# FCM Project 3

Jonathan Engle

November, 13th 2023

## 1 Executive Summary

In this report we look to solve the classical $Ax = b$ problem by utilizing important methods . For this problem we consider Symmetric positive definite matrices and perform the Steepest Descent method as well as the Conjugate gradient method. For the implementation of these programs we consider three different types of preconditions and analyze the convergence speed as well as error. We compare our results to the theory behind these methods, which is to be further discussed. Multiple tables and figures are used to graphically depict our results.

## 2 Statement of the Problem

For this particular problem we move to analyze $Ax = b$ using the Steepest Descent method as well as the Conjugate gradient method. To do this, we generate a Symmetric positive definite matrices on which we call $A$ this matrices contain entries in $\mathbb{R}$. The user can also add additional elements to these diagonals to create a diagonally dominant symmetric positive matrix. The theory suggests that any symmetric positive matrix will converge for these methods, further analysis of convergence speed for diagonally dominant matrices vs non diagonally dominant matrices in Section 4. We consider problems where the exact solution $x$ is known as well as solutions where $x$ is unknown. We use Steepest Descent method as well as the Conjugate gradient method to approximate a real solution. To monitor how well our approximation is we monitor $\frac{\|r_k\|}{\|b\|}$ and tell our code to stop accordingly with this condition.

Furthermore, to analyze these issues we consider three major pre-conditioners which our algorithms take into account. The first and most trivial pre-conditioner is the identity matrix which consists of size $n$ which is predetermined prior to running our algorithms. The second pre-conditioner is the Jacobi, which consists of extracting our diagonal entries of the randomly generated $A$ and taking the square root of each entry. The third and final pre-conditioner we consider is the Symmetric Gauss Seidel pre-conditioner which is a lower triangular matrix consisting of the lower composition of $A$ divided by the sqaure root of their respective diagonal entry. It is important to note that the theory and intuition behind these pre-conditioners comes from the fact that if $A$ is symmetric positive definite then the Jacobi pre-conditioner is symmetric and positive definite since the diagonal elements of $A$ must be positive which we ensure with the range chosen. Also that the Symmetric Gauss Seidel pre-conditioner has a Cholesky factorization when $A$ is symmetric positive definite. Which implies that the Symmetric Gauss Seidel pre-conditioner is also symmetric positive definite hence the name.

## 3 Description of the Algorithms and Implementation

The following algorithms are implemented to solve our problem of $Ax = b$. The overall goal of these algorithms is to build and test random matrices with the two given methods of Steepest Descent as well as the Gradient method. With these generations we are able to implement the three different

## 3.1 Generate a random matrix $A \in \mathbb{R}^{n\times n}$

First and foremost, we must generate a random matrix $A \in \mathbb{R}^{n\times n}$ which is square and non-singular. Note for this method entries are randomly generated on the closed interval $[1, 10]$. While randomly generated matrices tend to be non-singular and reasonably conditioned, multiple checks are in place in our code to verify these two conditions. First, to verify that the matrix is non-singular we compute the determinant of the matrix and if it is equal to zero we have the program stop and print out that there is an issue prompting the user to try again. Also, to ensure that our matrix is well conditioned we use the condition command in **Matlab** to check that our matrix is well conditioned and suited for our problem.

An additional method we consider in generating a random matrix $A$ is by generating a random square matrix $\hat{A}$ and computing a symmetric positive definite matrix $A$ as $A = \hat{A}\hat{A}^T + I * \gamma$ where $\gamma$ is an integer inputted by the user to have influence over the diagonal entries of $A$. For example if the user was to input $\gamma = 0$ then we would just be left with $A = \hat{A}\hat{A}^T$ which is still SPD. To verify that the matrix is square we have the user input an $n$ which is globalized and used in generating our $A$, for the purposes of this assignment we test matrices for $n = 20$ and $n = 200$.

## 3.2 Steepest Descent and Conjugate Gradient methods

We first discuss the steepest Descent method. To implement this algorithm we set an initial guess $(x_0)$ to be a vector of zeros and then take the randomly generated $A, P, b$ as arguments and solve for $\tilde{x} = x_k, r - 0 = b - Ax_0$ and $Pz_0 = r_0$. This method then runs until our error is under the desired threshold, multiple error thresholds were considered while testing but the standard was set at $10^{-6}$. It is important to note that we are able to compute $Pz_{k+1} = r_{k+1}$ directly by taking advantage of the structure of $P$ being lower triangular for all pre-conditioners with $n$ computational cost using the same backward substitution method from our previous project.

