}\right) - 2\frac{(uv)_{i+1,j+1}^{n} - (uv)_{i-1,j+1}^{n} - (uv)_{i+1,j-1}^{n} + (uv)_{i-1,j-1}^{n}}{4\Delta x \Delta y}$$

Define the following constant:

$$\beta = \frac{\Delta x}{\Delta y}$$

Rewriting the equation in the form of Line Gauss Seidel:

$$P_{i+1,j}^{n+1} - 2(1+\beta^2)P_{ij}^{n+1} + P_{i-1,j}^{n+1} = -\beta^2 P_{i,j+1}^{n} - -\beta^2 P_{i,j-1}^{n+1} - \Delta x^2 * RHS_{ij}$$

*Equation 5- Line Gauss Seidel*

Introduce a relaxation parameter, w, turns the Line Gauss Seidel into Line Successive Over Relaxation:

$$wP_{i+1,j}^{n+1} - 2(1+\beta^2)P_{ij}^{n+1} + wP_{i-1,j}^{n+1} = -(1-w)[2(1+\beta^2]P_{ij}^{n} - w\beta^2(P_{i,j+1}^{n} + P_{i,j-1}^{n+1}) - \Delta x^2 * RHS_{ij}$$

*Equation 6- LSOR Scheme*

For the dilatation term, we set $D^{n+1}{}_{ij} = 0$

### 3. Scheme for the Dilatation Equation

a. For Interior Nodes (running from 2 to imax-1, and 2 to jmax-1). Central difference in both u and v

$$D_{ij}^{n} = \frac{u_{i+1,j}^{n} - u_{i-1,j}^{n}}{2\Delta x} + \frac{v_{i,j+1}^{n} - v_{i,j-1}^{n}}{2\Delta y}$$

*Equation 7- Dilatation Interior*

b. Walls (do not include corners)
   i. Left Wall (i = 2, all j ). Forward difference in in u, central difference in v

$$D_{ij}^{n} = \frac{-3u_{i,j}^{n} + 4u_{i+1,j}^{n} - u_{i+2,j}^{n}}{2\Delta x} + \frac{v_{i,j+1}^{n} - v_{i,j-1}^{n}}{2\Delta y}$$

*Equation 8- Dilation Left Wall*

ii. Right Wall (i = imax, all j). Backward difference in u, central difference in v

$$D_{ij}^n = \frac{3u_{i,j}^n - 4u_{i-1,j}^n + u_{i-2,j}^n}{2\Delta x} + \frac{v_{i,j+1}^n - v_{i,j-1}^n}{2\Delta y}$$

*Equation 9- Dilatation Right Wall*

iii. Bottom Wall (all i, j = jmax). Backward difference in v, central difference in u

$$D_{ij}^n = \frac{u_{i+1,j}^n - u_{i-1,j}^n}{2\Delta x} + \frac{3v_{i,j}^n - 4v_{i,j-1}^n + v_{i,j-2}^n}{2\Delta y}$$

*Equation 10- Dilatation Bottom Wall*

iv. Top Wall (all i, j = 2). Forward difference in v, central difference in u.

$$D_{ij}^n = \frac{u_{i+1,j}^n - u_{i-1,j}^n}{2\Delta x} + \frac{-3v_{i,j}^n + 4v_{i,j+1}^n - v_{i,j+2}^n}{2\Delta y}$$

*Equation 11- Dilatation Top Wall*

c. Corners

i. Top Left(i = 1, j = 1). Forward in u and v.

$$D_{11}^n = \frac{-3u_{11}^n + 4u_{21}^n - u_{31}^n}{2\Delta x} + \frac{-3v_{11}^n + 4v_{12}^n - v_{13}^n}{2\Delta y}$$

*Equation 12- Dilatation Top Left*

ii. Top Right (i = imax, j = 1). Backward in u, forward in v

$$D_{imax,1}^n = \frac{3u_{imax,1}^n - 4u_{imax-1,1}^n + u_{imax-2,1}^n}{2\Delta x} + \frac{-3v_{imax,1}^n + 4v_{imax,2}^n - v_{imax,3}^n}{2\Delta y}$$

*Equation 13- Dilatation Top Right*

iii. Bottom Left (i = 1, j = jmax). Forward in u, backward in v

$$D_{1,jmax}^n = \frac{-3u_{1jmax}^n + 4u_{2,jmax}^n - u_{3,jmax}^n}{2\Delta x} + \frac{3v_{1,jmax}^n - 4v_{2,jmax-1}^n + v_{3,jmax-2}^n}{2\Delta y}$$

*Equation 14- Dilatation Bottom Left*

iv. Bottom Right (i = imax, j = jmax). Backward in u and v

$$D^n_{1,jmax} = \frac{3u^n_{imax,jmax} - 4u^n_{imax-1,jmax} + u^n_{imax-2,jmax}}{2\Delta x} + \frac{3v^n_{imax,jmax} - 4v^n_{imax,jmax-1} + v^n_{imax,jmax-2}}{2\Delta y}$$

