2D INCOMPRESSIBLE NAVIER STOKES SOLVER FOR LAMINAR FLOW OVER BACKARD FACING STEP

MAE540 ENDTERM: COMPUTATIONAL FLUID MECHANICS

SUMEET LULEKAR (PERSON NUMBER- 50203322)

DATE OF SUBMISSION: 19th MAY, 2017

To Prof. Iman Borazjani

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 which had no gradient of Pressure in the momentum equation. Similarly, SIMPLEC by Doormal and Kaithby are some of the methods developed to couple Pressure Poisson to Navier Stokes equation to solve for incompressible flow.

The work presented here uses Pressure Poisson equation coupled to Navier Stokes equation to solve laminar flow over a backward facing step of height h. SIMPLE method proposed by Patankar and Spalding 1972 used for pressure and velocity correlations. In addition, a scalar, fourth difference, third order accurate artificial dissipation for stability introduced. To accelerate the convergence rate of the method, local time stepping and the method of residual smoothing used. In the end solutions obtained compared with benchmarked solutions of Armaly et al[2].

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 $x\xi$, $x\eta$, $y\xi$ and $y\eta$ 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;

Γ = 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).

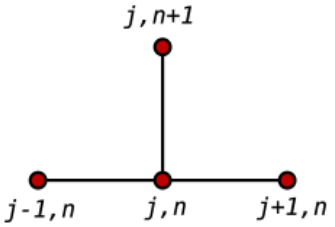

Figure 1: Three Point Stencil

METHODOLOGY

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, U2 = V

The spectral radius calculated using:

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

The dissipation calculated using the following finite difference equations:

$$Diss\ (i, j) = \tilde{\delta}_\xi D^1_{(i,j)} + \tilde{\delta}_\eta D^2_{(i,j)}$$

$$= (D^1_{(i+1/2,\,j)} - D^1_{(i-1/2,\,j)}) + (D^2_{(i,\,j+1/2)} - D^2_{(i,\,j-1/2)})$$

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

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

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

SIMPLE (SEMI IMPLICIT METHOD FOR PRESSURE CORRECTION) ALGORITHM [1]

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

Algorithm used presented in the fig. 4 [2]

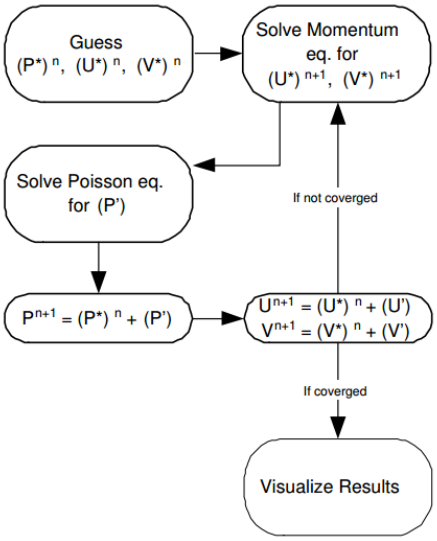

Figure 2: SIMPLE Algorithm Flowchart

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 equation

Pressure Poisson solved using Jacobi Method.

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

Where, $\Phi = \Delta P$ (Pressure correction)

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

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

BOUNDARY CONDITIONS AND INTIAL CONDITIONS

Every problem in computational fluid dynamics is unique because of the boundary conditions. We have considered, the flow has a parabolic velocity profile at the inlet given by,

$$u(y) = U_{bulk} * \left(1 - \left(\frac{y - 1.5*h}{0.5*h}\right)^2\right)$$

Where, Ubulk is the bulk velocity at the inlet.

For the stationary walls, the top wall and the bottom wall, we have no slip condition. Fully developed flow at the right boundary, which dictates gradient of u-velocity and v–velocity in the x-direction, are zero.

