ME630A
Assignment-2

Name: Rikesh Sharma
Roll: 180606

Name : Rikesh Sharma
Roll no : 180606

ME630

Assignment - 2

Given Poisson equation

as $\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = S\phi$ for $x \in [0,1]$, $y \in [0,1]$

where $S\phi = 2\sinh[10(x-\frac{1}{2})] + 40(x-\frac{1}{2})\cosh[10(x-\frac{1}{2})]$
$+ 100(x-\frac{1}{2})^2 \sinh[10(x-\frac{1}{2})]$
$+ 2\sinh[10(y-\frac{1}{2})] + 40(y-\frac{1}{2})\cosh[10(y-\frac{1}{2})]$
$+ 100(y-\frac{1}{2})^2 \sinh[10(y-\frac{1}{2})]$
$+ 4(x^2+y^2)e^{2xy}$

Given boundary conditions as

$\phi(0,y) = \frac{1}{4}\sinh(-5) + (y-\frac{1}{2})^2 \sinh[10(y-\frac{1}{2})] + 1$

$\phi(1,y) = \frac{1}{4}\sinh(5) + (y-\frac{1}{2})^2 \sinh[10(y-\frac{1}{2})] + e^{2y}$

$\phi(x,0) = \frac{1}{4}\sinh(-5) + (x-\frac{1}{2})^2 \sinh[10(x-\frac{1}{2})] + 1$

$\phi(x,1) = \frac{1}{4}\sinh(5) + (x-\frac{1}{2})^2 \sinh[10(x-\frac{1}{2})] + e^{2x}$

Also given analytical solution

$$\phi(x,y) = (x-\tfrac{1}{2})^2 \sinh[10(x-\tfrac{1}{2})] + (y-\tfrac{1}{2})^2 \sinh[10(y-\tfrac{1}{2})] + e^{2xy}.$$

We need to find ~~numberi~~ numerical solution of above poisson equation using

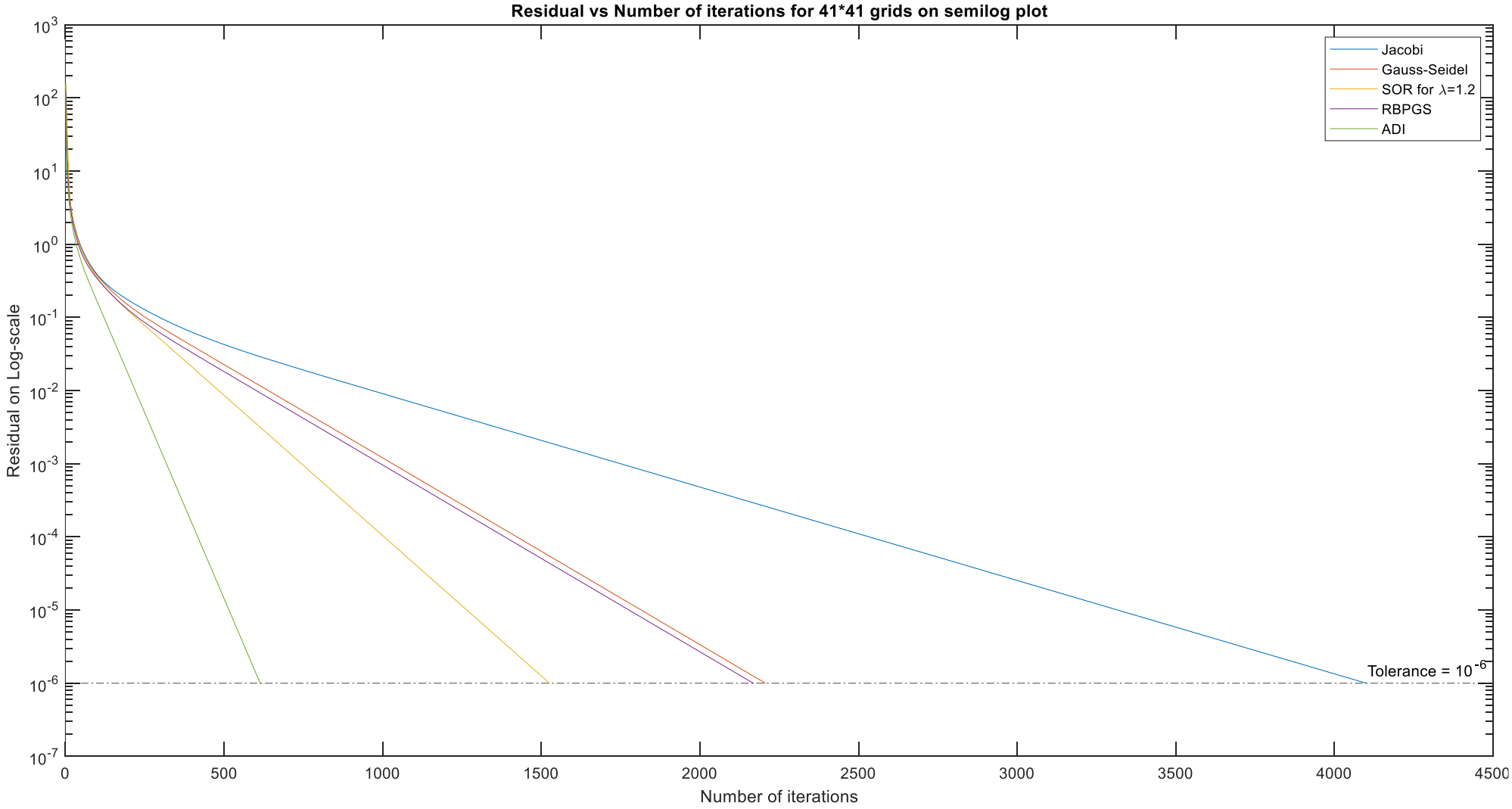
1. Jacobi
2. Gauss-Seidel
3. SOR for different value of λ especially $\lambda = 1.2$
4. RBPGS
5. ADI

using 41×41 Grids.
and 81×81 Grids.

All the above numerical solution technique is implemented through MATLAB code.
Please find the code (.m file) attached with this report.

Plots for 41*41 grids

1. Residual plots for different iterative methods



After plotting residual for various iterative method vs number of iterations took for convergence we can make following observation. (Note: Tolerance = 10^{-6}).

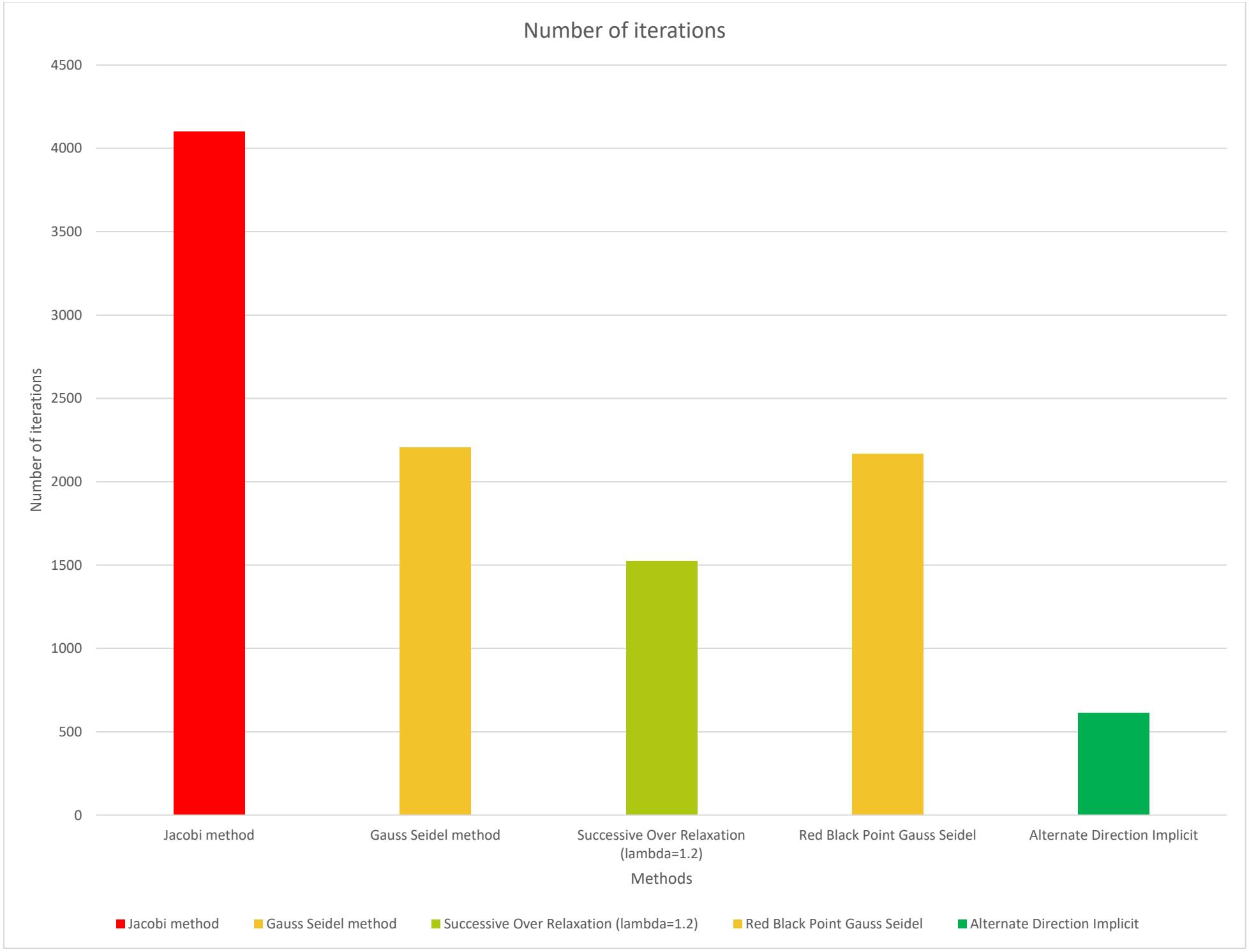
- Jacobi method took 4101 iterations to converge to the tolerance value.
- Gauss Seidel method took 2207 iterations to converge to the tolerance value.
- SOR method with λ=1.2 took 1526 iterations to converge to the tolerance value.
- RBPGS method took 2169 iterations to converge to the tolerance value.
- ADI method took 615 iterations to converge to the tolerance value.

It was also observed during analysis that for λ=1.76 for SOR the Convergence rate increase further.

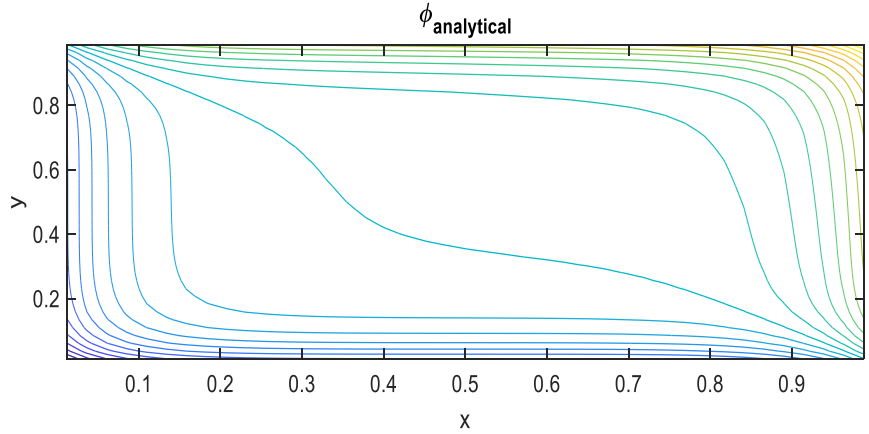
The rate of convergence is slowest for the Jacobi method as it is an explicit method and uses all the n^{th} value of Φ for computing the $(n+1)^{th}$ value of Φ. This is improved by Gauss Seidel method by taking $(n+1)^{th}$ value of Φ (i.e., for (i-1, j), (i, j-1))which has been computed for that iteration. The Successive over relaxation method further accelerates the rate of convergence by introducing λ ($0 < \lambda < 2$). Here λ=1.2 is used. The Red Black Point Gauss Seidel enable parallel computing. However, here it is implemented with simple MATLAB code. It is observed that its rate of convergence is of the order of Gauss Seidel method but a little less than it. We note that Alterative Direction Implicit method has the fastest convergence rate. This was expected as we solve for entire x-coordinate for some y and entire y-coordinate for some x in one go using TDMA, hence the effect of Boundary Condition propagates faster inside the domain. Therefore, it converges faster.