*Equation 15- Dilatation Bottom Right*

## 4. Scheme for X and Y Momentum Equations

$$u^{n+1}_{ij} = u^n_{ij} + \Delta t * \left( \frac{1}{Re}\left( \frac{u^n_{i+1,j} - 2u^n_{ij} + u^n_{i-1,j}}{\Delta x^2} + \frac{u^n_{i,j+1} - 2u^n_{ij} + u^n_{i,j-1}}{\Delta y^2} \right) - \frac{1}{2}(1+\epsilon_x)\frac{(u^2)^n{}_{ij} - (u^2)^n{}_{i,-1j}}{\Delta x} - \frac{1}{2}(1-\epsilon_x)\frac{(u^2)^n{}_{i+1,j} - (u^2)^n{}_{i,j}}{\Delta x} - \frac{1}{2}(1+\epsilon_x)\frac{P^n{}_{ij} - P^n{}_{i-1,j}}{\Delta x} - \frac{1}{2}(1-\epsilon_x)\frac{P^n{}_{i+1,j} - P^n{}_{i,j}}{\Delta x} - \frac{(uv)^n{}_{i,j+1} - (uv)^n{}_{i,j-1}}{2\Delta y} \right)$$

*Equation 16- X Momentum*

$$v^{n+1}_{ij} = v^n_{ij} + \Delta t * \left( \frac{1}{Re}\left( \frac{v^n_{i+1,j} - 2v^n_{ij} + v^n_{i-1,j}}{\Delta x^2} + \frac{v^n_{i,j+1} - 2v^n_{ij} + v^n_{i,j-1}}{\Delta y^2} \right) - \frac{1}{2}(1+\epsilon_y)\frac{(v^2)^n{}_{ij} - (v^2)^n{}_{i,j-1}}{\Delta y} - \frac{1}{2}(1-\epsilon_y)\frac{(v^2)^n{}_{i,j+1} - (v^2)^n{}_{i,j}}{\Delta y} - \frac{1}{2}(1+\epsilon_y)\frac{P^n{}_{ij} - P^n{}_{i,j-1}}{\Delta y} - \frac{1}{2}(1-\epsilon_y)\frac{P^n{}_{i,j+1} - P^n{}_{i,j}}{\Delta y} - \frac{(uv)^n{}_{i+1,j} - (uv)^n{}_{i-1,j}}{2\Delta x} \right)$$

*Equation 17-Y Momentum*

Where the following conditions are embedded:

$$If\ u_{ij} > 0 \rightarrow \epsilon_x = 1 \rightarrow Use\ backward\ difference$$

$$Else, \epsilon_x = -1 \rightarrow Use\ forward\ difference$$

$$If\ v_{ij} > 0 \rightarrow \epsilon_y = 1 \rightarrow Use\ backward\ difference$$
$$Else, \epsilon_y = -1 \rightarrow Use\ forward\ difference$$

