INCOMPRESSIBLE FLOW IN LID DRIVEN CAVITY FLOW USING SEMI IMPICIT METHOD FOR PRESSURE CORRECTION.

MAE 540 COMPUTATIONAL FLUID MECHANICS

Submitted by: Sumeet Lulekar - 50203322

Date: 5/2/2017

The Semi Implicit Method for Pressure Correction (SIMPLE) algorithm used to couple the Navier-Stokes equation (N-S) along with the Pressure Poisson equation in generalized curvilinear co-ordinates to solve laminar flow in square (L x L) driven cavity. Solutions are obtained for configurations with Reynolds number as high as 1000. Because of the appearance of two or more than one secondary vortices in the flow field near the boundaries of the cavity, centrally stretched grids are used.

INTRODUCTION

As we know, the major difficulty encountered during solution of incompressible flow is the non-availability of the equation of state, which gives us a direct relationship between pressure and temperature. We need to find formulations that can couple pressure with the Navier Stokes equation. Pressure Poisson is one of the methods to solve the problem. There are many algorithms developed to couple Pressure Poisson equation with the Navier Stokes equations. One of them is SIMPLE (Semi Implicit Method for Pressure Correction) originally proposed by Patankar and Spalding [1] in 1972. A revised version of the SIMPLE proposed by Patankar, 1981 [3] which had no gradient of Pressure in the momentum equation. Similarly, SIMPLEC by Doormal and Kaithby,[3] are some of the methods developed to couple Pressure Poisson to Navier Stokes equation to solve for incompressible flow. Lid driven cavity is very widely referred model to test the numerical codes developed, as a lot of literature is available.

The work presented here uses Pressure Poisson equation coupled to Navier Stokes equation to solve a laminar flow in square (LxL) driven cavity. SIMPLE method proposed by Patankar and Spalding 1972 used for pressure and velocity corrections. In addition, a scalr, fourth difference, third order accurate artificial dissipation for stability introduced. In the end solutions obtained compared with benchmarked solutions of Ghia et al [2]

Grid (NxN)	Stretching Ratio	Reynolds Number
41x41	1, and 1.01	100, 400 and 1000
61x61	1, and 1.01	100, 400 and 1000
101x101	1, and 1.01	100, 400 and 1000
129x129	1, and 1.01	100, 400 and 1000
140x140	1, and 1.01	100, 400 and 1000

Table 1: Different Cases Considered for the simulation

METHOD OF SOLUTION

I. GRID GENERATION

Uniform grid generated across the square domain (LxL). Here a general geometric stretching algorithm is used with stretching ratio of r = 1 (fig. 1) to produce a uniform grid and r = 1.01 to produce centrally stretched grid (fig. 2).

Stretching ratio is defined as $r = \frac{(x_{i+1} - x_i)}{(x_i - x_{i-1})}$

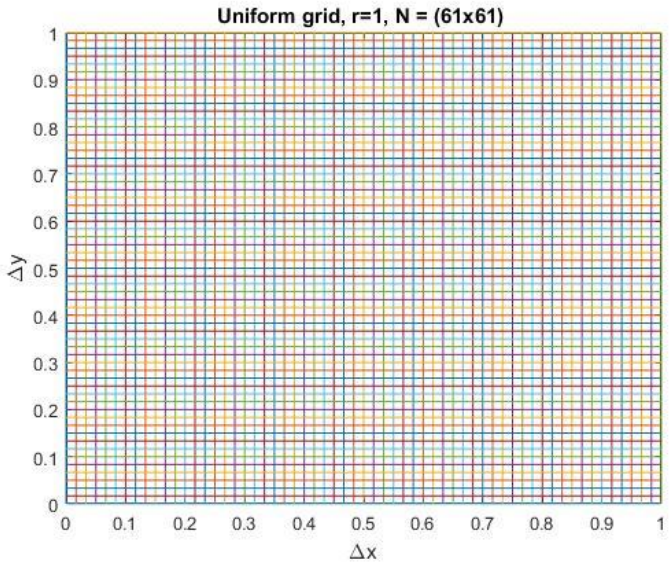

Figure 1: Uniform grid, N = (61x61)

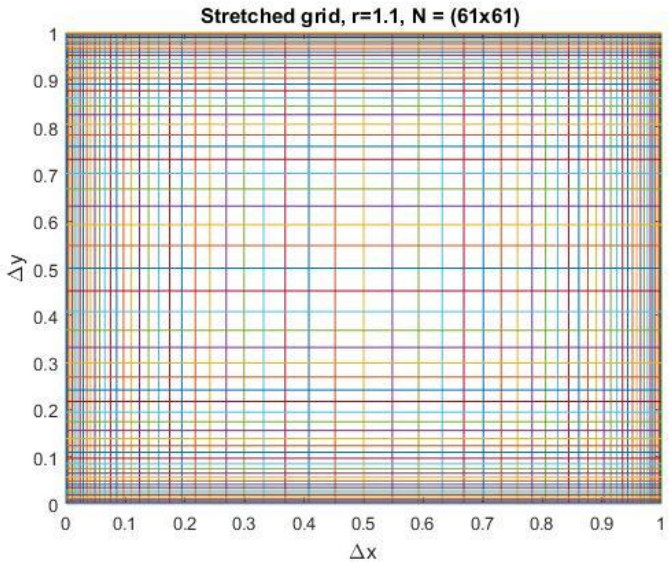

Figure 2: Stretched grid, r=1.1, N=(61x61)

II. GENERALIZED CUVILINEAR COORDINATES

Firstly, the generalized coordinate transformation achieved for curvilinear coordinate system $(x,y) \rightarrow (\xi, \eta)$ where $\xi = \xi(x,y)$, $\eta = \eta(x,y)$. For this purpose first x_ξ, x_η, y_ξ and y_η were calculating central finite difference scheme in the central region, and forward and backward finite difference scheme near the boundaries.