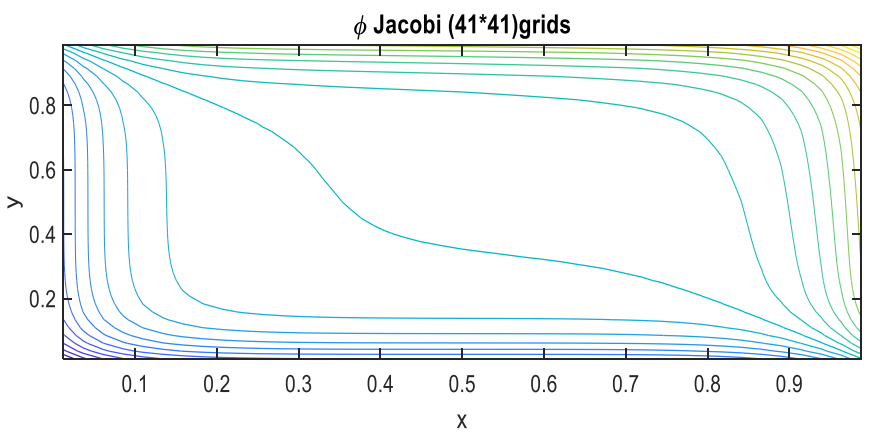
S.no	Method	Number of iterations	Number of times faster that Jacobi
1	Jacobi method	4101	1 time
2	Gauss Seidel method	2207	1.86 times
3	Successive Over Relaxation (λ=1.2)	1526	2.69 times
4	Red Black Point Gauss Seidel	2169	1.89 times
5	Alternate Direction Implicit	615	6.67 times

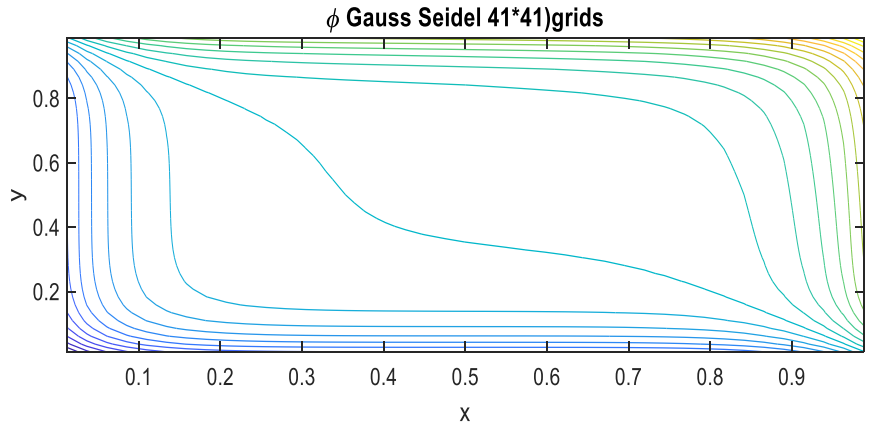
Hence, the rate of convergence is fastest for ADI and slowest for Jacobi method.

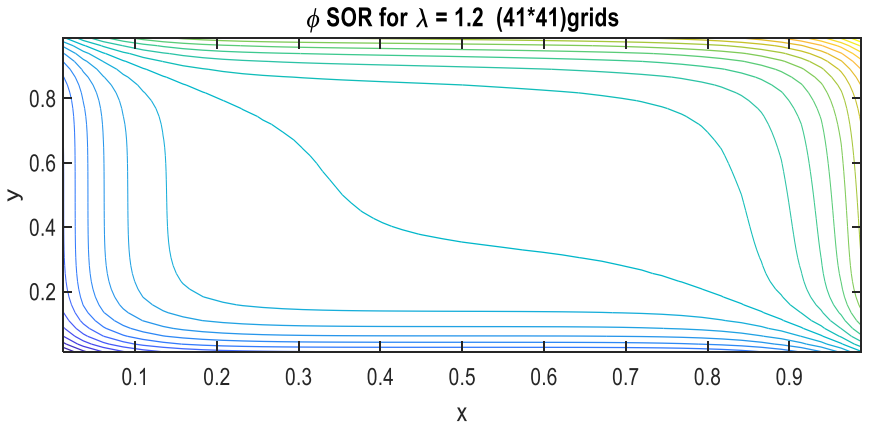

Number of iterations
4500
4000
3500
3000
2500
2000
1500
1000
500
0
Number of iterations
Jacobi method
Gauss Seidel method
Successive Over Relaxation (lambda=1.2)
Red Black Point Gauss Seidel
Alternate Direction Implicit
Methods
Jacobi method
Gauss Seidel method
Successive Over Relaxation (lambda=1.2)
Red Black Point Gauss Seidel
Alternate Direction Implicit

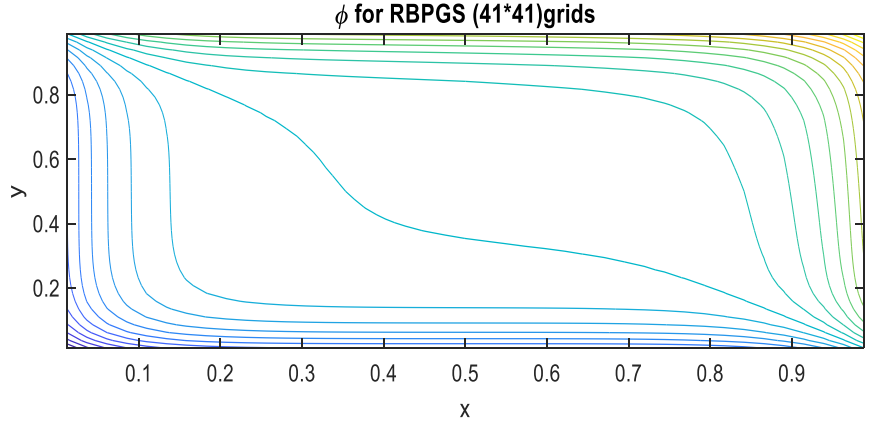
2. Contour plots for Φ for analytical solution and different iterative methods

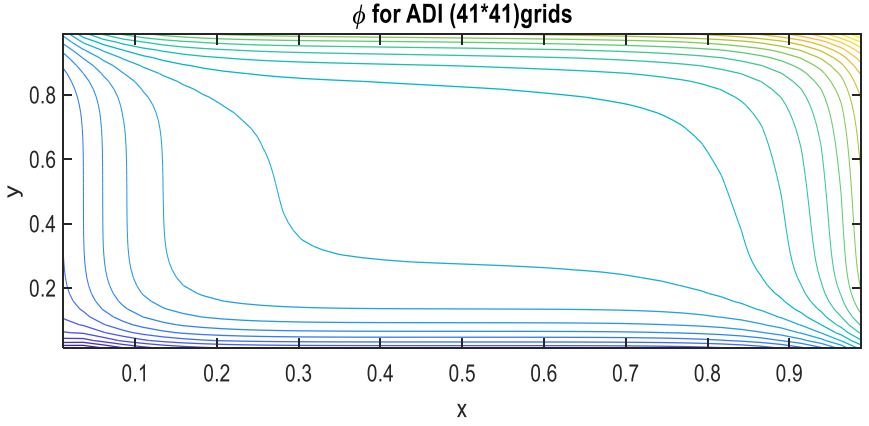




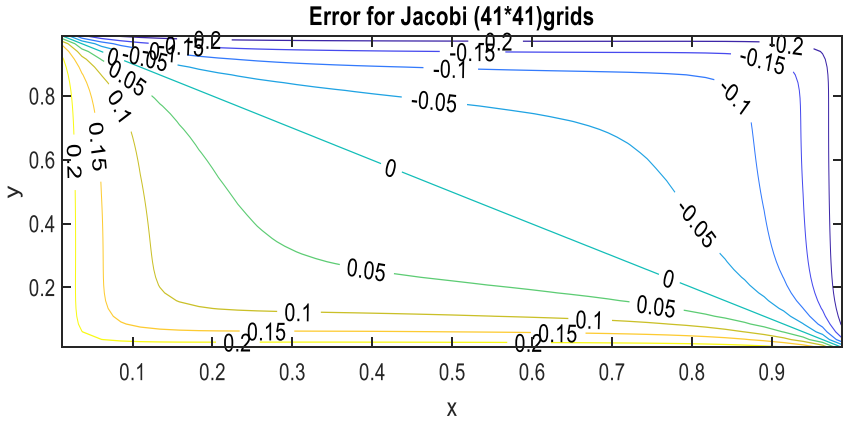


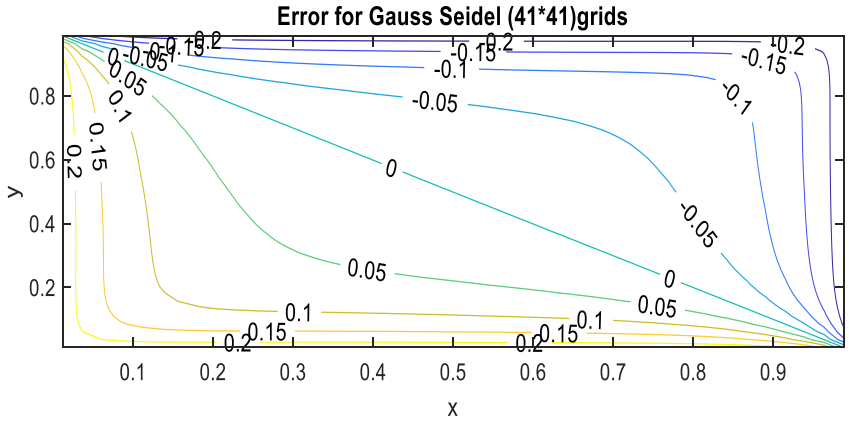


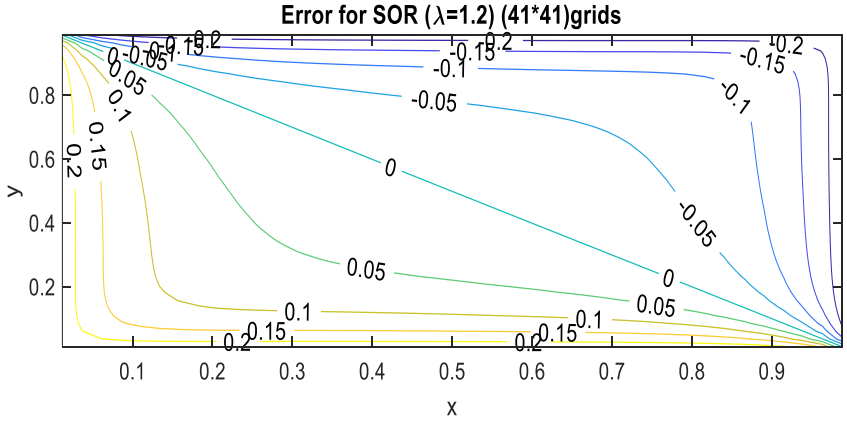


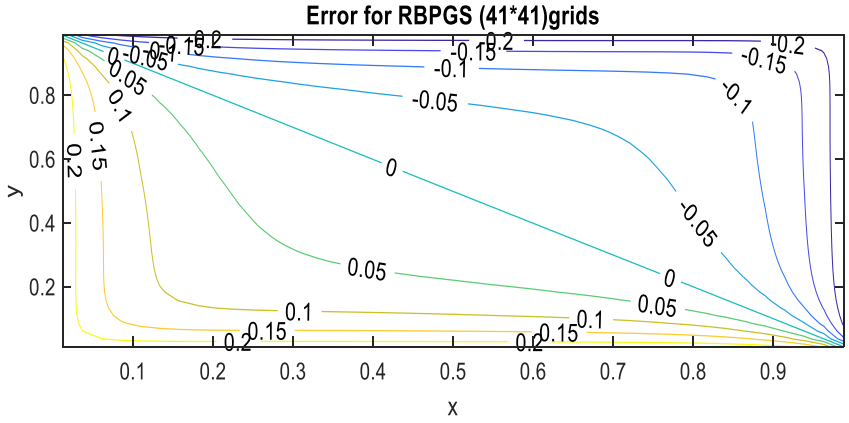


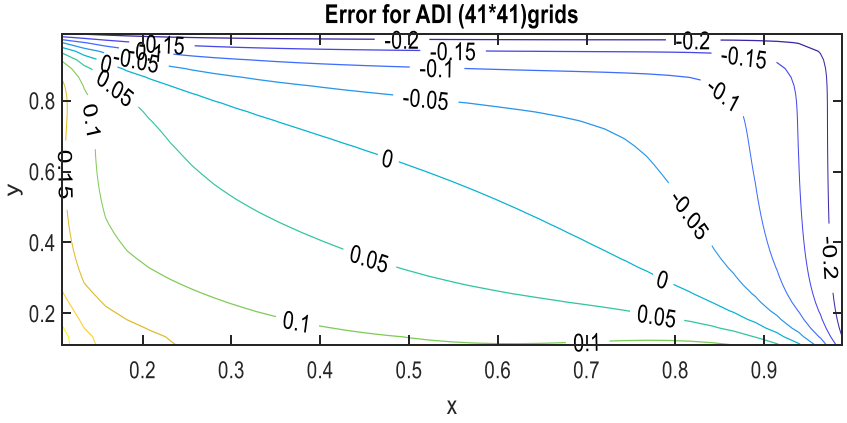
3. Contour plots of error ($\Phi_{converged} - \Phi_{analytical}$) for different iterative methods