In addition to the steepest Descent method, we consider the Conjugate gradient method. In a similar fashion, this method takes in arguments $A, P, b$ as well as our initial guess of zeros, and and performs the Conjugate gradient method which solves for $Pz_0 = r_0, p_0 = z_0$. This method also takes advantage of the structure of our pre-conditioners being lower triangular and efficiently solves for $Cy = r_{k+1}$ and $C^T z_{k+1} = y$ using forward and backward substitution respectively.

## 3.3 Pre-conditioners

To generate the pre-Conditioners for the implementation of these methods we first generate the identity matrix trivially. Next, we generate the Jacobi pre-conditioner by extracting the diagonal entries of the generated $A$ and applying the square root operator. Finally, we generate the Symetric Gauss Seidel pre-conditioner by inputting the lower triangular matrix elements consisting of the lower composition of $A$ divided by the square root of their respective diagonal entry. Visually they look as follows (Identity not shown):

$$Jacobi = \begin{bmatrix} \sqrt{a_{11}} & 0 & \cdots & 0 \\ 0 & \sqrt{a_{22}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sqrt{a_{nn}} \end{bmatrix}, SGS = \begin{bmatrix} \frac{a_{11}}{\sqrt{a_{11}}} & 0 & \cdots & 0 \\ \frac{a_{21}}{\sqrt{a_{11}}} & \frac{a_{22}}{\sqrt{a_{22}}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{a_{n1}}{\sqrt{a_{11}}} & \frac{a_{n2}}{\sqrt{a_{22}}} & \cdots & \frac{a_{nn}}{\sqrt{a_{nn}}} \end{bmatrix}$$

# 4 Description of the Experimental Design and Results

To test the methods listed above we generate a random matrix using the algorithm above in **Matlab**. We run through our checks as listed above to ensure non-singularity and well condition of our matrix $A$ as well as using proven methods to generate a true SPD matrix. For our testing we consider a randomly generated symmetric positive definite matrix $A \in \mathbb{R}^{n\times n}$ with $n = 20, 200$. Entries for our matrix $A$ are pulled from the interval $[0, 10]$. We then test these randomly generated SPD matrices using the steepest Descent method as well as the Conjugate gradient method with the three pre-conditioners listed above. With initial guesses of zeros and a maximum error threshold of $10^{-6}$ we were able to find that the Conjugate gradient method with the Symmetric Gauss Seidel pre-conditioner converged the fastest under these conditions. This is expected as the Conjugate gradient method is said to converge in at most $n$ steps. We are able to graphically depict these results in tables and charts down bellow. Due to the exponential decay nature of the error of these methods we can also graph the logs against the itteration number to obtain a linear graph which is monotonically decreasing. Furthermore, in our graphs we are able to test the convergence under different thresholds which is to be shown and depicted in the final section. We move to discuss a more specific example in the next section with the correctness test.

## 4.1 Correctness test

To evaluate if our code and implementation has run correctly and efficiently we consider the following correctness test. For the correctness test of our algorithms we consider the following matrix $A_{test}$ and vector $\vec{b}_{test}$ as follows

$$A_{test} = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix}, \vec{b}_{test} = \begin{bmatrix} 57 \\ 79 \\ 88 \\ 86 \end{bmatrix}$$

Prior to running the correctness test we set our stop condition for $\|r_k\|_2 = \|b - Ax\|_2 < 10^{-6}$ and our initial guess to be $[0, 0, 0, 0]^T$. With this test matrix and vector $b$ set up we are able to proceed solving $Ax = b$. Via both the Steepest descent method and the Conjugate gradient method we are able to obtain a solution $x = [1, 2, 3, 4]^T$. To check the amount of steps it takes for this method to converge under the desired threshold of error we obtain the following table.

| Method/ Preconditioner | Identity | Jacobi | Symmetric GS |
|---|---|---|---|
| Steepest Descent | 4,201 steps | 4,062 steps | 3,971 steps |
| Conjugate Gradient | 4 steps | 4 steps | 4 steps |

Via this table we are able to see that the methods are relatively the same. That is due to the simplicity of our matrix, as shown for the larger matrices the number of steps towards convergence will vary depending on the method. The second row of this table further verifies that the conjugate gradient method converges in at most $n$ steps, since our matrix is a $4 \times 4$ matrix it adds validity to our method that we have converged in 4 steps for this method. This result is further verifies the important theorem of **Theorem 4.11** page 155.

