MATH 6644
Homework 1

Stephan Boettcher

February 22, 2015

Question 1

Construct a dense 3×3 matrix G with ρ(G)>1. Define an iteration and find an example of $\vec{x}_0$ and $\vec{c}$ such that the iteration does not converge.

Beginning with the dense A matrix of: $A = \begin{bmatrix} 1 & 2 & 1 \\ 4 & 9 & 2 \\ 7 & 20 & 2 \end{bmatrix}$, we generate the G matrix by first constructing the N and M matrices. The M matrix is defined as the diagonal elements of A and the N matrix is defined as: N=M-A, or the off diagonal elements multiplied by -1. The G matrix is then constructed by $G = M^{-}1 * N$, which creates the following matrix: $G = \begin{bmatrix} 0 & -2 & -1 \\ -\frac{4}{9} & 0 & -\frac{2}{9} \\ \frac{7}{2} & -10 & 0 \end{bmatrix}$. The spectral radius of the G matrix is 2.9413. This spectral radius makes for an unstable iteration matrix that will most likely result in the iterative method going to infinity, rather than converging on a value. An iteration for this matrix is defined as:

$$\vec{x}_{k+1} = G\vec{x}_k + \vec{c}$$

Where $\vec{c}$ is defined as: $\vec{c} = M^{-1}\vec{b}$. The $\vec{b}$ is the final vector we are iterating to. To find an $\vec{x}_0$,$\vec{c}_0$, the vector $\vec{b}$ must be defined. For this assignment, the following values were chosen for these vectors:

$$\vec{b} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}, \vec{x}_0 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \vec{c}_0 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

With these values, combined with the unstable G matrix, the iterative method quickly spirals out of control and goes to infinity within 100 iterations.

Question 2

Give the matrix expression for the symmetric Gauss-Seidel iterations.

The Gauss-Seidel (G-S) method can be expressed as both an element-wise expression and in matrix expressions. To express the G-S method in matrix expression form, the notation must first be developed. Given an n×n nonsingular matrix A, $\vec{x} \in \mathbb{R}^n$, and $\vec{b} \in \mathbb{R}^n$, we can split the A matrix into:

$$A = D - E - F$$

where D is the diagonal elements of A, E is the lower triangular matrix of A, and F is the upper triangular matrix of A. With these matrices defined, the symmetric G-S iteration can be expressed in matrix form.

The symmetric G-S requires two sweeps: a forward sweep followed by a backward sweep. This is done to increase the rate of convergence of the G-S method. The forward sweep portion of the iteration is defined as:

$$\vec{x}^{k+\frac{1}{2}} = (D-E)^{-1}F\vec{x}^{\,(k)} + (D-F)^{-1}\vec{b}$$

The forward sweep is then followed by the backward sweep which is defined as:

$$\vec{x}^{k+1} = (D-F)^{-1}E\vec{x}^{\,(k+\frac{1}{2})} + (D-F)^{-1}\vec{b}$$

Thus, the matrix expression for the symmetric G-S iterations is:

$$\vec{x}^{k+\frac{1}{2}} = (D-E)^{-1}F\vec{x}^{\,(k)} + (D-F)^{-1}\vec{b}$$
$$\vec{x}^{k+1} = (D-F)^{-1}E\vec{x}^{\,(k+\frac{1}{2})} + (D-F)^{-1}\vec{b}$$

Question 3

Assume A is a n×n symmetric positive definite matrix with all eigenvalues in the intervals (1; 1.1) ∪ (10; 10.2). Assume that the cost of a matrix vector multiplication is about 4n floating point multiplications. Estimate the number of floating point operations reduce the A-norm of the error by a factor of 10^{-3} *using CG iterations.*

Because the n×n A matrix is symmetric positive definite, convergence can be guaranteed for the Conjugate Gradient(CG) method. This allows an upper limit on the number of floating point operations (FLOPS) to be calculated. This upper bound will be dependent on the theoretical convergence rate of the CG method. The analysis of the CG method assumes that variable assignment and vector addition/subtraction can occur in $\mathcal{O}(1)$ or constant time. This is theoretically possible on vector machines. The analysis also assumes that 'vector devision' and vector multiplication (i.e. the computation of the α and β values) can be completed in $\mathcal{O}$(n). The breakdown of the FLOPS cost per step of the CG method is below:

