Programming Assignment 3

Jenny Petrova

November 25th, 2024

1 Executive Summary

This report presents several iterative methods to approximate solutions to a linear system of equations. Routines are developed to observe the behaviors of Richardson's First Order Stationary Stationary (RF) method, the Steepest Descent (SD) method, and the Conjugate Gradient (CG) method on the eigen-coordinate system defined by the eigenvectors of A, when $A \in \mathbb{R}^{n\times n}$ is a symmetric positive definite (SPD) matrix. We investigate the implementation of the Jacobi, (Forward) Gauss-Seidel (GS), and Symmetric Gauss-Seidel (SGS) stationary methods on several matrices $A \in \mathbb{R}^{n\times n}$ with varying properties (SPD, diagonally dominant, tri-diagonal). Lastly, these methods are scaled to large, sparse symmetric matrices stored efficiently in the compressed row storage (CSR) matrix representation. We consider the theoretical results of each algorithm, and test the algorithms on problems for which we know the solution (i.e. problems where convergence to the true solution is guaranteed). We will provide a comprehensive comparison of the iterative methods examined and discuss proper selection of iterative methods based on the matrix properties, problem size, and desired solution accuracy. We will also discuss the performance of each method and identify the most efficient method for various matrix types.

2 Statement of the Problem

Consider a linear system of equations of the form

$$Ax = b, \tag{1}$$

where the matrix $A \in \mathbb{R}^{n\times n}$ and the right-hand side vector $b \in \mathbb{R}^n$ are known. Each row of A and b represent a linear equation, hence the problem represents a system of linear equations. We aim to solve for the vector $x \in \mathbb{R}^n$ satisfying the system. When the matrix A is dense and well-conditioned, one such approach to solving the system is using direct methods.

Direct methods are algorithms that produce an exact solution in a finite number of operations (e.g. Gaussian elimination, LU decomposition, Cholesky decomposition, and QR decomposition). Although these methods solve the linear system with high degree of precision, they can be slow and computationally

expensive for large linear systems, particularly when A is large and sparse and has no structure that can be exploited in a factorization. So we introduce iterative methods.
Iterative methods use an initial "guess" (denoted x_0) to compute a sequence of improving approximate solutions to the system. The iteration process terminates when we satisfy some predefined criterion, i.e. when the approximated solution vector converges to the true solution. Note that iterative methods are guaranteed to converge when A is SPD, but convergence is not guaranteed otherwise. Thus we will examine several iterative methods and explore both the cases where the solution converges and does not converge.

3 Description of the Mathematics

In this section, we review necessarily terminology required for the remainder of the report.

3.1 Eigenvalues and Eigenvectors

Let $A \in \mathbb{R}^{n\times n}$. We say $\lambda \in \mathbb{R}$ is an **eigenvalue** of A if there exists a vector $v \in \mathbb{R}^n$, $v \neq 0$ such that $Av = \lambda v$. Then v is the corresponding **eigenvector** to λ. The matrix A has at most n distinct eigenvalues and linearly independent eigenvectors.
The **spectral radius** of A, denoted $\rho(A)$, is the magnitude of the largest eigenvalue of A, i.e.

$$\rho(A) = \max_{1\leq i\leq n} |\lambda_i|. \tag{2}$$

3.2 Symmetry

Let $A \in \mathbb{R}^{n\times n}$. If $A = A^T$, then A is **symmetric**. If A is symmetric, then A has a set of real orthonormal eigenvectors, denoted q_i, and real eigenvalues λ_i, $1 \leq i \leq n$, such that we can factorize A into

$$A = Q\Lambda Q^T, \tag{3}$$

where $Q = [q_1, \ldots, q_n]$ and $\Lambda = diag(\lambda_1, \ldots, \lambda_n)$. We note that Q is an orthogonal matrix such that

$$Q^TQ = QQ^T = I, \tag{4}$$

hence $Q^T = Q^{-1}$. Now if A is symmetric and $|\lambda_1| \geq \cdots \geq |\lambda_n|$, then

$$\rho(A) = |\lambda_1| = ||A||_2. \tag{5}$$

If $A \in \mathbb{R}^{n\times n}$ is symmetric, then A is a **convergent** matrix if and only if

$$\rho(A) = ||A||_2 < 1. \tag{6}$$

We say that A is **symmetric positive definite** (SPD) if $A = A^T$ and

$$v^T Av > 0 \tag{7}$$

for all $v \in \mathbb{R}^n$, $v \neq 0$. If A is SPD, then it is nonsingular and it defines an inner product and vector norm, i.e.

$$\langle v, v \rangle_A = v^T Av = ||v||_A^2 = ||A^{\frac{1}{2}} v||_2. \tag{8}$$

3.3 Iterative Methods and Convergence

Let $A \in \mathbb{R}^{n \times n}$. We want to find the true solution x to the linear system $Ax = b$. For each iterative method, we must generate an initial guess x_0. Our initial guess provides us the residual vector $r_0 = b - Ax_0$. We use the residual vector to better approximate the solution. Each iteration k modifies the previous value of the approximate solution vector, x_k, until the solution converges (the residual r_{k+1} is sufficiently small). Consider a sequence of iterates of the form

$$x_{k+1} = x_k + \alpha r_k, \tag{9}$$

where

$$r_k = b - Ax_k \tag{10}$$

is the residual vector at iteration k. Our choice of the step-size α depends on A, and determines convergence. When α is constant, we call the iterative method **stationary**. A method is called **non-stationary** when α changes at each iteration. We will discuss the specific choices of α further in Section 4. At each iteration, the error difference between our current value of x_k and the true solution x is defined by

$$e^{(k)} = x_k - x. \tag{11}$$

Combining the above equations, we examine the behavior of the error terms by showing there exists a matrix $G \in \mathbb{R}^{n \times n}$ such that

$$e^{(k+1)} = (I - \alpha A)e^{(k)} = G^k e^{(k)}, \tag{12}$$

where I is the identity matrix. The residuals behave the same way. We can show

$$r_{k+1} = (I - \alpha A)e^{(k)} = G^k r_k. \tag{13}$$

Hence both vectors converge ($e^{(k+1)} \to 0$, $r_{k+1} \to 0$) when a proper α is chosen. We call the matrix G the **iteration matrix**. Therefore

$$G = I - \alpha A \tag{14}$$

and a sufficient condition for convergence is

$$\rho(G) = ||G|| < 1. \tag{15}$$

If $\rho(G) > 1$ then there exists at least one x_0 such that the sequence $\{x_k\}$ does not converge to the true solution x. We can set a bound on how accurate our

final solution x_{k+1} will be after approximation by the **condition number**, κ, of a matrix. We define the condition number by

$$\kappa = \frac{\lambda_{max}}{\lambda_{min}}, \tag{16}$$

where λ_{max} is the largest eigenvalue of A and λ_{min} is the smallest eigenvalue of A.

3.4 Matrix Splitting and Preconditioning

Given a non-singular matrix $A \in \mathbb{R}^{n\times n}$, we can split the matrix a into

$$A = D - L - U \tag{17}$$

where $D = diag(\alpha_{11}, \ldots, \alpha_{nn})$ for $a_{ii} \in A$, and L and U are the strictly lower triangular and upper triangular parts of A. To improve the convergence of an iterative method, we can apply a **preconditioning matrix** $P \in \mathbb{R}^{n\times n}$ to the system such that we solve

$$P^{-1}Ax = P^{-1}b. \tag{18}$$

Let $A \in \mathbb{R}^{n\times n}$ be SPD. If $(P + P^T) - A$ is SPD, then P^{-1} exists and we can construct the iterative method

$$x_{k+1} = x_k + P^{-1}r_k. \tag{19}$$

Now we have $G = I - P^{-1}A$. Our choice of P for solving the preconditioned system can be determined from the matrix splitting form. We will discuss the particular choice of P for different iterative methods in Section 4.

4 Description of the Algorithm and Implementation

The following algorithms are implemented in Python, on the PyCharm integrated development environment. A driver code stores the routines for each method and routines for the generation of test matrices. Each routine is defined as a Python function. The routines for each part are stored in separate driver files. A solver code calls the routines from the driver code, and runs several tests to assess the behavior of the routines on varying matrices.

4.1 Part 1: Richardson's First Order, Steepest Descent, and Conjugate Gradient Methods

Our construction of the algorithms in this part will exploit the fact that we can diagonalize a SPD matrix $A \in \mathbb{R}^{n\times n}$ into $A = Q\Lambda Q^T$ (3), where Q is an

orthogonal $(n \times n)$ matrix. We consider the system

$$\begin{aligned} Ax &= b \\ (Q\Lambda Q^T)x &= b \\ Q\Lambda(Q^T x) &= b \\ \Lambda(Q^T x) &= Q^{-1}b \\ \Lambda\tilde{x} &= \tilde{b}, \end{aligned}$$

where $\tilde{x} = Q^T x$ and $\tilde{b} = Q^T b$. Therefore for developing and examining the performance of the following methods, it is sufficient for us to consider systems of the form $\Lambda\tilde{x} = \tilde{b}$. Let $e^{(k)} = x_k - x$ be the error term for $Ax = b$ at iteration k, and let $\tilde{e}^{(k)} = \tilde{x}_k - \tilde{x}$ be the error term for $\Lambda\tilde{x} = \tilde{b}$ at iteration k. Then we can show

$$e^{(k)} = x_k - x = Q\tilde{x}_k - Q\tilde{x} = Q\tilde{e}^{(k)}. \tag{20}$$

This gives us the equivalence

$$||x_k - A^{-1}b||_2 = ||e^{(k)}||_2 = ||Q\tilde{e}^{(k)}||_2 = ||\tilde{x}_k - \Lambda^{-1}\tilde{b}||_2, \tag{21}$$

meaning the error in the eigenvector coordinates is sufficient for understanding the convergence behavior of the matrix (given that A is SPD), and we do not need to construct Q specifically. Since Λ is a diagonal matrix, we can simplify the computations for our routines by storing the Λ as a vector in $\mathbb{R}^n$. Additionally, since Λ stores the eigenvalues of A, we easily construct Λ by generating positive eigenvalues and storing these values in the vector. We we will directly pass this vector of eigenvalues into the routines for the iterative methods in Part 1.