Furthermore, we notice that the steepest Descent method takes a long time to converge. We are able to compute directly the condition number of the $A_{test}$ matrix which is approximately 300. This makes sense since the steepest Descent method takes so long to converge. On this note we move to discuss the relationship between these methods and the spectrum of the initial matrix.

## 4.2 Extra Credit

In this section we consider the influence of the spectrum and its relationship with the Steepest Descent method as well as the Conjugate Gradient method. We consider $\Lambda \in \mathbb{R}^{n\times n}$ to be a diagonal matrix with positive elements with a wide variety of ranges for testing. What really matters when discussing the spectrum and these methods is

$$K_2(A)\frac{\lambda_{max}}{\lambda_{min}} = \rho(A)\rho(A^{-1})$$

Notice from this that given any entries of the diagonal matrix $\Lambda$ $\Lambda_{ii} \in R^{n\times n}$ the lowest this condition number can be is 1. This makes sense intuitively as given in the handout notes which states that: "the behavior of CG and SD depends only on the spectrum, by using the eigenvectors then the solution iterates in the two coordinate systems are related via $Q$ and $QT$ and the errors at each step have the same 2-norm, i.e. convergence behavior is identical in both coordinate systems." We can observe this relationship via our testing where we pick a special case where $\Lambda$ is an SPD matrix with the similar eigenvalues with size $n = 200$ $K(A) = 1$, this particular matrix converges quickly and the errors are relatively small. Note, that we can scale this matrix by any scalar and obtain the same result as $K_2(A)$ would not change. To analyze one of the worst possible results, we consider a different case with the same matrix but choose our $\lambda_{1,1} = 2,000$ which yields a $K_2(\tilde{\Lambda}) = \frac{2000}{1} = 2000$ which takes a long time to converge under our error threshold. The tables bellow depict these errors and run times. Notice how the well conditioned case converges quickly as well for steepest Descent method, compared to the Correctness task. The correctness task contained a $K(A_{\text{test}}) \approx 300$ where as the extra credit matrix had a $K(\Lambda) \approx 1$. We can conclude that the Steepest Descent is more sensitive to the condition number and change in spectrum. This adds validity to using the Conjugate gradient method as we are guaranteed convergence in $n$ steps regardless of spectrum relation via **Theorem 4.11** on page 155. While many additional experiments were conducted, these two specific cases shed light to the actually intuition behind the relationship between the spectrum and these methods, both run time and error were reduced in the simplest case.

## 5 Conclusion

Throughout this assignment we were able to verify the methods of the Steepest Descent and the Conjugate Gradient method with respective Pre- Conditioners. We were able to compare our theoretical lessons learned in class and apply them to verify the theory of Steepest Descent and the Conjugate Gradient method. This project also sheds light on the iterative methods and the complexities of solving $Ax = b$. Optimization problems such as this one are fascinating and allow the user to gain deeper intuition about the issues at hand. One of the largest components from this project is the influence of the spectrum on the iterative methods. We were able to see during in depth testing that if the matrix was made diagonally dominant by rows and columns that the methods would converge faster.

# 6 Tables, Figures and Theorems

## 6.1 Theorems

1. **Theorem 4.10** : Let $A$ be symmetric and positive definite matrix; then the gradient method (Steepest Descent method) is convergent for any choice of initial datum $\mathbf{x}^{(\mathbf{0})}$. Moreover,

$$\|\mathbf{e}^{(k+1)}\|_A \leq \frac{K_2(A)-1}{K_2(A)+1}\|\mathbf{e}^{(k)}\|_A, k = 0, 1, \ldots,$$

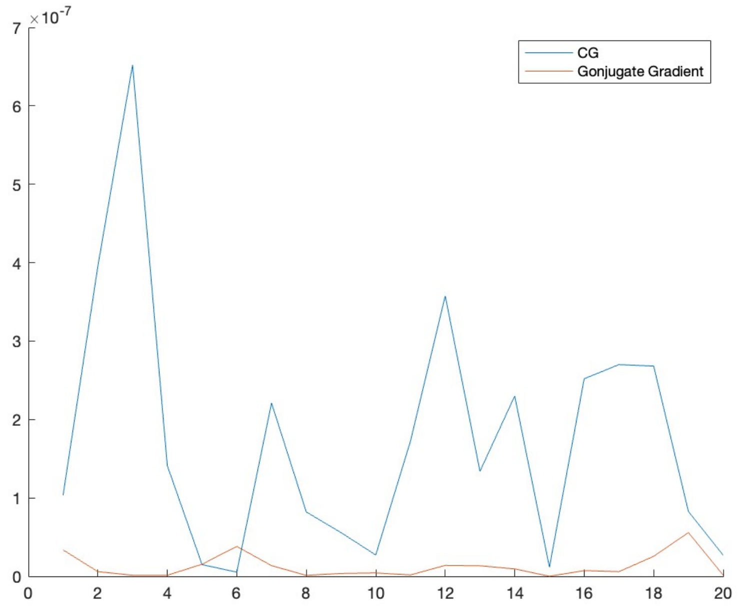
   Where $\|\dot{\|}_A$ is the energy norm.

