CS 510 Final Project Report

Name: Shang-Hao Huang
Netid: sh1384

Date: April 2021

1 Problem Statement

Compare the three iterative methods for solving a linear system of equations, which can be represented as $\mathbf{Ax} = \mathbf{b}$, and how they compare for various sizes of general matrices and how they affect the convergence of methods. In particular, find a good value parameter in the solution to optimize the convergence rate. Then, find a large practical data set and make a numerical comparison of the algorithms.

2 Introduction

There are two approaches regarding solving a linear system, the direct methods try to compute the exact solution by a finite number of operations, while the iterative methods start with an initial solution and then attempt to improve the approximate solution through iterations until it converges or some measure of error reaches the termination criteria. Traditionally, there are two types of error measurement:

1. **Forward Error** is the absolute difference between the true $\mathbf{x}$ and the approximate $\mathbf{x}^*$; that is, $|\mathbf{x}^* - \mathbf{x}|$.
2. **Backward Error** is how much the data must be perturbed to produce the approximate solution, i.e., $|\mathbf{b} - \mathbf{Ax}^*|$.

Then, three iterative methods for solving linear systems are briefly introduced. Note that we write $A = D + L + U$, where D is the diagonal component, L and U are strictly lower and upper triangular components.

1. **Jacobi Method:** Start with the initial guess, $\mathbf{x^{(0)}}$. Then, at k-th iteration, we compute $\mathbf{x}^{(k+1)}$ as follows:

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

Note that Jacobi does not guarantee convergence, except A is strictly diagonally dominant.

2. **Gauss-Seidel:** It$'$ a modification of the Jacobi Method. In Jacobi method, all x_i's calculated in the n-th iteration remains the same until the whole $(n + 1)$-th iteration has been completely calculated. However, in Gauss-Seidel Method, the newly obtained x_i is used immediately to obtain the x_{i+1}. At k-th iteration, $\mathbf{x}^{(k+1)}$ is computed as follows:

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

Similar to Jacobi Method, Gauss-Seidel guarantees convergence when A is strictly diagonally dominant; otherwise, it does not.

3. **SOR (Successive over-relaxation):** SOR is a variant of the Gauss-Seidel to obtain a faster convergence. It achieves this by incorporating a extrapolation factor ω into the Gauss-Seidel method, which appears in the form of weighted average of the previous iterate and the newly computed iterates. At k-th iteration, $\mathbf{x}^{(k+1)}$ is computed as follows:

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

   The idea is to determine a value for ω that will accelerate the rate of convergence. Note that When $\omega = 1$, SOR is equivalent to Gauss-Seidel.

For this project, we use MATLAB to implement the above three methods as well as the experiment part.

3 Dataset

To compare all these three iterative methods, we need to carefully select the matrix A such that they are all convergent for the linear system. For the experiment, we use the MATLAB function "gallery()" [2] to generate the test matrices. The following are the types of matrices that we′re interested in and the command used to generate each of them:

1. **Diagonally dominant matrix** guarantees convergence for all methods. For this type of matrix, we can use "A = gallery('dorr',n,θ)" to generate it. It returns an n-by-n, row diagonally dominant, tridiagonal matrix that is ill conditioned for small nonnegative values of θ. By default, $\theta = 0.01$. Note that all methods guarantee convergence if **A** is symmetric and positive definite.
2. **Positive definite matrix** works in most of the cases, but not for the Jacobi method. For positive definite matrix, we use "A = gallery('tridiag',n,c,d,e)", which returns the sparse tridiagonal matrix of size n-by-n with subdiagonal elements c, diagonal elements d, and superdiagonal elements e. By default, $c = e = -1$ and $d = 2$.
3. **Symmetric matrix:** we use the same command provided in "2." to generate this type of matrix since it is also symmetric.

Regarding the **x**, we simply use built-in function "rand()" in MATLAB to generate it. Then, we just calculate **Ax** to obtain **b**.

