Successive Over-Relaxation Solver for Linear Systems

Akik Kothekar, *Grade 12*
Under the guidance of Prof. S. Baskar,
Mathematics Department,
IIT Bombay

October 14, 2016

1 Introduction

During this project, we studied some basic linear system solvers that can approximate the solution of a linear system of the form

$$A\mathbf{x} = \mathbf{b}, \tag{1}$$

where $\mathbf{b}$ is a given n-dimensional vector, A is an $n \times n$ coefficient matrix and $\mathbf{x}$ is the unknown vector which needs to be approximated.

The aim of this project is to understand the numerical techniques behind the iterative procedures for approximating solution of a linear system of the form (1).

Almost anything involving computer graphics, animation, computer vision, image processing, scientific computing, or simulation of physical phenomena will involve extensive use of vectors and matrices (linear algebra) from simple things like representing spatial transformations and orientations, to very complex algorithms. These things used to be the domain of supercomputing, but now these very same fields are the core of all the latest apps on our desktop, phones, and everywhere else, from video games to computational photography to self-driving cars. Linear systems have been implemented everywhere. Thus, to operate on such applications and to bring about more functionality, we need to solve Linear Systems.

Linear systems, however, can also be solved using methods such as multiplying the inverse of the coefficient matrix A with the given n dimensional vector $\mathbf{b}$. Such methods are called *direct methods*. These methods are efficient only when used to solve linear systems involving a coefficient matrix for which n is relatively small. If n becomes greater than a particular limit, direct methods turn out to be relatively costly. The above methods are deemed costly in comparison to iterative methods. Iterative methods, as the name suggests, involve repeating a set of steps to reach a desired approximation of the solution of the system and are considerably efficient. However, this method tend to approximate the solution, that is, they cannot give the exact answer but an answer which has an error value that is almost insignificant.

We started this project by exploring the three types of iterative solvers for linear systems of the form (1).
The three types are-
1) Jacobi
2) Gauss-Seidel
3) Successive Over-Relaxation Method (SORM)

All the above methods were analysed, keeping in mind efficiency, that is, the time each of the methods required to converge when the coefficient matrix A and given vector $\mathbf{b}$ were kept constant for each and every method to be tested. The size of the coefficient matrix was also changed several times to test the workability of the methods. The methods were transformed into algorithms which were run using a software for numerical computation called Scilab. With the help of this software, we were able to test the efficiency of all the methods by plotting graphs of error vs number of iterations. Here error is the distance between the approximate solution and the exact solution. The error, by definition, is given by

$$\mathbf{e} = \mathbf{x}^* - \mathbf{x_a},$$

where $\mathbf{x}^*$ is the exact solution of the system and $\mathbf{x}_a$ denotes an approximation to the exact solution. However, in the practical situations, one does not know the exact solution and therefore the error as defined above is not of any use in our algorithm. As an alternate, we use the residual error which is calculated by subtracting the product of $\mathbf{A}$ and $\mathbf{x}$ (after every iteration), from the given vector $\mathbf{b}$, that is, the residual error vector $\mathbf{r}$ is defined as

$$\mathbf{r} = A\mathbf{x_a} - \mathbf{b}.$$

Note that if $\mathbf{a_x} = \mathbf{x}^*$, then $\mathbf{r} = \mathbf{0}$, the zero vector, and hence it can be an alternate to the actual error vector. We finally calculate the distance (in the maximum sense to be defined later) between the zero vector and the residual vector in order to decide the convergence of an iterative method. We expect that the value keeps on decreasing as the number of iterations increases.

Another term that plays a very important role in the development of this project, is **convergence**. **Convergence**, in mathematics, is the property (exhibited by certain infinite series and sequence) of approaching a limit more and more closely as the index of the sequence increases or as the number of terms of the series increases. In our case, at the end of every iteration, we get a error vector, and therefore, we get a sequence of error vectors. Therefore convergence is the property of the error to decrease with every iteration and as a limit it should approach to zero.

2 Iterative Methods

General form of the coefficient matrix in (1) is taken to be

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \tag{2}$$

General form of the unknown vector which needs to be approximated is denoted by

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \tag{3}$$

and the general form of the given right hand side vector is

$$\mathbf{b} = \begin{pmatrix} 1 \\ 2 \\ \vdots \\ n \end{pmatrix} \tag{4}$$

The coefficient matrix (2) is split into a Diagonal matrix (**D**) and **C**, which is the difference between **A** and **D**, and are given by

$$\mathbf{D} = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix} \tag{5}$$

$$\mathbf{C} = \begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix} \tag{6}$$

The matrices that were used for this exploration, and for the convergence analysis, are diagonally dominant, that is, the entries of the matrix **D** are nonzero and are greater than or equal to the sum of absolute values of all the terms in the corresponding row of the matrix **C**.