To examine the convergence behavior of the RF, SD, and CG algorithms, we will construct problems for which the know the solution. We will generate a random solution vector $\tilde{x} \in \mathbb{R}^n$, and use our choice of Λ to compute $\Lambda\tilde{x} = \tilde{b}$. Then we will generate an initial guess to the system, x_0, which we will pass into the iterative routines. The algorithms will compute the iterations until our approximation vector x_k converges to the true solution $\tilde{x}$.

Algorithm 1 Richardson's First Order Stationary Method

Require: $\Lambda \in \mathbb{R}^n$, $\tilde{x} \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$, $\tilde{b} \in \mathbb{R}^n$
Ensure: $\tilde{b} = \Lambda\tilde{x}$ and $\lambda_i > 0$ for all $\lambda_i \in \Lambda$

$\alpha \leftarrow \frac{2}{\lambda_{\min}+\lambda_{\max}}$
$r_0 \leftarrow \tilde{b} - \Lambda x_0$ ▷ Initial residual
$e^{(0)} \leftarrow x_0 - \tilde{x}$ ▷ Initial error
iterations $\leftarrow 0$
while iterations < 1000 **and** $\frac{\|r_k\|_2}{\|r_0\|_2} > 10^{-6}$ **do**
    $x_{k+1} \leftarrow x_k + \alpha r_k$
    $r_{k+1} \leftarrow r_k - \alpha\Lambda r_k$ ▷ Update residual
    Compute $\|r_{k+1}\|_2$
    $e^{(k+1)} \leftarrow x_{k+1} - \tilde{x}$ ▷ Update error
    Compute $\frac{\|e^{(k+1)}\|_2}{\|e^{(k)}\|_2}$ ▷ Error ratio
    iterations $\leftarrow$ iterations $+ 1$
end while
return x_{k+1}, iterations, residuals, errors, error ratios

For Richardson's iteration, the optimal parameter α to ensure fastest convergence is

$$\alpha_{opt} = \frac{2}{\lambda_{min} + \lambda_{max}}. \tag{22}$$

Since α remains constant at each iteration, we see that Richardson's method is stationary. In this case, our iteration matrix is $G = I - \alpha\Lambda$. Hence we can derive the spectral radius

$$\rho(G) = max\{|1 - \alpha\lambda_{max}|, |1 - \alpha\lambda_{min}|\}. \tag{23}$$

If we substitute α_{opt} into the spectral radius, we get

$$\rho_{opt} = \frac{\kappa - 1}{\kappa + 1}, \tag{24}$$

where $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$ (15). The optimal spectral radius ρ_{opt} provides a bound for the convergence of the error terms, and is the damping factor for our algorithm. This means that the error decreases by a factor determined by ρ_{opt} at each iteration:

$$||e^{(k+1)}||_A \leq \frac{\kappa - 1}{\kappa + 1}||e^{(k)}||_A. \tag{25}$$

Therefore the error ratios at each iteration are bounded by

$$\frac{||e^{(k+1)}||_2}{||e^{(k)}||_2} \leq \frac{\kappa - 1}{\kappa + 1}. \tag{26}$$

Algorithm 2 Steepest Descent Method

Require: $\Lambda \in \mathbb{R}^n$, $\tilde{x} \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$, $\tilde{b} \in \mathbb{R}^n$
Ensure: $\tilde{b} = \Lambda\tilde{x}$ and $\lambda_i > 0$ for all $\lambda_i \in \Lambda$

$r_0 \leftarrow b - Ax_0$ ▷ Initial residual
$e^{(0)} \leftarrow x_0 - \tilde{x}$ ▷ Initial error
iterations $\leftarrow 0$
while iterations < 1000 **and** $\frac{\|r_k\|_2}{\|r_0\|_2} > 10^{-6}$ **do**
    $v \leftarrow Ar_k$ ▷ Compute search direction
    $\alpha \leftarrow \frac{r_k^T r_k}{r_k^T v}$ ▷ Compute step size
    $x_{k+1} \leftarrow x_k + \alpha r_k$ ▷ Update solution
    $r_{k+1} \leftarrow r_k - \alpha v$ ▷ Update residual
    Compute $\|r_{k+1}\|_2$
    $e^{(k+1)} \leftarrow x_{k+1} - \tilde{x}$ ▷ Update error
    Compute $\frac{\|e^{(k+1)}\|_2}{\|e^{(k)}\|_2}$ ▷ Error ratio
    iterations $\leftarrow$ iterations $+ 1$
end while
return x_{k+1}, iterations, residuals, errors, error ratios

The Steepest Descent is a non-stationary algorithm. We compute the step size α at each iteration k:

$$\alpha = \frac{r_k^T r_k}{r_k^T v}. \tag{27}$$

As with Richardson's, the damping inequality for the error terms is determined by

$$||e^{(k+1)}||_A \leq \frac{\kappa - 1}{\kappa + 1} ||e^{(k)}||_A. \tag{28}$$

Note that here we examine the A-norm of the error terms.

Algorithm 3 Conjugate Gradient Method

Require: $\Lambda \in \mathbb{R}^n$, $\tilde{x} \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$, $\tilde{b} \in \mathbb{R}^n$
Ensure: $\tilde{b} = \Lambda\tilde{x}$ and $\lambda_i > 0$ for all $\lambda_i \in \Lambda$

$r_0 \leftarrow b - Ax_0$ ▷ Initial residual
$d_0 \leftarrow r_0$ ▷ Initial search direction
$\sigma_0 \leftarrow r_0^T r_0$ ▷ Initialize σ
$e^{(0)} \leftarrow x_0 - \tilde{x}$ ▷ Initial error
iterations $\leftarrow 0$
while iterations < 1000 **and** $\frac{\|r_k\|_2}{\|r_0\|_2} > 10^{-6}$ **do**
    $v \leftarrow Ad_k$ ▷ Matrix-vector product
    $\mu \leftarrow d_k^T v$ ▷ Compute scaling factor
    $\alpha \leftarrow \frac{\sigma_k}{\mu}$ ▷ Step size
    $x_{k+1} \leftarrow x_k + \alpha d_k$ ▷ Update solution
    $r_{k+1} \leftarrow r_k - \alpha v$ ▷ Update residual
    $\sigma_{k+1} \leftarrow r_{k+1}^T r_{k+1}$ ▷ Update σ
    $\beta \leftarrow \frac{\sigma_{k+1}}{\sigma_k}$ ▷ Direction update factor
    $d_{k+1} \leftarrow r_{k+1} + \beta d_k$ ▷ Update search direction
    Compute $\|r_{k+1}\|_2$
    $e^{(k+1)} \leftarrow x_{k+1} - \tilde{x}$ ▷ Update error
    Compute $\frac{\|e^{(k+1)}\|_2}{\|e^{(0)}\|_2}$ ▷ Error ratio
    iterations $\leftarrow$ iterations $+ 1$
end while
return x_{k+1}, iterations, residuals, errors, error ratios

The Conjugate Gradient method is also non-stationary. At each step, we compute

$$\alpha = \frac{\sigma_k}{\mu}. \tag{29}$$

However, for CG, the error terms are bounded by

$$||e^{(k)}||_A \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k ||e^{(0)}||_A. \tag{30}$$

The bound for the error terms changes with each iteration.

4.2 Part 2: Behavior of Stationary Methods Jacobi, Gauss-Seidel, and Symmetric Gauss-Seidel

To investigate the convergence of the following algorithms, we again construct problems for which we know the solution. We generate a solution vector $\tilde{x}$ and compute $\tilde{b} = A\tilde{x}$. In this part, the algorithms will perform iterations on small, dense matrices $A \in \mathbb{R}^{n\times n}$. We will again assume A to be non-singular, in order to have a unique solution to the system. All three methods will converge when the test matrix $A \in \mathbb{R}^{n\times n}$ is diagonally dominant or SPD (this is a sufficient condition to guarantee convergence). For matrices that do not satisfy these conditions, preconditioning can transform the system to improve convergence. The following methods require a preconditioning matrix P. We examine the decomposition $A = D - L - U$ for each method to construct the appropriate preconditioner form such that we can approximate the solution

$$x_{k+1} = x_k + P^{-1}r_k \tag{31}$$

at each iteration, where $r_k = b - Ax_k$.

Algorithm 4 Jacobi Method

Require: $A \in \mathbb{R}^{n\times n}$, $\tilde{x} \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$, $b \in \mathbb{R}^n$
Ensure: $\tilde{b} = A\tilde{x}$
1: Initialize $r_0 \leftarrow b - Ax_0$
2: Initialize *iterations* $\leftarrow 0$
3: **while** iterations < 1000 and $\frac{||x_k - \tilde{x}||_2}{||\tilde{x}||_2} > 10^{-6}$ **do**
4: Update $x_{k+1} \leftarrow x_k + (r_k/D)$
5: Update residual $r_{k+1} \leftarrow b - Ax_{k+1}$
6: Compute relative error $\frac{||x_k - \tilde{x}||_2}{||\tilde{x}||_2}$
7: iterations $\leftarrow$ iterations $+ 1$
8: **end while**
9: **return** x, relative errors, iterations

For the Jacobi method, we use the diagonal of A as the preconditioner, i.e. $P = D$. Hence we have the iteration matrix form $G = I - D^{-1}A$. Since D is a diagonal matrix, we can easily compute the inverse of the matrix (if $a_{ii} \in D$, then $\frac{1}{a_{ii}} \in D^{-1}$) to solve

$$x_{k+1} = x_k + D^{-1}r_k. \tag{32}$$

With the Jacobi method, we also note that only the values of x_k computed in the k^{th} iteration are used to compute the following approximation x_{k+1} in next iteration. This is because all elements in the current solution vector x_{k+1} are updated simultaneously, and each update is independent of others in the same iteration.