4 Method

In this section, we determine what factors in the linear system might affect the performance of these three iterative methods, which allows us to compare them from different perspectives:

1. **Relaxation Factor** ω: As mentioned in the lecture, different choices of ω may cause SOR to function differently. For example, $\omega > 1$ (over-relaxation) means we penalize the previous iterates while $\omega < 1$ (under-relaxation) means we favor them. Typically, $\omega \in (1, 2)$, and thus, we want to see the impact that different choices of relaxation factor may have on the performance of the SOR Method.
2. **Order of A**: Intuitively speaking, a matrix of greater dimension will require more iterations to stabilize. When using a matrix version notation, a higher-dimensional matrix A means a larger number of equations in the linear system.

3. **Initial Iterate $\mathbf{x^{(0)}}$**: Different initial iterates may affect convergence differently. Some good initial guesses may lead to fewer number of iterations while others may make the algorithm difficult to stabilize. In this project, we experiment on the following three initilization strategies:
    - $\mathbf{x^{(0)}}$ is all zeros.
    - $\mathbf{x^{(0)}}$ is all ones.
    - $\mathbf{x^{(0)}} = D^{-1}\mathbf{b}$.
4. **Types of A**: We can see certain types of matrix A guarantee the convergence of the methods while others don′t. Thus, we believe the properties that matrix A possesses definitely will affect the performance of each method. In particular, there are several types of matrix that we're interested in, including Diagonally Dominant Matrix, Positive Definite Matrix, and Symmetric Matrix.

5 Evaluation Metrics

The following are the evaluation metrics used to measure the performance of each method:

1. **Number of Iterations:** An iterative method is usually more preferable if it requires fewer iterations for an iterative method to reach the convergence (or obtain an error of less than a pre-selected value), since this means that method become stabilized faster than others.
2. **Absolute Backward Error:** In this project, we use absolute backward error with the infinity norm as thte measure of error, $||\mathbf{b} - \mathbf{A}\mathbf{x}^*||_\infty$. Each time we get the approximate root $\mathbf{x}^*$ at the end of iterative process, we compute the backward error.

6 Experiment & Result

In each experiment, for any given size of matrix $\mathbf{A}$, we run each iterative method 50 times, compute the average of the number of iterations and the absolute backward error, and then compare the three methods using a plot. Regarding the parameter setting for all experiments:

1. We use the infinity norm of the difference between any two consecutive iterates as the measure of error. That is, an algorithm terminates when $||\mathbf{b} - \mathbf{A}\mathbf{x}^*||_\infty < \epsilon$, where $\epsilon = 10^{-5}$ is the prespecified threshold.
2. The maximum number of iterations allowed is set to 10^5.
3. The default initial iterate $\mathbf{x^{(0)}}$ is $D^{-1}\mathbf{b}$.
4. From the experiment where we′re trying to understand how different choices of ω may affect the performance of the SOR Method, we can determine a reasonably good value for ω. Then, this value will be used for the other experiments, such as comparing the convergence rate for all three methods.

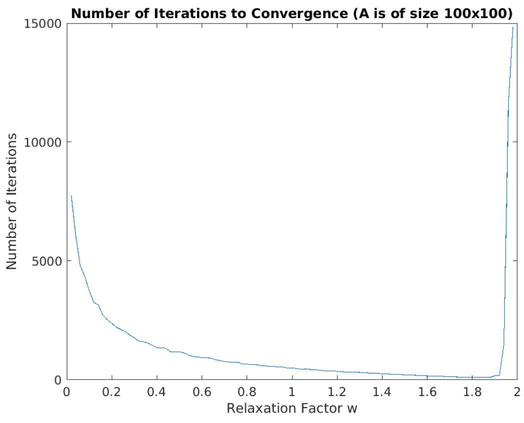
Note that, to observe the sole effect of a tested variable in an experiment, such as how the size of $\mathbf{A}$ affect the performance of a method, we keep all the others variables, such as type of $\mathbf{A}$ and initial iterate $\mathbf{x^{(0)}}$, unchanged throughout the experiment.