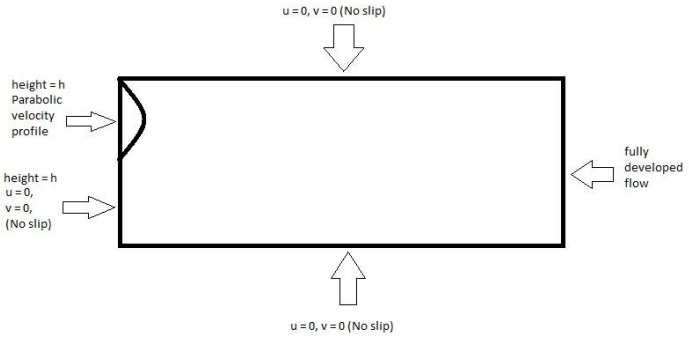

Figure 3: Boundary conditions for backward facing step

LOCAL TIME STEPPING

To obtain the local time step, we use both Courant Friedrich's Lewy number (CFL number) and Von Neumann number (VN number). Based on the CFL number, $d\tau$ is calculated as follows,

$$d\tau_{CFL} = \frac{CFL}{J(\max(\rho(A^1), \rho(A^2), \rho(A^3)_{(i,j)})}$$

Where,

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

Now, using VN, $d\tau$ is calculated as follows.

$$d\tau_{VN} = \frac{VN * Re}{max(g^{11}, g^{22}, g^{33})_{(i,j)}}$$

Where, g^{11}, g^{22} and g^{33} are calculated from the matrix of transformation. From the above methods, the minimum of the two ($d\tau_{CFL}$, $d\tau_{VN}$) is taken as the local time step. Local time stepping used to increase the convergence rate and thereby reach the steady state faster.

RESIDUAL SMOOTHING

Residual smoothing implemented to increase the stability of the numerical scheme. Due to large fluctuations in the RHS of the Navier Stokes equations defined as,

$$\frac{\overrightarrow{\partial Q}}{\partial t} = \overrightarrow{RHS}$$

We need to smooth it to increase the stability and reach the steady state faster. Residual smoothing carried as follows,

$$(1-\epsilon_\xi \delta_{\xi\xi})(1-\epsilon_\eta \delta_{\eta\eta})\, \overrightarrow{RHS}_{smooth} = \overrightarrow{RHS}_{original}$$

As, the equation is implicit in nature, the ADI scheme is employed for solving the equation. It is discretized in ξ-direction, first for which the equation is given as,

$$(1-\epsilon_\xi \delta_{\xi\xi})\, \vec{U} = \overrightarrow{RHS}$$

The discretized form of the above equation given by,

$$\vec{U}_{(i,j)} - \epsilon_\xi \frac{\vec{U}_{(i,j+1)} - 2\vec{U}_{(i,j)} + \vec{U}_{(i-1,j)}}{(\Delta\xi)^2} = \overrightarrow{RHS}_{(i,j)}$$

The equation is implicit in nature; Thomas algorithm used to solve the above-mentioned equation. After

smoothing in ξ-direction, we discretized the smoothed values in η-direction as follows,

$$(1-\epsilon_\eta \delta_{\eta\eta})\ \overrightarrow{RHS}_{\text{smooth}} = \vec{U}$$

As done previously, the equation can be discretized using the 2nd order accurate central difference scheme, and can be given as,

$$\overrightarrow{RHS}_{(i,j)} - \epsilon \frac{\overrightarrow{RHS}_{(i,j+1)} - 2\overrightarrow{RHS}_{(i,j)} + \overrightarrow{RHS}_{(i,j)}}{2(\Delta\eta)^2} = \vec{U}$$

Again, the above equation being implicit in nature and is a tridiagonal system, it is solved using Thomas algorithm. Thomas algorithm is computationally less expensive hence preferred to solve the tridiagonal system.

GRID INDEPENDENCE

Grid independence is necessary for every computational fluid dynamics analysis as it shows the results obtained as independent of the grid. Grid independence study conducted by first generating solutions using coarse mesh. Finer mesh generated and solutions calculated again. If the solutions are not changing with refining the mesh, we can say that the solution is now grid independent. Grid independence study carried out further in the sections.