For example,

- $\frac{\partial x}{\partial \xi} = \frac{(x_{i+1} - x_{i-1})}{2}$ (Central difference in space)
- $\frac{\partial x}{\partial \xi} = \frac{(x_{i+1} - x_i)}{2}$ (Forward Difference in space near the boundaries)
- $\frac{\partial x}{\partial \xi} = \frac{(x_i - x_{i-1})}{2}$ (Forward Difference in space near the boundaries)

Jacobian calculated in the following manner:

$$G = \det\begin{bmatrix} x_\xi & y_\xi \\ x_\eta & y_\eta \end{bmatrix} = (x_\xi\, y_\eta - x_\eta\, y_\xi)$$

Jacobian: $J = \frac{1}{G}$

Matrix of transformation calculated in the manner shown below:

$\xi_x = Jy_\eta$, $\xi_y = -Jx_\eta$, $\eta_x = -Jy_\xi$, $\eta_y = Jx_\xi$

Using these, metric tensor matrix is calculated

$$g^{ij} = \begin{bmatrix} g^{11} & g^{12} \\ g^{21} & g^{22} \end{bmatrix} = \begin{bmatrix} \xi_x{}^2 + \xi_y{}^2 & \xi_x\eta_x + \xi_y\eta_y \\ \xi_x\eta_x + \xi_y\eta_y & \xi_x{}^2 + \xi_y{}^2 \end{bmatrix}$$

Co-variant velocities calculated as follows:

$U = u\xi_x + v\xi_y$, $V = u\eta_x + v\eta_y$

Using all above equations, we can do a partial transformation of non-dimensionalised Navier-Stokes equation in curvilinear co-ordinates.

$$\frac{1}{J}\Gamma\frac{\partial Q}{\partial t} + \frac{\partial E^{*1}}{\partial \xi} + \frac{\partial E^{*2}}{\partial \eta} - \frac{\partial E_v^{*1}}{\partial \xi} - \frac{\partial E_v^{*2}}{\partial \eta} = 0$$

where;

$\Gamma = \mathrm{diag}(0, 1, 1)$,

$$Q = \begin{bmatrix} P \\ u \\ v \end{bmatrix},\ E^{*1} = \frac{1}{J}\begin{bmatrix} U \\ uU \\ vU \end{bmatrix},\ E^{*2} = \frac{1}{J}\begin{bmatrix} V \\ uV \\ vV \end{bmatrix},\ E_v^{*1} = \frac{1}{J}\frac{1}{Re}\begin{bmatrix} 0 \\ g^{11}\frac{\partial u}{\partial \xi} + g^{12}\frac{\partial u}{\partial \xi} \\ g^{11}\frac{\partial v}{\partial \eta} + g^{12}\frac{\partial v}{\partial \eta} \end{bmatrix},\ E_v^{*2} = \frac{1}{J}\frac{1}{Re}\begin{bmatrix} 0 \\ g^{12}\frac{\partial u}{\partial \xi} + g^{22}\frac{\partial u}{\partial \xi} \\ g^{12}\frac{\partial v}{\partial \eta} + g^{22}\frac{\partial v}{\partial \eta} \end{bmatrix}$$

All the terms are discretized using three-point stencil (fig. 3).

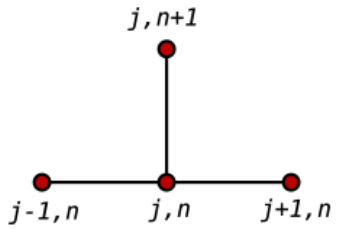

Figure 3: Three Point Stencil

III. *ARTIFICIAL DISSIPATION*

It is a numerical concept pertaining to CFD. It is addition in the numerical scheme used for obtaining stable and smooth solution. Adding dissipation to scheme changes the effective viscosity of the fluid. Hence, it is advisable to add as little as possible.

To calculate artificial dissipation, first the Jacobian matrix calculated as follows:

$$A^j = \frac{1}{J}\begin{bmatrix} 0 & \xi_x^j & \xi_y^j \\ \xi_x^j & U^j + u\xi_x^j & u\xi_y^j \\ \xi_y^j & v\xi_x^j & U^j + v\xi_y^j \end{bmatrix}$$

$\xi^1 = \xi$, $\xi^2 = \eta$, and

$U^1 = U$, $U^2 = V$

The spectral radius calculated using:

$$\rho(A^j) = \frac{1}{J}\left(\left|U^j\right| + \sqrt{(U^j)^2 + g^{jj}}\right)$$

The dissipation calculated using the following finite difference equations:

$$Diss_{(i,j)} = \tilde{\delta}_{\xi} D^1_{(i,j)} + \tilde{\delta}_{\eta} D^2_{(i,j)} = \left(D^1_{(i+1/2,j)} - D^1_{(i-1/2,j)}\right) + \left(D^2_{(i,j+1/2)} - D^2_{(i,j-1/2)}\right)$$

$$D^1_{(i+1/2,j)} = \epsilon\rho(A^1)\left(Q_{(i+2,j)} - 3Q_{(i+1,j)} + 3Q_{(i,j)} - Q_{(i-1,j)}\right)$$

Where ϵ is a small number that controls dissipation. RHS with artificial dissipation used along with grad (P^l) added in the equation*****. The non-dimensionalised system of becomes:

$$\Gamma\frac{\partial Q}{\partial t} = J\left(-\frac{\partial E^{*1}}{\partial \xi} - \frac{\partial E^{*2}}{\partial \eta} + \frac{\partial E_v^{*1}}{\partial \xi} + \frac{\partial E_v^{*2}}{\partial \eta} + Diss - grad(P)\right) = RHS$$

IV. *SIMPLE (**S**EMI **I**MPLICIT **M**ETHOD FOR **P**RESSURE **C**ORRECTION) ALGORITHM* [1]

SIMPLE algorithm is one of the fundamental algorithm to solve incompressible NS equations.

Algorithm used presented in the fig. 4 [2]

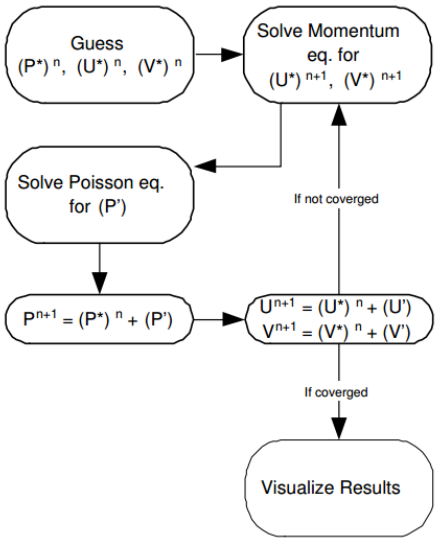

Figure 4: SIMPLE Algorithm Flowchart

Discretization of real time component of time component using Euler Explicit to find u*

$u^* = u_{old} - \Delta t * (RHS)$

$u^* = u_{old} - \Delta t * (RHS)$

Pressure Correction as referred in fig 3 are:

$P^{n+1} = P^n + (\Delta P)$

Where ΔP are the corrections obtained by solving Pressure- Poisson equation.