2.1 Basic Iterative Methods For (1).

2.1.1 Jacobi Method

In numerical linear algebra, the Jacobi method (or Jacobi iterative method) is an algorithm for determining the solutions of a diagonally dominant system of linear equations. Here is the general formula for the Jacobi method-

$$A\mathbf{x} = \mathbf{b}$$

As $A = D + C$, we have

$$\begin{aligned} (D+C)\mathbf{x} &= \mathbf{b} \\ D\mathbf{x} + \mathbf{C}\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &= D^{-1}(\mathbf{b} - \mathbf{C}\mathbf{x}) \end{aligned}$$

This equation, when being used to solve the linear system, looks like this-

$$\begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{pmatrix} = \begin{pmatrix} \frac{b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} \ldots - a_{1n}x_n^{(k)}}{a_{11}} \\ \frac{b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)} \ldots - a_{2n}x_n^{(k)}}{a_{22}} \\ \frac{b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)} \ldots - a_{1n}x_n^{(k)}}{a_{33}} \\ \vdots \\ \frac{b_n - a_{n1}x_1^{(k)} - a_{n2}x_2^{(k)} \ldots - a_{n-1}x_{n-1}^{(k)}}{a_{nn}} \end{pmatrix} \tag{7}$$

In this equation, b_j's are elements of the given n dimensional right hand side vector $\mathbf{b}$, $x_j^{(k)}$ and x_j^{k+1} are elements in the approximation of the unknown vector $\mathbf{x}$ after k and $(k+1)$ iterations, respectively. The values $a_{11}, a_{22}, a_{33} \ldots a_{nn}$ are elements of the Diagonal matrix D.
From this equation, it can be inferred that the values of $\mathbf{x}$ calculated after k iterations, are substituted back into the equation to get another set of values which are closer to the exact value, that is, the error keeps on decreasing after every iteration. This equation is solved till the error becomes 0, negligible or insignificant.

I have developed an algorithm to model the Jacobi method using Scilab. It is as follows-

```
clear;
A= fscanfMat("file.txt");
b= [1:size(A,1)]';
D= diag(A);
C= A-(eye(A).*A);
n= size(b,1)
x= zeros(n,1);

count= 1;
err(count)=0;
temp2= A*x;
for j=1:n
    err(count)= err(count)+(sqrt((b(j)-temp2(j))^2));
end

epsilon=0.00001;

while err(count) > epsilon
```

```
    count = count + 1;

    temp= b-(C*x);
    x=temp/D;

    temp2= A*x;
    err(count)=0
    for j=1:n
        err(count)= err(count)+(sqrt((b(j)-temp2(j))^2));
    end
    disp([count err(count)]);
    if err(count) > 10^10 then
        disp("vector x seems to be not converging");
        disp(x);
        return;
    end
end
disp("The method converged");
//disp(x);
NoofIteration = [0:count-1]';
plot(NoofIteration,err,'r-*')
```

Variable Description for Jacobi method Algorithm	
Variable Name	Description
A	The variable in which the Coefficient Matrix is stored
b	The variable in which the given vector is stored. In this case it is an array with values from 1 to n, where n is the size of a column of matrix A
D	This variable consists of an array of the diagonal elements of matrix A
C	This variable consists the matrix A, devoid of its diagonal elements
n	This variable consists of the size of the array *b* and is used to terminate the loop while calculating the error (number of iterations).
x	This is the unknown vector to be approximated. All the initial elements are zero as that is the first set of values that are needed to be input when using iterative methods to solve Linear Systems
count	This variable keeps track of the number of iterations required to approximate the solution for the provided data.
epsilon	It is the error value that I have taken as the least possible, that is if the error is equal to this value during the process, the algorithm stops execution as it has approximated a precise answer

2.1.2 Gauss-Seidal Iteration for (1)

Just like the Jacobi Method, the Gauss-Seidal method is an iterative method used to solve a linear system of equations. Though it can be applied to any matrix with non-zero elements in the diagonal, convergence is only guaranteed if the matrix is either diagonally dominant, or symmetric and positive definite. It is named after the German mathematicians Carl Friedrich Gauss and Philipp Ludwig von Seidel.
The Gauss-Seidal, method, however, is much more efficient than the Jacobi method, that is, the unknown vector $\mathbf{x}$ can be approximated in relatively less number of iterations using the Gauss-Seidal method. The reason for this is clearly evident in its

general form.