DISCUSSION OF RESULTS

FLOW OVER BACKWARD FACING STEP OF HEIGHT h

For present case, length of domain is sufficiently large (height = 2 units, length = 40) so that fully developed is obtained. Simulations are carried out for Reynolds number = 50, 200, 325 and 500 with grid size as 400x40. (400 nodes in x-direction and 40 nodes in y-direction). As mentioned earlier in the boundary conditions section, the profile for inlet velocity considered parabolic in nature with bulk velocity equal to 1 (unity). 2D incompressible solver integrated with local stepping and residual smoothing in addition with artificial dissipation. CFL and VN numbers taken as 0.1 respectively.

To consider any results we have to determine that the solutions are grid independent. Grid independence study is carried for Reynold number = 50. Table 1 shows

Table 1: Grid independence study for Re=50

Grid Size (Nodes)	Reattachment Length
100x10	2.8
200x20	2.9
200x40	3
300x30	3
400x40	3

It can be clearly seen that the solution is not changing after 400x40 nodes with (height = 2, Length =40). Earlier the simulations carried out for length =60 but it was observed that domain of such a large length is not needed and reducing its size provides similar results with no dependency on the length > 40. Hence, Length = 40 was chosen as the domain size. The height (= 2) of the domain is dictated by the problem statement.

Simulations are carried out for Re =50, 200, 325 and 400 and the streamlines for each are plotted fig 3, 5, 7, 9. Figure 4, 6, 8, 10 shows the circulation zone or the coherent structures in the flow.