$$
\begin{aligned}
\vec{r}^{\,(0)} &= \vec{b} - A\vec{x} && 4n\ FLOPS \\
\vec{p}(0) &= \vec{r}^{\,(0)} && 1\ FLOPS \\
for\ i &= 1 : k && k\ iterations \\
&\{ \\
\vec{q}^{\,(k-1)} &= A\vec{p}^{\,(k-1)} && 4n\ FLOPS \\
\alpha_{k-1} &= \frac{<\vec{r}^{\,(k-1)}, \vec{r}^{\,(k-1)}>}{<\vec{p}^{\,(k-1)}, \vec{q}>} && 3n\ FLOPS\ total \\
\vec{x}^{\,(k)} &= \vec{x}^{\,(k-1)} + \alpha_{k-1}\vec{p}^{\,(k-1)} && n\ FLOPS \\
\vec{r}^{\,(k)} &= \vec{r}^{\,(k-1)} - \alpha_{k-1}\vec{q} && n\ FLOPS \\
\beta_{k-1} &= \frac{<\vec{r}^{\,(k)}, \vec{r}^{\,(k)}>}{<\vec{r}^{\,(k-1)}, \vec{r}^{\,(k-1)}>} && 3n\ FLOPS\ total \\
\vec{p}^{\,(k)} &= \vec{r}^{\,(k)} + \beta_{k-1}\vec{p}^{\,(k-1)} && 4n\ FLOPS \\
&\}
\end{aligned}
$$

Thus it takes $\mathcal{O}$(5n) to setup the problem and $\mathcal{O}$(16n) FLOPS to complete one iteration. The error equation is given by:

$$||\vec{e}^{\,(k+1)}||_A \leq \left(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right)||\vec{e}^{\,(k)}||_A$$

For this problem, $\kappa(A) = \frac{10.2}{1.1} = 9.\overline{2727}$. By the error equation above, for each iteration, the error will be reduced by $\left(\frac{\sqrt{9.\overline{2727}}-1}{\sqrt{9.\overline{2727}}+1}\right) \approx 0.5056$. As a result, after 10 iterations, the error will be reduced by $\approx 10^{-3}$. Thus, the number of FLOPS to reduce the A-norm of the error by a factor of 10^{-3} is $\mathcal{O}$(160n).

Question 4

Assume that A is a n×n symmetric positive definite matrix. If κ(A) =$\mathcal{O}$(n), give a rough estimate of the number of CG iterations required to reduce the relative residual to $\mathcal{O}(\frac{1}{n})$.

The residual matrix is defined as $\vec{r}^{\,(k)} = \vec{b} - A\vec{x}^{\,(k)}$. The error vector can be written as $A\vec{e}^{\,(k)} = A(\vec{x}^{\,(*)} - \vec{x}^{\,(k)}) = \vec{b} - A\vec{x}^{\,(k)}$. Thus, $\vec{r}^{\,(k)} = A\vec{e}^{\,(k)}$. Since κ(A) =$\mathcal{O}$(n), the error equation can be written as:

$$||A^{-1}\vec{r}^{\,(k+1)}||_A \leq \left(\frac{\sqrt{\mathcal{O}(n)}-1}{\sqrt{\mathcal{O}(n)}+1}\right)||A^{-1}\vec{r}^{\,(k)}||_A$$

To determine the number of iterations necessary to reduce the error to a given value, the following equation must be solved:

$$\left(\frac{\sqrt{\mathcal{O}(n)}-1}{\sqrt{\mathcal{O}(n)}+1}\right)^x = \mathcal{O}\left(\frac{1}{n}\right)$$

where x is the number of iterations. Solving for x, we get:

$$x = \frac{-log(\mathcal{O}(n))}{log(\sqrt{\mathcal{O}(n)} - 1) - log(\sqrt{\mathcal{O}(n)} + 1)}$$

To provide a quicker, rougher estimate, the above equation follows the equation $x' = \frac{3*2^{log(n)}}{4}$ fairly closely for multiple values of n. While the values provided by x' do not exactly match that generated by x, the x' values can be computed quicker and will give the correct order of magnitude.

Question 5

Discreteize the following differential equation:

$$\begin{cases} -u'' + 4u = & x \in [0,1] \\ u(0) = -1, & u(1) = 2 \end{cases}$$

by the central difference scheme. Write your linear system of equations (you must give the matrix A and $\vec{b}$). Solve the system by using classical iteration such as Jacobi, Gauss-Seidel and SOR with $n = 1000$. Test you relaxation parameter in SOR for several values and decide which one is better. You need to discuss your results.

