Coding Report 3

May 22, 2023

1 Problem Description

In this report, we present three scripts implementing iterative methods for solving linear equations: Jacobi, Gauss-Seidel, and SOR methods. In addition, two parameterizations were tested for the SOR method: $\omega = 1.05$ and $\omega = 0.95$. These methods were applied to a specific square matrix $A = \begin{bmatrix} 4 & 1 & -1 \\ -1 & 3 & 1 \\ 2 & 2 & 6 \end{bmatrix}$, as studied in problems 2,3 and 4 from problem set 3, with $b = [5, -4, 1]^T$ to verify their correctness. To assess the effectiveness of these methods, an experiment was conducted using a class of random full-rank matrices, defined as $A = \frac{n}{2}I + \text{random}(n, n)$, where R is an n × n matrix with random entries from a normal distribution with a mean of 0 and a standard deviation of 1, and I is the n × n identity matrix. We also defined the n × 1 vector b with random entries from the same normal distribution. The goal was to compute the solution to the equation Ax = b, iteratively updating until the relative update fell below 10^{-8}. Matrices of sizes n = 10, 25, 50, 100, 200, and 500 were considered. The results of these timing experiments were plotted on two graphs without the errors $\|Ax - b\|_2$ when generating the first two graphs: one showing wall-clock time vs. n, and the other showing the number of iterations vs. n. These graphs provided insights into the computational efficiency of the methods and allowed for observations and discussions on their performance. Furthermore, the residual error $\left\|Ax^{(k)} - b\right\|_2$ was plotted against the iteration number (k) for each method, specifically for the case where n = 500. This analysis provided a visualization of the convergence behavior and the progress made towards minimizing the error. Lastly, experiments were conducted using a matrix family $\hat{A} = R$, and the observations and possible reasons behind the obtained results were discussed. Overall, these experiments and analyses aim to evaluate the performance and effectiveness of iterative methods in solving linear equations.

1.1 Code Description

In the coding environment, we set up the following variables and matrices:
The matrix $A = \begin{bmatrix} 4 & 1 & -1 \\ -1 & 3 & 1 \\ 2 & 2 & 6 \end{bmatrix}$, the vector $b = \begin{bmatrix} 5 \\ -4 \\ 1 \end{bmatrix}$. The tolerance, 'tol', was set to $1e-8$, which determines the desired level of accuracy for convergence. The maximum number of iterations, 'N', was set to 1000. If the iterative methods do not converge within this limit, the computation will stop immediately. Two parameterizations for the SOR method were considered: $\omega = 1.05$ and $\omega = 0.95$. The initial approximation, 'x0', was set to zeros with the same size as the vector b. To generate a class of random full-rank matrices, we used the formula: $A = \frac{n}{2}I + \text{random}(n, n)$, where the identity matrix $I = eye(n)$ and $b = randn(n, 1)$. The goal was to compute the solution to the equation Ax = b using iterative methods, updating the solution until the relative update fell below 10^{-8}. Matrices of sizes $n = 10, 25, 50, 100, 200$, and 500 were considered for this purpose.

2 Results

In this section, we discuss our implementation of three scripts for solving linear equations iteratively: Jacobi, Gauss-Seidel, and SOR methods. Firstly, we applied these methods to a square matrix A and a vector b given by:

$$A = \begin{bmatrix} 4 & 1 & -1 \\ -1 & 3 & 1 \\ 2 & 2 & 6 \end{bmatrix}, b = \begin{bmatrix} 5 \\ -4 \\ 1 \end{bmatrix}$$

which were taken from problems 2, 3, and 4 of problem set 3 to validate the correctness of our implementations. We compared the approximate solutions obtained from our scripts with the hand calculations performed in the third problem set and found them to be the same. Thus, the correctness of the functions is confirmed.