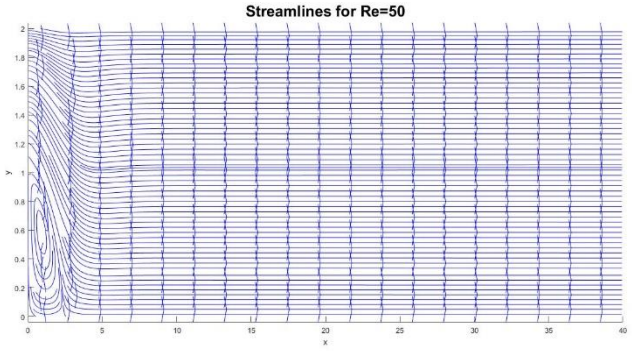

Figure 4: Streamline Plot for Re = 50

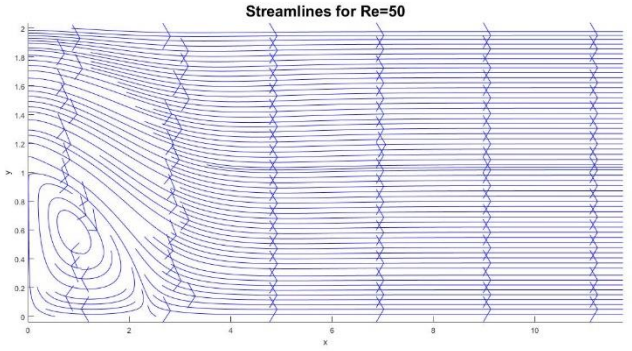

Figure 5: Streamline plot for realization of circulation zone for Re=50

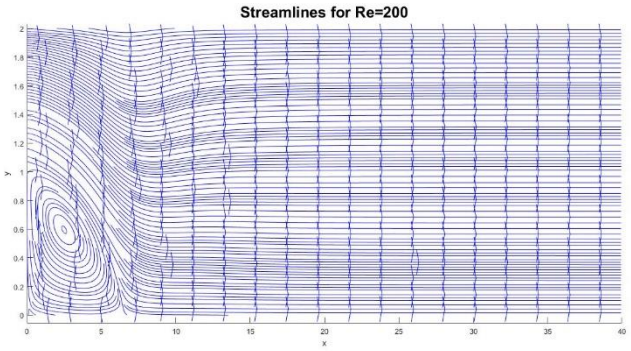

Figure 6: Streamline Plot for Re = 200

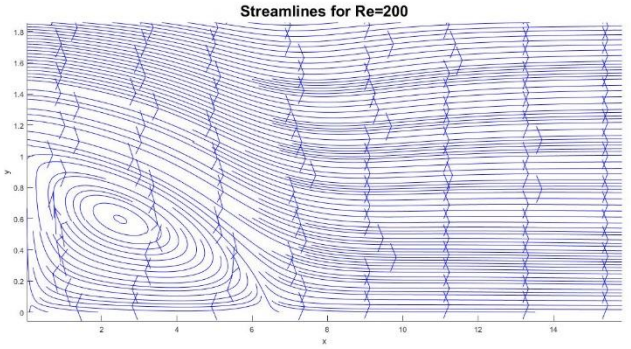

Figure 7: Streamline plot for realization of circulation zone for Re=200

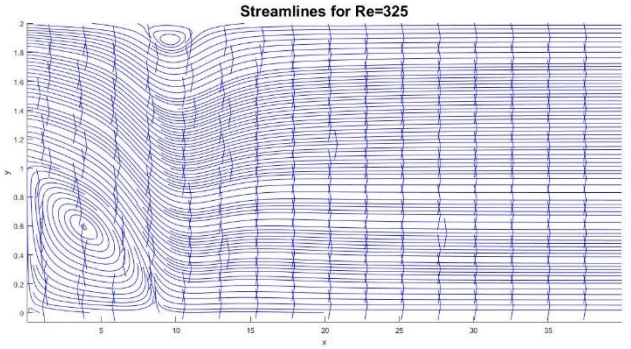

Figure 8: Streamline Plot for Re=325

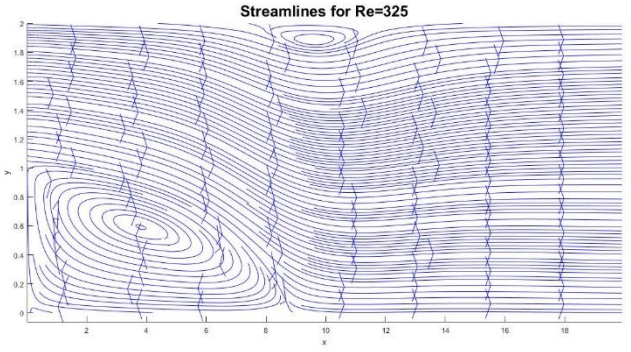

Figure 9: Streamline plot for realization of circulation zones for Re= 325

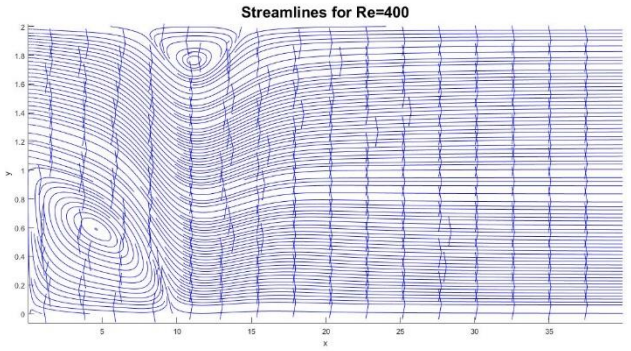

Figure 10: Streamline plot for Re = 400

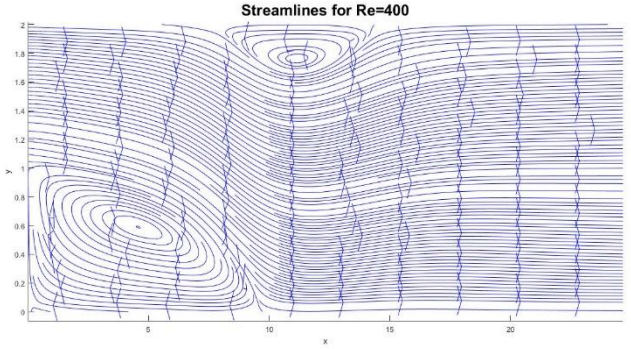

Figure 11: Streamline plot for realization of circulation zone for Re = 400

As Reynolds number increases it is evident that the reattachment length increases and for Re greater than 200 we can see vortex structures on the top of the wall. The second region of the recirculating flow then be due to the sharp change in the direction of the flow[3], as evident for Reynolds number higher than 100 (here Re = 325 and 400).

CFL = 0.1 taken for all the simulations carried with explicit Euler method for time marching. Further, artificial dissipation introduced to stabilize the numerical schemes. Although introducing artificial dissipation results in change in the viscosity of the fluid as we have solving for some other fluid. Hence, epsilon = 0.001 was the magnitude of the artificial dissipation introduced. Formulation for artificial dissipation discussed in the previous sections.

Below in the table 2. Numerical results for reattachment length compared with Armaly et al.

Table 2: Comparison of results with Armaly et al

