CE-580

COMPUTATIONAL TECHNIQUES

FOR

FLUID DYNAMICS

HOMEWORK #7

Vorticity-Stream Function Solution to Driven Cavity Flow

Taha Yaşar Demir

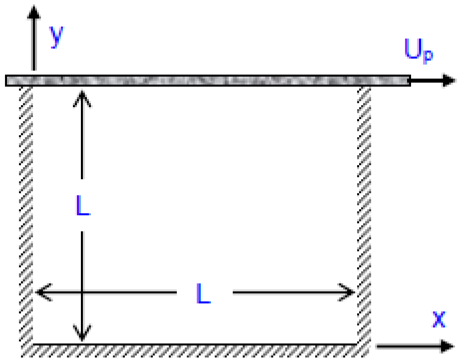
1881978

CE 580 COMPUTATIONAL TECHNIQUES FOR FLUID DYNAMICS

Homework - 7

Vorticity-Stream Function Solution to Driven Cavity Flow

An incompressible fluid is contained in a 2-D square cavity as shown in the figure. The plate on the upper face is moved horizontally at a constant speed U_p. The laminar vortex motion driven by the moving plate is to be computed by solving the Navier-Stokes equations numerically.



1. Write down the vorticity-stream function formulation of the governing equations.
2. Obtain finite-difference equations for vorticity transport equation with first order upwind differences for convective terms. Use constant mesh size.
3. Use ADI method for solution of the vorticity transport equation.
4. Use PSOR method for solution of the Poisson equation for stream function.
5. Define the boundary conditions for vorticity and stream function and indicate their numerical implementation.
6. Define an initial data for vorticity and stream function.
7. Discuss the stability conditions if required.
8. Define an overall error and the condition of convergence.
9. Obtain the solution using 101X101 nodal points for the data given:

$$L = 0.01 \text{ m}, \quad \nu = 1\text{X}10^{-6} \text{ m}^2/\text{s}$$
$$U_p = 0.01, 0.02, 0.05, 0.1, 1.0 \text{ m/s}$$

10. Make a contour plot of streamlines for each case.
11. Compute the Reynolds number of the cavity flow for each solution, $R_e = U_p L/\nu$.
12. Determine the maximum stream function value at the core of the main vortex.
13. Determine the drag force (per unit width) on the moving plate.
14. Report your results in a table and write a discussion of results.

Governing Equations

For a 2-D incompressible, viscous flow in x-y

Vorticity vector written as $\zeta = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$ and vorticity transport equation as

$$\frac{\partial \zeta}{\partial t} + u\frac{\partial \zeta}{\partial x} + v\frac{\partial \zeta}{\partial y} = \nu\left(\frac{\partial^2 \zeta}{\partial x^2} + \frac{\partial^2 \zeta}{\partial y^2}\right)$$

Or in conservation form

$$\frac{\partial \zeta}{\partial t} + \frac{\partial (u\zeta)}{\partial x} + \frac{\partial (v\zeta)}{\partial y} = \nu\left(\frac{\partial^2 \zeta}{\partial x^2} + \frac{\partial^2 \zeta}{\partial y^2}\right)$$

Velocity components can be written in terms of stream function

$$u = \frac{\partial \psi}{\partial y}\,, \qquad v = -\frac{\partial \psi}{\partial x}$$

Poisson equation for stream function can be obtained from the vorticity component in x-y plane

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\zeta$$

Now we have one parabolic and one elliptic equation to solve

Discretization

Vorticity transport equation is discretized with First-Order Upwind method. This method is stable and dissipative but introduces artificial viscosity. Thus, a fine mesh should be used.

$$\frac{\zeta_{i,j}^{n+1} - \zeta_{i,j}^{n}}{\Delta t} + \frac{(1-\epsilon_x)}{2}\left[\frac{(u\zeta)_{i+1,j}^{n} - (u\zeta)_{i,j}^{n}}{\Delta x}\right] + \frac{(1+\epsilon_x)}{2}\left[\frac{(u\zeta)_{i,j}^{n} - (u\zeta)_{i-1,j}^{n}}{\Delta x}\right]$$
$$+ \frac{(1-\epsilon_y)}{2}\left[\frac{(v\zeta)_{i,j+1}^{n} - (v\zeta)_{i,j'}^{n}}{\Delta y}\right] + \frac{(1+\epsilon_y)}{2}\left[\frac{(v\zeta)_{i,j}^{n} - (v\zeta)_{i,j-1}^{n}}{\Delta y}\right]$$
$$= \nu\left[\frac{\zeta_{i+1,j}^{n} - 2\zeta_{i,j}^{n} + \zeta_{i-1,j}^{n}}{(\Delta x)^2} + \frac{\zeta_{i,j+1}^{n} - 2\zeta_{i,j}^{n} + \zeta_{i,j-1}^{n}}{(\Delta y)^2}\right]$$

If $u_{i,j} > 0$, backward difference must be used Thus, $\epsilon_x = 1$

If $u_{i,j} < 0$, forward difference must be used Thus, $\epsilon_x = -1$

If $v_{i,j} > 0$, backward difference must be used Thus, $\epsilon_y = 1$

If $v_{i,j} > 0$, backward difference must be used Thus, $\epsilon_y = -1$

If $\epsilon_x = \epsilon_y = 0$,second-order central differences are recovered

Poisson equation for stream function is discretized using second order central differences and yields FDE such as

$$\frac{\psi_{i41,j} - 2\psi_{i,j} + \psi_{i-1,j}}{(\Delta x)^2} + \frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i-1,j}}{(\Delta y)^2} = \zeta_{i,j}$$

Solution Method

Discretized vorticity transport equation is solved using ADI method which is an implicit and fast method. Also, equation is linearized by lagging u and v velocity components one-time step behind.

x-sweep:

$$\frac{\zeta_{i,j}^{n+\frac{1}{2}} - \zeta_{i,j}^{n}}{\frac{\Delta t}{2}} + \frac{(1-\epsilon_x)}{2}\left[\frac{u_{i-1,j}^{n}\zeta_{i+1,j}^{n+\frac{1}{2}} - u_{i,j}^{n}\zeta_{i,j}^{n+\frac{1}{2}}}{\Delta x}\right] + \frac{(1+\epsilon_x)}{2}\left[\frac{u_{i,j}^{n}\zeta_{i,j}^{n+\frac{1}{2}} - u_{i-1,j}^{n}\zeta_{i-1,j}^{n+\frac{1}{2}}}{\Delta x}\right] + \frac{(1-\epsilon_y)}{2}\left[\frac{u_{i,j}^{n}\,\zeta_{i,j+1}^{n} - u_{i,j}^{n}\zeta_{i,j}^{n}}{\Delta y}\right] + \frac{(1+\epsilon_y)}{2}\left[\frac{u_{i,j}^{n}\zeta_{i,j}^{n} - u_{i,j-1}^{n}\zeta_{i,j-1}^{n}}{\Delta y}\right] = \nu\left[\frac{\zeta_{i+1,j}^{n+\frac{1}{2}} - 2\zeta_{i,j}^{n+\frac{1}{2}} + \zeta_{i-1,j}^{n+\frac{1}{2}}}{(\Delta x)^2} + \frac{\zeta_{i,j+1}^{n} - 2\zeta_{i,j}^{n} + \zeta_{i,j-1}^{n}}{(\Delta y)^2}\right]$$

y-sweep:

$$\frac{\zeta_{i,j}^{n+1} - \zeta_{i,j}^{n+\frac{1}{2}}}{\frac{\Delta t}{2}} + \frac{(1-\epsilon_x)}{2}\left[\frac{u_{i-1,j}^{n}\zeta_{i+1,j}^{n+\frac{1}{2}} - u_{i,j}^{n}\zeta_{i,j}^{n+\frac{1}{2}}}{\Delta x}\right] + \frac{(1+\epsilon_x)}{2}\left[\frac{u_{i,j}^{n}\zeta_{i,j}^{n+\frac{1}{2}} - u_{i-1,j}^{n}\zeta_{i-1,j}^{n+\frac{1}{2}}}{\Delta x}\right] + \frac{(1-\epsilon_y)}{2}\left[\frac{u_{i,j}^{n}\,\zeta_{i,j+1}^{n+1} - u_{i,j}^{n}\zeta_{i,j}^{n+1}}{\Delta y}\right] + \frac{(1+\epsilon_y)}{2}\left[\frac{u_{i,j}^{n}\zeta_{i,j}^{n+1} - u_{i,j-1}^{n}\zeta_{i,j-1}^{n+1}}{\Delta y}\right] = \nu\left[\frac{\zeta_{i+1,j}^{n+\frac{1}{2}} - 2\zeta_{i,j}^{n+\frac{1}{2}} + \zeta_{i-1,j}^{n+\frac{1}{2}}}{(\Delta x)^2} + \frac{\zeta_{i,j+1}^{n+1} - 2\zeta_{i,j}^{n+1} + \zeta_{i,j-1}^{n+1}}{(\Delta y)^2}\right]$$

Each equation has 3 unknowns and can be solved by Thomas algorithm. ADI steps in tridiagonal form

x-sweep:

$$-A_i\zeta_{i-1,j}^{n+\frac{1}{2}} + B_i\zeta_{i,j}^{n+\frac{1}{2}} - C_i\zeta_{i+1,j}^{n+\frac{1}{2}} = D_i$$

Coefficients A, B, C, D are

$$A_i = \frac{1}{2}\left[d_x + \frac{1}{2}(1+\epsilon_x)(c_{rx})_{i-1,j}\right]$$

$$B_i = 1 + d_x + \frac{1}{2}\epsilon_x (c_{rx})_{i,j}$$

$$C_i = \frac{1}{2}\left[d_x - \frac{1}{2}(1-\epsilon_x)(c_{rx})_{i+1,j}\right]$$

$$D_i = \frac{1}{2}\left[d_y - \frac{1}{2}(1-\epsilon_y)\left(c_{ry}\right)_{i,j+1}\right]\zeta_{i,j+1}^{n} + \left[1 - d_y - \frac{1}{2}\epsilon_y\left(c_{ry}\right)_{i,j}\right]\zeta_{i,j}^{n} + \frac{1}{2}\left[d_y + \frac{1}{2}(1+\epsilon_y)\left(c_{ry}\right)_{i,j-1}^{n}\right]\zeta_{i,j-1}^{n}$$

Where

$$(c_{rx})_{i,j} = u_{i,j}^{n}\frac{\Delta t}{\Delta x}\ , \qquad \left(c_{ry}\right)_{i,j} = v_{i,j}^{n}\frac{\Delta t}{\Delta y}\ , \qquad d_x = \frac{\nu \Delta t}{(\Delta x)^2}\ , \qquad d_y = \frac{\nu \Delta t}{(\Delta y)^2}$$

y-sweep:

$$-A_i\zeta_{i,j-1}^{n+1} + B_i\zeta_{i,j}^{n+1} - C_i\zeta_{i,j+1}^{n+1} = D_j$$