$\nabla^2(\Delta P) = \frac{1}{\Delta t}\nabla.u^*$

Pressure Poisson solved using Jacobi Method.

$$\Phi^k_{(i+1,j)} + \Phi^k_{(i,j+1)} - 4\Phi^{k+1}_{(i,j)} - \Phi^k_{(i,j+1)} + \Phi^k_{(i,j-1)} = h^2 f^k_{(i,j)}$$

Solve for $\Phi^{k+1}_{(i,j)}$ using iterative method (Jacobi)

Note: Φ = ΔP

Error = $|u - u_{old}|$

V. BOUNDARY CONDITIONS

Boundary conditions must be specified for the momentum equations and the pressure correction equation. The velocity components are zero everywhere along the stationary walls from the noslip condition and the kinematic condition. Along the top moving wall the y-velocity is zero while the xvelocity is taken to be that of the wall. A homogeneous Neumann condition applied for the derivative of the pressure correction field. This is justifiable since the normal components of velocity are zero at all the walls according to the kinematic condition.

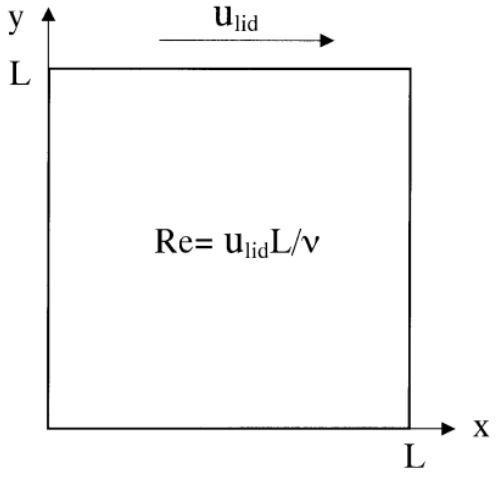

Figure 5: Schematics of the model problem

$\frac{\partial P}{\partial x}$ = 0 at x = 0, 1 $\frac{\partial P}{\partial y}$ = 0 at y = 0, 1

Results and Discussions

MATLAB used to run the above-mentioned procedure. Using geometric stretching, both uniform and stretched grids used to solve the Navier Stokes equation. A mesh sensitivity analysis was performed on stretched grid and it was found out that the stretched grid with r=1.01 provides solution in the margin of ±10% of the benchmarked solution [2].

Figure 6, 7 shows v-velocity calculated at the geometric center of the cavity for Re=100 with number of grid nodes = (41, 61, 80). Observed that the velocity profiles for N= (61 and 80) almost overlap on each other. Hence, the solution is grid independent after N=60.

Concluding that coarse mesh is well suited for flows with low Reynolds as low as 100. Here, Re = 100 was specifically chosen because of the ample literature available. Centrally stretched grid is coarser in the middle of the cavity and finer near the boundaries. Hence, vorticity near the boundaries captured are accurate than uniform grid for same number of node points.

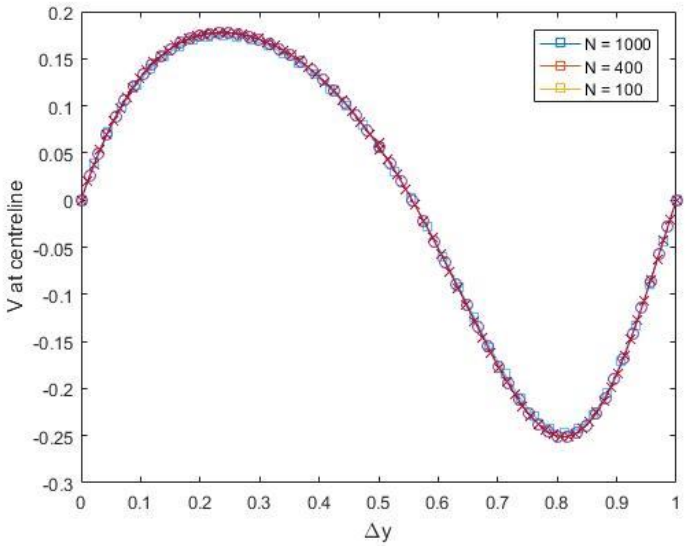

Figure 6: V velocity calculated along the geometric centerline

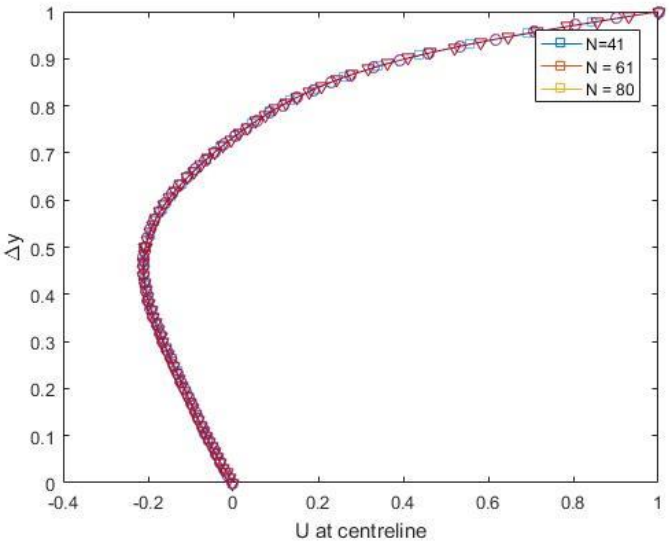

Figure 7: U velocity calculated along the geometric centerline

Furthermore, the stretched grid is not stable for higher Reynolds number as large as 1000. The physical dissipation near the walls make the numerical scheme to be unstable. Because the central differencing scheme used in the central node (not near the boundary) is non-dissipative is nature, at high Reynolds number the value of epsilon (equation **) has to be increase to an approximate value of 0.001 until then adding no dissipation also makes the scheme stable.

