Iterative Methods for Linear Systems

We present the basic concepts for three of the most fundamental and well known iterative techniques for solving a system of linear equations of the form $Ax = b$. Iterative techniques have become increasingly important since the advent of scientific computing and its use for solving large systems of linear equations [1]. Numerical methods for solving differential equations are used in applications ranging from thermal modeling of transistors, structural modeling of bridges, determining the stress profile in airplane components and biomedical devices, and has countless other applications. Often these techniques yield very large and sparse matrices, for which direct methods may be slow and prone to round off error [2]. It was Gauss in 1823 who first presented a technique for "guessing" a solution to a linear system and iteratively improving this solution until an "acceptable" solution is reached. He even noted that this "indirect procedure can be done while one is half asleep" which explains why these methods are readily adapted to solving systems via computers [3]. Therefore, iterative methods are widely used in many disciplines. In this report we present algorithms for the Jacobi Method, Gauss-Seidel Method, and Successive Over Relaxation Method, in addition we work simple examples to demonstrate the concepts, as well as providing pseudo code for Matlab implementation. In addition we explore the question of when do these methods converge to the exact solution as well as what is the rate of this convergence. Finally, we present a comparison between the rates of convergence of each method.

The Jacobi Method

The Jacobi Method is a simple iterative method for solving a system of linear equations. The basic idea of the method is to continually solve for each diagonal element until convergence is reached. The method is very simple, but unfortunately does not converge for all linear systems. Before introducing the algorithm for using the Jacobi Method, the conditions for using the matrix will be discussed.

System of Equations. The Jacobi Method is designed to work for linear systems that have the corresponding matrix equation Ax = b. The matrix A should be an n x n square for the Jacobi Method to work properly. The vectors x and b should have length n that matches the matrix dimensions.

Diagonal dominance. The matrix A must also be diagonally dominant in order to guarantee the convergence of the Jacobi Method. Specifically, the Jacobi Method is guaranteed to converge if the matrix A has row diagonal dominance. Diagonal dominance means that the absolute value of the diagonal element is greater than the sum of the absolute values of the other terms in the row. Simply put, $|a_{ii}| > \sum |a_{ij}|$ for $i \neq j$ [3].

Example 1. Determine if the following matrix A is diagonally dominant:

$$A = \begin{bmatrix} 3 & -1 & 1 \\ 0 & 6 & -2 \\ 2 & -4 & -8 \end{bmatrix}$$

The absolute value of the first diagonally element is |3|=3. Comparing that value to the other terms in the row it is seen that |3| > |-1| + |1| or 3 > 2. Now using the same approach for rows 2 and 3 it is seen that |6| > |0| + |-2| and |-8| > |2| + |-4|. Thus, the matrix A is diagonally dominant.

Decomposition of Matrices. The first approach to using the Jacobi Method is to treat the linear system as a matrix system. As stated earlier, the matrix equation corresponding to the linear system is Ax=b. The first step in implementing the Jacobi Method is to break the matrix A into two parts; the diagonal component D and the remainder R so that A = D + R.

Example 2. Decompose the matrix A from example 1 into the diagonal and remainder matrices.

$$\begin{bmatrix} 3 & -1 & 1 \\ 0 & 6 & -2 \\ 2 & -4 & -8 \end{bmatrix} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & -8 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 1 \\ 0 & 0 & -2 \\ 2 & -4 & 0 \end{bmatrix}$$

A = D + R

Jacobi Method using Matrices. After splitting the matrix A into the diagonal and remainder matrices, the vector x is solved for using the equation $x^{(k+1)} = D^{-1} (b - Rx^{(k)})$ until a desired convergence is reached [4]. In order to begin using this equation an approximation must be made for the first term $x^{(0)}$. It is often convenient to use $x^{(0)} = [0 \ \dots \ 0]^T$ to ease computational load especially if the method is done by hand. Another common approximation is $x^{(0)} = [1 \ \dots \ 1]^T$.

Example 3. Use the Jacobi Method to find $x^{(1)}$ for the following linear system using $x^{(0)} = [0 \ 0]^T$:

$$\begin{bmatrix} 3 & 1 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \end{bmatrix}$$

$$D = \begin{bmatrix} 3 & 0 \\ 0 & 5 \end{bmatrix} \quad D^{-1} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/5 \end{bmatrix} \quad R = \begin{bmatrix} 0 & 1 \\ 2 & 0 \end{bmatrix}$$

$$x^{(k+1)} = D^{-1}(b - Rx^{(k)}), \ x^{(0)} = [0 \ 0]^T$$

$$x^{(1)} = D^{-1}b - D^{-1}Rx^{(0)}$$

$$x^{(1)} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/5 \end{bmatrix} \begin{bmatrix} 4 \\ 7 \end{bmatrix} - \begin{bmatrix} 1/3 & 0 \\ 0 & 1/5 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 2 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$x^{(1)} = [4/3 \ 7/5]^T$$

Note on simplicity. The simplicity of the Jacobi Method can be seen if one treats $D^{-1}b$ and $D^{-1}R$ as constants. Notice:

$$x^{(k+1)} = D^{-1}b - D^{-1}Rx^{(k)}$$
$$Let\ D^{-1}b = C\ and\ D^{-1}R = F$$
$$x^{(k+1)} = C - Fx^{(k)}$$
$$Only\ x^{(k)}\ changes!$$

Jacobi Method using Equations. Another way to use the Jacobi Method is by keeping the linear system of equations in equation form. Of course, it is important to remember to check if the system is diagonally dominant before proceeding with this method. To implement the Jacobi Method in this way, each equation is rearranged to solve for the diagonal element. Once the equations are rearranged, the approximate values of $x^{(n)}$ are plugged in to find $x^{(n+1)}$. The previous example will be used again to show this method.