Coefficients A. B. C. D are

$$A_j = \frac{1}{2}\left[d_y + \frac{1}{2}(1+\epsilon_y)\left(c_{ry}\right)_{i,j-1}\right]$$

$$B_j = 1 + d_y + \frac{1}{2}\epsilon_y\left(c_{ry}\right)_{i,j}$$

$$C_j = \frac{1}{2}\left[d_y - \frac{1}{2}(1-\epsilon_y)\left(c_{ry}\right)_{i,j+1}\right]$$

$$D_j = \frac{1}{2}\left[d_x - \frac{1}{2}(1-\epsilon_x)(c_{rx})_{i+1,j}\right]\zeta_{i+1,j}^{n+\frac{1}{2}} + \left[1 - d_x - \frac{1}{2}\epsilon_x(c_{rx})_{i,j}\right]\zeta_{i,j}^{n+\frac{1}{2}} + \frac{1}{2}\left[d_x + \frac{1}{2}(1+\epsilon_x)(c_{rx})_{i-1,j}\right]\zeta_{i-1,j}^{n+\frac{1}{2}}$$

Where

$$(c_{rx})_{i,j} = u_{i,j}^{n}\frac{\Delta t}{\Delta x}\ , \qquad \left(c_{ry}\right)_{i,j} = v_{i,j}^{n}\frac{\Delta t}{\Delta y}\ , \qquad d_x = \frac{\nu \Delta t}{(\Delta x)^2}\ , \qquad d_y = \frac{\nu \Delta t}{(\Delta y)^2}$$

Finally, for solution of discretized Poisson equation Point Successive Over-Relaxation method is used

$$\psi_{i,j}^{n+1} = \psi_{i,j}^{n} + \omega R_{i,j}^{n+1}$$

Where

$$R_{i,j}^{n+1} = \frac{1}{4}(u_{i+1,j}^{n} + u_{i-1,j}^{n+1} + u_{i,j+1}^{n} + u_{i,j-1}^{n+1} - 4u_{i,j}^{n} + \Delta^2\zeta_{i,j}^{n})$$

And

$$1 < \omega < 2$$

Domain and Boundary Conditions

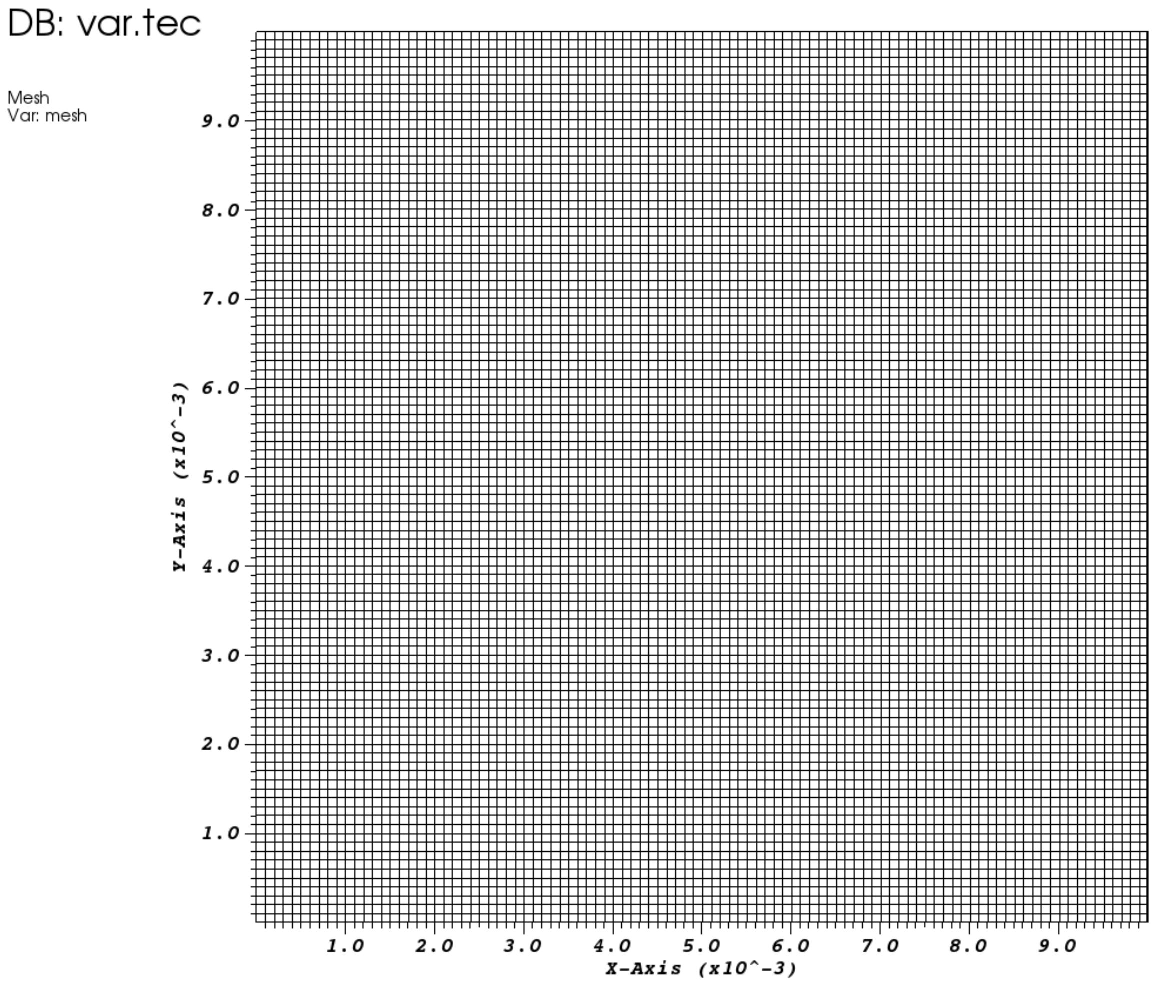


Figure 1: Computational Domain

Computational domain has dimensions of 0.01 meter to 0.01 meter and it is divided to 100x100 cells with constant spacing. Upper edge is moving with velocity U_p , all the other edges are solid boundaries.

Imposing no slip condition on velocities and considering a sloid surface as a streamline whose values would be constants

Bottom wall: $u(i,1) = 0$; $v(i,1) = 0$; $\psi(i,1) = c_1 = 0$

Left wall: $u(1,j) = 0$; $v(1,j) = 0$; $\psi(1,j) = c_2 = 0$

Right wall: $u(N,j) = 0$; $v(N,j) = 0$; $\psi(N,j) = c_3 = 0$

Upper wall: $u(i,N) = U_p$; $v(i,N) = 0$; $\psi(i,N) = c_4 = 0$

There are no boundary conditions for vorticity, but it should be approximated using velocity and stream function values at every iteration.

For left wall:

$$\psi(1,j) = c_1 = 0$$

Substituting above relation to Poisson equation

$$\left(\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2}\right)_{1,j} = \left(\frac{\partial^2\psi}{\partial x^2}\right)_{1,j} = -\zeta_{1,j}$$

Using Taylor series expansion for stream function

$$\psi_{2,j} = \psi_{1,j} + \frac{\partial\psi}{\partial x}|_{1,j}\Delta x + \frac{\partial^2\psi}{\partial x^2}|_{1,j}\frac{\Delta x^2}{2} + \cdots$$

Where, by definition,

$$\frac{\partial\psi}{\partial x}|_{1,j} = -v_{1,j} = 0$$

Solving for the second derivative

$$\frac{\partial^2\psi}{\partial x^2}|_{1,j} = \frac{2(\psi_{2,j} - \psi_{1,j})}{(\Delta x)^2} + O(\Delta x)$$

Boundary value of vorticity along left wall can now be expressed as

$$\zeta_{1,j} = \frac{2(\psi_{1,j} - \psi_{2,j})}{(\Delta x)^2} + O(\Delta x)$$

Similarly, for right and bottom walls

$$\zeta_{1,j} = \frac{2(\psi_{N,j} - \psi_{N-1,j})}{(\Delta x)^2} + O(\Delta x)$$

$$\zeta_{i,1} = \frac{2(\psi_{i,1} - \psi_{i,2})}{(\Delta y)^2} + O(\Delta x)$$

Vorticity at upper wall is a little bit different since that wall is moving

Taylor series expression for stream function at upper wall

$$\psi_{i,N-1} = \psi_{i,N} + \frac{\partial\psi}{\partial y}|_{i,N}\Delta y + \frac{\partial^2\psi}{\partial y^2}|_{i,N}\frac{\Delta y^2}{2} + \cdots$$

Where

$$\frac{\partial\psi}{\partial y}|_{i,N} = U_p \ , \qquad \left(\frac{\partial^2\psi}{\partial y^2}\right)_{i,N} = -\zeta_{i,N}$$

Vorticity is expressed at upper wall such as

$$\zeta_{i,N} = \frac{2(\psi_{i,N} - \psi_{i,N-1})}{(\Delta y)^2} - \frac{2U_p}{\Delta y}$$

Initial Conditions

Initial condition for velocity can be defined by interpolation from the moving plate towards interior points or equal to U_p everywhere or zeros everywhere. Considering final solution and velocity magnitudes, setting zero velocity at interior points is the best option. The same thing is valid for stream function values, too. So initial values are set as

At interior points

$$u_{i,j} = 0\,, v_{i,j} = 0$$

$$\psi_{i,j} = 0$$

Initial values are calculated at interior points for vorticity using velocity values at boundaries and interior points such as

$$\zeta = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

Stability

Theoretically there is no stability restrictions on time step for an implicit scheme. Still the courant number may be a criterion for maximum time step size. The effect of time step is observed and discussed in the Results section.

For Poisson equation, because it is an elliptic problem, just the final solution is concerned and there is no real time derivative in the discretized equation. In this case, over-relaxation parameter will be adjusted for best solution and fastest convergence.

Error Definitions

Since there is no an analytical solution or reference value, error must be defined as relative error. Also, the problem is steady, which means change of ζ in time should go to zero. To represent change of ζ in time relatively we need a reference value. This value may be chosen as old ζ values (from previous iteration). The problem with this approach is that we defined initial ζ values as zero on interior points, which gives division by zero. Other option for reference value is taking the biggest absolute ζ value from previous iteration. This option is better, however, finding biggest value of 100x100 matrix is computationally intense especially at every iteration. It is known that biggest absolute ζ values are located at upper moving wall. So, average of the ζ zeta values at upper wall is taken as a reference value.