Reynolds Number	**Reattachment length (x/h)**		
	Computed	**Armaly**	
		Experiment	**Computed**
50	3	3	3
200	7	8	8
325	9	9.8	6.8
400	9.8	10	7.9

The reattachment length visualized from the fig. 4, 6, 8, and10. From table 2 we can conclude that the solutions obtained from the numerical simulations strongly agree with the experimental results of Armaly et al for Re = 50. For Re > 100 (here Re = 200,325 and 400) we see that our solutions closely agree with the experimental results. Thereby, concluding that our numerical schemes are reliable and the solutions obtained also grid independent. Note that all the data presented here are for the domain with height =2, length = 40 and grid nodes = 400x40 (Nx,Ny).

For higher Reynolds number (greater than 100), we see that the numerical solutions and the experimental results do not match exactly. This can be due to the 3D dimensionality of the flow. As Armaly also predicted for Reynolds number higher than 400, the flow transitions from laminar to turbulence.

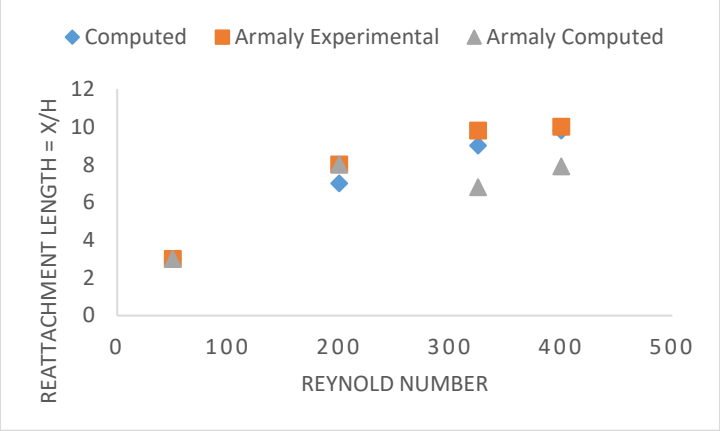

Figure 12: Comparison of numerical solutions with Armaly et al.

Here, the reattachment length is non-dimensionalised with height of the backward facing step (h=1). Figure 11, shows the comparison of solutions with Armaly et al.

STUDY OF EFFECT OF LOCAL TIME STEPPING ON THE RATE OF CONVERGENCE

To accelerate the rate of convergence we used local time stepping method. The formulations for local time stepping discussed in the previous sections. The general idea to use local stepping is to keep the Courant Freidrich's Lewy (CFL) number constant throughout the domain. CFL number generally dictates the stability of the numerical scheme used to discretize the governing equations. The effect of local time stepping shown in table 3.