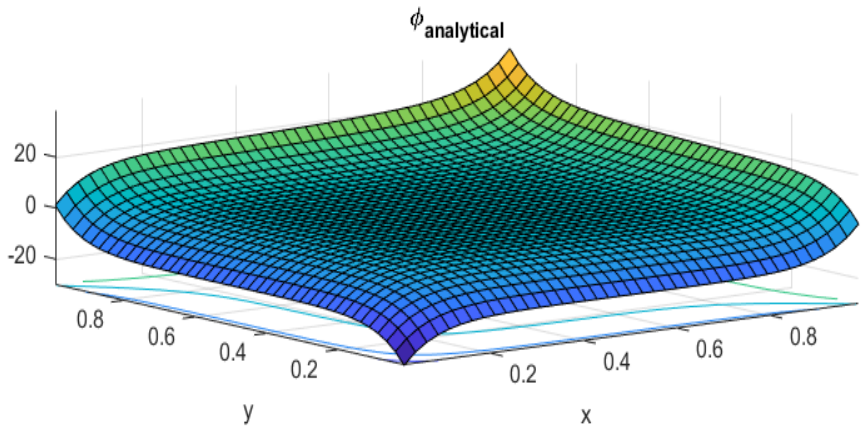


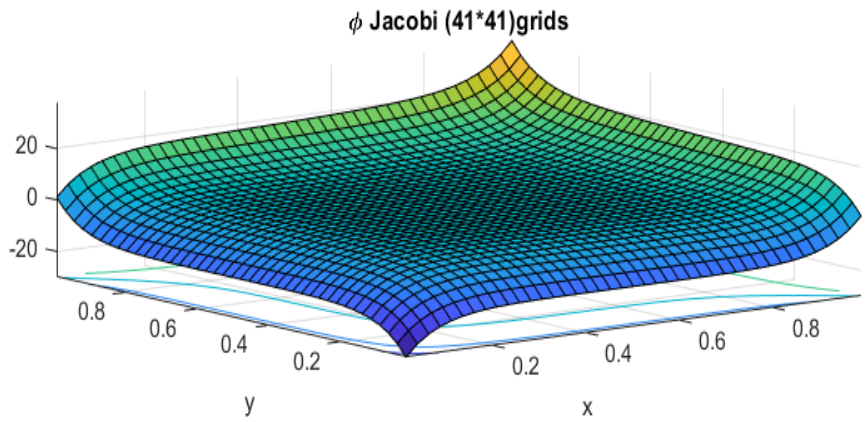


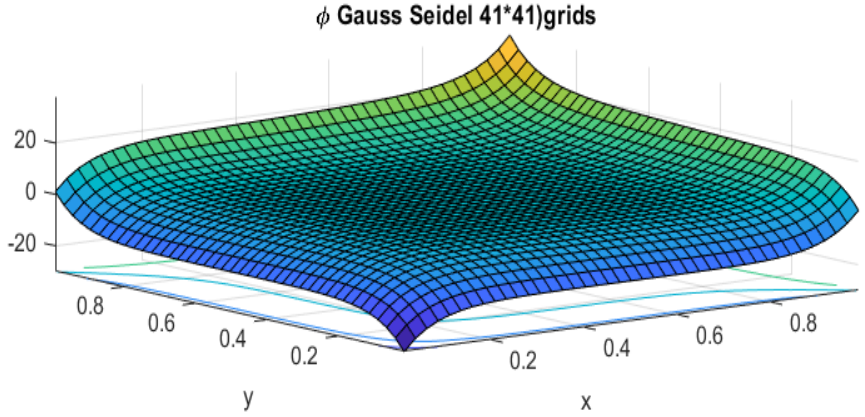


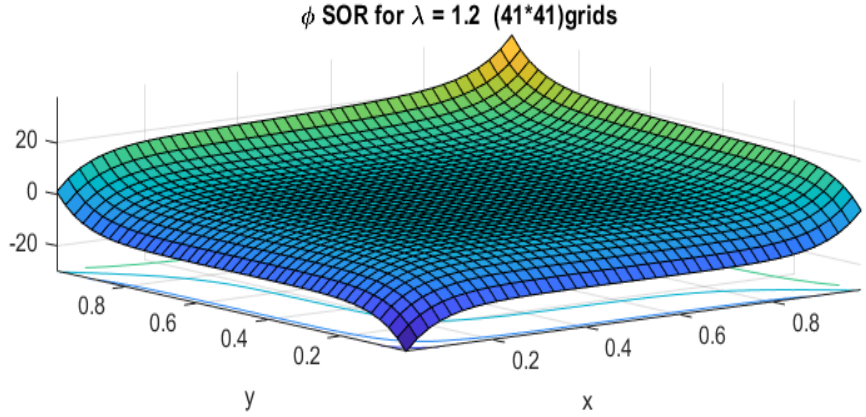


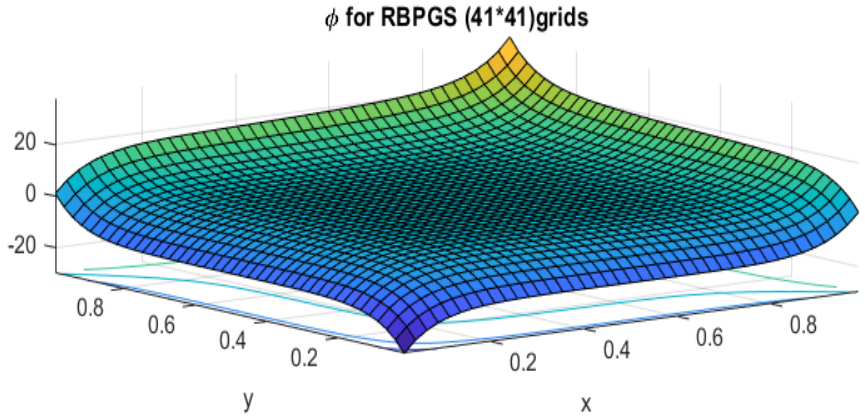
4. Surface plots for Φ for analytical solution and different iterative methods

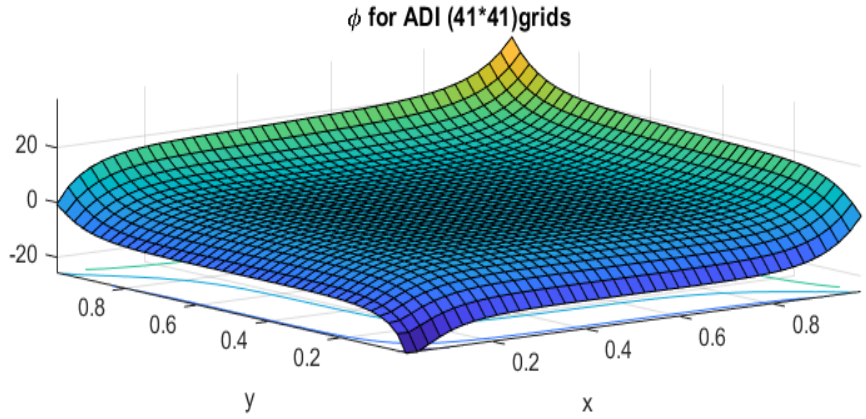






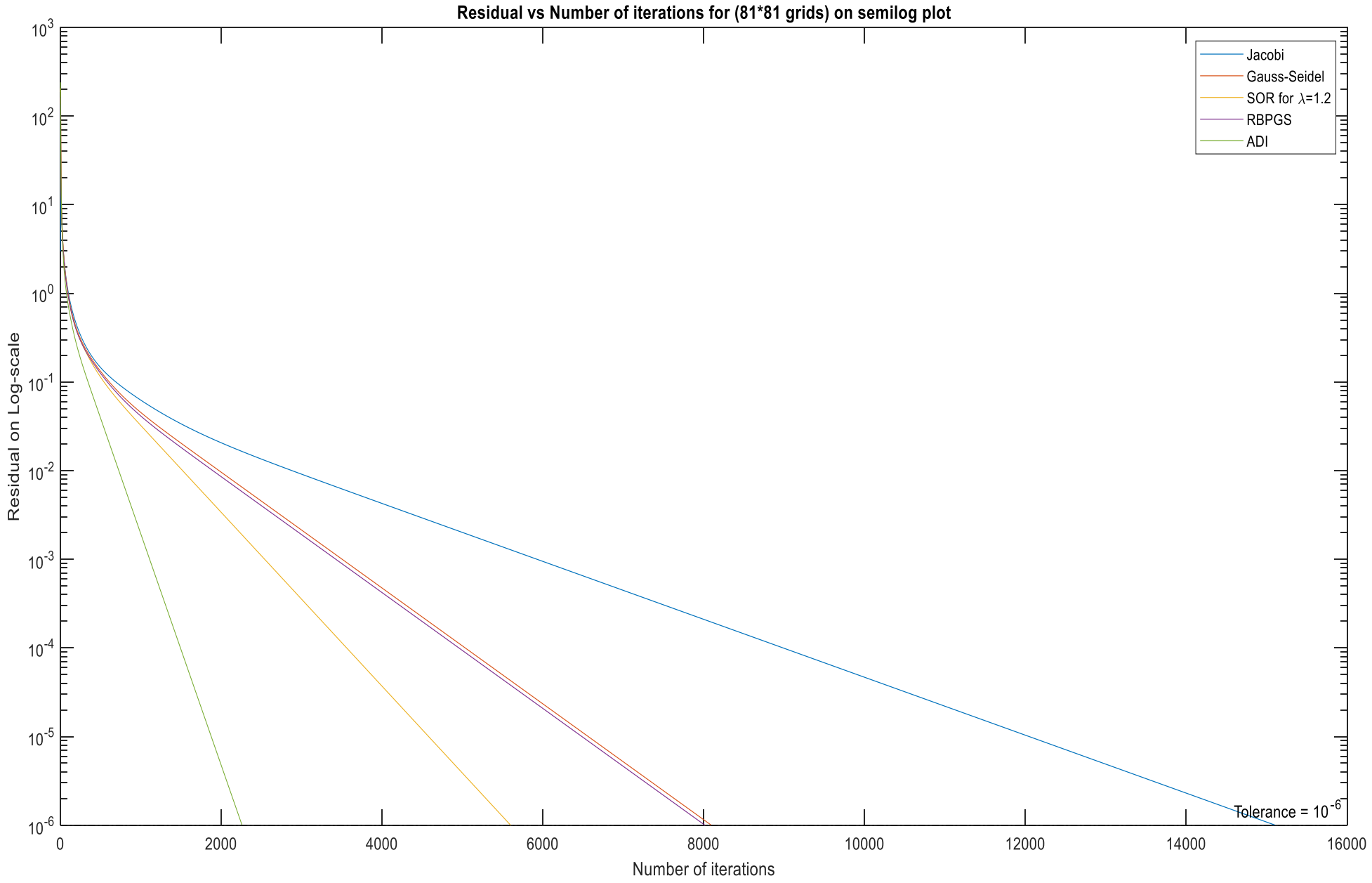






Plots for 81*81 grids

1. Residual plots for different iterative methods



After plotting residual for various iterative method vs number of iterations took for convergence we can make following observation. (Note: Tolerance = 10^{-6}).

