1 Problem Description

This chapter teach about how to use used 3 different techniques to solve for Ax= b linear system. First method is called Jacobi method. This method will arrange 'n' equation with 'n' unknowns. Then solve those equation by approximation solution. In other two methos gauss-seidel and SOR method we use the same techniques, but we use updated value of unknown from previous approximation.

2 Results

The given report shows the results of 3 iteration using above method to solve Ax= b.
A = [4,1, -1;-1,3,1;2,2,6] and b as [5,-4,1]
As we know (x1,x2,x3) =[0,0,0] for initial approximation (x^k) where k = 0.

Jacobi method

k	K=1	K=2	K=3
X^k1	1.2500	1.5417	1.4306
X^k2	-1.3333	-0.8611	-0.7731
X^k3	-.01667	-0.1389	-0.3935

Gauss-Seidel Method

K	1	2	3
X^k1	1.2500	1.4097	1.3478
X^k2	-0.91167	-0.7708	-0.7575
X^K3	-0.2778	-0.3796	-0.3634

SOR Method (omega = 1.1)

k	1	2	3
X1k	1.3750	1.4102	1.3252
X2k	-0.9625	-0.7307	-0.7614
X3k	-0.3346	-0.3990	-0.3502

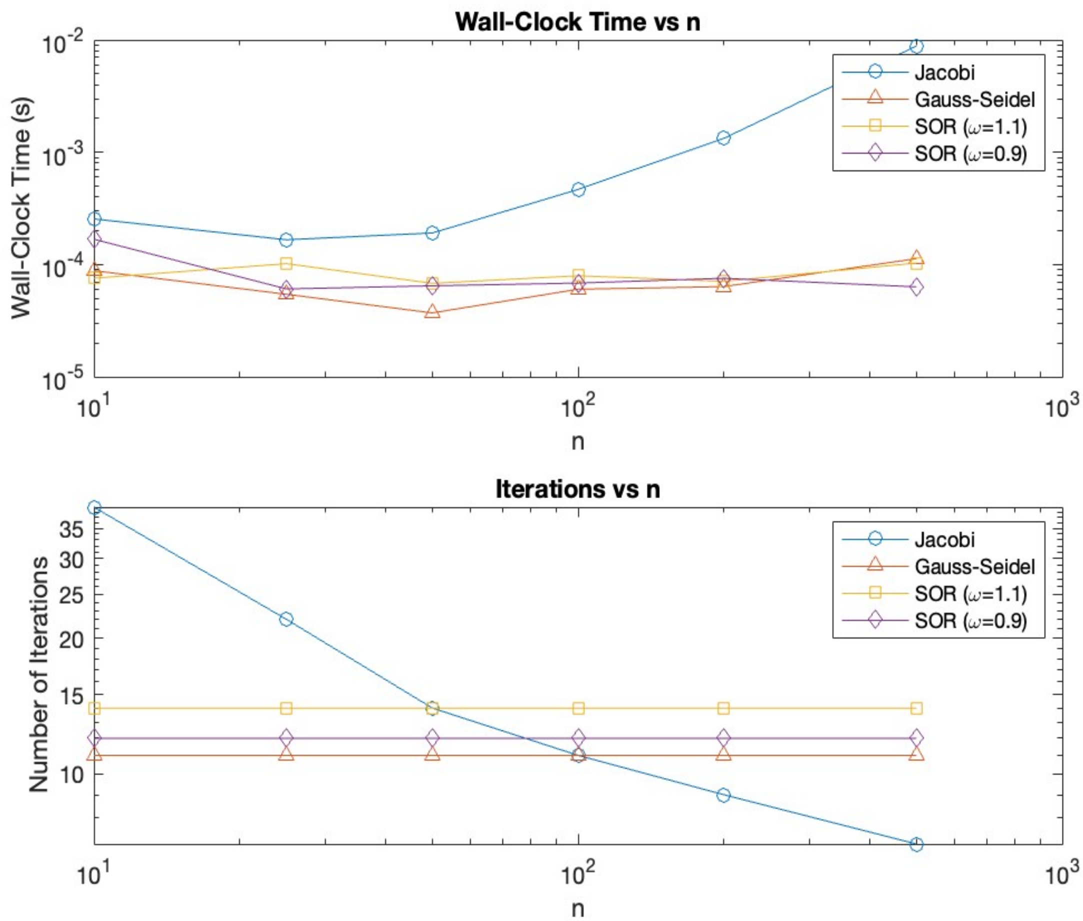
SOR Method (omega= 0.9)