2. **Theorem 4.11** Let $A$ be a symmetric and positive definite matrix. Any method which implies conjugate directions to solves $Ax = b$ terminates at most $n$ steps, yielding an exact solution.

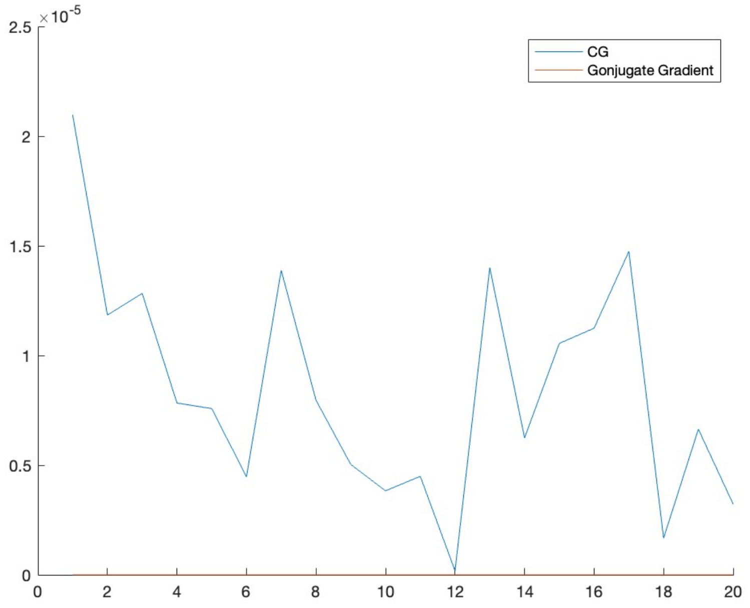
## 6.2 Testing

### 6.2.1 $n = 20$

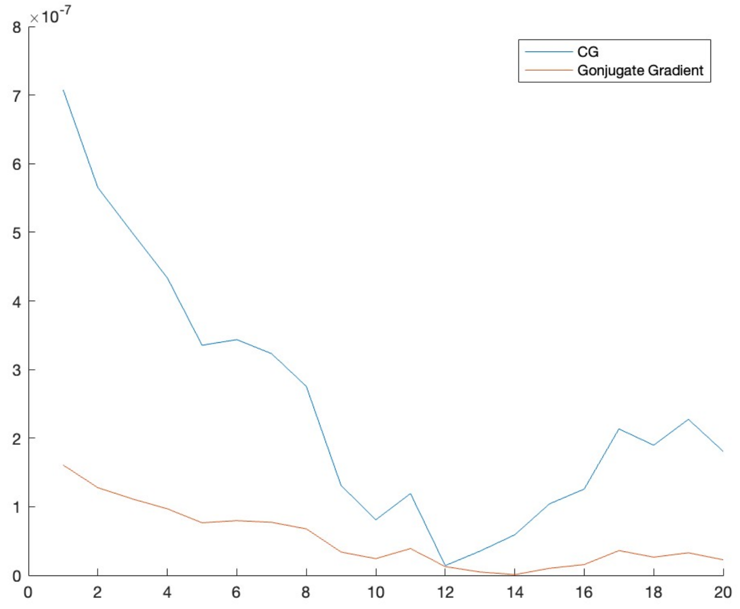
1. No Pre-Conditioner. Please note that there should be no bounce and the code should stop there.