Figure 8, 9 and 10 shows the streamlines of the flow in the cavity for Reynolds number = 100, 400 and 1000 respectively. It is highly noticeable that with increase in Reynolds number, the number vortices increase near the boundary. For Re = 100, there are no significant secondary vortices

present in the domain, for Re = 400, there is a vortex formation near the wall of the cavity. Here, we can significantly note that there are two vortices formed at two different places near the boundaries. There are called cavities or secondary vortices.

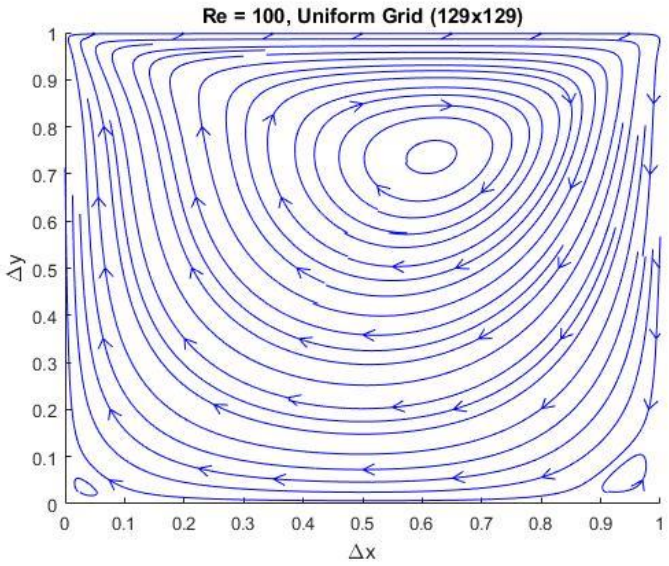

Figure 8: Streamline for Re = 100 on Uniform grid

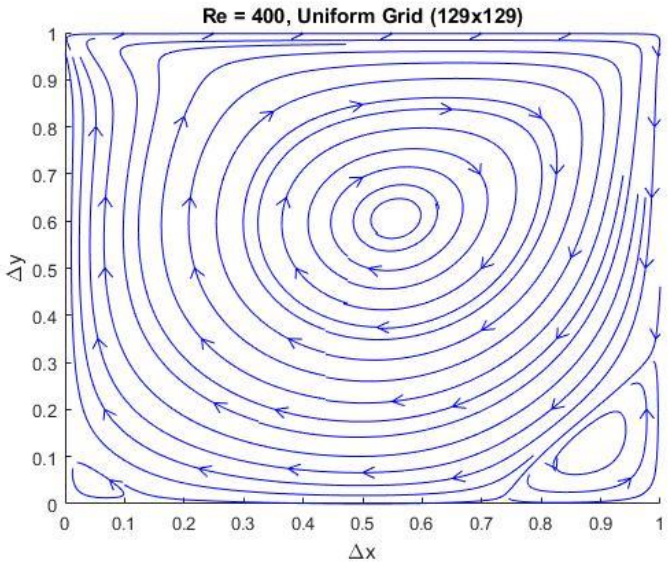

Figure 9: Streamlines for Re = 400 on Uniform grid

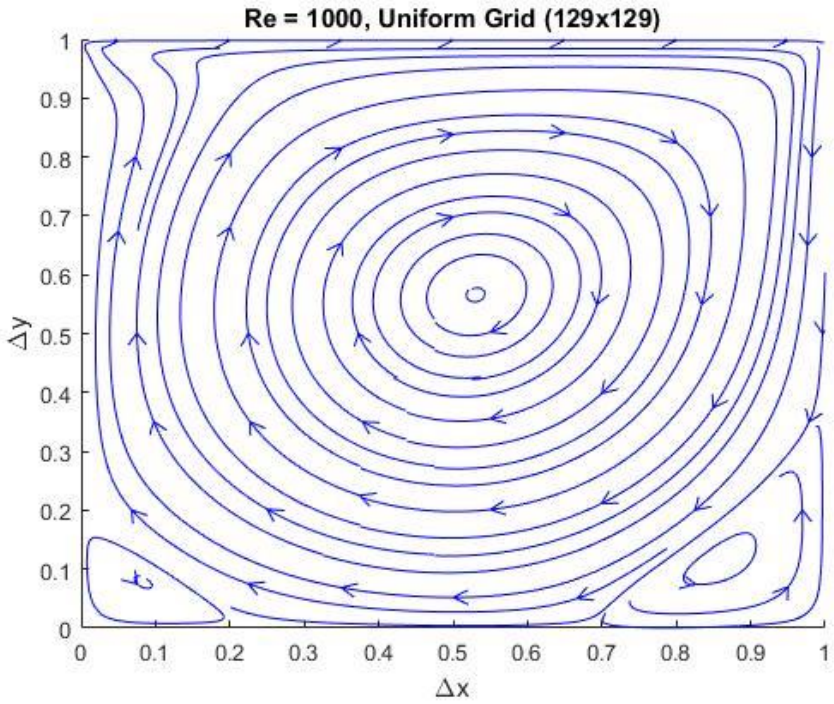

Figure 10: Streamlines for Re = 1000 on Uniform grid

Velocity components (u,v) are plotted along the geometric center to determine the minimum and maximum value of the components. Figure 6 represents the v-velocity plotted against x-

coordinate and similarly, u-velocity plotter against y-coordinate for different Reynolds number. Specifically, N = 129x129 is used, so that we can compare our solution with the bench marked solution [2]. It can be noted that the velocity profile follow the similar trend as present in the bench marked solution. Earlier we have calculated similar profile for Re= 100 with stretched grid.

