FCM Project 3

Chase Wiederstein

November, 13th 2023

1 Executive Summary

This paper analyzes the differences in using the preconditioned steepest descent (PSD) and the preconditioned conjugate gradient (PCG) methods, two foundational algorithms essential for efficiently solving large-scale linear systems and optimization problems in fields such as machine learning, engineering, and scientific computing. In this study, I consider a $n \times n$ symmetric positive definite (SPD) matrix A and will utilize three preconditioners: the identity, Jacobi, and symmetric Gauss-Seidel preconditioners. The analysis includes evaluating the outcomes of PSD and PCG methods with different matrix generation techniques for $n = 20$ and $n = 200$. The performance metrics, including convergence speeds and errors, will be discussed below.

2 Description of the Algorithms and Implementation

2.1 Generating A

I employed two distinct methods for generating a symmetric positive definite matrix: direct symmetrization and a Cholesky-based construction. For the direct symmetrization, I generated A_1 by randomly generating double-precision floating point numbers in C++ between 0 and 1 ensuring $A_1[i, j] = A_1[j, i]$ for all i, j. Additionally, a constant of 10 was added to the diagonal elements of A_1. For the Cholesky-based construction, I created A_2 using the Cholesky decomposition $A_2 = LL^T$. First, a random lower triangular matrix L was generated with double-precision floating point numbers between 0 and 1. Next, I calculate $A_2 = LL^T + 5I$ which guarantees symmetric positive definiteness due to the properties of Cholesky decomposition. The Cholesky decomposition also offers significant benefits in terms of storage space and computation time. By storing only the matrix L instead of all of A_2, memory usage is reduced, which is crucial for large-scale matrices in solving linear systems. Moreover, the triangular structure of L and L^T allows for efficient forward and backward substitution saving in computation time. The addition of constants 10 and 5 were added to the diagonal entries of the matrices to positively influence the numerical stability and condition of A_1 and A_2.

2.2 Generating Preconditioners

For evaluating the performance of preconditioners, I employed several methods starting with the identity matrix as the simplest. Subsequently, I implemented the Jacobi preconditioner setting $P_{Jacobi} = diag(A)$. The more interesting preconditioner used was the symmetric Gauss-Seidel given as $P_{SGS} = (D-E)D^{-1}(D-E^T)$ and its Cholesky decomposition: $P_{SGS} = (D - E)D^{-1/2}D^{-1/2}(D - E^T) = CC^T$ where C is a lower triangular matrix and C^T is an upper triangular matrix. P_{SGS} was computed by setting $diag(A) = D$ and setting the upper triangular elements of A to be E. Its Cholesky factorization will help simplify solving the preconditioned residual involved in the PSD and PCG algorithms.

2.3 The PSD and PCG Methods

Both algorithms are parametrized by A, P, ϵ and b. Matrices A and P are generated as described previously, ϵ is a fixed tolerance set at 10^{-6}, and b is a randomly generated vector of size n. Beginning with x_0 initialized as the zero vector, the algorithms then compute $r_0 = b - Ax_0$, which initially simplifies to $r_0 = b$. As for

solving the preconditioned residual $Pz_0 = r_0$, if $P = I$ or $P = P_{Jacobi}$, z_0 as well as z_k for all $k > 0$ can be computed with $O(n)$ computational cost. Alternatively, for $P = P_{SGS}$, the algorithm will incorporate forward and backward substitution solving $Cy = r_0$ and $C^T z_0 = y$ with $O(n^2)$ computational cost. The primary distinction between the two algorithms lies in PCG's utilization of a set of A-orthogonal directional vectors p_k, which are theoretically expected to enhance convergence more effectively than PSD. During the iterative process, both algorithms update x_k and r_k until the convergence criterion $||r_{k+1}||_2 < 10^{-6}$ is met. Below are the full algorithms.

Algorithm 1 Preconditioned Steepest Descent (PSD)

Input: A, P, b, tolerance ϵ
Output: Approximate solution x
Set initial guess $x_0 = 0$
Compute $r_0 = b - Ax_0$
Solve $Pz_0 = r_0$
for $k = 0, 1, 2, \ldots$ **do**
  $\alpha_k = \frac{z_k^T r_k}{z_k^T A z_k}$
  $x_{k+1} = x_k + \alpha_k z_k$
  $r_{k+1} = r_k - \alpha_k A z_k$
  if $\|r_{k+1}\| < \epsilon$ **then**
    break
  end if
  Solve $Pz_{k+1} = r_{k+1}$
end for

Algorithm 2 Preconditioned Conjugate Gradient (PCG)

