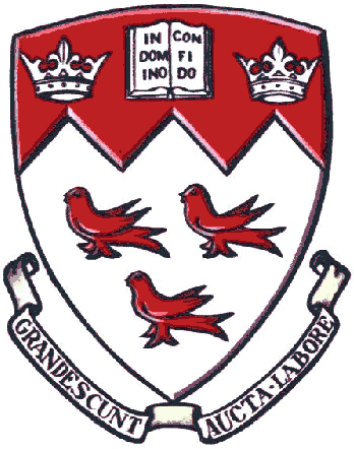
MECH 539: Computational Aerodynamics

Project #2 (Corrected Version)



Justin ChanWoo Yang
260368098
chanwoo.yang@mail.mcgill.ca

March 12th, 2013

Jacobi Method Algorithm (Corrected)

```
function [ u, Residual, numberOfIteration, time ] = Jacobi( n )
%INPUT
%n: nxn matrix

%OUTPUT
%u: Solution array
%Residual: Returns residual at each iteration
%numberOfIteration: number of iteration to get valid solutions
%time: CPU times elapsed at each iteration
%conditionNumber: Condition Number of a matrix

Tstart = tic;

%Declare the parameters
Tolerance = eps('single');
numberOfIteration = 0;
maxIteration = 10^5;
% EPS = 1.1921e-07;   %Obtained by typeing eps('single') on command window

%Initialize matrix of u with rough geuss

u = zeros(n,n);
residual = zeros(n,n);

%Initilize boundary conditions
for i=1:n
    u(i,n)=1;
end

%Do the rough guess
for i=2:(n-1)
    for j=2:(n-1)
        u(i,j)=0.25;
    end
end

uPrevious = u;

while (numberOfIteration < maxIteration)


    for i=2:(n-1)
        for j=2:(n-1)
            u(i,j)=(uPrevious(i+1,j)+uPrevious(i-1,j)+uPrevious(i,j+1)+uPrevious(i,j-1))/4;
        end
    end

    if(norm(abs(u-uPrevious))<Tolerance)
        break;
    end

    end

    numberOfIteration = numberOfIteration + 1;
    Residual(numberOfIteration+1) = norm(residual);
    time(numberOfIteration+1) = toc(Tstart);

    if(norm(abs(u-uPrevious))<Tolerance)
        break;
    end

    uPrevious = u;

end

% conditionNumber = (norm(u-uInitial)/norm(uInitial))*1/EPS;

end
```

Gauss-Seidel Method Algorithm (Corrected)

```
function [ u, Residual, numberOfIteration, time ] = GaussSeidel( n )
%INPUT
%n: nxn matrix

%OUTPUT
%u: Solution array
%Residual: Returns residual at each iteration
%numberOfIteration: number of iteration to get valid solutions
%time: CPU times elapsed at each iteration
%conditionNumber: Condition Number of a matrix

Tstart = tic;

%Declare the parameters
Tolerance = eps('single');
numberOfIteration = 0;
maxIteration = 10^5;
% EPS = 1.1921e-07;   %Obtained by typeing eps('single') on command window

%Initialize matrix of u with rough geuss (u=0)
u = zeros(n,n);
residual = zeros(n,n);

%Initilize boundary conditions
for i=1:n
    u(i,n)=1;
end

%Do the rough guess
for i=2:(n-1)
    for j=2:(n-1)
        u(i,j)=0.25;
    end
end

% uInitial = u;

while (numberOfIteration < maxIteration)

    for i=2:(n-1)
        for j=2:(n-1)
            u(i,j)=(u(i-1,j)+u(i,j-1)+u(i+1,j)+u(i,j+1))/4;
        end
    end

    for i=2:(n-1)
        for j=2:(n-1)
            residual(i,j) = abs(0 - (u(i+1,j)+u(i-1,j)+u(i,j+1)+u(i,j-1)-4*u(i,j)));
        end
    end

    numberOfIteration = numberOfIteration + 1;
    Residual(numberOfIteration+1) = norm(residual);
    time(numberOfIteration+1) = toc(Tstart);

    if(Residual(numberOfIteration+1)<Tolerance)
        break;
    end

end

% conditionNumber = (norm(abs(u-uInitial))/norm(uInitial))*1/EPS;

end
```

SOR Method Algorithm (Corrected)

```
function [ u, Residual, numberOfIteration, time ] = SOR( n, w )
%INPUT
%n: nxn matrix
%w: Relaxation parameter

%OUTPUT
%u: Solution array
%Residual: Returns residual at each iteration
%numberOfIteration: number of iteration to get valid solutions
%time: CPU times elapsed at each iteration
%conditionNumber: Condition Number of a matrix

Tstart = tic;

%Declare the parameters
Tolerance = eps('single');
numberOfIteration = 0;
maxIteration = 10^6;
% EPS = 1.1921e-07;   %Obtained by typeing eps('single') on command window

%Initialize matrix of u with rough geuss (u=0)
u = zeros(n,n);
uGS = zeros(n,n);

for i=1:n
    u(i,n)=1;           %Initilize boundary conditions
end

%Do the rough guess
for i=2:(n-1)
    for j=2:(n-1)
        u(i,j)=0.25;
    end
end

% uInitial = uPrevious;

uPrevious = u;

while (numberOfIteration < maxIteration)



    for i=2:(n-1)
        for j=2:(n-1)
            uGS(i,j)=(u(i-1,j)+u(i,j-1)+uPrevious(i+1,j)+uPrevious(i,j+1))/4;
            u(i,j)=(1-w)*u(i,j)+w*uGS(i,j);
        end
    end

    for i=2:(n-1)
        for j=2:(n-1)
            residual(i,j) = abs(0 - (u(i+1,j)+u(i-1,j)+u(i,j+1)+u(i,j-1)-4*u(i,j)));
        end
    end

    numberOfIteration = numberOfIteration + 1;
    Residual(numberOfIteration+1) = norm(residual);
    time(numberOfIteration+1) = toc(Tstart);

    if(Residual(numberOfIteration+1)<Tolerance)
        break;
    end

    uPrevious = u;

end

% conditionNumber = (norm(u-uInitial)/norm(uInitial))*1/EPS;

end
```

Using three different methods to find solution at three different grid sizes, the parameters (initial guess, tolerance, and relaxation parameter) were set as below, and following results were obtained