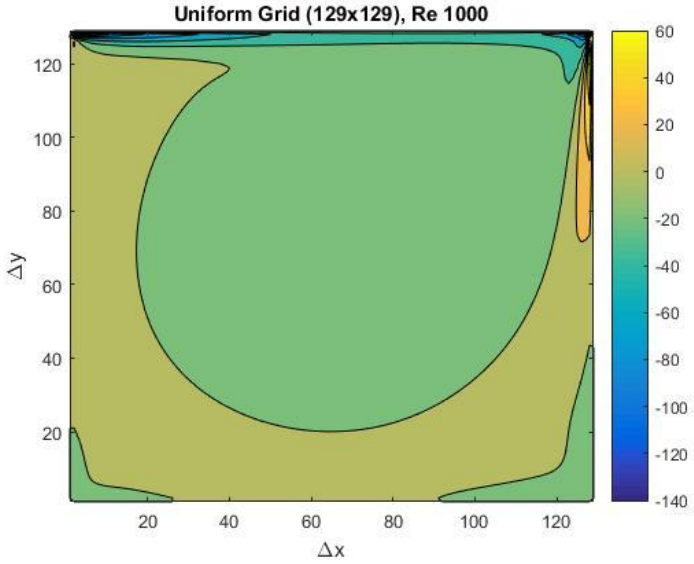

Figure 11: Vorticity field, Re 1000, Uniform grid (129x129)

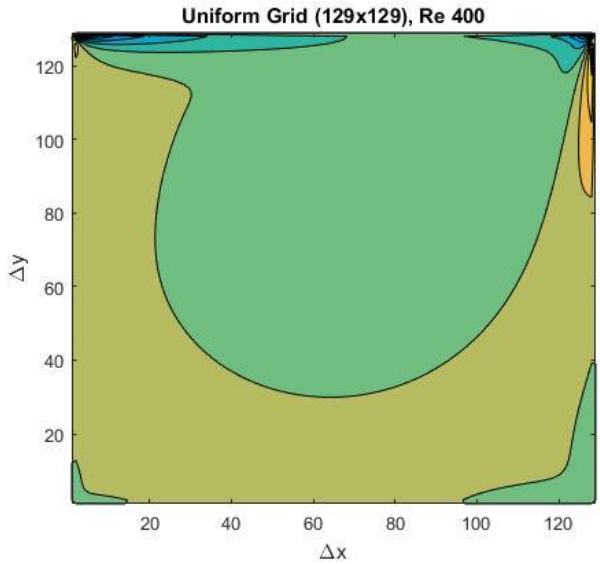

Figure 12: Vorticity field, Re=400, Uniform grid (129x129)

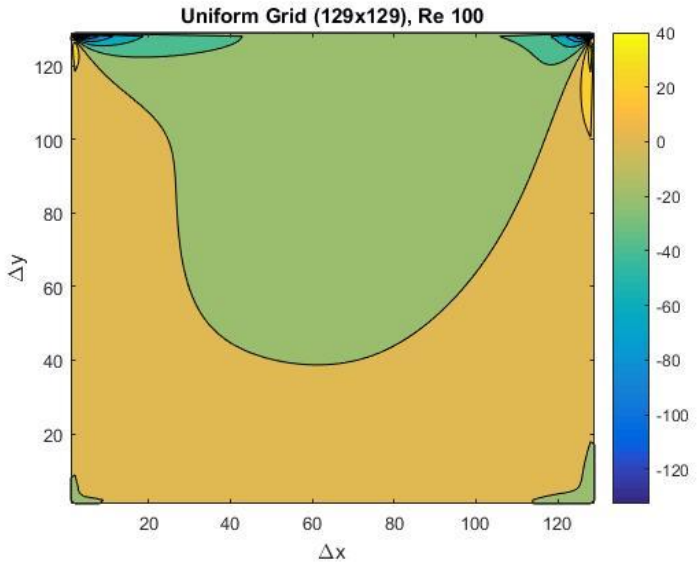

Figure 13: Vorticity field, Re = 100, Uniform grid (129x129)

Vorticity plotted using uniform grid (129x129). Figure 11, 12 and 13 shows vorticity contours field for Reynolds number = 100, 400 and 1000. It can be note that the vortex location changes with

increase in the Reynolds number and the vortex tends to travel in the lower direction with increase in Re. Referring the fig 6c, we can confirm that the vortex has travelled to the bottom of the cavity.

Conclusions

From the above results, concluded that the solution is grid independent for uniform grid with grid nodes greater than (80x80) and (60x60) for centrally stretched grid. There is a limitation to the centrally stretched grid, with increase in Reynolds number the viscous dissipation tends to destabilize the numerical scheme, hence artificial dissipation is added to stabilize the numerical scheme. We cannot add more artificial dissipation because it changes the apparent viscosity of the fluid. Alternative method is reduce the time stepping but it increases the computational cost.

Euler Explicit used to carry out these simulations, which is second order in nature. To improve the convergence a higher order explicit time marching scheme can be used such as 4 order Runge Kutta Method. In addition, revised simple and PISO methods implemented to improve convergence of the simulation.

Here, if the pressure calculated without the reference value, it take a longer time for the variables to converge. We can set arbitrarily any point in the domain to zero and keep is as a reference point for all the pressures in the domain. Here, Jacobi method was used to solve Pressure Poisson method, alternatively direct methods can also be used but they are computational expensive.

REFERENCES

1. Patankar, S. V.; Spalding, D. B. A calculation procedure for Heat, Mass and MomentumTransfer in Three-Dimensional Parabolic Flows. Int. J. Heat Mass Transfer 1972, 15, 1787-1806

2. Ghia, U.; Ghia, K. N.; Shin, C. T. High-Re Solutions for Incompressible Flow using theNavier-Stokes Equations and a Multigrid Method. J. Comput. Phys.1982, 48,387- 411.

APPENDIX

Grid Generation:

```
clc
clear all
sum = 0;
N=61;
r = 1;
L=1;

for i=1:(N-1)/2
   sum = sum + r^(i-1);
end

dx = L/(2*sum);
dy = L/(2*sum);

x(1) = 0;
y(1) = 0;
x(N) = L;
y(N) = L;
j=0;
for i=1:(N-1)
   if i <= (N-1)/2
      x(i+1) = x(i) + dx*r^(i-1);
      y(i+1) = y(i) + dy*r^(i-1);
      x(N-i) = x(N-i+1) - dx*r^(i-1);
      y(N-i) = y(N-i+1) - dy*r^(i-1);
   end
end


[X Y] = meshgrid(x,y);
%
plot(X,Y)
title('Uniform grid, r=1, N = (61x61) ')
xlabel('{\Delta}x')
ylabel('{\Delta}y')
hold on
plot(Y,X)
tic;
```