Reference value

$$reference = \frac{\sum_{i=1}^{N} \zeta_{i,N}}{N}$$

$$Error_{ADI} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} \frac{\left|\zeta_{i,j} - \zeta_{i,j_{old}}\right|}{reference}}{N^2}$$

Error definition for stream function values is easier since it is an elliptic problem, the residual should go to zero as iteration proceeds. We can define error such as

$$Error_{PSOR} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} \psi_{i,j}}{N^2}$$

Solution Algorithm

1. Construct Grid
2. Initialize Solution Variables
3. Start Solution Loop
    a. Impose and calculate boundary conditions at every iteration
    b. Evaluate ADI coefficients
    c. Solve ADI with
        i. Perform x-sweep for every j index and get half time step solution
        ii. Perform y-sweep for every I index using half time step solution and get full time step solution
    d. Solve PSOR with 20 iteration for every solution loop using $\omega = 1.8$
    e. Evaluate new velocity components using ψ values from PSOR solution
    f. Evaluate error at each solution loop
4. End Solution Loop
5. Output the variables x, y, u, v, ζ, ψ
6. Calculate the drag and print it out

Results and Discussion

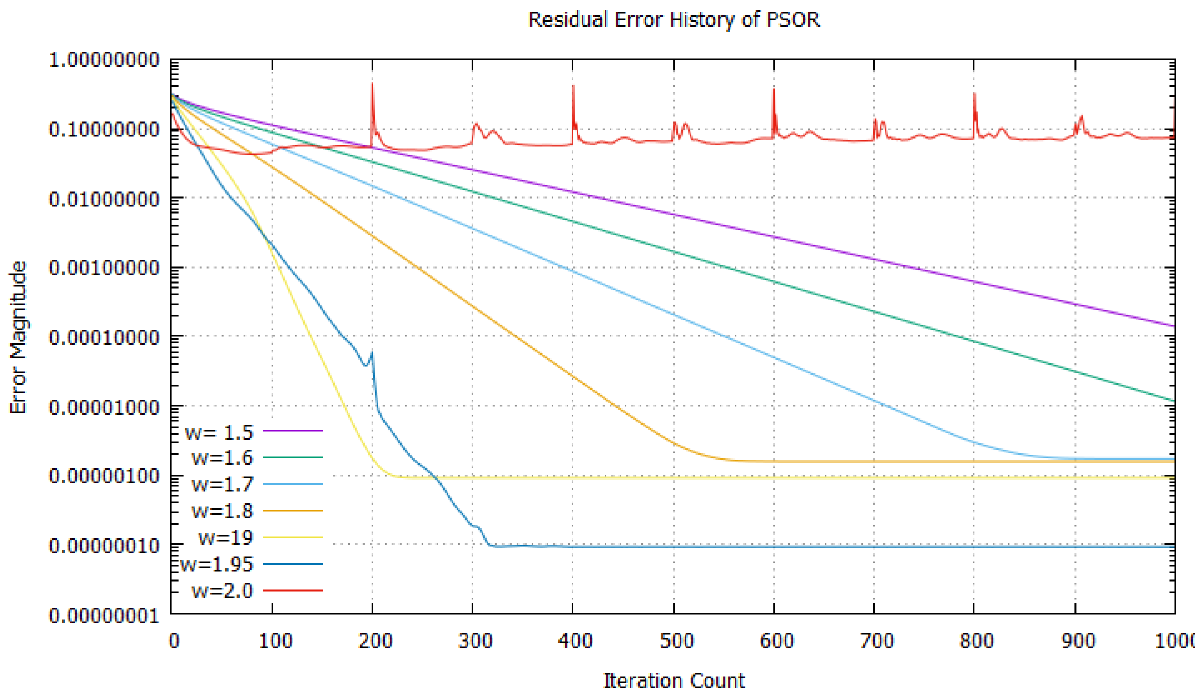


Figure 2: Effect of Over-Relaxation parameter on PSOR convergence

Figure 2 represent the residual(R) behavior of PSOR solution from previous homework. To get best convergence ω would be chosen 1.95. However, that value is for the first iteration and boundary values are changing at every iteration. Thus, it is logical to not use the extreme ω value. It seems ω 1.8 gives enough convergence speed and stability throughout the solution.

In addition, there is no need to get a converged solution for PSOR at each program loop. The intermediate steps do not represent a valid solution so there is no point to put computational effort on to solution. 20 iterations on PSOR for each program loop seem enough to get a solution.

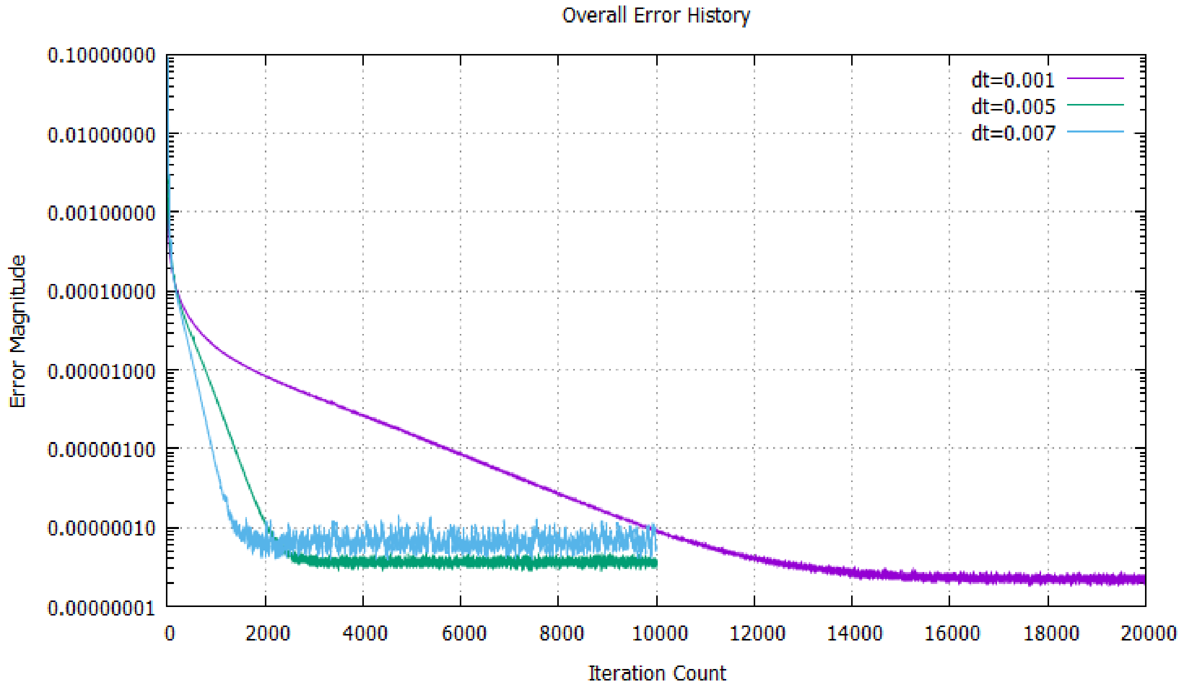


Figure 3: Effect of Time Step on Convergence $R_e = 10^2$

Another parameter to adjust is time step to use in ADI scheme. Although, there is no theoretical limitation on time step, the solution diverges at $R_e = 10^2$ and time step greater than 7×10^{-3}. In addition, taking larger time steps results in more oscillatory behavior. This can also be seen from figure 3. For fast convergence and good accuracy time step is calculated for different U_p as