6.1 Determine Relaxation Factor ω for SOR Method

In the first experiment, I try to find a reasonably good value of ω. A theorem by Kahan shows that SOR fails to converge when $\omega \notin (0, 2)$ [3], so we only test *omega* within that interval. The experiment proceeds as follows:

- For $\omega = 0.02, 0.04, 0.06, \cdots, 1.98$.
  1. Generate a diagonally dominant matrix $\mathbf{A}$ of size $n \times n$ using "A = gallery('dorr',n,$\theta = 0.01$)".
  2. For each $\mathbf{A}$, use "rand()" to generate $\mathbf{x}$ of size $n \times 1$, and compute $\mathbf{b} = \mathbf{A} * \mathbf{x}$.
  3. Solve it using the SOR method with the given ω.
  4. Repeat step 3 & 4 50 times, and compute average of the two metrics.

We$'$ve run the same test for $n = 25, 50, 100$. The conclusions from the plots of different n are consistent, so we show only the resultss obtained from $n = 100$ as an example. From Fig. 1, we can observe that the number of iterations to convergence decreases as the ω increases until ω is greater than 1.9, where the minimum number of iterations to convergence is around 100 at $\omega = 1.82$. Yet, from the error plot, we see that the absolute backward error decreases from 190 to 0.0015 as ω goes $0 \to 1$ and then it increases as ω goes $1 \to 2$.



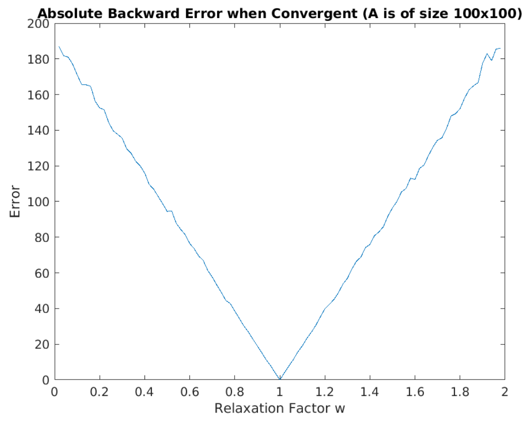


Figure 1: The Number of Iterations and Absolute Backward Error for SOR when $\omega \in (0.02, 1.98)$ given $\mathbf{A}$ is of size 100×100

Since, at $\omega = 1$, SOR simplifies to Gauss-Seidel and has the smallest error given that it does not converge the fastest, we want to further explore the performance of SOR when $\omega \in (0.9, 1.1)$, with finer step size $= 0.002$, to determine a value of ω that maintain a good balance between convergence rate and the error. As shown in Figure 2, the SOR at $\omega = 1.094$ (421 iterations) converges about 15% faster than that at $\omega = 1.0$ (487 iterations). However, the error at $\omega = 1.094$ (18.2) is much higher than that at $\omega = 1$ (0.0015). Therefore, we think $\omega = 1.024$ would be a good choices as it requires 5% fewer iterations to converge with error ≈ 4, and we decide to use this value in all remaining experiments involving the SOR method.

