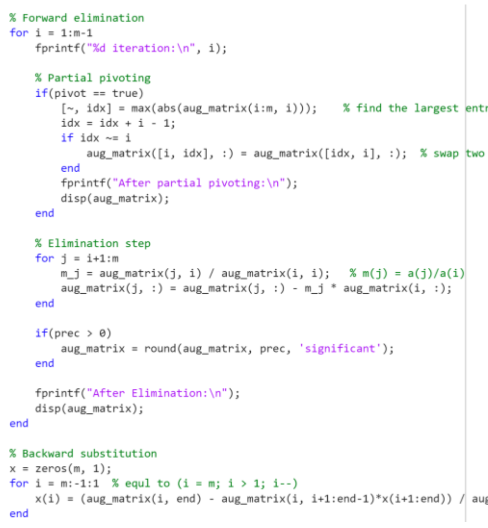
# Assignment 1

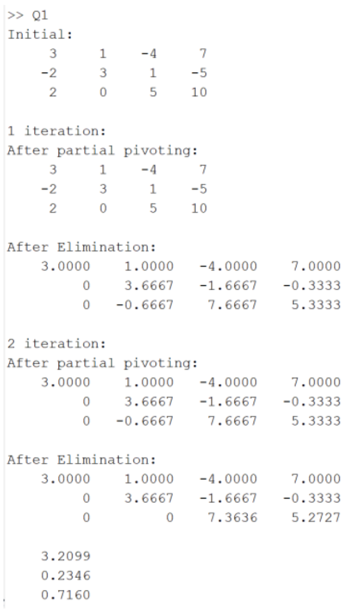
111550142 尤瑋辰

I use MATLAB to calculate the result. Here is the result and code:

1. It doesn't need to interchange row during Gaussian elimination.

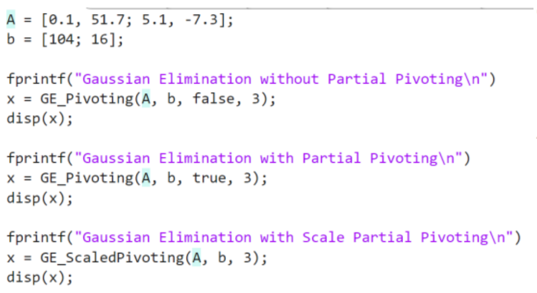


```
% Forward elimination
for i = 1:m-1
    fprintf("%d iteration:\n", i);

    % Partial pivoting
    if(pivot == true)
        [~, idx] = max(abs(aug_matrix(i:m, i)));    % find the largest entry below the diagonal
        idx = idx + i - 1;
        if idx ~= i
            aug_matrix([i, idx], :) = aug_matrix([idx, i], :);  % swap two rows
        end
        fprintf("After partial pivoting:\n");
        disp(aug_matrix);
    end

    % Elimination step
    for j = i+1:m
        m_j = aug_matrix(j, i) / aug_matrix(i, i);   % m(j) = a(j)/a(i)
        aug_matrix(j, :) = aug_matrix(j, :) - m_j * aug_matrix(i, :);
    end

    if(prec > 0)
        aug_matrix = round(aug_matrix, prec, 'significant');
    end

    fprintf("After Elimination:\n");
    disp(aug_matrix);
end

% Backward substitution
x = zeros(m, 1);
for i = m:-1:1  % equl to (i = m; i > 1; i--)
    x(i) = (aug_matrix(i, end) - aug_matrix(i, i+1:end-1)*x(i+1:end)) / aug_matrix(i, i);
end
```

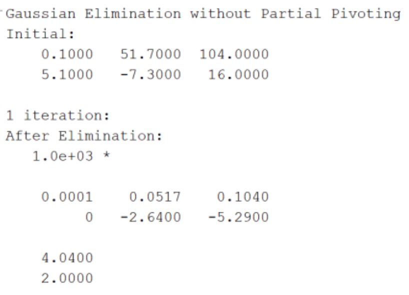


```
>> Q1
Initial:
     3     1    -4     7
    -2     3     1    -5
     2     0     5    10

1 iteration:
After partial pivoting:
     3     1    -4     7
    -2     3     1    -5
     2     0     5    10

After Elimination:
    3.0000    1.0000   -4.0000    7.0000
         0    3.6667   -1.6667   -0.3333
         0   -0.6667    7.6667    5.3333

2 iteration:
After partial pivoting:
    3.0000    1.0000   -4.0000    7.0000
         0    3.6667   -1.6667   -0.3333
         0   -0.6667    7.6667    5.3333

After Elimination:
    3.0000    1.0000   -4.0000    7.0000
         0    3.6667   -1.6667   -0.3333
         0         0    7.3636    5.2727

    3.2099
    0.2346
    0.7160
```

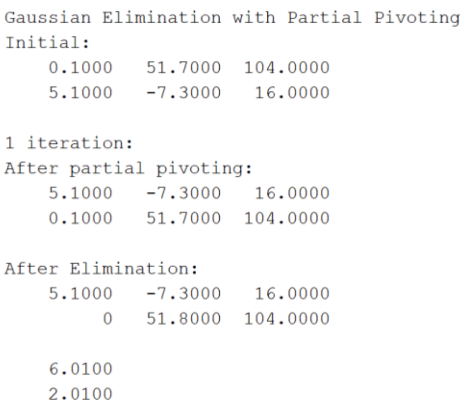
2. The function is identical to that of question 1. The part (c) is the same as part (b) because the outcomes of both partial pivoting and scaled partial pivoting are the same in this case.