Next, we extended our analysis to solve systems with different matrix sizes: n = 10, 25, 50, 100, 200, and 500. To generate these matrices, we defined a class of random full-rank matrices using the formula:

$$A = \frac{n}{2}I + \text{random}(n, n)$$

We applied three iterative methods (Jacobi, Gauss-Seidel, SOR with $\omega = 1.05$ and $\omega = 0.95$) to solve the system Ax = b for each matrix size. We recorded the number of iterations (k) and the wall-clock time taken by each method. To visualize the performance of each methods, we created two plots (one with wall-clock time vs n in Figure 1 and another with iterations vs n in Figure 2). For the case of n = 500, we created one plot of the error $\|Ax - b\|_2$ vs iteration number (k) for each method in Figure 3. Furthermore, we created a log plot of log of the error $|Ax - b|_2$ vs log of iteration number (k) for the case of $n = 500$ in Figure 4. This log plot provides a clearer visualization of the convergence behavior and allows us to compare the convergence rates of the methods more effectively. In the following pages, we have attached four graphs with one-to-one analysis.

2.1 Graph Analysis of Wall Clock Time vs. n

From Figure 1, we could see that this plot shows the relationship between the matrix size and the time taken by each method. By observing how the execution time varies as the matrix size increases, we could understand the relative performance of these three methods and identify which approach is more efficient for larger matrix sizes.We found that there is a gap between the Jacobi method and the other three methods. The Jacobi method consistently requires more time to solve the system compared to Gauss-Seidel, SOR ($\omega = 1.05$), and SOR ($\omega = 0.95$). This can be seen in the plot where the Jacobi line has the steepest slope, indicating a slower increase in execution time with increasing matrix size. Among the Gauss-Seidel, SOR ($\omega = 1.05$), and SOR ($\omega = 0.95$) methods, their execution times are relatively similar and show a more gradual increase compared to Jacobi. The reason why Gauss-Seidel method has similar computational speed with SOR method is that Gauss-Seidel method is a special case of SOR method with relaxation parameter ω= 1. When $\omega = 1$, SOR method is reduced to Gauss-Seidel method. It is important to note that the reason SOR ($\omega = 1.05$) and SOR ($\omega = 0.95$) are slower than Gauss-Seidel is because the chosen values of the relaxation parameter (omega) are not optimized for achieving the fastest convergence.

The order of execution times from fastest to slowest is as follows: Gauss-Seidel -> SOR ($\omega = 1.05$) -> SOR ($\omega = 0.95$) -> Jacobi. This ranking is determined by comparing the slopes of the lines in the plot, where a steeper slope indicates a slower increase in execution time.