General form of the Gauss-Seidal method when used to solve equation (1) is given by

$$\begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{pmatrix} = \begin{pmatrix} \frac{b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} \ldots - a_{1n}x_n^{(k)}}{a_{11}} \\ \frac{b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} \ldots - a_{2n}x_n^{(k)}}{a_{22}} \\ \frac{b_3 - a_{31}x_1^{(k+1)} - a_{32}x_2^{(k+1)} \ldots - a_{1n}x_n^{(k)}}{a_{33}} \\ \vdots \\ \frac{b_n - a_{n1}x_1^{(k+1)} - a_{n2}x_2^{(k+1)} \ldots - a_{n-1}x_{n-1}^{(k+1)}}{a_{nn}} \end{pmatrix} \tag{8}$$

As we can see, the element wise formula of the method is quite similar to that of the Jacobi Method. The computation of $x_j^{(k+1)}$ (where j is the element number in the $k+1$th iteration of the unknown vector $\mathbf{x}$) uses only the elements of $x^{(k+1)}$ that have already been computed, and the elements of $x^{(k)}$ that have not yet been advanced to iteration $k+1$. This means that, unlike the Jacobi method, only one storage vector is required as elements can be overwritten as they are computed, which can be advantageous for very large problems.
What this also does, is that it reduces the number of iterations required to approximate the unknown vector $\mathbf{x}$ as the newly computed value is used for computation of every value after it.
However, unlike the Jacobi method, the computations for each element cannot be done in parallel, that is, all elements in the unknown vector $\mathbf{x}$ have to be computed individually as compared to the Jacobi method where all elements can be computed collectively.

I have developed an algorithm to model the Gauss-Seidal method in Scilab. It is as follows-

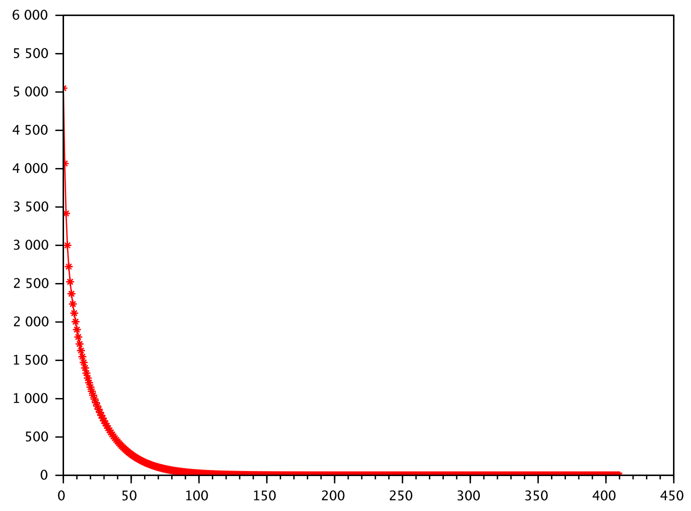
```
clear;
A= fscanfMat("file.txt");
b= [1:size(A,1)]';
D= diag(A);
C= A-(eye(A).*A);
n= size(b,1)
x= zeros(n,1);

count= 1;
```

```
err(count)=0;
temp2= A*x;
for j=1:n
    err(count)= err(count)+(sqrt((b(j)-temp2(j))^2));
end

epsilon=0.00001;

while err(count) > epsilon
    count = count + 1;
    for i=1:n
        temp= b-(C*x);
        x(i)=temp(i)/D(i);
    end

    temp2= A*x;
    err(count)=0
    for j=1:n
        err(count)= err(count)+(sqrt((b(j)-temp2(j))^2));
    end
    disp([count err(count)]);
    if err(count) > 10^10 then
        disp("vector x seems to be not converging");
        disp(x);
        return;
    end
end
disp("The method converged");
//disp(x);
NoofIteration = [0:count-1]';
plot(NoofIteration,err,'r-*')
```