k	1	2	3
X1k	1.250	1.3801	1.3660
X2k	-0.8625	-0.8036	-0.7668
X3k	-0.2288	-0.3458	-0.3643

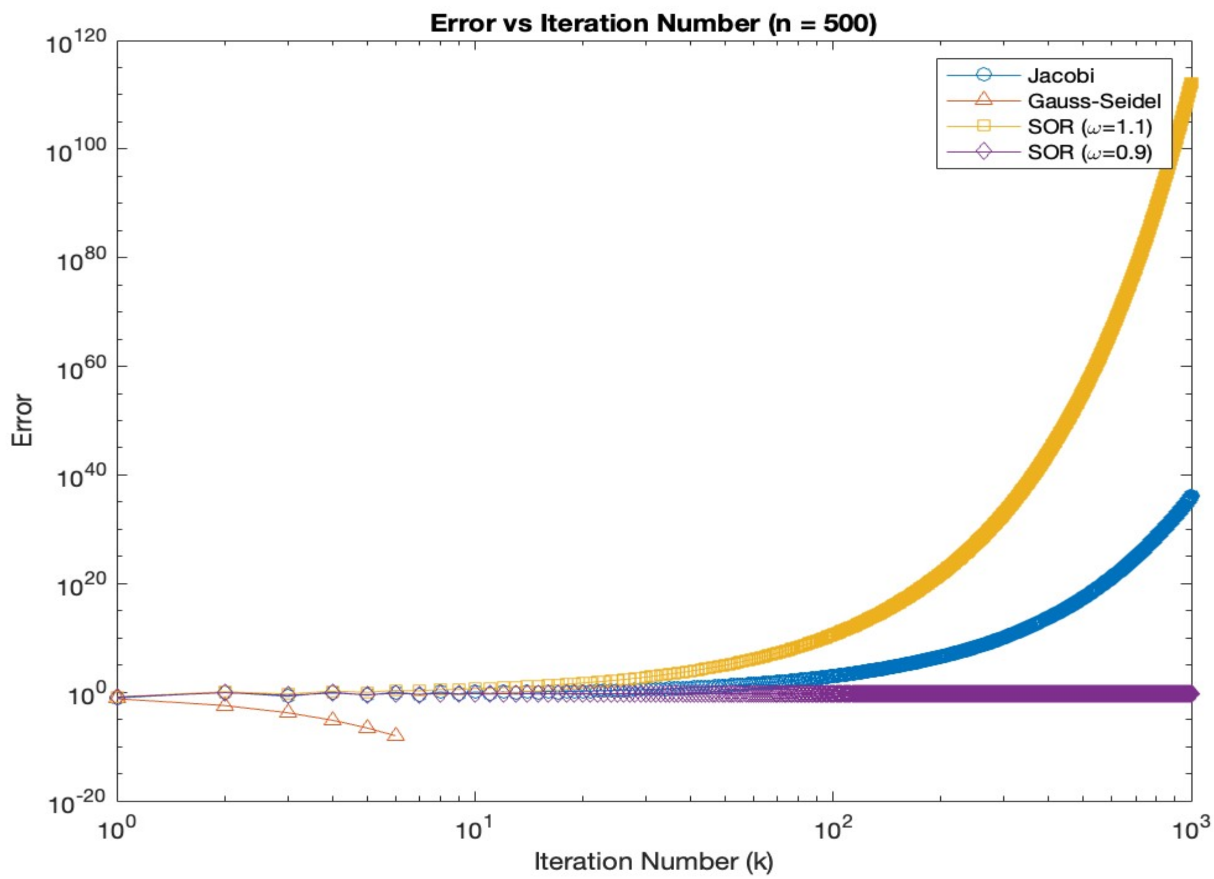
From the below graph we can analyze given information:

Jacobi method took a greater number of iterations to come to solution. It looks like gauss seidel methos is not better than SOR method. But. It might be dependent on value of omega for SOR Method. If omega is big number, then number of iterations is more.so choosing right value for omega is crucial.

Graph below shows number of iterations each method takes as well as the wall-clock time vs n.



Graph below shows error number of iteration.



Above graph impels that error get smaller and converge but when size of matrix is increasing error can be numerous large after doing many iterations.

For a matrix A = rand(n,n) program took too long and output is not generated.

3 Collaboration

No collaboration on this project.

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

Attached code

```
% Jacobi method
function [x, iterations] = jacobi(A, b, x0, tol, max_iterations)
n = length(b);
x = x0;
iterations = 0;
while iterations < max_iterations
    x_old = x;
    for i = 1:n
        sigma = 0;
        for j = 1:n
            if j ~= i
                sigma = sigma + A(i, j) * x_old(j);
            end
        end
        x(i) = (b(i) - sigma) / A(i, i);
    end
    if norm(x - x_old) < tol
        break;
    end
    iterations = iterations + 1;
end
end

% Gauss-Seidel method
function [x_gs, iterations_gs] = gauss_seidel(A, b, x0, tol, max_iterations)
A = [4 1 -1; -1 3 1; 2 2 6];
b = [5; -4; -1];
x0 = [0; 0; 0];
tol = 1e-8;
max_iterations = 1000;
n = length(b);
x_gs = x0;
iterations_gs = 0;

while iterations_gs < max_iterations
    x_old = x_gs;
    for i = 1:n
        sigma1 = 0;
        for j = 1:i-1
            sigma1 = sigma1 + A(i, j) * x_gs(j);
```

```
        end
        sigma2 = 0;
        for j = i+1:n
            sigma2 = sigma2 + A(i, j) * x_old(j);
        end
        x_gs(i) = (b(i) - sigma1 - sigma2) / A(i, i);
    end
    if norm(x_gs - x_old) < tol
        break;
    end
    iterations_gs = iterations_gs + 1;
end



%SOR methods
function [x_sor1, iterations_sor1] = sor(A, b, x0, omega, tol,
max_iterations)
A = [4 1 -1; -1 3 1; 2 2 6];
b = [5; -4; -1];
x0 = [0; 0; 0];
tol = 1e-8;
max_iterations = 1000;

n = length(b);
x_sor1 = x0;
iterations_sor1 = 0;
while iterations_sor1 < max_iterations
    x_old = x_sor1;
    for i = 1:n
        sigma1 = 0;
        for j = 1:i-1
            sigma1 = sigma1 + A(i, j) * x_sor1(j);
        end
        sigma2 = 0;
        for j = i+1:n
            sigma2 = sigma2 + A(i, j) * x_old(j);
        end
        x_sor1(i) = (1 - omega) * x_old(i) + omega * (b(i) - sigma1 - sigma2)
/ A(i, i);
    end
    if norm(x_sor1 - x_old) < tol
        break;
    end
    iterations_sor1 = iterations_sor1 + 1;
end

end

%SOR2
function [x_sor2, iterations_sor2] = sor2(A, b, x0, omega, tol,
max_iterations)
A = [4 1 -1; -1 3 1; 2 2 6];
b = [5; -4; -1];
x0 = [0; 0; 0];
tol = 1e-8;
```

```
max_iterations = 1000;

n = length(b);
x_sor2 = x0;
iterations_sor2 = 0;
while iterations_sor2 < max_iterations
    x_old = x_sor2;
    for i = 1:n
        sigma1 = 0;
        for j = 1:i-1
            sigma1 = sigma1 + A(i, j) * x_sor2(j);
        end
        sigma2 = 0;
        for j = i+1:n
            sigma2 = sigma2 + A(i, j) * x_old(j);
        end
        x_sor2(i) = (1 - omega) * x_old(i) + omega * (b(i) - sigma1 - sigma2)
/ A(i, i);
    end
    if norm(x_sor2 - x_old) < tol
        break;
    end
    iterations_sor2 = iterations_sor2 + 1;
end
end
```