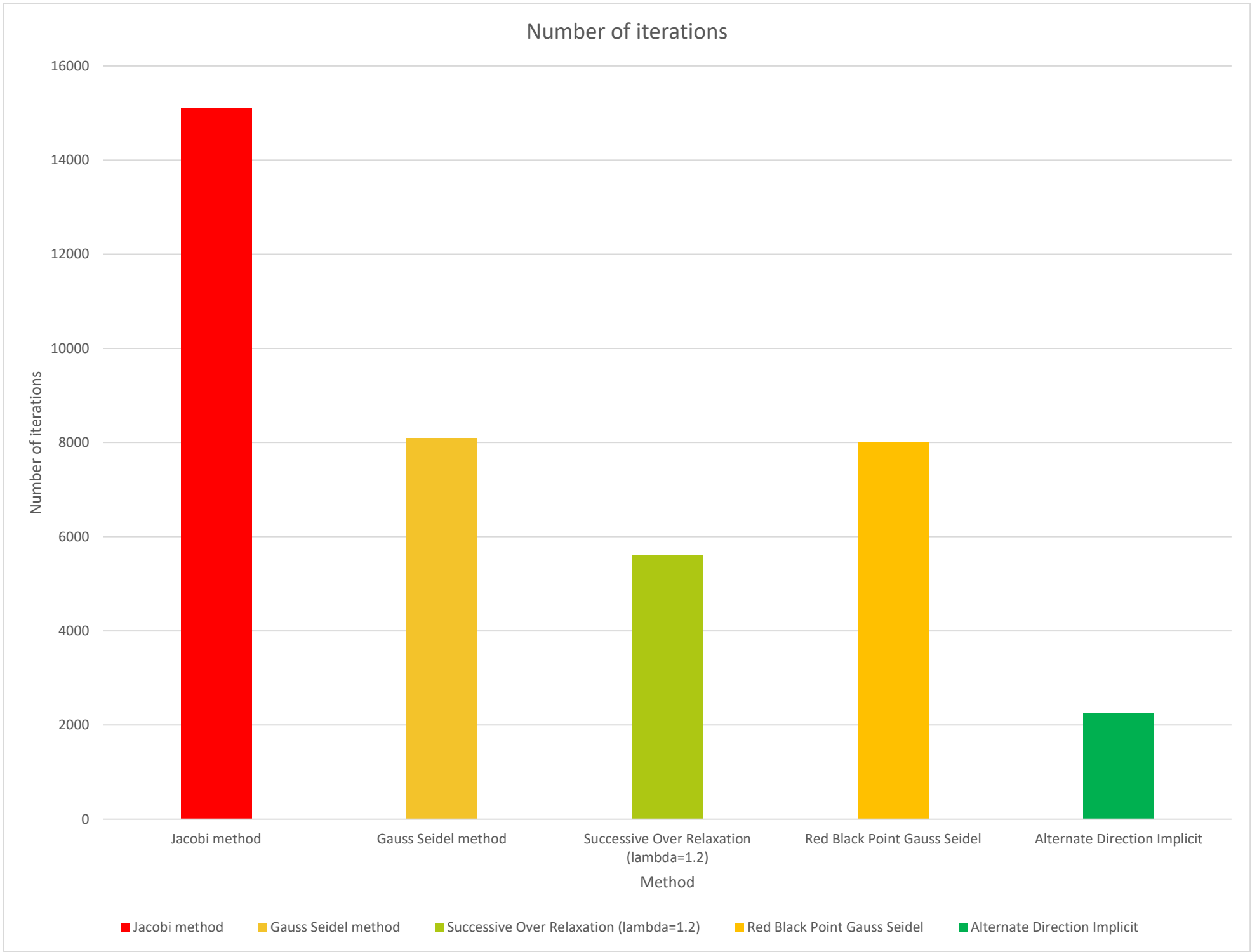
- Jacobi method took 15107 iterations to converge to the tolerance value.
- Gauss Seidel method took 8093 iterations to converge to the tolerance value.
- SOR method with λ=1.2 took 5598 iterations to converge to the tolerance value.
- RBPGS method took 8014 iterations to converge to the tolerance value.
- ADI method took 2258 iterations to converge to the tolerance value.

It was also observed during analysis that for λ=1.76 for SOR the Convergence rate increase further.

We can make following observations and comments from residual plots. The rate of convergence is slowest for the Jacobi method as it is an explicit method and uses all the n^{th} value of Φ for computing the $(n+1)^{th}$ value of Φ. This is improved by Gauss Seidel method by taking $(n+1)^{th}$ value of Φ (i.e., for (i-1, j), (i, j-1))which has been computed for that iteration. The Successive over relaxation method further accelerates the rate of convergence by introducing λ ($0 < \lambda < 2$). Here λ=1.2 is used. The Red Black Point Gauss Seidel enable parallel computing. However, here it is implemented with simple MATLAB code. It is observed that its rate of convergence is of the order of Gauss Seidel method but a little less than it. We note that Alterative Direction Implicit method has the fastest convergence rate. This is because as we solve for entire x-coordinate for some y and entire y-coordinate for some x in one go using TDMA, hence the effect of Boundary Condition propagates faster inside the domain. Therefore, it converges faster.

S.no	Method	Number of iterations	Number of times faster that Jacobi
1	Jacobi method	15107	1 time
2	Gauss Seidel method	8093	1.87 times
3	Successive Over Relaxation (λ=1.2)	5598	2.70 times
4	Red Black Point Gauss Seidel	8014	1.89 times
5	Alternate Direction Implicit	2258	6.69 times

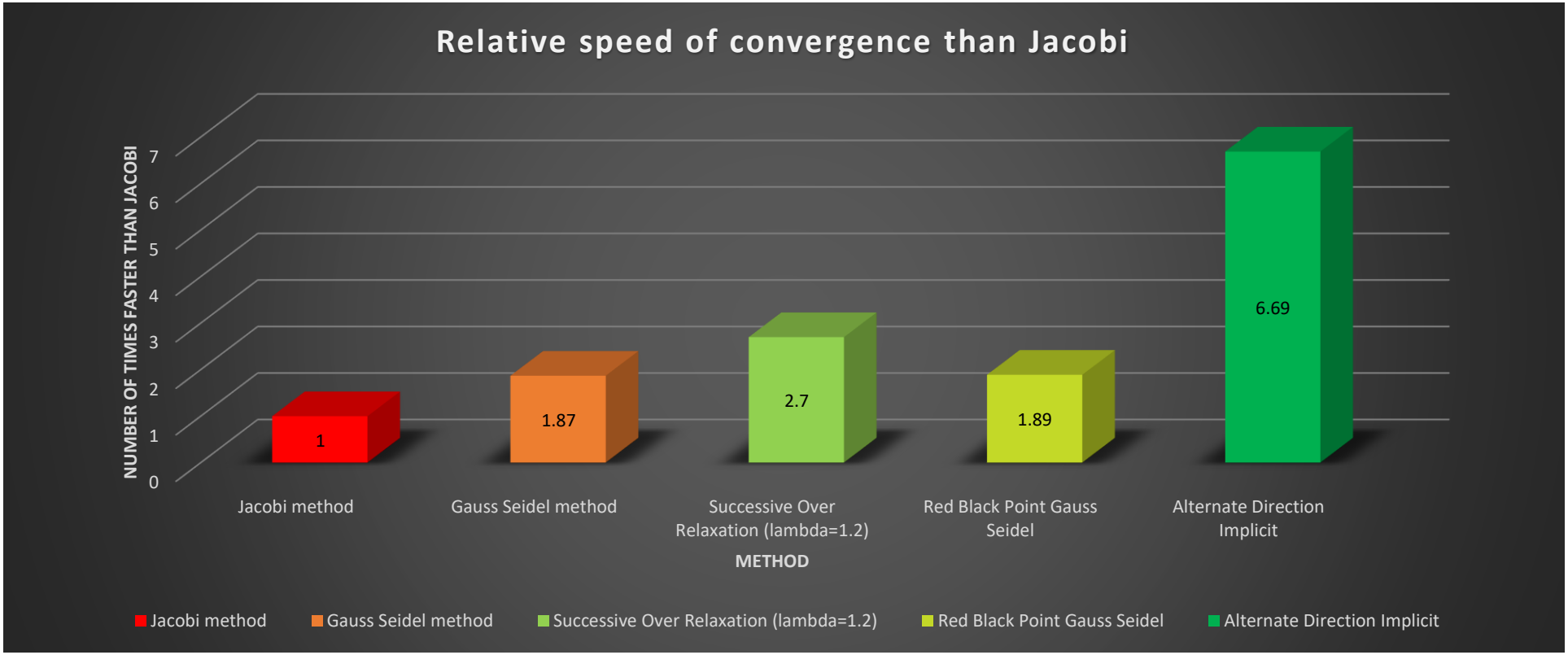
Hence, the rate of convergence is fastest for ADI and slowest for Jacobi method.


Number of iterations
16000
14000
12000
10000
8000
6000
4000
2000
0
Number of iterations
Jacobi method
Gauss Seidel method
Successive Over Relaxation (lambda=1.2)
Red Black Point Gauss Seidel
Alternate Direction Implicit
Method
Jacobi method
Gauss Seidel method
Successive Over Relaxation (lambda=1.2)
Red Black Point Gauss Seidel
Alternate Direction Implicit

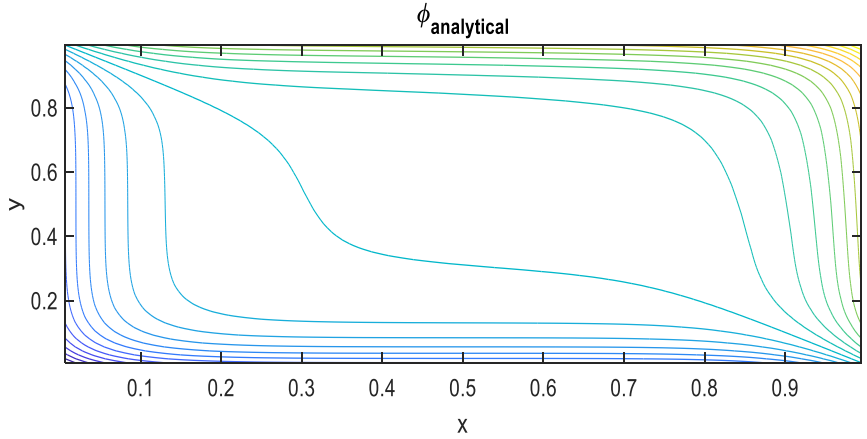
After looking at plots for residual for 41x41 grids and 81x81 grids we finally conclude that with increase in number of grids the number of iterations required for convergence for same tolerance value (here, 10^{-6})increases rapidly for all iterative methods used here. This is because we have to solve for Φ at a greater number of nodes.