$$\Delta t = 0.5 \times \frac{\Delta x}{U_p}$$

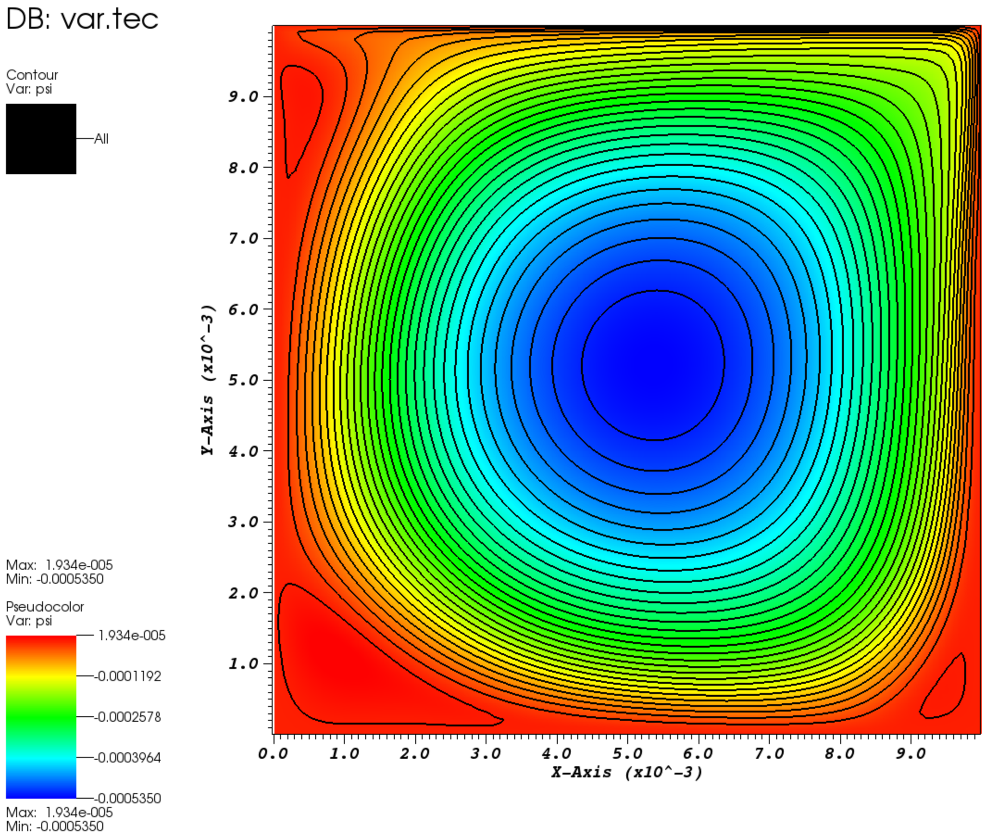


Figure 4: Streamlines $R_e = 10^4$

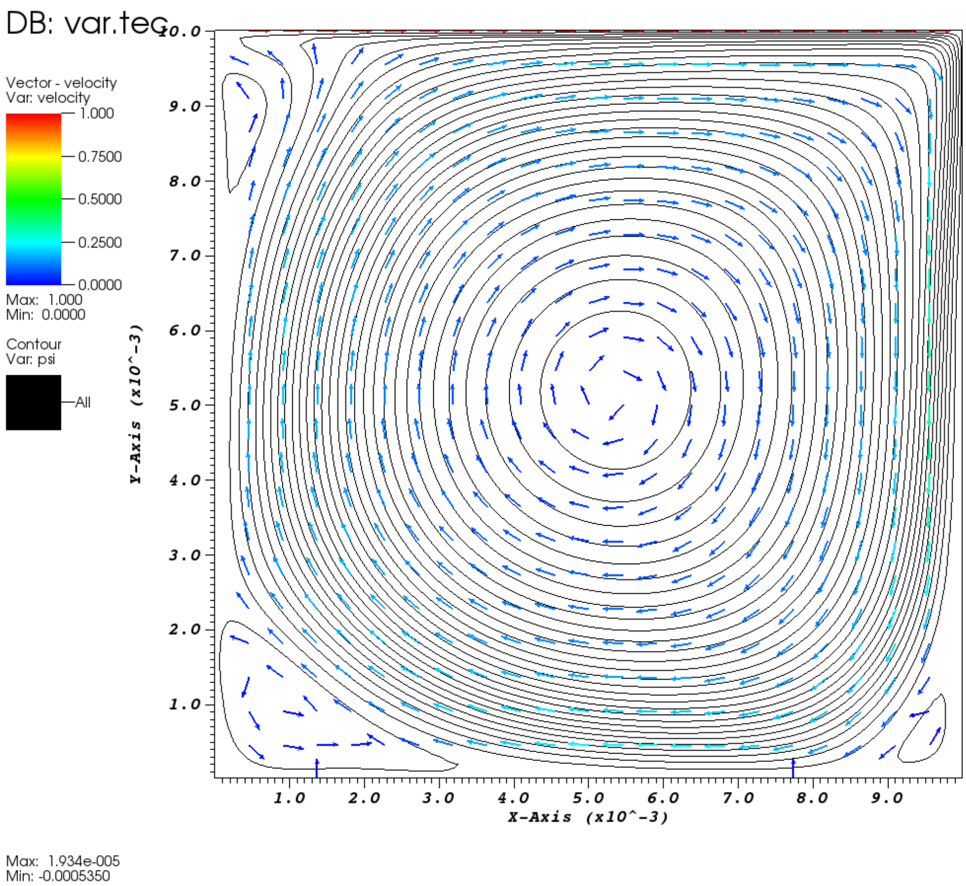


Figure 5: Velocity Vectors $R_e = 10^4$

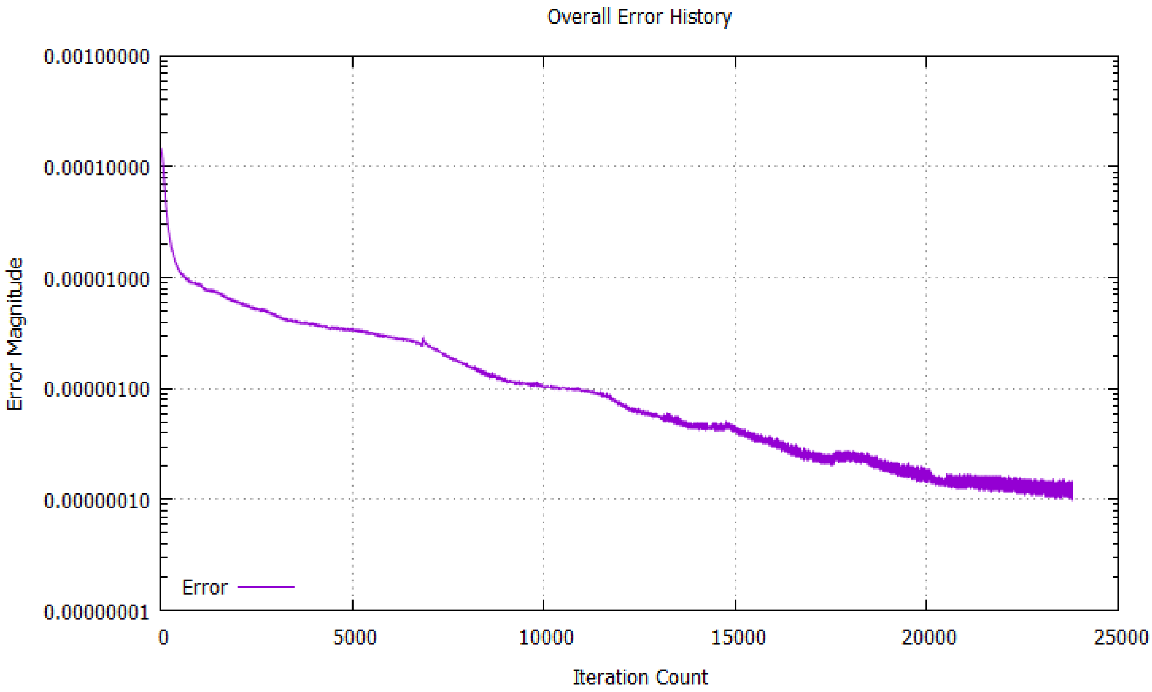


Figure 6: Convergence History $R_e = 10^4$

First case is with $U_p = 1\ m/s$ and resultant Reynold number is $R_e = 10^4$. There are 3 separation regions that can be seen in figure 3 and as velocity vectors in figure 4. It takes around 24000 iterations to get a relative error below 10^{-7}.

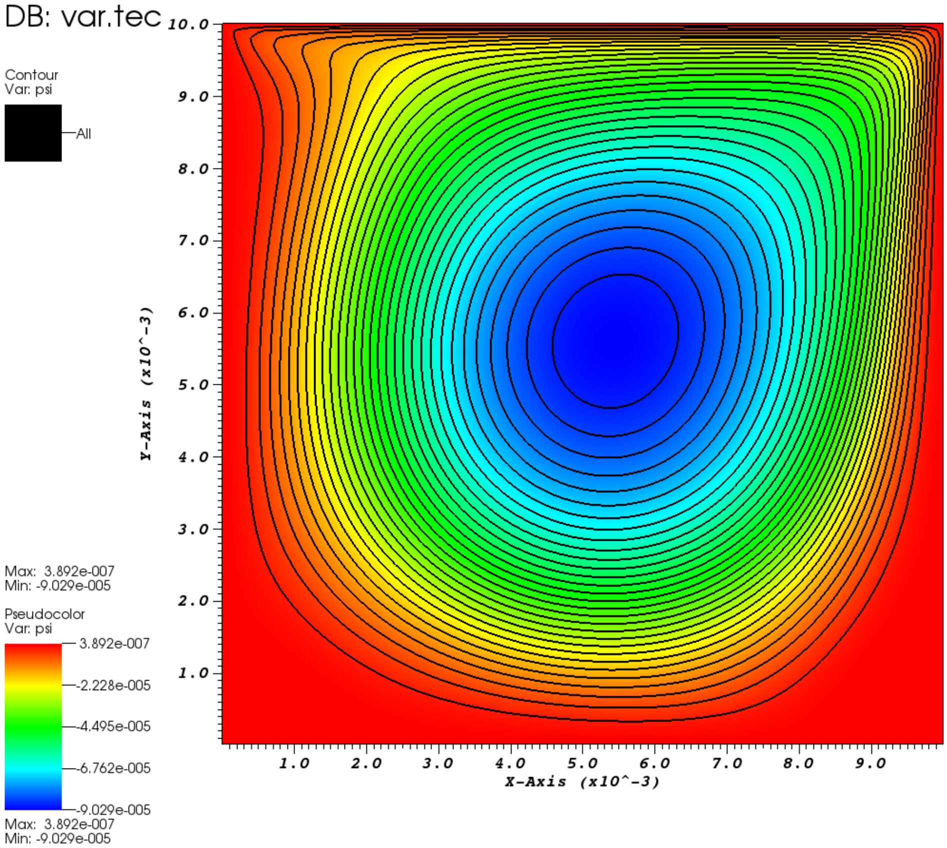


Figure 7: Streamlines $R_e = 10^3$

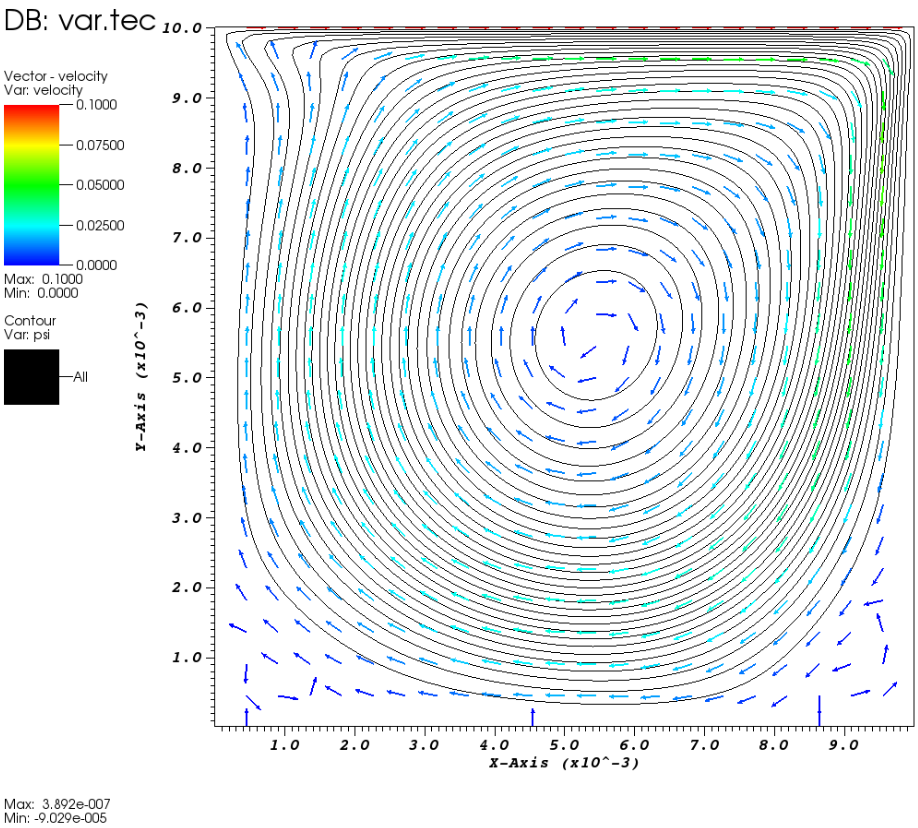


Figure 8: Velocity Vectors $R_e = 10^3$

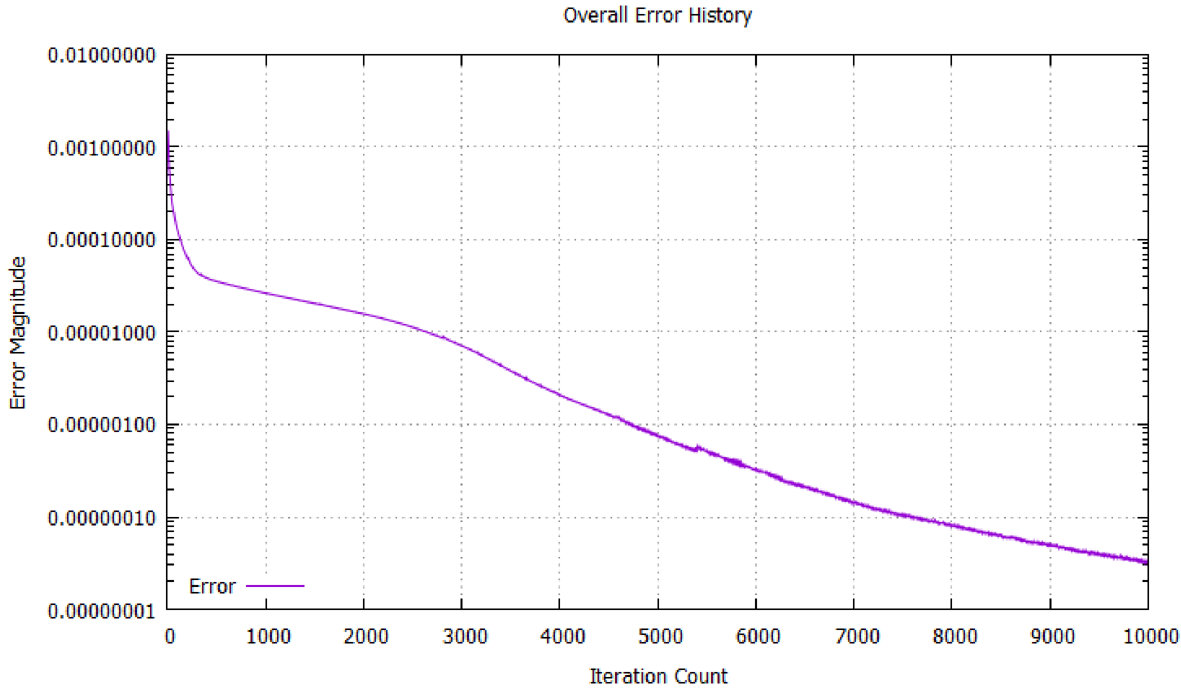


Figure 9: Convergence History $R_e = 10^3$

When $R_e = 10^3$ separation only occurs at bottom corners as indicated in figures 7 and 8. The convergence is much faster in this case, 10 thousand iteration is enough to get a relative error below 10^{-7}.