Code for Laminar incompressible flow in Generalized Curvilinear Co-ordinates

```
clc
clear all
format longEng
r = 1.01;
Ng = [41 61 101 129 141];
for ni = 2

    N = Ng(ni);
    L = 1;
    epsi = 0.0001;
    Re = 100;
    dt = 0.001;

    %Generating Grid
    gridPoints(1) = 0;
    sum = 0;
    x_zeta = ones(N,N); y_eta = ones(N,N);
    zeta_x = ones(N,N); eta_y = ones(N,N);
    G = ones(N,N); J = ones(N,N);

    for i=1:N-1
        sum = sum + r^(i-1);
    end
    dx = L/sum;
    for i=1:N-1
        gridPoints(i+1) = gridPoints(i) + dx*(r^(i-1));
    end

    %storing values of x in 2D array
    for i=1:N
        for j=1:N
            x(i,j) = gridPoints(j);
            y(i,j) = gridPoints(i);
        end
    end

    %Calculating Matrices of transformation x-zeta and y-eta using central diff
    for j=2:N-1
        for i=2:N-1
            x_zeta(j,i) = 0.5*(x(j,i+1) - x(j,i-1));
            x_zeta(1,i) = 0.5*(x(j,i+1) - x(j,i-1));
            x_zeta(N,i) = 0.5*(x(j,i+1) - x(j,i-1));
            y_eta(j,i) = 0.5*(y(j+1,i) - y(j-1,i));
            y_eta(j,1) = 0.5*(y(j+1,1) - y(j-1,1));
            y_eta(j,N) = 0.5*(y(j+1,N) - y(j-1,N));
```

```
    end
end
x_zeta(:,1) = x(:,2) - x(:,1);
x_zeta(:,N) = x(:,N) - x(:,N-1);
y_eta(1,:) = y(2,:) - y(1,:);
y_eta(N,:) = y(N,:) - y(N-1,:);

%Calculating Jacobian
for j=1:N
    for i=1:N
        G(j,i) = x_zeta(j,i)*y_eta(j,i);
        J(j,i) = 1/G(j,i);
    end
end

%Calculating zeta_x, eta_y, g11 and g22
for j=1:N
    for i=1:N
        zeta_x(j,i) = J(j,i)*y_eta(j,i);
        g11(j,i) = zeta_x(j,i)*zeta_x(j,i);
        eta_y(j,i) = J(j,i)*x_zeta(j,i);
        g22(j,i) = eta_y(j,i)*eta_y(j,i);
    end
end

%Calculating g11/J and g22/J at half points
for j=2:N-1
    for i=2:N-1
        g11_half(j,i) = 0.5*(g11(j,i+1) + g11(j,i));
        g22_half(j,i) = 0.5*(g22(j+1,i) + g22(j,i));
    end
end

%Initializing Matrices
for j=1:N
    for i=1:N
        Q(j,i,1) = 0; %Initial Pressure
        Q(j,i,2) = 0; %Initial Ux
        Q(j,i,3) = 0; %Initial Vx
        P_old(j,i) = 0;
        U_old(j,i) = 0;
        V_old(j,i) = 0;
        dp(j,i) = 0; %Initial Matrix for delta pressure in SIMPLE
        ustar(j,i) = 0;
        vstar(j,i) = 0;
        vorticity(j,i) = 0;
```

```
    end
end

%Imposing Boundary conditions on U and ustar
Q(N,:,2) = 1;
u_new = zeros(N,N);
v_new = zeros(N,N);
u_new(N,:) = 1;
ustar(N,:) = 1;
rho_1 = zeros(N,N); %similar to variable declaration
rho_2 = zeros(N,N); %similar to variable declaration

Diss_x = zeros(N,N);
Diss_y = zeros(N,N);
div = zeros(N,N);

epsilon_p = 1E-5;
epsilon_u = 1E-5;
epsilon_v = 1E-5;

error_p = 1;
error_u = 1;
error_v = 1;

total_error_p = 1;
total_error_u = 1;
total_error_v = 1;

itr = 0;
iteration = 0;
%Convergence loop for time
while total_error_u > epsilon_u && total_error_v > epsilon_v
    %for iteration=1:40000

    iteration = iteration + 1

    for j=1:N
        for i=1:N
            P_old(j,i) = Q(j,i,1);
            U_old(j,i) = Q(j,i,2);
            V_old(j,i) = Q(j,i,3);
        end
    end

    error_p = 1;
    error_u = 1;
```

```
error_v = 1;

while error_p > epsilon_p && error_u > epsilon_u && error_v > epsilon_v


   %Computing Contravarient velocities.
   for j=1:N
      for i=1:N
         U(j,i) = Q(j,i,2) * zeta_x(j,i);
         V(j,i) = Q(j,i,3) * eta_y(j,i);
      end
   end

   %Calculating Spectral Radius
   for j=2:N-1
      for i=2:N-1
         rho_1(j,i) = (abs(U(j,i)) + sqrt(U(j,i)^2 + g11(j,i)))/J(j,i);
         rho_2(j,i) = (abs(V(j,i)) + sqrt(V(j,i)^2 + g22(j,i)))/J(j,i);
      end
   end

   %Calculating Spectral Radius at half Points
   for j=2:N-1
      for i=2:N-1
         rhox(j,i) = 0.5*(rho_1(j,i+1) + rho_1(j,i));
         rhoy(j,i) = 0.5*(rho_2(j+1,i) + rho_2(j,i));
      end
   end

   %Calculating Q at half points to be used in dissipation function
   for k=1:3
      for j=3:N-2
         for i=3:N-2
            Q_half_x(j,i,k) = Q(j,i+2,k) - 3*Q(j,i+1,k) + 3*Q(j,i,k) -  Q(j,i-1,k);
            Q_half_y(j,i,k) = Q(j+2,i,k) - 3*Q(j+1,i,k) + 3*Q(j,i,k) -  Q(j-1,i,k);
         end
      end
   end

   %Calculating Dissipation Function
   for k=1:3
      for j=3:N-2
         for i=3:N-2
            Diss_x(j,i,k) = epsi*((rhox(j,i)*Q_half_x(j,i,k)) - (rhox(j,i-1)*Q_half_x(j,i-1,k)));
            Diss_y(j,i,k) = epsi*((rhoy(j,i)*Q_half_y(j,i,k)) - (rhoy(j-1,i)*Q_half_y(j-1,i,k)));