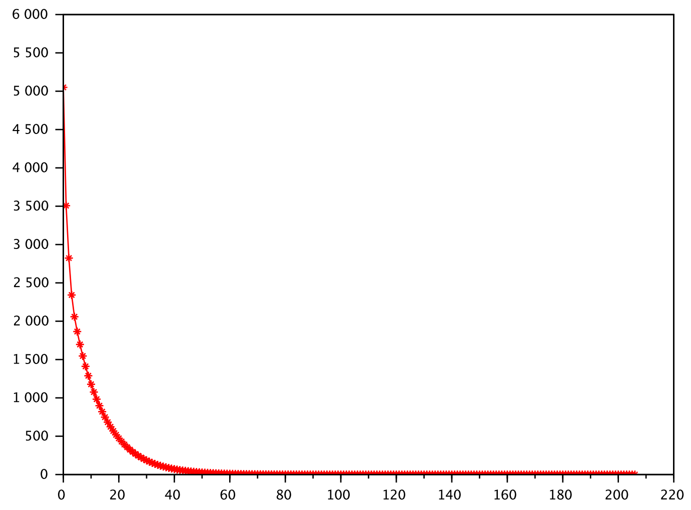
Variable Description for Gauss-Seidal method Algorithm	
Variable Name	Description
A	The variable in which the Coefficient Matrix is stored
b	The variable in which the given vector is stored. In this case it is an array with values from 1 to n, where n is the size of a column of matrix A
D	This variable consists of an array of the diagonal elements of matrix A
C	This variable consists the matrix A, devoid of its diagonal elements
n	This variable consists of the size of the array *b* and is used to terminate the loop while calculating the error (number of iterations).
x	This is the unknown vector to be approximated. All the initial elements are zero as that is the first set of values that are needed to be input when using iterative methods to solve Linear Systems
count	This variable keeps track of the number of iterations required to approximate the solution for the provided data.
epsilon	It is the error value that I have taken as the least possible, that is if the error is equal to this value during the process, the algorithm stops execution as it has approximated a precise answer

I used both the algorithms to test which method turns out to be faster when the coefficient matrix is the Poisson Matrix. I plotted graphs of error vs iteration. the steeper the graph, the faster the method.

The graph for the Jacobi method was as follows-



And the graph for the Gauss-Seidal method was as follows-



The Gauss-Seidal graph is steeper as the method is able to approximate the unknown vector in 207 iterations whereas the Jacobi method solves it in 411 iterations

3 Convergence Analysis

Let A be a 3×3 matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \tag{9}$$

which is diagonally dominant. Let the unknown vector $\mathbf{x}$ be

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{10}$$

and let the known vector $\mathbf{b}$ be

$$\begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}. \tag{11}$$

Therefore, these variables when plugged into the formula (1), can be written like this-

$$\begin{pmatrix} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \tag{12}$$

For the sake of establishing a general formula, we will calculate x_2, which can be written as

$$x_2 = \frac{b_2 - a_{21}x_1 - a_{23}x_3}{a_{22}} \tag{13}$$

(This formula has been calculated from equation - $\mathbf{x} = \frac{\mathbf{b-Cx}}{\mathbf{D}}$)

However, from (8)

$$x_2^{(k+1)} = \frac{b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)}}{a_{22}} \tag{14}$$

Let x^* be the solution to the unknown vector $\mathbf{x}$. Therefore, error (from its definition), $\mathbf{e}$, is

$$\mathbf{e^{(k+1)}} = \mathbf{x}^* - \mathbf{x^{(k+1)}} \tag{15}$$

Therefore, the second element of the error vector is

$$e_2^{(k+1)} = x_2^* - x_2^{(k+1)} \tag{16}$$

which further takes the form

$$e_2^{(k+1)} = (\frac{b_2 - a_{21}x_1^* - a_{23}x_3^*}{a_{22}}) - (\frac{b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)}}{a_{22}}) \tag{17}$$

After simplification, we get

$$e_2^{(k+1)} = -\frac{a_{21}(x_1^* - x_1^{(k)})}{a_{22}} - \frac{a_{23}(x_3^* - x_3^{(k)})}{a_{22}} \tag{18}$$

Thus,

$$e_2^{(k+1)} = -\frac{a_{21}(e_1^{(k)})}{a_{22}} - \frac{a_{23}(e_3^{(k)})}{a_{22}} \tag{19}$$

