Assignment - 2

1. Solve the potential flow governing equation $\nabla^2\psi = 0$ around a circular cylinder

   (a) *Consider a circular cylinder of diameter d_1=1.*

   (b) *Take the outer boundary to be at a diameter of* $20 * d_1$

   (c) *Generate a grid around the cylinder; take uniform distribution in θ and in radial direction.*

   (d) *Solve the equation in cylindrical coordinates.*

   (e) *Since it is a stationary cylinder, the value of ψ on the cylinder is constant - no penetration boundary condition. Take this value to be $\psi_{cylinder} = constant = 20$*

   (f) *On the outer boundary, define $\psi = V_\infty y + constant(i.e.\ 20)$, where y is the y coordinate of the point on the boundary.*

   (g) *Since the domain is multiply connected, take a cut line downstream of the cylindear at $y = 0$ in the domain. This makes the domain 4 sided.*

   (h) *Apply periodic boundary condition on this boundary.*

   (a) Use the following methods to solve the equation

      i. Jacobi method

      ii. Gauss-Seidel method

      iii. Both the above with relaxation

   (b) Compare the convergence of the residue.

   (c) Plot and compare the velocity on the cylinder as obtained from the above 3 methods with the analytical solution.

Assignment no. 2

Potential Flow around a circular cylinder

Computational Methods for Compressible Flows
Date – 12/04/18

Vasu Bansode
SC15B009
Department of Aerospace engineering
Indian Institute of Space Science and Technology, Thiruvananthapuram-695547

Abstract –

Computational fluid dynamics provide efficient way to solve complex flow problems. Here, two dimensional potential flow over a rectangular cylinder of given dimensions is solved with stream function formulation. Various iterative methods such as Jacobi method, Gauss Seidel method, Jacobi method with relaxation, Gauss Seidel method with relaxation are used. Residue for each method is calculated and their relative efficiency is compared in terms of number of iterations required. Various graphs are obtained for each case. It's found that Gauss Seidel method with relaxation is the most efficient and fast compared to other methods.

Variables –

1. Diameter of cylinder = 1 m
2. Free stream velocity = 1 m/s

Boundary Conditions-

1. Physical domain is from 0.5 m to 10 m.
2. No penetration boundary condition on the cylinder surface i.e. $\psi = constant = 20$
3. On the outer boundary we have $\psi = V_{inf}\, y + 20$ where y is the co-ordinate of the point on the boundary,
4. Periodic boundary condition at $y = 0$ since the domain is multiply connected.

Problem Scheme-

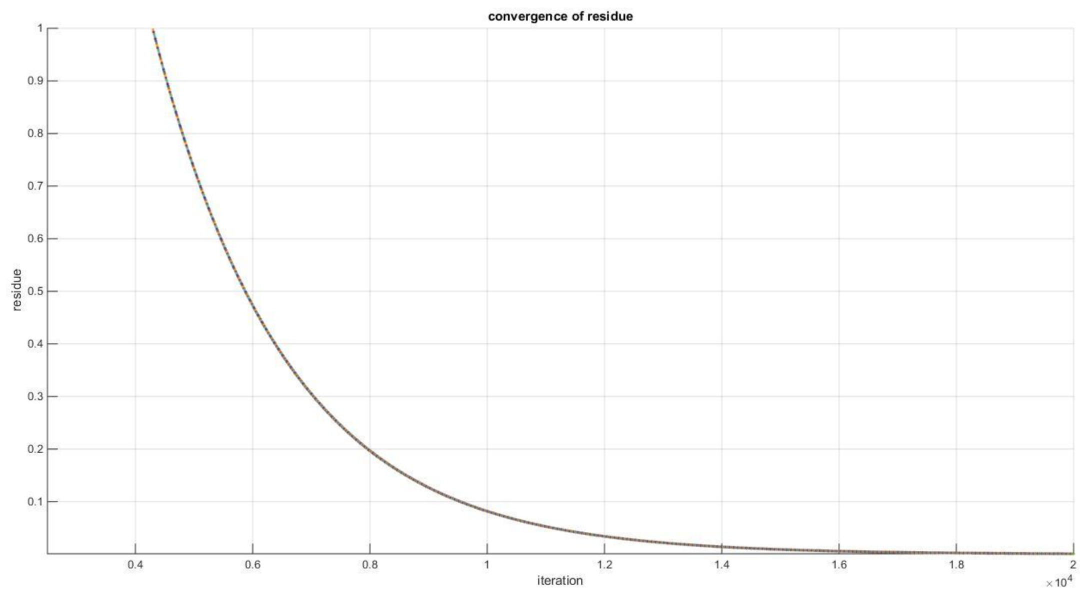
1. The problem is solved in the cylindrical co-ordinates.
2. Radial distance is discretised into 30 elements.
3. Angular position is varied from 0 degree to 360 degree with 30 elements.
4. Total residue is set as convergence criteria.

5. Convergence is achieved when total residue goes beyond 0.001.
6. For Jacobi method with relaxation, relaxation factor is equal to 0.9.
7. For Gauss Seidel method with relaxation, relaxation factor is equal to 1.8.