Example 4. Solve example 3 again but do not use the matrix form.

$$3x_1 + 1x_2 = 4$$
$$2x_1 + 5x_2 = 7$$
$$Rearraning:$$
$$x_1 = \frac{4 - x_2}{3},\ x_2 = \frac{7 - 2x_1}{5}$$
$$Using\ x^{(0)} = [0\ 0]^T$$
$$x_1 = 4/3,\ x_2 = 7/5$$

Example 5. Solve example 4 up to $x^{(5)}$.

$$x_1 = \frac{4 - x_2}{3},\ x_2 = \frac{7 - 2x_1}{5}$$
$$Using\ x^{(1)} = [4/3\ 7/5]^T$$
$$x_1^{(2)} = 0.8\overline{6}, x_2^{(2)} = 0.8\overline{6}$$
$$Using\ x^{(2)} = [0.8\overline{6}\ 0.8\overline{6}]^T$$
$$x_1^{(3)} = 1.0\overline{4},\ x_2^{(3)} = 1.05\overline{3}$$
$$x_1^{(4)} = 0.98\overline{2},\ x_2^{(4)} = 0.98\overline{2}$$
$$x_1^{(5)} = 1.006,\ x_2^{(5)} = 1.007$$

Algorithm for using the Jacobi Method. The easiest way to implement the Jacobi Method in a computer program like MatLab is to use the matrix form of the linear system. An important thing to remember is that in a program like MatLab the matrixes are best solved element by element. The following algorithm can be used to implement the Jacobi Method:

- Set the initial approximation $x^{(0)}$
- While convergence is not reached:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij}x_j^{(k)}\right), \quad i = 1, 2, \ldots, n.$$

- Or equivalently:

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

- End when convergence is reached [3]

Of course, in order for the algorithm to end, there must be a definition for convergence. Some possible definitions include a set number of iterations, or a tolerance level (i.e. $x - x^{(k)} \leq T$). The problem with these definitions of convergence is that they may not get a satisfactory answer or are impossible to implement. Thus, a more practical definition of convergence is a tolerance level for successive iterations (i.e. $x^{(k+1)} - x^{(k)} \leq T$). It is also common to set a maximum number of iterations to avoid iterating forever (especially if the method does not convergence for a matrix).

Convergence. The condition for convergence for iterative methods is that the spectral radius must be less than 1. The spectral radius of the Jacobi Method is $\rho(D^{-1}R) < 1$. The definition of $\rho(M)$ is the maximum eigenvalue of M, or $\max\{|\lambda_1|, |\lambda_2|, \dots, |\lambda_n|\}$. For the Jacobi Method, $\rho(M) = \rho(D^{-1}R)$, which is equivalent to:

$$\max_{j=1}^{n} \frac{\sum_{i=1, i\neq j}^{n} |A_{ji}|}{|A_{jj}|}$$

Looking closely at the definition of convergence for the Jacobi Method, it is clearly seen that if the matrix A is diagonally dominant, the method will converge [6].

Jacobi Method vs. Gaussian Elimination. To conclude the discussion of the Jacobi Method, some comparisons will be made with Gaussian Elimination. Gaussian Elimination is more accurate than the Jacobi Method, since it finds the exact solution. Gaussian Elimination is easier to solve for smaller matrices, since it does not require multiple iterations. Gaussian Elimination also does not have the same restrictions on the matrix A as the Jacobi Method, so it can be used for more linear systems. The advantage of the Jacobi Method is that it is less costly than Gaussian Elimination, so it is cheaper to implement. The Jacobi Method is much better suited for solving large linear system of equations. It only requires two vectors for storage, $x^{(k)}$ and $x^{(k+1)}$. Although the Jacobi Method is very simple and cheap to implement, it does not converge very quickly. Thus, there are other methods such as the Gauss-Seidel method and the Successive Over Relaxation method that were created to converge faster.

Gauss Seidel Iterative Method

The Gauss Seidel method is very similar in principle to the Jacobi method and was created to improve the Jacobi method. The advantages of the Gauss Seidel method are that only one storage vector is required, round off error is reduced, and the number of iterations is drastically reduced compared to the Jacobi method. The disadvantages of using this method are that the computations cannot be done in parallel and that the results depend on the initial equations.

Why use Gauss-Seidel over Gaussian Elimination

Firstly, in Gauss elimination, the only way you can control the round-off error is by either by choosing Gauss elimination with partial pivoting, or by increasing the number of bits you use to store your numbers, going from single precision to double precision, or to quad precision, but here, because it's an iterative procedure, you can control the round-off error, because you can stop the procedure as soon as the absolute relative approximate error is less than some pre-

specified tolerance. You don't have that luxury in Gauss elimination. Secondly, for large matrices it becomes much faster to solve a system using the Gauss Seidel method. [6]

Requirements for Convergence

The Gauss Seidel method first requires that there are no zero diagonal entries on the matrix A (Ax=b). Secondly, to guarantee convergence the Gauss Seidel method requires that one of the following is true:

1. A is symmetric positive definite
    - A matrix A is positive definite if $z^T Az$ is positive for every non-zero column vector z
2. A is strictly or irreducibly diagonally dominant
    - A matrix is diagonally dominant if the sum of the elements on the diagonal are greater than or equal to the sum of the other elements. [7]

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|$$

Spectral Radius

The spectrum of matrix A is the set of eigenvalues of A and is given by $\rho(A)$. For an iterative method the reduction matrix of the method is given by G. In the case of Gauss Seidel the reduction matrix $G = -(D + L)^{-1} * U$. For an iterative method if $|\rho(G)| < 1$ then the iterative method is guaranteed convergence. Additionally for any iterative method the error in the kth step can be given as $\varepsilon_k = G^k \varepsilon_0$. The number of iterations required for 10^{-d} accuracy is roughly given by $\sim \frac{-d}{\log(\rho(G))}$. [8]

Proof of Convergence

1. The reduction matrix G always has n linearly independent eigenvectors $e_1, \dots, e_n$ with corresponding eigenvalues $\lambda_1, \dots, \lambda_n$.
2. Therefore $\{e_i\}$ is a basis and $\varepsilon_0 = \sum_{i=1}^{n} c_i e_i$ for some constants c_i
3. Then $\varepsilon_1 = G\varepsilon_0 = \sum_{i=1}^{n} c_i e_i \lambda_i$ and $\varepsilon_k = G\varepsilon_{k-1} = \sum_{i=1}^{n} c_i e_i \lambda_i^k$
4. So if $|\lambda_i| < 1$ for all i then $e_k \to 0 \; as \; k \to 0$
5. Therefore the iteration converges if and only if the eigenvalues of G all have absolute value less than one.