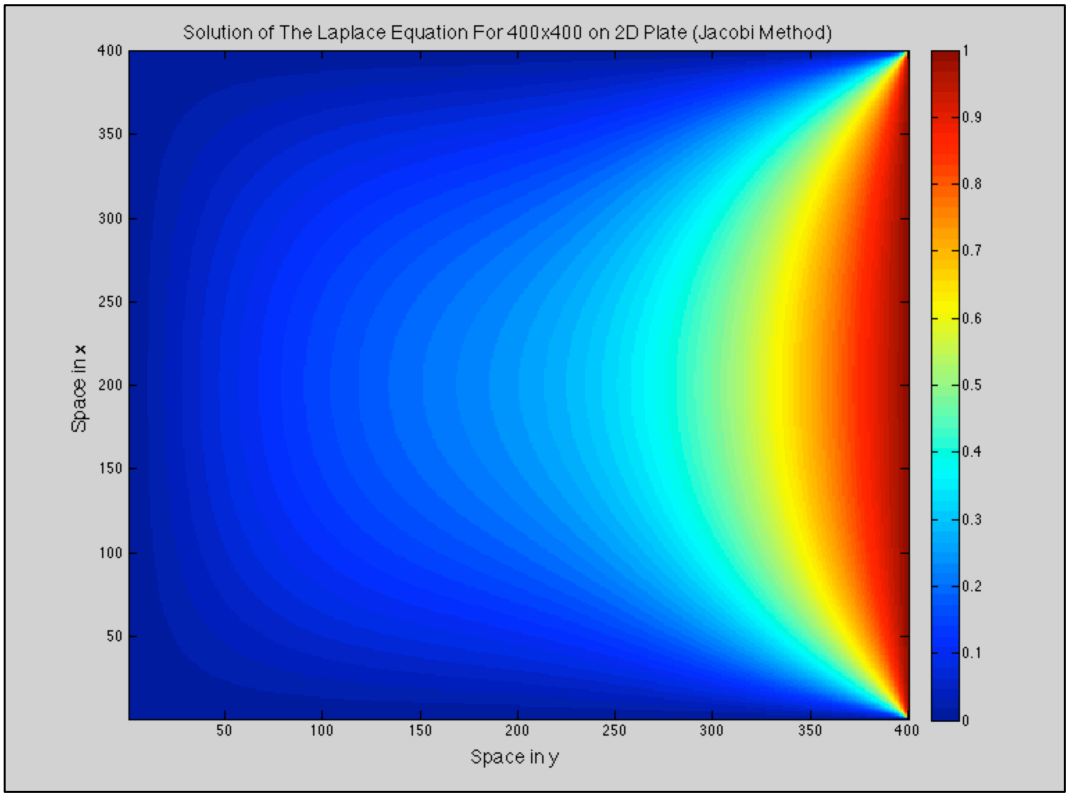
Method	Initial Guess	Tolerance	Relaxation Parameter	Grid Size	Iteration	CPU Time	Condition Number
Jacobi	0.25	0.001		100x100	2387	31.4700	7.4594E+06
	0.25	0.001		200x200	7414	435.7700	7.6336E+06
	0.25	0.001		400x400	20932	5114.4700	7.4904E+06
GaussSeidel	0.25	0.001		100x100	1480	21.3600	7.7436E+06
	0.25	0.001		200x200	4835	328.2400	8.0181E+06
	0.25	0.001		400x400	14919	4173.3000	8.0394E+06
SOR	0.25	0.001	1.5	100x100	1094	16.3900	7.5708E+06
	0.25	0.001	1.5	200x200	3657	253.5300	7.8548E+06
	0.25	0.001	1.5	400x400	11686	3371.9200	7.9232E+06

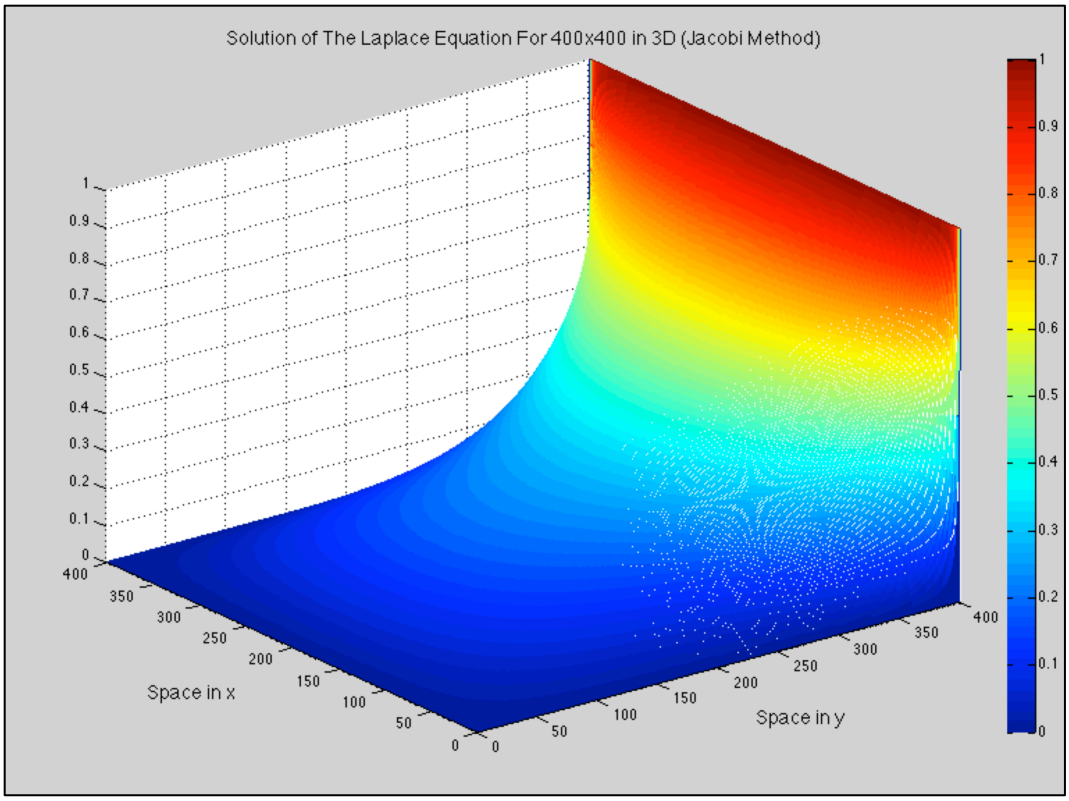
(Corrected Version: Red values are corrected data)