*Equation 18- Switch Statements*

## 5. Plots

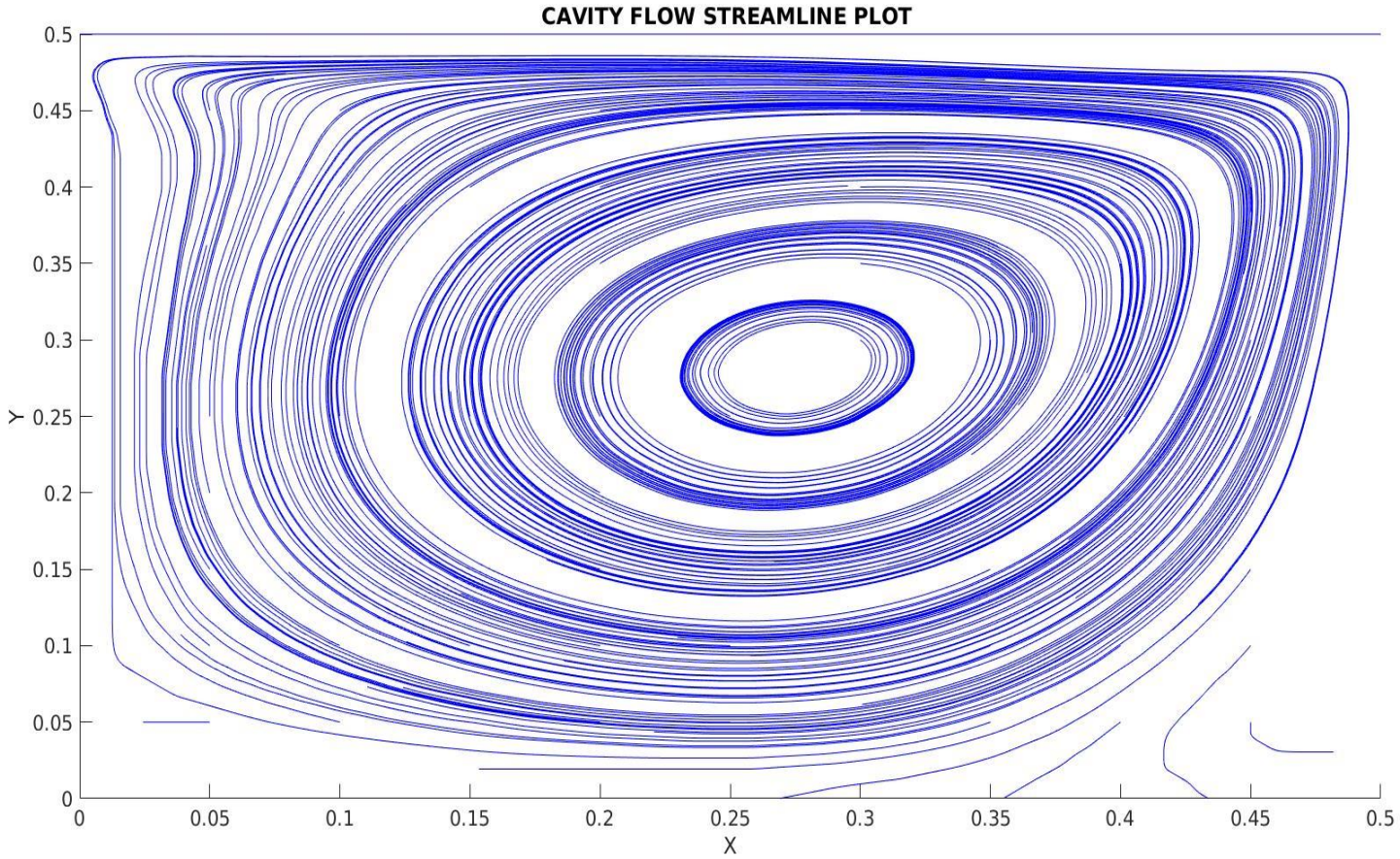



*Figure 1- Streamline Plot*

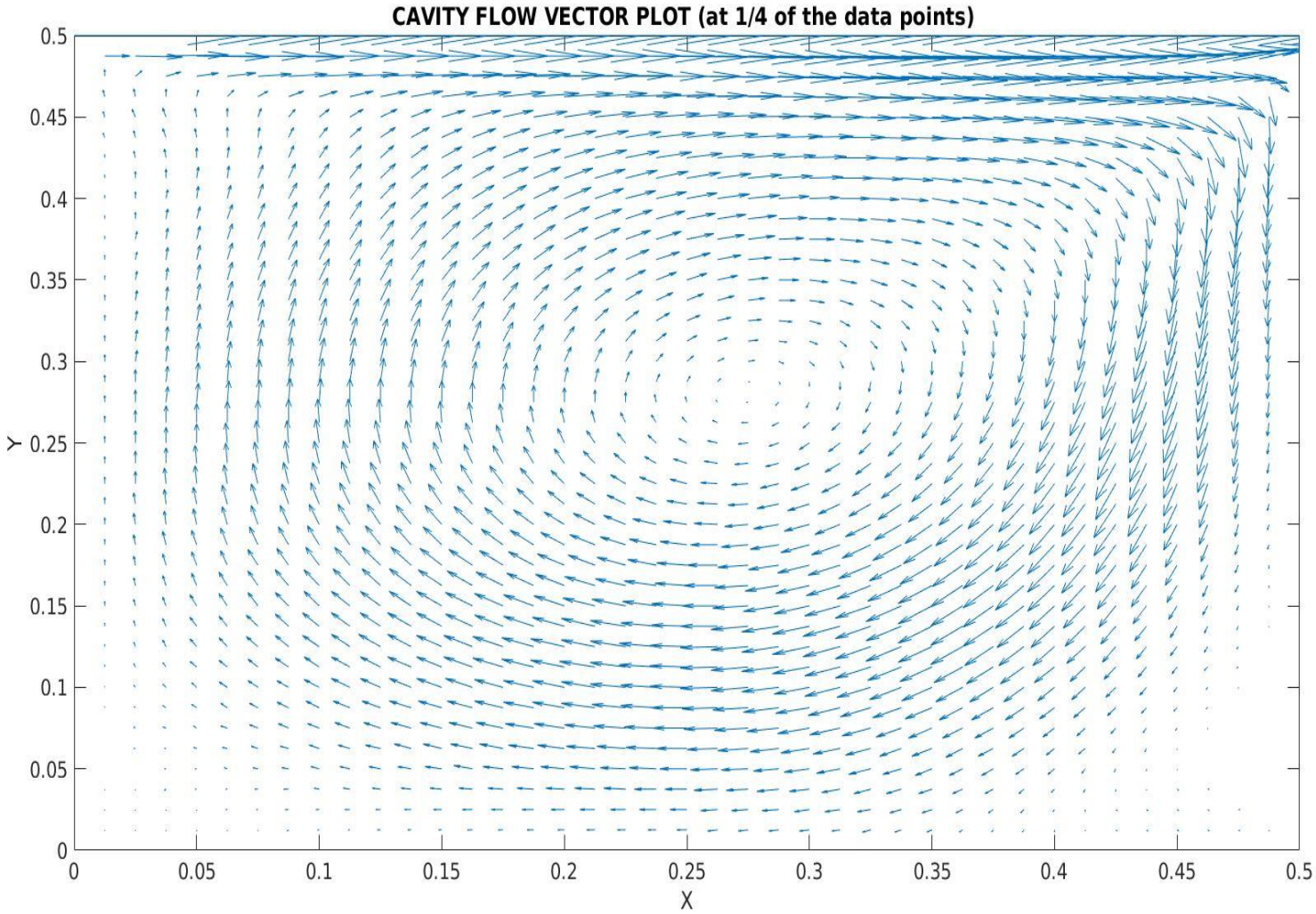



*Figure 2- Vector Plot*

# APPENDIX A

C++ CODES

```
#include "cavityfuncs.h"
using namespace std;

NavierStoke cavity;
//Main code
int main(){

        Initialize(&cavity);
        int tmax=15000;

  for (int time = 0; time<=tmax ;time++){
                Compute_Dilatation(&cavity); //get D
                LineSOR(&cavity);// get P
                MomentumSolver(&cavity); //get U and V
                printf("t = %d LSOR %.6f \t iteration = %d\n",time,cavity.error,
cavity.convergenceIter);
        }


  FILE *outfile1;
  outfile1 = fopen("Yew_results_15000.dat","w");
        for (int j = 1;j<=cavity.jmax;j++){
                for (int i = 1; i<=cavity.imax;i++){
                        fprintf(outfile1,"%.2f\t",cavity.u_updated[j][i]);
                }
                fprintf(outfile1,"\n");
        }

  FILE *outfile2;
  outfile2 = fopen("Vee_results_15000.dat","w");
        for (int j = 1;j<=cavity.jmax;j++){
                for (int i = 1; i<=cavity.imax;i++){
                        fprintf(outfile2,"%.2f\t",cavity.v_updated[j][i]);
                }
                fprintf(outfile2,"\n");
        }

  FILE *outfile3;
  outfile3 = fopen("Pressure_results_15000.dat","w");
        for (int j = 1;j<=cavity.jmax;j++){
                for (int i = 1; i<=cavity.imax;i++){
                        fprintf(outfile3,"%.2f\t",cavity.p_updated[j][i]);
                }
                fprintf(outfile3,"\n");
        }

  FILE *outfile4;
  outfile4 = fopen("Dilation.dat","w");
        for (int j = 1;j<=cavity.jmax;j++){
                for (int i = 1; i<=cavity.imax;i++){
                        fprintf(outfile4,"%.2f\t",cavity.dilation_N[j][i]);
                }
                fprintf(outfile4,"\n");
        }

        fcloseall;
        return 0;
}
```

```
#include <iostream>
#include <stdio.h>
#include <cmath>
#include <fstream>
#include <vector>
using namespace std;

struct NavierStoke{
        double dx;
        double dy;
        double Re; //Reynolds number
        double p0; //constant pressure for top and bottom surface
        double u0; //top velocity of top surface
        int tmax; //max time
        int imax;
        int jmax;
  int convergenceIter;
  double nmax; //max time step
        double dt;
        double w_optimal; //omega for LineSOR
        double errormax;//error max for SOR
        double error;
        int lsormax;
        vector< vector<double> > u; //U velocity component
        vector< vector<double> > v; //V velocity component
        vector< vector<double> > u_updated; //Holder for U velocity component
        vector< vector<double> > v_updated; //Holder for V velocity component
        vector< vector<double> > p; //Pressure field
        vector< vector<double> > p_updated; //Pressure field holder
        vector< vector<double> > dilation_N; //ensure continuity at N

};


void Initialize(struct NavierStoke *cavity){

        //DECLARE BASIC PARAMETERS
        cavity->dx = 0.00625;
        cavity->dy = 0.00625;
        cavity->Re = 1000;
  cavity->nmax = 15000;
        cavity->p0 = 3350.;
        cavity->u0 = 1.;
        cavity->imax = 81;
        cavity->jmax = 81;
        cavity->dt = 0.003;
  cavity->tmax = 10;
        // cavity->tmax = cavity->nmax*cavity->dt;
        cavity->w_optimal = 1.315;
        cavity->errormax = 0.0001;
        cavity->lsormax = 100000000;

        //DECLARE FIELDS

        cavity->u.resize(cavity->jmax+1);
        cavity->u_updated.resize(cavity->jmax+1);
        cavity->v.resize(cavity->jmax+1);
        cavity->v_updated.resize(cavity->jmax+1);
        cavity->p.resize(cavity->jmax+1);
        cavity->p_updated.resize(cavity->jmax+1);
        cavity->dilation_N.resize(cavity->jmax+1);
```

```
        //INITIALIZE VECTORS
        for (int j = 1; j<=cavity->jmax;j++){
                cavity->u[j].resize(cavity->imax+1);
                cavity->u_updated[j].resize(cavity->imax+1);
                cavity->v[j].resize(cavity->imax+1);