Table 3 Comparison of local and constant time step for Re=50

Time Step	No. of Iterations
0.001	67007
0.005	50214
Local time stepping	24125

Reducing the constant time step increases the number of iterations i.e. reduction in convergence rate. Hence, local time stepping implemented to increase the rate of convergence. Table 3 clearly shows that local time stepping increases the rate of convergence twice.

STUDY OF EFFECT OF CFL NUMBER, ARTIFICIAL DISSIPATION, IMPLICIT RESIDUAL SMOOTHING ON THE RATE OF CONVERGENCE OF THE TIME MARCHING ALGORITHM

Effect of CFL number

The effect of CFL number discussed in this section. Table 4 shows the effect of cfl number. It is evident that decreasing the cfl number the rate of convergence decreases. Here the local time step value increases which leads to faster simulations towards steady state.

Table 4: Effect of cfl number for Re = 50

CFL Number	No. of Iterations
0.1	24125
0.01	42486

Effect of artificial dissipation

Artificial dissipation introduced to increase the stability of the numerical schemes used. The aim is to introduce artificial dissipation as low as possible because it changes the viscosity of the fluid. Here, epsilon = 0.001 is found to be optimum to stabilize the numerical simulation. Formulations mentioned in the previous section.

Effect of Residual Smoothing

Implicit Residual smoothing applied to dampen out the oscillations with high frequency and obtain smooth solutions. Carried out to increase the rate of convergence. It has the same effect as artificial dissipation. Table 5 shows the effect of residual smoothing

Table 5: Effect of Residual Smoothing for Re= 50

Smoothing Constant	No. of Iterations
0.1	24125
0.01	23140
0.001	23021
0.0001	23001

Here, we can say that for Re = 50 we do not have higher frequency oscillations hence the effect of residual smoothing on the rate of convergence is not that critical.

SUMARY AND CONCLUSIONS

2D incompressible Navier Stokes solver couple with local time stepping and residual smoothing used to simulate the laminar flow over backward facing step. The solver implements SIMPLE algorithm proposed by Patankar and Splading in 1972 with artificial dissipation for numerical stability. The grid independence study was conducted to find out that the length = 40, height = 2 and grid node = 400x40 is sufficient to simulate the flow over a backward facing step. It was found out that the results of the simulation agree with experimental results of Armaly et al. for Re < 100. The results slightly deviate from the experimental results due the fact that the flow in the experiments are 3D dimensional in nature. In fact Armaly et al claims that the flow transitions from laminar to turbulent region for Re > 400 hence the deviations in the numerically results from the experimental results are justified.

The simulations were carried for Re = 50, 200, 325, and 400. It is observed that the reattachment length increases with increases in Reynolds number. For higher Reynolds number a secondary circulation zones spotted. They are formed because of the sudden change in the direction of the flow. These structures are obtained for Re > 200.

The effect of local time stepping was studied and it was found out that local time stepping increases the convergence rate of the time marching algorithm almost twice than the constant time step method. Also, the effect of CFL number was studies and it was found out that increasing the cfl number increases the rate of convergence of the time marching algorithm. CFL and VN number directly related to local time stepping hence the effects mentioned for cfl are justified.

References:

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

2. Armaly, B., Durst, F., Pereira, J., & Schönung, B. (1983). Experimental and theoretical investigation of backward-facing step flow. Journal of Fluid Mechanics, 127, 473-496.

APPENDIX

Thomas Algorithm for solving tridiagonal matrix

```
function RHS = thomas(a,b,c,d)
N=max(size(b)); %To find the limits
for k=2:N
    m=a(k)/b(k-1);
    b(k) = b(k) - m*c(k-1);
    d(k) = d(k) - m*d(k-1);
end
d(N) = d(N)/b(N);
for k=N-1:-1:1
    d(k) = (d(k) - c(k)*d(k+1))/b(k);
end
RHS = d;
```