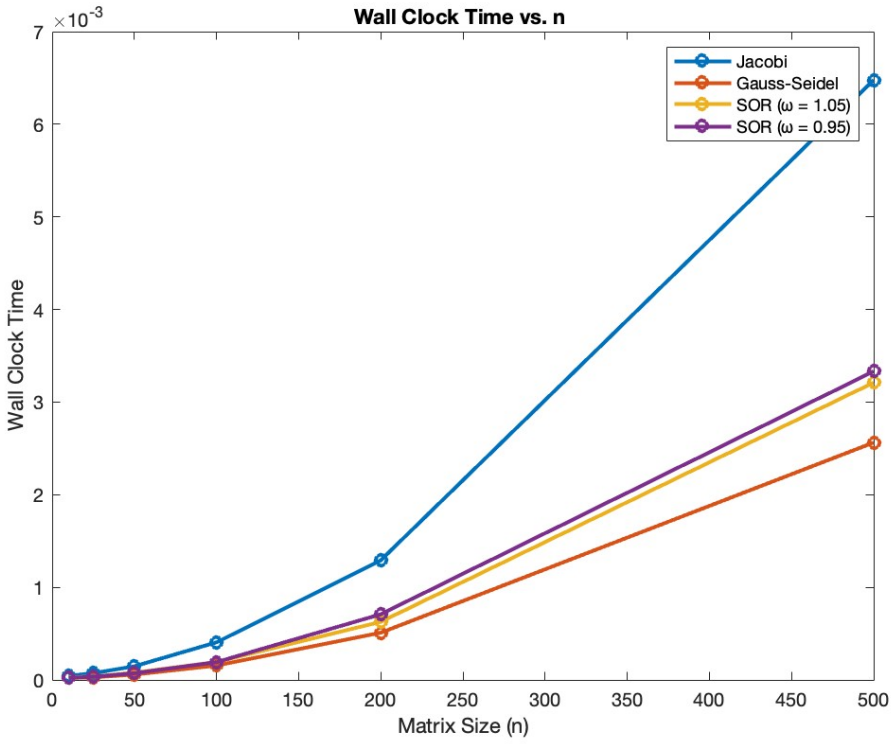

Figure 1: Plot matrix size (n) (x-axis) vs. wall clock time(s) (y-axis) to evaluate the performance of the Jacobi, Gauss-Seidel, and SOR methods. **Reminder:** When generating this graph, I commented out the errors $\|Ax - b\|_2$.

2.2 Graph Analysis of Iterations vs. n

From Figure 2, we could see that this plot illustrates the number of iterations required by each method to converge for different matrix sizes. By observing this plot, we can compare the convergence rates of the methods for different matrix sizes. If a method consistently requires fewer iterations to converge across different problem sizes, it indicates that the method has a faster convergence rate. Conversely, if a method requires a higher number of iterations, it suggests slower convergence. After carefully observing it, we found that the Gauss-Seidel method exhibits fewer iterations with increasing matrix size compared to other methods. This can be attributed to the fact that Gauss-Seidel always uses the most recently calculated values, leading to faster convergence. The Jacobi method is the slowest, because in Jacobi the solution of each iteration is calculated using all the values of the previous iteration, which means that updates are not merged immediately. This can lead to slower convergence because the method relies on the old values to compute the new solution.

- For matrix sizes 10 to 50, the methods in order of fewer to more iterations are: Gauss-Seidel -> SOR ($\omega = 0.95$) -> SOR ($\omega = 1.05$) -> Jacobi.
- For matrix sizes 100 to 200, the methods in order of fewer to more iterations are: Gauss-Seidel -> SOR ($\omega = 0.95$) and SOR ($\omega = 1.05$) have the same number of iterations -> Jacobi.
- For matrix size 500, the methods in order of fewer to more iterations are: Gauss-Seidel -> SOR ($\omega = 0.95$), SOR ($\omega = 1.05$), and Jacobi have the same number of iterations.