Similarly we can write

$$e_1^{(k)} = x_1^* - x_1^{(k)} \quad e_3^{(k)} = x_3^* - x_3^{(k)} \tag{20}$$

which can be written as

$$e_1^{(k+1)} = -\frac{a_{12}(e_2^{(k)})}{a_{11}} - \frac{a_{13}(e_3^{(k)})}{a_{11}} \tag{21a}$$

$$e_3^{(k+1)} = -\frac{a_{31}(e_1^{(k)})}{a_{33}} - \frac{a_{32}(e_2^{(k)})}{a_{33}} \tag{21b}$$

Taking mod on both sides in equation(19)-

$$\left|e_2^{(k+1)}\right| = \left|-\frac{a_{21}(e_1^{(k)})}{a_{22}} - \frac{a_{23}(e_3^{(k)})}{a_{22}}\right| \tag{22a}$$

$$= \left|\frac{a_{21}(e_1^{(k)})}{a_{22}} + \frac{a_{23}(e_3^{(k)})}{a_{22}}\right| \tag{22b}$$

Therefore, by triangle inequality, we get

$$\left|e_2^{(k+1)}\right| \leq \left|\frac{a_{21}}{a_{22}}\right|\left|(e_1^{(k)})\right| + \left|\frac{a_{23}}{a_{22}}\right|\left|(e_3^{(k)})\right| \tag{23}$$

Similarly, we have

$$\left|e_1^{(k+1)}\right| \leq \left|\frac{a_{12}}{a_{11}}\right|\left|(e_2^{(k)})\right| + \left|\frac{a_{13}}{a_{11}}\right|\left|(e_3^{(k)})\right| \tag{24a}$$

$$\left|e_3^{(k+1)}\right| \leq \left|\frac{a_{31}}{a_{33}}\right|\left|(e_1^{(k)})\right| + \left|\frac{a_{32}}{a_{33}}\right|\left|(e_2^{(k)})\right| \tag{24b}$$

Define maximum norm

$$\left\|\mathbf{e^{(k)}}\right\|_\infty := max\ \left\{\left|e_1^{(k)}\right|,\left|e_2^{(k)}\right|,\left|e_3^{(k)}\right|\right\} \tag{25}$$

Thus, from equation (25)

$$\left|e_2^{(k+1)}\right| \leq \left(\left|\frac{a_{21}}{a_{22}}\right| + \left|\frac{a_{23}}{a_{22}}\right|\right)\left\|\mathbf{e^{(k)}}\right\|_\infty \tag{26}$$

Let μ be defined as

$$\mu := max\ \left\{\left|\frac{a_{21}}{a_{22}}\right| + \left|\frac{a_{23}}{a_{22}}\right|,\left|\frac{a_{12}}{a_{11}}\right| + \left|\frac{a_{13}}{a_{11}}\right|,\left|\frac{a_{31}}{a_{33}}\right| + \left|\frac{a_{32}}{a_{33}}\right|\right\} \tag{27}$$

Then from equations (25) and (27), we get

$$\left|e_2^{(k+1)}\right| \leq \mu\left\|\mathbf{e^{(k)}}\right\|_\infty \tag{28}$$

Similarly-

$$\left|e_1^{(k+1)}\right| \leq \mu\left\|\mathbf{e^{(k)}}\right\|_\infty \tag{29a}$$

$$\left|e_3^{(k+1)}\right| \leq \mu\left\|\mathbf{e^{(k)}}\right\|_\infty \tag{29b}$$

Taking maximum on the left side quantities of the equations (28), (29a) and (29b), we get

$$\left\|\mathbf{e^{(k+1)}}\right\|_{\infty} \leq \mu \left\|\mathbf{e^{(k)}}\right\|_{\infty} \tag{30}$$

Using the same inequality on the right hand side recursively, we get

$$\left\|\mathbf{e^{(k+1)}}\right\|_{\infty} \leq \mu^{2} \left\|\mathbf{e^{(k-1)}}\right\|_{\infty} \tag{31a}$$

$$\left\|\mathbf{e^{(k+1)}}\right\|_{\infty} \leq \mu^{3} \left\|\mathbf{e^{(k-2)}}\right\|_{\infty} \tag{31b}$$

$$\left\|\mathbf{e^{(k+1)}}\right\|_{\infty} \leq \mu^{k+1} \left\|\mathbf{e^{(0)}}\right\|_{\infty} \tag{31c}$$

Here, $\left\|\mathbf{e^{(0)}}\right\|_{\infty}$ is the initial value of $\mathbf{e}$, that is the error.