                cavity->v_updated[j].resize(cavity->imax+1);
                cavity->p[j].resize(cavity->imax+1);
                cavity->p_updated[j].resize(cavity->imax+1);
                cavity->dilation_N[j].resize(cavity->imax+1);
        }


        //zeros out all fields
        for (int j = 1; j<=cavity->jmax;j++){
                for (int i = 1; i<=cavity->imax;i++){
                        cavity->u[j][i] = 0.0;
                        cavity->u_updated[j][i] = 0.0;
                        cavity->v[j][i] = 0.0;
                        cavity->v_updated[j][i] = 0.0;
                        cavity->p[j][i] = 0.0;
                        cavity->p_updated[j][i] = 0.0;
                        cavity->dilation_N[j][i] = 0.0;
                }

        }

        int jmax = cavity->jmax;
        //INITIAL CONDITIONS
        for (int i = 1; i<=cavity->imax;i++){
                cavity->u_updated[jmax][i] = cavity->u0; //top layer
                cavity->u[jmax][i] = cavity->u0; //top layer

                cavity->p_updated[1][i] = cavity->p0;//top
                cavity->p_updated[cavity->jmax][i] = cavity->p0; //bottom
                cavity->p[1][i] = cavity->p0; //top
                cavity->p[cavity->jmax][i] = cavity->p0; //bottom

        }

        cavity->p_updated = cavity->p;
}

//Thomas Algorithm
void thomasTriDiagonal(int j, double a[],double b[],double c[], double d[], struct
NavierStoke *cavity){

        int imax = cavity->imax;
        double dprime[imax+1];
        double cprime[imax+1];
        dprime[2] = d[2];
        cprime[2] = c[2];

        //FORWARD LOOP
        for (int i = 3; i<=imax-1;i++){
                dprime[i] = d[i] - ((b[i]*a[i-1])/(dprime[i-1]));
                cprime[i] = c[i] - ((cprime[i-1]*b[i])/(dprime[i-1]));
        }

        cavity->p_updated[j][imax-1] = cprime[imax-1]/dprime[imax-1];

        //BACKWARD LOOP
        for (int i = imax-2;i>=2;i--){
```

```
                cavity->p_updated[j][i] = (cprime[i]-(a[i]*cavity->p_updated[j][i
+1]))/(dprime[i]);
        }
}

//CALCULATE DILATATION

void Compute_Dilatation(struct NavierStoke *cavity){
        int imax =      cavity->imax;
        int jmax = cavity->jmax;

        double dx = cavity->dx;
        double dy = cavity->dy;

        // INTERIOR NODES
        // ALL 2ND CENTRAL DIFF

        for (int j = 2; j<=jmax-1;j++){
                for (int i = 2; i<=imax-1;i++){
                        cavity->dilation_N[j][i] = (
                    (cavity->u[j][i+1]-
                    cavity->u[j][i-1])/(2.*dx)
                )
                +
                            (
                    (cavity->v[j+1][i]-
                    cavity->v[j-1][i])/(2.*dy)
                );
                        }
        }

        for (int j = 2; j<=jmax-1;j++){
                //LEFT WALL
                //FORWARD IN U, CENTRAL IN V

                cavity->dilation_N[j][1] = (
                (
                    -cavity->u[j][3]+
                    4.*cavity->u[j][2]-
                    3.*cavity->u[j][1]
                )/(2.*dx)
            )+
            (
                (
                    cavity->v[j+1][1]-
                    cavity->v[j-1][1]
                )/(2.*dy)
            );

                //RIGHT WALL
                //BACKWARD IN U, CENTRAL IN V
                cavity->dilation_N[j][imax] = (
                (
                    cavity->u[j][imax-2]-
                    4.*cavity->u[j][imax-1]+
                    3.*cavity->u[j][imax]
                )/(2.*dx)
            )+
                        (
                (
                    cavity->v[j+1][imax]-
                    cavity->v[j-1][imax]
```

```
                )/(2.*dy)
            );
        }

        for (int i = 2; i<=imax-1;i++){
                //TOP WALL
                //CENTRAL IN U, FORWARD IN V

                cavity->dilation_N[1][i] = (
            (
                cavity->u[1][i+1]-
                cavity->u[1][i-1]
            )/(2.*dx)
        )+
                (
            (
                -cavity->v[3][i]+