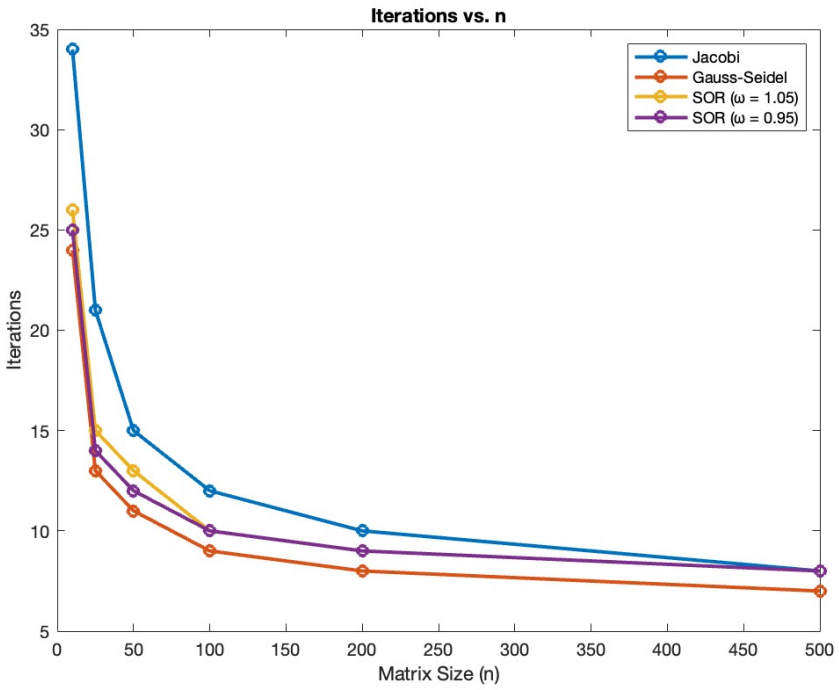

Figure 2: Plot the matrix size (n) (x-axis) versus the number of iterations (y-axis) for the Jacobi, Gauss-Seidel, and SOR methods. **Reminder:** When generating this graph, I commented out the errors $\|Ax - b\|_2$.

2.3 Graph Analysis of Error vs. Iteration Number

For the case of n = 500, we recorded the residual errors $\left\|Ax^{(k)} - b\right\|_2$ in each iteration and plot residual errors vs. iteration number k, shown in Figure 3. From Figure 3, we could see that this plot illustrates the relationship between the iteration number (k) and the residual errors $\left\|Ax^{(k)} - b\right\|_2$ for the case of matrix size n = 500. A smaller residual error indicates a closer approximation to the true solution. In other words, if the residual errors decrease gradually and converge to a small value, it indicates that the method is converging to a solution. Upon observing the plot, it shows that all four lines for the Jacobi, Gauss-Seidel, and SOR methods ($\omega = 1.05$ and ($\omega = 0.95$)) in the case of matrix size n = 500 start at the same point and exhibit similar trends throughout the iterations. The residual errors gradually decrease for all methods, indicating the convergence of the iterative algorithms. To visualize their performance more clearly, we have plotted a logarithmic plot in the following page. The logarithmic scale allows us to better analyze the trend and convergence performance of the methods.