Algorithm 5 Gauss-Seidel Forward Method

Require: $A \in \mathbb{R}^{n \times n}$, $\tilde{x} \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$, $b \in \mathbb{R}^n$
Ensure: $\tilde{b} = A\tilde{x}$
Initialize $r_0 \leftarrow b - Ax_0$
Initialize *iterations* $\leftarrow 0$
while iterations < 1000 and $\frac{||x_k - \tilde{x}||_2}{||\tilde{x}||_2} > 10^{-6}$ **do**
    Perform forward sweep:
    for $i \leftarrow 0$ to $n - 1$ **do**
        Compute $\sigma \leftarrow \sum_{j<i} A[i,j]x[j] + \sum_{j>i} A[i,j]x[j]$
        Update $x[i] \leftarrow (b[i] - \sigma)/A[i,i]$
    end for
    Compute relative error $\frac{||x_k - \tilde{x}||_2}{||\tilde{x}||_2}$
end while
return x, relative errors, iterations

When computing Forward Gauss-Seidel, we require the preconditioner to be $P = D - L$, where D is the diagonal of A and L is the strictly lower triangular part of A. Here we have the iteration matrix $G = I - (D - L)^{-1}A$. For Gauss-Seidel, we do not want to compute $P^{-1} = (D - L)^{-1}$ directly, since we are not guaranteed numerical stability in the computation. So we implement a forward sweep. Let us rearrange the iteration equation such that

$$\begin{aligned} x_{k+1} &= x_k + (D - L)^{-1} r_k \\ x_{k+1} - x_k &= (D - L)^{-1} r_k \\ z_k &= (D - L)^{-1} r_k \\ (D - L) z_k &= r_k. \end{aligned}$$

By applying a forward sweep (forward substitution), we can solve for z_k. This gives us

$$x_{k+1} = x_k + z_k. \tag{33}$$

Notice that, unlike the Jacobi method, Gauss-Seidel updates the solution vector x_{k+1} one component at a time, updating immediately after the new component is determined. Hence Gauss-Seidel works in a sequential manner.

Algorithm 6 Gauss-Seidel Symmetric Method

Require: $A \in \mathbb{R}^{n \times n}$, $\tilde{x} \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$, $b \in \mathbb{R}^n$
Ensure: $\tilde{b} = A\tilde{x}$
Initialize $r_0 \leftarrow b - Ax_0$
Initialize $iterations \leftarrow 0$
while iterations < 1000 and $\frac{||x_k - \tilde{x}||_2}{||\tilde{x}||_2} > 10^{-6}$ **do**
    Perform forward sweep:
    for $i \leftarrow 0$ to $n-1$ **do**
        Compute $sum \leftarrow \sum_{j<i} A[i,j]x[j] + \sum_{j>i} A[i,j]x[j]$
        Update $x[i] \leftarrow (b[i] - sum)/D[i]$
    end for
    Perform backward sweep:
    for $i \leftarrow n-1$ to 0 **do**
        Compute $\sigma \leftarrow \sum_{j<i} A[i,j]x[j] + \sum_{j>i} A[i,j]x[j]$
        Update $x[i] \leftarrow (b[i] - \sigma)/A[i,i]$
    end for
    Compute relative error $\frac{||x_k - \tilde{x}||_2}{||\tilde{x}||_2}$
end while
return x, relative errors, iterations

Symmetric Gauss-Seidel work similar to Forward Gauss-Seidel, with the added step of performing a backward sweep (backward substitution). For this method, we set the preconditioner to be $P = (D-L)D^{-1}(D-U)$. Our iteration matrix is then $G = I - (D-U)^{-1}D(D-L)^{-1}A$. Therefore we need to solve

$$\begin{aligned} x_{k+1} &= x_k + (D-U)^{-1}D(D-L)^{-1}r_k \\ x_{k+1} - x_k &= (D-U)^{-1}D(D-L)^{-1}r_k \\ y_k &= (D-U)^{-1}D(D-L)^{-1}r_k \\ (D-L)D^{-1}(D-U)y_k &= r_k \\ (D-L)D^{-1}w_k &= r_k \\ (D-L)z_k &= r_k. \end{aligned}$$

We apply a forward sweep to solve

$$(D-L)z_k = r_k \tag{34}$$

for z_k. Then we set $w_k = Dz_k$ and compute diagonal scaling to solve for w_k. Lastly, we apply a backward sweep to solve

$$(D-U)y_k = w_k \tag{35}$$

for y_k. Therefore we update the solution vector by computing

$$x_{k+1} = x_k + y_k.$$

For all three methods, our algorithm implements the Python library Numpy to compute the eigenvalues of the iteration matrix G. We find the absolute maximum eigenvalue to define the spectral radius $\rho(G)$. We also compute the norm of G. Ideally, if the matrix converges, the algorithm will show

$$\rho(G) = ||G||_2 < 1. \tag{36}$$

4.3 Part 3: Behavior of Stationary Methods on Large Sparse Symmetric Positive Definite Matrices

In Part 2, we constructed algorithms that behave for small, dense matrices. We will now modify these algorithms to take in large, sparse matrices stored in a compressed format, and we will examine the behaviors of these stationary methods.

4.3.1 Algorithm to Generate a Sparse Matrix

We first develop an algorithm to generate a symmetric, diagonally dominant sparse matrix $A \in \mathbb{R}^{n \times n}$. We can exploit the structure of symmetric matrices by considering the decomposition $A = D - L - L^T$, where D contains the diagonal elements of A and L represents the strictly lower triangular part of A.

Algorithm 7 Generate Sparse Symmetric Positive Definite Matrix

Require: n ▷ Dimension of the square matrix
Ensure: $A \in \mathbb{R}^{n \times n}$ ▷ Sparse symmetric positive definite matrix
Initialize $L \leftarrow \text{zeros}(n, n)$
for $i \leftarrow 0$ to $n - 1$ **do**
  for $j \leftarrow 0$ to $n - 1$ **do**
    if $i > j$ **then** ▷ Lower triangular part
      $x \leftarrow$ randomly choose from $\{0, 1\}$
      if $x = 1$ **then**
        $L[i, j] \leftarrow$ (random integer from 1 to 9) $\div\, i$
      end if
    end if
  end for
end for
$A \leftarrow L + L^T$ ▷ Make the matrix symmetric
Fill diagonal elements of A with 20 to ensure diagonal dominance
return A

Since A is symmetric, L^T exactly represents the strictly upper triangular part of A. Thus we only need to generate values for D and L to develop A. At each entry, the algorithm either places a random integer (on a scale of 1-9) or leaves the value 0. Each row is scaled by the number of entries to the left of the diagonal. This gives us L. We add L^T and populate the diagonal with the value 20. The sum of the elements on either side of the diagonal will be < 10,

so this ensures the sum of all off-diagonal elements in each row will be strictly < 20, hence satisfying the condition for diagonal dominance.

4.3.2 Compressed Sparse Row (CSR) Matrix Representation

Storing a large, sparse matrix as a 2D array requires substantial memory and unnecessary computations with the zero elements. We want to efficiently store and process sparse matrices to optimize sparse matrix-vector multiplication. We consider a particular storage scheme, called the Compressed Sparse Row (CSR) format, to store only the non-zero elements in the sparse matrix.

Algorithm 8 Convert Matrix to Compressed Sparse Row Format

Require: $A \in \mathbb{R}^{n \times n}$ ▷ Sparse matrix
Ensure: AA, JA, IA ▷ CSR representation
Initialize $AA \leftarrow [\,]$, $JA \leftarrow [\,]$, $IA \leftarrow [0]$
for each row in A **do**
    for each column index j and value a in the row **do**
        if $a \neq 0$ **then**
            Append a to AA
            Append j to JA
        end if
    end for
    Append len(AA) to IA ▷ End of the current row in AA
end for
Convert AA, JA, and IA to arrays
return AA, JA, IA

We store the nonzero elements of A in the array AA and the corresponding column indices in the array JA. We store the row points in IA, which indicate where each row begins in AA, by appending the length of AA whenever we reach a nonzero element in a new row. Therefore IA[i] points to the index in AA of the first non-zero element in row i, and IA[$i + 1$] points to the first non-zero element in the following row. The last element in IA will represent the total number of non-zero elements in A, hence the length of IA will be $n+1$, where n is the number of rows in A. Once the sparse matrix is stored in CSR format, we must develop a routine to efficiently compute CSR matrix-vector multiplication. This will allow us to adapt the algorithms in Part 2 to compressed matrices.

Algorithm 9 Matrix-Vector Multiplication in CSR Format

Require: AA (non-zero values), JA (column indices), IA (row pointers), x (input vector)
Ensure: b (result vector)
1: Initialize $b \leftarrow \text{zeros}(\text{len}(IA) - 1)$
2: **for** $i \leftarrow 0$ to $\text{len}(b) - 1$ **do**
3: **for** $k \leftarrow IA[i]$ to $IA[i+1] - 1$ **do**
4: $b[i] \leftarrow b[i] + AA[k] \cdot x[JA[k]]$
5: **end for**
6: **end for**
7: **return** b

We initialize a result vector b. Since IA has $n+1$ elements, where n corresponds to the number of rows in A, we want b to have $(n+1)-1$ elements, i.e. we want $b \in \mathbb{R}^n$. To compute the product, we iterate over each row i in A and use the range $(IA[i], IA[i+1]-1)$ to determine the range of indices in AA corresponding to the non-zero elements in row i. For each nonzero element in row i corresponding to $k \in (IA[i], IA[i+1]-1)$, we multiply the value in $AA[k]$ with the corresponding element in the vector $x[JA[k]]$. The sum of these values contribution to the the entry $b[i]$. We use this algorithm to adjust the Jacobi, GS, and SGS stationary iterative methods to perform on this form of matrix representation.

4.3.3 Modified Jacobi, Gauss-Seidel, and Symmetric Gauss-Seidel Methods

We modify the routines for stationary iterative methods to allow inputs of either sparse matrix form (stored as a 2D-array) or CSR form (stored as a tuple of arrays).

Algorithm 10 Stationary Iterative Methods for Linear Systems

Require: matrix_representation, $\tilde{x}, x_0, b$, flag ▷ Input parameters
Ensure: $x, G, \rho, \|G\|$, iter, rel_err_arr ▷ Outputs

Initialize: $x \leftarrow x_0$, $n \leftarrow \text{len}(x_0)$, rel_err_arr $\leftarrow [\,]$, iter $\leftarrow 0$
Define matrix_vector_multiply and get_diagonal based on the input type (**dense** or **CSR**)
Compute $r \leftarrow b - \text{matrix\_vector\_multiply}(x)$
Set $D \leftarrow \text{get\_diagonal}()$
while iter $<$ max_iter **and** rel_err $>$ tol **do**
    Compute relative error:

$$\text{rel\_err} \leftarrow \frac{\|x - x_{\text{tilde}}\|}{\|x_{\text{tilde}}\|}$$

    Append rel_err to rel_err_arr
    if flag $= 1$ **then** ▷ Jacobi Method
        Update $x \leftarrow x + \frac{r}{D}$
        Compute $r \leftarrow b - \text{matrix\_vector\_multiply}(x)$
    else if flag $= 2$ **or** 3 **then** ▷ Gauss-Seidel/Symmetric Gauss-Seidel
        Perform forward sweep to update x
        if flag $= 3$ **then** ▷ Symmetric Gauss-Seidel
            Perform backward sweep to update x
        end if
    end if
    iterations $\leftarrow$ iterations $+ 1$
end while
if dense **then**
    Compute $L, U \leftarrow$ Lower/Upper triangular parts of A
    if flag $= 1$ **then** ▷ Jacobi
        Compute iteration matrix $G \leftarrow I - A/D$
    else if flag $= 2$ **then** ▷ Gauss-Seidel
        Compute $G \leftarrow (L + D)^{-1}U$
    else if flag $= 3$ **then** ▷ Symmetric Gauss-Seidel
        Compute $G \leftarrow G_{\text{forward}} \cdot (L + D)^{-1}$
    end if
    Compute $\rho \leftarrow \max(\text{eigenvalues of } G)$
    Compute $\|G\|$
end if
Return: $x, G, \rho, \|G\|$, iter, rel_err_arr

We pass Algorithm 9 to allow matrix-vector multiplications when the input matrix is formatted in CSR storage. We again compute and store the relative errors between the current iteration solution x_{k+1} and the true solution $\tilde{x}$. We will use this algorithm to compare convergence results and time complexity between the two matrix representations.