Solving the System

The Gauss Seidel method is defined by the iteration $Lx^{(k+1)} = b - Ux^{(k)}$. Where x^k is the kth iteration of x, L is a lower triangular matrix, and U is an upper triangular matrix, such that A=U+L. Solving for x gives $x^{(k+1)} = L^{-1}(b - Ux^{(k)})$. Because L is lower triangular we can use forward substitution to solve resulting in $x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{i<j} a_{ij} x_j^{(k)}$.

Computational Complexity

The computation complexity of the Gauss Seidel method is given by $O(n^2)$ for each iteration. This can be clearly seen when looking at the formula $x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{i<j} a_{ij}x_j^{(k)}$. Looking at this formula we can equate the summations to simply multiplying a column vector of length n-1 times a constant, which has computational complexity of $O(n)$. The subtraction and the division are trivial compared to the complexity of the vector multiplication, so we can simply ignore them. However, we do have to compute the vector multiplication n times, once for each element in $x^{(k+1)}$. This results in $(n * n) = O(n^2)$. Although both the Jacobi method and the Gauss Seidel method have the same complexity the Gauss Seidel method is more efficient because it requires less iterations to solve the system. [9]

Gauss-Seidel Example (Equation Form)

$12x_1 + 3x_2 - 5x_3 = 1$

$x_1 + 5x_2 + 3x_3 = 28$

$3x_1 + 7x_2 + 13x_3 = 76$

Find (x_1, x_2, x_3) after 2 iterations. Starting from $(x_1, x_2, x_3) = (1, 0, 1)$

Rewrite the system of equations:

$$x_1 = \frac{1 - 3x_2 + 5x_3}{12}$$

$$x_2 = \frac{28 - x_1 - 3x_3}{5}$$

$$x_3 = \frac{76 - 3x_1 - 7x_2}{13}$$

Iteration #1

Substitute $(x_1, x_2, x_3) = (1, 0, 1)$ into above equation

$$x_1 = \frac{1 - 3(0) + 5(1)}{12} = 0.5$$

Substitute $(x_1, x_2, x_3) = (0.5, 0, 1)$ into above equation

$$x_2 = \frac{28 - 0.5 + 3(1)}{5} = 4.9$$

Substitute $(x_1, x_2, x_3) = (0.5, 4.9, 1)$ into above equation

$$x_3 = \frac{76 - 3(0.5) - 7(4.9)}{13} = 3.0923$$

Error Analysis:

New $(x_1, x_2, x_3) = (0.5, 4.9, 3.0923)$

Old $(x_1, x_2, x_3) = (1, 0, 1)$

$$|\epsilon_a^1| = \left|\frac{0.5 - 1}{0.5}\right| \times 100 = 100\%$$

$$|\epsilon_a^2| = \left|\frac{4.9 - 0}{4.9}\right| \times 100 = 100\%$$

The errors turn out to be 100%

$$|\epsilon_a^3| = \left|\frac{3.0923 - 1}{3.0923}\right| \times 100 = 67.662\%$$

The error of x_3 is 67.662%

To find the maximum number of three error

$\max(100, 100, 67.662) = 100\%$

The maximum absolute relative approximate error is 100%

Iteration #2

$(x_1, x_2, x_3) = (0.5, 4.9, 3.0923)$

$$x_1 = \frac{1 - 3x_2 + 5x_3}{12} = \frac{1 - 3(4.9) + 5(3.0923)}{12} = 0.14679$$

$(x_1, x_2, x_3) = (0.14679, 4.9, 3.0923)$

$$x_2 = \frac{28 - x_1 - 3x_3}{5} = \frac{28 - 0.14679 - 3(3.0923)}{5} = 3.7153$$

$(x_1, x_2, x_3) = (0.14679, 3.7153, 3.0923)$

$$x_3 = \frac{76 - 3x_1 - 7x_2}{13} = \frac{76 - 3(0.14679) - 7(3.7153)}{13} = 3.8118$$

We get $(x_1, x_2, x_3) = (0.14679, 3.7153, 3.8118)$

Error analysis:

New $(x_1, x_2, x_3) = (0.14679, 3.7153, 3.8118)$

Old $(x_1, x_2, x_3) = (0.5, 4.9, 3.0923)$

$$|\epsilon_a^1| = \left|\frac{0.14679 - 0.5}{0.14679}\right| \times 100 = 240.61\%$$

$$|\epsilon_a^2| = \left|\frac{3.7153 - 4.9}{3.7153}\right| \times 100 = 31.889\%$$

$$|\epsilon_a^3| = \left|\frac{3.8118 - 3.0923}{3.8118}\right| \times 100 = 18.874\%$$

To find the maximum number of three error

$$\max(240.61, 31.889, 18.874) = 240.61\%$$

Continue doing the iteration. We get the following table.

Iteration	max$\|\epsilon_a\|$%
1	100
2	240.6
3	80.23
…	…
6	0.743

The error is getting smaller and smaller.