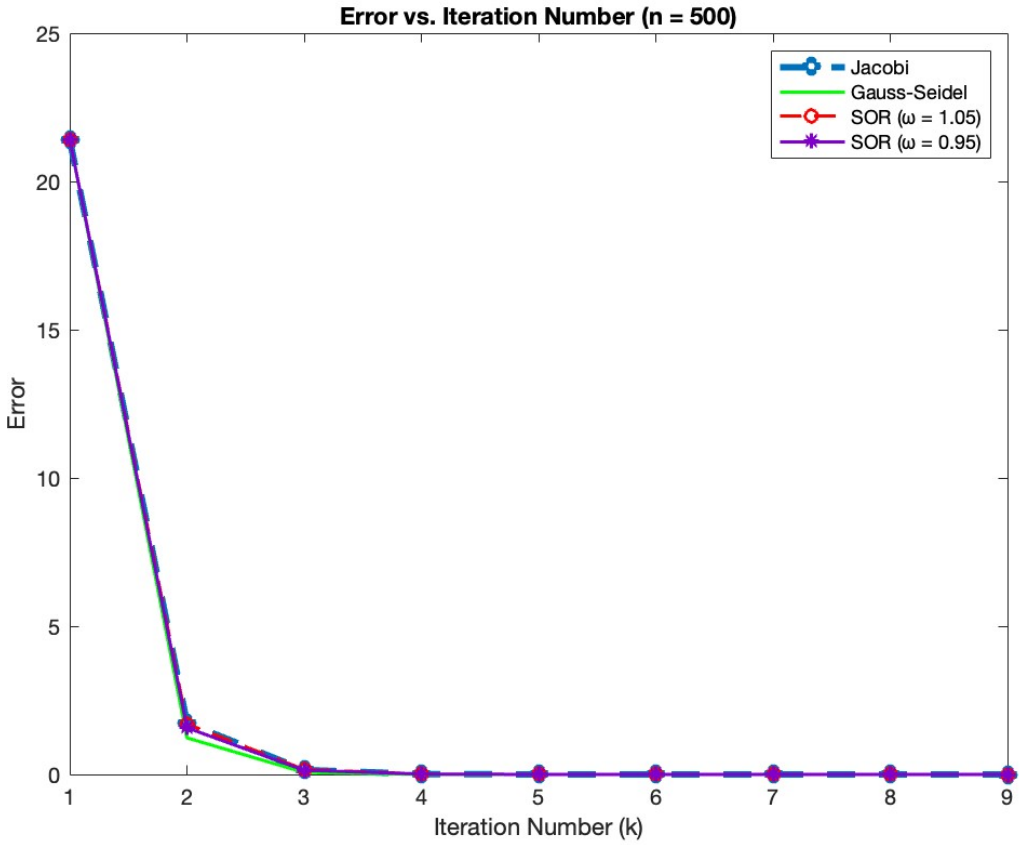

Figure 3: Plot the number of iterations (k) (x-axis) versus the residual error (y-axis) for the Jacobi, Gauss-Seidel, and SOR methods in the case of matrix size n = 500 provides insights into the convergence behavior and accuracy of these methods.

2.4 Graph Analysis of Log of Error vs. Iteration Number

From Figure 4, the log plot provides a clearer visualization of the convergence behavior and allows us to compare the convergence rates of the Jacobi, Gauss-Seidel, and SOR methods ($\omega = 0.95$ and $\omega = 1.05$). The Gauss-Seidel method, as observed in the Log plot, exhibits the fastest convergence rate due to the steeper slope. The convergence rates of SOR ($\omega = 0.95$), Jacobi, and SOR ($\omega = 1.05$) are very close, showing similar convergence behavior for these methods.