                4.*cavity->v[2][i]-
                3.*cavity->v[1][i]
            )/(2.*dy)
        );

                //BOTTOM WALL
                //CENTRAL IN U, BACKWARD IN V
                cavity->dilation_N[jmax][i] = (
            (
                cavity->u[jmax][i+1]-
                cavity->u[jmax][i-1]
            )/(2.*dx)
        )+
                (
            (
                cavity->v[jmax-2][i]-
                4.*cavity->v[jmax-1][i]+
                3.*cavity->v[jmax][i]
            )/(2.*dy)
        );
        }
        //CORNERS

        //TOP LEFT
        //FORWARD IN U, FORWARD IN V
        cavity->dilation_N[1][1] = ((-cavity->u[1][3]+4.*cavity->u[1][2]-3.*cavity->u
[1][1])/(2.*dx))+
        ((-cavity->v[3][1]+4.*cavity->v[2][1]-3.*cavity->v[1][1])/(2.*dx));

        //TOP RIGHT
        //BACKWARD IN U, FORWARD IN V

        cavity->dilation_N[1][imax] = ((cavity->u[1][imax-2]-4.*cavity->u[1][imax-1]
+3.*cavity->u[1][imax])/(2.*dx))+
        ((-cavity->v[3][imax]+4.*cavity->v[2][imax]-3.*cavity->v[1][imax])/(2.*dx));

        //BOTTOM LEFT
        //FORWARD IN U, BACKWARD IN V
        cavity->dilation_N[jmax][1] = ((-cavity->u[jmax][3]+4.*cavity->u[jmax]
[2]-3.*cavity->u[jmax][1])/(2.*dx))+
        ((cavity->v[jmax-2][1]-4.*cavity->v[jmax-1][1]+3.*cavity->v[jmax][1])/
(2.*dy));

        //BOTTOM RIGHT
        //BACKWARD IN U, BACKWARD IN V
```

```
        cavity->dilation_N[jmax][imax] = ((cavity->u[jmax][imax-2]-4.*cavity->u[jmax]
[imax-1]+3.*cavity->u[jmax][imax])/(2.*dx))+((cavity->v[jmax-2][imax]-4.*cavity->v
[jmax-1][imax]+3.*cavity->v[jmax][imax])/(2.*dy));

}



//LineSOR- SOLVING THE PRESSURE POISSON EQUATION
void LineSOR(struct NavierStoke *cavity){
        double beta = cavity->dx/cavity->dy;
        double BetaSquare = pow(beta,2.);
        int imax = cavity->imax;
        int jmax = cavity->jmax;


        //THOMAS PARAMETERS
        double ax[imax+1]; //above
        double bx[imax+1]; //below
        double cx[imax+1];//rhs
        double diagonalX[imax+1];//diagonal
        double dx = cavity->dx;
        double dy = cavity->dy;
        double dt = cavity->dt;

        //Terms for the RHS

        double first;
        double second;
        double third;
        double fourth;
        double fifth;
        double sixth;
        double seventh;


        for(int lsoriter=0; lsoriter <= cavity->lsormax; lsoriter++) {
                cavity->error = 0.0;

                //loop through jTH
                for (int  j = 2; j<=jmax-1;j++){
                        //loop through iTH
                        for (int i = 2; i<=imax-1;i++){

                                //SET UP THOMAS VECTORS

                                        //A- thomas
                                        ax[i] = cavity->w_optimal;

                                        //B-thomas
                                        bx[i] = cavity->w_optimal;

                                        //D-thomas
                                        diagonalX[i] = -2.*(1.+pow(beta,2.));

                                        //C - VECTORS
                                        first = -(1.-cavity->w_optimal)*(
                                    2.*(1+pow(beta,2.))
                                )*cavity->p[j][i];

                                        second = (cavity->w_optimal*BetaSquare)*
```

```
(cavity->p[j+1][i]+cavity->p_updated[j-1][i]);

                                        third = (cavity->dilation_N[j][i])/dt;

                                        fourth = (
                        pow((cavity->u[j][i+1]),2.)-
                        2.*pow((cavity->u[j][i]),2.)+
                        pow((cavity->u[j][i-1]),2.)
                    )/(pow(dx,2.));

                                        fifth =  (
                        pow((cavity->v[j+1][i]),2.)-
                        2.*pow((cavity->v[j][i]),2.)+
                        pow((cavity->v[j-1][i]),2.)
                    )/(pow(dy,2.));

                                        sixth = (1./cavity->Re)*(
                        (