In the 6th iteration, $(x_1, x_2, x_3) = (0.99919, 3.00001, 4.00001)$

By using Matlab, the exact answer is $(x_1, x_2, x_3) = (1, 3, 4)$

In conclusion, by doing more iterations, we are going to get closer and closer to the exact value. The solution converges. [11]

Example 2 (Matrix Form)

A=$\begin{bmatrix} 7 & 1 \\ 1 & 4 \end{bmatrix}$ $b = \begin{bmatrix} 8 \\ 10 \end{bmatrix}$

A quick matlab computation gives $G = \begin{bmatrix} 0 & -.1429 \\ 0 & .0357 \end{bmatrix}$ with $\lambda = 0, .0357$

Clearly the max eigenvalue of G is less than 1, therefore $\rho(G) < 1$ and the Gauss Seidel method is guaranteed convergence.

$L = \begin{bmatrix} 7 & 0 \\ 1 & 4 \end{bmatrix}$ $U = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ $x^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$$T = -L^{-1}U = -\begin{bmatrix} \frac{1}{7} & 0 \\ -\frac{1}{28} & \frac{1}{4} \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1}{7} \\ 0 & \frac{1}{28} \end{bmatrix}$$

$$C = L^{-1}b = \begin{bmatrix} \frac{1}{7} & 0 \\ -\frac{1}{28} & \frac{1}{4} \end{bmatrix}\begin{bmatrix} 8 \\ 10 \end{bmatrix} = \begin{bmatrix} 8/7 \\ 31/14 \end{bmatrix}$$

$x^{(k+1)} = Tx^{(k)} + C$

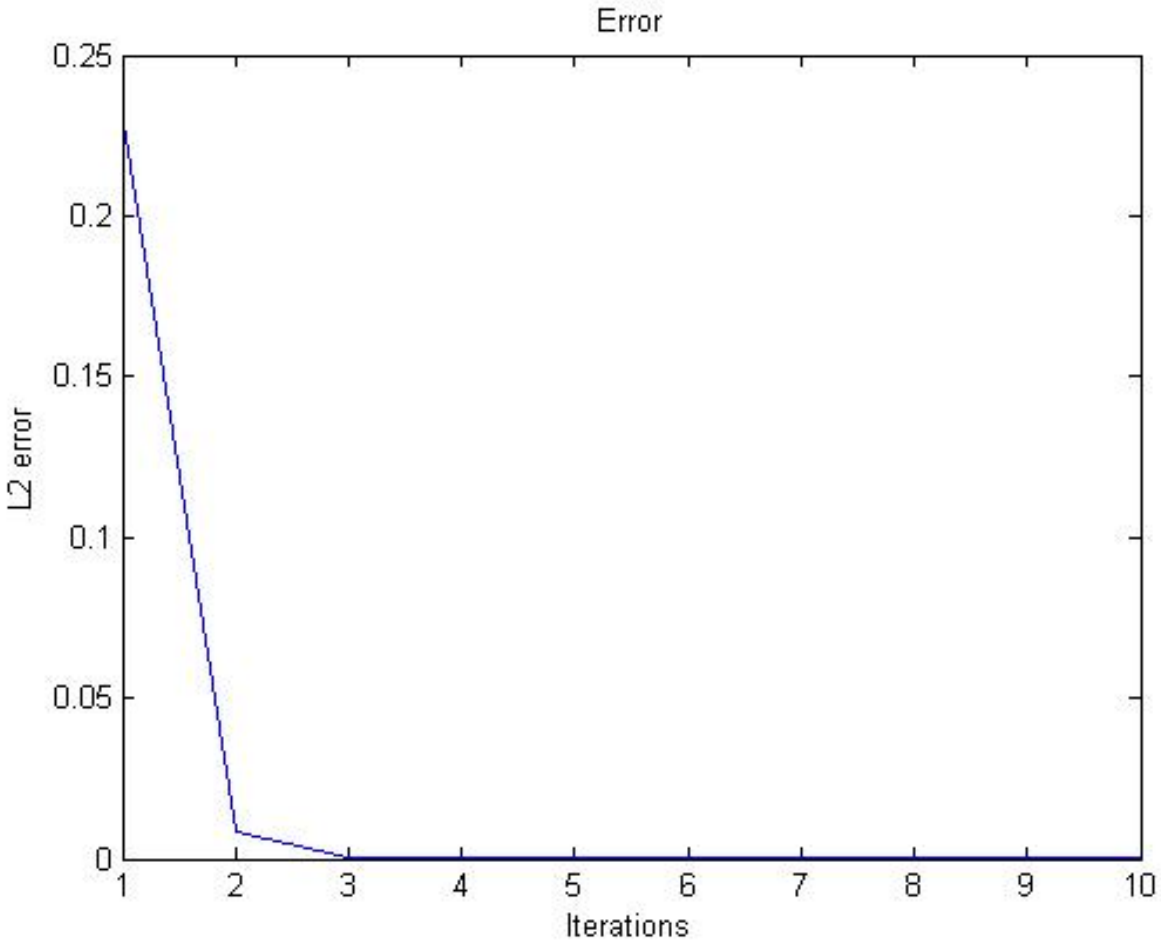
$$x^{(1)} = Tx^{(0)} + C = \begin{bmatrix} 0 & -\frac{1}{7} \\ 0 & \frac{1}{28} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 8/7 \\ 31/14 \end{bmatrix} = \begin{bmatrix} 1 \\ 9/4 \end{bmatrix}$$

$$x^{(2)} = Tx^{(1)} + C = \begin{bmatrix} 0 & -\frac{1}{7} \\ 0 & \frac{1}{28} \end{bmatrix} \begin{bmatrix} 1 \\ 9/4 \end{bmatrix} + \begin{bmatrix} 8/7 \\ 31/14 \end{bmatrix} = \begin{bmatrix} .821 \\ 2.294 \end{bmatrix}$$

$$x^{(3)} = Tx^{(2)} + C = \begin{bmatrix} 0 & -\frac{1}{7} \\ 0 & \frac{1}{28} \end{bmatrix} \begin{bmatrix} .821 \\ 2.294 \end{bmatrix} + \begin{bmatrix} 8/7 \\ 31/14 \end{bmatrix} = \begin{bmatrix} .815 \\ 2.296 \end{bmatrix}$$

$$x^{(4)} = Tx^{(3)} + C = \begin{bmatrix} 0 & -\frac{1}{7} \\ 0 & \frac{1}{28} \end{bmatrix} \begin{bmatrix} .815 \\ 2.296 \end{bmatrix} + \begin{bmatrix} 8/7 \\ 31/14 \end{bmatrix} = \begin{bmatrix} .8148 \\ 2.2963 \end{bmatrix}$$

As you can see at some point the difference in successive solutions will be below a "tolerance" which signifies that you can stop. In this case 4 iterations results in an answer accurate to the ten thousandth place.