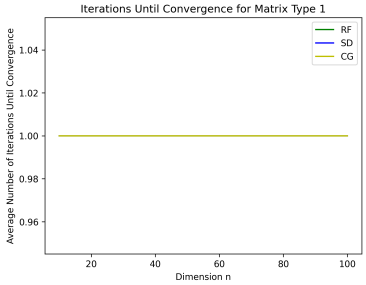
5 Description of the Experimental Design and Results

5.1 Testing Richardson's First, Steepest Descent, and Conjugate Gradient Methods on Diagonal Matrices

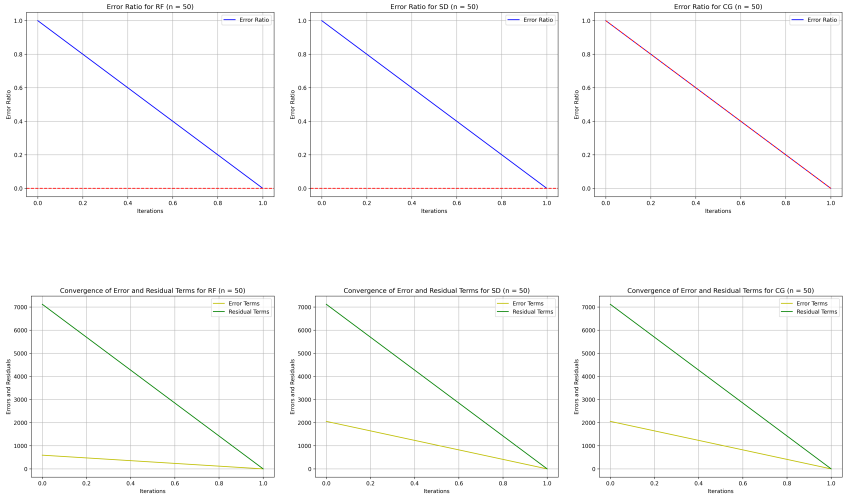
In Part 1, we created routines to perform the RF, SD, and CG iterative methods on diagonal matrices stored in a single vector. In this section, we evaluate the performance of these routines. We generate 5 types of diagonal matrices Λ to test (each type is subject to a different condition). Each Λ will have positive eigenvalues (positive entries) λ that take values on the range $(1, 100)$. Since such diagonal matrices are SPD, we should examine convergence for each method and each matrix tested. For each of the 5 matrix conditions we examine, we will conduct the same tests to evaluate the performance of the iterative algorithms. We generate Λ for dimensions $n = 10$ to $n = 100$ with step size 10. For each dimension size n, we examine the error ratios for each iteration, for each of the three methods (RF, SD, and CG), for several solutions $\tilde{x}$. The entries of $\tilde{x}$ and each intiial guess x_0 take in values on the range $(-100, 100)$. We plot the convergence bound to show the error ratios remain below this bound, which follows directly from the damping inequality for the error terms. We also plot the error terms and residual terms for each method, and show that both converge to 0 as the the approximations x_{k+1} converge to the true solution $\tilde{x}$. We also examine the average number of iterations until convergence for matrices of order $n = 10$ to $n = 100$ with step size 10. This average is determined by testing five randomly generated solution vectors at each dimension. We examine the following conditions:

(1) *All eigenvalues are the same.*

Since the eigenvalues are all equal, we can think of the result vector $\tilde{b}$ as simply the solution vector $\tilde{x}$ scaled by a value. Hence for this problem, all three methods converge to the solution vector in one iteration. This is true for all dimensions n tested.

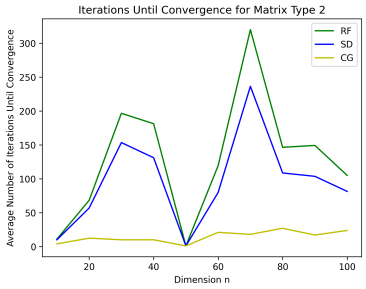


The solution converges exactly in one iteration, so the error and residual terms reduces to 0 at iteration $k = 1$. This is true for all dimension sizes n tested. Additionally, we set the step size $\frac{\kappa+1}{\kappa-1}$ as a bound for the error ratios. Since the eigenvalues are all the same, $\kappa = 1$. We see that the error ratios are bounded by 0. These results are shown in the plots below, for dimensions size $n = 50$:

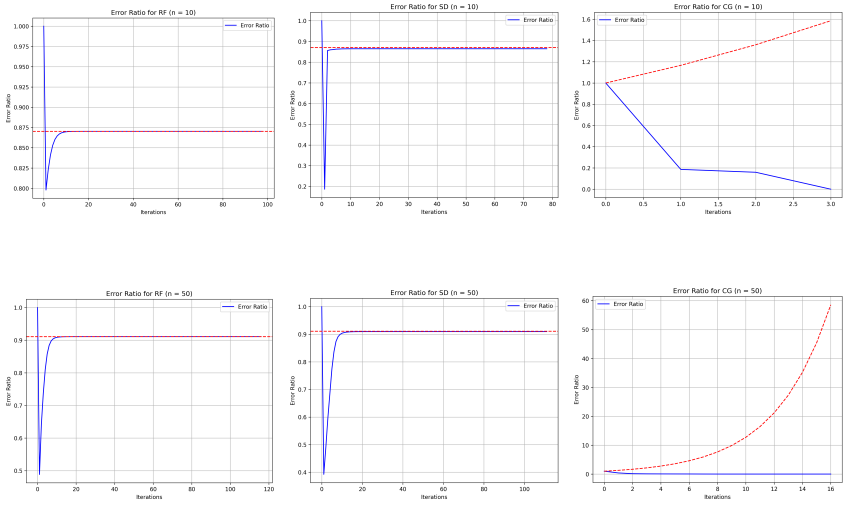


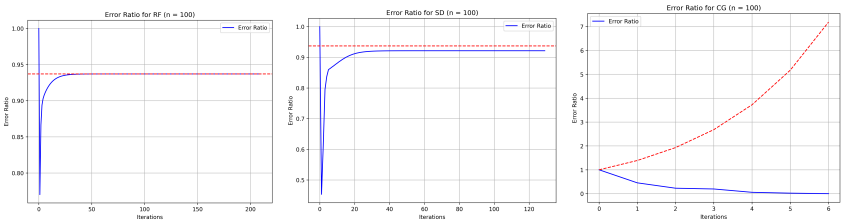
(2) *k distinct eigenvalues with multiplicities randomly chosen.*

For each dimension n tested, the average number of iterations until convergence are similar for RF and SD, ranging from ≈ 50 to 300 on average for these two methods. This is significantly higher than the average iterations for CG. We also note that CG converges in approximately the same number of iterations (< 20) for each n. This is shown in the plot below.

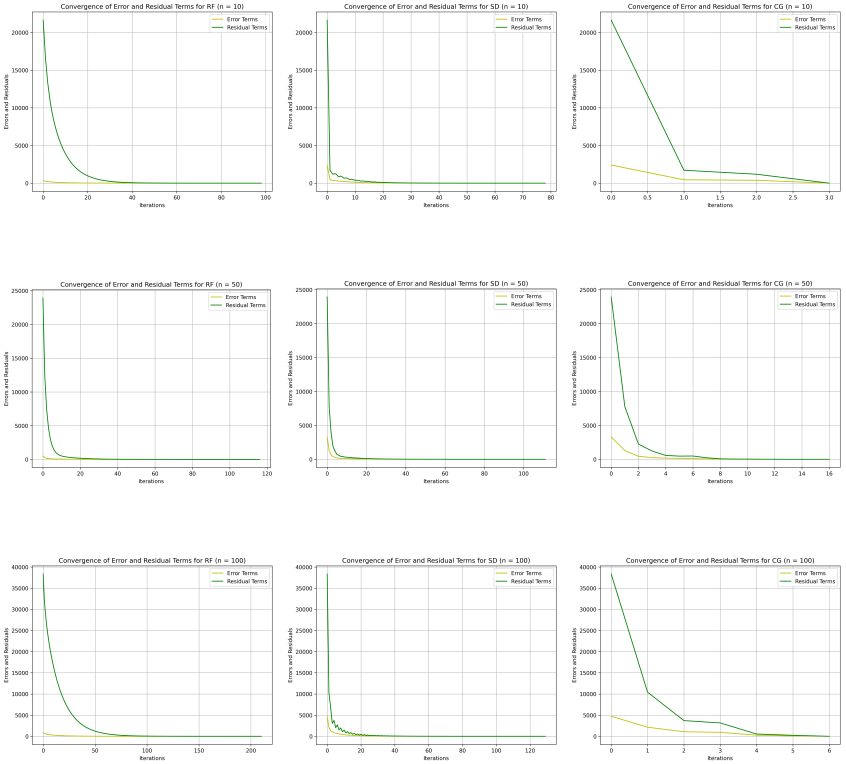


At each dimension n tested, the error ratios for RF and SD are bounded by a value close to 1. This tells us that the condition number for this matrix is close to 0. Since the error terms for RF and SD decrease by a constant step size with each iteration, the error ratios are the same value at each step, and equal to the step size (since the step size is the upper bound for the error ratios). The error ratios remain strictly below the bound for CG, and decrease exponentially. This follows from the damping inequality for CG, since the step size changes at each iteration. The plots below show these results for dimensions $n = 10$, $n = 50$, $n = 100$.