         end
```

```
    end
end

%Calculating Convective terms and adding dissipation to them
for j=1:N
    for i=1:N
        E_1_compute_x(j,i,1) = U(j,i)/J(j,i);
        E_1_compute_x(j,i,2) = ((Q(j,i,2)*U(j,i))/J(j,i));
        E_1_compute_x(j,i,3) = (Q(j,i,3)*U(j,i))/J(j,i);

        E_2_compute_y(j,i,1) = V(j,i)/J(j,i);
        E_2_compute_y(j,i,2) = ((Q(j,i,2)*V(j,i)))/J(j,i);
        E_2_compute_y(j,i,3) = ((Q(j,i,3)*V(j,i)))/J(j,i);
    end
end

for k=1:3
    for j=2:N-1
        for i=2:N-1
            DE_1_compute_x(j,i,k) = 0.5*(E_1_compute_x(j,i+1,k) - E_1_compute_x(j,i-
1,k)) + Diss_x(j,i,k);
            DE_2_compute_y(j,i,k) = 0.5*(E_2_compute_y(j+1,i,k) - E_2_compute_y(j-
1,i,k)) + Diss_y(j,i,k);
        end
    end
end

%Calculating Diffusive terms, here no numerical dissipating needed
%First Calculating at half points to maintain 3 point stencil
for j=1:N-1
    for i=1:N-1
        J_half_x(j,i) = 0.5*(J(j,i+1) + J(j,i));
        J_half_y(j,i) = 0.5*(J(j+1,i) + J(j,i));

        zeta_x_half(j,i) = (0.5*(zeta_x(j,i+1) + zeta_x(j,i)))^2;
        eta_y_half(j,i) = (0.5*(eta_y(j+1,i) + eta_y(j,i)))^2;

        u_11(j,i) = (Q(j,i+1,2)-Q(j,i,2));
        u_12(j,i) = (Q(j,i+1,3)-Q(j,i,3));

        Ev_1_compute_x(j,i,1) = 0;
        Ev_1_compute_x(j,i,2) = (zeta_x_half(j,i)*u_11(j,i)/J_half_x(j,i));
        Ev_1_compute_x(j,i,3) = (zeta_x_half(j,i)*u_12(j,i)/J_half_x(j,i));

        u_21(j,i) = (Q(j+1,i,2)-Q(j,i,2));
        u_22(j,i) = (Q(j+1,i,3)-Q(j,i,3));
```

```
            Ev_2_compute_y(j,i,1) = 0;
            Ev_2_compute_y(j,i,2) = (eta_y_half(j,i)*u_21(j,i)/J_half_y(j,i));
            Ev_2_compute_y(j,i,3) = (eta_y_half(j,i)*u_22(j,i)/J_half_y(j,i));
        end
    end

    %Calculating viscous terms
    for k=2:3
        for j=2:N-1
            for i=2:N-1
                DEv_1_compute_x(j,i,k) = (Ev_1_compute_x(j,i,k) - Ev_1_compute_x(j,i-
1,k))/Re;
                DEv_2_compute_y(j,i,k) = (Ev_2_compute_y(j,i,k) - Ev_2_compute_y(j-
1,i,k))/Re;
            end
        end
    end

    %Calculating Gradient of Pressure
    for j=2:N-1
        for i=2:N-1
            grad_P_x(j,i) = J(j,i)*0.5*((Q(j,i+1,1)*zeta_x(j,i+1)/J(j,i+1)) - (Q(j,i-1,1)*zeta_x(j,i-
1)/J(j,i-1)));
            grad_P_y(j,i) = J(j,i)*0.5*((Q(j+1,i,1)*eta_y(j+1,i)/J(j+1,i)) - (Q(j-1,i,1)*eta_y(j-
1,i)/J(j-1,i)));
        end
    end

    %Calcuting RHS in computational domain
    for k=1:3
        for j=2:N-1
            for i = 2:N-1
                RHS_compute(j,i,k) = J(j,i)*(-DE_1_compute_x(j,i,k) -DE_2_compute_y(j,i,k)
+DEv_1_compute_x(j,i,k) +DEv_2_compute_y(j,i,k));
            end
        end
    end

    %Calcuting ustar in SIMPLE Algorithm
    for j=2:N-1
        for i=2:N-1
            ustar(j,i) = Q(j,i,2) + dt*(RHS_compute(j,i,2) - grad_P_x(j,i));
            vstar(j,i) = Q(j,i,3) + dt*(RHS_compute(j,i,3) - grad_P_y(j,i));
        end
    end
```

```
epsilon_dp = 1E-6;
error_dp = 1;

%Calculating Divergence of ustar
for j=2:N-1
   for i=2:N-1
      div(j,i) = 0.5*J(j,i)*((ustar(j,i+1)*zeta_x(j,i+1)/J(j,i+1))-(ustar(j,i-1)*zeta_x(j,i-
1)/J(j,i-1)) + (vstar(j+1,i)*eta_y(j+1,i)/J(j+1,i)) - (vstar(j-1,i)*eta_y(j-1,i)/J(j-1,i)));
   end
end

dp = zeros(N,N);
dp1 = zeros(N,N);

while error_dp > epsilon_dp % Loop for convergence of dp

   %Storing Pressure in new Variable
   for j=2:N-1
      for i=2:N-1
         dp(j,i) = dp1(j,i);
      end
   end

   %Solve Pressure Poisson
   for j=2:N-1
      for i=2:N-1
         a = J(j,i)*((g11_half(j,i)*dp(j,i+1)/J_half_x(j,i)) + (g11_half(j,i-1)*dp(j,i-
1)/J_half_x(j,i-1)) + (g22_half(j,i)*dp(j+1,i)/J_half_y(j,i)) + (g22_half(j-1,i)*dp(j-1,i)/J_half_y(j-
1,i)));
         b = -J(j,i)*((g11_half(j,i)/J_half_x(j,i)) + (g11_half(j,i-1)/J_half_x(j,i-1)) +
(g22_half(j,i)/J_half_y(j,i)) + (g22_half(j-1,i)/J_half_y(j-1,i)));
         dp1(j,i) = ((div(j,i)/dt) - a)/b;

         error_dp = error_dp + (abs((dp1(j,i)) - (dp(j,i))))^2; %Calculating error for delta
Pressure
         error_dp = sqrt(error_dp)/(N*N);
      end
   end
end

%Calculating Gradient of delta pressure
for j=2:N-1
   for i=2:N-1
      dp(j,i) = dp1(j,i);
```

```
            grad_dp_x(j,i) = 0.5*((dp(j,i+1)*zeta_x(j,i+1)/J(j,i+1))- (dp(j,i-1)*zeta_x(j,i-1)/J(j,i-1))); %Using Partial Transformation
            grad_dp_y(j,i) = 0.5*((dp(j+1,i)*eta_y(j+1,i)/J(j+1,i)) - (dp(j-1,i)*eta_y(j-1,i)/J(j-1,i))); %Using Partial Transformation