code to call, find error and plot graph

```
% Example usage
A = [4 1 -1; -1 3 1; 2 2 6];
b = [5; -4; -1];
x0 = [0; 0; 0];
tol = 1e-8;
max_iterations = 1000;

% Jacobi method
[x, iterations_jacobi] = jacobi(A, b, x0, tol, max_iterations);
fprintf('Jacobi method:\n');
fprintf('  Solution: x = [%f, %f, %f]\n', x);
fprintf('  Number of iterations: %d\n', iterations_jacobi);

% Gauss-Seidel method
[x_gs, iterations_gs] = gauss_seidel(A, b, x0, tol, max_iterations);
fprintf('Gauss-Seidel method:\n');
fprintf('  Solution: x = [%f, %f, %f]\n', x_gs);
fprintf('  Number of iterations: %d\n', iterations_gs);

% SOR method (omega = 1.1)
omega = 1.1;
[x_sor1, iterations_sor1] = sor(A, b, x0, omega, tol, max_iterations);
fprintf('SOR method (omega = 1.1):\n');
fprintf('  Solution: x = [%f, %f, %f]\n', x_sor1);
fprintf('  Number of iterations: %d\n', iterations_sor1);
```

```matlab
% SOR method (omega = 0.9)
omega = 0.9;
[x_sor2, iterations_sor2] = sor2(A, b, x0, omega, tol, max_iterations);
fprintf('SOR method (omega = 0.9):\n');
fprintf('  Solution: x = [%f, %f, %f]\n', x_sor2);
fprintf('  Number of iterations: %d\n', iterations_sor2);

% Define a class of random full rank matrices
sizes = [10, 25, 50, 100, 200, 500];
solutions = cell(1, length(sizes));
for i = 1:length(sizes)
    n = sizes(i);
    A = (n/2)*eye(n) + randn(n);
    b = randn(n, 1);
    x0 = zeros(n, 1);
    tol = 1e-8;
    max_iterations = 1000;
    [x, iterations_jacobi] = jacobi(A, b, x0, tol, max_iterations);
    while true
        x_old = x;
        [x, iterations_jacobi] = jacobi(A, b, x_old, tol, max_iterations);
        relative_update = norm(x - x_old)/norm(x_old);
        if relative_update < tol
            break;
        end
    end
    solutions{i} = x;
    fprintf('n = %d, iterations = %d, relative update = %e\n', n,
iterations_jacobi, relative_update);
end


sizes = [10, 25, 50, 100, 200, 500];
tol = 1e-8;
max_iterations = 1000;
methods = {'jacobi', 'gauss_seidel', 'sor', 'sor2'};

% Initialize arrays to store timing results
iterations_jacobi = zeros(length(methods), length(sizes));
time_elapsed = zeros(length(methods), length(sizes));

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

    % Timing experiments
    for j = 1:length(methods)
        method = methods{j};
        tic;
        switch method
            case 'jacobi'
                [x, iter] = jacobi(A, b, x0, tol, max_iterations);
            case 'gauss_seidel'
```

```
                [x, iter] = gauss_seidel(A, b, x0, tol, max_iterations);
            case 'sor'
                omega = 1.1;
                [x, iter] = sor(A, b, x0, omega, tol, max_iterations);
            case 'sor2'
                omega = 0.9;