Input: A, P, b, tolerance ϵ
Output: Approximate solution x
Set initial guess $x_0 = 0$
Compute $r_0 = b - Ax_0$
Solve $Pz_0 = r_0$
Set $p_0 = z_0$
for $k = 0, 1, 2, \ldots$ **do**
  $\alpha_k = \frac{r_k^T z_k}{p_k^T A p_k}$
  $x_{k+1} = x_k + \alpha_k p_k$
  $r_{k+1} = r_k - \alpha_k A p_k$
  if $\|r_{k+1}\| < \epsilon$ **then**
    break
  end if
  Solve $Pz_{k+1} = r_{k+1}$
  $\beta_k = \frac{r_{k+1}^T z_{k+1}}{r_k^T z_k}$
  $p_{k+1} = z_{k+1} + \beta_k p_k$
end for

3 Description of the Experimental Design and Results

During each iteration of both algorithms, I recorded the values of $||r_k||_2$ and the relative error $\frac{||x_k - x^*||_2}{||x^*||_2}$ to assess the accuracy of the current solution estimate. Note that x^* is the true solution to the system $Ax = b$ and that the relative error will only be recorded if the true solution is known. Using R, I plotted the logarithm of these errors against the iteration number to visualize the performance of each method and preconditioner.

3.1 Results for Matrix A_1

Recall that I generated A_1 by randomly generating double-precision floating-point numbers between 0 and 1, ensuring the matrix was symmetric, and adding 10 to each randomly generated diagonal value. The results of the log of the residual two norm versus the iteration number for matrix A_1 can be seen in figures 1-4. It is evident that using P_{SGS} outperforms the other preconditioners, resulting in faster convergence for both $n = 20$ and $n = 200$. For SPD matrices such as A_1, P_{SGS} retains symmetry and positive definiteness, an advantage for iterative solvers. Additionally, P_{SGS} has a smoothing effect on error components, leading to faster convergence. Another interesting observation can be seen in Figure 2 which reveals fluctuations or "jumping" in the data, highlighting the inefficiency of PSD compared to PCG. PSD often retraces the same direction multiple times due to not using A-orthogonal p_k vectors, which PCG utilizes to enhance convergence.

Table 1: Summary Statistics of Iterations per Method for A_1

Method	Preconditioner	Median	IQR	Minimum	Maximum
PSD	Identity	274	34.5	219	347
PSD	Jacobi	268	32	223	313
PSD	SGS	109	6.5	97	126
PCG	Identity	23	2	22	26
PCG	Jacobi	23	1	22	26
PCG	SGS	15	0	14	16

In the table above, I conducted 100 trials to gather this data for matrix size $n = 200$. From this table, it is evident for both the PCG and PSD methods that P_{SGS} converges in the fewest steps on average relative to $P = I$ and P_{Jacobi}. P_{SGS} also has the smallest interquartile range (IQR) indicating minimal variability in the number of iterations required for convergence possibly due to its smoothing effect on error components. $P = I$ and P_{Jacobi} exhibit statistically similar performance. Figures 5 and 6 offer an alternative visualization of the data presented in Table 1 for PSD and PCG, respectively.

3.2 Results for Matrix A_2

Recall that A_2 was generated with a lower triangular matrix L such that $A = LL^T = 5I$. The results for this matrix, represented by the graphs of the logarithm of the residual two-norm versus the iteration number, are depicted in Figures 7-10. These results underscore the significant impact of matrix generation on the performance of the two methods. Similar to A_1, employing P_{SGS} resulted in quicker convergence for PSD. However, for PCG, P_{SGS} converged in a similar number of iterations as the other preconditioners when $n = 20$, but required many more iterations when $n = 200$. The cause of this remains unclear. It is possible that the Cholesky-based construction of A_2, when combined with P_{SGS}, may influence the conditioning number of A_2 in a way that results in slower convergence. However, further testing would be necessary to confirm this hypothesis.

Table 2: Summary Statistics of Iterations per Method for A_2

Method	Preconditioner	Median	IQR	Minimum	Maximum
PSD	Identity	6051	88.5	5859	6159
PSD	Jacobi	5791	179.5	5337	6145
PSD	SGS	893	91.75	732	1114
PCG	Identity	36	0	35	37
PCG	Jacobi	45	1	42	46
PCG	SGS	85	4	76	91

This table was generated the same way as the table for matrix A_1 for $n = 200$. The performance metrics for each preconditioner and method exhibit notable differences compared to those obtained for A_1. For the PSD

method, the results for each preconditioner show a significantly reduced spread relative to the median, an expected outcome due to higher iterations as compared to the results from A_1. As already mentioned, P_{SGS} underperforms the other preconditioners in the PCG method. Furthermore, P_{SGS} exhibits a much greater spread in iteration counts compared to the other two preconditioners for PCG. Notably, this is the only case where the absence of a preconditioner ($P = I$) statistically outperforms the other two preconditioners in terms of convergence speed and consistency. Figures 11 and 12 present violin plots for a further analysis of tables 1 and 2 offering a visual representation of the data distribution.