2. Jacobi pre-conditioner. Please note that there should be no bounce and the code should stop there.


×10⁻⁵
2.5
2
1.5
1
0.5
0
0 2 4 6 8 10 12 14 16 18 20
CG
Gonjugate Gradient

3. Symmetric Gauss Seidel pre-conditioner. Please note that there should be no bounce and the code should stop there.

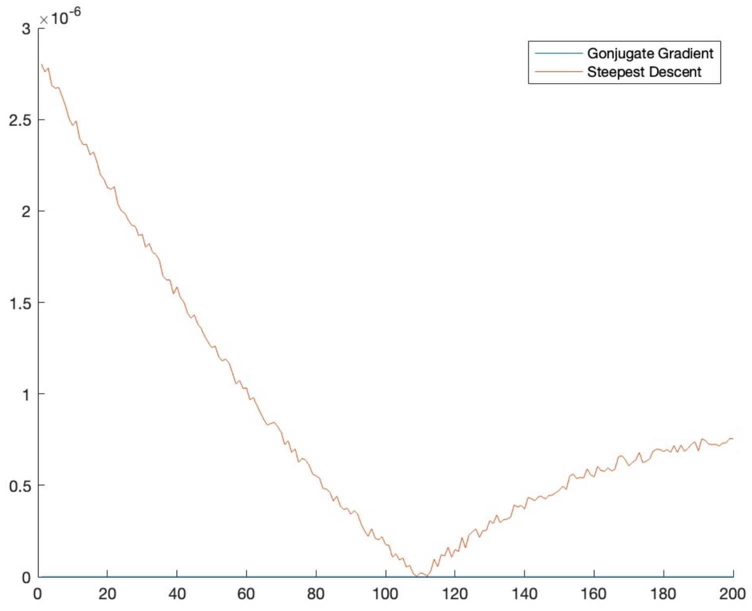


4. Relative Error for $x$

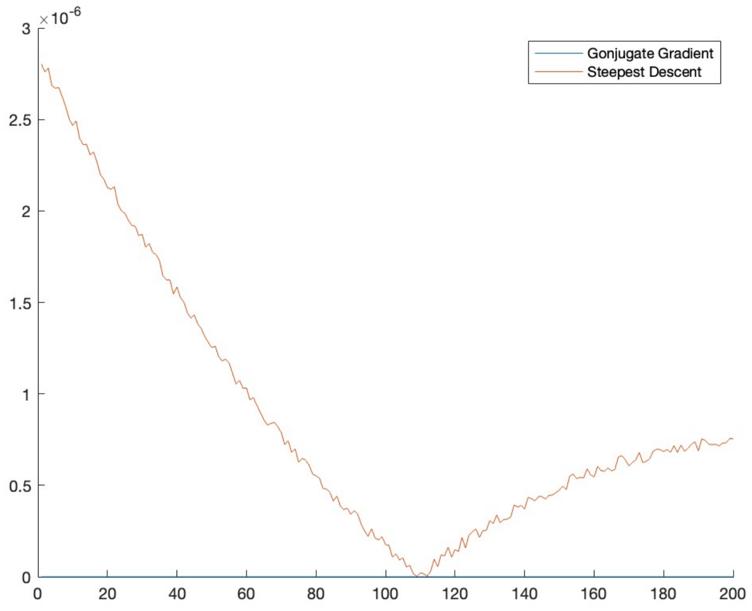
| Method/ Preconditioner | Identity | Jacobi | Symmetric GS |
|---|---|---|---|
| Steepest Descent | $3.6386e^{-6}$ | $2.6179e^{-6}$ | $5.4861e^{-7}$ |
| Conjugate Gradient | $3.6380e^{-6}$ | $3.2723e^{-6}$ | $2.9014e^{-7}$ |

### 6.2.2 $n = 200$

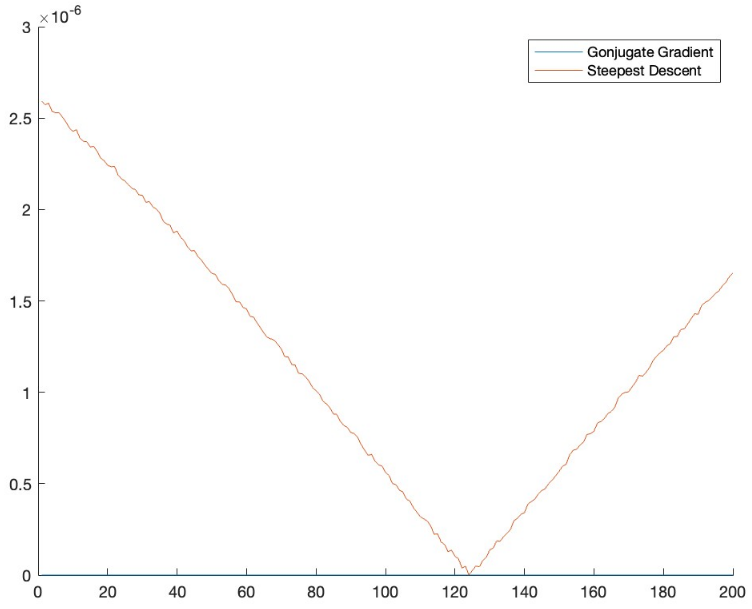
1. No pre-conditioner. Please note that there should be no bounce and the code should stop there.