                            (cavity->dilation_N[j][i+1]-
                            2.*cavity->dilation_N[j][i]+
                            cavity->dilation_N[j][i-1]
                            )/(pow(dx,2.))
                        )+(
                            (cavity->dilation_N[j+1][i]-
                            2.*cavity->dilation_N[j][i]+
                            cavity->dilation_N[j-1][i]
                            )/(pow(dy,2.))
                        )
                    );

                                        seventh = (
                        (
                            (cavity->u[j+1][i+1]*cavity->v[j+1][i+1])+
                            (cavity->u[j-1][i-1]*cavity->v[j-1][i-1])-
                            (cavity->u[j+1][i-1]*cavity->v[j+1][i-1])-
                            (cavity->u[j-1][i+1]*cavity->v[j-1][i+1])
                        )
                    )/(2.*dx*dy);

                    // printf("%.3f %.3f %.3f %.3f %.3f %.3f %.3f \n", first, second,
third, fourth, fifth, sixth, seventh);

                                        cx[i] = (first-second)+pow(dx,2.)*(third-
fourth-fifth+sixth-seventh);
                        }
                        ax[1] = 0.;
                        bx[1] = 0.;
                        ax[imax] = 0.;
                        bx[imax] = 0.;
                        diagonalX[1] = 0.;
                        diagonalX[imax] = 0.;
                        cx[1] = 0.;
                        cx[imax] = 0.;

                        //boundary conditions
                        ax[2] = cavity->w_optimal;
                        diagonalX[2] = -2.*(1.+pow(beta,2.)) + cavity->w_optimal;
                        bx[2] = cavity->w_optimal;
                        bx[imax-1] = cavity->w_optimal;
                        diagonalX[imax-1] = -2.*(1.+pow(beta,2.)) + cavity->w_optimal;
                        ax[imax-1] = cavity->w_optimal;
```

```
                        //CALL THOMAS
                        thomasTriDiagonal(j,ax,bx,cx,diagonalX,cavity);

                        cavity->p_updated[j][1] = cavity->p_updated[j][2];
                        cavity->p_updated[j][imax] = cavity->p_updated[j][imax-1];

                        // CALCULATE ERROR
                        for (int i = 1; i<=imax;i++){
                                cavity->error += pow(cavity->p_updated[j][i]-cavity->p
[j][i],2.);

                        }

                }
                cavity->error = sqrt(cavity->error);
                cavity->p = cavity->p_updated;

                //UPDATE ITERATION COUNTER

                lsoriter = lsoriter +1;
    // printf("%d\n",lsoriter);
    if(cavity->error <= cavity->errormax){

                        break;
                }
    cavity->convergenceIter = lsoriter;
        }
}



//MAIN MOMENTUM SOLVER-> GET U AND V
void MomentumSolver(struct NavierStoke *cavity){
        int imax = cavity->imax;
        int jmax = cavity->jmax;
        double dt = cavity->dt;
        double dx = cavity->dx;
        double dy = cavity->dy;
        double reynolds = cavity->Re;
        double epsilonX; //ex for U equation
        double epsilonY; //ey for V equation

        double dsqudx, dsqudy, dusqdxp, dusqdxm, dpdxp, dpdxm;
        double duvdy, delsqu, dudt;

        double dsqvdx, dsqvdy, dvsqdyp, dvsqdym, dpdyp, dpdym;
        double duvdx, delsqv, dvdt;

// CALCULATE U
        for (int j = 2; j<=jmax-1;j++){
                for (int i = 2; i<=imax-1;i++){

                        if (cavity->u[j][i]>=0){
                                epsilonX = 1.;
                        }else {
                                epsilonX = -1.;
                        }

                        dsqudx = (cavity->u[j][i+1]-
```

```
                                                2.*cavity->u[j][i]
                                                +cavity->u[j][i-1]
                                        )/(pow(dx,2.));
                dsqudy = (cavity->u[j+1][i]-
                                                2.*cavity->u[j][i]
                                                +cavity->u[j-1][i]