The above differential equation can be solved analytically to generate a truth curve, against which the numerical methods can be compared. The solution for this particular ODE is:

$$u(x) = \frac{e^{-2x}(e^{4x} + 2e^{4x+2} - 2e^2 - e^4)}{e^4 - 1}$$

To solve this equation numerically, the equation must first be discretized via the central difference theorem. The first step is to discretize the term $-u''$:

$$u'' = \frac{d^2u}{dx^2} = \frac{1}{h}\left(\frac{u(x+h) - u(x)}{h} - \frac{u(x) - u(x-h)}{h}\right) = \frac{1}{h^2}\Big(u(x+h) - 2u(x) + u(x-h)\Big)$$

This gives the final form of the differential equation, with the same boundary conditions, of:

$$0 = -\frac{1}{h^2}\Big(u(x+h) - 2u(x) + u(x-h)\Big) + 4u(x)$$

Realizing that $h = x_i - x_{i-1}$, this can be simplified and rewritten as:

$$0 = u_{i+1} - 2u_i + u_{i-1} - 4h^2u_i$$

This form of the equation can be used to generate the A and $\vec{b}$ variables, where $A\vec{u} = \vec{b}$:

$$A = \begin{bmatrix} -2-4h^2 & 1 & 0 & 0 & 0 & \dots & 0 \\ 1 & -2-4h^2 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & -2-4h^2 & 1 & 0 & \dots & 0 \\ & & \ddots & \ddots & \ddots & & \\ 0 & \dots & \dots & \dots & \dots & 1 & -2-4h^2 \end{bmatrix}, \vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix}, \vec{b} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ -2 \end{bmatrix}$$

where $h = \frac{1}{n}$. This discretized ODE was then solved using the Gauss-Jacobi , Gauss-Seidel, and SOR with $n = 1000$.

The A matrix as well as the b and u vectors were created with the following code:

```
%Creates the A matrix for problem 5!

n=1000;
A=full(gallery('tridiag',n,1,-2-4*(1/1000)^2,1));
b=zeros(n,1);
b(1)=1;
b(end)=-2;
u=0:1/(n-1):1; %creates a 1000x1 vector bounded by [0,1]
```

The first algorithm tested was the Gauss-Jacobi method. This is the simplest algorithm to implement and has a simple iteration. The G-J method is computed by the equation:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i + \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j^{(k)}\right) \qquad i = 1:n$$

The G-J algorithm can quickly be implemented in just a few lines in Matlab, using Matlab's vector notation. The implementation used for this problem is shown below.

```
function [ x,count ] = Jacobi( A,b,mindiff )
%JACOBI Custom implementation of the jacobi method
%   This function takes in the A matrix, the b vector, the maximum number
%   of iterations, and the minimum error. returns the x values.
if(~any(diag(A)))
    error 'There is a diagonal entry that is 0!'
end

count=0;
x0=zeros(size(b));
while(1)
    count=count+1;
    x=(b-A*x0+(diag(A).*x0))./diag(A);

    if(abs(x-x0)<mindiff)
        break;
    end

    x0=x;
end


end
```

The G-J method was run with a number of different convergence criteria to demonstrate the convergence properties of this method. Figure 1 shows these convergence values. The convergence criteria is defined as $|\vec{x}^{\,(k+1)} - \vec{x}^{\,(k)}|$. When the method is no longer making any significant changes to the $\vec{x}$ values, it has been deemed converged. This is used throughout this homework question as a metric for performance.

The Gauss-Jacobi method was used to check for the solution of the ODE. The solutions presented in Figure 2 show that as the maximum error decreases, the solution approaches that of the analytical