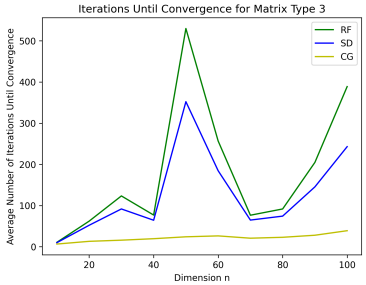
We also examine the plots of the error and residual terms for dimensions $n = 10$, $n = 50$, $n = 100$.



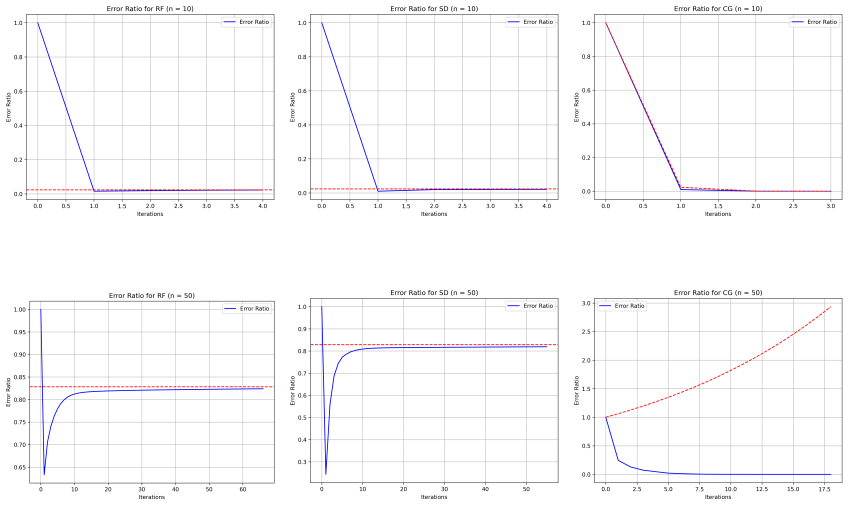
For each dimension n, the error and residual terms for each method approach a value close to 0 within the first few iterations. Since CG converges in significantly less iterations, error and residual terms approach 0 at a lower iteration number. For this matrix type, all three methods are able to quickly approximate a solution vector close to the true solution $\tilde{x}$.

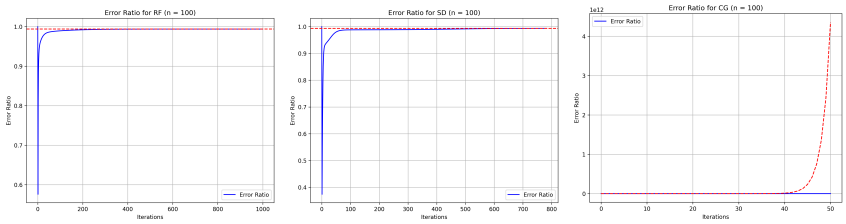
(3) *k distinct eigenvalues with a "cloud" of normal distributions around each.*

For each dimension n tested, the average number of iterations until convergence are again similar for RF and SD. The average iterations range from ≈ 100 to 400 for each n for these two methods. This is significantly higher than the average number of iterations for CG. Note that CG again converges in approximately the same number of iterations for each n. This is shown in the plot below.

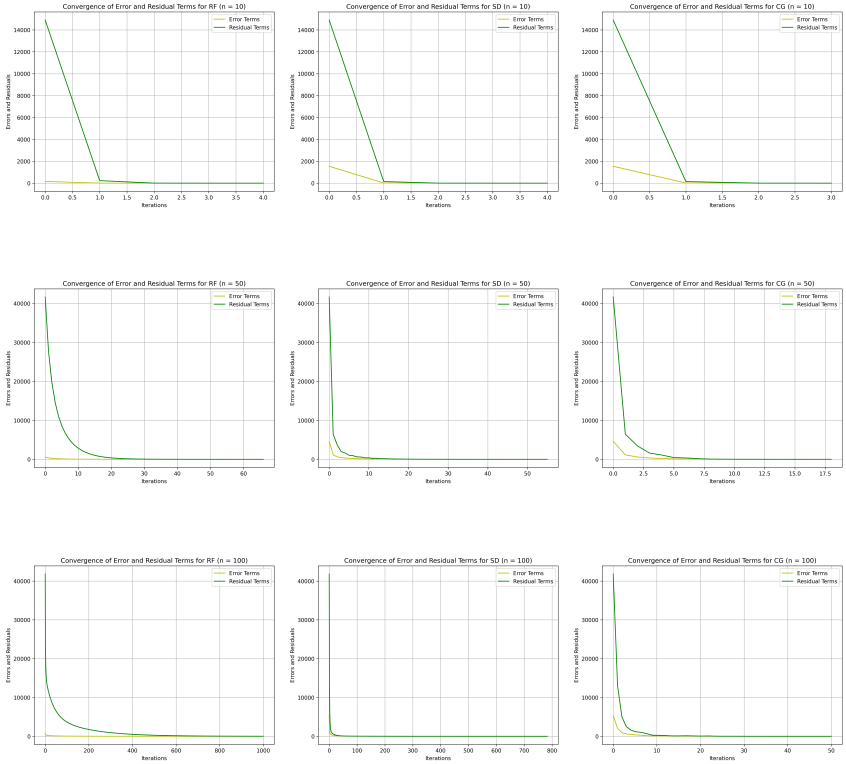


For smaller dimensions n tested, the error ratios for all three methods are close to 0 in value. The bound for the error ratios for RF and SD increases to 1 for larger dimensions tested, and remain constant at each iteration. The error ratios again decrease exponentially for CG, and the values of the error ratios decrease more quickly as n increases. The plots below show these results for dimensions $n = 10$, $n = 50$, $n = 100$.





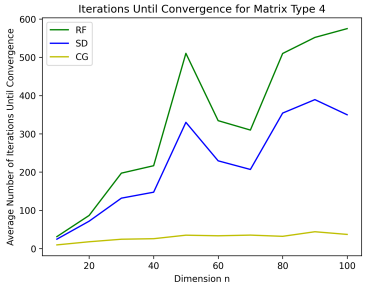
We also examine the plots of the error and residual terms for dimensions $n = 10$, $n = 50$, $n = 100$.



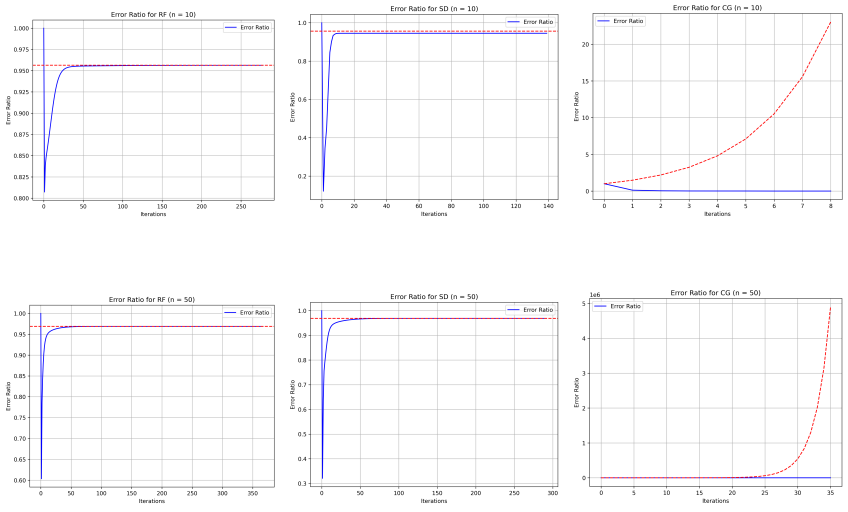
For each dimension n, the error and residual terms for each method approach a value close to 0 within the first few iterations. However, we must note that the average number of iterations per method increase with dimension size n. Hence the iteration number at which the terms decrease to 0 *increases* with n. Since CG converges in significantly less iterations, error and residual terms approach 0 at a lower iteration number. For this matrix type, all three methods are again able to quickly approximate a solution vector close to the true solution $\tilde{x}$.

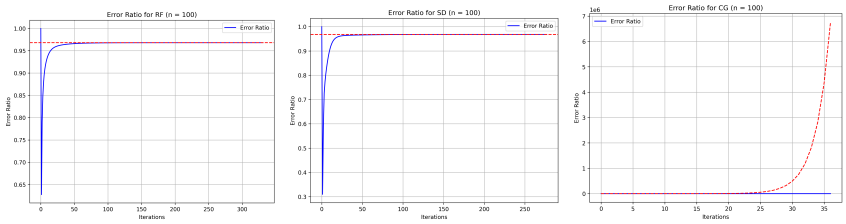
(4) *Eigenvalues chosen from a uniform distribution between parameters* λ_{min} *and* λ_{max}.

For each dimension n tested, the average number of iterations until convergence again follow similar patterns for RF and SD, with SD performing better for larger values of n. For this choice of matrix Λ, the average number of iterations for RF and SD generally increase as n increases. We again note that CG converges in approximately the same number of iterations for each n, and converges in significantly less iterations than RF and SD. This is shown in the plot below.