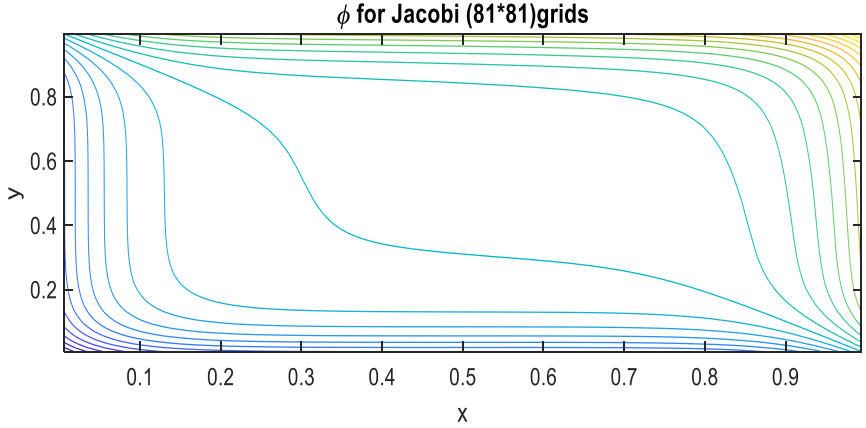
Finally, we conclude that

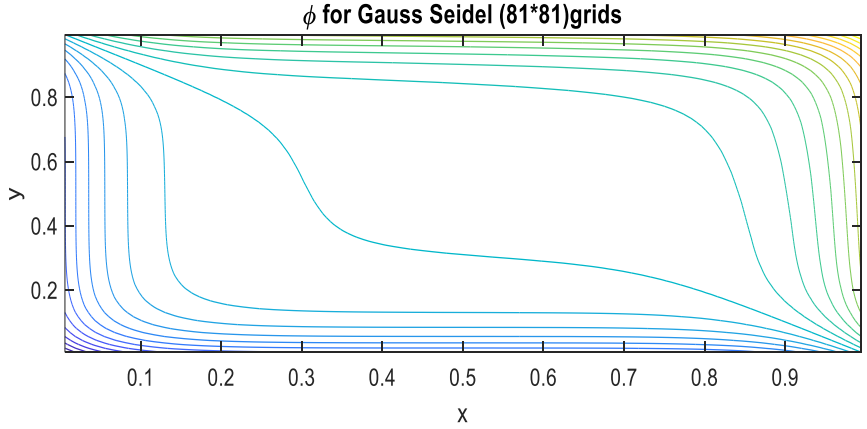
S.no	Method	Number of times faster than Jacobi
1	Jacobi method	1 time
2	Gauss Seidel method	1.87 times
3	Successive Over Relaxation (λ=1.2)	2.70 times
4	Red Black Point Gauss Seidel	1.89 times
5	Alternate Direction Implicit	6.69 times

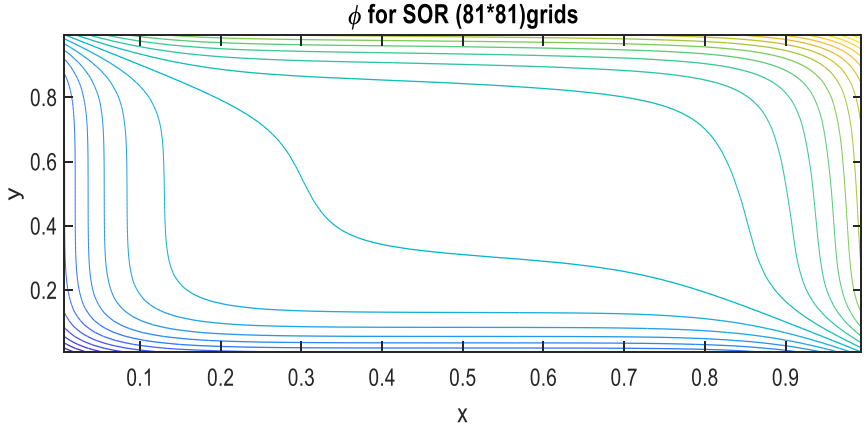


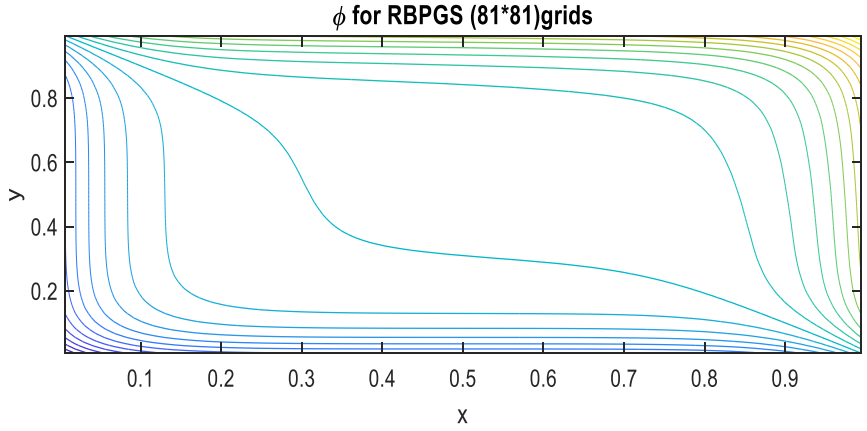
2.Contour plots for Φ for analytical solution and different iterative methods

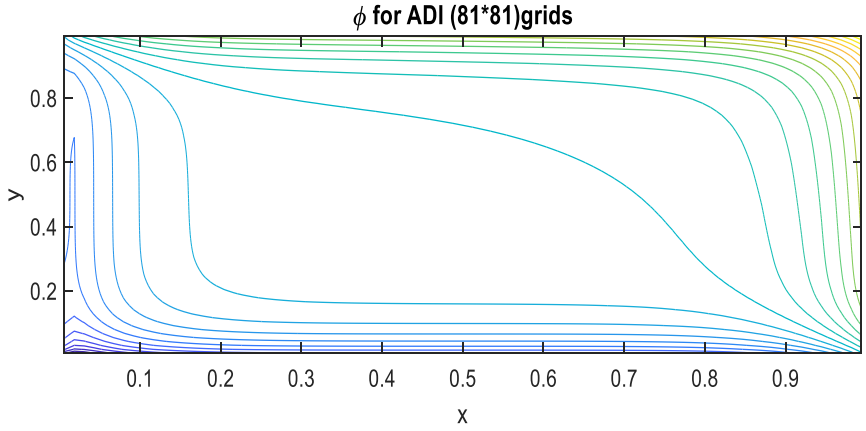




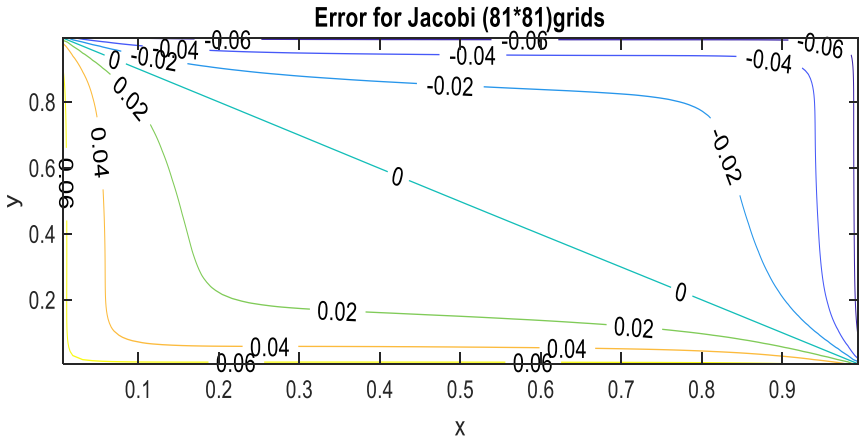


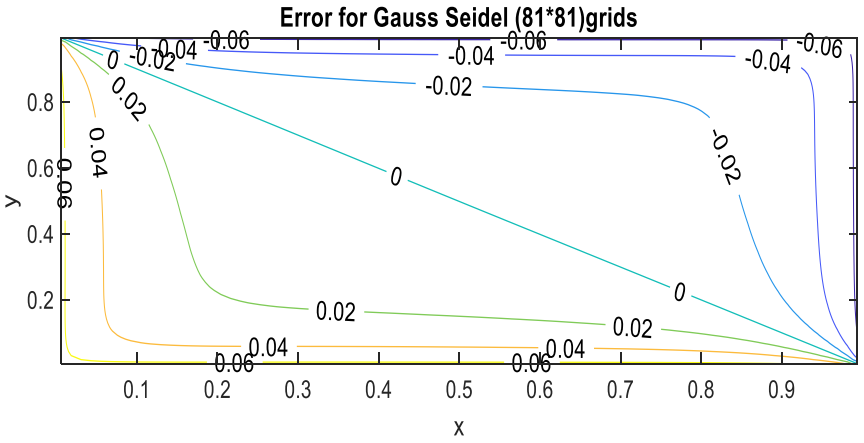


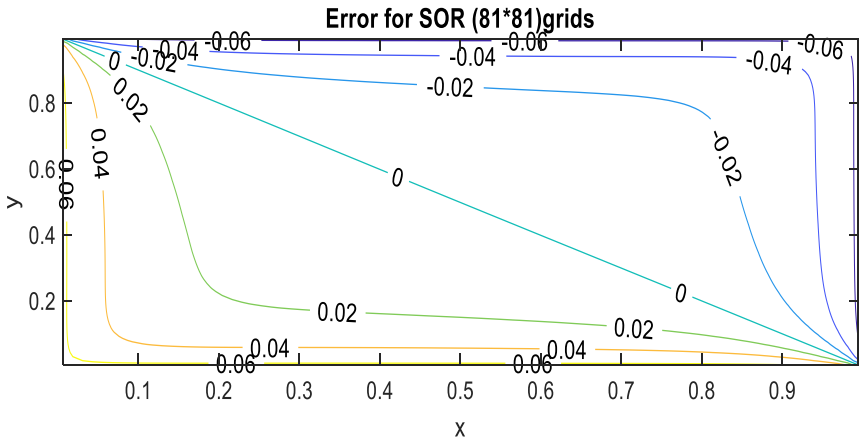


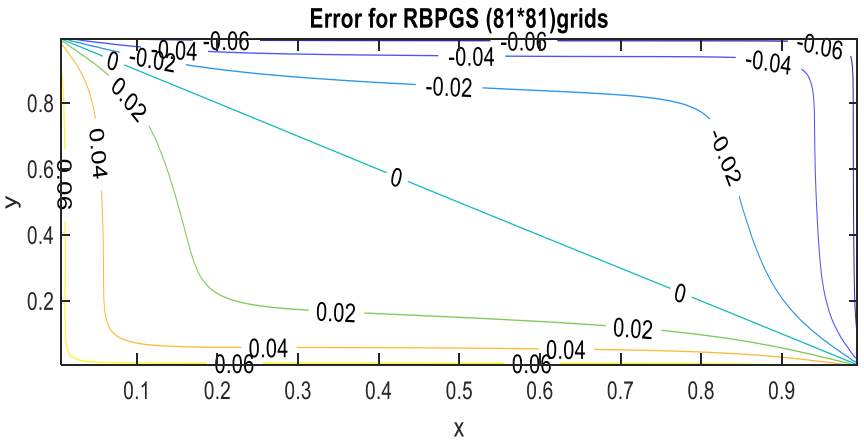


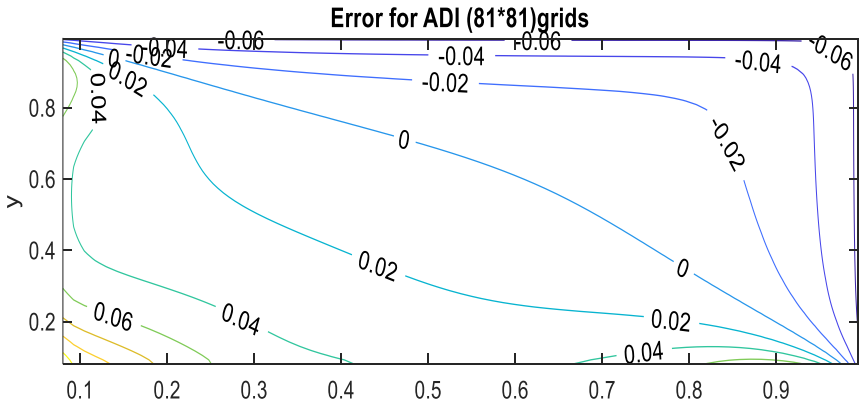
3.Contour plots of error ($\Phi_{converged} - \Phi_{analytical}$) for different iterative methods