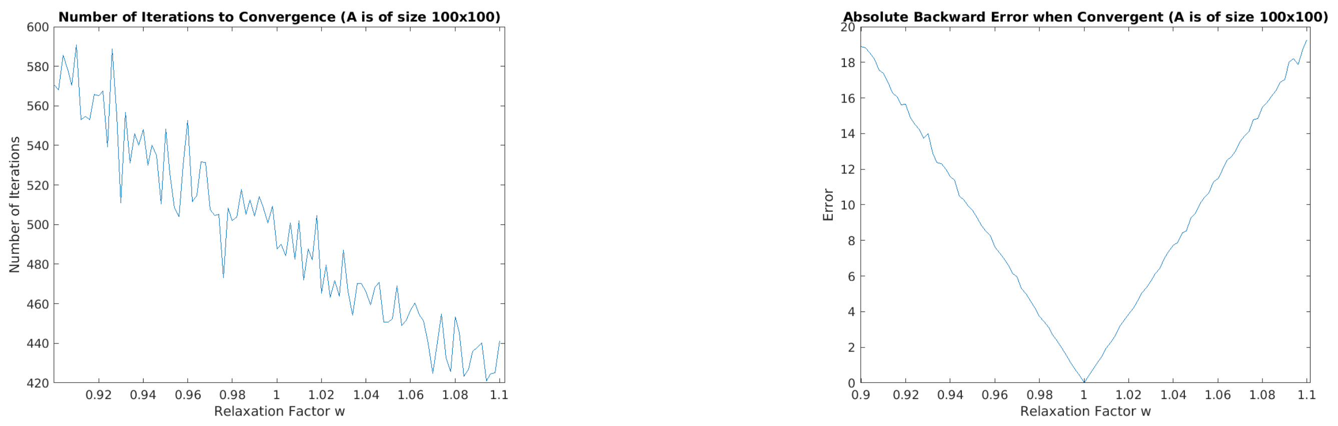


Figure 2: The Number of Iterations and Absolute Backward Error for SOR when $\omega \in (0.9, 1.1)$ given $\mathbf{A}$ is of size 100×100

6.2 Comparison of Different Sizes of A

In this experiment, we want see how the size of **A** affect the behavior of convergence of each method. The experiment consists of the following steps:

- For $n = 10, 15, 20, \cdots, 100$.

  1. Generate a diagonally dominant matrix **A** of size $n \times n$ using "A = gallery('dorr',n,$\theta = 0.01$)".
  2. For each **A**, use "rand()" to generate **x** of size $n \times 1$, and compute $\mathbf{b} = \mathbf{A} * \mathbf{x}$.
  3. Solve it using all three methods.
  4. Repeat step 3 & 4 50 times, and compute average of the two metrics.

The result is shown in Figure 3. As expected, the error increases as the order of **A** increases, where the Gauss-Seidel has smallest error. In the convergence rate plot, we can observe that, for $n \leq 30$, the number of iterations increases as the order n of **A** increases as hypothesized. However, when $n > 30$, both SOR and Gauss-Seidel start to converge faster. This is against our intuition and we're still trying to find a way to explain this situation.

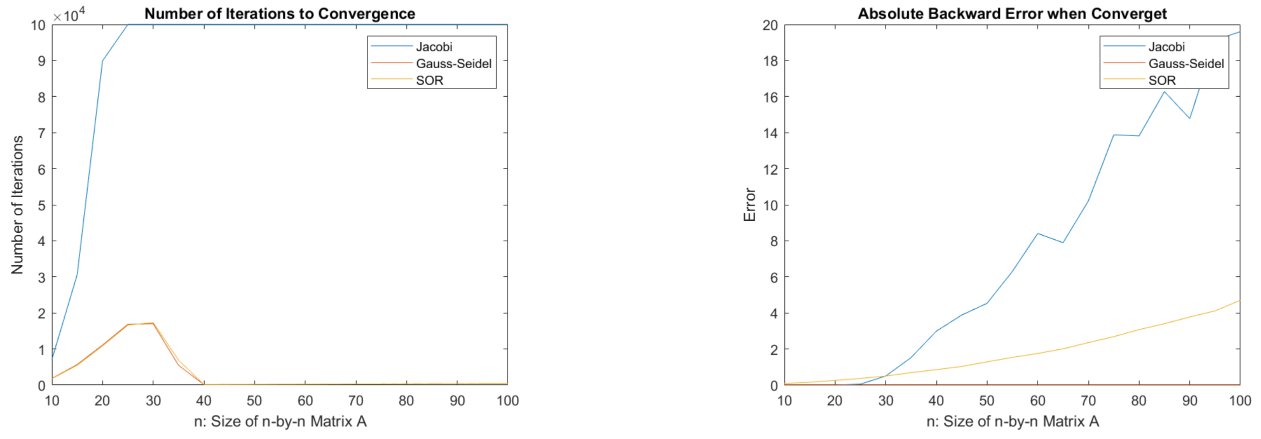


Figure 3: Comparison of Number of Iterations and Absolute Backward Error for Different Order of **A** (Diagonally Dominant)

6.3 Comparison of Different Initialization Strategies

As discussed previously, we′d like to compare different initialization strategies for each method. The following is the procedure for this experiment:

- For $n = 10, 15, 20, \cdots, 100$.