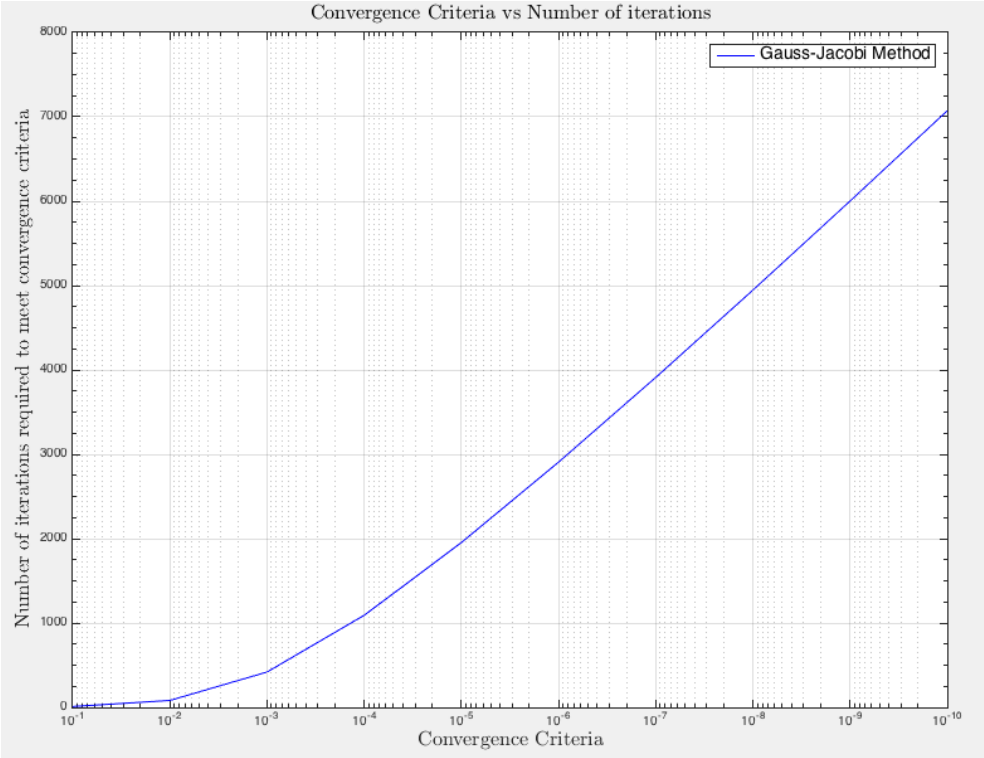

Figure 1: Convergence of the G-J method.

solution, which is a straight line from -1 to 2. All of the numerical methods were checked to ensure the correct solutions were generated.

The second algoritm implemented was the Gauss-Seidel method. The G-S method splits the A matrix into the D, E, and F subparts. The D matrix contains all of the diagonal elements of A. The E and F matrices are the upper and lower triangular matrices, respectively, of A. The G-S algorithm then computes the following iteration:

$$\vec{x}^{\,(k+1)} = (D-E)^{-1}F\vec{x}^{\,(k)} + (D-F)^{-1}\vec{b}$$

The G-S algorithm can also be quite quickly implemented in just a few lines in Matlab, using Matlab's vector notation. Notably, precomputing the values of $(D-E)^{-1}F$ and $(D-F)^{-1}\vec{b}$ greatly speedup the runtime of the algorithm, as these values are invariant. The implementation used for this problem is shown below.

```
function  [ x,count ] = GS( A,b,mindiff )
%GS Custom implementation of the Seidel method
%   This function takes in the A matrix, the b vector, the maximum number
%   of iterations, and the minimum error. returns the x values.

if(~any(diag(A)))
    error 'There is a diagonal entry that is 0!'
end

D=diag(diag(A));
d=diag(A);
E=triu(A-D);
F=tril(A-D);
```

```
deinv=((D-E)^-1)*F;
dfinv=(D-F)^-1*b;
count=0;
x0=zeros(size(b));
n=length(A);
while(1)
    count=count+1;
%     x=deinv*x0+dfinv;
%     for i=1:n
%
%         x(i,1)=(1/D(i,i))*(b(i)-A(i,1:n)*x0+A(i,i)*x0(i));
%     end
    x=(b-A*x0+D*x0).*(1./d);
    if(max(abs(x-x0))<mindiff)
        break;
    end
    if(count>1000000) %for this assignment, should converge by this.
        break;
    end

    x0=x;
end


end
```

The G-S method was run with a number of different convergence criteria to demonstrate the convergence properties of this method. Figure 3 shows these convergence values. The chart also compares