                                        )/(pow(dy,2.));
                delsqu = dsqudx + dsqudy;

                dusqdxm = (
                        pow(cavity->u[j][i],2.)-
                                pow(cavity->u[j][i-1],2.)
                        )/dx;
                dusqdxp = (
                        pow(cavity->u[j][i+1],2.)-
                                pow(cavity->u[j][i],2.)
                        )/dx;

                dpdxm = (
                        cavity->p[j][i]-
                        cavity->p[j][i-1]
                )/dx;
                dpdxp = (
                        cavity->p[j][i+1]-
                        cavity->p[j][i]
                )/dx;
                duvdy = (
                        (cavity->u[j+1][i]*cavity->v[j+1][i])-
                        (cavity->u[j-1][i]*cavity->v[j-1][i])
                )/(2.*dy);

                dudt = (1./reynolds)*delsqu-
                        0.5*(1.+epsilonX)*dusqdxm-
                        0.5*(1.-epsilonX)*dusqdxp-
                        0.5*(1.+epsilonX)*dpdxm-
                        0.5*(1.-epsilonX)*dpdxp-
                        duvdy;
                cavity->u_updated[j][i] = cavity->u[j][i]+dt*dudt;
        }
}



//CALCULATE V

for (int j = 2; j<=jmax-1;j++){
        for (int i = 2; i<=imax-1;i++){
                if (cavity->v[j][i]>=0){
                        epsilonY = 1.;
                }

                else{

                        epsilonY = -1.;
                }

                dsqvdx = (cavity->v[j][i+1]-
                                                2.*cavity->v[j][i]
                                                +cavity->v[j][i-1]
                                        )/(pow(dx,2.));
                dsqvdy = (cavity->v[j+1][i]-
                                                2.*cavity->v[j][i]
```

```
                                                        +cavity->v[j-1][i]
                                                )/(pow(dy,2.));
                        delsqv = dsqvdx + dsqvdy;

                        dvsqdym = (
                                pow(cavity->v[j][i],2.)-
                                        pow(cavity->v[j-1][i],2.)
                                )/dx;
                        dvsqdyp = (
                                pow(cavity->v[j+1][i],2.)-
                                        pow(cavity->v[j][i],2.)
                                )/dy;

                        dpdym = (
                                cavity->p[j][i]-
                                cavity->p[j-1][i]
                        )/dy;
                        dpdyp = (
                                cavity->p[j+1][i]-
                                cavity->p[j][i]
                        )/dy;
                        duvdx = (
                                (cavity->u[j][i+1]*cavity->v[j][i+1])-
                                (cavity->u[j][i-1]*cavity->v[j][i-1])
                        )/(2.*dx);

            dvdt = (1./reynolds)*delsqv
                                -0.5*(1.+epsilonY)*dvsqdym
                                -0.5*(1.-epsilonY)*dvsqdyp
                                -0.5*(1.+epsilonY)*dpdym
                                -0.5*(1.-epsilonY)*dpdyp
                                -duvdx;

                        cavity->v_updated[j][i] = cavity->v[j][i]+dt*dvdt;
                }
        }

        // FORCING
        for (int j = 1; j<=jmax;j++){
                cavity->u_updated[j][1] = 0.0; //left wall
                cavity->u_updated[j][imax] = 0.0; //right wall
                cavity->v_updated[j][1] = 0.0; //left wall
                cavity->v_updated[j][imax] = 0.0; //right wall

        }

        for (int i = 1; i<=imax;i++){
                cavity->u_updated[jmax][i] = cavity->u0; //bottom wall
                cavity->v_updated[jmax][i] = 0.0; //bottom wall

                cavity->u_updated[1][i] = 0.0;
                cavity->v_updated[1][i] = 0.0;

        }

        // Update U and V
        cavity->u = cavity->u_updated;
        cavity->v = cavity->v_updated;
}
```