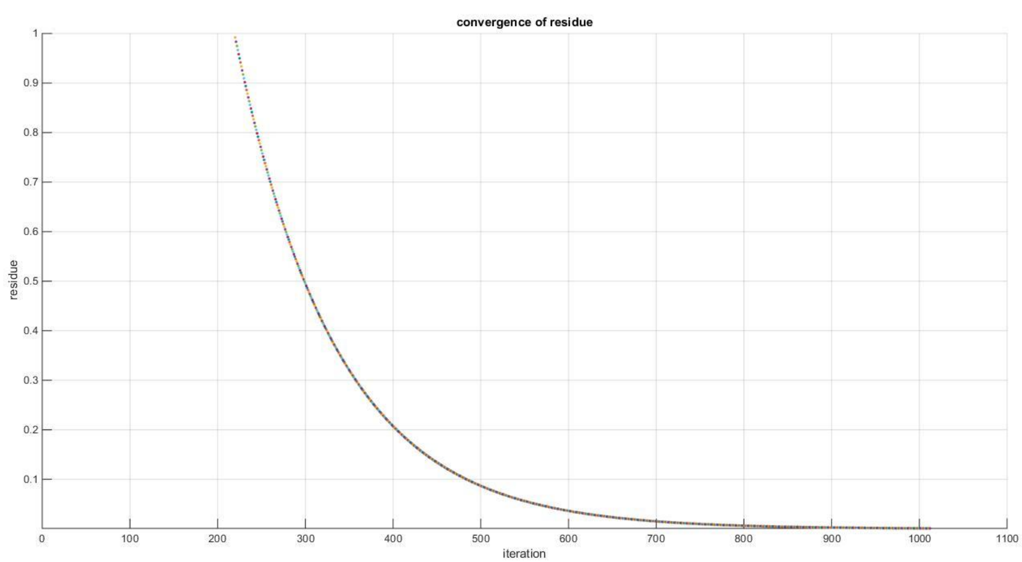
Results –

1. **Rate of Convergence**

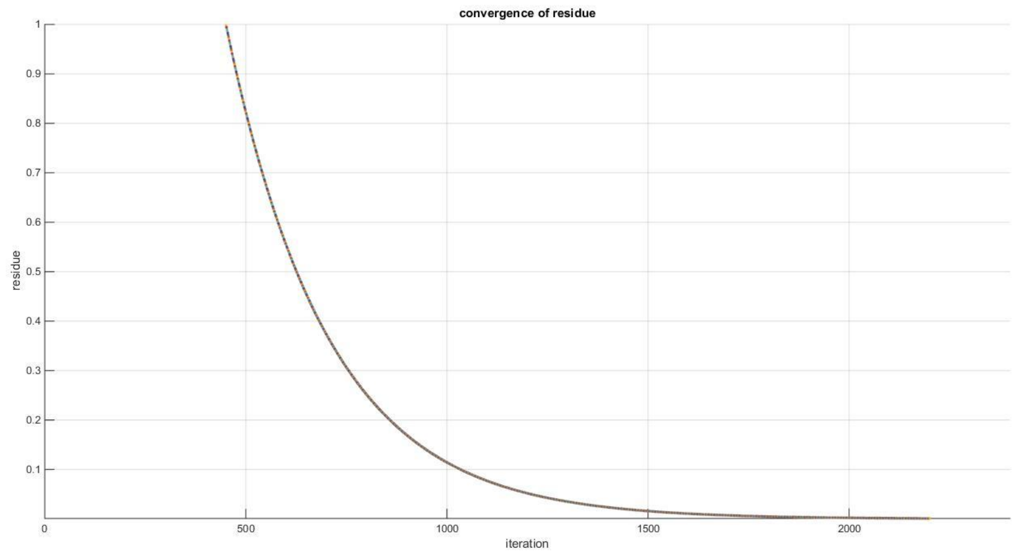
a) Jacobi Method –



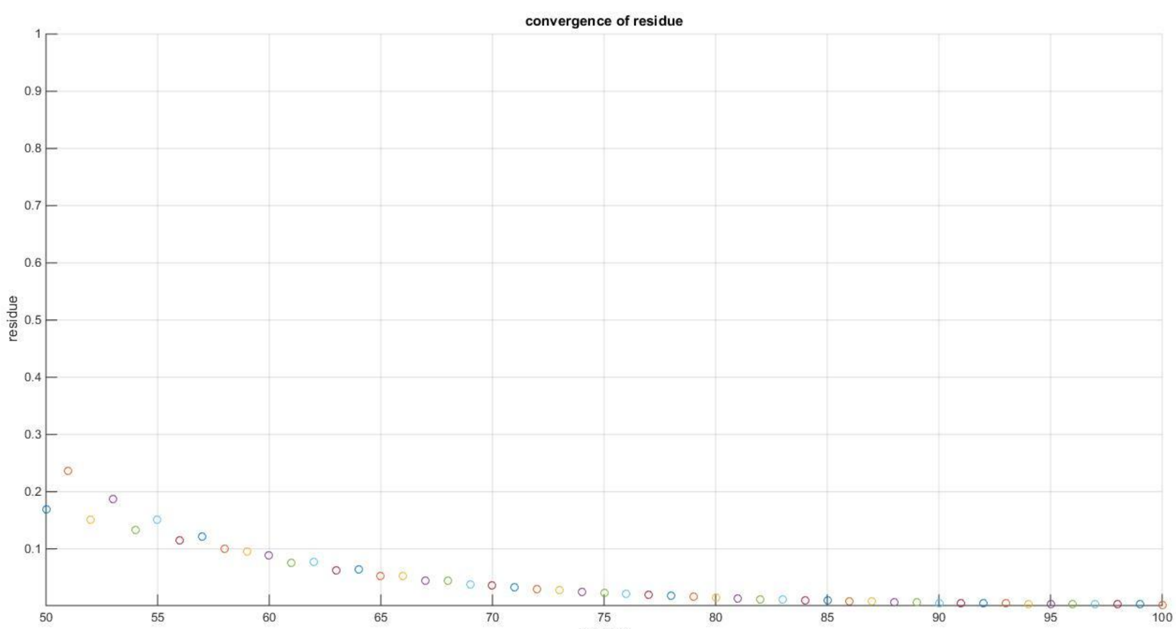
b) Gauss Seidel Method –



c) **Jacobi Method with relaxation –**

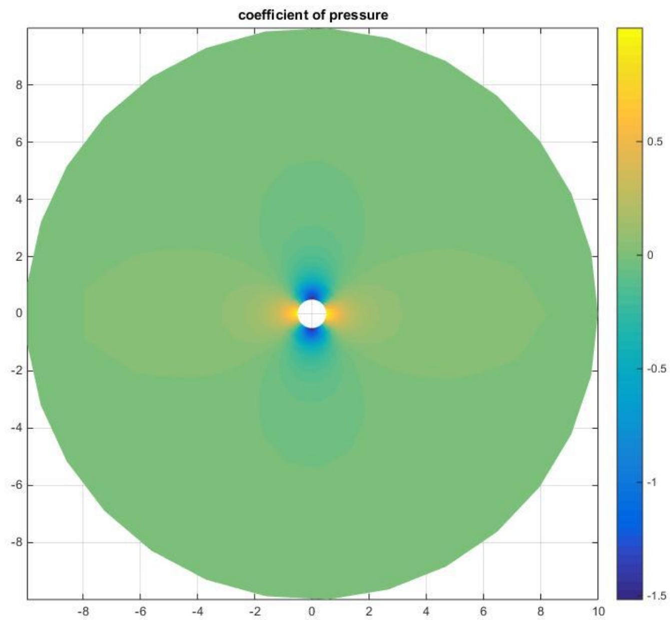


d) **Gauss Seidel with relaxation –**

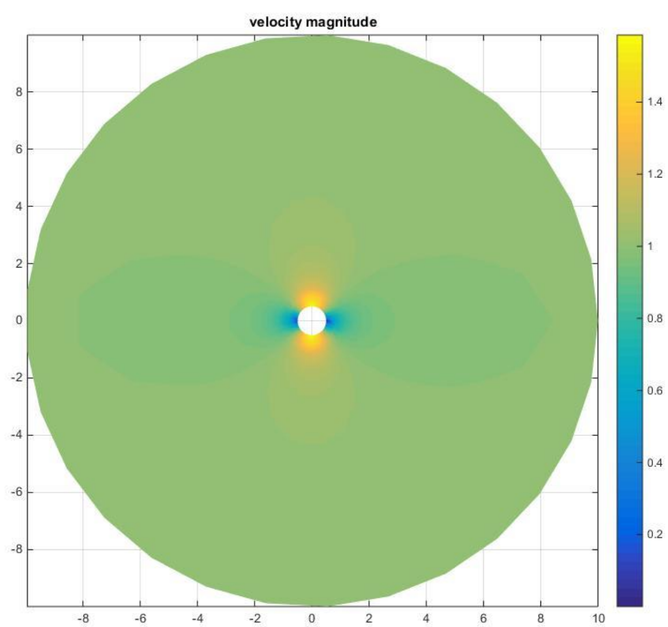


2. **Various plots –**

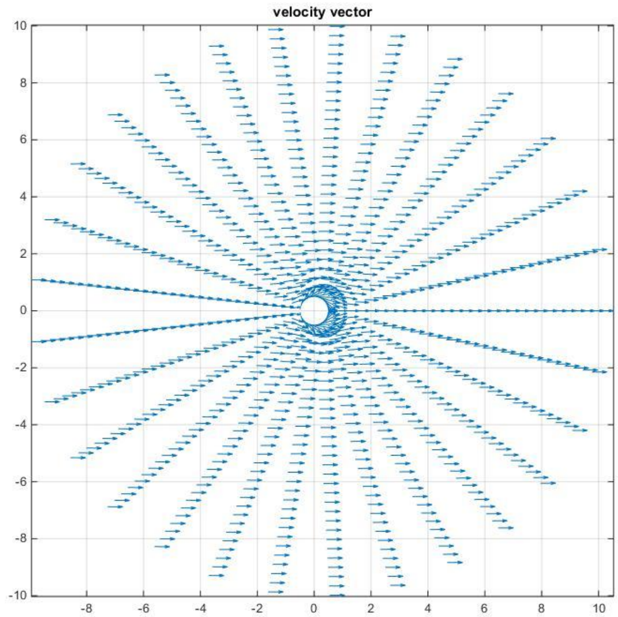
a) **Coefficient of pressure**



b) **Velocity magnitude –**

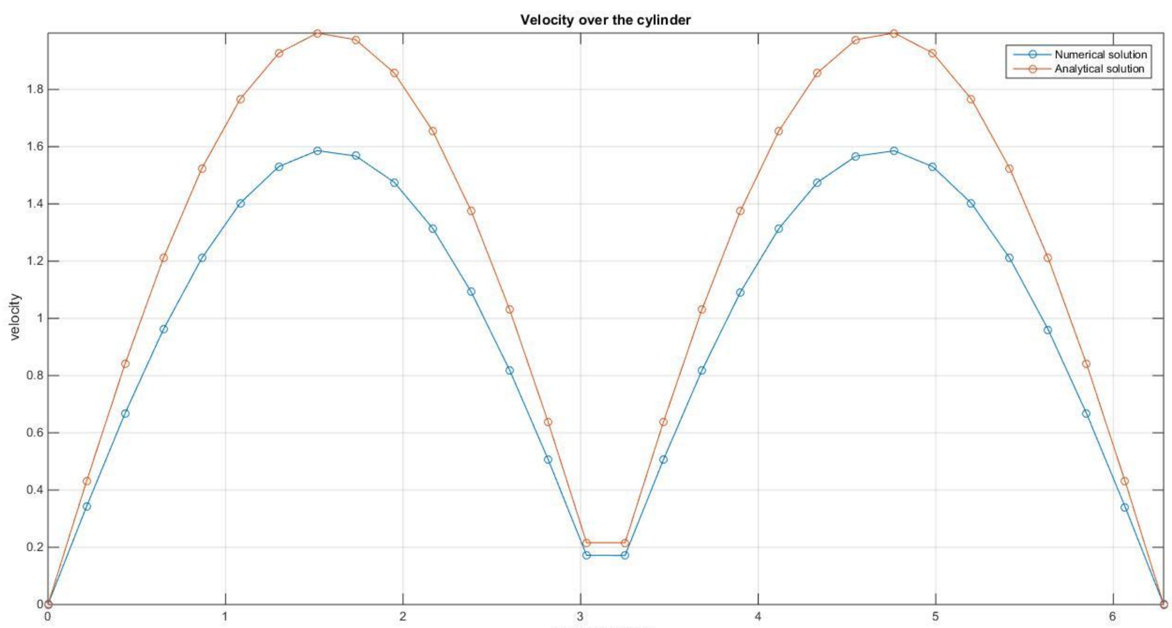


c) **Velocity Vector –**

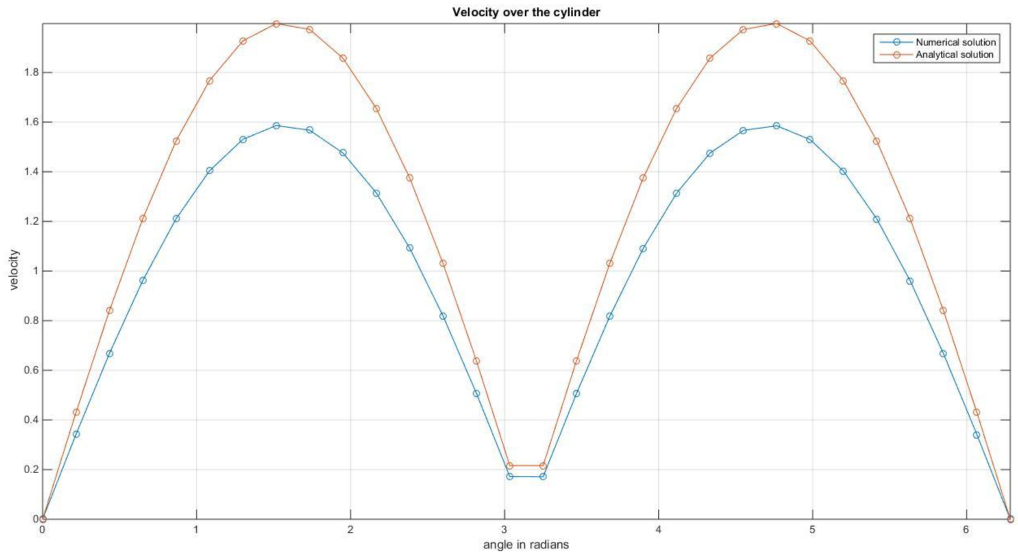


d) **Velocity magnitude over the cylinder –**

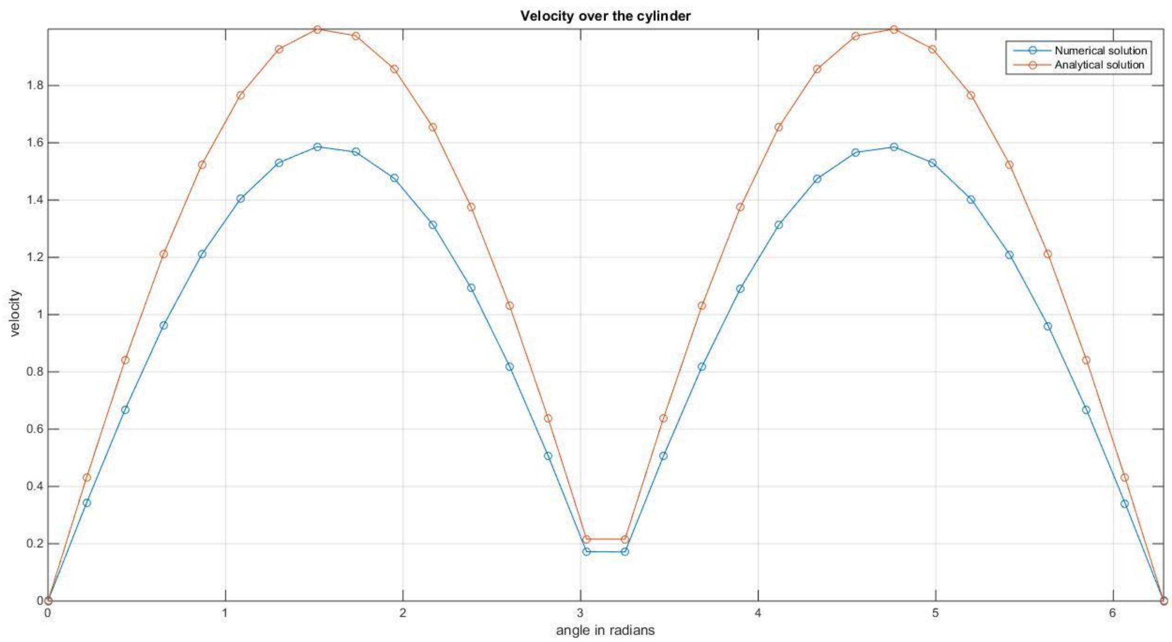
1. **Gauss Siedel with relaxation -**



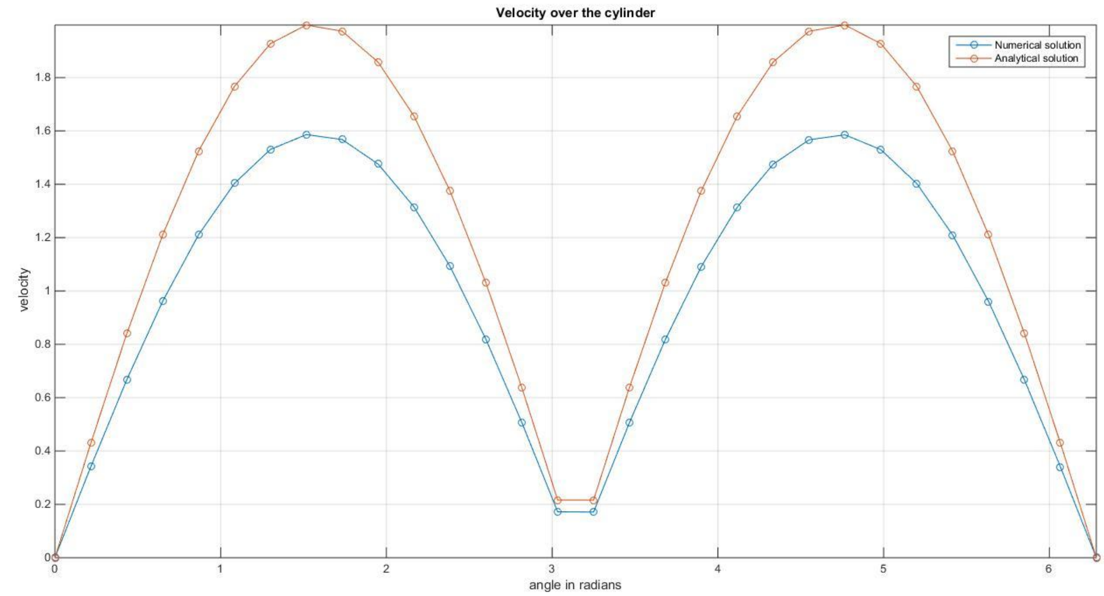
2. **Gauss Seidel method –**



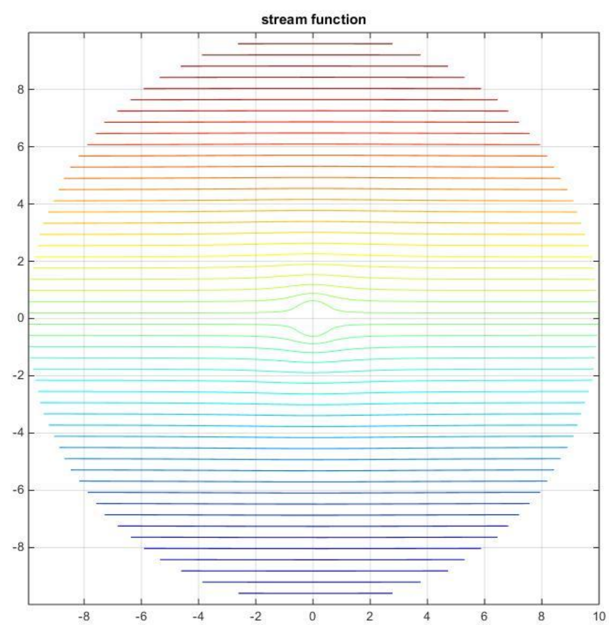
3. **Jacobi method with relaxation –**