2D incompressible solver

```
tic;
clc
clear all
Re = 50;
cfl = 0.1;
VN = 0.1;
Nx = 400;
Ny = 40;
h=1;
Ly = 2*h;
Lx = 40*h;
r = 1.01;
Ub = 1;
epsi = 0.001;
dt = 0.001;

x(1) = 0;
y(1) = 0;
y(Ny) = 2;
sum = 0;

for i=1:Nx-1
    sum = sum + r^(i-1);
end

dx = Lx/sum;
for i=1:Nx
    x(i+1) = x(i) + dx*(r^(i-1));
end
```

```
sum = 0;
r=1;

for i = 1:Ny-1
    sum = sum + r^(i-1);
end
dy = (Ly)/(sum);
for i=1:(Ny)
    y(i+1)= y(i) + dy*(r^(i-1));
end
for i=1:Nx
    for j=1:Ny
        X(j,i) = x(i);
        Y(j,i) = y(j);
    end
end
clear x y
x = X; y = Y;

%Calculating Matrices of transformation
x_zeta = ones(Ny,Nx); y_eta = ones(Ny,Nx);
zeta_x = ones(Ny,Nx); eta_y = ones(Ny,Nx);
G = ones(Ny,Nx); J = ones(Ny,Nx);

for j=2:Ny-1
    for i=2:Nx-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(Ny,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,Nx) = 0.5*(y(j+1,Nx) - y(j-1,Nx));
    end
end
x_zeta(:,1) = x(:,2) - x(:,1);
x_zeta(:,Nx) = x(:,Nx) - x(:,Nx-1);
y_eta(1,:) = y(2,:) - y(1,:);
y_eta(Ny,:) = y(Ny,:) - y(Ny-1,:);

%Calculating Jacobian
for j=1:Ny
    for i=1:Nx
        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:Ny
    for i=1:Nx
        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:Ny-1
    for i=2:Nx-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:Ny
     for i=1:Nx
         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;
         u_new(j,i) = 0;
         v_new(j,i) = 0;
         vorticity(j,i) = 0;
     end
 end

 %Imposing boundary conditions
 for j=(Ny/2+1):Ny
     Q(j,1,2) = Ub*(1-(((y(j,1) - 1.5*h)/(0.5*h))^2));
     Q(Ny,1,2) = 0;
     ustar(j,1) = Ub*(1-(((y(j,1) - 1.5*h)/(0.5*h))^2));
     ustar(Ny,1) = 0;
     u_new(j,1) = Ub*(1-(((y(j,1) - 1.5*h)/(0.5*h))^2));
     u_new(Ny,1) = 0;
 end
 rho_1 = zeros(Ny,Nx); %similar to variable declaration
 rho_2 = zeros(Ny,Nx); %similar to variable declaration

 Diss_x = zeros(Ny,Nx);
```

```
 Diss_y = zeros(Ny,Nx);
 div = zeros(Ny,Nx);

 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:Ny
         for i=1:Nx
             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

            %Calculating RHS
            %Computing Contravarient velocities.
            for j=1:Ny
                for i=1:Nx
                    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:Ny-1
                for i=2:Nx-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:Ny-1
                for i=2:Nx-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:Ny-2
                    for i=3:Nx-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:Ny-2
                    for i=3:Nx-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:Ny
                for i=1:Nx
                    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:Ny-1
                    for i=2:Nx-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:Ny-1
                for i=1:Nx-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:Ny-1
                    for i=2:Nx-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:Ny-1
                for i=2:Nx-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:Ny-1
                    for i = 2:Nx-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
            %Residual Smoothing
            smo = 0.001;
            for j=2:Ny-1
                a(1:Nx-2) = -smo;
                b(1:Nx-2) = 1+2*smo;
                c(1:Nx-2) = -smo;
                a(1) = 0;
                c(Nx-2) = 0;
                for i=2:Nx-1
                    d1p(i-1) = RHS_compute(j,i,1);
                    d1u(i-1) = RHS_compute(j,i,2);
                    d1v(i-1) = RHS_compute(j,i,3);
                end
                phi_p(j,2:Nx-1) = thomas(a,b,c,d1p);