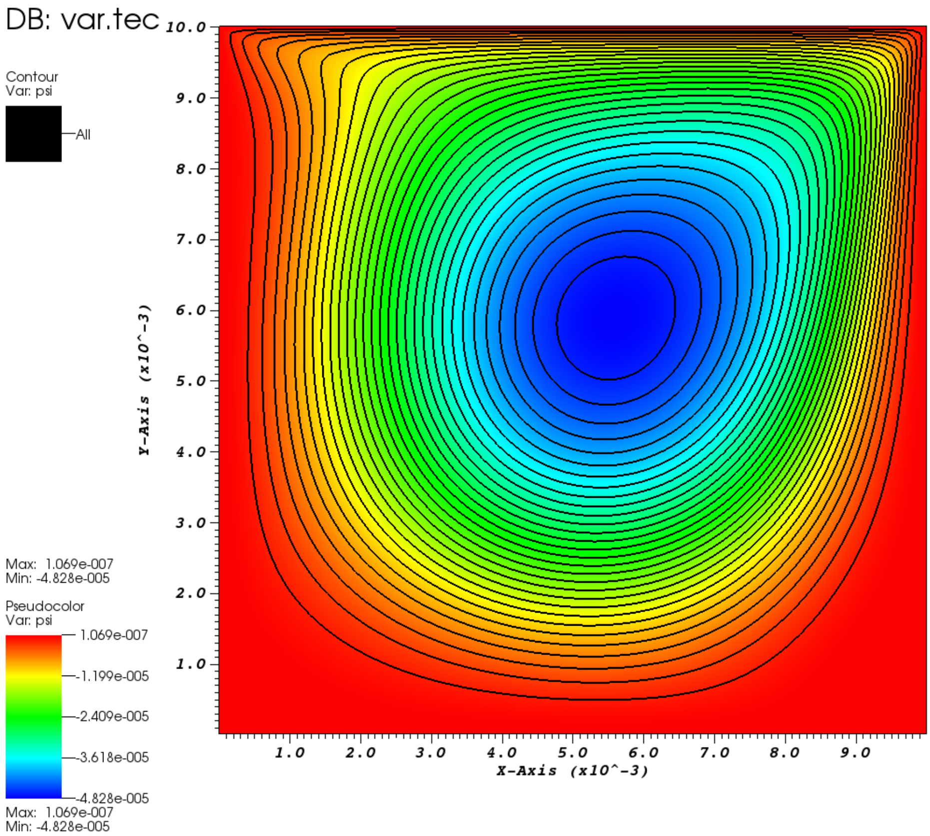


Figure 10: Streamlines $R_e = 5 \times 10^2$

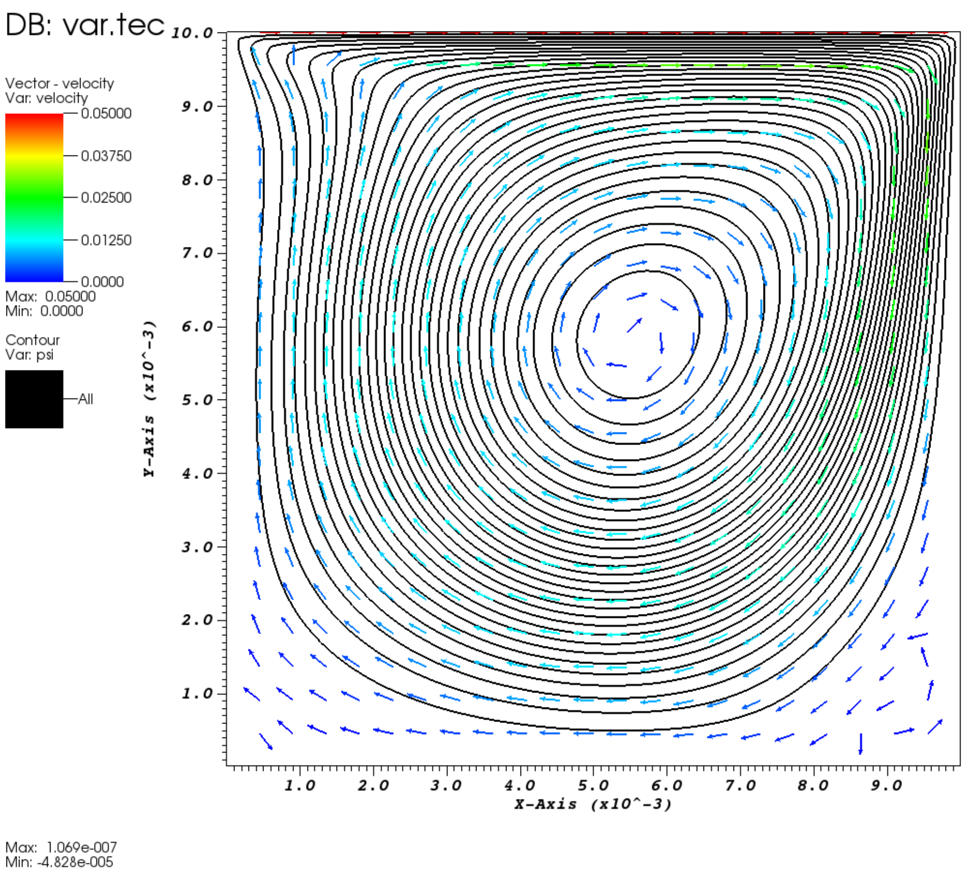


Figure 11: Velocity Vectors $R_e = 5 \times 10^2$

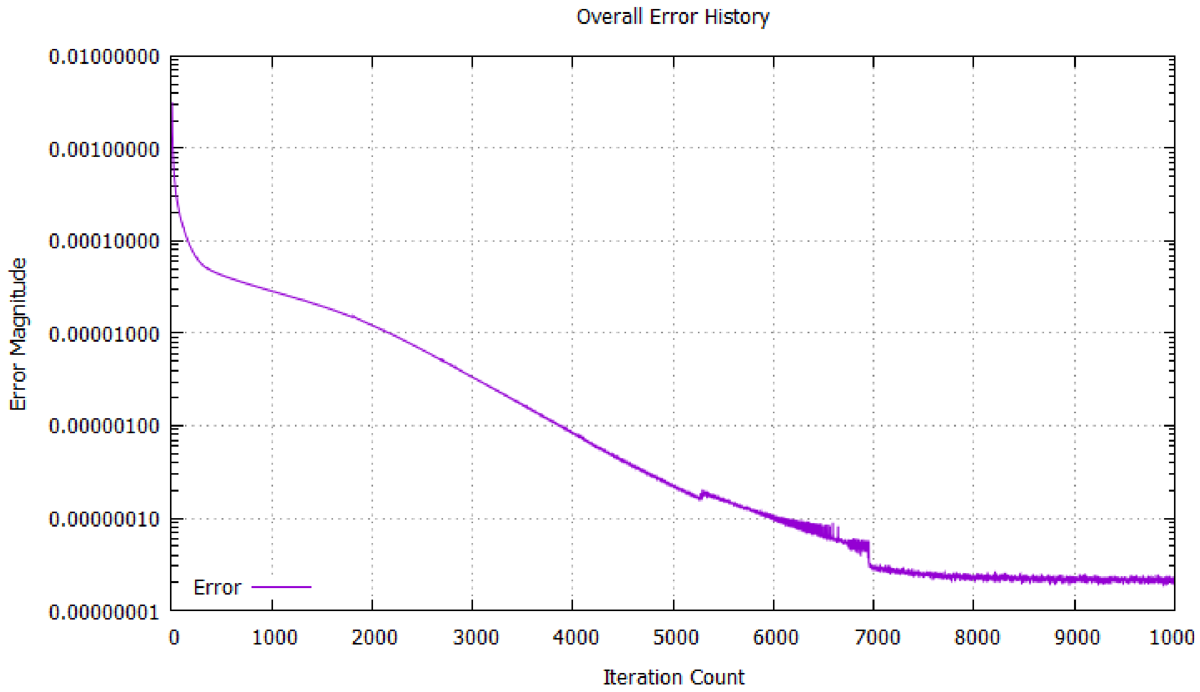


Figure 12: Convergence History $R_e = 5 \times 10^2$

Figure 10 and 11 shows the stream function behavior and velocity vectors, as can be seen the separation at bottom left corner is very small ($R_e = 5 \times 10^2$) compared to previous case. Also, the convergence is getting faster, figure 12 shows that around 7 thousand of iteration gives relative error below 10^{-7}.