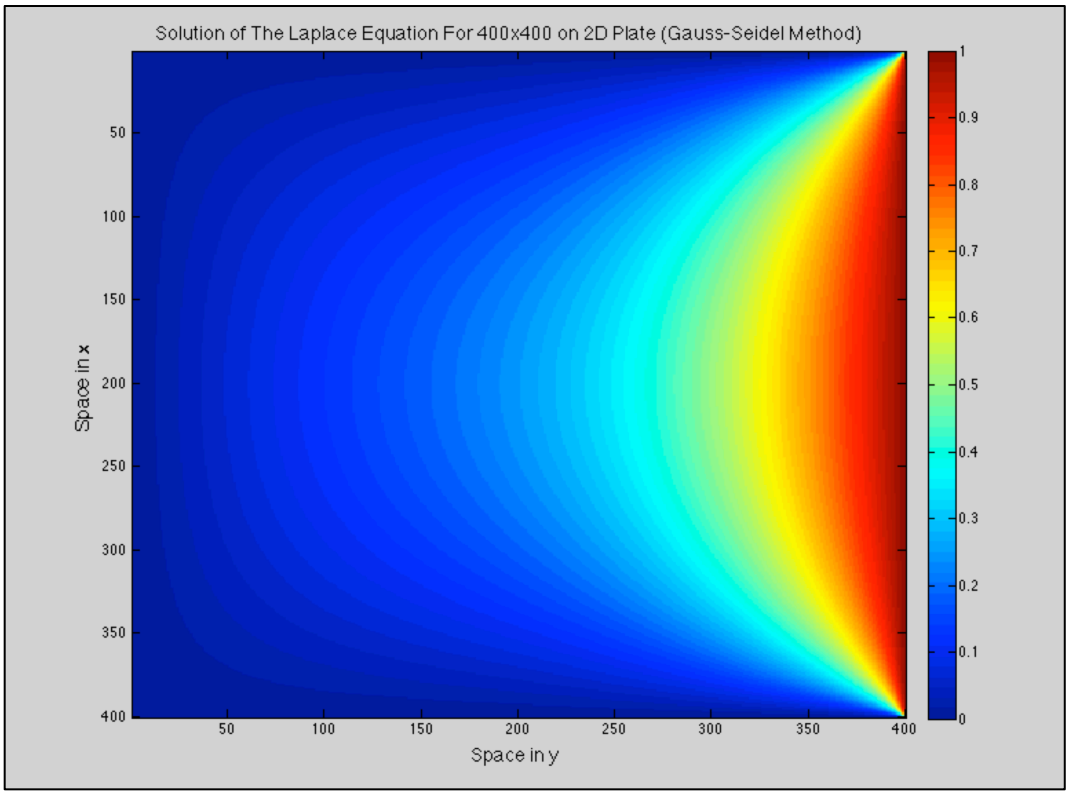
Method	Initial Guess	Tolerance	Relaxation Parameter	Grid Size	Iteration	CPU Time	Condition Number
Jacobi	0	eps('single')		100x100	22534	82.7455	7.4594E+06
	0.25	eps('single')		200x200	7414	435.7700	7.6336E+06
	0.25	eps('single')		400x400	20932	5114.4700	7.4904E+06
GaussSeidel	0	eps('single')		100x100	12668	35.8686	7.7436E+06
	0.25	eps('single')		200x200	4835	328.2400	8.0181E+06
	0.25	eps('single')		400x400	14919	4173.3000	8.0394E+06
SOR	0	eps('single')	1.9	100x100	642	2.4086	7.5708E+06
	0.25	eps('single')	1.5	200x200	3657	253.5300	7.8548E+06
	0.25	eps('single')	1.9	400x400	7786	577.8166	7.9232E+06

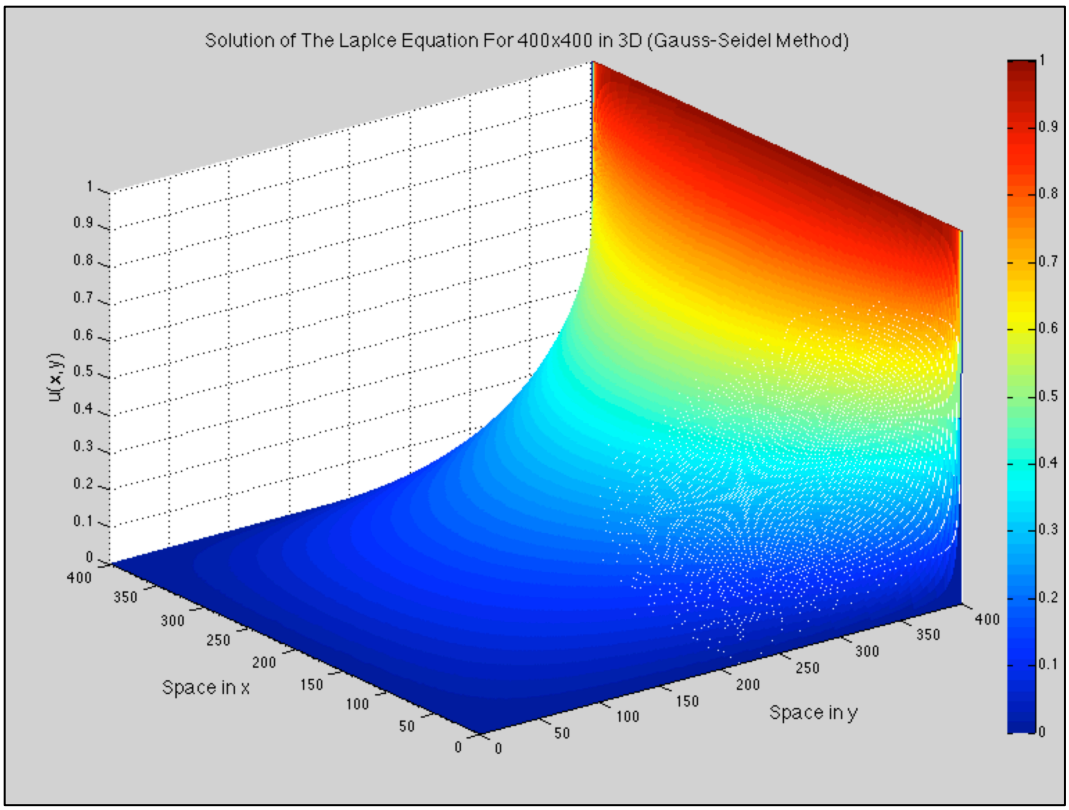
** The convergence rate of Gauss-Seidel and of SOR has increased.

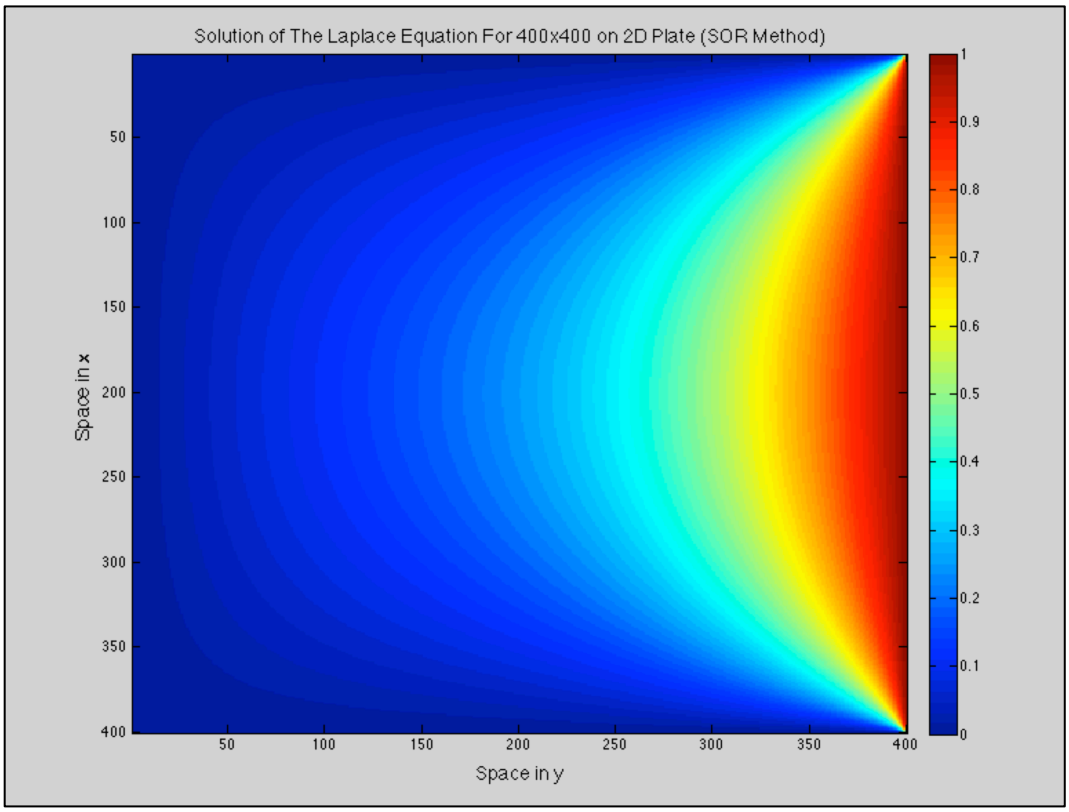
Q1.