# APPENDIX B

MATLAB PLOTTING CODE

```
clear all
clc
clf

filename1 = 'Yew_results_15000.dat';
U =importdata(filename1);

filename2 = 'Vee_results_15000.dat';
V =importdata(filename2);

filename3 = 'Pressure_results_15000.dat';
P =importdata(filename3);

sx = [0.0:0.05:0.5];
sy = [0.0:0.05:0.5];
[SX, SY] = meshgrid(sx,sy);

step_size = 0.1;
max_vertices = 2500;



dx = 0.00625;
nx = 81;
l = dx*(nx-1);

[x,y] = meshgrid(0:dx:0.5,0:dx:0.5);


%plotting
figure(1)
streamline(x,y,U,V,SX,SY,[step_size,max_vertices]);
xlim([0 0.5])
ylim([0 0.5])
title('CAVITY FLOW STREAMLINE PLOT ','FontSize',15)
xlabel('X','FontSize',15);
ylabel('Y','FontSize',15)
xt = get(gca,'XTick');
set(gca, 'FontSize', 16);




k = 2;
figure(2)
hq = quiver(x(1:k:end,1:k:end),y(1:k:end,1:k:end),U(1:k:end,1:k:end),V
(1:k:end,1:k:end),4);
xlim([0 0.5])
ylim([0 0.5])
title('CAVITY FLOW VECTOR PLOT (at 1/4 of the data points) ','FontSize',15)
xlabel('X','FontSize',15);
ylabel('Y','FontSize',15)
xt = get(gca,'XTick');
set(gca, 'FontSize', 16);
```