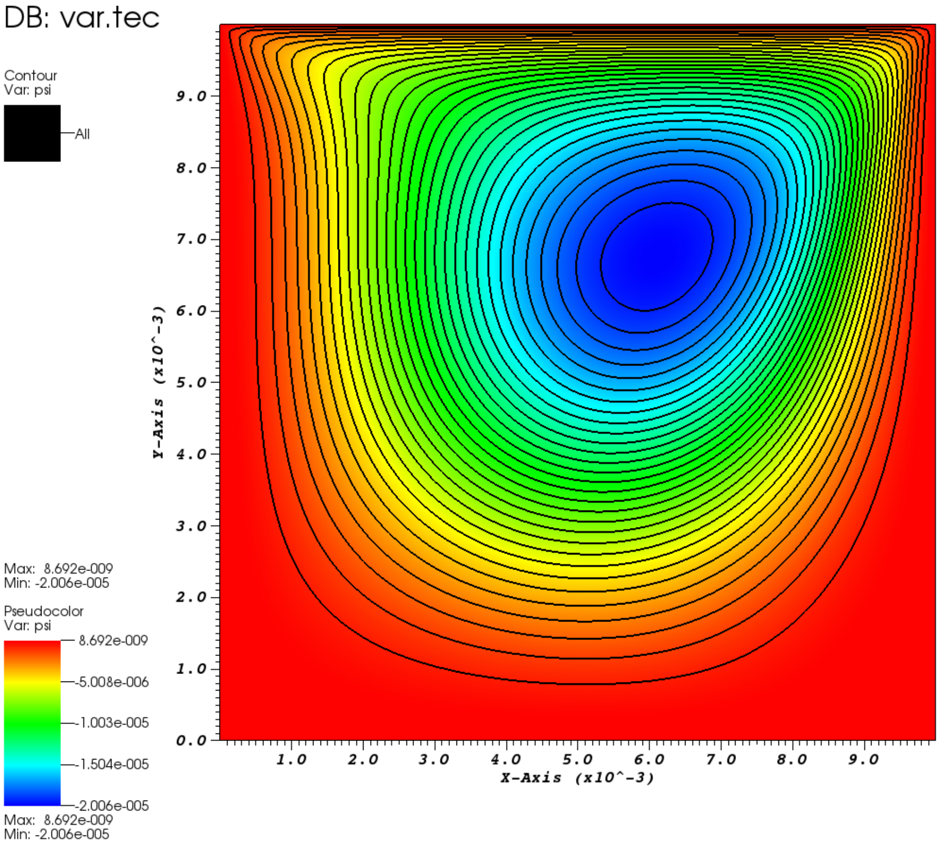


Figure 13: Streamlines $R_e = 2 \times 10^2$

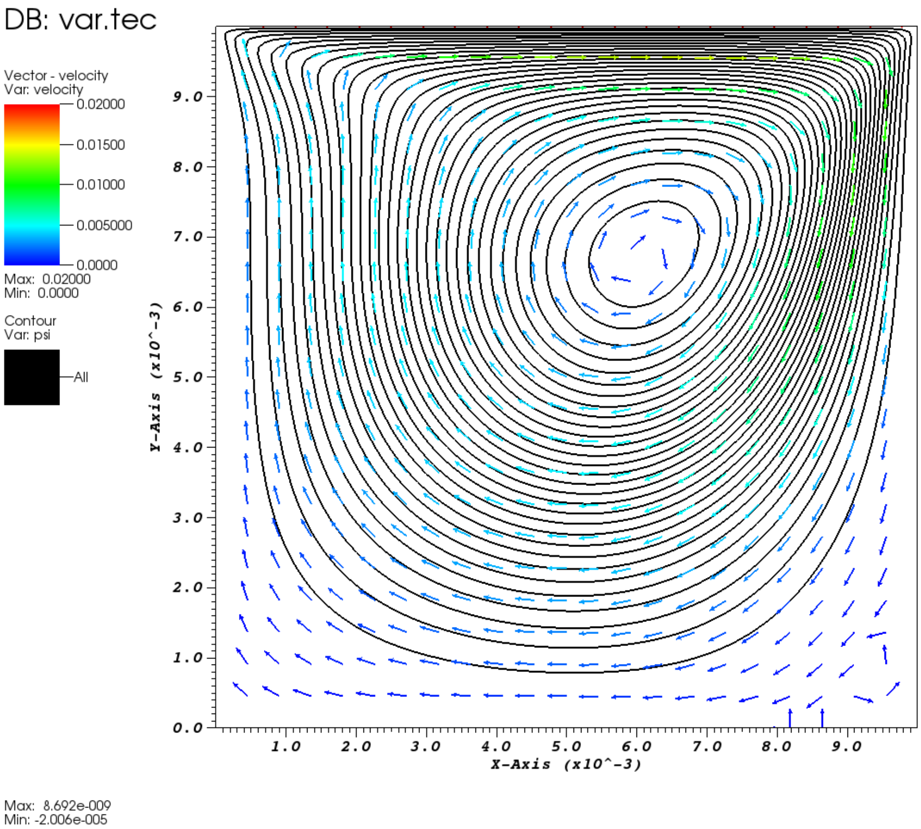


Figure 14: Velocity Vectors $R_e = 2 \times 10^2$

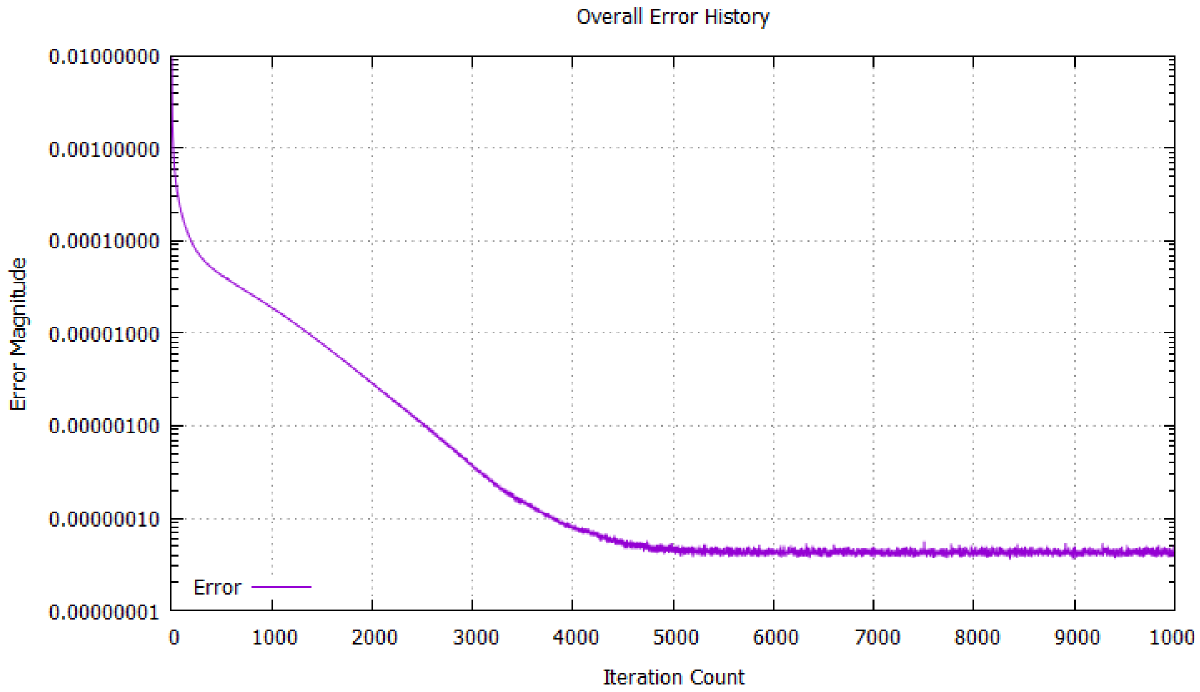


Figure 15: Convergence History $R_e = 2 \times 10^2$

With reduced further reduction of Reynolds number to $R_e = 2 \times 10^2$ eliminates the separation at bottom left corner. Again, the convergence is faster than previous case.