        end
    end

    error_p = 0;
    error_u = 0;
    error_v = 0;
    %Correcting Pressure
    for j=2:N-1
        for i=2:N-1
            Pressure_new(j,i) = Q(j,i,1) + dp(j,i);
            u_new(j,i) = ustar(j,i) - dt*J(j,i)*grad_dp_x(j,i);
            v_new(j,i) = vstar(j,i) - dt*J(j,i)*grad_dp_y(j,i);
        end
    end

    Pressure_new(1,:) = Pressure_new(2,:);
    Pressure_new(N,:) = Pressure_new(N-1,:);
    Pressure_new(:,1) = Pressure_new(:,2);
    Pressure_new(:,N) = Pressure_new(:,N-1);

    %Calculating Error of corrections
    for j=2:N-1
        for i=2:N-1
            error_p = error_p + (abs(Pressure_new(j,i) - (Q(j,i,1))))^2;
            error_u = error_u + (abs(u_new(j,i) - (ustar(j,i))))^2;
            error_v = error_v + (abs(v_new(j,i) - (vstar(j,i))))^2;
        end
    end

    error_p = sqrt(error_p)/(N*N);
    error_u = sqrt(error_u)/(N*N);
    error_v = sqrt(error_v)/(N*N);

    for j=1:N
        for i=1:N
            Q(j,i,1) = Pressure_new(j,i);
            Q(j,i,2) = u_new(j,i);
            Q(j,i,3) = v_new(j,i);
            Q(1,i,1) = Q(2,i,1);
            Q(N,i,1) = Q(N-1,i,1);
            Q(j,1,1) = Q(j,2,1);
            Q(j,N,1) = Q(j,N-1,1);
```

```
            end
        end
    end

    total_error_p = 0;
    total_error_u = 0;
    total_error_v = 0;

    for j=1:N
        for i=1:N
            total_error_p = total_error_p + abs(Q(j,i,1) - P_old(j,i));
            total_error_u = total_error_u + abs(Q(j,i,2) - U_old(j,i));
            total_error_v = total_error_v + abs(Q(j,i,3) - V_old(j,i));
            itr = itr + 1;

            Error_iteration_p(iteration) = total_error_p;
            Error_iteration_u(iteration) = total_error_u;
            Error_iteration_v(iteration) = total_error_v;
        end
    end
end

%Calculating Vorticity
for j=2:N-1
    for i=2:N-1
        vorticity(j,i) = J(j,i)*(((v_new(j,i+1)*zeta_x(j,i+1)/J(j,i+1)) - (v_new(j,i-1)*zeta_x(j,i-
1)/J(j,i-1))) - ((u_new(j+1,i)*eta_y(j+1,i)/J(j+1,i)) - (u_new(j-1,i)*eta_y(j-1,i)/J(j-1,i))));
    end
end

if ni == 1
    xlswrite('x1.xlsx',x);
    xlswrite('y1.xlsx',y);
    xlswrite('u1.xlsx',u_new);
    xlswrite('v1.xlsx',v_new);
    xlswrite('vor1.xlsx',vorticity);
    save('error_u1.mat','Error_iteration_u');
    save('error_v1.mat','Error_iteration_v');
    save('error_p1.mat','Error_iteration_p');
end

if ni == 2
    xlswrite('x2.xlsx',x);
    xlswrite('y2.xlsx',y);
    xlswrite('u2.xlsx',u_new);
    xlswrite('v2.xlsx',v_new);
```

```
        xlswrite('vor2.xlsx',vorticity);
        save('error_u2.mat','Error_iteration_u');
        save('error_v2.mat','Error_iteration_v');
        save('error_p2.mat','Error_iteration_p');
    end

    if ni == 3
        xlswrite('x3.xlsx',x);
        xlswrite('y3.xlsx',y);
        xlswrite('u3.xlsx',u_new);
        xlswrite('v3.xlsx',v_new);
        xlswrite('vor3.xlsx',vorticity);
        save('error_u3.mat','Error_iteration_u');
        save('error_v3.mat','Error_iteration_v');
        save('error_p3.mat','Error_iteration_p');
    end

    if ni == 4
        xlswrite('x4.xlsx',x);
        xlswrite('y4.xlsx',y);
        xlswrite('u4.xlsx',u_new);
        xlswrite('v4.xlsx',v_new);
        xlswrite('vor4.xlsx',vorticity);
        save('error_u4.mat','Error_iteration_u');
        save('error_v4.mat','Error_iteration_v');
        save('error_p4.mat','Error_iteration_p');
    end

    if ni == 5
        xlswrite('x5.xlsx',x);
        xlswrite('y5.xlsx',y);
        xlswrite('u5.xlsx',u_new);
        xlswrite('v5.xlsx',v_new);
        xlswrite('vor5.xlsx',vorticity);
        save('error_u5.mat','Error_iteration_u');
        save('error_v5.mat','Error_iteration_v');
        save('error_p5.mat','Error_iteration_p');
    end

% xlsxwrite('p.xlsx',Pressure_new);

%     figure(1);
%     contour(x,y,vorticity)
%     colorbar;
%     title('Vorticity for Re,N = ');
%     xlabel('{\Delta}x')
```

```
%     ylabel('{\Delta}y')
%     % saveas(fig,'vorticity.jpg')
%
%     figure(2);
%     quiver(x,y,u_new,v_new);
%     title('Velocity vectors for Re,N = ');
%     xlabel('{\Delta}x');
%     ylabel('{\Delta}y');
%     % saveas(fig,'velocity_vec.jpg')
%
%     figure(3);
%     streamslice(x,y,u_new,v_new);
%     title('RE = Re, Uniform Grid (N*N)')
%     xlabel('{\Delta}x');
%     ylabel('{\Delta}y')
    % saveas(fig,'streamlines.jpg')
end
toc;
```