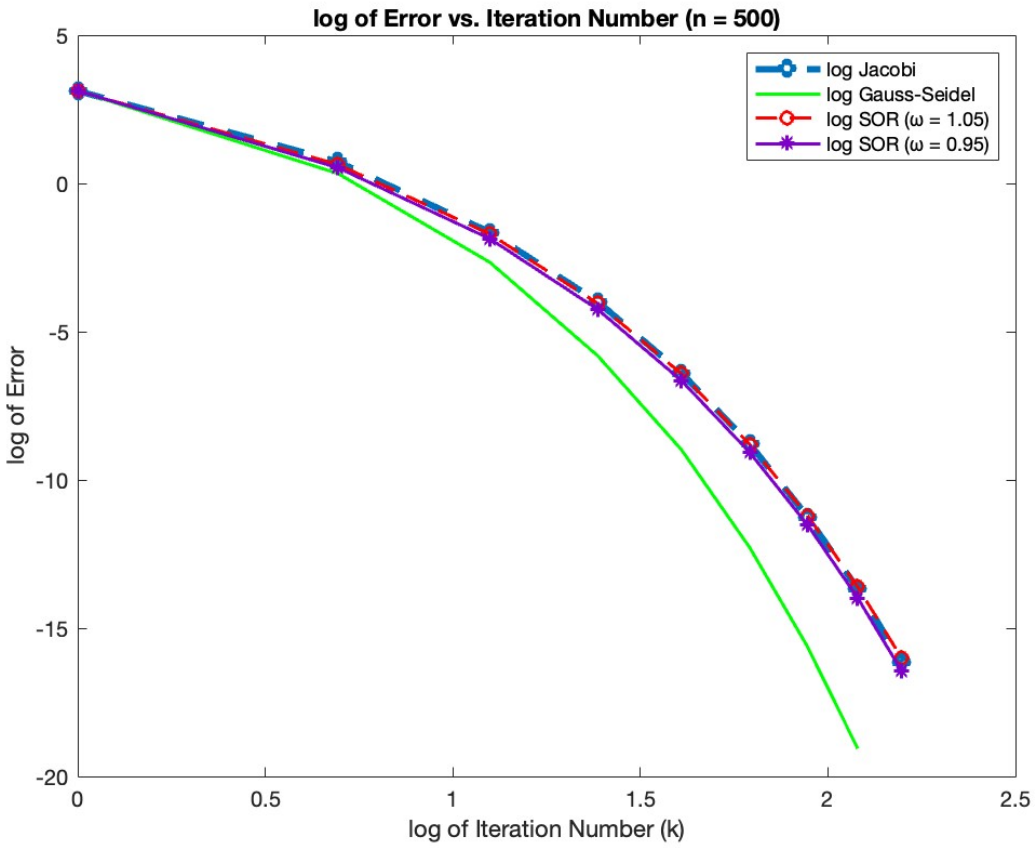

Figure 4: Plot log of number of iteration (k) (x-axis) vs. log of residual error (y-axis) in the case of matrix size n = 500 allows us to evaluate the convergence performance of the Jacobi, Gauss-Seidel, and SOR methods ($\omega = 1.05$ and $\omega = 0.95$).

Lastly, we conducted an experiment using a different matrix family $\hat{A} = R$ instead of $A = \frac{n}{2}I + \text{random}(n, n)$, where R is an n × n matrix with random entries from a normal distribution with a mean of 0 and a standard deviation of 1. We solved the system $\hat{A}x = b$ using the same vector b as defined above. We repeated some experiments and compared the performance of the Jacobi, Gauss-Seidel, SOR with $\omega = 1.05$, and SOR with $\omega = 0.95$ methods for matrix sizes n = 10, 25, 50, 100, 200, and 500. However, in this case, we encountered a different behavior. We found that the maximum number of iterations was always exceeded before the residual errors fell below the desired tolerance of 10^{-8} for all matrix sizes. The reason for this is that $\hat{A} = R$ is not guaranteed to be diagonally dominant, which can lead to instability in the iterative methods. As a result, the iterative sequence may diverge instead of converging to a solution.

3 Collaboration

Yirong Xu and Sabrina Zhu

4 Academic Integrity

On my personal integrity as a student and member of the UCD community, I have not given nor received any unauthorized assistance on this assignment.