  1. Generate a positive definite matrix $\mathbf{A}$ of size $n \times n$ using "A = gallery('tridiag',n)".
  2. For each $\mathbf{A}$, use "rand()" to generate $\mathbf{x}$ of size $n \times 1$, and compute $\mathbf{b} = \mathbf{A} * \mathbf{x}$.
  3. Solve it using all three methods with three different $\mathbf{x^{(0)}}$.
  4. Repeat step 3 & 4 50 times, and compute average of the two metrics.

The result is shown in Figure 4 and Figure 5. In Figure 4, we see that all three methods converge consistently regardless of the $\mathbf{x^{(0)}}$. Yet, there exists little difference in the Jacobi method for larger $\mathbf{A}$, where it requires slightly more iterations to converge when $\mathbf{x^{(0)}} = \mathbf{D^{-1}b}$ than when $\mathbf{x^{(0)}} = \mathbf{1}$ and $\mathbf{x^{(0)}} = \mathbf{0}$, where $\mathbf{x^{(0)}} = \mathbf{0}$ has the fastest convergence. Similarly, in Figure 5, we can see that all methods are consistent across different strategies. Again, the only yet slight inconsistency happens in the Jacobi method, where it has a smaller error when $\mathbf{x^{(0)}} = \mathbf{D^{-1}b}$ and a larger error when $\mathbf{x^{(0)}} = \mathbf{0}$.

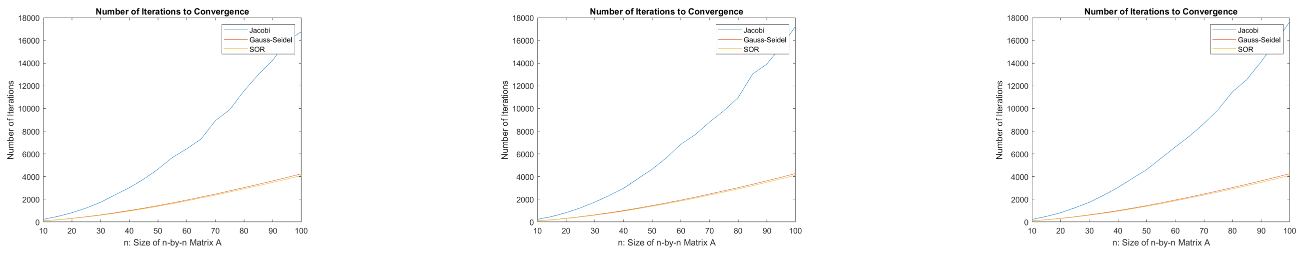


Figure 4: Compare Number of Iterations for Different Initial Iterate $\mathbf{x^{(0)}} = \mathbf{0}$, $\mathbf{1}$, $D^{-1}\mathbf{b}$ (left to right)

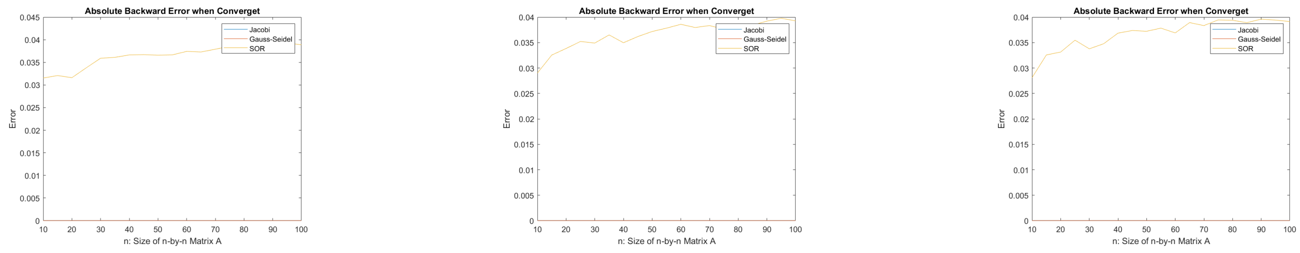


Figure 5: Compare Error for Different Initial Iterate $\mathbf{x^{(0)}} = \mathbf{0}$, $\mathbf{1}$, $D^{-1}\mathbf{b}$ (left to right)