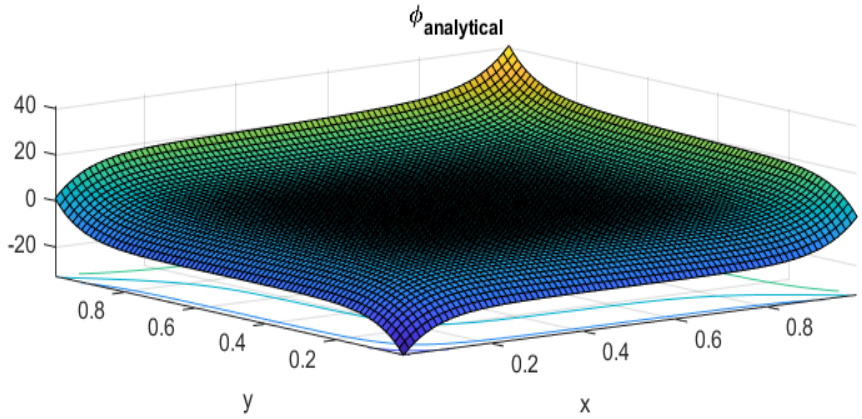


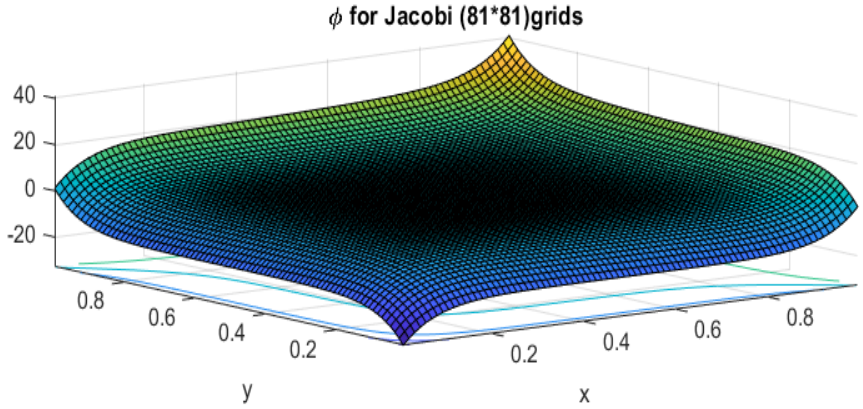


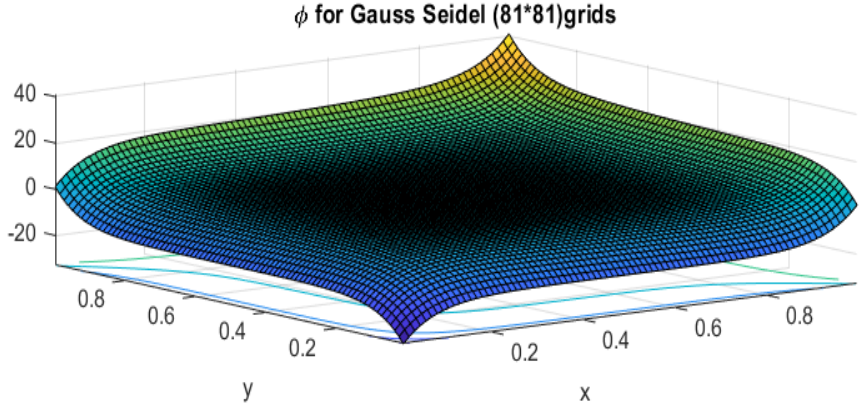


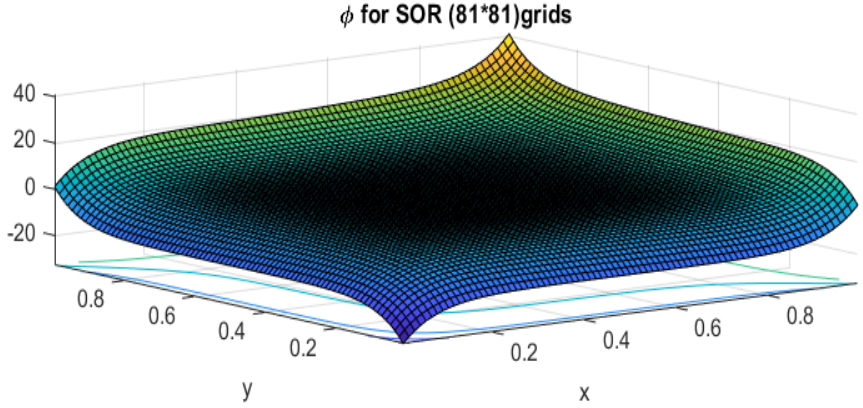


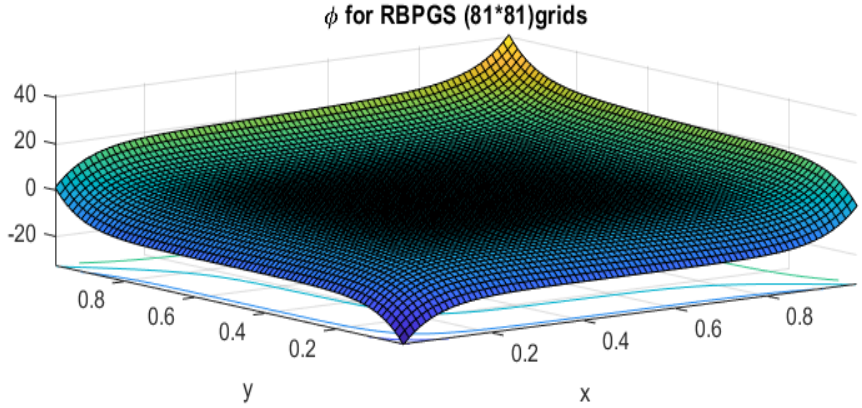
4.Surface plots for Φ for analytical solution and different iterative methods

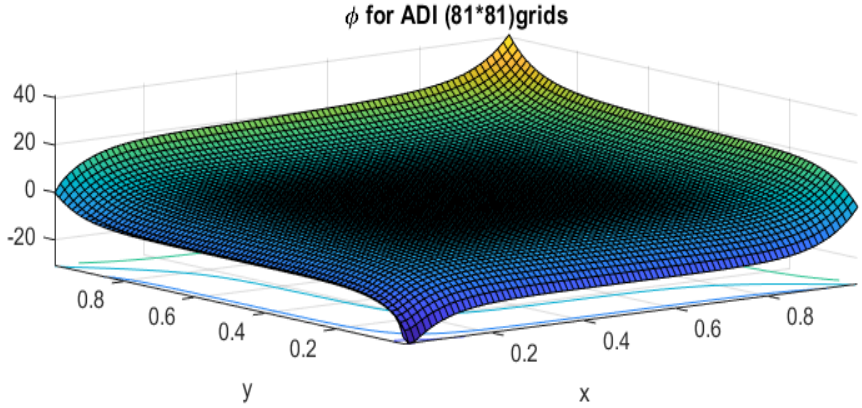












1. Please find the MATLAB code (*.m files) for iterative methods and TDMA included with the submission.
2. Please find the plots also included with the submission.
3. Also find Φ solutions and residual variable file (in *.mat format) for both 41x41 grids and 81x81 grids generated after executing the code included with submission.
4. I am also including code (whose *.m file in point 1 is included in submission) in this *.pdf for your quick reference.