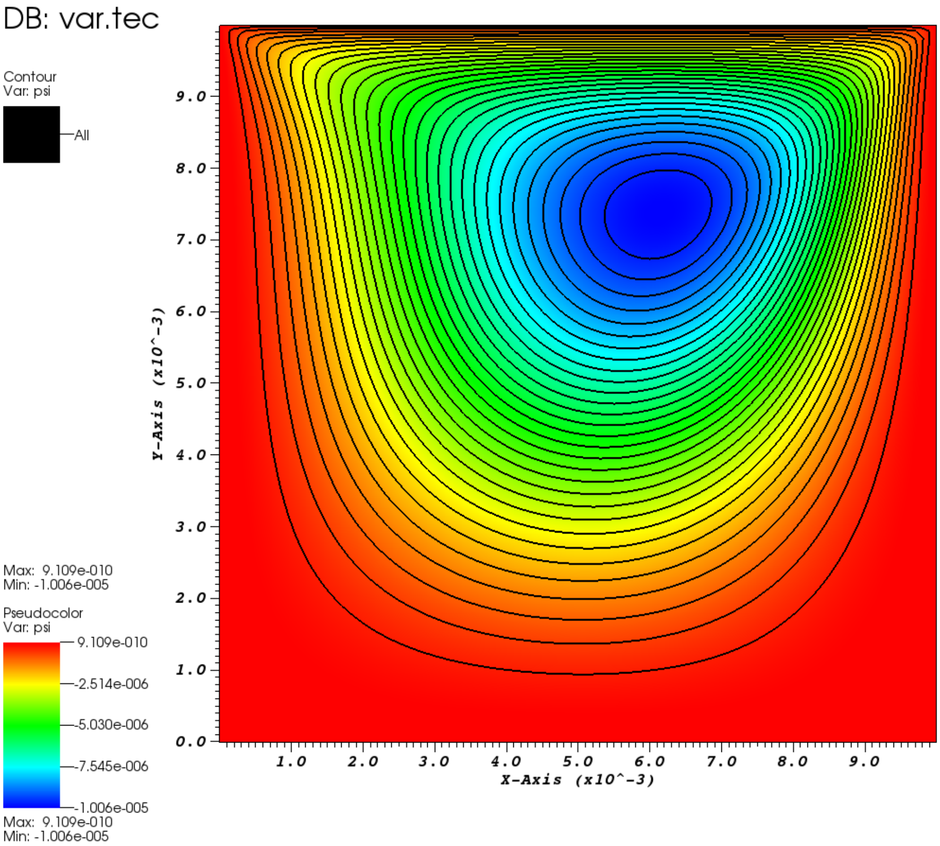


Figure 16: Streamlines $R_e = 10^2$

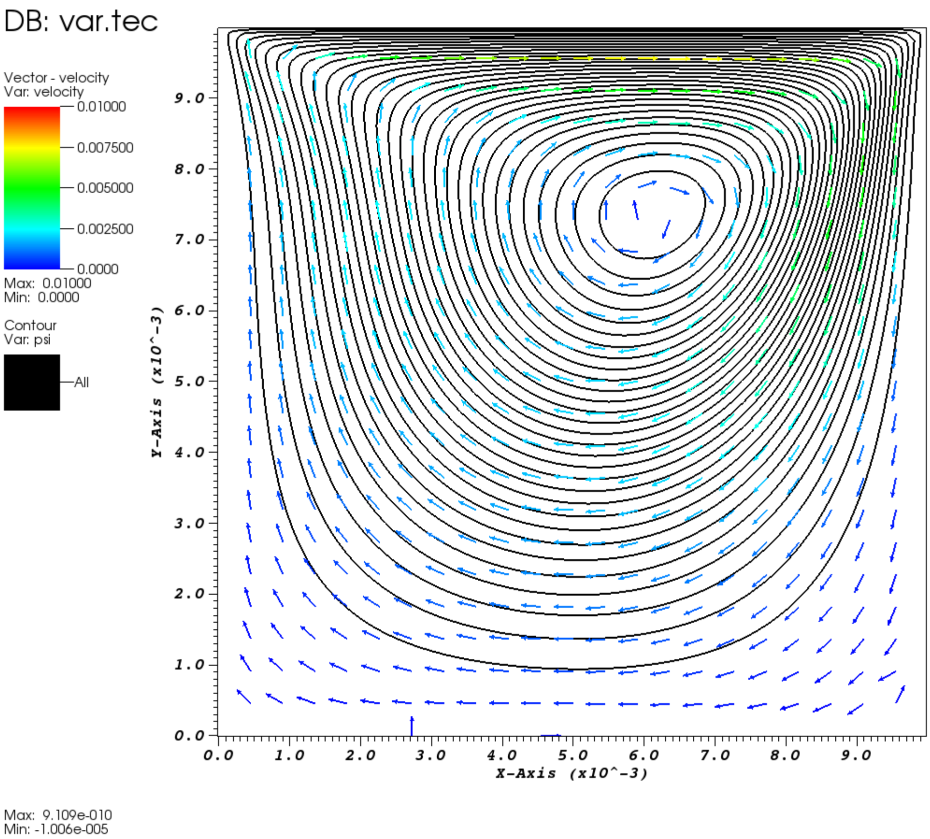


Figure 17: Velocity Vectors $R_e = 10^2$

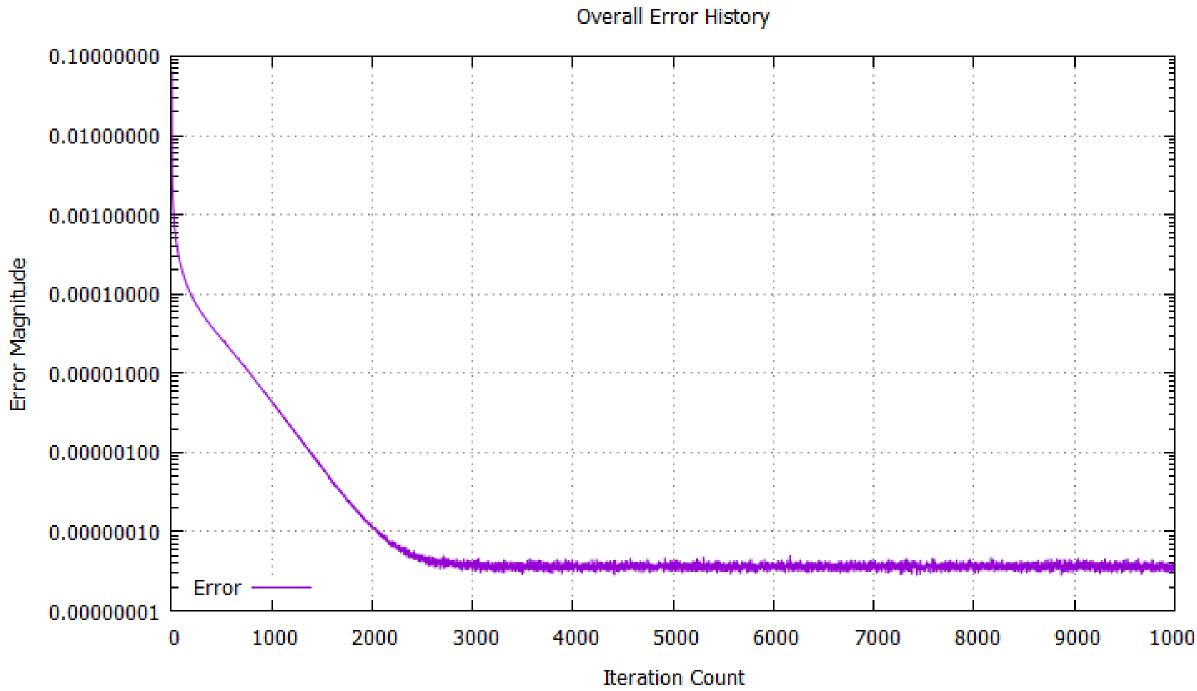


Figure 18: Convergence History $R_e = 10^2$

In the final case where $R_e = 10^2$, there is almost no separation of flow and the convergence is fastest with around 2500 iterations.

Up [m/s]	Re	Max ψ value at center	Max ψ at Separation	Drag Force [N/m]
1	1.0E+04	-5.350E-04	1.934E-05	7.553E-02
0.1	1.0E+03	-9.029E-05	3.892E-07	3.347E-03
0.05	5.0E+02	-4.828E-05	1.069E-07	1.346E-03
0.02	2.0E+02	-2.006E-05	8.692E-09	4.302E-04
0.01	1.0E+02	-1.006E-05	9.109E-10	1.924E-04

Table 1: Results

Overall results are given in table 1. Higher velocity at upper plates gives bigger stream function value at the core. The reason for that is stream function value at the upper wall is set to zero and stream function values calculated with integration of velocity differences. Bigger the velocity difference bigger the stream function values (The negative sigh at center ψ values are a result of notation). In addition, the core ψ value is affected by presence of separation. Stronger separation results in bigger stream function value.

As the definition of Reynolds Number, ratio of inertial forces to viscous forces, indicates, in higher Re flows separation becomes stronger.

At separation regions velocity vector directions are changes more than no separation cases, this might be the reason of convergence speed difference. At $R_e = 10^4$ convergence is achieved with 24000 iterations and at $R_e = 10^2$ solution converged at 3000 iterations. Although we used upwinding method, changing solution direction through solution makes convergence slower but it prevents divergence.

Drag Force is calculated as unit width using trapezoidal integration at upper moving plate. Using shear stress relation

$$\tau = \mu \frac{du}{dy}$$

Where μ is taken as 10^{-3} and $\rho = 1000\ kg/m^3$

It is expected to have higher drag with higher plate velocity. As the Reynolds number gets bigger the relation between U_p and Drag Force becomes highly nonlinear.