6.4 Comparison of Different Types of A

6.4.1 A is Positive Definite

For all methods to be convergent, $\mathbf{A}$ has to be either diagonally dominant or symmetric and positive definite. To observe the sole effect of $\mathbf{A}$ is positive definite on each method, we have to make $\mathbf{A}$ diagonally dominant while being able to make it positive definite and non-positive definite. But we could not achieve that, and thus, for this experiment, we compare all three methods when $\mathbf{A}$ is positive definite matrix (using "A = gallery('tridiag',n-1,2,-1)"). The result is presented in Figure 6. As can be seen in Figure 6, SOR converges slightly faster than Gauss-Seidel but with a much larger error. And the Jacobi converges much more slower than the others while it has nearly the same error as the Gauss-Seidel.

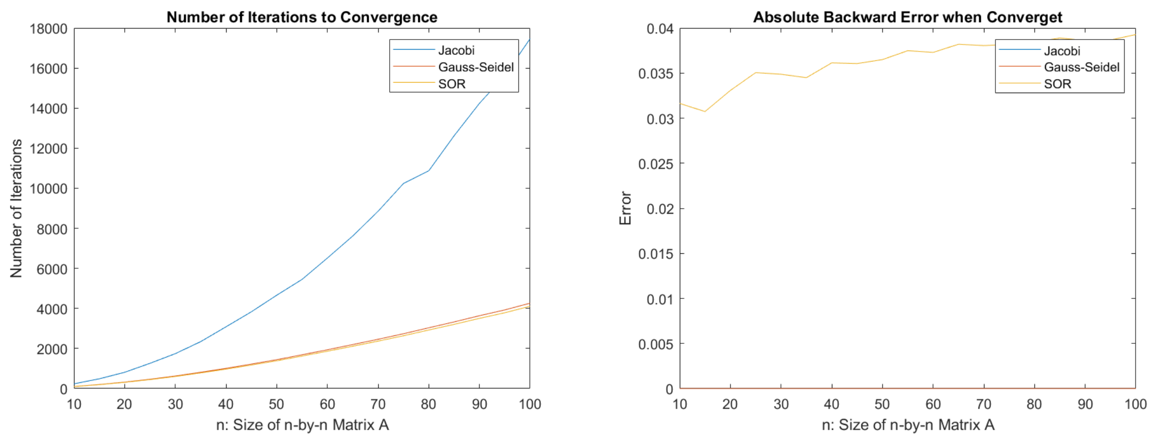


Figure 6: Comparison of Number of Iterations and Absolute Backward Error for Different Order of Symmetric Positive Definite **A**

6.4.2 If A is Symmetric or not

Similarly, to observe the sole effect of **A** is symmetric or not on each method, we have to make **A** diagonally dominant while being able to make it both symmetric and nonsymmetric. Though we could not do so, we could do that when **A** is positive definite, yet, in that case, the Jacobi method does not guarantee convergence. As a result, for this experiment we only compare the Gauss-Seidel and SOR. To make **A** unsymmetric, we can use "A = gallery('tridiag',n,-1,2,1)". The result for **A** being non-Symmetric Positive Definite matrix is shown in Figure 7. Comparing both Figure 6 and 7, surprisingly, as **A** becomes nonsymmetric, we see that the SOR method becomes divergent while the Gauss-Seidel method remains the same in terms of convergence rate and error.