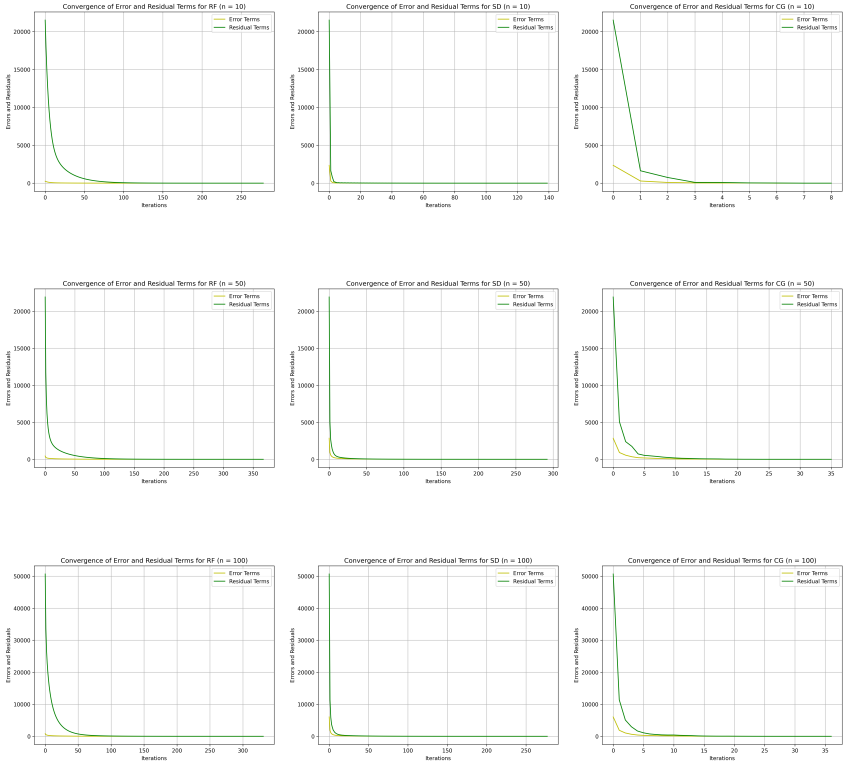


The error ratios for RF and SD are bounded by values close to 1 for each dimension n, and remain constant at this bound for each iteration. The error ratios for CG are bounded by a value that changes exponentially as the number of iterations increase. The bound for CG increases exponentially with each iteration k. The error ratios for CG are close to 0 for each iteration, at each dimension size n. The plots below show these results for dimensions $n = 10$, $n = 50$, $n = 100$.





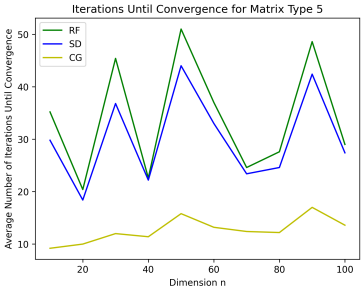
We also examine the plots of the error and residual terms for dimensions $n = 10$, $n = 50$, $n = 100$.



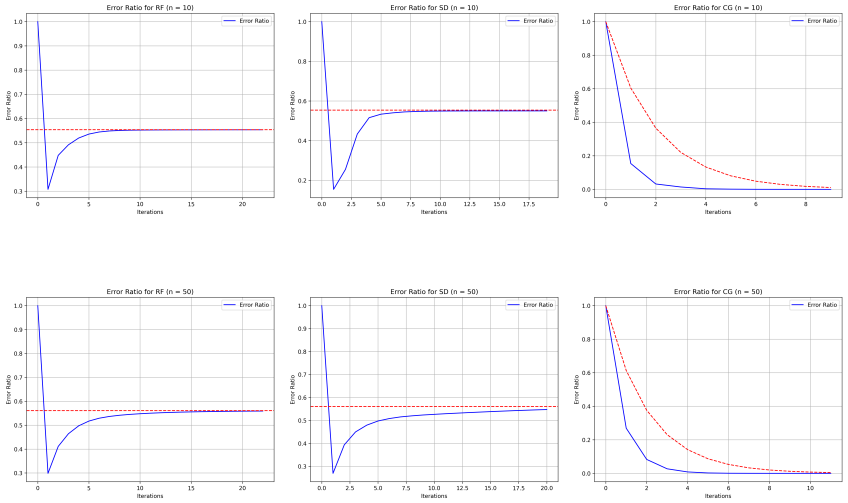
For this matrix type, we see the error and residual terms for each method approach a value close to 0 within the first few iterations. We note that although RF and SD converge in approximately the same number of iterations per dimension n, the SD method reaches an approximation close to the true solution $\tilde{x}$ in fewer iterations than RF. Since CG converges in significantly less iterations, error and residual terms approach 0 at a lower iteration number. Particularly, for $n = 50$ and $n = 100$, CG reaches an approximation close to the true solution after 5 iterations.

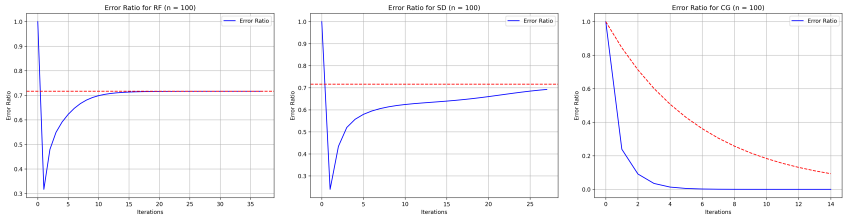
(5) *Eigenvalues chosen from a normal distribution with mean* $\mu = \frac{\lambda_{min}+\lambda_{max}}{2}$ *and standard deviation* $\sigma^2 = \frac{\lambda_{max}-\lambda_{min}}{6}$.

The average number of iterations until convergence are again similar for RF and SD, for each dimension size n. For this choice of matrix Λ, we see random spikes in the average number of iterations as n increases. However, for this matrix type, RF and SD converge in significantly less iterations than the previous matrix conditions we examined (between ≈ 20 to 50 iterations for all n). We again note that CG converges in approximately the same number of iterations for each n, but the average number of iterations increases slightly as n increases. This is shown in the plot below.

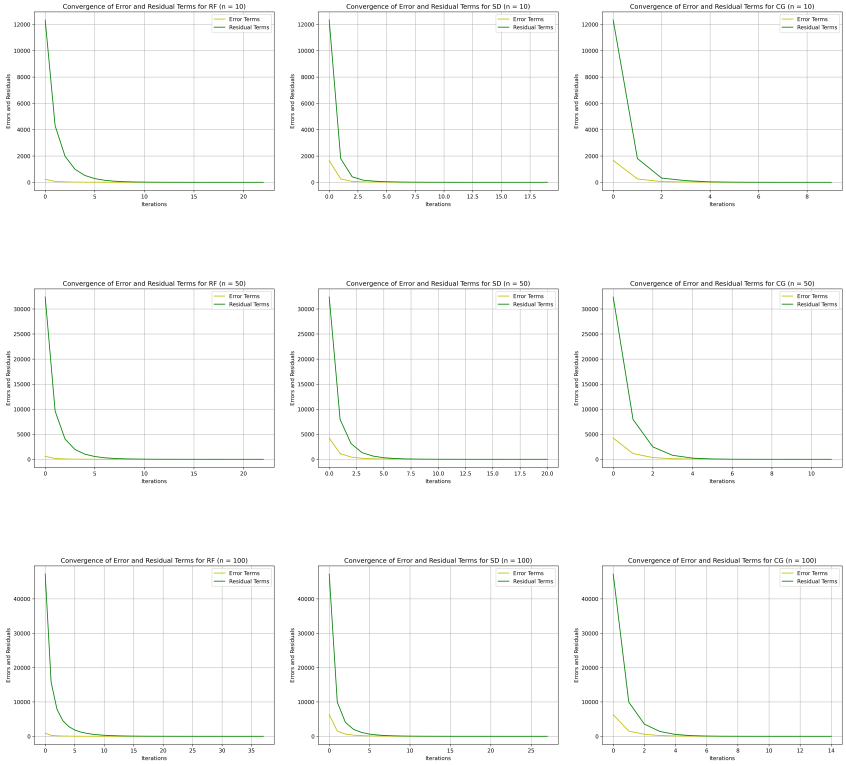


The error ratios for RF and CG are bounded by a value close to .6. The error ratios for CG are bounded by a value that decreases exponentially as the number of iterations increase. Theoretically, this is what the bound *should* look for the error ratios for CG. The error ratios for CG are close to this upper bound for each iteration, at each dimension size n. The plots below show these results for dimensions $n = 10$, $n = 50$, $n = 100$.





We also examine the plots of the error and residual terms for dimensions $n = 10$, $n = 50$, $n = 100$.



For this matrix type, we see the error and residual terms for each method again follow a similar pattern, each approaching a value close to 0 within the first few iterations. RF approximates a value of x close to the true solutions within the first 5 to 10 iterations, as n increases. SD performs slightly better, generally reaching a close approximation within the first 5 iterations. Since CG converges in significantly less iterations, we examine the error and residual terms approach 0 at lowest iteration number (in approximately 2 iterations, on average).

5.2 Testing Jacobi, Gauss-Seidel, and Symmetric Gauss-Seidel Methods on Dense Matrices

In this section, we evaluate the performance of our routines for Jacobi, Gauss-Seidel (GS), and Symmetric Gauss-Seidel (SGS) on 9 different dense square matrices A. We use the built-in library function Numpy to assess the properties of each matrix, to determine whether the matrices are symmetric, diagonally dominant, and SPD. These are sufficient conditions for convergence. We check if each matrix is SPD by performing the Cholesky decomposition, which works only for SPD matrices. If Cholesky fails, we determine the matrix is *not* SPD. After we examine the properties of the matrices we will test, we generate a table with the results of running each iterative method. We examine the average number of iterations until convergence, the spectral radius ($\rho(G)$ and A-norm of the corresponding iteration matrix G ($||G||_A$), and the average time it takes for the method to converge to the true solution. We will average the results of testing 5 different solution vectors $\tilde{x}$ and initial guesses x_0 per matrix. The values for these vectors are randomly chosen on the range $(-100, 100)$. For our results, if $\rho(G) < 1$, this is a *necessary* and sufficient condition for convergence. For each matrix we test, we also plot the average relative errors per iteration for each method.

We first observe the following table of the analysis results for each of matrices we will examine:

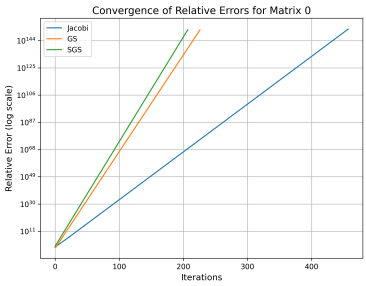
Table 1: Matrix Properties Analysis

Matrix	Symmetric	Diagonally Dominant	SPD	Condition Number
0	True	False	False	4.49
1	False	False	False	18.67
2	False	False	False	11.82
3	False	False	False	4.09
4	False	False	False	5.98
5	False	False	False	35.76
6	False	False	False	49.35
7	True	True	True	2.72
8	True	True	True	25.27

We will reference the above table as we present the following matrices and the corresponding results for each.