When to Stop Iterating

Because it requires the actual solution to calculate the error at each iteration it doesn't make sense to have the number of iterations depend on the error. Therefore we look at how much the solution changes after each iteration and when the change between successive iterations becomes sufficiently small we stop.

Matlab Implementation (using inverse)

```
function [x,iterations] = gaussSeidel(A,b,x0,t)
% x=solution, iterations=iterations of method, A=matrix Ax=b, b=column
% vector, x0=initial guess, t=tolerance
x2 = x0;
iterations = 0;
D = diag(diag(A));
U = triu(A-D);
L = tril(A-D);
invL = inv(L+D);
iterations=0;
difference=9;
while difference<tolerance
    iterations = iterations + 1;
    x1 = x2;
    x2 = invL*(b-(U*x1));
    difference = norm(abs(x2-x1));
end
x = x2;
end
```

Matlab Implementation (no inverse)

```
function [X,iterations] = gseidel(A,B,x0,t)
% x=solution, iterations=iterations of method, A=matrix Ax=b, b=column
% vector, x0=initial guess, t=tolerance
iterations=0;
difference=inf;
while difference>t
d = diag(A);
D = diag(d);
notD = A - D;
n = length(d);
X=x0;
for j=1:n
    X(j,1) = (B(j) - notD(j,:)*X)/d(j);
end
difference = norm(abs(X-x0));
iterations=iterations+1;
x0=X;
end
end
```

Successive Over-Relaxation

Successive Over-Relaxation (SOR) is a generalization of the Gauss-Seidel Method, developed by David Young for his Ph.D. thesis[12]. First we consider the practical implication of this method and compare the algorithms of the Jacobi, Gauss-Seidel, and SOR Methods. Then we will turn to the theoretical aspects governing convergence.

Algorithm for using the Successive Over Relaxation Method.

Just like the Jacobi and Gauss-Seidel Methods, we will use the matrix form of the linear system to implement the SOR Method in computer programs such as Matlab. Below contains a more detailed algorithm of the SOR Method [13]:

- Set the initial approximation $x^{(0)}$, ω (the relaxation parameter), *TOL* (the tolerance), and *N* (the maximum number of iterations)
- Create a loop, let k = 1
- While convergence is not reached, $|x - x^{(0)}| > TOL$, or $k < N$, do the following:
- For $i = 1, \dots, n$, Set $x_i = (1-\omega)x_i^{(0)} + \omega/a_{ii}[b_i - \sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij}x_i^{(0)}]$
- Set $k = k + 1$, $x_i^{(0)} = x_i$
- End loop if convergence is reached, $|x - x^{(0)}| < TOL$, or $k > N$

Comparison of the Jacobi, Gauss-Seidel, and SOR Method. Below, we have a side-by-side comparison of the three aforementioned methods [14].

Jacobi	$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i}^{n} a_{ij}x_j^{(k)}\right)$
Gauss-Seidel	$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij}x_j^{(k)}\right)$
SOR	$x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij}x_j^{(k)}\right)$

From the table above, we can see that the Gauss-Seidel Method is the SOR Method with $\omega = 1$.

Theory

The most general form of the SOR method can be written by splitting the coefficient matrix *A* (recall we are solving the linear system corresponding to *Ax*=*b*) as follows:

$$A = (\omega^{-1}D + L) + \left((1-\omega^{-1})D + U\right)$$

The definitions of the matrices appearing above are as follows: *L* is the strictly lower triangular portion of *A*, *D* is the diagonal portion of *A*, and *U* is the strictly upper triangular portion of *A*. Under this splitting our system can be re-written as:

$$(\omega^{-1}D + L)x = \big((\omega^{-1} - 1)D - U\big)x + b$$

This leads to the natural choice for the iteration,

$$(\omega^{-1}D + L)x^{k+1} = \big((\omega^{-1} - 1)D - U\big)x^k + b$$

Two things to note is that this iteration reduces to Gauss-Seidel when the relaxation parameter ω is equal to 1. In addition we note that the iteration is consistent, i.e. if the solution vector *x* is provided as the guess, $x^k = x$, then $x^{k+1} = x$. The procedure for obtaining the solution is identical to that presented earlier for Gauss-Seidel, and will not be discussed here. Instead we will focus on the convergence properties of this method.

To examine whether this iterative method does converge to the correct solution, as well as the rate of this convergence, we introduce the recurrence relation for the error. The error vector at the *k*-th iteration is defined as $e^k = x^k - x$ where *x* is the vector solving $Ax = b$, then the recurrence relation can be written as

$$e^{k+1} = Ge^k$$

$$G = (\omega^{-1}D + L)^{-1}\big((\omega^{-1} - 1)D - U\big)$$

Through induction, one can see that:

$$e^k = G^k e^0$$

Written in this form we can see that the convergence of the SOR method depends on the properties of the reduction matrix *G*. More precisely we state that if $A \in C^{nxn}$ is a Hermitian positive definite matrix, *G* is Hermitian, and $\omega \in (0,2)$ then the SOR method converges for *any* starting iterate.