                [x, iter] = sor2(A, b, x0, omega, tol, max_iterations);
        end
        time_elapsed(j, i) = toc;
        iterations_jacobi(j, i) = iter;
    end
end

% Plot the timing results
figure;
subplot(2,1,1)
loglog(sizes, time_elapsed(1,:), '-o', sizes, time_elapsed(2,:), '-^', sizes,
time_elapsed(3,:), '-s', sizes, time_elapsed(4,:), '-d');
title('Wall-Clock Time vs n')
xlabel('n')
ylabel('Wall-Clock Time (s)')
legend('Jacobi', 'Gauss-Seidel', 'SOR (\omega=1.1)', 'SOR (\omega=0.9)')

subplot(2,1,2)
loglog(sizes, iterations_jacobi(1,:), '-o', sizes, iterations_jacobi(2,:), '-
^', sizes, iterations_jacobi(3,:), '-s', sizes, iterations_jacobi(4,:), '-
d');
title('Iterations vs n')
xlabel('n')
ylabel('Number of Iterations')
legend('Jacobi', 'Gauss-Seidel', 'SOR (\omega=1.1)', 'SOR (\omega=0.9)')

finding error of different method:
n = 500;
A =  (n/2)*eye(n) + randn(n);
b = randn(n, 1);
x0 = zeros(n, 1);
tol = 1e-8;
max_iterations = 1000;
omega = 1.1;

% Jacobi method
for iter = 1:max_iterations
    x_jacobi_new = (b - A*x0)./diag(A);
    error_jacobi_norm(iter) = norm(A*x_jacobi_new - b, 2)/norm(b, 2);
    if error_jacobi_norm(iter) < tol
        break;
    end
    x0 = x_jacobi_new;
end
iter_jacobi = iter;
x = x_jacobi_new;

% Gauss-Seidel method
x0 = zeros(n, 1);
```

```
for iter = 1:max_iterations
    x_gs_new = x0;
    for i = 1:n
        x_gs_new(i) = (b(i) - A(i, 1:i-1)*x_gs_new(1:i-1) - A(i,
i+1:n)*x0(i+1:n))/A(i, i);
    end
    error_gs_norm(iter) = norm(A*x_gs_new - b, 2)/norm(b, 2);
    if error_gs_norm(iter) < tol
        break;
    end
    x0 = x_gs_new;
end
iter_gs = iter;
x_gs = x_gs_new;

% SOR method (omega = 1.1)
x0 = zeros(n, 1);
for iter = 1:max_iterations
    x_sor_w1p1_new = (1-omega)*x0 + omega*(b - A*x0)./diag(A);
    error_sor_w1p1_norm(iter) = norm(A*x_sor_w1p1_new - b, 2)/norm(b, 2);
    if error_sor_w1p1_norm(iter) < tol
        break;
    end
    x0 = x_sor_w1p1_new;
end
iter_sor_w1p1 = iter;
x_sor_w1p1 = x_sor_w1p1_new;

% SOR method (omega = 0.9)
omega = 0.9;
x0 = zeros(n, 1);
for iter = 1:max_iterations
    x_sor_w0p9_new = (1-omega)*x0 + omega*(b - A*x0)./diag(A);
    error_sor_w0p9_norm(iter) = norm(A*x_sor_w0p9_new - b, 2)/norm(b, 2);
    if error_sor_w0p9_norm(iter) < tol
        break;
    end
    x0 = x_sor_w0p9_new;
end
iter_sor_w0p9 = iter;
x_sor_w0p9 = x_sor_w0p9_new;

% Plot the error vs iteration number for each method
figure;
loglog(1:iter_jacobi, error_jacobi_norm, '-o', 1:iter_gs, error_gs_norm, '-
^', 1:iter_sor_w1p1, error_sor_w1p1_norm, '-s', 1:iter_sor_w0p9,
error_sor_w0p9_norm, '-d');
title('Error vs Iteration Number (n = 500)')
xlabel('Iteration Number (k)')
ylabel('Error')
legend('Jacobi', 'Gauss-Seidel', 'SOR (\omega=1.1)', 'SOR (\omega=0.9)')
```