Matrix 0:

$$A_0 = \begin{pmatrix} 3 & 7 & -1 \\ 7 & 4 & 1 \\ -1 & 1 & 2 \end{pmatrix}$$

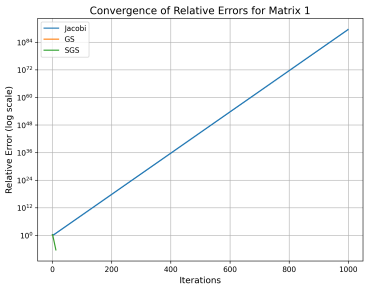


Method	Iterations	Spectral Radius	G Norm	Time to Converge
Jacobi	921.2	2.15537	3.02994	0.00985
GS	1000.0	4.72835	5.79287	0.01271
SGS	1000.0	2.92224	4.14899	0.01696

This matrix is symmetric, but not diagonally dominant or SPD (Table 1). The properties indicate the matrix may not converge. The above results confirm the matrix does not converge. As the number of iterations approach our maximum threshold (1000), the relative error between the true solution $\tilde{x}$ and the approximate solution x_{k+1} at each iteration k approach infinite values. We also note that the spectral radius and iteration matrix G-norm are > 1. This supports our results. The matrix does not converge for Jacobi, GS, or SGS.

Matrix 1:

$$A_1 = \begin{pmatrix} 3 & 0 & 4 \\ 7 & 4 & 2 \\ -1 & -1 & 2 \end{pmatrix}$$

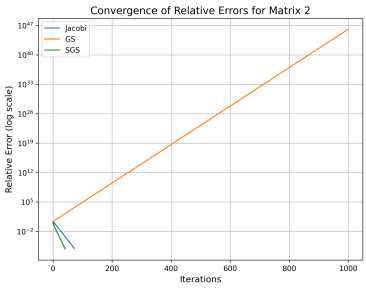


Method	Iterations	Spectral Radius	G Norm	Time to Converge
Jacobi	1000.0	1.22964	3.25854	0.01009
GS	12.4	0.25	2.28066	0.00032
SGS	12.4	0.52083	1.2095	0.00038

This matrix is neither symmetric, diagonally dominant, nor SPD (Table 1). The properties suggests the matrix may not converge. The above results confirm the matrix does not converge for the Jacobi method. For this method, we reach the iteration threshold ($k = 1000$) and observe the relative error approach infinite values as $k \to 1000$. However, we examine convergence for GS and SGS in approximately 12 iterations. Notice that the spectral radius is < 1 for these two methods. This guarantees convergence for Matrix 1, because $\rho(G) < 1$ is a necessary and sufficient condition for convergence of any iterative method.

Matrix 2:

$$A_2 = \begin{pmatrix} -3 & 3 & -6 \\ -4 & 7 & -8 \\ 5 & 7 & -9 \end{pmatrix}$$

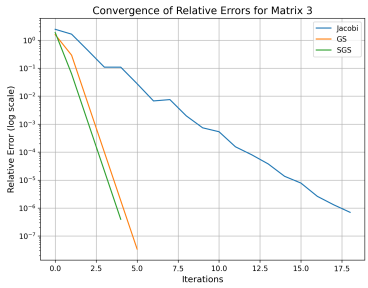


Method	Iterations	Spectral Radius	G Norm	Time to Converge
Jacobi	69.8	0.81331	2.64079	0.00098
GS	1000.0	1.11111	2.74975	0.01179
SGS	44.2	0.5792	0.5907	0.0009

This matrix is neither symmetric, diagonally dominant, nor SPD (Table 1). The properties suggests the matrix may not converge. The above results show that the matrix does not converge for GS, but *does* converge for Jacobi and SGS. Jacobi converges in nearly 70 iterations and SGS converges in about 44 iterations. Both of these methods have a spectral radius < 1, which guarantees the observed convergence. This observation is possible because Jacobi treats all rows independently, and SGS may stabilize error by alternating between forward and backward sweeps. GS may fail as a result of the sequential nature of its updates. We observe the spectral radius for GS is > 1.

Matrix 3:

$$A_3 = \begin{pmatrix} 4 & 1 & 1 \\ 2 & -9 & 0 \\ 0 & -8 & -6 \end{pmatrix}$$

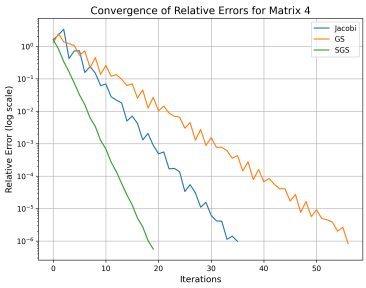


Method	Iterations	Spectral Radius	G Norm	Time to Converge
Jacobi	19.6	0.44382	1.03935	0.00032
GS	5.8	0.01852	0.37702	0.00025
SGS	5.0	0.00977	0.04578	0.00025

This matrix is neither symmetric, diagonally dominant, nor SPD. But we do observe a low condition number (4.09) (Table 1). The above results show that the matrix converges for all three methods. For each method, the spectral radius is < 1. Jacobi converges in nearly 20 iterations, whereas GS and SGS converge in around 5 iterations, and the relative errors for GS and SGS follow nearly the same pattern for convergence in the convergence plot. The average time to convergence further confirms our result: GS and SGS converge faster than Jacobi for this matrix. The spectral radius is < 1 for all three methods. We observe lower spectral radius for GS and SGS, with values much closer to 0 than the spectral radius for Jacobi.

Matrix 4:

$$A_4 = \begin{pmatrix} 7 & 6 & 9 \\ 4 & 5 & -4 \\ -7 & -3 & 8 \end{pmatrix}$$

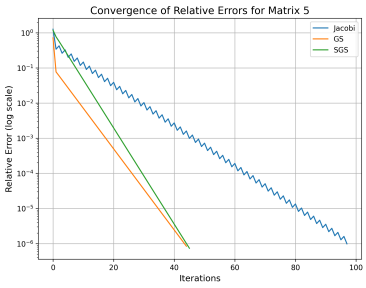


Method	Iterations	Spectral Radius	G Norm	Time to Converge
Jacobi	34.6	0.64113	2.15457	0.00047
GS	57.8	0.7746	2.57634	0.00085
SGS	18.8	0.18925	0.50099	0.00049

This matrix is neither symmetric, diagonally dominant, nor SPD (Table 1). The above results show that the matrix converges for all three methods. GS has the slowest time to convergence (about 0.00085 seconds) and, on average, takes the highest number of iterations to converge (nearly 35 iterations). SGS has the fastest time to convergence (about 0.00049 seconds) and, on average, takes the lowest number of iterations to converge (nearly 19 iterations). These results are confirmed in the values for the spectral radii: SGS has a lower spectral radius (≈ 0.18925) than Jacobi (≈ 0.64113). In the convergence plot, we also note that the relative errors decrease nearly perfectly step-wise for each iteration, whereas the relative errors for Jacobi and GS follow a less smooth pattern. For this matrix, we see that SGS outperforms both Jacobi and GS.

Matrix 5:

$$A_5 = \begin{pmatrix} 6 & -2 & 0 \\ -1 & 2 & -1 \\ 0 & -6/5 & 1 \end{pmatrix}$$

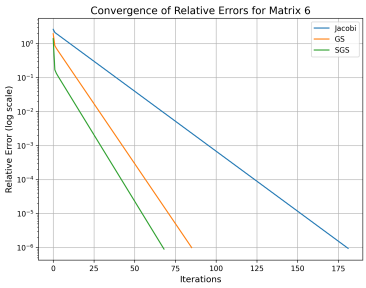


Method	Iterations	Spectral Radius	G Norm	Time to Converge
Jacobi	104.0	0.8756	1.54524	0.00102
GS	51.8	0.76667	0.88819	0.00066
SGS	45.2	0.99435	1.00352	0.00086

This matrix is neither symmetric, diagonally dominant, nor SPD (Table 1). The above results show that the matrix converges for all three methods. For this matrix, Jacobi has the slowest time to convergence (about 0.00102 seconds) and, on average, takes the highest number of iterations to converge (nearly 104 iterations). SGS takes the lowest number of iterations to converge (nearly 45 iterations). Unlike the previous matrix, however, we observe that SGS has the largest spectral radius (≈ 0.99435), with a value that is nearly 1. GS has the lowest spectral radius (≈ 0.76667) and is the fastest to converge (about 0.00066 seconds). Both GS and SGS perform better than Jacobi for this matrix, converging in nearly half the amount of iterations.

Matrix 6:

$$A_6 = \begin{pmatrix} 5 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -3/2 & 1 \end{pmatrix}$$

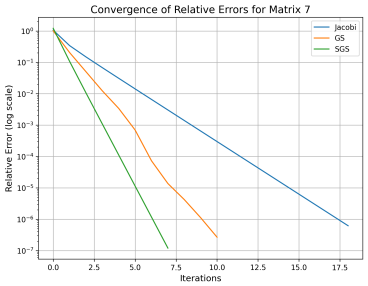


Method	Iterations	Spectral Radius	G Norm	Time to Converge
Jacobi	164.4	0.92195	1.36107	0.00262
GS	85.0	0.85	0.94074	0.00163
SGS	77.0	1.18231	1.19725	0.0018

This matrix is neither symmetric, diagonally dominant, nor SPD (Table 1). The above results show that the matrix converges for all three methods. We observe SGS converge in the lowest number of iterations (approximately 77), on average. Jacobi has the slowest time to convergence (about 0.00262 seconds) and, on average, takes the highest number of iterations to converge (nearly 164 iterations). However, we observe that SGS has a spectral radius > 1. Theoretically, this matrix should not converge for SGS. Therefore, we consider the possibility that, for SGS, numerical approximations and rounding may be stabilizing the errors with each iteration, causing the matrix to converge in practice. For this matrix, SGS is potentially unstable, hence GS offers the most reliable performance.

Matrix 7:

$$A_7 = \begin{pmatrix} 4 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 4 \end{pmatrix}$$

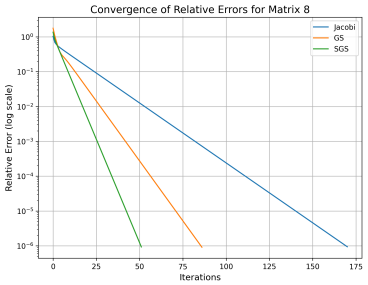


Method	Iterations	Spectral Radius	G Norm	Time to Converge
Jacobi	18.4	0.46194	0.86603	0.00035
GS	11.6	0.21339	0.63224	0.00057
SGS	7.2	0.0928	0.1719	0.0008

This matrix is symmetric, diagonally dominant, and SPD, and has a low condition number of 2.72 (Table 1). The above results show that the matrix converges for all three methods. SGS is the slowest to converge (about 0.00080 seconds), but takes the least number of iterations to converge (about 7 iterations on average). We observe the lowest spectral radius for SGS (≈ 0.0928). Jacobi is the fastest to converge (about 0.00035 seconds), but takes the most iterations to converge (about 18 iterations on average). Because this matrix is well-conditioned and SPD, we observe all three methods perform very well. Jacobi is the fastest but SGS performs best, with the lowest spectral radius and average number of iterations to convergence.