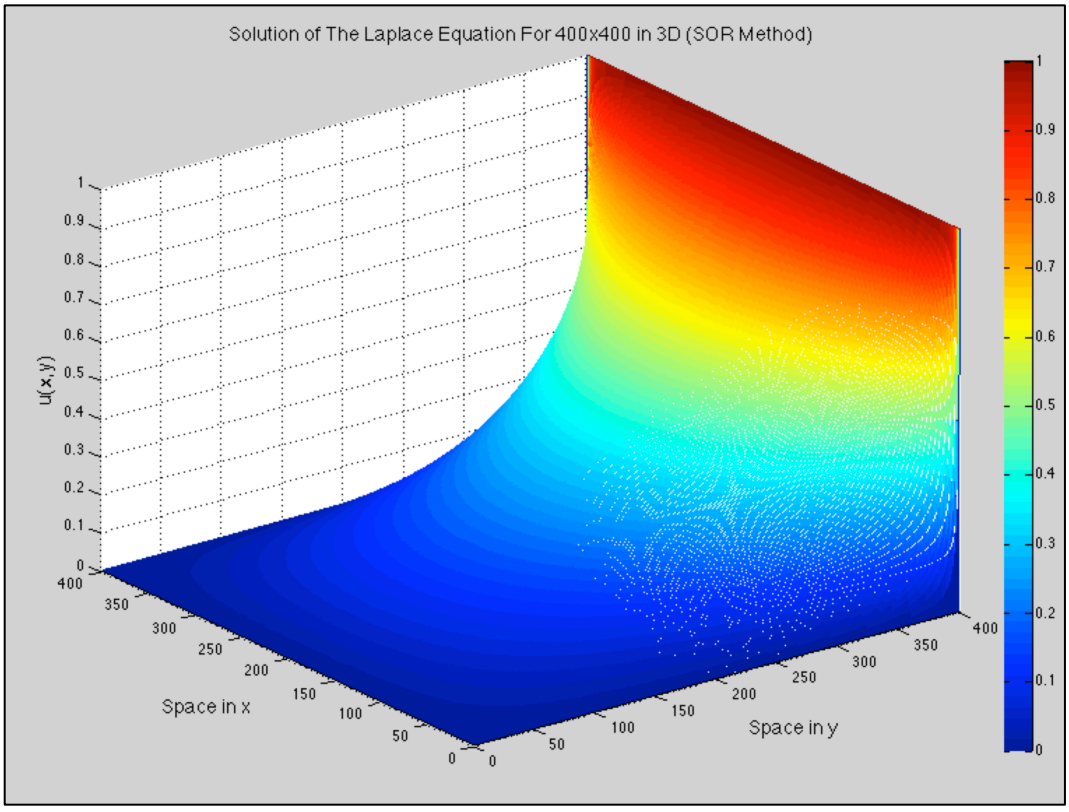

Solution of The Laplace Equation For 400x400 on 2D Plate (Jacobi Method)
Space in x
Space in y
400
350
300
250
200
150
100
50
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


Solution of The Laplace Equation For 400x400 in 3D (Jacobi Method)
Space in x
Space in y
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
400
350
300
250
200
150
100
50
0


Solution of The Laplace Equation For 400x400 on 2D Plate (Gauss-Seidel Method)
Space in x
Space in y
50
100
150
200
250
300
350
400
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

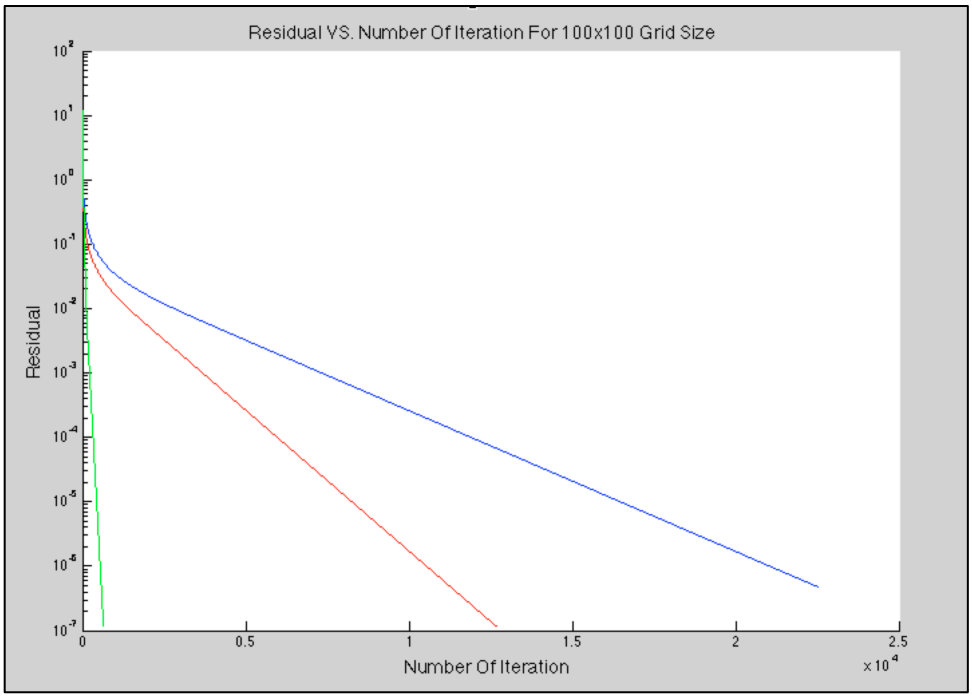

Solution of The Laplce Equation For 400x400 in 3D (Gauss-Seidel Method)
u(x,y)
Space in x
Space in y
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
50
100
150
200
250
300
350
400


Solution of The Laplace Equation For 400x400 on 2D Plate (SOR Method)
Space in x
Space in y
50
100
150
200
250
300
350
400
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Solution of The Laplace Equation For 400x400 in 3D (SOR Method)
u(x,y)
Space in x
Space in y
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
50
100
150
200
250
300
350
400