The coefficient matrix under consideration in this convergence analysis is diagonally dominant square matrix, which means that the absolute value of the diagonal element in each row of the matrix is greater than the absolute value of all the other elements in the corresponding row. This makes it clear that μ from equation (27), is always less than one. Thus as the power of μ keeps on increasing till it reaches it maximum power $k+1$ from equation (31c)), it tends to reach 0. Subsequently, this fact makes the RHS of the inequality (31c), 0.

According to the definition of the maximum norm of the error, the value of $\left\|\mathbf{e}^{(k+1)}\right\|_{\infty}$ cannot be negative, and according to the inequality (31c) and the aforementioned value of its RHS, the value of $\left\|\mathbf{e}^{(k+1)}\right\|_{\infty}$ can only be less than or equal to 0, as $k \to \infty$. Thus, the only value that $\left\|\mathbf{e}^{(k+1)}\right\|_{\infty}$ can take on is 0, as $k \to 0$.

Therefore,

$$\lim_{k\to\infty} \left\|\mathbf{e^{(k+1)}}\right\|_{\infty} = \mathbf{0} \tag{32}$$

Hence, the error in the approximate solution computed using both the iterative methods converges to zero, when the Coefficient matrix A is diagonally dominant.

4 Successive Over-Relaxation Method

A third iterative method, called the Successive Over-relaxation (SOR) Method, is a generalisation of and improvement on the Gauss-Seidel Method. There are only two differences from the Gauss-Seidal method that make this method much more efficient. They are-

- Presence of a parameter w, and
- Presence of a Relaxation matrix R

The general equation of the SORM is as follows-

$$\mathbf{x}^{(k+1)} = w\ GS(\mathbf{x}^{(k)}) + (1-w)\ R(\mathbf{x}^{(k)}), \tag{33}$$

where GS denotes the Gauss-Seidal method and R is a relaxation matrix which needs to be chosen in such a way that the convergence is accelerated. That is, the aim is to choose the parameter w and the matrix R such that the number of iterations taken to achieve the desired error is significantly low when compared to the Gauss-Seidal method. A methodology to obtain a suitable relaxation matrix R and a corresponding suitable parameter w to minimize the number of iterations is an open problem. Instead of attempting to solve this open problem, let us work on a simpler problem as discussed below.

Here, w is the parameter which is a real number ranging from 1 to 2. The Relaxation matrix (R), usually and in the case of this project as well, is taken as an identity matrix with dimensions similar to that of the coefficient matrix A. However, the parameter (w) is not the same for all matrices. I have developed an algorithm in Scilab which helps me find the best parameter for any given matrix. The algorithm is as follows-