5 Appendix

```
clc;clear;close all;

% Matrix from PS3 problem 2 3, 4
A = [4 1 -1; -1 3 1; 2 2 6];
b = [5; -4; 1];
% Tolerance
tol = 1e-8;
% Maximum number of iterations
N = 1000;
% Initial approximation
x0 = zeros(size(b));
% Relaxation parameter for SOR 1.05
w1 = 1.05;
% Relaxation parameter for SOR 0.95
w2 = 0.95;
% Call Jacobi function
[x_jacobi, k_jacobi, t_jacobi] = Jacobi(A, b, tol, N, x0);
% Display solution vector x
disp('Jacobi method:');
disp(x_jacobi);

% Call GaussSeidel function
[x_gaussseidel, k_gaussseidel, t_gaussseidel] = GaussSeidel(A
   , b, tol, N, x0);
% Display solution vector x
disp('Gauss-Seidel method:');
disp(x_gaussseidel);

% Call SOR function
[x_sor1, k_sor1, t_sor1] = SOR(A, b, tol, N, x0, w1);
% Display solution vector x
disp('SOR method (   = 1.05):');
disp(x_sor1);

% Call SOR function
[x_sor2, k_sor2, t_sor2] = SOR(A, b, tol, N, x0, w2);
% Display solution vector x
disp('SOR method (   = 0.95):');
disp(x_sor2);

% --For n= [10, 25, 50, 100, 200, 500]---
% Timing experiments
matrix_sizes = [10, 25, 50, 100, 200, 500];
iterations_jacobi = zeros(size(matrix_sizes));
iterations_gaussseidel = zeros(size(matrix_sizes));
iterations_sor1 = zeros(size(matrix_sizes));
```

```
iterations_sor2 = zeros(size(matrix_sizes));
time_jacobi = zeros(size(matrix_sizes));
time_gaussseidel = zeros(size(matrix_sizes));
time_sor1 = zeros(size(matrix_sizes));
time_sor2 = zeros(size(matrix_sizes));


% ---- For loop through each martix size ----

for i = 1:length(matrix_sizes)
    n = matrix_sizes(i);
    A = (n/2)* eye(n) + randn(n, n);
    b = randn(n, 1);
    x0 = zeros(n, 1);

    tic;
    [~, iterations_jacobi(i),~,ej] = Jacobi(A, b, tol, N, x0)
       ;
    time_jacobi(i) = toc;

    tic;
    [~, iterations_gaussseidel(i),~,eg] = GaussSeidel(A, b,
       tol, N, x0);
    time_gaussseidel(i) = toc;

    tic;
    [~, iterations_sor1(i),~,es1] = SOR(A, b, tol, N, x0, w1)
       ;
    time_sor1(i) = toc;

    tic;
    [~, iterations_sor2(i),~,es2] = SOR(A, b, tol, N, x0, w2)
       ;
    time_sor2(i) = toc;
end


% --- Plotting Wall Clock Time vs. n Matrix Size---
figure;
plot(matrix_sizes, time_jacobi, '-o', 'LineWidth', 2)
hold on
plot( matrix_sizes, time_gaussseidel, '-o', 'LineWidth', 2)
hold on
plot( matrix_sizes, time_sor1, '-o', 'LineWidth', 2)
hold on
plot( matrix_sizes, time_sor2, '-o', 'LineWidth', 2)
xlabel('Matrix Size (n)');
ylabel('Wall Clock Time');
```

```
legend('Jacobi', 'Gauss-Seidel', 'SOR (   = 1.05)', 'SOR (
   = 0.95)');
title('Wall Clock Time vs. n ');


% ---Plotting Iterations vs. n Matrix Size---
figure;
plot(matrix_sizes, iterations_jacobi, '-o', 'LineWidth', 2)
hold on
plot(matrix_sizes, iterations_gaussseidel, '-o', 'LineWidth',
    2)
hold on
plot(matrix_sizes, iterations_sor1, '-o', 'LineWidth', 2)
hold on
plot(matrix_sizes, iterations_sor2, '-o', 'LineWidth', 2)
xlabel('Matrix Size (n)');
ylabel('Iterations');
legend('Jacobi', 'Gauss-Seidel', 'SOR (   = 1.05)', 'SOR (
   = 0.95)');
title('Iterations vs. n');

% The error for n = 500 case
n = 500;
A_hat = randn(n, n);
b_hat = randn(n, 1);
x0_hat = zeros(n, 1);
tol = 1e-8;

% Call Jacobi function and obtain the solution (x)
x_jacobi_hat = Jacobi(A_hat, b_hat, tol, N, x0_hat);

% Call GaussSeidel function and obtain the solution (x)
x_gaussseidel_hat = GaussSeidel(A_hat, b_hat, tol, N, x0_hat)
   ;

% Call SOR function and obtain the solution (x)
x_sor1_hat = SOR(A_hat, b_hat, tol, N, x0_hat, w1);
x_sor2_hat = SOR(A_hat, b_hat, tol, N, x0_hat, w2);

% ----Plot Error vs. Iteration Number (n = 500) ----
figure;
plot(1:length(ej), ej, '--o', 'LineWidth', 3)
hold on
plot(1:length(eg), eg, '-green', 'LineWidth', 1.5)