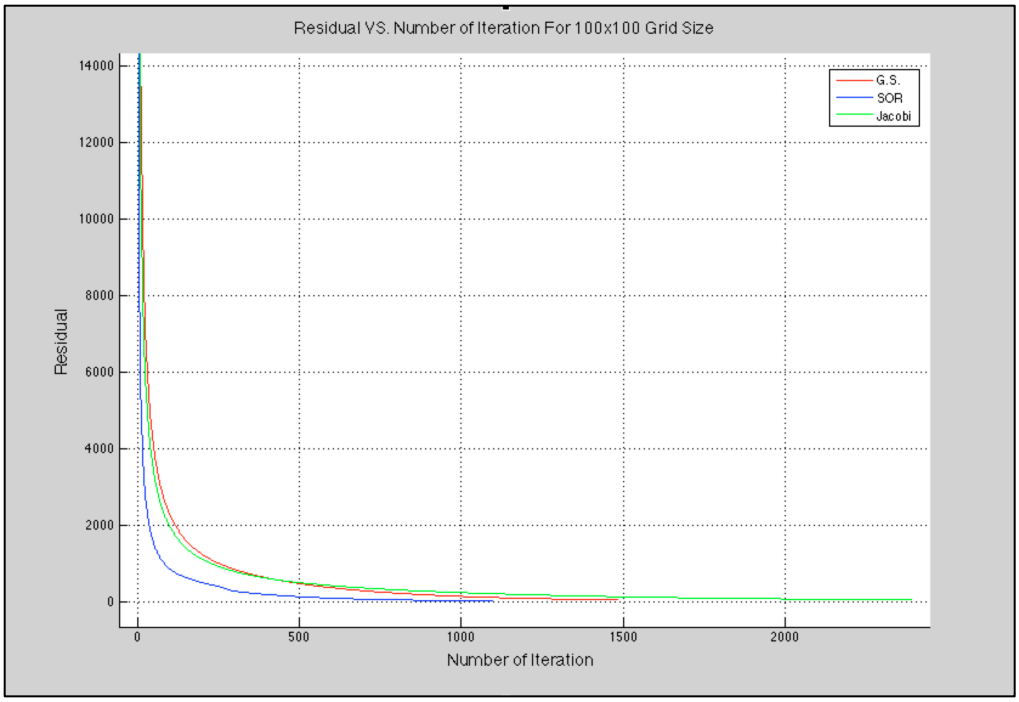
Q2.

(For corrected version, tolerance was set to eps('single'), which is the single machine precision)