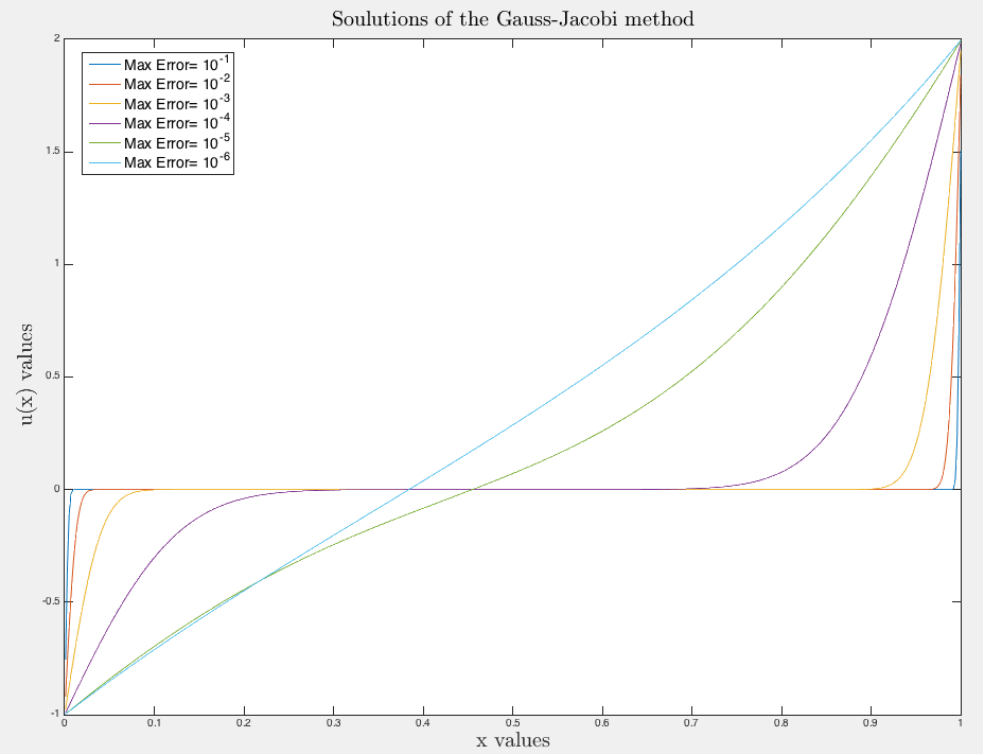

Figure 2: The solutions of the G-J method. As the error decreases, the solution approaches a straight line.

the relative performance of the G-J and G-S methods. As can be seen, the G-S method converges much quicker than the G-J method as the constraints on convergence are tightened.