4. **Jacobi method –**



e) **Stream function –**



Observation –

1. For Jacobi method residue converges after 20031 iterations.
2. For Gauss Seidel method residue converges after 1012 iterations.
3. For Jacobi method with relaxation residue converges after 2021 iterations.
4. For Gauss Seidel method with relaxation residue converges after 110 iterations.
5. From stream function plot and velocity graph it can be seen that magnitude of the velocity over the cylinder is symmetric with respect to the x – axis,

Discussion –

1. Convergence rate of residue largely depends upon the initial guess for the iterative method. Here, initial guess is made as follows -

$$\psi(i,j) \;=\; Vinf\; y\left(R(i) - \frac{R(n)^2}{R(i)}\right)\sin(\theta) + 20$$

2. The Gauss Seidel method uses values calculated at previous grid points for calculating values at the current grid point in the current iteration cycle. This is the reason for its high convergence rate.
3. The convergence rate for relaxation methods depends upon the relaxation factor, for given discretization there is an optimal value of relaxation factor for each method.
4. Under relaxation makes both the methods slow whereas effect is completely opposite for over relaxation.
5. Relaxation factor for Jacobi method is kept around 0.9 for maximum efficiency. Whereas for Gauss Seidel method it is kept around 1.8 for maximum efficiency.
6. The optimal value of relaxation factor for Jacobi method lies in the interval of 0.9 to 1 and for Gauss Seidel method it lies in the interval of 1.8 to 1.9.
7. The difference between the velocities calculated numerically and theoretically is maximum at 90 degree and 270 degree. But on the most of the surface they are in good agreement with sufficient accuracy.