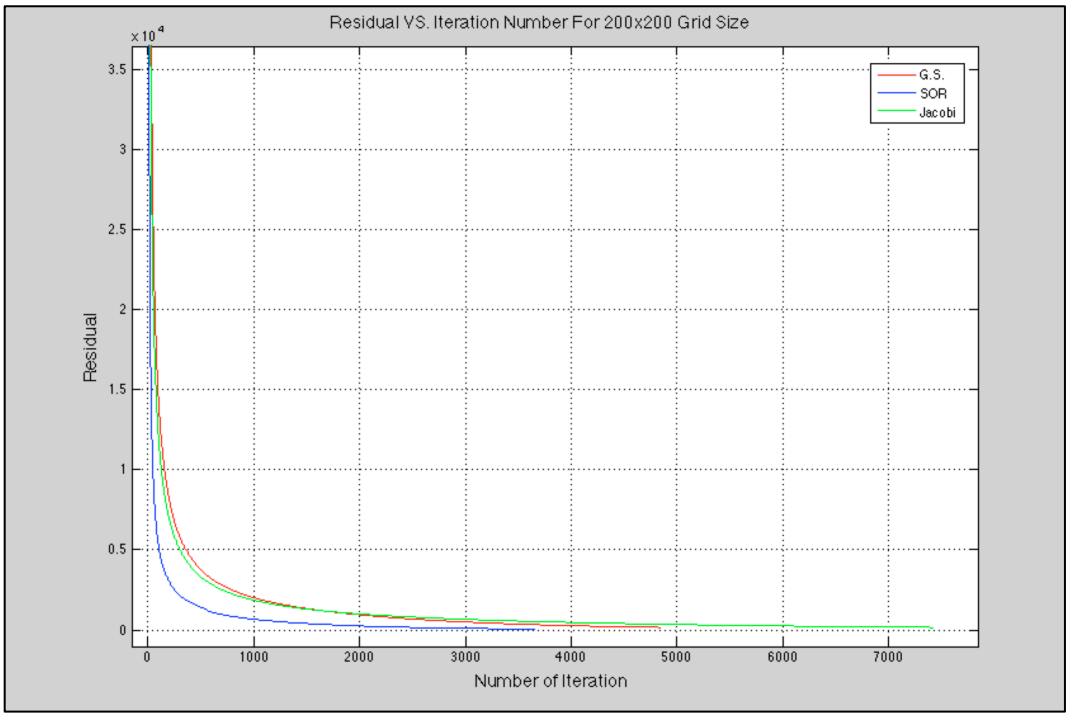


----------Old Version----------

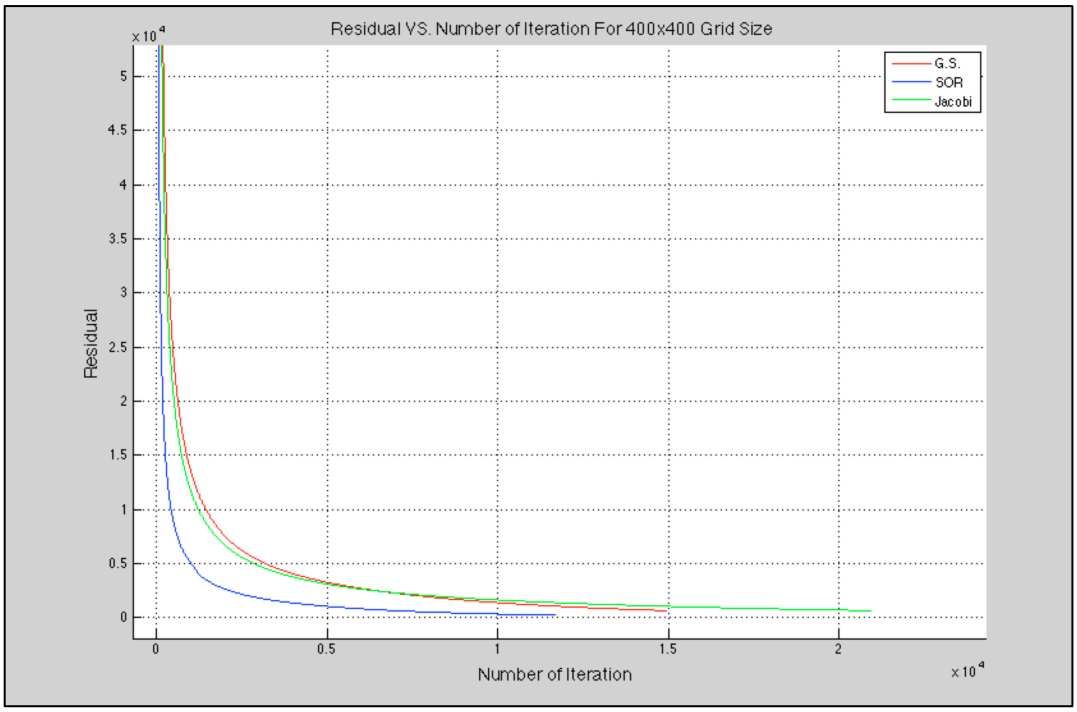
<u>100 x 100 Grid Size</u>



200 x 200 Grid Size



400 x 400 Grid Size



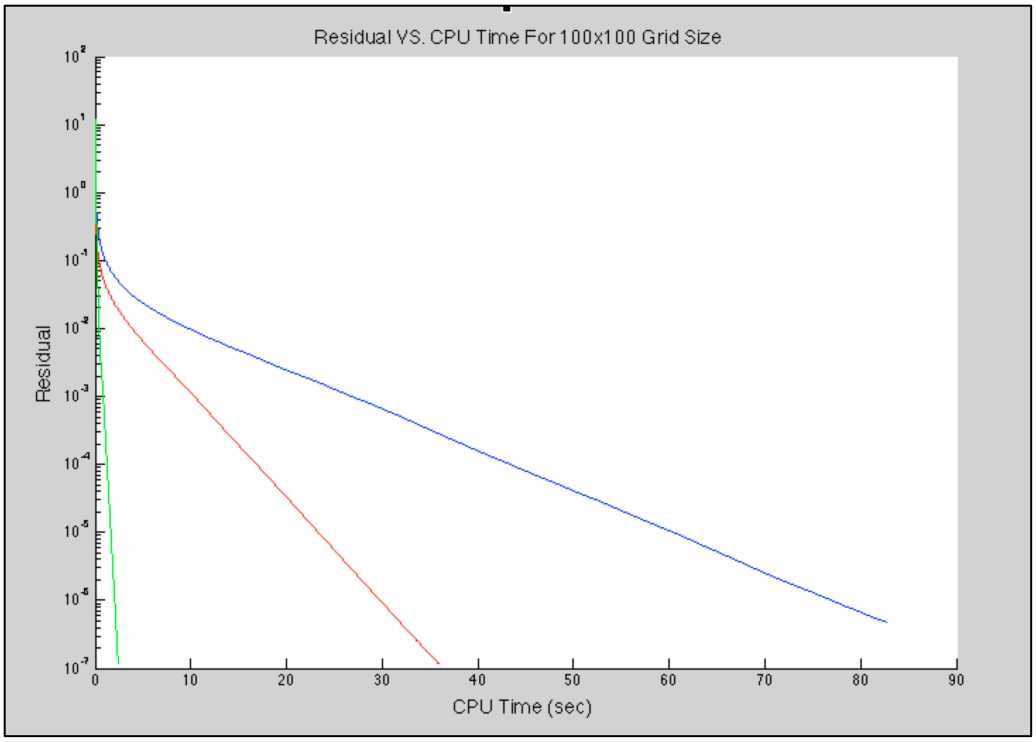
First of all, all three schemes iterated a lot more as the grid resolution becomes finer. And, SOR method resulted the fastest convergence of the residual no matter what the grid size was. And, the convergence rate of Gauss-Seidel was about twice faster than that of Jacobi and iteration of Gauss-Seidel method ended much earlier. These phenomena were the same for all grid sizes.

Condition Number computed by Forsythe-Moler Method

Method	Grid Size	Condition Number
Jacobi	100x100	7.4594E+06
	200x200	7.6336E+06
	400x400	7.4904E+06
Gauss-Seidel	100x100	7.7436E+06
	200x200	8.0181E+06
	400x400	8.0394E+06
SOR	100x100	7.5708E+06
	200x200	7.8548E+06
	400x400	7.9232E+06

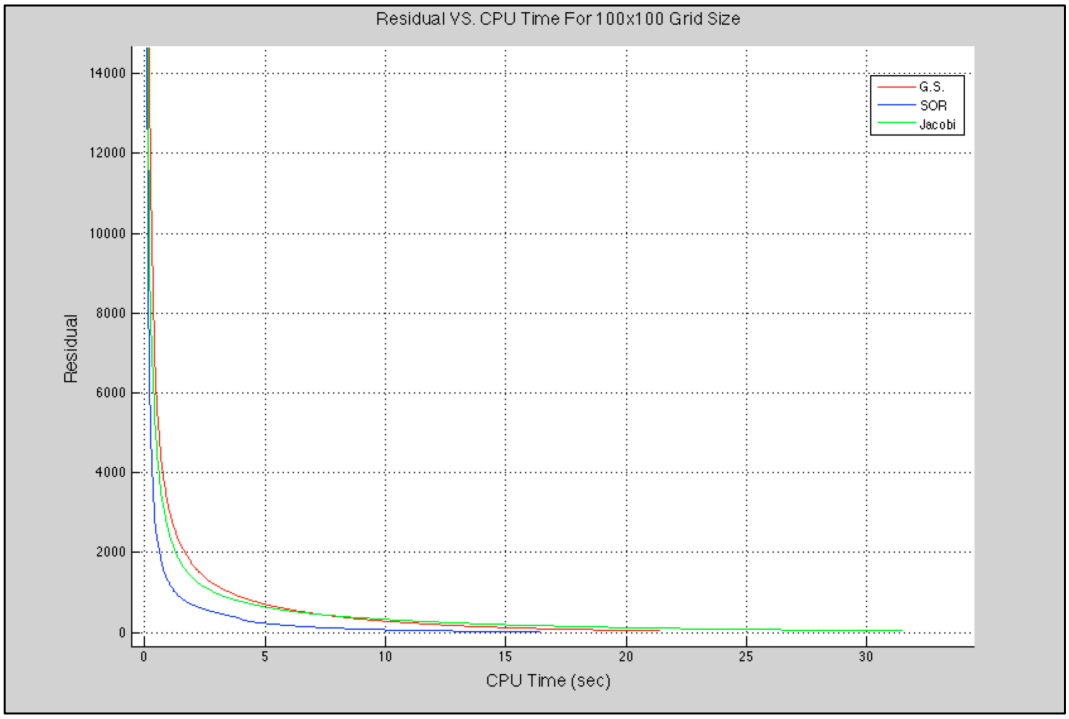
According to the result, as the grid size increases, the condition numbers of all three methods tended to increase also. And, all computed condition numbers were very high, so all the matrices are ill conditioned. In other words, these matrices were not possible to do inverse by direct method, so they needed to be computed by stationary iterative methods.

Q3.
(For corrected version, tolerance was set to eps('single'), which is the single machine precision)