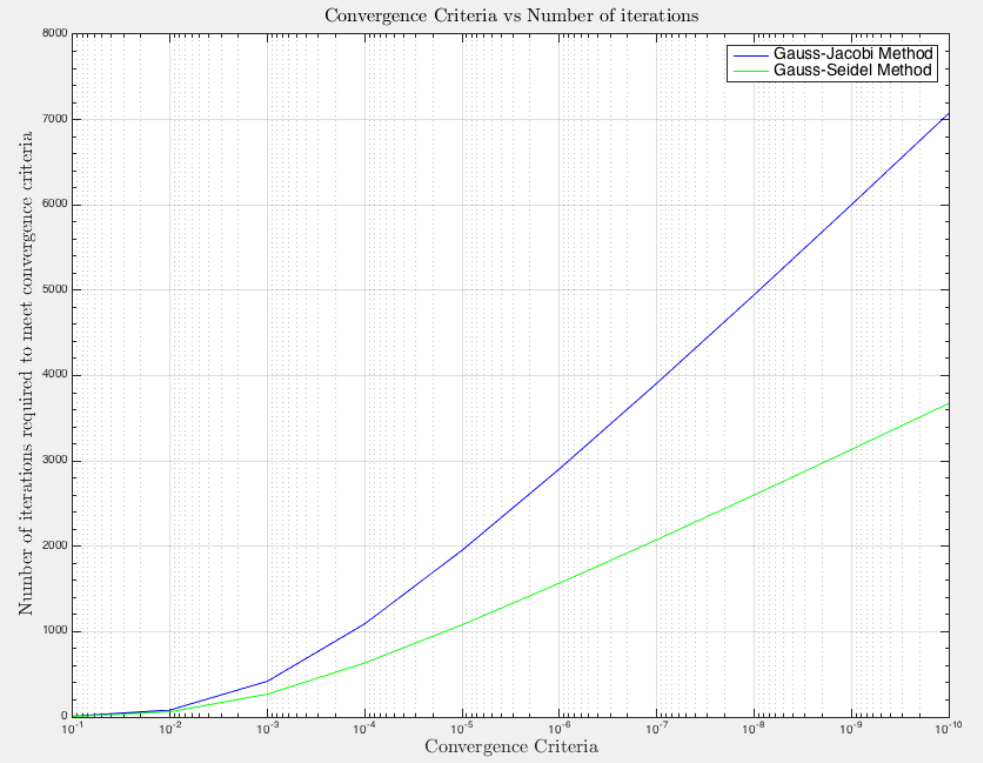

Figure 3: Convergence of the G-S method compared to the G-J method.

The final method that was implemented was the Selective Over Relaxation (SOR) method. The SOR method is a variant of the Gauss-Seidel method that uses a relaxation parameter, ω, to converge on the solution more efficiently. If $\omega = 1$, this method behaves exactly the same as the Gauss-Seidel method. In fact, the code to generate the SOR method is nearly identical to that of the G-S method, except for the addition of computational line and one exit condition:

```
function  [ x,count ] = SOR( A,b,mindiff,w )
%GS Custom implementation of the Seidel method
%   This function takes in the A matrix, the b vector, the minimum error
%   and the relaxation parameter, w. returns the x values.

if(~any(diag(A)))
    error 'There is a diagonal entry that is 0!'
end

D=diag(diag(A));
d=diag(A);
E=triu(A-D);
F=tril(A-D);
deinv=((D-E)^-1)*F;
dfinv=(D-F)^-1*b;

 count=0;
x0=zeros(size(b));
while(1)
```

```
20        count=count+1;
21        xgs=(b-A*x0+D*x0).*(1./d);
22        x=w*xgs+(1-w)*x0;
23
24        if((abs(x-x0)<mindiff))
25
26            break;
27        end
28
29        if(count>1000000)
30           break;
31        end
32
33        x0=x;
34    end
35
36
37    end
```

As can be seen in the SOR code, the value `xgs` is actually the output of the G-S method. This value is then modified by ω and blended with $(1-\omega)x_0$. The choice of the ω parameter is critical for this method to work effectively. For this particular problem, ω values greater than 1.55 would result in the algorithm taking orders of magnitude longer to converge than the G-J or G-S methods. Thus, an additional exit condition was added to ensure the method would eventually cease. The optimal value of ω is defined as:

$$\omega_{opt} = \frac{2}{1+\sqrt{1-\rho(B)^2}}$$

where $B = D^{-1}(E+F)$ and $\rho(B)$ is the spectral radius of B. For this discretized ODE, this value sits at ≈ 1.49375. The effect of the ω on the convergence of the SOR method can be seen in Figure 4. In this plot, 34 different values of ω were used in the SOR calculation to determine the number of iterations necessary to converge to a solution. The 34 points are not equally spaced in this plot, but rather, there are more points near the minima value of $\omega = 1.49375$ to determine the actual best point.

Using the optimal value of $\omega = 1.49375$, the SOR method was compared to the G-S and G-J methods, as seen in Figure 5. The SOR method is able to converge to a given solution in fewer iterations than either the G-J or G-S methods, due completely to the relaxation parameter.