                phi_u(j,2:Nx-1) = thomas(a,b,c,d1u);
                phi_v(j,2:Nx-1) = thomas(a,b,c,d1v);
```

```
            end
            clear a b c d1p d1u d1v
            for i=2:Nx-1
                a(1:Ny-2) = -smo;
                b(1:Ny-2) = 1+2*smo;
                c(1:Ny-2) = -smo;
                a(1) = 0;
                c(Ny-2) = 0;

                for j=2:Ny-1
                    d1p(j-1) = phi_p(j,i);
                    d1u(j-1) = phi_u(j,i);
                    d1v(j-1) = phi_v(j,i);
                end
                RHS_compute(2:Ny-1,i,1) = thomas(a,b,c,d1p);
                RHS_compute(2:Ny-1,i,2) = thomas(a,b,c,d1u);
                RHS_compute(2:Ny-1,i,3) = thomas(a,b,c,d1v);
            end




            %Performing local timestepping
            rho_1 = zeros(Ny,Nx);
            rho_2 = zeros(Ny,Nx);
            for j = 1:Ny
                for i = 1:Nx
                    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);
                    dtau_cfl(j,i) = cfl/(J(j,i)*max(rho_1(j,i),rho_2(j,i)));
                    dtau_VN(j,i) = Re*VN/(max(g11(j,i),g22(j,i)));
                    dtau(j,i) = min(dtau_cfl(j,i),dtau_VN(j,i));
                end
            end


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

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

            %Updating boundary conditions for ustar
            for j=(Ny/2+1):Ny
                ustar(j,1) = Ub*(1-(((y(j,1) - 1.5*h)/(0.5*h))^2));
                ustar(j,Nx) = ustar(j,Nx-1);
            end

            ustar(:,Nx) = ustar(:,Nx-1);

            %Calculating Divergence of ustar
            for j=2:Ny-1
                for i=2:Nx-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(Ny,Nx);
            dp1 = zeros(Ny,Nx);

            while error_dp > epsilon_dp % Loop for convergence of dp

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

                %Solve Pressure Poisson
                for j=2:Ny-1
                    for i=2:Nx-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)/dtau(j,i)) - a)/b;

                        error_dp = error_dp + (abs((dp1(j,i)) - (dp(j,i))))^2;
%Calculating error for delta Pressure
```

```
                    error_dp = sqrt(error_dp)/(Ny*Nx);
                end
            end
        end

        %Calculating Gradient of delta pressure
        for j=2:Ny-1
            for i=2:Nx-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:Ny-1
            for i=2:Nx-1
                Pressure_new(j,i) = Q(j,i,1) + dp(j,i);
                u_new(j,i) = ustar(j,i) - dtau(j,i)*J(j,i)*grad_dp_x(j,i);
                v_new(j,i) = vstar(j,i) - dtau(j,i)*J(j,i)*grad_dp_y(j,i);
            end
        end

        Pressure_new(1,:) = Pressure_new(2,:);
        Pressure_new(Ny,:) = Pressure_new(Ny-1,:);
        Pressure_new(:,1) = Pressure_new(:,2);
        Pressure_new(:,Nx) = Pressure_new(:,Nx-1);

        u_new(:,Nx) = u_new(:,Nx-1);
        v_new(:,Nx) = v_new(:,Nx-1);

        %Calculating Error of corrections
        for j=2:Ny-1
            for i=2:Nx-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)/(Ny*Nx);
        error_u = sqrt(error_u)/(Ny*Nx);
```

```
            error_v = sqrt(error_v)/(Ny*Nx);

            for j=1:Ny
                for i=1:Nx
                    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(Ny,i,1) = Q(Ny-1,i,1);
                    Q(j,1,1) = Q(j,2,1);
                    Q(j,Nx,1) = Q(j,Nx-1,1);
                end
            end
        end

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

        for j=1:Ny
            for i=1:Nx
                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
    toc;
```