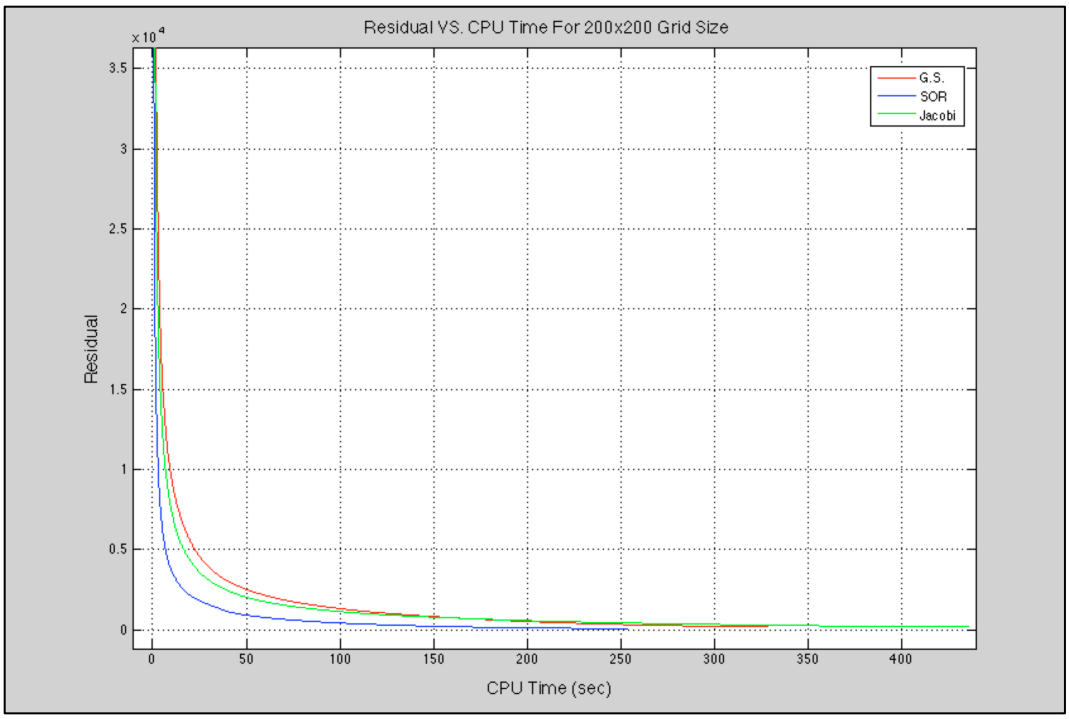


----------Old Version----------

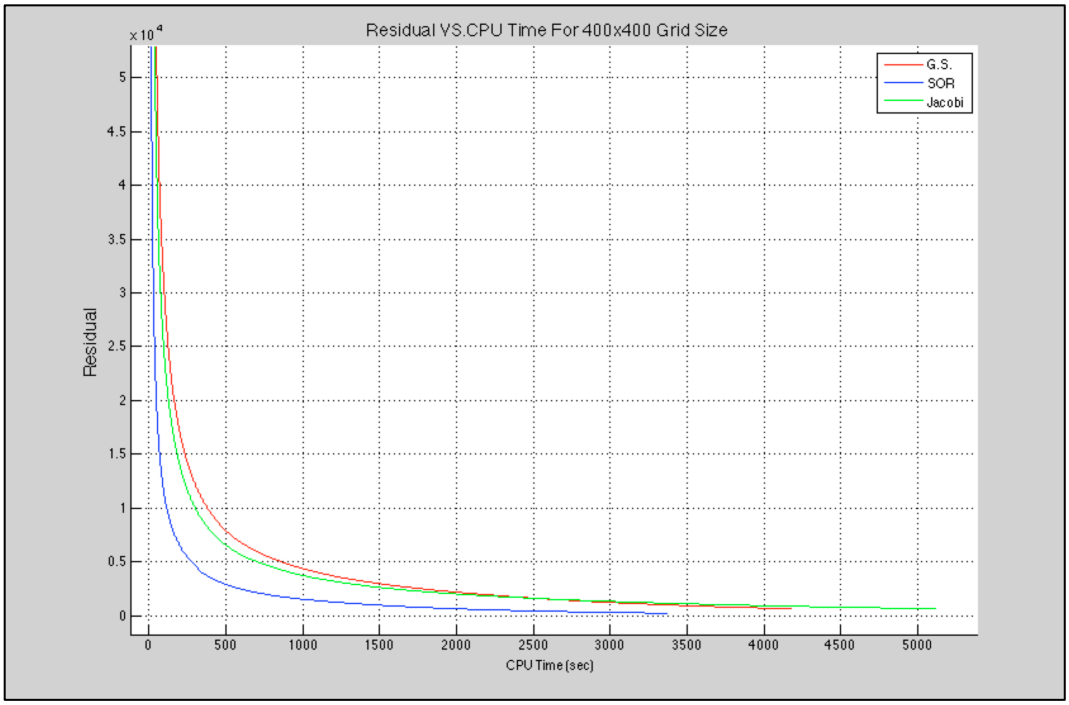
100 x 100 Grid Size



200 x 200 Grid Size



400 x 400 Grid Size



(Edited)

Like 'Residual VS. Number of Iteration' graphs, all three schemes took a lot more time as the grid resolution becomes finer and the SOR method resulted the fastest convergence rate of residual. Also, Gauss-Seidel was about twice faster and its residual converged much earlier than Jacobi.

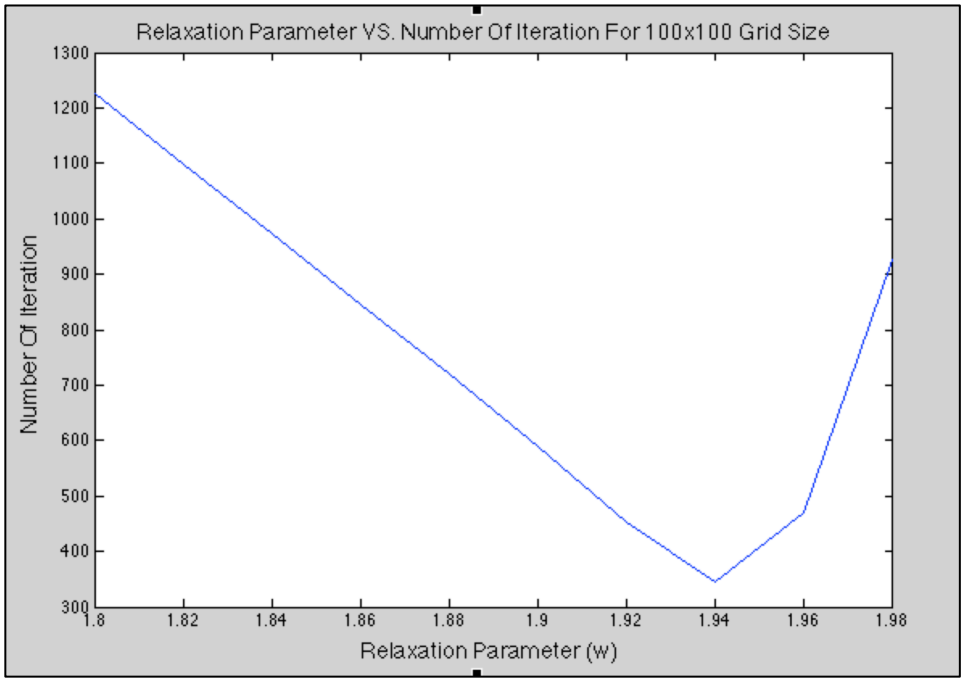
The Jacobi method needs two N by N matrices: one for storing computed solution at previous iteration and the other for storing current computed solution using solution from previous iterations. But, Jacobi method computes the current solution only with the solution from previous iteration.
Gauss-Seidel method requires only single N by N matrix. During its iteration, newly computed result can be overwritten on the same vector so that new solution can be ready to use as soon as it is produced. So, Jacobi requires twice more memory and takes more time. However, Gauss-Seidel uses less memory, but more information is used than Jacobi does, so Gauss-Seidel iterates more efficiently.
On the other hand, SOR method requires three N by N matrices: one for storing solution from previous iteration, another for storing currently computed solution, and the other for current Gauss-Seidel solution. So, SOR method can access a lot more information at each iteration than other two methods, so it would converge to solution at much faster rate at much less iteration numbers. But, this method requires more memory space to store a lot more information.