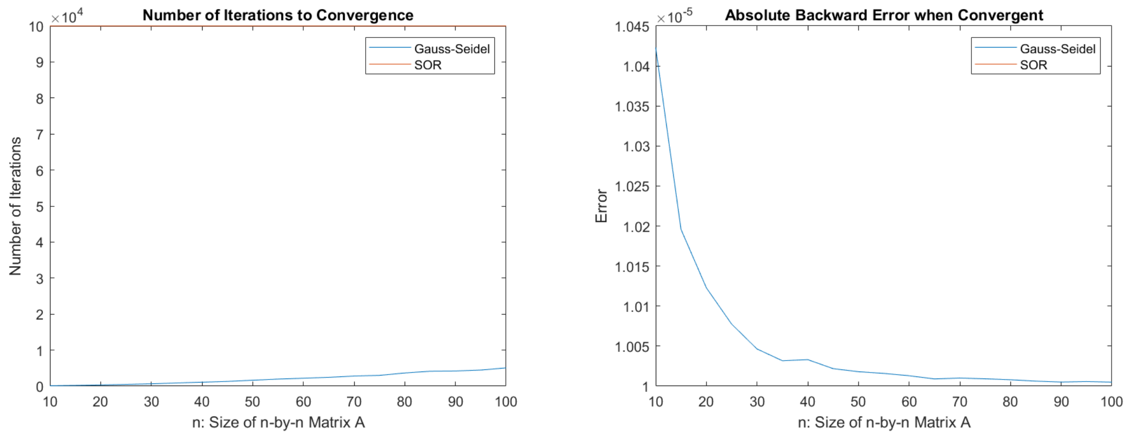


Figure 7: Comparison of Number of Iterations and Absolute Backward Error for Different Order of non-Symmetric Positive Definite **A**

7 MATLAB Implementation

The implementation of the three iterative methods are saved in "jacobi.m", gauss-seidel.m", and "sor.m". For the experiment, "test-omega.m" contains the code used for the experimentation with values of ω, "comparison_initial_iterates.m" contains the code for the experimentation with different initialization strategies, and "comparison_all_sizes.m" contains the code for all remaining experiments.

8 Conclusions & Future Work

In summary, the following are some insights we′ve found during this project regarding the convergence of the three iterative methods to solve a linear system:

1. In most of our test cases, SOR converges the fastest while Gauss-Seidel has the smallest absolute backward error. The Jacobi method always converges the slowest yet it sometimes gives smaller error than SOR.
2. For SOR method, greater relaxation factor ω usually converges faster, except when it is very close to 2.
3. $\mathbf{A}$ of larger order results in greater absolute backward error.
4. Though at first we believe bad initial iterate may strongly impact the performance of a method, in our experiments, all three method performs surprisingly consistently in convergence rate and error, except that there′s a little inconsistency in the Jacobi Method.
5. For symmetric positive definite $\mathbf{A}$, SOR converges slightly faster thatn Gauss-Seidel but yields larger error.
6. SOR may be divergent when A is positive definite but nonsymmetric.

Due to time constraints, there are still aspects left untouched about the convergence and we′d like to explore more in the future:

1. How the density of matrix $\mathbf{A}$ impact the behavior of convergence for each method.
2. A more systematic way to compute ω. It′s difficult to determine ω for the SOR method as it′s not possible to compute in advance [1] which will maximize the convergence rate of SOR. It′s also hard to choose the value ω that maintain a good balance between convergence rate and error.
3. Different types matrix $\mathbf{A}$, such as Q-matrix and Z-matrix.
4. $\mathbf{A}$ of much higher dimension, such as 5000×5000.

References

[1] Noel Black and Shirley. Moore. "Successive Overrelaxation Method." From MathWorld–A Wolfram Web Resource, created by Eric W. Weisstein." In: *https://mathworld.wolfram.com/SuccessiveOverrelaxationMethod.html* (2021).

[2] Desmond J. Higham and Nicholas J. Higham. "The MATLAB® gallery of test matrices is based upon the work of Nicholas J. Higham at the Department of Mathematics, University of Manchester, Manchester, England. Further background can be found in the books MATLAB Guide, 2nd ed, by Desmond J. Higham and Nicholas J. Higham, Philadelphia: SIAM, 2005, and Accuracy and Stability of Numerical Algorithms, by Nicholas J. Higham, Philadelphia: SIAM, 1996." In: *https://www.mathworks.com/help/matlab/ref/gallery.htmlmw*$_5 6263992 - c50b - 48ad - b8bd - d842618dd775$ (2021).

[3] W. Kahan. "Gauss-Seidel Methods of Solving Large Systems of Linear Equations." In: *PhD Thesis, University of Toronto, Toronto.* (1958).