hold on
plot(1:length(es1), es1, '--or', 'LineWidth', 1.5)
hold on
```

```
plot(1:length(es2), es2, '-*','Color',[0.5 0 0.8], 'LineWidth
   ', 1.5)
xlabel('Iteration Number (k)');
ylabel('Error');
legend('Jacobi', 'Gauss-Seidel', 'SOR (   = 1.05)', 'SOR (
   = 0.95)');
title('Error vs. Iteration Number (n = 500)');


% ----Plot Error vs. Iteration Number (n = 500) ----
figure;
plot(log(1:length(ej)),log( ej), '--o', 'LineWidth', 3)
hold on
plot(log(1:length(eg)), log(eg), '-green', 'LineWidth', 1.5)
hold on
plot(log(1:length(es1)), log(es1), '--or', 'LineWidth', 1.5)
hold on
plot(log(1:length(es2)), log(es2), '-*','Color',[0.5 0 0.8],
   'LineWidth', 1.5)
xlabel('log of Iteration Number (k)');
ylabel('log of Error');
legend('Jacobi', 'Gauss-Seidel', 'SOR (   = 1.05)', 'SOR (
   = 0.95)');
title('log of Error vs. Iteration Number (n = 500)');



%---------------Jacobi--------------
function [x, k, t,error] = Jacobi(A, b, tol, N, x0)
    % A: Matrix
    % b: Matrix
    % tol: Tolerance
    % N: Maximum iteration
    % x0:Initial approximation
    k = 1;
    error=[];
    [n, m] = size(A);
    x = zeros(n, 1);
    t = zeros(n, N);

    while k <= N
        x0 = x;
        error(end+1)=norm(A*x-b);% When generating this first
            two graph, commented out errors
        for i = 1:n
            sum = 0;
            for j = 1:n
                if j ~= i
```

```
                    sum = sum + A(i, j) * x0(j);
                end
            end
            x(i) = (1 / A(i, i)) * (b(i) - sum);
        end


        if norm(x - x0) < tol
            error(end+1)=norm(A*x-b);% When generating this
               first two graph, commented out errors
            break;
        end

        k = k + 1;
    end
end

%---------------GaussSeidel--------------

function [x, k, t,error] = GaussSeidel(A, b, tol, N, x0)
    % A: Matrix
    % b: Matrix
    % tol: Tolerance
    % N: Maximum iteration
    % x0:Initial approximation
    k = 1;
    error=[];
    [n, m] = size(A);
    x = zeros(n, 1);
    t = [];  % Store all iterations of x

    while k <= N
        x0 = x;
        error(end+1)=norm(A*x-b); % When generating this
           first two graph, commented out errors
        for i = 1:n
            sum = 0;
            for j = 1:i-1
                sum = sum + A(i, j) * x(j);
            end
            for j = i+1:n
                sum = sum + A(i, j) * x0(j);
            end
            x(i) = (b(i) - sum) / A(i, i);
        end


        if norm(x - x0) < tol
```

```
            error(end+1)=norm(A*x-b); % When generating this
               first two graph, commented out errors
            break
        end
        k = k + 1;
    end
end

%---------------SOR--------------

function [x, k, t,error] = SOR(A, b, tol, N, x0, w)
    % A: Matrix
    % b: Matrix
    % tol: Tolerance
    % N: Maximum iteration
    % x0:Initial approximation
    k = 1;
    error=[];
    [n, m] = size(A);
    x = zeros(n, 1);
    t = zeros(n, N);  % Store all iterations of x

    while k <= N
        x0 = x;
        error(end+1)=norm(A*x-b); % When generating this
           first two graph, commented out errors
        for i = 1:n
            sum = 0;
            for j = 1:i-1
                sum = sum + A(i, j) * x(j);
            end
            for j = i+1:n
                sum = sum + A(i, j) * x0(j);
            end
            x(i) = (1 - w) * x0(i) + (w * (b(i) - sum)) / A(i
               , i);
        end



        if norm(x - x0) < tol
            error(end+1)=norm(A*x-b); % When generating this
               first two graph, commented out errors
            break
        end
        k = k + 1;
    end
end
```