Conclusion –

1. Rate of convergence for above mentioned methods follows the order –

Gauss Seidel method with relaxation > Gauss Seidel method > Jacobi method > Jacobi with relaxation

2. The difference in the velocities calculated theoretically and numerically can be minimized by using higher order schemes.
3. Other iterative methods better than above mentioned exists and give better results.

Acknowledgement –

I greatly thank you to Dr. Manoj Nair for giving us this assignment and helping us to get a deep insight into the subject. I also appreciate the help provided by Mr. Arun Govind Neelan throughout the work.

Appendix –

a) MATLAB code for Jacobi Method

```
clear all; clc;close all

%% Defining constants

Vinf = 1;

d1 = 1;

din = d1/2;

d2 = 20*d1;

dout = d2/2;
%%

m = 30;                                             % length of theta

n = 30;                                             % length of R

theta=linspace(0,2*pi,m);                            % variation in theta

R = linspace(din,dout,n);                            % variation in R

dR = R(4) - R(3);

dt = theta(4) - theta(3);
%% Calculating x and y co-ordinates and intializing xi = stream function

for j = 1:m

    for i = 1:n

        x(i,j) = R(i)*cos(theta(j));

        y(i,j) = R(i)*sin(theta(j));

        xi(i,j) = 22 + Vinf*(R(i) - R(n)^2/R(i))*sin(theta(j));

    end

end

 xi(1,:) = 20;

   for j = 1:m
```

```
        xi(n,j) = Vinf*y(n,j) + 20;

    end

    %% Main body of Jacobi iteration

XI = xi;

total_residue  = 10;

k = 0 ;

tic

while abs(total_residue)>10^(-3)

    % calculations for interior points

for j = 2:m - 1

    for i = 2:n - 1

        a(i,j) = (R(i+1) + R(i))/(2*dR^2);

        b(i,j) = (R(i-1) + R(i))/(2*dR^2);

        c(i,j) = 1/(R(i)*dt^2);

        d(i,j) = 1/(R(i)*dt^2);

        e(i,j) = (2*R(i) + R(i+1) + R(i-1))/(2*dR^2) + 2/(R(i)*dt^2);

        XI(i,j) = (a(i,j)*xi(i+1,j) + b(i,j)*xi(i-1,j) +...
            c(i,j)*xi(i,j+1) + d(i,j)*xi(i,j-1))/e(i,j);

    end

end

% calculations at periodic boundary

for i = 2:n-1

         a(i,1) = (R(i+1) + R(i))/(2*dR^2);

        b(i,1) = (R(i-1) + R(i))/(2*dR^2);

        c(i,1) = 1/(R(i)*dt^2);

        d(i,1) = 1/(R(i)*dt^2);

        e(i,1) = (2*R(i) + R(i+1) + R(i-1))/(2*dR^2) + 2/(R(i)*dt^2);

        XI(i,1) = (a(i,1)*xi(i+1,1) + b(i,1)*xi(i-1,1) +...
            c(i,1)*xi(i,1+1) + d(i,1)*xi(i,m-1))/e(i,1);
end

XI(:,m) = XI(:,1);
```

```
% residue calculation

total_residue = 0;

for j = 2:m - 1

    for i = 2:n - 1

        residue(i,j) = XI(i,j) - ((a(i,j)*XI(i+1,j) + b(i,j)*XI(i-1,j) +...
            c(i,j)*XI(i,j+1) + d(i,j)*XI(i,j-1))/e(i,j));

    end

end

for i = 2:n - 1

        residue(i,1) = XI(i,1) - ((a(i,1)*XI(i+1,1) + b(i,1)*XI(i-1,1) +...
            c(i,1)*XI(i,1+1) + d(i,1)*XI(i,m-1))/e(i,1));

end

residue(:,m) = residue(:,1);

for j = 1:m

    for i = 2:n-1

     total_residue = total_residue + residue(i,j);

    end

end

toc

k = k+1;

if mod(k,10)==0

    hold on

 plot(k,total_residue,'.');

 xlabel('iteration')

 ylabel('residue')

 axis tight

 axis([2500 20000 10^(-3) 1])

 title('convergence of residue')

 grid on
```

```
 drawnow;

 xi = XI;

end

end

%% Analytical solution for velocity

for j = 1:m

    for i = 1:n

        V_r(i,j) = Vinf*(1 - (din/R(i))^2)*cos(theta(j));  % velocity in R direction

        V_t(i,j) = Vinf*(1 + (din/R(i))^2)*sin(theta(j));  % velocity in theta direction

        V_x(i,j) = -sin(theta(j))*V_t(i,j) + cos(theta(j))*V_r(i,j); % velocity in X
direction

        V_y(i,j) = sin(theta(j))*V_r(i,j) + cos(theta(j))*V_t(i,j);  % velocity in y
direction

        V(i,j) =  sqrt(V_x(i,j)^2 + V_y(i,j)^2); % total velocity

        CpA(i,j)  = 1-(V(i,j)/Vinf)^2; % coefficient of pressure.

    end

end

%% Extracting radial component and tangential component of velocity

for j = 2:m-1

    for i = 1:n

        U_r(i,j) = (1/R(i))*((XI(i,j+1) - XI(i,j-1))/(2*dt));

    end
end

for i = 1:n

    U_r(i,1) = (1/R(i))*((XI(i,2) - XI(i,1))/(dt));

    U_r(i,m) = (1/R(i))*((XI(i,m) - XI(i,m-1))/(dt));

end

for j = 1:m

    for i = 2:n-1

U_t(i,j) = -(XI(i+1,j) - XI(i-1,j))/(2*dR);

    end
```

```
end

for j = 1:m

    U_t(1,j) = -(XI(2,j) - XI(1,j))/(dR);

    U_t(n,j) = -(XI(n,j) - XI(n-1,j))/(dR);

end

%% Extracting x and y direction components

for j = 1:m

    for i = 1:n


        U_x(i,j) = -sin(theta(j))*U_t(i,j) + cos(theta(j))*U_r(i,j);

        U_y(i,j) = sin(theta(j))*U_r(i,j) + cos(theta(j))*U_t(i,j);

        U(i,j) =  sqrt(U_x(i,j)^2 + U_y(i,j)^2);

        Cp(i,j)  = 1-(U(i,j)/Vinf)^2;

    end

end

figure;

pcolor(x,y,Cp);

colorbar

shading interp;

axis equal

title('coefficient of pressure')

axis tight

grid on


figure;

quiver(x,y,U_x,U_y);

axis equal

title('velocity vector')

axis tight
```

```
grid on

figure;
plot(theta,U(1,:),'-o')
hold on
plot(theta,V(1,:),'-o')
hold off
axis tight
grid on
legend('Numerical solution','Analytical solution')
title('Velocity over the cylinder')
ylabel('velocity')
xlabel('angle in radians')

figure;
contour(x,y,XI,50);
shading interp;
colormap jet
axis equal
axis equal
title('stream function')
axis tight
grid on

figure;
pcolor(x,y,U);
colorbar
shading interp;
axis equal
title('Velocity magnitude')
```

```
axis tight

grid on
```