Proof [15]

For convenience we introduce the matrix $Q = (\omega^{-1}D + L)$ which allows us to write the reduction matrix as $G = Q^{-1}\big((\omega^{-1} - 1)D - U\big)$. Now consider (*I* is identity matrix)

$$\begin{aligned} I - G &= I - [Q^{-1}(\omega^{-1} - 1)D - Q^{-1}U] \\ &= Q^{-1}[Q + (1 - \omega^{-1})D + U] \\ &= Q^{-1}A \end{aligned} \qquad (1)$$

Now let *x* be any eigenvector of *G* with a corresponding eigenvalue λ so that $Gx = \lambda x$ or $(\lambda I - G)x = 0$. Furthermore, let *y* be the vector defined as $y = (I - G)x = (1 - \lambda x)$, note that $x, y \in C^{nxn}$. Therefore from (1) it can be seen that:

$$Qy = Ax \qquad (2)$$

So,

$$\begin{aligned}(Q-A)y &= A(x-y)\\ &= A[x-(I-G)x]\\ &= A[x-x+Gx]\\ &= \lambda Ax \end{aligned} \qquad (3)$$

To proceed with the proof we will make use of the inner product which is defined as $(x,y)=\sum_i x_i\bar{y}_i$ where the overbar denotes the complex conjugate. With this definition in hand we introduce the following expressions:

$$(Qy,y)=(Ax,y) \qquad (4)$$

$$(y,(Q-A)y)=(y,\lambda Ax) \qquad (5)$$

In obtaining the above equalities we used equations (2) and (3) . Now rewrite equation (4) as follows:

$$\begin{aligned}(Qy,y)&=(Ax,y)\\ \left((\omega^{-1}D+L)y,y\right)&=(Ax,(1-\lambda)x)\\ \omega^{-1}(Dy,y)+(Ly,y)&=\left(1-\bar{\lambda}\right)(Ax,x)\end{aligned} \qquad (6)$$

In deriving this equation we used the definition of *Q*, the definition of *y*, and the definition of inner product. Next we rewrite equation (5)

$$\begin{aligned}(y,(Q-A)y)&=(y,\lambda Ax)\\ (y,Qy)-(y,Ay)&=(1-\lambda)\bar{\lambda}(x,Ax)\\ (y,Qy)-(y,Qy)-(y,(1-\omega^{-1})Dy)-(y,Uy)&=(1-\lambda)\bar{\lambda}(x,Ax)\\ (\omega^{-1}-1)(y,Dy)-(y,Uy)&=(1-\lambda)\bar{\lambda}(x,Ax)\end{aligned} \qquad (7)$$

In obtaining (7) we used the properties of the inner product, the definition of *y*, and the splitting of *A*. We can now add equations (6) and (7) however, before we do note that by requiring that *A* be Hermitian we have the relations: $(Dy,y)=(y,Dy)$, $(Ly,y)=(y,Uy)$, and $(Ax,x)=(x,Ax)$, and finally *D* must be Hermitian as it is the diagonal of *A*. With these relations in mind we add (6) and (7) to get:

$$\begin{aligned}(2\omega^{-1}-1)(Dy,y)&=(1-\bar{\lambda}+\bar{\lambda}-|\lambda|^2)(x,Ax)\\ (2\omega^{-1}-1)(Dy,y)&=(1-|\lambda|^2)(x,Ax)\end{aligned} \qquad (8)$$

Now let us examine equation (8), the first term on the LHS must be positive because we assumed that $\omega<2$ implying $2\omega^{-1}-1>0$. The second term on the LHS must also be positive, this comes from the fact that *A* is positive definite and therefore $(Ae_i,e_i)=D_{ii}>0$. Furthermore, $(Dy,y)=\sum_i D_{ii}y_i\bar{y}_i$ and because $y_i\bar{y}_i>0$ for all *i*, this expression is positive. From this we see that the LHS is positive.

Consider the second term on the RHS of equation (8), because *A* Hermitian, then from the definition of positive definite this quantity is positive. Finally this leads us to the logical conclusion that for the RHS to be positive (which it must since the LHS is), then the following must be true:

$$1 - |\lambda|^2 > 0$$
$$|\lambda| < 1 \qquad (9)$$

Because *x* was a general eigenvector of *G* we see that the spectral radius of *G* is less than one, $\rho(G) < 1$.We now have the crucial property needed to prove convergence.

Next recall a property of Hermitian matrices is that their eigenvectors are orthogonal and span the space C^n. Therefore we can express the initial error vector in terms of the eigenbasis of *G*, $\{g_i\}$.

$$e^0 = \sum_i c_i g^i$$

Where c_i is a constant. Recalling the recurrence relation for the error formula, we can write the *k*-th error vector as

$$\begin{aligned} e^k &= G^k \sum_i c_i g^i \\ &= \sum_i c_i G^k g^i \\ &= \sum_i c_i {\lambda_i}^k g^i \end{aligned} \qquad (10)$$

Here λ_i is the eigenvalues of *G* corresponding to the eigenvector g^i. Taking the norm of both sides (10) yields

$$\|e^k\| \leq \rho(G)^k \sum_i |c_i| \|g^i\|$$
$$\|e^k\| \leq C\rho(G)^k \qquad (11)$$