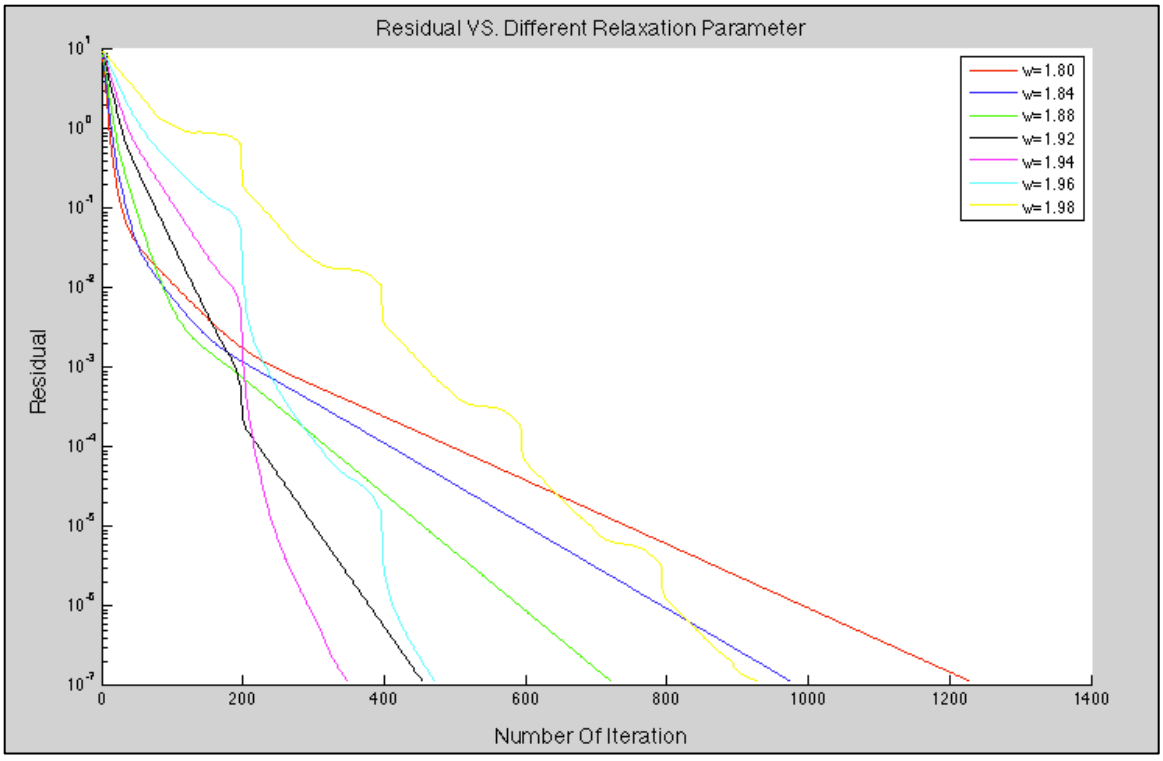
Q4.

(Corrected)
The effect of relaxation parameter on the SOR method was analyzed at two different grid sizes and following result was obtained.

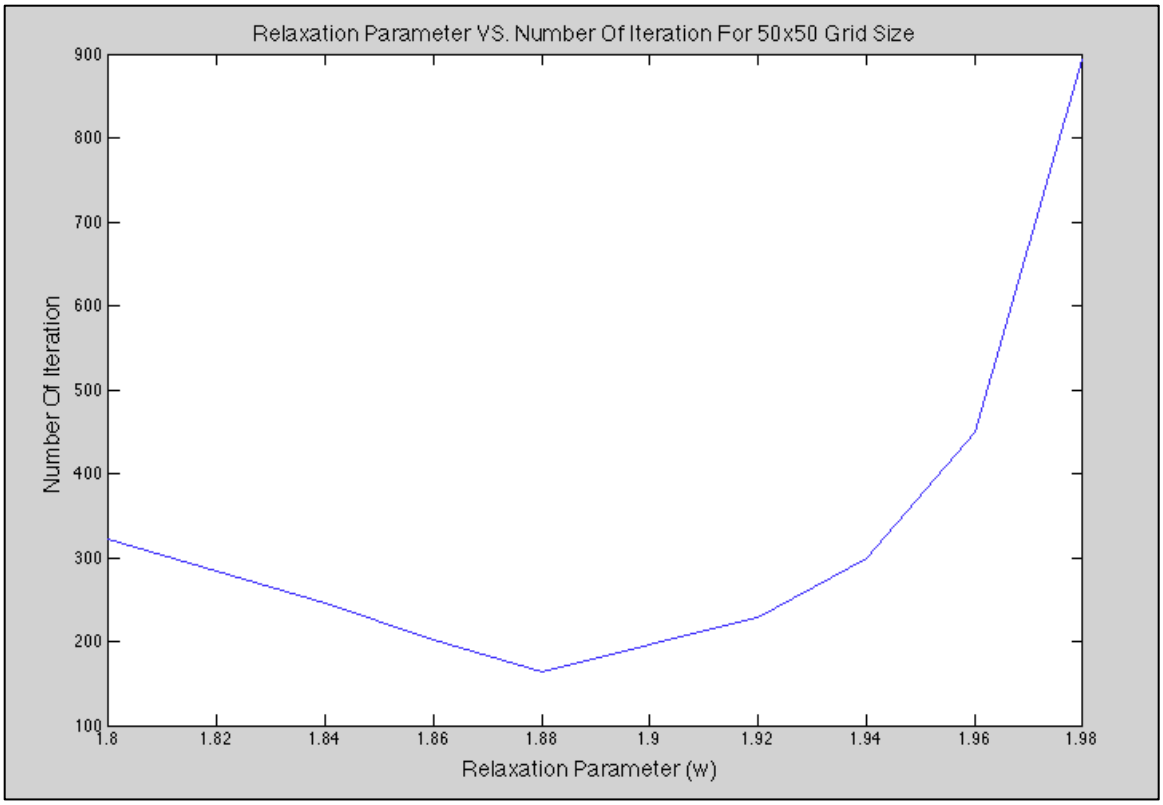
100x100			50x50		
w	Iteration	CPU Time	w	Iteration	CPU Time
1.8	1225	4.613	1.8	322	0.2776
1.82	1098	4.0551	1.82	284	0.2442
1.84	972	3.7013	1.84	245	0.2098
1.86	846	3.231	1.86	202	0.1738
1.88	719	2.7646	1.88	163	0.1427
1.9	589	2.1108	1.9	196	0.1689
1.92	452	1.8115	1.92	229	0.1974
1.94	346	1.2586	1.94	300	0.2596
1.96	469	1.739	1.96	449	0.3904
1.98	925	1.2586	1.98	893	0.7818

100 x 100 Grid Size





50 x 50 Grid Size



According to the result of 100x100 grid size, the increase in relaxation parameter made the SOR method to converge faster up to the value of 1.94, but after that point the convergence rate decreased. Also for the result of 50x50 grid size, the increase in relaxation parameter up to value of 1.88 resulted the faster convergence rate, but after that point, the SOR method converged at much larger iteration number and convergence rate decreased quite significantly. Therefore, there is an optimal relaxation parameter and the optimal relaxation parameter is different from the grid sizes.