Program Listing

```
            program CavityFlow
c..Taha Yaşar Demir / 1881978
c..CE-580 HomeWork #7
            parameter (mx=101)
            common/grd/ x(mx,mx),y(mx,mx),dydx,N,rL,epsx(mx,mx),epsy(mx,mx)
            common/flw/ u(mx,mx),v(mx,mx),psi(mx,mx),zeta(mx,mx)
            common/par/ u_p, visc, dt, time, crx(mx,mx), cry(mx,mx),dx,dy
            common/err/ Erv, Ers, R(mx,mx)
            common/tho/ a(mx),b(mx),c(mx),d(mx)
            real tolerance
            integer iter
            open(22,file="erroradi.dat")
            call grid
            call init
            Ers = 1.
            Erv = 1.
            iter= 0.
            tolerance = 1e-7
c           do while(Erv.gt.tolerance)
            do m=1,20000
                  iter = iter + 1
                  call boundary
                  call evalcoef
                  call ADI
                  call psor
                  call velocity
                  call error(iter)
                  print*, iter,Erv,Ers
            enddo
            call output
            call dragcal
            close(22)
            close(33)
            stop
            end

c-----------------------------------------------------------------------
            subroutine grid
            parameter (mx=101)
            common/grd/ x(mx,mx),y(mx,mx),dydx,N,rL,epsx(mx,mx),epsy(mx,mx)
            common/flw/ u(mx,mx),v(mx,mx),psi(mx,mx),zeta(mx,mx)
            common/par/ u_p, visc, dt, time, crx(mx,mx), cry(mx,mx),dx,dy
            common/err/ Erv, Ers, R(mx,mx)
            common/tho/ a(mx),b(mx),c(mx),d(mx)

            rL   = 0.01 !m
            N    = 101
            dydx = rL/(N-1)
```

```
        do j=1,N
              x(1,j) = 0.
              do i=2,N-1
                     x(i,j) = x(i-1,j) + dydx
              enddo
              x(N,j) = rL
        enddo


        do i=1,N
              y(i,1) = 0.
              do j=2,N-1
                     y(i,j) = y(i,j-1) + dydx
              enddo
              y(i,N) = rL
        enddo


        return
        end
c---------------------------------------------------------------------
        subroutine init
        parameter (mx=101)
        common/grd/ x(mx,mx),y(mx,mx),dydx,N,rL,epsx(mx,mx),epsy(mx,mx)
        common/flw/ u(mx,mx),v(mx,mx),psi(mx,mx),zeta(mx,mx)
        common/par/ u_p, visc, dt, time, crx(mx,mx), cry(mx,mx),dx,dy
        common/err/ Erv, Ers, R(mx,mx)
        common/tho/ a(mx),b(mx),c(mx),d(mx)

        time = 0.
        u_p  = 0.01 ! 0.01-0.02-0.05-0.1-1
        visc = 1E-6
        dt   = 0.1*(dydx/u_p)
        call boundary
        do i=2,N-1
              do j=2,N-1
                     u(i,j)   = 0.
                     v(i,j)   = 0.
                     psi(i,j) = 0.
                     zeta(i,j)= (v(i,j)-v(i-1,j))/dydx - (u(i,j)-u(i-1,j))/dydx
              enddo
        enddo

        return
        end
c---------------------------------------------------------------------
        subroutine boundary
        parameter (mx=101)
        common/grd/ x(mx,mx),y(mx,mx),dydx,N,rL,epsx(mx,mx),epsy(mx,mx)
        common/flw/ u(mx,mx),v(mx,mx),psi(mx,mx),zeta(mx,mx)
        common/par/ u_p, visc, dt, time, crx(mx,mx), cry(mx,mx),dx,dy
```

```
      common/err/ Erv, Ers, R(mx,mx)
      common/tho/ a(mx),b(mx),c(mx),d(mx)

      do i=2,N-1
            psi(i,1) = 0.
            psi(i,N) = 0.
            u(i,1)   = 0.
            u(i,N)   = u_p
            v(i,1)   = 0.
            v(i,N)   = 0.
            zeta(i,1)= 2*(psi(i,1)-psi(i,2))/(dydx**2)
            zeta(i,N)= (2*(psi(i,N)-psi(i,N-1))/(dydx**2)) - 2*u_p/dydx
      enddo

      do j=1,N
            psi(1,j) = 0.
            psi(N,j) = 0.
            u(1,j)   = 0.
            u(N,j)   = 0.
            v(1,j)   = 0.
            v(N,j)   = 0.
            zeta(1,j)= 2*(psi(1,j)-psi(2,j))/(dydx**2)
            zeta(N,j)= 2*(psi(N,j)-psi(N-1,j))/(dydx**2)
      enddo

      return
      end
c-----------------------------------------------------------------------
      subroutine evalcoef
      parameter (mx=101)
      common/grd/ x(mx,mx),y(mx,mx),dydx,N,rL,epsx(mx,mx),epsy(mx,mx)
      common/flw/ u(mx,mx),v(mx,mx),psi(mx,mx),zeta(mx,mx)
      common/par/ u_p, visc, dt, time, crx(mx,mx), cry(mx,mx),dx,dy
      common/err/ Erv, Ers, R(mx,mx)
      common/tho/ a(mx),b(mx),c(mx),d(mx)

      do i=1,N
            do j=1,N
                  if (u(i,j).gt.0.) then
                        epsx(i,j) = 1.
                  else
                        epsx(i,j) =-1.
                  endif
                  if (v(i,j).gt.0.) then
                        epsy(i,j) = 1.
                  else
                        epsy(i,j) =-1.
                  endif
                  crx(i,j) = u(i,j)*dt/dydx
                  cry(i,j) = v(i,j)*dt/dydx
                  dx       = visc*dt/(dydx**2)
                  dy       = visc*dt/(dydx**2)
```

```
                enddo
        enddo
        return
        end
c-----------------------------------------------------------------------
        subroutine ADI
        parameter (mx=101)
        common/grd/ x(mx,mx),y(mx,mx),dydx,N,rL,epsx(mx,mx),epsy(mx,mx)
        common/flw/ u(mx,mx),v(mx,mx),psi(mx,mx),zeta(mx,mx)
        common/par/ u_p, visc, dt, time, crx(mx,mx), cry(mx,mx),dx,dy
        common/err/ Erv, Ers, R(mx,mx)
        common/tho/ a(mx),b(mx),c(mx),d(mx)

c...x_sweep
        do j=2,N-1
                call x_sweep(j)
        enddo

c...y_sweep
        do i=2,N-1
                call y_sweep(i)
        enddo

        return
        end
c-----------------------------------------------------------------------
        subroutine x_sweep(j)
        parameter (mx=101)
        common/grd/ x(mx,mx),y(mx,mx),dydx,N,rL,epsx(mx,mx),epsy(mx,mx)
        common/flw/ u(mx,mx),v(mx,mx),psi(mx,mx),zeta(mx,mx)
        common/par/ u_p, visc, dt, time, crx(mx,mx), cry(mx,mx),dx,dy
        common/err/ Erv, Ers, R(mx,mx)
        common/tho/ a(mx),b(mx),c(mx),d(mx)


        do i=2,N-1
                if(i.eq.2) then
                        a(i) =  0.
                        b(i) =  1 + dx + 0.5*epsx(i,j)*crx(i,j)
                        c(i) = -0.5*(dx - 0.5*(1-epsx(i,j))*crx(i+1,j))
                        d(i) =  0.5*(dx + 0.5*(1+epsx(i,j))*crx(i-1,j))*zeta(i-1,j)
    &         + 0.5*(dy - 0.5*(1-epsy(i,j))*cry(i,j+1))*zeta(i,j+1)
    &         + (1-dy-0.5*epsy(i,j)*cry(i,j))*zeta(i,j)
    &         + 0.5*(dy + 0.5*(1+epsy(i,j))*cry(i,j-1))*zeta(i,j-1)
                elseif(i.eq.N-1) then
                        a(i) = -0.5*(dx + 0.5*(1+epsx(i,j))*crx(i-1,j))
                        b(i) =  1 + dx + 0.5*epsx(i,j)*crx(i,j)
                        c(i) =  0.
                        d(i) =  0.5*(dx - 0.5*(1-epsx(i,j))*crx(i+1,j))*zeta(i+1,j)
    &         + 0.5*(dy - 0.5*(1-epsy(i,j))*cry(i,j+1))*zeta(i,j+1)
    &         + (1-dy-0.5*epsy(i,j)*cry(i,j))*zeta(i,j)
    &         + 0.5*(dy + 0.5*(1+epsy(i,j))*cry(i,j-1))*zeta(i,j-1)
```

```
            else
                  a(i) = -0.5*(dx + 0.5*(1+epsx(i,j))*crx(i-1,j))
                  b(i) =  1 + dx + 0.5*epsx(i,j)*crx(i,j)
                  c(i) = -0.5*(dx - 0.5*(1-epsx(i,j))*crx(i+1,j))
                  d(i) =  0.5*(dy - 0.5*(1-epsy(i,j))*cry(i,j+1))*zeta(i,j+1)
     &      + (1-dy-0.5*epsy(i,j)*cry(i,j))*zeta(i,j)
     &      + 0.5*(dy + 0.5*(1+epsy(i,j))*cry(i,j-1))*zeta(i,j-1)

            endif
      enddo

      call THOMAS(2,N-1,a,b,c,d)

      do i=2,N-1 ! extract the solution from thomas algorithm
            zeta(i,j) = d(i)
      enddo

      return
      end
c---------------------------------------------------------------------
      subroutine y_sweep(i)
      parameter (mx=101)
      common/grd/ x(mx,mx),y(mx,mx),dydx,N,rL,epsx(mx,mx),epsy(mx,mx)
      common/flw/ u(mx,mx),v(mx,mx),psi(mx,mx),zeta(mx,mx)
      common/par/ u_p, visc, dt, time, crx(mx,mx), cry(mx,mx),dx,dy
      common/err/ Erv, Ers, R(mx,mx)
      common/tho/ a(mx),b(mx),c(mx),d(mx)

      do j=2,N-1
            if(j.eq.2) then
                  a(j) =  0.
                  b(j) =  1 + dy + 0.5*(epsy(i,j))*cry(i,j)
                  c(j) = -0.5*(dy - 0.5*(1-epsy(i,j))*cry(i,j+1))
                  d(j) =  0.5*(dy + 0.5*(1+epsy(i,j))*cry(i,j-1))*zeta(i,j-1)
     &      +0.5*(dx - 0.5*(1-epsx(i,j))*crx(i+1,j))*zeta(i+1,j)
     &      + (1-dx-0.5*epsx(i,j)*crx(i,j))*zeta(i,j)
     &      +0.5*(dx + 0.5*(1+epsx(i,j))*crx(i-1,j))*zeta(i-1,j)
            elseif(j.eq.N-1) then
                  a(j) = -0.5*(dy + 0.5*(1+epsy(i,j))*cry(i,j-1))
                  b(j) =  1 + dy + 0.5*(epsy(i,j))*cry(i,j)
                  c(j) =  0.
                  d(j) =  0.5*(dy - 0.5*(1-epsy(i,j))*cry(i,j+1))*zeta(i,j+1)
     &      +0.5*(dx - 0.5*(1-epsx(i,j))*crx(i+1,j))*zeta(i+1,j)
     &      + (1-dx-0.5*epsx(i,j)*crx(i,j))*zeta(i,j)
     &      +0.5*(dx + 0.5*(1+epsx(i,j))*crx(i-1,j))*zeta(i-1,j)
            else
                  a(j) = -0.5*(dy + 0.5*(1+epsy(i,j))*cry(i,j-1))
                  b(j) =  1 + dy + 0.5*(epsy(i,j))*cry(i,j)
                  c(j) = -0.5*(dy - 0.5*(1-epsy(i,j))*cry(i,j+1))
                  d(j) =  0.5*(dx - 0.5*(1-epsx(i,j))*crx(i+1,j))*zeta(i+1,j)
     &      + (1-dx-0.5*epsx(i,j)*crx(i,j))*zeta(i,j)
     &      +0.5*(dx + 0.5*(1+epsx(i,j))*crx(i-1,j))*zeta(i-1,j)
```

```
            endif
        enddo

        call THOMAS(2,N-1,a,b,c,d)

        do j=2,N-1
            zeta(i,j) = d(j)
        enddo

        return
        end
c---------------------------------------------------------------------
        subroutine psor
        parameter (mx=101)
        common/grd/ x(mx,mx),y(mx,mx),dydx,N,rL,epsx(mx,mx),epsy(mx,mx)
        common/flw/ u(mx,mx),v(mx,mx),psi(mx,mx),zeta(mx,mx)
        common/par/ u_p, visc, dt, time, crx(mx,mx), cry(mx,mx),dx,dy
        common/err/ Erv, Ers, R(mx,mx)
        common/tho/ a(mx),b(mx),c(mx),d(mx)
        real omega,sum
        open(33,file="errorpsor.dat")
        omega = 1.8 ! over-relaxation parameter
        sum   = 0.
        do k=1,20
        do j=2,N-1
            do i=2,N-1
                R(i,j) = 0.25*(psi(i+1,j)+psi(i-1,j)+psi(i,j+1)+psi(i,j-1)
     &          - 4*psi(i,j) + (dydx**2)*zeta(i,j))
                psi(i,j) = psi(i,j) + omega*R(i,j)
                sum = sum + abs(R(i,j))
            enddo
        enddo
        sum = sum/((N-2)**2)
        write(33,*) k,sum
        enddo


        return
        end
c---------------------------------------------------------------------
        subroutine velocity
        parameter (mx=101)
        common/grd/ x(mx,mx),y(mx,mx),dydx,N,rL,epsx(mx,mx),epsy(mx,mx)
        common/flw/ u(mx,mx),v(mx,mx),psi(mx,mx),zeta(mx,mx)
        common/par/ u_p, visc, dt, time, crx(mx,mx), cry(mx,mx),dx,dy
        common/err/ Erv, Ers, R(mx,mx)
        common/tho/ a(mx),b(mx),c(mx),d(mx)

        do i=2,N-1
            do j=2,N-1
                u(i,j) = (psi(i,j+1) - psi(i,j))/dydx
                v(i,j) =-(psi(i+1,j) - psi(i,j))/dydx
```

```
                enddo
        enddo

        return
        end
c---------------------------------------------------------------------
        subroutine error(iteration)
        parameter (mx=101)
        common/grd/ x(mx,mx),y(mx,mx),dydx,N,rL,epsx(mx,mx),epsy(mx,mx)
        common/flw/ u(mx,mx),v(mx,mx),psi(mx,mx),zeta(mx,mx)
        common/par/ u_p, visc, dt, time, crx(mx,mx), cry(mx,mx),dx,dy
        common/err/ Erv, Ers, R(mx,mx)
        common/tho/ a(mx),b(mx),c(mx),d(mx)
        real adier , psorer, zetaold(mx,mx), psiold(mx,mx), sum
        integer iteration

        adier = 0.
        psorer= 0.
        sum   = 0.
        do i=1,N
                sum = sum + abs(zeta(i,N))
        enddo
        sum = sum/N
c.. L2 normalization is used for vorticity and stream function values
        if(iteration.eq.1) then ! store the previous zeta and psi values
                do j=1,N
                        do i=1,N
                                zetaold(i,j) = zeta(i,j)
                                psiold(i,j)  = psi(i,j)
                        enddo
                enddo
                print*,
        else
                do j=1,N
                        do i=1,N
                        adier=adier+abs((zeta(i,j)-zetaold(i,j))/sum)
                        zetaold(i,j) = zeta(i,j) ! update the old values for next iteration
                        psorer      = psorer+abs(psi(i,j)-psiold(i,j))
                        psiold(i,j)  = psi(i,j)
                        enddo
                enddo

                Erv = adier/(N**2) ! Vorticity transport equation error
                Ers = psorer/(N**2)! Stream function solution errror

                write(22,*) iteration,Erv,Ers

        endif

        return
        end
c---------------------------------------------------------------------
```

```
      subroutine output
      parameter (mx=101)
      common/grd/ x(mx,mx),y(mx,mx),dydx,N,rL,epsx(mx,mx),epsy(mx,mx)
      common/flw/ u(mx,mx),v(mx,mx),psi(mx,mx),zeta(mx,mx)
      common/par/ u_p, visc, dt, time, crx(mx,mx), cry(mx,mx),dx,dy
      common/err/ Erv, Ers, R(mx,mx)
      common/tho/ a(mx),b(mx),c(mx),d(mx)

      open(11,file='var.tec',form='formatted')
      write(11,*) ' variables="x","y","zeta","psi","u","v" '
      write(11,*) ' zone i=',N, 'j=',N
      do j=1,N
          do i=1,N
              write(11,'(8E12.4)') x(i,j),y(i,j),zeta(i,j),psi(i,j),
     +            u(i,j),v(i,j)
          enddo
      enddo

      return
      end
c---------------------------------------------------------------------
      subroutine dragcal
      parameter (mx=101)
      common/grd/ x(mx,mx),y(mx,mx),dydx,N,rL,epsx(mx,mx),epsy(mx,mx)
      common/flw/ u(mx,mx),v(mx,mx),psi(mx,mx),zeta(mx,mx)
      common/par/ u_p, visc, dt, time, crx(mx,mx), cry(mx,mx),dx,dy
      common/err/ Erv, Ers, R(mx,mx)
      common/tho/ a(mx),b(mx),c(mx),d(mx)
      real mu,rho,shear,drag

      rho   = 1000.
      mu    = visc * rho
      drag  = 0.
      do i=2,N
          shear = mu*(u(i,N)-u(i,N-1))/dydx
          drag  = drag + shear*(x(i,N)-x(i-1,N))
      enddo
      print*, "2-D drag force = ", drag
      return
      end
c---------------------------------------------------------------------
   subroutine THOMAS(il,iu,aa,bb,cc,ff)
c.........................................................
c  Solution of a tridiagonal system of n equations of the form
c  A(i)*x(i-1) + B(i)*x(i) + C(i)*x(i+1) = R(i)  for i=il,iu
c  the solution X(i) is stored in F(i)
c  A(il-1) and C(iu+1) are not used
c  A,Bb,C,R are arrays to bbe provided bby the user
c.........................................................
   parameter (mx=101)
   dimension aa(mx),bb(mx),cc(mx),ff(mx),tmp(mx)
```

```
      tmp(il)=cc(il)/bb(il)
      ff(il)=ff(il)/bb(il)
      ilp1 = il+1
      do i=ilp1,iu
        z=1./(bb(i)-aa(i)*tmp(i-1))
        tmp(i)=cc(i)*Z
        ff(i)=(ff(i)-aa(i)*ff(i-1))*z
      enddo
      iupil=iu+il
      do ii=ilp1,iu
        i=iupil-ii
        ff(i)=ff(i)-tmp(i)*ff(i+1)
      enddo
      return
      end
```

Fortran Code