b) MATLAB code for Gauss Seidel Method

```
clear all; clc;close all
%% Defining constants an initial data

Vinf = 1;

d1 = 1;

din = d1/2;

d2 = 20*d1;

dout = d2/2;

%%

m = 30;                                              % length of theta

n = 30;                                              % length of R

theta=linspace(0,2*pi,m);                             % variation in theta

R = linspace(din,dout,n);                             % variation in R direction

dR = R(4) - R(3);

dt = theta(4) - theta(3);

xi = zeros(n,m);

%% calculating x and y co-ordinates and initializing xi

for j = 1:m

    for i = 1:n

        x(i,j) = R(i)*cos(theta(j));

        y(i,j) = R(i)*sin(theta(j));

        xi(i,j) = 22 + Vinf*(R(i) - R(n)^2/R(i))*sin(theta(j));

    end

end

 xi(1,:) = 20;

   for j = 1:m
```

```
        xi(n,j) = Vinf*y(n,j) + 20;

    end

    %% Main program- gauss siedel

XI = xi;

total_residue  = 10;

k = 0;

tic

while abs(total_residue)>10^(-3)

% for interior points

for j = 2:m - 1

        for i = 2:n - 1

        a(i,j) = (R(i+1) + R(i))/(2*dR^2);

        b(i,j) = (R(i-1) + R(i))/(2*dR^2);

        c(i,j) = 1/(R(i)*dt^2);

        d(i,j) = 1/(R(i)*dt^2);

        e(i,j) = (2*R(i) + R(i+1) + R(i-1))/(2*dR^2) + 2/(R(i)*dt^2);

        XI(i,j) = (a(i,j)*xi(i+1,j) + b(i,j)*XI(i-1,j) +...
            c(i,j)*xi(i,j+1) + d(i,j)*XI(i,j-1))/e(i,j);

        end


end

% at periodic boundary

for i = 2:n-1

         a(i,1) = (R(i+1) + R(i))/(2*dR^2);

        b(i,1) = (R(i-1) + R(i))/(2*dR^2);

        c(i,1) = 1/(R(i)*dt^2);

        d(i,1) = 1/(R(i)*dt^2);

        e(i,1) = (2*R(i) + R(i+1) + R(i-1))/(2*dR^2) + 2/(R(i)*dt^2);

        XI(i,1) = (a(i,1)*xi(i+1,1) + b(i,1)*XI(i-1,1) +...
            c(i,1)*xi(i,1+1) + d(i,1)*XI(i,m-1))/e(i,1);
```

```
end

XI(:,m) = XI(:,1);

xi = XI;

% residue calculation

total_residue = 0;


for j = 2:m - 1

    for i = 2:n - 1

        residue(i,j) = XI(i,j) - ((a(i,j)*XI(i+1,j) + b(i,j)*XI(i-1,j) +...
            c(i,j)*XI(i,j+1) + d(i,j)*XI(i,j-1))/e(i,j));

    end

end


for i = 2:n - 1

        residue(i,1) = XI(i,1) - ((a(i,1)*XI(i+1,1) + b(i,1)*XI(i-1,1) +...
            c(i,1)*XI(i,1+1) + d(i,1)*XI(i,m-1))/e(i,1));
end

residue(:,m) = residue(:,1);


for j = 1:m

    for i = 2:n-1

     total_residue = total_residue + residue(i,j);

    end

end

toc

k = k+1;

if mod(k,1)==0

    hold on

 plot(k,total_residue,'.');

 xlabel('iteration')

 ylabel('residue')
```

```
 axis tight

 axis([0 1100 10^(-3) 1])

 title('convergence of residue')

 grid on

 drawnow;

end


 total_residue

end
%% velocity from analytical solution

for j = 1:m

    for i = 1:n

        V_r(i,j) = Vinf*(1 - (din/R(i))^2)*cos(theta(j));  % velocity in R direction

        V_t(i,j) = Vinf*(1 + (din/R(i))^2)*sin(theta(j));  % velocity in theta direction

        V_x(i,j) = -sin(theta(j))*V_t(i,j) + cos(theta(j))*V_r(i,j); % velocity in X
direction

        V_y(i,j) = sin(theta(j))*V_r(i,j) + cos(theta(j))*V_t(i,j);  % velocity in y
direction

        V(i,j) =  sqrt(V_x(i,j)^2 + V_y(i,j)^2); % total velocity

        CpA(i,j)  = 1-(V(i,j)/Vinf)^2; % coefficient of pressure.


    end

end

%% extracting velocity from stream function

for j = 2:m-1

    for i = 1:n

        U_r(i,j) = (1/R(i))*((XI(i,j+1) - XI(i,j-1))/(2*dt));

    end

end


for i = 1:n

    U_r(i,1) = (1/R(i))*((XI(i,2) - XI(i,1))/(dt));
```

```
    U_r(i,m) = (1/R(i))*((XI(i,m) - XI(i,m-1))/(dt));

end


for j = 1:m

    for i = 2:n-1

U_t(i,j) = -(XI(i+1,j) - XI(i-1,j))/(2*dR);

    end

end

for j = 1:m

    U_t(1,j) = -(XI(2,j) - XI(1,j))/(dR);

    U_t(n,j) = -(XI(n,j) - XI(n-1,j))/(dR);

end

%% Extracting x and y direction velocity components

for j = 1:m

    for i = 1:n


        U_x(i,j) = -sin(theta(j))*U_t(i,j) + cos(theta(j))*U_r(i,j);

        U_y(i,j) = sin(theta(j))*U_r(i,j) + cos(theta(j))*U_t(i,j);

        U(i,j) =  sqrt(U_x(i,j)^2 + U_y(i,j)^2);

        Cp(i,j)  = 1-(U(i,j)/Vinf)^2;

    end

end

figure;

pcolor(x,y,Cp);

colorbar

shading interp;

axis equal

title('coefficient of pressure')

axis tight
```

```
grid on

figure;
quiver(x,y,U_x,U_y);
axis equal
title('velocity vector')
axis tight
grid on

figure;
plot(theta,U(1,:),'-o')
hold on
plot(theta,V(1,:),'-o')
hold off
axis tight
grid on
legend('Numerical solution','Analytical solution')
title('Velocity over the cylinder')
ylabel('velocity')
xlabel('angle in radians')

figure;
contour(x,y,XI,50);
shading interp;
colormap jet
axis equal
title('stream function')
axis tight
grid on

figure;
```

```
pcolor(x,y,U);

colorbar

shading interp;

axis equal

title('coefficient of pressure')

axis tight

grid on
```