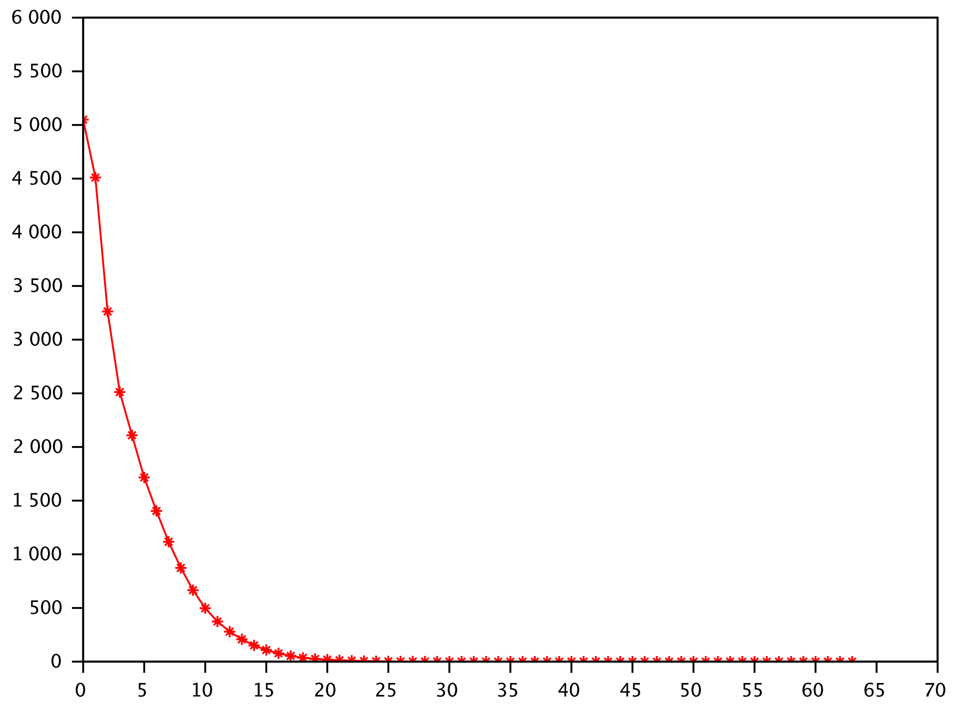
```
clear;

function [Ind,count]=sorm(w)
    A= fscanfMat("file.txt");
    b= [1:size(A,1)]';
    D= diag(A);
    C= A-(eye(A).*A);
    n= size(b,1)
    x= zeros(n,1);
    Ind=1;
    count= 1;
    err(count)=0;
    temp2= A*x;
    for j=1:n
        err(count)= err(count)+(sqrt((b(j)-temp2(j))^2));
    end

    epsilon=0.00001;

    while err(count) > epsilon
        Ind=1;
        count = count + 1;
        for i=1:n
            temp= b-(C*x);
            x(i)=w*(temp(i)/D(i))+((1-w)*x(i))
        end

        temp2= A*x;
        err(count)=0
        for j=1:n
            err(count)= err(count)+(sqrt((b(j)-temp2(j))^2));
        end
       // disp([count err(count)]);
        if (err(count) > 10^4 & count<1000)  then
            disp("vector x seems to be not converging");
            //disp(x);
            Ind=0;
            return;
        end
    end
    disp("The method converged");
    //disp(x);
endfunction


bw=0;
bit=0;
w=[1.0:0.1:2];
```

```
loopv=0.1;
loopst=0.1;

for i=1:3
    for j=1:10
        [Ind,count]= sorm(w(j));
        if bit==0 then
            bit=count;
            bw= w(j);
        end
        if (count<bit & Ind==1) then
            bit=count;
            bw= w(j);
        end
        disp(bit,bw);
    end
    loopv= bw-loopst;
    loopst= loopst/10;
    if (loopv+(loopst*10))>2 then
        return;
    end
    w=[loopv:loopst:(loopv+(loopst*10))];
end
disp("best iteration and parameter-");
disp(bit,bw);
```

The variable description for this algorithm is similar to that of the Jacobi and Gauss-Seidal methods.
This algorithm takes the coefficient matrix as an input from a specified file. The main part of the algorithm (positioned below the function) sends values of the parameter w to the function. The function uses these values to calculate the number of iterations required to approximate the unknown vector $\mathbf{x}$ with given parameter value. These iterations are sent to the main method which finds the smallest value amongst all the values sent back. The initial range of the values of w is from 1 to 2. After the first iteration, the range is much more concentrated. For example: if, after the first iteration, the best value for w was found to be 1.7, the range for the second iteration would be from 1.6 to 1.8, with a step value of 0.01. The similar would happen for the third iteration, in which the step value would further decrease to 0.001.

I used this algorithm to find the best parameter for the Poisson Matrix. The result came out to be **1.47**. When this parameter was substituted in the SOR Method, the unknown vector **x** was approximated in 64 iterations, which was significantly lesser than the 207 iterations required by the Gauss-Seidal method.

The Graph for the SORM with parameter as 1.47 is given below-



5 Conclusion

I would like to conclude by saying that, along with Professor Baskar, I was able to explore three iterative methods used to approximate the unknown vector **x**. Three methods are-

- Jacobi method

- Gauss-Seidal method
- Successive Over-Relaxation method

I also developed algorithms to model these iterative methods along with an algorithm for finding the best parameter for a given coefficient matrix, when the relaxation matrix is taken as an identity matrix, of size similar to that of the coefficient matrix.

6 Bibliography

- Definition of 'error': https://www.britannica.com/topic/error-mathematics
- Definition of convergence: https://www.britannica.com/topic/convergence-mathematics
- Description of Jacobi method: http://www.maa.org/press/periodicals/loci/joma/iterative-methods-for-solving-iaxi-ibi-jacobis-method
- Description of Gauss-Seidal method: http://www.maa.org/press/periodicals/loci/joma/iterative-methods-for-solving-iaxi-ibi-gauss-seidel-method
- Description of SORM: http://www.maa.org/press/periodicals/loci/joma/iterative-methods-for-solving-iaxi-ibi-the-sor-method