3.3 Correctness test

To check the accuracy of our code, we were tasked with solving for x in the system $Ax = b$ using the following:

$$A_{test} = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 11 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix} b_{test} = \begin{bmatrix} 57 \\ 79 \\ 88 \\ 86 \end{bmatrix}$$

To perform this test, I initialized x_0 to be the zero vector and set the stopping criteria to be $||r_k||_2 < 10^{-6}$. For each method and preconditioner, I successfully obtained the solution $x = [1, 2, 3, 4]^T$. For the PCG method, each preconditioner converged after four iterations, which is expected since methods using conjugate directions will terminate after at most n steps. Notably, using P_{SGS} resulted in the least error compared to the other preconditioners, a consistent finding. For the PSD method, the convergence speed was slower than that of the PCG method, again a consistent finding to previously mentioned data. The graphs showing the logarithm of the residual two-norm versus the iteration number can be found in Figures 9-10. After confirming the correctness of the solution $x = [1, 2, 3, 4]^T$, I calculated the relative error $\frac{||x_k - x^*||_2}{||x^*||}$ to monitor the accuracy of the methods. This data is presented in Figures 13-16.

4 Conclusion

This assignment involved several interconnected components that significantly influenced one another. The method chosen to generate matrix A substantially affects the approach for solving $Ax = b$ as seen in the large difference in iterations to converge between A_1 and A_2. P_{SGS} generally outperformed $P = I$ and P_{Jacobi}, particularly for the preconditioned steepest descent method. The only scenario that P_{SGS} underperformed was for the Cholesky-based construction of A_2 for a 200×200 matrix for the PCG method. This suggests that while P_{SGS} can be a robust choice, the optimal preconditioner and solution method should be determined based on the specific characteristics of the matrix A. Ultimately, this assignment highlights the importance of a tailored approach to solving linear systems, emphasizing that the choice of method and preconditioner should be informed by the matrix's properties to achieve the best performance.

5 Tables and Figures

Key: Green is for using the P_{SGS}, red is for P_{Jacobi}, and blue is for $P = I$.
Figure 1:

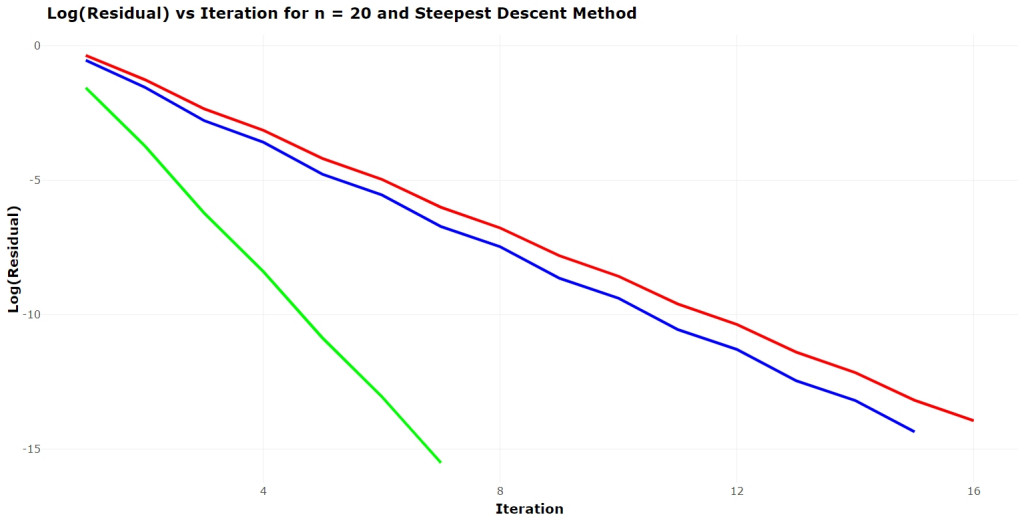


Figure 2:

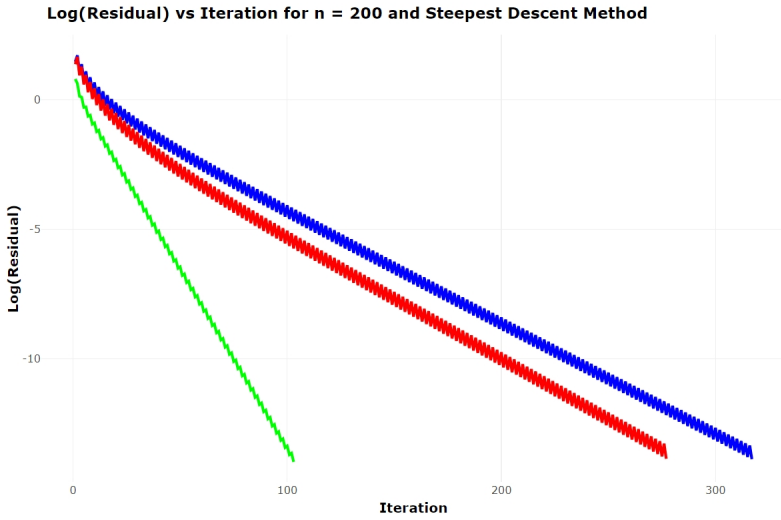


Figure 3:

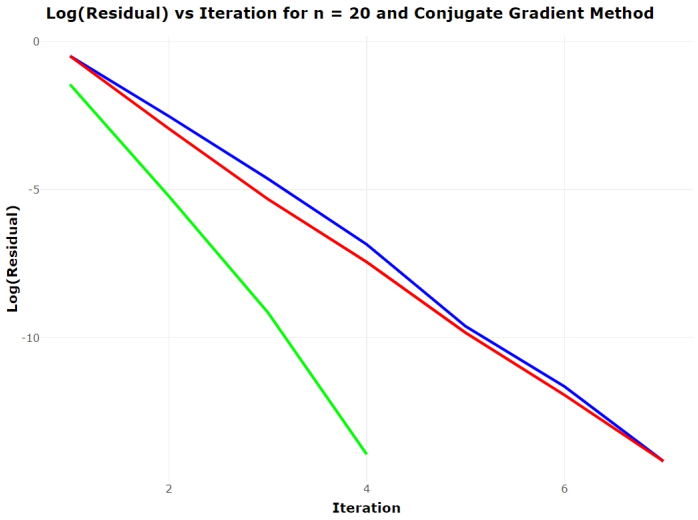


Figure 4:

Figure 5:

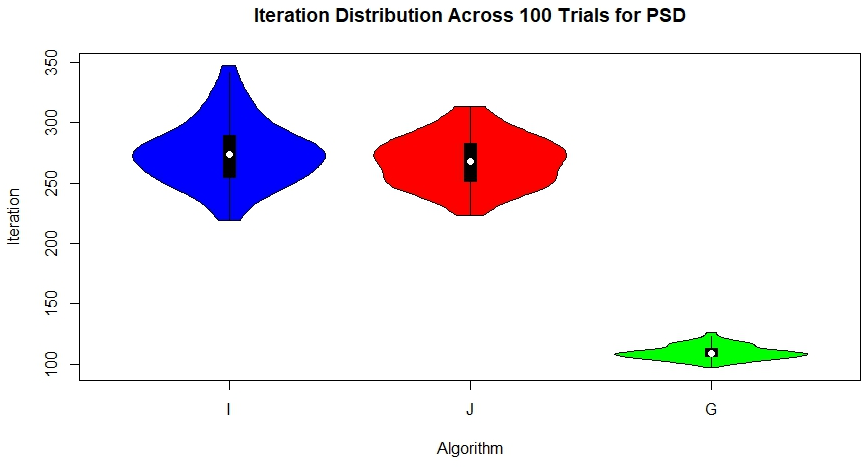


Figure 6:

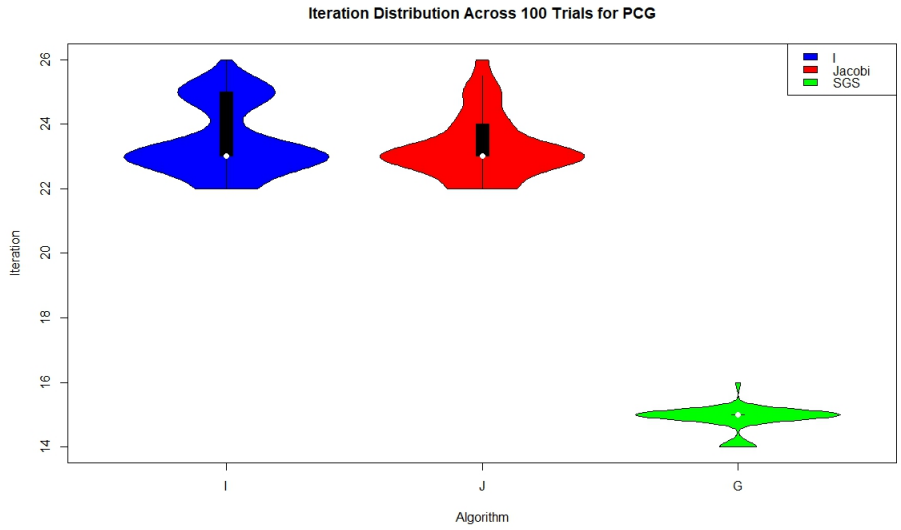


Figure 7:

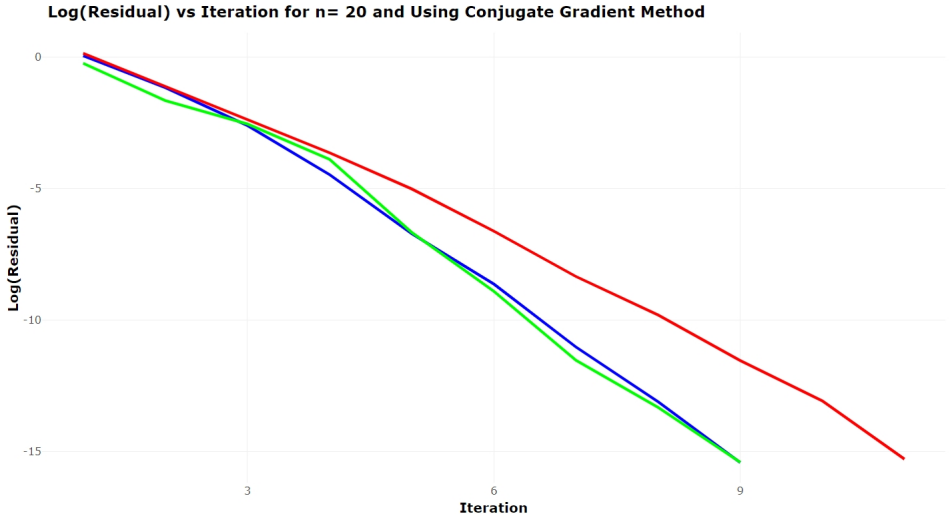


Figure 8:

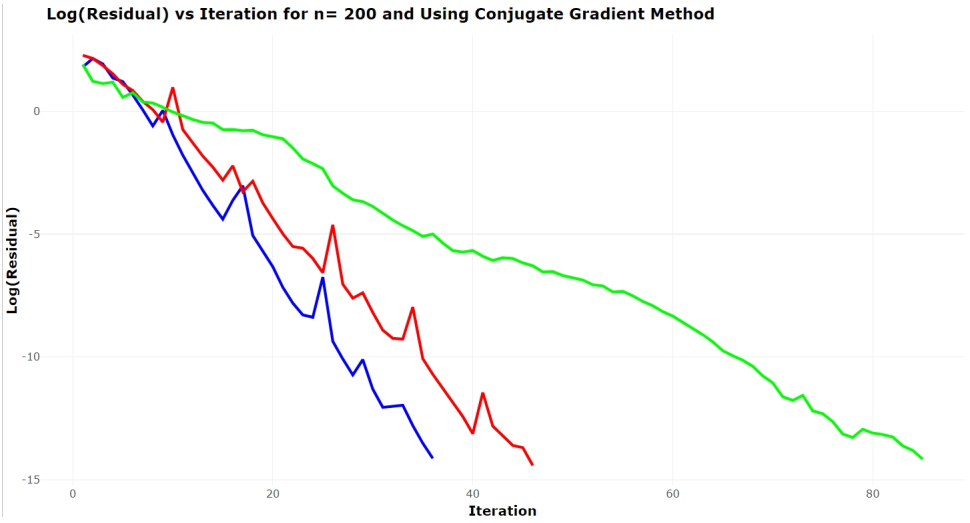


Figure 9:

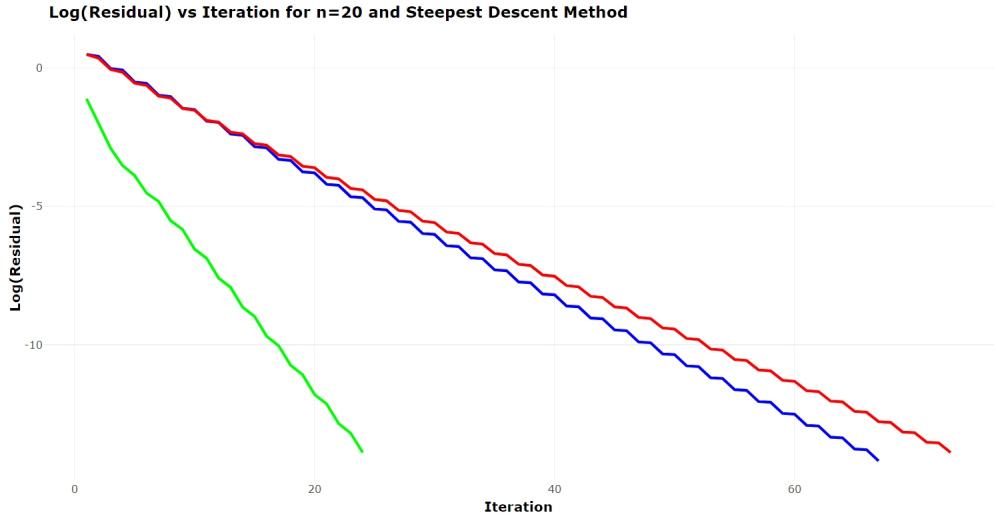


Figure 10:

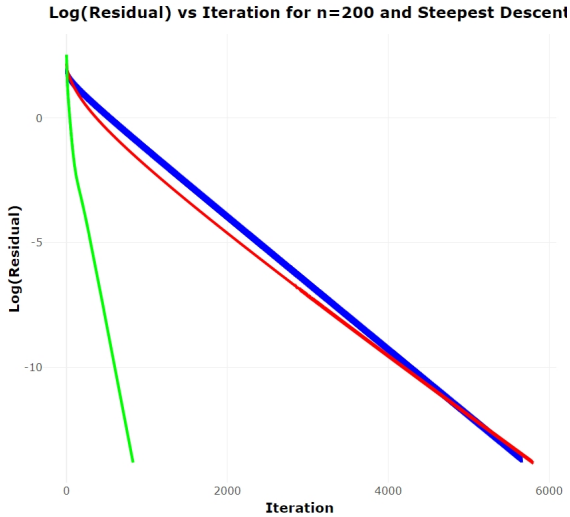


Figure 11:

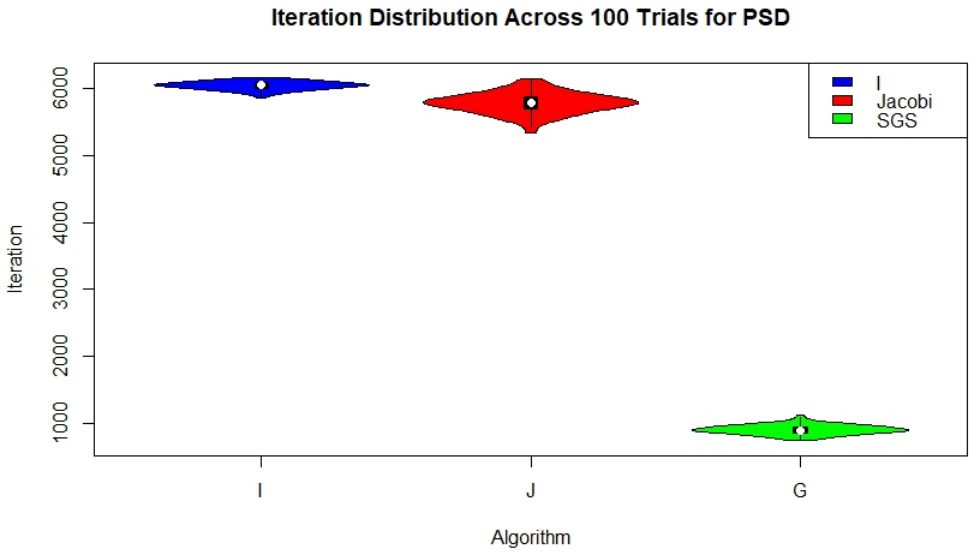


Figure 12:

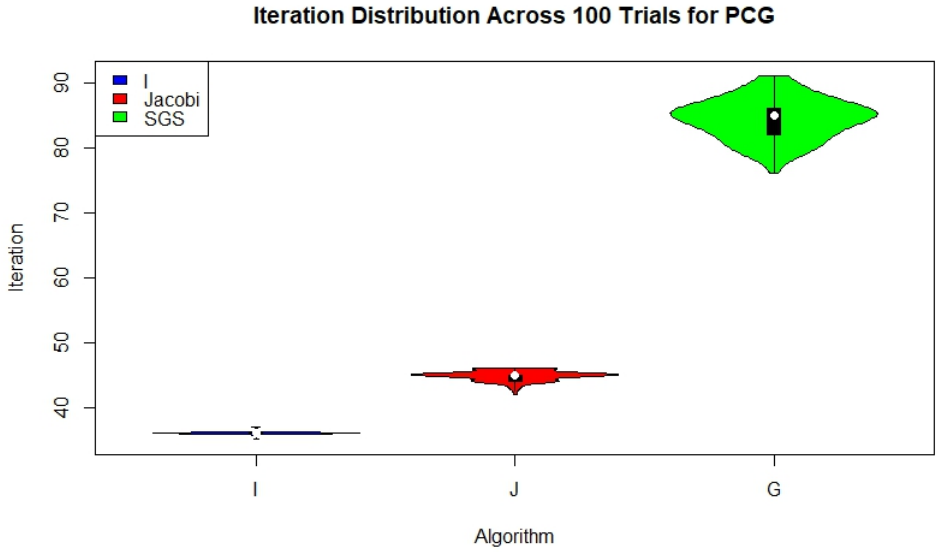


Figure 13:

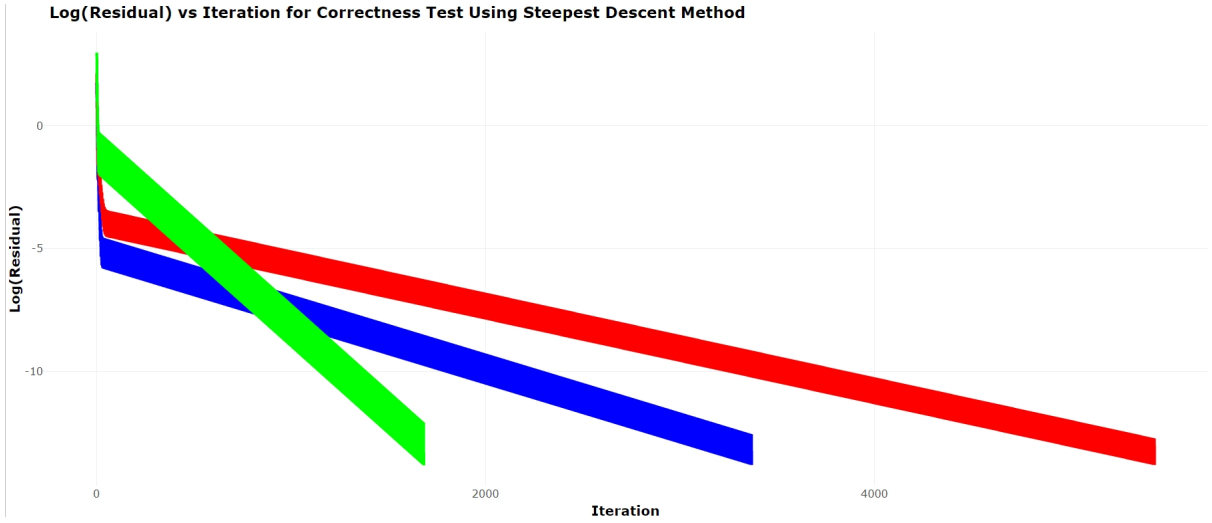


Figure 14:

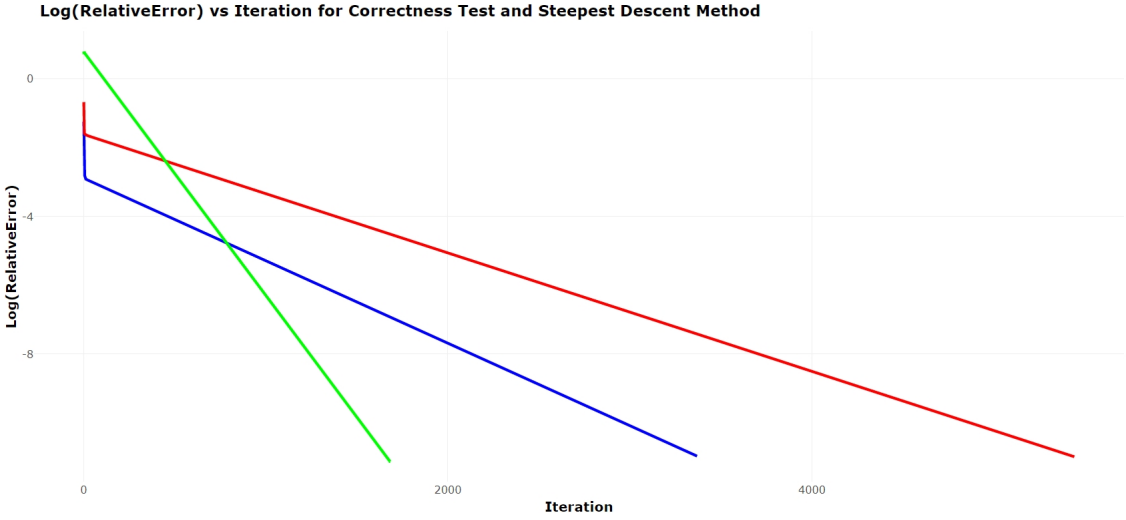


Figure 15:

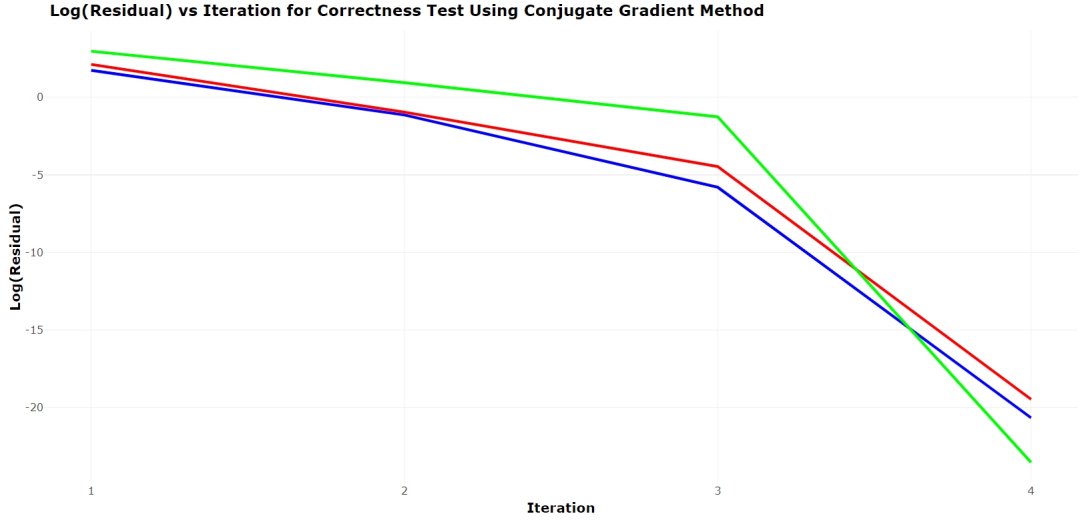
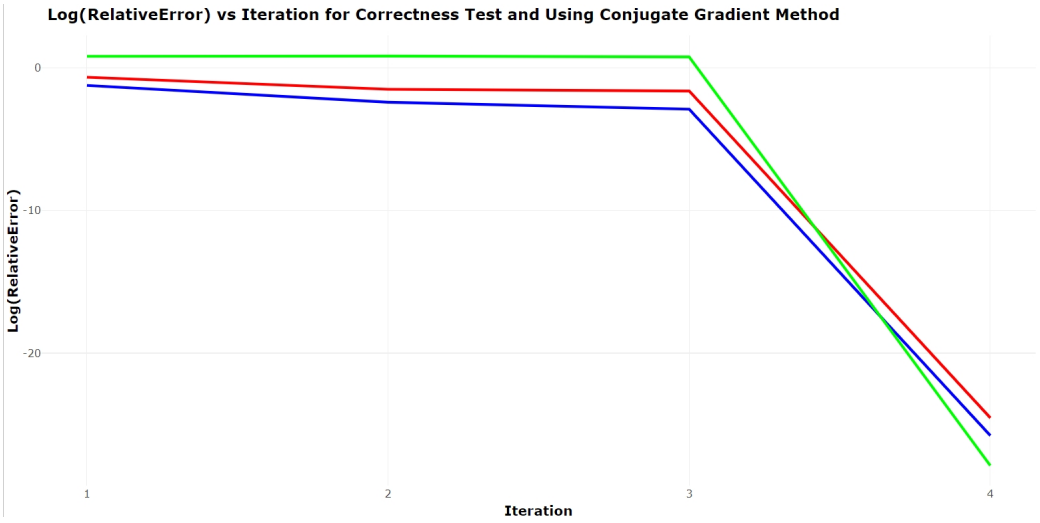


Figure 16:


Log(RelativeError) vs Iteration for Correctness Test and Using Conjugate Gradient Method
0
-10
-20
Log(RelativeError)
1
2
3
4
Iteration