c) MATLAB code for Jacobi method with relaxation –

```
clear all; clc;close all

%% Defining constants

Vinf = 1;

d1 = 1;

din = d1/2;

d2 = 20*d1;

dout = d2/2;
%%

m = 30;                                          % length of theta

n = 30;                                          % length of R

theta=linspace(0,2*pi,m);                        % variation in theta

R = linspace(din,dout,n);                        % variation in R

dR = R(4) - R(3);

dt = theta(4) - theta(3);

w = 0.9;                                         % relaxation factor

%% Getting x and y co-ordinates and initializing xi = streamline function

k = 0;

for j = 1:m

    for i = 1:n
```

```
        x(i,j) = R(i)*cos(theta(j));

        y(i,j) = R(i)*sin(theta(j));

        xi(i,j) = 22 + Vinf*(R(i) - R(n)^2/R(i))*sin(theta(j));

    end

end

 xi(1,:) = 20;

   for j = 1:m

       xi(n,j) = Vinf*y(n,j) + 20;

   end

   %% Main body - Jacobi iteration with relaxation

XI = xi;

total_residue  = 10;

tic;

while abs(total_residue)>10^(-3)

    % for interior points

for j = 2:m - 1

    for i = 2:n - 1

        a(i,j) = (R(i+1) + R(i))/(2*dR^2);

        b(i,j) = (R(i-1) + R(i))/(2*dR^2);

        c(i,j) = 1/(R(i)*dt^2);

        d(i,j) = 1/(R(i)*dt^2);

        e(i,j) = (2*R(i) + R(i+1) + R(i-1))/(2*dR^2) + 2/(R(i)*dt^2);

        XI(i,j) = (1-w)*xi(i,j) + w*((a(i,j)*xi(i+1,j) + b(i,j)*xi(i-1,j) +...
            c(i,j)*xi(i,j+1) + d(i,j)*xi(i,j-1))/e(i,j));

    end

end

% at periodic boundary

for i = 2:n-1

         a(i,1) = (R(i+1) + R(i))/(2*dR^2);
```

```
        b(i,1) = (R(i-1) + R(i))/(2*dR^2);

        c(i,1) = 1/(R(i)*dt^2);

        d(i,1) = 1/(R(i)*dt^2);

        e(i,1) = (2*R(i) + R(i+1) + R(i-1))/(2*dR^2) + 2/(R(i)*dt^2);

        XI(i,1) = (1-w)*xi(i,1)+ w*((a(i,1)*xi(i+1,1) + b(i,1)*xi(i-1,1) +...
            c(i,1)*xi(i,1+1) + d(i,1)*xi(i,m-1))/e(i,1));

end

XI(:,m) = XI(:,1);

% calculating the residue

total_residue = 0;


for j = 2:m - 1

    for i = 2:n - 1


        residue(i,j) = XI(i,j) - ((1-w)*XI(i,j)+w*(((a(i,j)*XI(i+1,j) + b(i,j)*XI(i-1,j)
+...
            c(i,j)*XI(i,j+1) + d(i,j)*XI(i,j-1))/e(i,j))));

    end

end

for i = 2:n - 1

        residue(i,1) = XI(i,1) - ((1-w)*XI(i,1)+ w*(((a(i,1)*XI(i+1,1) + b(i,1)*XI(i-1,1)
+...
            c(i,1)*XI(i,1+1) + d(i,1)*XI(i,m-1))/e(i,1))));

end

residue(:,m) = residue(:,1);

for j = 1:m

    for i = 2:n-1

     total_residue = total_residue + residue(i,j);

    end

end

toc;

k = k+1;
```

```
if mod(k,1)==0

    hold on

 plot(k,total_residue,'.');

 xlabel('iteration')

 ylabel('residue')

 axis tight

 axis([0 2400 10^(-3) 1])

 title('convergence of residue')

 grid on

 drawnow;

 total_residue

end

 xi = XI;

end
%% velocity from analytical solution

for j = 1:m

    for i = 1:n

       V_r(i,j) = Vinf*(1 - (din/R(i))^2)*cos(theta(j));  % velocity in R direction

        V_t(i,j) = Vinf*(1 + (din/R(i))^2)*sin(theta(j));  % velocity in theta direction

        V_x(i,j) = -sin(theta(j))*V_t(i,j) + cos(theta(j))*V_r(i,j); % velocity in X
direction

        V_y(i,j) = sin(theta(j))*V_r(i,j) + cos(theta(j))*V_t(i,j);  % velocity in y
direction

        V(i,j) =  sqrt(V_x(i,j)^2 + V_y(i,j)^2); % total velocity

        CpA(i,j)  = 1-(V(i,j)/Vinf)^2; % coefficient of pressure.

    end

end

%% Extracting the radial and tangential velocity from the stream function

for j = 2:m-1

    for i = 1:n
```

```
        U_r(i,j) = (1/R(i))*((XI(i,j+1) - XI(i,j-1))/(2*dt));

    end
end

for i = 1:n

    U_r(i,1) = (1/R(i))*((XI(i,2) - XI(i,1))/(dt));

     U_r(i,m) = (1/R(i))*((XI(i,m) - XI(i,m-1))/(dt));

end


for j = 1:m

    for i = 2:n-1

U_t(i,j) = -(XI(i+1,j) - XI(i-1,j))/(2*dR);

    end

end


for j = 1:m

    U_t(1,j) = -(XI(2,j) - XI(1,j))/(dR);

    U_t(n,j) = -(XI(n,j) - XI(n-1,j))/(dR);

end

%% Extracting x and y direction velocity

for j = 1:m

    for i = 1:n


        U_x(i,j) = -sin(theta(j))*U_t(i,j) + cos(theta(j))*U_r(i,j);

        U_y(i,j) = sin(theta(j))*U_r(i,j) + cos(theta(j))*U_t(i,j);

        U(i,j) =  sqrt(U_x(i,j)^2 + U_y(i,j)^2);

        Cp(i,j)  = 1-(U(i,j)/Vinf)^2;

    end

end

figure;

pcolor(x,y,Cp);
```

```
colorbar
shading interp;
axis equal
title('coefficient of pressure')
axis tight
grid on

figure;
quiver(x,y,U_x,U_y);
axis equal
title('velocity vector')
axis tight
grid on

figure;
plot(theta,U(1,:),'-o')
hold on
plot(theta,V(1,:),'-o')
hold off
axis tight
grid on
legend('Numerical solution','Analytical solution')
title('Velocity over the cylinder')
ylabel('velocity')
xlabel('angle in radians')

figure;
contour(x,y,XI,50);
shading interp;
colormap jet
```

```
axis equal

axis equal

title('stream function')

axis tight

grid on


figure;

pcolor(x,y,U);

colorbar

shading interp;

axis equal

title('coefficient of pressure')

axis tight

grid on
```