Thank You

Rikesh Sharma
180606

Algorithm for implementing different iterative methods

```
%Rikesh Sharma
%180606
%ME630A (CFD) | < submission date: 22/09/2021 >
function poisson_solution()

%For 41*41 GRIDS
%Initialization of domain, grids, coeffients of equations and Boundary Conditions
domainX=1;
domainY=1;
Nx=41;
Ny=41;
nxp2=Nx+2;
nyp2=Ny+2;
deltaX=domainX/Nx;
deltaY=domainY/Ny;
pc = -2*(1/deltaX^2 +1/deltaY^2);
px = 1/deltaX^2;
py = 1/deltaY^2;
x=(-deltaX/2 : deltaX : domainX+deltaX/2);
y=(-deltaY/2 : deltaY : domainY+deltaY/2);
bcx=zeros(2,nyp2); bcy=zeros(nxp2,2);
bcx(1,:)=0.25.*sinh(-5)+(y-0.5).^2.*sinh(10.*(y-0.5))+1;
bcx(2,:)=0.25.*sinh(5)+(y-0.5).^2.*sinh(10.*(y-0.5))+exp(2.*y);
```

```
bcy(:,1)=0.25.*sinh(-5)+(x-0.5).^2.*sinh(10.*(x-0.5))+1;
bcy(:,2)=0.25.*sinh(5)+(x-0.5).^2.*sinh(10.*(x-0.5))+exp(2.*x);

[x,y]=meshgrid(x,y);
phi=zeros(nxp2,nyp2); phi_old=zeros(nxp2,nyp2);
Sphi=2.*sinh(10.*(x-0.5))+40.*(x-0.5).*cosh(10.*(x-0.5))+100.*((x-0.5).^2).*sinh(10.*(x-0.5))+2.*sinh(10.*(y-
0.5))+40.*(y-0.5).*cosh(10.*(y-0.5))+100.*((y-0.5).^2).*sinh(10.*(y-0.5))+4.*(x.^2+y.^2).*exp(2.*x.*y);

%Solution for 41*41 Grids%
residual_Jacobi=zeros(2,1);
iter_Jacobi=zeros(2,1);
maxError = 1e-6;
error=1;
iter=0;
%Jacobi method%
while(error>maxError && iter<100000)
    iter=iter+1;

    phi(1,2:Ny+1)   = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
    phi(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
    phi(2:Nx+1,1)   = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
    phi(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

    for i=2:Nx+1
        for j=2:Ny+1
            phi(i,j)=(Sphi(i,j)-(px*phi_old(i-1,j)+px*phi_old(i+1,j)+py*phi_old(i,j-1)+py*phi_old(i,j+1)))/pc;
        end
    end
%     disp(iter);

    errorMatrix=(phi(2:Nx+1,2:Ny+1)-phi_old(2:Nx+1,2:Ny+1));
    error = norm(errorMatrix,2);
    residual_Jacobi(iter,1)=error;
    iter_Jacobi(iter,1)=iter;
    phi_old=phi;
%     disp(error);
end
x=(deltaX/2 : deltaX : domainX-deltaX/2);
y=(deltaY/2 : deltaY : domainY-deltaY/2);
[x,y]=meshgrid(x,y);
phi_analytic=(x-0.5).^2.*sinh(10.*(x-0.5))+(y-0.5).^2.*sinh(10.*(y-0.5))+exp(2.*x.*y);
phi_Jacobi=phi(2:Nx+1,2:Ny+1);


phi=zeros(nxp2,nyp2); phi_old=zeros(nxp2,nyp2);
residual_Gauss=zeros(2,1);
iter_Gauss=zeros(2,1);
```

```
error=1;
iter=0;
%Gauss Seidel method%
while(error>maxError && iter<100000)
    iter=iter+1;


    phi(1,2:Ny+1)   = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
    phi(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
    phi(2:Nx+1,1)   = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
    phi(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

    for i=2:Nx+1
        for j=2:Ny+1
            phi(i,j)=(Sphi(i,j)-(px*phi(i-1,j)+px*phi_old(i+1,j)+py*phi(i,j-1)+py*phi_old(i,j+1)))/pc;
        end
    end
%     disp(iter);

    errorMatrix=(phi(2:Nx+1,2:Ny+1)-phi_old(2:Nx+1,2:Ny+1));
    error = norm(errorMatrix,2);
    residual_Gauss(iter,1)=error;
    iter_Gauss(iter,1)=iter;
    phi_old=phi;
%     disp(error);
end

phi_Gauss=phi(2:Nx+1,2:Ny+1);

phi=zeros(nxp2,nyp2); phi_old=zeros(nxp2,nyp2);
residual_SOR=zeros(2,1);
iter_SOR=zeros(2,1);
lambda=1.2;
error=1;
iter=0;
%Succesive Over Relaxation method%
while(error>maxError && iter<100000)
    iter=iter+1;

    phi(1,2:Ny+1)   = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
    phi(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
    phi(2:Nx+1,1)   = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
    phi(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

    for i=2:Nx+1
        for j=2:Ny+1
```

```
            phi(i,j)=(1-lambda)*phi(i,j)+lambda*(Sphi(i,j)-(px*phi(i-1,j)+px*phi_old(i+1,j)+py*phi(i,j-
1)+py*phi_old(i,j+1)))/pc;
        end
    end
%     disp(iter);

    errorMatrix=(phi(2:Nx+1,2:Ny+1)-phi_old(2:Nx+1,2:Ny+1));
    error = norm(errorMatrix,2);
    residual_SOR(iter,1)=error;
    iter_SOR(iter,1)=iter;
    phi_old=phi;
%     disp(error);
end

phi_SOR=phi(2:Nx+1,2:Ny+1);

phi=zeros(nxp2,nyp2); phi_old=zeros(nxp2,nyp2);
residual_RBPGS=zeros(2,1);
iter_RBPGS=zeros(2,1);
error=1;
iter=0;
%RBPGS method%
while(error>maxError && iter<100000)
    iter=iter+1;


    phi(1,2:Ny+1)   = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
    phi(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
    phi(2:Nx+1,1)   = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
    phi(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

    for i=2:Nx+1
        m = 2+mod(i,2);
        for j=m:2:Ny+1
            phi(i,j) = (Sphi(i,j)-(px*phi_old(i-1,j)+px*phi_old(i+1,j)+py*phi_old(i,j-1)+py*phi_old(i,j+1)))/pc;
        end
    end

    for i=2:Nx+1
        m = 2+mod(i+1,2);
        for j=m:2:Ny+1
            phi(i,j) = (Sphi(i,j)-(px*phi(i-1,j)+px*phi(i+1,j)+py*phi(i,j-1)+py*phi(i,j+1)))/pc;
        end
    end

%     disp(iter);
```

```
    errorMatrix=(phi(2:Nx+1,2:Ny+1)-phi_old(2:Nx+1,2:Ny+1));
    error = norm(errorMatrix,2);
    residual_RBPGS(iter,1)=error;
    iter_RBPGS(iter,1)=iter;
    phi_old=phi;
%     disp(error);
end

phi_RBPGS=phi(2:Nx+1,2:Ny+1);

%ADI method%
phi=zeros(nxp2,nyp2); phi_old=zeros(nxp2,nyp2);
residual_ADI=zeros(2,1);
iter_ADI=zeros(2,1);
error=1;
iter=0;
phi_old(1,2:Ny+1)   = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
phi_old(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
phi_old(2:Nx+1,1)   = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
phi_old(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

ar=zeros(Nx,1);  ar(2:Nx,1)=-px;
br=zeros(Nx,1);  br(:,1)=-pc;
cr=zeros(Nx,1);  cr(2:Nx,1)=-px;
dr=zeros(Nx,1);
ac=zeros(Ny,1);  ac(2:Ny,1)=-py;
bc=zeros(Ny,1);  bc(:,1)=-pc;
cc=zeros(Ny,1);  cc(2:Ny,1)=-py;
dc=zeros(Ny,1);

while(error>maxError && iter<100000)
    iter=iter+1;

    phi(1,2:Ny+1)   = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
    phi(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
    phi(2:Nx+1,1)   = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
    phi(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

    for j=2:Ny+1
        dr(1,1)=py.*phi(2,j-1)+py.*phi_old(2,j+1)-Sphi(2,j)+px*phi(1,j);
        dr(2:Nx-1,1)=py.*phi(3:Nx,j-1)+py.*phi_old(3:Nx,j+1)-Sphi(3:Nx,j);
        dr(Nx,1)=py.*phi(Nx+1,j-1)+py.*phi_old(Nx+1,j+1)-Sphi(Nx+1,j)+px*phi(Nx+2,j);
        phi(2:Nx+1,j)=TDMA(ar,br,cr,dr);
    end


    phi(1,2:Ny+1)   = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
```

```
    phi(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
    phi(2:Nx+1,1)  = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
    phi(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

    for i=2:Nx+1
        dc(1,1)=px.*phi(i-1,2)+px.*phi(i+1,2)-Sphi(i,2)+py*phi(i,1);
        dc(2:Ny-1,1)=px.*phi(i-1,3:Ny)+px.*phi(i+1,3:Ny)-Sphi(i,3:Ny);
        dc(Ny,1)=px.*phi(i-1,Ny+1)+px.*phi(i+1,Ny+1)-Sphi(i,Ny+1) + py*phi(i,Ny+2);
        phi(i,2:Ny+1)=TDMA(ac,bc,cc,dc);
    end
    phi(1,2:Ny+1)  = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
    phi(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
    phi(2:Nx+1,1)  = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
    phi(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

%     disp(iter);

    errorMatrix=(phi(2:Nx+1,2:Ny+1)-phi_old(2:Nx+1,2:Ny+1));
    error = norm(errorMatrix,2);
    residual_ADI(iter,1)=error;
    iter_ADI(iter,1)=iter;
    phi_old=phi;
%     disp(error);
end

phi_ADI=phi(2:Nx+1,2:Ny+1);

%Plots for 41*41 Grids%
x=(deltaX/2 : deltaX : domainX-deltaX/2);
y=(deltaY/2 : deltaY : domainY-deltaY/2);
[x,y]=meshgrid(x,y);

%Plotting phi contours
figure
subplot(3,2,1)
contour(x,y,phi_analytic,25);
xlabel('x');
ylabel('y');
title('{\phi}_{analytical}');
subplot(3,2,2)
contour(x,y,phi_Jacobi,25);
xlabel('x');
ylabel('y');
title('{\phi} Jacobi (41*41)grids');
subplot(3,2,3)
contour(x,y,phi_Gauss,25);
xlabel('x');
```

```
ylabel('y');
title('{\phi} Gauss Seidel 41*41)grids');
subplot(3,2,4)
contour(x,y,phi_SOR,25);
xlabel('x');
ylabel('y');
title('{\phi} SOR for \lambda = 1.2  (41*41)grids');
subplot(3,2,5)
contour(x,y,phi_RBPGS,25);
xlabel('x');
ylabel('y');
title('{\phi} for RBPGS (41*41)grids');
subplot(3,2,6)
contour(x,y,phi_ADI,25);
xlabel('x');
ylabel('y');
title('{\phi} for ADI (41*41)grids');

%Plotting phi surfaces
figure
subplot(3,2,1)
surfc(x,y,phi_analytic);
xlabel('x');
ylabel('y');
title('{\phi}_{analytical}');
subplot(3,2,2)
surfc(x,y,phi_Jacobi);
xlabel('x');
ylabel('y');
title('{\phi} Jacobi (41*41)grids');
subplot(3,2,3)
surfc(x,y,phi_Gauss);
xlabel('x');
ylabel('y');
title('{\phi} Gauss Seidel 41*41)grids');
subplot(3,2,4)
surfc(x,y,phi_SOR);
xlabel('x');
ylabel('y');
title('{\phi} SOR for \lambda = 1.2  (41*41)grids');
subplot(3,2,5)
surfc(x,y,phi_RBPGS);
xlabel('x');
ylabel('y');
title('{\phi} for RBPGS (41*41)grids');
subplot(3,2,6)
surfc(x,y,phi_ADI);
```

```
xlabel('x');
ylabel('y');
title('{\phi} for ADI (41*41)grids');

%Plotting Errors
figure
subplot(3,2,1)
contour(x,y,phi_Jacobi-phi_analytic,'ShowText','on');
xlabel('x');
ylabel('y');
title('Error for Jacobi (41*41)grids');
subplot(3,2,2)
contour(x,y,phi_Gauss-phi_analytic,'ShowText','on');
xlabel('x');
ylabel('y');
title('Error for Gauss Seidel (41*41)grids');
subplot(3,2,3)
contour(x,y,phi_SOR-phi_analytic,'ShowText','on');
xlabel('x');
ylabel('y');
title('Error for SOR (\lambda=1.2) (41*41)grids');
subplot(3,2,4)
contour(x,y,phi_RBPGS-phi_analytic,'ShowText','on');
xlabel('x');
ylabel('y');
title('Error for RBPGS (41*41)grids');
subplot(3,2,5)
x=(9*deltaX/2 : deltaX : domainX-deltaX/2);
y=(9*deltaY/2 : deltaY : domainY-deltaY/2);
[x,y]=meshgrid(x,y);
contour(x,y,phi_ADI(5:Nx,5:Ny)-phi_analytic(5:Nx,5:Ny),'ShowText','on');
xlabel('x');
ylabel('y');
title('Error for ADI (41*41)grids');

%Ploting residuals for rate of convergence
figure
semilogy(iter_Jacobi ,residual_Jacobi);
hold on
semilogy(iter_Gauss  ,residual_Gauss);
semilogy(iter_SOR    ,residual_SOR);
semilogy(iter_RBPGS  ,residual_RBPGS);
semilogy(iter_ADI    ,residual_ADI);
yline(1e-6,'-.k','Tolerance = 10^-^6');
legend('Jacobi','Gauss-Seidel','SOR for \lambda=1.2','RBPGS','ADI');
xlabel('Number of iterations');
ylabel('Residual on Log-scale');
```

```
title('Residual vs Number of iterations for 41*41 grids on semilog plot');

save('phi41.mat','phi_analytic','phi_Jacobi','phi_Gauss','phi_SOR','phi_RBPGS','phi_ADI');
save('residual41.mat','residual_Jacobi','residual_Gauss','residual_SOR','residual_RBPGS','residual_ADI');
%Solution for 41*41 Grids Ends%

%For 81*81 GRIDS
%Initialization of domain, grids, coeffients of equations and Boundary Conditions
domainX=1;
domainY=1;
Nx=81;
Ny=81;
nxp2=Nx+2;
nyp2=Ny+2;
deltaX=domainX/Nx;
deltaY=domainY/Ny;