Therefore, because of (9) we may complete the proof by saying

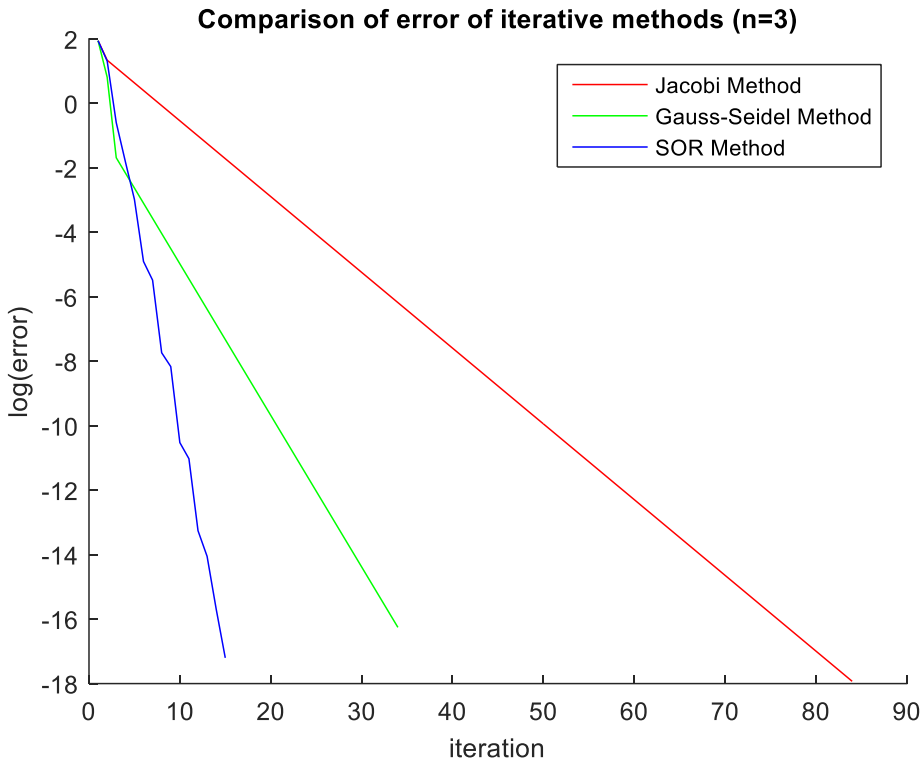
$$\lim_{k\to\infty} \|e^k\| = 0 \qquad (12)$$

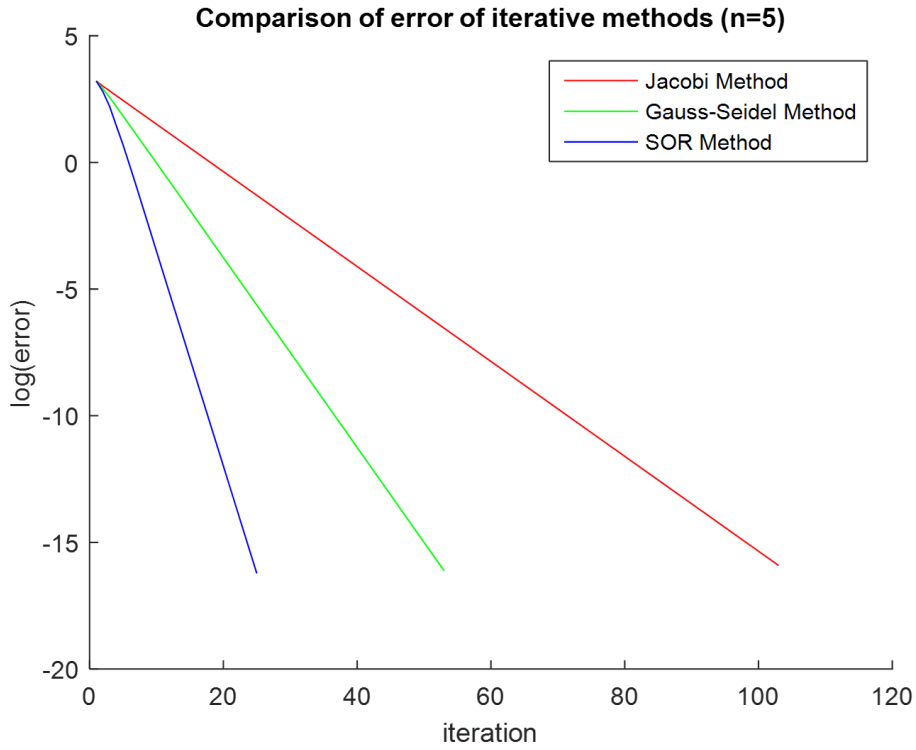
Thus we see that the method converges to the solution *x*, equivalent to saying the error vanishes, and we see the error at each iteration is of the order $\rho(G)^k$.

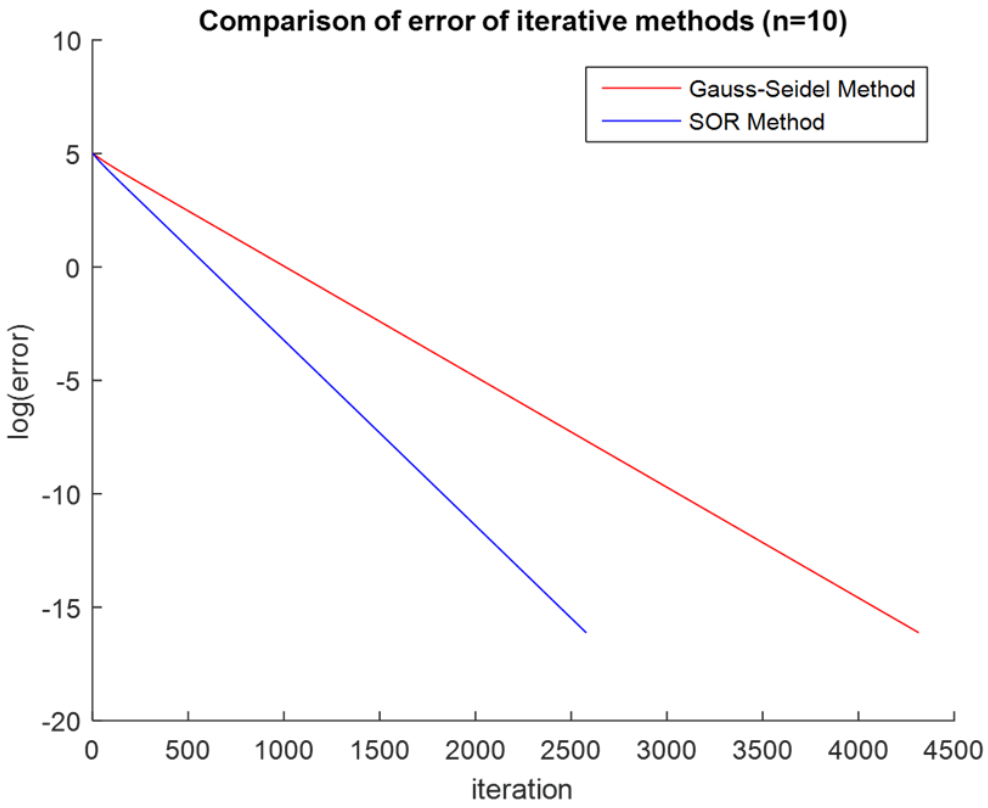
Now that we have convergence and know how the error behaves at each iteration we can make some general comments about the method. The speed at which the method converges depends heavily on how large or small $\rho(G)$ is. The main idea of SOR is that by judiciously selecting ω, we can make $\rho(G)$ as small as possible and therefore greatly increase the rate of convergence. Unfortunately, there is no general formula for determining what ω should be. Therefore heuristics are often used which provide a "good enough" value. Therefore the SOR method offers an improvement, in the sense of convergence rate, over the Gauss-Seidel and Jacobi method.