d) MATLAB Code for Gauss Seidel method with relaxation –

```
clear all; clc;close all

%% Defining constants

Vinf = 1;

d1 = 1;

din = d1/2;

d2 = 20*d1;

dout = d2/2;
%%

m = 30;                                             % length of theta

n = 30;                                             % length of R

theta=linspace(0,2*pi,m);                           % variation in theta

R = linspace(din,dout,n);                           % variation in R

dR = R(4) - R(3);

dt = theta(4) - theta(3);
```

```
w = 0.9;                                        % relaxation factor

%% Getting x and y co-ordinates and initializing xi = streamline function

k = 0;

for j = 1:m

    for i = 1:n


        x(i,j) = R(i)*cos(theta(j));

        y(i,j) = R(i)*sin(theta(j));

        xi(i,j) = 22 + Vinf*(R(i) - R(n)^2/R(i))*sin(theta(j));

    end

end

 xi(1,:) = 20;

   for j = 1:m

       xi(n,j) = Vinf*y(n,j) + 20;

   end

   %% Main body - Jacobi iteration with relaxation

XI = xi;

total_residue  = 10;

tic;

while abs(total_residue)>10^(-3)

    % for interior points

for j = 2:m - 1

    for i = 2:n - 1

        a(i,j) = (R(i+1) + R(i))/(2*dR^2);

        b(i,j) = (R(i-1) + R(i))/(2*dR^2);

        c(i,j) = 1/(R(i)*dt^2);

        d(i,j) = 1/(R(i)*dt^2);

        e(i,j) = (2*R(i) + R(i+1) + R(i-1))/(2*dR^2) + 2/(R(i)*dt^2);

        XI(i,j) = (1-w)*xi(i,j) + w*((a(i,j)*xi(i+1,j) + b(i,j)*xi(i-1,j) +...
```

```
            c(i,j)*xi(i,j+1) + d(i,j)*xi(i,j-1))/e(i,j));

    end

end

% at periodic boundary

for i = 2:n-1

         a(i,1) = (R(i+1) + R(i))/(2*dR^2);

        b(i,1) = (R(i-1) + R(i))/(2*dR^2);

        c(i,1) = 1/(R(i)*dt^2);

        d(i,1) = 1/(R(i)*dt^2);

        e(i,1) = (2*R(i) + R(i+1) + R(i-1))/(2*dR^2) + 2/(R(i)*dt^2);

        XI(i,1) = (1-w)*xi(i,1)+ w*((a(i,1)*xi(i+1,1) + b(i,1)*xi(i-1,1) +...
            c(i,1)*xi(i,1+1) + d(i,1)*xi(i,m-1))/e(i,1));

end

XI(:,m) = XI(:,1);

% calculating the residue

total_residue = 0;


for j = 2:m - 1

    for i = 2:n - 1


        residue(i,j) = XI(i,j) - ((1-w)*XI(i,j)+w*(((a(i,j)*XI(i+1,j) + b(i,j)*XI(i-1,j)
+...
            c(i,j)*XI(i,j+1) + d(i,j)*XI(i,j-1))/e(i,j))));

    end

end

for i = 2:n - 1

        residue(i,1) = XI(i,1) - ((1-w)*XI(i,1)+ w*(((a(i,1)*XI(i+1,1) + b(i,1)*XI(i-1,1)
+...
            c(i,1)*XI(i,1+1) + d(i,1)*XI(i,m-1))/e(i,1))));

end

residue(:,m) = residue(:,1);

for j = 1:m

    for i = 2:n-1
```

```
     total_residue = total_residue + residue(i,j);

    end

end

toc;

k = k+1;

if mod(k,1)==0

    hold on

 plot(k,total_residue,'.');

 xlabel('iteration')

 ylabel('residue')

 axis tight

 axis([0 2400 10^(-3) 1])

 title('convergence of residue')

 grid on

 drawnow;

 total_residue

end

 xi = XI;

end
%% velocity from analytical solution

for j = 1:m

    for i = 1:n

       V_r(i,j) = Vinf*(1 - (din/R(i))^2)*cos(theta(j));  % velocity in R direction

        V_t(i,j) = Vinf*(1 + (din/R(i))^2)*sin(theta(j));  % velocity in theta direction

        V_x(i,j) = -sin(theta(j))*V_t(i,j) + cos(theta(j))*V_r(i,j); % velocity in X
direction

        V_y(i,j) = sin(theta(j))*V_r(i,j) + cos(theta(j))*V_t(i,j);  % velocity in y
direction

        V(i,j) =  sqrt(V_x(i,j)^2 + V_y(i,j)^2); % total velocity

        CpA(i,j)  = 1-(V(i,j)/Vinf)^2; % coefficient of pressure.
```

```
    end

end

%% Extracting the radial and tangential velocity from the stream function

for j = 2:m-1

    for i = 1:n

        U_r(i,j) = (1/R(i))*((XI(i,j+1) - XI(i,j-1))/(2*dt));

    end
end

for i = 1:n

    U_r(i,1) = (1/R(i))*((XI(i,2) - XI(i,1))/(dt));

     U_r(i,m) = (1/R(i))*((XI(i,m) - XI(i,m-1))/(dt));

end


for j = 1:m

    for i = 2:n-1

U_t(i,j) = -(XI(i+1,j) - XI(i-1,j))/(2*dR);

    end

end


for j = 1:m

    U_t(1,j) = -(XI(2,j) - XI(1,j))/(dR);

    U_t(n,j) = -(XI(n,j) - XI(n-1,j))/(dR);

end

%% Extracting x and y direction velocity

for j = 1:m

    for i = 1:n


        U_x(i,j) = -sin(theta(j))*U_t(i,j) + cos(theta(j))*U_r(i,j);

        U_y(i,j) = sin(theta(j))*U_r(i,j) + cos(theta(j))*U_t(i,j);

        U(i,j) =  sqrt(U_x(i,j)^2 + U_y(i,j)^2);
```

```
        Cp(i,j)  = 1-(U(i,j)/Vinf)^2;

    end

end

figure;

pcolor(x,y,Cp);

colorbar

shading interp;

axis equal

title('coefficient of pressure')

axis tight

grid on


figure;

quiver(x,y,U_x,U_y);

axis equal

title('velocity vector')

axis tight

grid on


figure;

plot(theta,U(1,:),'-o')

hold on

plot(theta,V(1,:),'-o')

hold off

axis tight

grid on

legend('Numerical solution','Analytical solution')

title('Velocity over the cylinder')

ylabel('velocity')

xlabel('angle in radians')
```

```
figure;
contour(x,y,XI,50);
shading interp;
colormap jet
axis equal
axis equal
title('stream function')
axis tight
grid on

figure;
pcolor(x,y,U);
colorbar
shading interp;
axis equal
title('coefficient of pressure')
axis tight
grid on
```