pc = -2*(1/deltaX^2 +1/deltaY^2);
px = 1/deltaX^2;
py = 1/deltaY^2;
x=(-deltaX/2 : deltaX : domainX+deltaX/2);
y=(-deltaY/2 : deltaY : domainY+deltaY/2);
bcx=zeros(2,nyp2); bcy=zeros(nxp2,2);
bcx(1,:)=0.25.*sinh(-5)+(y-0.5).^2.*sinh(10.*(y-0.5))+1;
bcx(2,:)=0.25.*sinh(5)+(y-0.5).^2.*sinh(10.*(y-0.5))+exp(2.*y);
bcy(:,1)=0.25.*sinh(-5)+(x-0.5).^2.*sinh(10.*(x-0.5))+1;
bcy(:,2)=0.25.*sinh(5)+(x-0.5).^2.*sinh(10.*(x-0.5))+exp(2.*x);

[x,y]=meshgrid(x,y);
phi=zeros(nxp2,nyp2); phi_old=zeros(nxp2,nyp2);
Sphi=2.*sinh(10.*(x-0.5))+40.*(x-0.5).*cosh(10.*(x-0.5))+100.*((x-0.5).^2).*sinh(10.*(x-0.5))+2.*sinh(10.*(y-
0.5))+40.*(y-0.5).*cosh(10.*(y-0.5))+100.*((y-0.5).^2).*sinh(10.*(y-0.5))+4.*(x.^2+y.^2).*exp(2.*x.*y);

%Solution for 41*41 Grids%
residual_Jacobi=zeros(2,1);
iter_Jacobi=zeros(2,1);
maxError = 1e-6;
error=1;
iter=0;
%Jacobi method%
while(error>maxError && iter<100000)
    iter=iter+1;

    phi(1,2:Ny+1)   = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
    phi(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
    phi(2:Nx+1,1)   = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
    phi(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);
```

```
    for i=2:Nx+1
        for j=2:Ny+1
            phi(i,j)=(Sphi(i,j)-(px*phi_old(i-1,j)+px*phi_old(i+1,j)+py*phi_old(i,j-1)+py*phi_old(i,j+1)))/pc;
        end
    end
%     disp(iter);

    errorMatrix=(phi(2:Nx+1,2:Ny+1)-phi_old(2:Nx+1,2:Ny+1));
    error = norm(errorMatrix,2);
    residual_Jacobi(iter,1)=error;
    iter_Jacobi(iter,1)=iter;
    phi_old=phi;
%     disp(error);
end
x=(deltaX/2 : deltaX : domainX-deltaX/2);
y=(deltaY/2 : deltaY : domainY-deltaY/2);
[x,y]=meshgrid(x,y);
phi_analytic=(x-0.5).^2.*sinh(10.*(x-0.5))+(y-0.5).^2.*sinh(10.*(y-0.5))+exp(2.*x.*y);
phi_Jacobi=phi(2:Nx+1,2:Ny+1);

phi=zeros(nxp2,nyp2); phi_old=zeros(nxp2,nyp2);
residual_Gauss=zeros(2,1);
iter_Gauss=zeros(2,1);
error=1;
iter=0;
%Gauss Seidel method%
while(error>maxError && iter<100000)
    iter=iter+1;


    phi(1,2:Ny+1)   = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
    phi(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
    phi(2:Nx+1,1)   = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
    phi(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

    for i=2:Nx+1
        for j=2:Ny+1
            phi(i,j)=(Sphi(i,j)-(px*phi(i-1,j)+px*phi_old(i+1,j)+py*phi(i,j-1)+py*phi_old(i,j+1)))/pc;
        end
    end
%     disp(iter);

    errorMatrix=(phi(2:Nx+1,2:Ny+1)-phi_old(2:Nx+1,2:Ny+1));
    error = norm(errorMatrix,2);
    residual_Gauss(iter,1)=error;
    iter_Gauss(iter,1)=iter;
    phi_old=phi;
```

```
%     disp(error);
end

phi_Gauss=phi(2:Nx+1,2:Ny+1);

phi=zeros(nxp2,nyp2); phi_old=zeros(nxp2,nyp2);
residual_SOR=zeros(2,1);
iter_SOR=zeros(2,1);
lambda=1.2;
error=1;
iter=0;
%Succesive Over Relaxation method%
while(error>maxError && iter<100000)
    iter=iter+1;

    phi(1,2:Ny+1)  = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
    phi(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
    phi(2:Nx+1,1)  = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
    phi(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

    for i=2:Nx+1
        for j=2:Ny+1
            phi(i,j)=(1-lambda)*phi(i,j)+lambda*(Sphi(i,j)-(px*phi(i-1,j)+px*phi_old(i+1,j)+py*phi(i,j-
1)+py*phi_old(i,j+1)))/pc;
        end
    end
%     disp(iter);

    errorMatrix=(phi(2:Nx+1,2:Ny+1)-phi_old(2:Nx+1,2:Ny+1));
    error = norm(errorMatrix,2);
    residual_SOR(iter,1)=error;
    iter_SOR(iter,1)=iter;
    phi_old=phi;
%     disp(error);
end

phi_SOR=phi(2:Nx+1,2:Ny+1);

phi=zeros(nxp2,nyp2); phi_old=zeros(nxp2,nyp2);
residual_RBPGS=zeros(2,1);
iter_RBPGS=zeros(2,1);
error=1;
iter=0;
%RBPGS method%
while(error>maxError && iter<100000)
    iter=iter+1;
```

```
    phi(1,2:Ny+1)  = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
    phi(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
    phi(2:Nx+1,1)  = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
    phi(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

    for i=2:Nx+1
        m = 2+mod(i,2);
        for j=m:2:Ny+1
            phi(i,j) = (Sphi(i,j)-(px*phi_old(i-1,j)+px*phi_old(i+1,j)+py*phi_old(i,j-1)+py*phi_old(i,j+1)))/pc;
        end
    end

    for i=2:Nx+1
        m = 2+mod(i+1,2);
        for j=m:2:Ny+1
            phi(i,j) = (Sphi(i,j)-(px*phi(i-1,j)+px*phi(i+1,j)+py*phi(i,j-1)+py*phi(i,j+1)))/pc;
        end
    end

%     disp(iter);

    errorMatrix=(phi(2:Nx+1,2:Ny+1)-phi_old(2:Nx+1,2:Ny+1));
    error = norm(errorMatrix,2);
    residual_RBPGS(iter,1)=error;
    iter_RBPGS(iter,1)=iter;
    phi_old=phi;
%     disp(error);
end

phi_RBPGS=phi(2:Nx+1,2:Ny+1);

%ADI method%
phi=zeros(nxp2,nyp2); phi_old=zeros(nxp2,nyp2);
residual_ADI=zeros(2,1);
iter_ADI=zeros(2,1);
error=1;
iter=0;
phi_old(1,2:Ny+1)  = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
phi_old(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
phi_old(2:Nx+1,1)  = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
phi_old(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

ar=zeros(Nx,1);  ar(2:Nx,1)=-px;
br=zeros(Nx,1);  br(:,1)=-pc;
cr=zeros(Nx,1);  cr(2:Nx,1)=-px;
dr=zeros(Nx,1);
```

```
ac=zeros(Ny,1);  ac(2:Ny,1)=-py;
bc=zeros(Ny,1);  bc(:,1)=-pc;
cc=zeros(Ny,1);  cc(2:Ny,1)=-py;
dc=zeros(Ny,1);

while(error>maxError && iter<100000)
    iter=iter+1;

    phi(1,2:Ny+1)  = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
    phi(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
    phi(2:Nx+1,1)  = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
    phi(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

    for j=2:Ny+1
        dr(1,1)=py.*phi(2,j-1)+py.*phi_old(2,j+1)-Sphi(2,j)+px*phi(1,j);
        dr(2:Nx-1,1)=py.*phi(3:Nx,j-1)+py.*phi_old(3:Nx,j+1)-Sphi(3:Nx,j);
        dr(Nx,1)=py.*phi(Nx+1,j-1)+py.*phi_old(Nx+1,j+1)-Sphi(Nx+1,j)+px*phi(Nx+2,j);
        phi(2:Nx+1,j)=TDMA(ar,br,cr,dr);
    end


    phi(1,2:Ny+1)  = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
    phi(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
    phi(2:Nx+1,1)  = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
    phi(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

    for i=2:Nx+1
        dc(1,1)=px.*phi(i-1,2)+px.*phi(i+1,2)-Sphi(i,2)+py*phi(i,1);
        dc(2:Ny-1,1)=px.*phi(i-1,3:Ny)+px.*phi(i+1,3:Ny)-Sphi(i,3:Ny);
        dc(Ny,1)=px.*phi(i-1,Ny+1)+px.*phi(i+1,Ny+1)-Sphi(i,Ny+1) + py*phi(i,Ny+2);
        phi(i,2:Ny+1)=TDMA(ac,bc,cc,dc);
    end
    phi(1,2:Ny+1)  = 2*bcx(1,2:Ny+1) - phi(2,2:Ny+1);
    phi(nxp2,2:Ny+1) = 2*bcx(2,2:Ny+1) - phi(Nx+1,2:Ny+1);
    phi(2:Nx+1,1)  = 2*bcy(2:Nx+1,1) - phi(2:Nx+1,2);
    phi(2:Nx+1,nyp2) = 2*bcy(2:Nx+1,2) - phi(2:Nx+1,Ny+1);

%     disp(iter);

    errorMatrix=(phi(2:Nx+1,2:Ny+1)-phi_old(2:Nx+1,2:Ny+1));
    error = norm(errorMatrix,2);
    residual_ADI(iter,1)=error;
    iter_ADI(iter,1)=iter;
    phi_old=phi;
%     disp(error);
end
```

```
phi_ADI=phi(2:Nx+1,2:Ny+1);

x=(deltaX/2 : deltaX : domainX-deltaX/2);
y=(deltaY/2 : deltaY : domainY-deltaY/2);
[x,y]=meshgrid(x,y);

%Plots for 81*81 Grids%
%Ploting phi contours
figure
subplot(3,2,1)
contour(x,y,phi_analytic,25);
xlabel('x');
ylabel('y');
title('{\phi}_{analytical}');
subplot(3,2,2)
contour(x,y,phi_Jacobi,25);
xlabel('x');
ylabel('y');
title('{\phi} for Jacobi (81*81)grids');
subplot(3,2,3)
contour(x,y,phi_Gauss,25);
xlabel('x');
ylabel('y');
title('{\phi} for Gauss Seidel (81*81)grids');
subplot(3,2,4)
contour(x,y,phi_SOR,25);
xlabel('x');
ylabel('y');
title('{\phi} for SOR (81*81)grids');
subplot(3,2,5)
contour(x,y,phi_RBPGS,25);
xlabel('x');
ylabel('y');
title('{\phi} for RBPGS (81*81)grids');
subplot(3,2,6)
contour(x,y,phi_ADI,25);
xlabel('x');
ylabel('y');
title('{\phi} for ADI (81*81)grids');

%Ploting phi surfaces
figure
subplot(3,2,1)
surfc(x,y,phi_analytic);
xlabel('x');
ylabel('y');
title('{\phi}_{analytical}');
```

```
subplot(3,2,2)
surfc(x,y,phi_Jacobi);
xlabel('x');
ylabel('y');
title('{\phi} for Jacobi (81*81)grids');
subplot(3,2,3)
surfc(x,y,phi_Gauss);
xlabel('x');
ylabel('y');
title('{\phi} for Gauss Seidel (81*81)grids');
subplot(3,2,4)
surfc(x,y,phi_SOR);
xlabel('x');
ylabel('y');
title('{\phi} for SOR (81*81)grids');
subplot(3,2,5)
surfc(x,y,phi_RBPGS);
xlabel('x');
ylabel('y');
title('{\phi} for RBPGS (81*81)grids');
subplot(3,2,6)
surfc(x,y,phi_ADI);
xlabel('x');
ylabel('y');
title('{\phi} for ADI (81*81)grids');

%Ploting Error contours
figure
subplot(3,2,1)
contour(x,y,phi_Jacobi-phi_analytic,'ShowText','on');
xlabel('x');
ylabel('y');
title('Error for Jacobi (81*81)grids');
subplot(3,2,2)
contour(x,y,phi_Gauss-phi_analytic,'ShowText','on');
xlabel('x');
ylabel('y');
title('Error for Gauss Seidel (81*81)grids');
subplot(3,2,3)
contour(x,y,phi_SOR-phi_analytic,'ShowText','on');
xlabel('x');
ylabel('y');
title('Error for SOR (81*81)grids');
subplot(3,2,4)
contour(x,y,phi_RBPGS-phi_analytic,'ShowText','on');
xlabel('x');
ylabel('y');
```

```
title('Error for RBPGS (81*81)grids');
subplot(3,2,5)
x=(13*deltaX/2 : deltaX : domainX-deltaX/2);
y=(13*deltaY/2 : deltaY : domainY-deltaY/2);
[x,y]=meshgrid(x,y);
contour(x,y,phi_ADI(7:Nx,7:Ny)-phi_analytic(7:Nx,7:Ny),'ShowText','on');
xlabel('x');
ylabel('y');
title('Error for ADI (81*81)grids');

%Ploting residuals
figure
semilogy(iter_Jacobi ,residual_Jacobi);
hold on
semilogy(iter_Gauss  ,residual_Gauss);
semilogy(iter_SOR    ,residual_SOR);
semilogy(iter_RBPGS  ,residual_RBPGS);
semilogy(iter_ADI    ,residual_ADI);
yline(1e-6,'-.k','Tolerance = 10^-^6');
legend('Jacobi','Gauss-Seidel','SOR for \lambda=1.2','RBPGS','ADI');
xlabel('Number of iterations');
ylabel('Residual on Log-scale');
title('Residual vs Number of iterations for (81*81 grids) on semilog plot');

save('phi81.mat','phi_analytic','phi_Jacobi','phi_Gauss','phi_SOR','phi_RBPGS','phi_ADI');
save('residual81.mat','residual_Jacobi','residual_Gauss','residual_SOR','residual_RBPGS','residual_ADI');

%Solutions for (81*81) Grids ends%


end
```

Algorithm for TDMA

```
%Rikesh Sharma
%180606
%ME630A (CFD) | < submission date: 22/09/2021 >

function x = TDMA(a,b,c,d)
%a, b, c are the column vectors for the compressed tridiagonal matrix, d is the right vector
n = length(b); % n is the number of rows

% Modify the first-row coefficients
c(1) = c(1) / b(1);    % Division by zero risk.
```

```
d(1) = d(1) / b(1);     % Division by zero would imply a singular matrix.

for i = 2:n-1
    mult = b(i) - a(i) * c(i-1);
    c(i) = c(i) / mult;
    d(i) = (d(i) - a(i) * d(i-1)) / mult;
end

d(n) = (d(n) - a(n) * d(n-1))/( b(n) - a(n) * c(n-1));

% Now back substitute.
x(n) = d(n);
for i = n-1:-1:1
    x(i) = d(i) - c(i) * x(i + 1);
end
end
```