To conclude we present a comparison of each of the three iterative methods that have been discussed. We used each of the three methods to solve the linear system $Ax = b$ for three systems of size 3x3, 5x5, and 10x10. All methods were initiated with the same starting vector x^0

and iterations were performed until the 2-norm of consecutive iterations fell below the tolerance value, i.e. when $\|x^{k+1} - x^k\| < 10^7$.







One point of interest about these plots are that our result regarding the behavior of error can be corroborated by noting that in these plots the logarithm of the error is linear with respect to the iteration. Furthermore the slope of the logarithm of the error is bounded by the spectral radius of the reduction matrix. Finally, we note that we used a constant value $\omega = 1.25$ in each of the three examples. To examine how convergence is altered when ω is changed, we repeated the solution of the 10x10 system for a range of ω within $(1,2)$. The results are reported in the table below:

ω	# of Iterations
1.1	3526
1.2	2868
1.3	2309
1.4	1829
1.5	1409
1.6	1036
1.7	693
1.8	283
1.9	430

We make the final comment that the value of ω clearly has a large effect on the number of iterations required for convergence and that $\omega \approx 1.8$ corresponds to the optimal relaxation factor and at this value $\rho(G)$ is minimized for this value.

To conclude, we have presented three of the traditional iterative methods for solving linear systems. The benefits of these methods over direct methods are greatest when the linear system is very large and sparse. We provided worked examples demonstrating how these methods determine more accurate approximations at each iteration. Furthermore, since these methods are for implementation on a computer we provided the (pseudo)code that can be used to solve linear systems computationally. Finally, comparison between the how many iterations each methods requires before reaching a solution was presented. This comparison highlighted the improvement of each method over the previous in terms of increasing the convergence rate.

Bibliography

[1] M. Benzi, The Early History of Matrix Iterations, 2009, https://www.siam.org/meetings/la09/talks/benzi.pdf

[2] K. Vuik, History of Mathematicians, http://ta.twi.tudelft.nl/users/vuik/burgers/hist.html

[3] Briggs, Keith. "Diagonally Dominant Matrix." From MathWorld--A Wolfram Web Resource, created by Eric W. Weisstein. http://mathworld.wolfram.com/DiagonallyDominantMatrix.html

[4] David M. Strong, "Iterative Methods for Solving [i]Ax[/i] = [i]b[/i] - Jacobi's Method," Loci (July 2005) http://www.maa.org/publications/periodicals/loci/joma/iterative-methods-for-solving-iaxi-ibi-jacobis-method

[5] Black, Noel; Moore, Shirley; and Weisstein, Eric W. "Jacobi Method." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/JacobiMethod.html

[6] J.E. Pasciak, "Norms and the Jacobi Method", 2012, http://www.math.tamu.edu/~pasciak/classes/639/l510.pdf

[7] University of South Florida, *Gauss-Seidel Method: Theory*, https://www.youtube.com/watch?v=Shh1ZV1I7eE

[8] Briggs, Keith. "Diagonally Dominant Matrix." From *MathWorld*--A Wolfram Web Resource, created by Eric W. Weisstein.http://mathworld.wolfram.com/DiagonallyDominantMatrix.html

[9] Karniadakis, George. "Convergence Analysis of Iterative Solvers."*Convergence Analysis of Iterative Solvers*. N.p., 22 Oct. 2000. Web. 12 May 2015. http://www.cfm.brown.edu/people/gk/chap7/node22.html

[10] "Iterative Methods." Duke University, n.d. Web. 12 May 2015.https://stat.duke.edu/courses/Spring12/sta376/lec/iterative.pdf

[11] University of South Florida, *Gauss-Seidel Method: Example*, https://www.youtube.com/watch?v=ajJD0Df5CsY

[12] D. M. Young, "Iterative Methods for Solving Partial Difference Equations of Elliptic Type," Ph.D. dissertation, Dept. of Math., Harvard University, Cambridge, Mass., 1950

[13] David M. Strong. *Iterative Methods for Solving Ax=b – The SOR Method*. http://www.maa.org/publications/periodicals/loci/joma/iterative-methods-for-solving-iaxi-ibi-the-sor-method

[14] R.L. Burden and J.D. Faires. *Iterative Techniques in Matrix Algebra – Relaxation Techniques for Solving Linear Systems*. 2011. https://www.math.ust.hk/~mamu/courses/231/Slides/CH07_4A.pdf

[15] J.E. Pasciak, More on the Successive Over Relaxation Method, 2012, http://www.math.tamu.edu/~pasciak/classes/639/l510.pdf