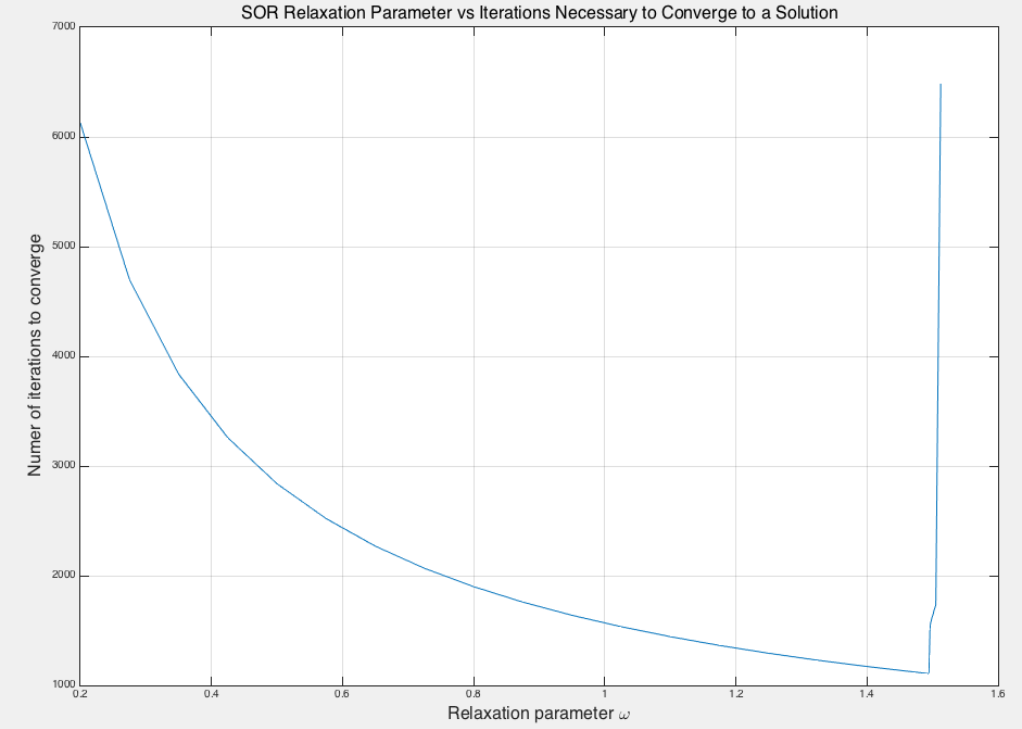

Figure 4: Number of iterations necessary to converge to 10^{-8} vs ω. The effects of ω on runtime can clearly be seen.

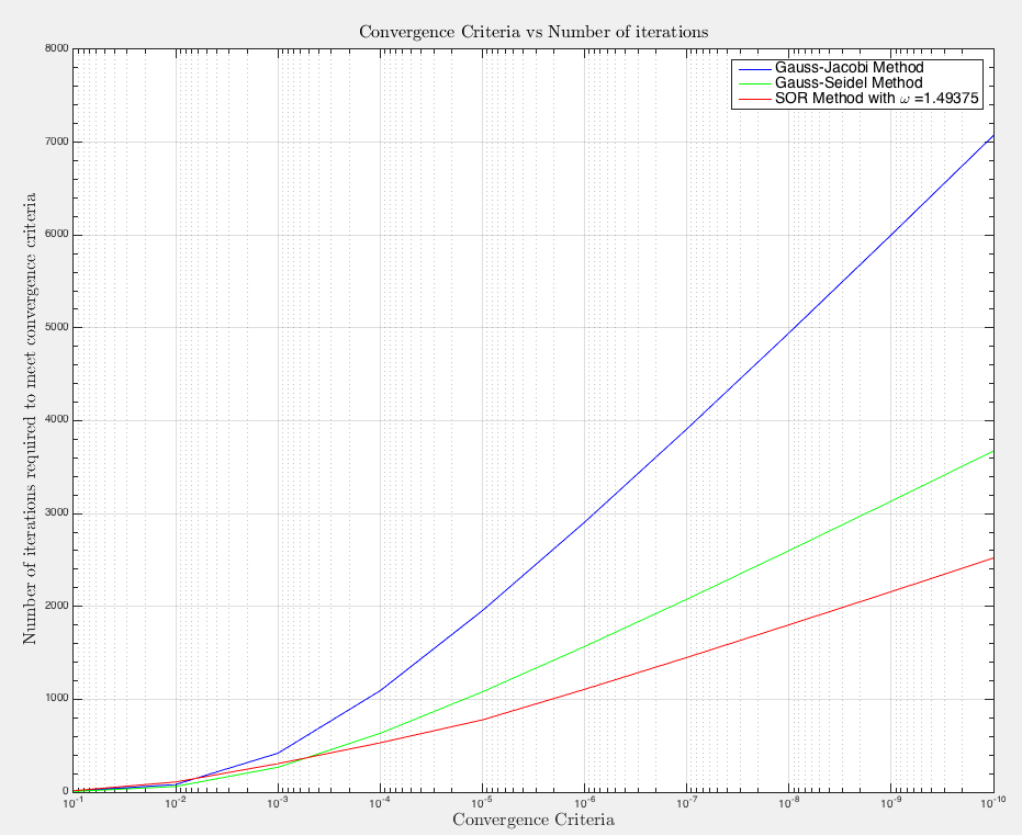

Figure 5: SOR method vs G-S and G-J.