2. Jacobi pre-conditioner. Please note that there should be no bounce and the code should stop there.



3. Symmetric Gauss Seidel pre-conditioner. Please note that there should be no bounce and the code should stop there.



4. Relative Error for $x$

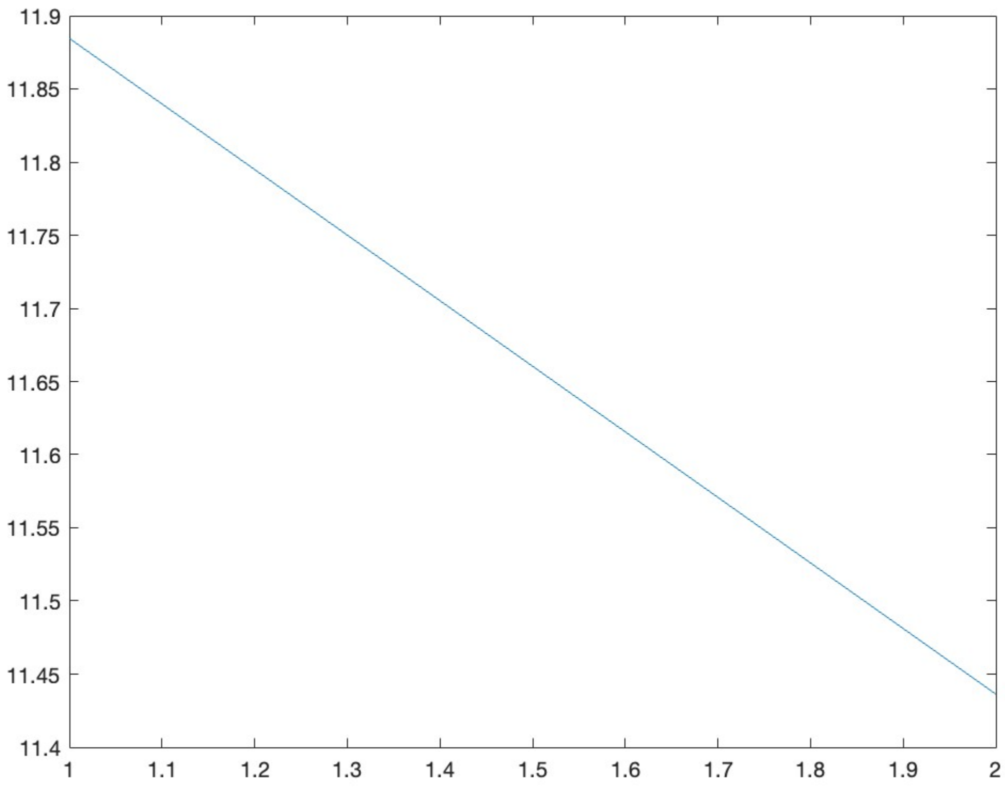
| Method/ Preconditioner | Identity | Jacobi | Symmetric GS |
|---|---|---|---|
| Steepest Descent | $4.2829e^{-6}$ | $2.1421e^{-6}$ | $1.4753e^{-6}$ |
| Conjugate Gradient | $4.2820e^{-6}$ | $1.9420e-24$ | $1.1163e^{-6}$ |

## 6.3 Extra Credit

1. Extra credit observations $n = 200$

| Method/ Matrix type | $\Lambda : \lambda_{max} \approx \lambda_{min}$ | $\hat{\Lambda} : \lambda_{max} >> \lambda_{min}$ |
|---|---|---|
| Steepest Descent | 245 steps | 10,000 steps |
| Conjugate Gradient | 200 steps | 200 steps |

2. Log graph for Steepest Descent Extra Credit: Toy matrix of $\Lambda = \begin{bmatrix} 200 & 0.01 \\ 0.01 & 200 \end{bmatrix}$



11.9
11.85
11.8
11.75
11.7
11.65
11.6
11.55
11.5
11.45
11.4
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2