Matrix 8:

$$A_8 = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix}$$



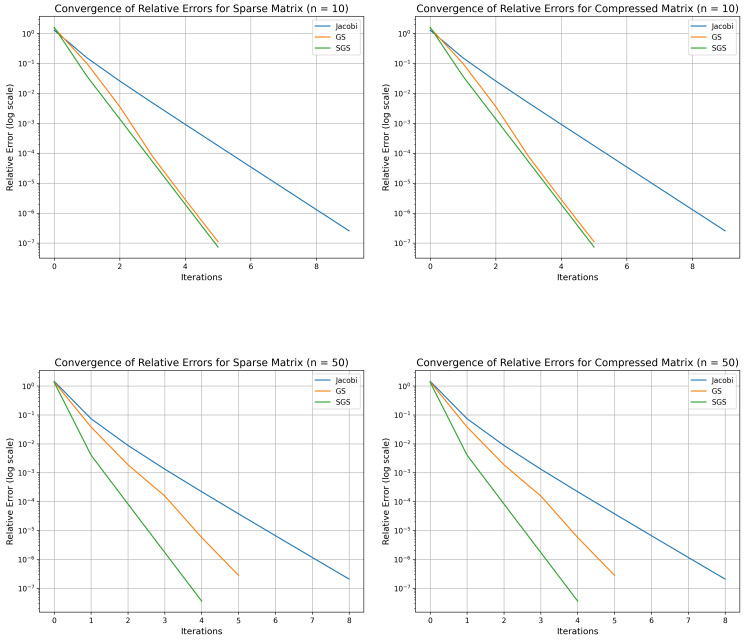
Method	Iterations	Spectral Radius	G Norm	Time to Converge
Jacobi	169.4	0.92388	1.73205	0.00216
GS	82.0	0.85355	1.40436	0.00355
SGS	50.2	0.74239	0.99075	0.00323

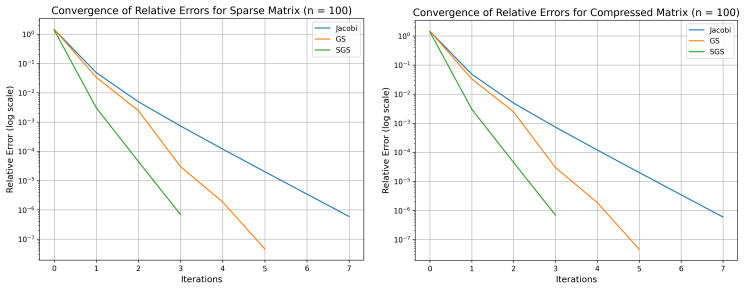
This matrix is symmetric, diagonally dominant, and SPD, but has a much higher condition number than the previous matrix, with a condition number of 25.27 (Table 1). The above results show that the matrix converges for all three methods. SGS is the slowest to converge (about 0.00325 seconds), but takes the least number of iterations to converge (about 50 iterations on average). We observe the lowest spectral radius for SGS (≈ 0.74239). Jacobi is the fastest to converge (about 0.00216 seconds), but takes the most iterations to converge (about 169 iterations on average). Because this matrix is well-conditioned and SPD, we observe all three methods perform very well. Jacobi is the fastest but SGS performs best, with the lowest spectral radius and average number of iterations to convergence. Jacobi takes over 3 times the number of iterations to converge.

5.3 Testing Jacobi, Gauss-Seidel, and Symmetric Gauss-Seidel on Sparse Matrices

We examined the performance of Jacobi, Gauss-Seidel (GS), and Symmetric Gauss-Seidel (SGS) on dense matrices. We will now examine the performance of these methods on symmetric, diagonally dominant sparse matrices. In Section 4, we described an algorithm to generate a random sparse matrix and an algorithm to store a sparse matrix in Compressed Sparse Row (CSR) format. We apply the three iterative methods to sparse matrices stored in both formats, and test matrix sizes ranging from dimension $n = 10$ to $n = 100$ with step size 10. We will average the results of testing 5 different solution vectors $\tilde{x}$ and initial guesses x_0 per dimension. The values for these vectors are randomly chosen on the range $(-100, 100)$.

Observe the following plots for convergence for dimension sizes $n = 10$, $n = 50$, and $n = 100$:





We notice that the plots are identical for each dimension. Therefore storage format does not affect the number of iterations until convergence for either of the iterative methods. However, we do want to consider the efficiency of CSR storage over sparse matrix storage, in terms of speed. We display tables demonstrating the time performance of Jacobi, GS, and SGS on the different matrix storage representations. We will also analyze the overall convergence results for each method based on the spectral radius of the matrices tested. Examine the following tables for dimension sizes $n = 10$, $n = 50$, $n = 100$:

Method	Iterations	Spectral Radius	Sparse Time	Compressed Time
Jacobi	7.0	0.09221	0.00119	0.00039
GS	5.0	0.00626	0.00108	0.00027
SGS	4.0	0.00068	0.00115	0.0003

Method	Iterations	Spectral Radius	Sparse Time	Compressed Time
Jacobi	10.6	0.24844	0.00721	0.00555
GS	6.4	0.05474	0.01249	0.00371
SGS	5.2	0.00517	0.01375	0.00426

Method	Iterations	Spectral Radius	Sparse Time	Compressed Time
Jacobi	9.8	0.24723	0.0243	0.01578
GS	6.6	0.07586	0.04711	0.011
SGS	4.8	0.00589	0.06094	0.01529

Oberall, we observe all three methods perform well on the matrices we tested. This can be attributed to the diagonal dominance and symmetry of the sparse matrices. On average, we observe the methods taking less than 10 iterations to converge for each dimension, and the matrices have low average spectral radii at each dimension. GS and SGS outperform Jacobi, converging in less iterations on average. Comparing the time to convergence between sparse matrix storage and CSR storage, we observe that the matrices converge in significantly less

time for compressed storage, for all three methods. This is due to the decreased computational overhead for compressed matrices, since we store and iterate only through the nonzero values, thus eliminating the time on computations with zeros. We also notice that for sparse storage, Jacobi runs in less time than GS and SGS. For compressed storage, however, the times to convergence are close in value for all three methods. Therefore compressed storage improves the time complexity for all three methods.

6 Conclusions

For symmetric-positive definite (SPD) matrices, the non-stationary Conjugate Gradient (CG) method demonstrates superior convergence rates (particularly, convergence in less iterations) for higher dimensions, in comparison to the non-stationary Steepest Descent and stationary Richardson's First Order methods. We observed the Jacobi, Gauss-Seidel (GS), and Symmetric Gauss-Seidel (SGS) stationary methods are effective for diagonally dominant or SPD matrices, with SGS typically outperforming Jacobi and GS due to its symmetric (forward and backward) sweeps. We also discussed storage format for sparse matrices, demonstrating Compressed Sparse Row (CSR) representation significantly reduces time complexity for all methods tested.

Our results necessarily highlight computational efficiency and convergence behavior, suggesting that SGS or CG should be preferred for well-conditioned SPD systems. We may extend this research by exploring different preconditioning techniques (the application of stationary or non-stationary damping factors) for ill-conditioned matrices to expand the applicability of these methods. Overall, iterative solvers are valuable for solving large-scale numerical problems, particularly when direct methods are computationally infeasible or complex.

References

[1] Alfio Quarteroni and Fausto Saleri, "Numerical Methods for Scientists and Engineers," *Springer*, 2000. https://link.springer.com/book/10.1007/978-3-662-04166-0, [Online; accessed 26-November-2024].

[2] David Bindel, "Lecture 31: Iterative Methods for Linear Systems," *CS 6210: Numerical Linear Algebra*, Cornell University, 2012. https://www.cs.cornell.edu/~bindel/class/cs6210-f12/notes/lec31.pdf, [Online; accessed 26-November-2024].

[3] Dr. Kyle A. Gallivan, "Set 6: Sparse Computational Primitives", Class Lecture, MAD5403: Foundations of Computational Mathematics I, Florida State University, November 2024.

[4] Dr. Kyle A. Gallivan, "Set 7: Iterative Methods for Linear Systems: Part 1", Class Lecture, MAD5403: Foundations of Computational Mathematics I, Florida State University, November 2024.

[5] Dr. Kyle A. Gallivan, "Set 8: Iterative Methods for Linear Systems: Part 2", Class Lecture, MAD5403: Foundations of Computational Mathematics I, Florida State University, November 2024.

[6] Dr. Kyle A. Gallivan, "Set 9: Iterative Methods for Linear Systems: Part 3", Class Lecture, MAD5403: Foundations of Computational Mathematics I, Florida State University, November 2024.

[7] Dr. Kyle A. Gallivan, "Set 10: Iterative Methods for Linear Systems, Part 4", Class Lecture, MAD5403: Foundations of Computational Mathematics I, Florida State University, November 2024.

[8] Wikipedia contributors, "Preconditioner," *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Preconditioner, [Online; accessed 26-November-2024].

[9] Wikipedia contributors, "Conjugate Gradient Method," *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Conjugate_gradient_method, [Online; accessed 26-November-2024].

[10] Wikipedia contributors, "Modified Richardson Iteration," *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Modified_Richardson_iteration, [Online; accessed 26-November-2024].

[11] Wikipedia contributors, "Jacobi Method," *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Jacobi_method, [Online; accessed 26-November-2024].

[12] Wikipedia contributors, "Gauss–Seidel Method," *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/GaussSeidel_method, [Online; accessed 26-November-2024].

[13] Yousef Saad, "Iterative Methods for Sparse Linear Systems," 2nd ed., SIAM, 2003. https://www-users.cse.umn.edu/~saad/IterMethBook_2ndEd.pdf, [Online; accessed 26-November-2024].

[14] Zhijun Xu, "Lecture 7.3: Preconditioning and Iterative Methods," *ACMS 40390: Numerical Analysis*, University of Notre Dame, Fall 2014. https://www3.nd.edu/~zxu2/acms40390F14/Lec-7.3.pdf, [Online; accessed 26-November-2024].