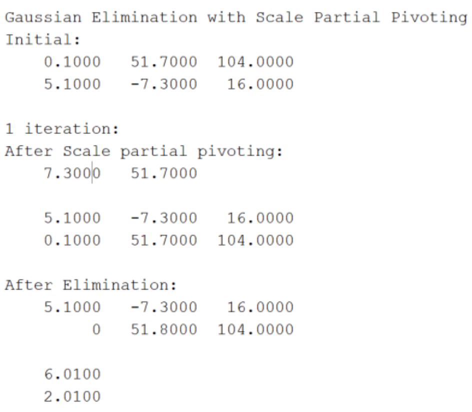
```
A = [0.1, 51.7; 5.1, -7.3];
b = [104; 16];

fprintf("Gaussian Elimination without Partial Pivoting\n")
x = GE_Pivoting(A, b, false, 3);
disp(x);

fprintf("Gaussian Elimination with Partial Pivoting\n")
x = GE_Pivoting(A, b, true, 3);
disp(x);

fprintf("Gaussian Elimination with Scale Partial Pivoting\n")
x = GE_ScaledPivoting(A, b, 3);
disp(x);
```



```
Gaussian Elimination without Partial Pivoting
Initial:
    0.1000   51.7000  104.0000
    5.1000   -7.3000   16.0000

1 iteration:
After Elimination:
   1.0e+03 *

    0.0001    0.0517    0.1040
         0   -2.6400   -5.2900

    4.0400
    2.0000
```

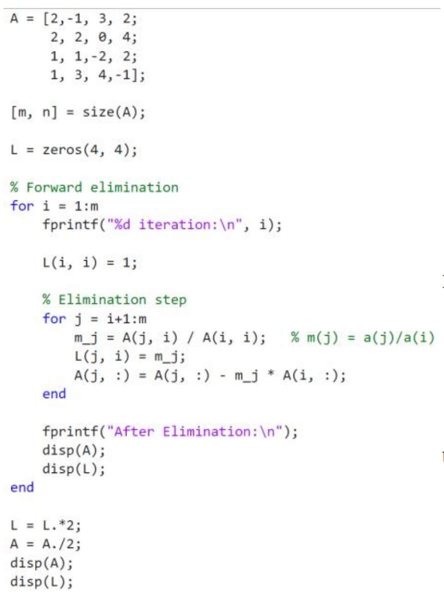


```
Gaussian Elimination with Partial Pivoting
Initial:
    0.1000   51.7000  104.0000
    5.1000   -7.3000   16.0000

1 iteration:
After partial pivoting:
    5.1000   -7.3000   16.0000
    0.1000   51.7000  104.0000

After Elimination:
    5.1000   -7.3000   16.0000
         0   51.8000  104.0000

    6.0100
    2.0100
```

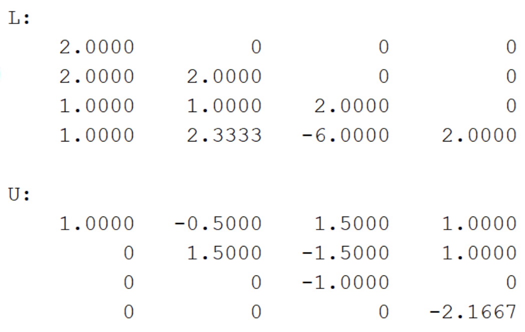


```
Gaussian Elimination with Scale Partial Pivoting
Initial:
    0.1000   51.7000  104.0000
    5.1000   -7.3000   16.0000

1 iteration:
After Scale partial pivoting:
    7.3000   51.7000

    5.1000   -7.3000   16.0000
    0.1000   51.7000  104.0000

After Elimination:
    5.1000   -7.3000   16.0000
         0   51.8000  104.0000

    6.0100
    2.0100
```

3.

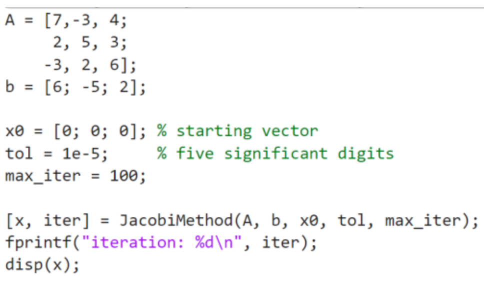


```
A = [2,-1, 3, 2;
     2, 2, 0, 4;
     1, 1,-2, 2;
     1, 3, 4,-1];

[m, n] = size(A);

L = zeros(4, 4);

% Forward elimination
for i = 1:m
    fprintf("%d iteration:\n", i);

    L(i, i) = 1;

    % Elimination step
    for j = i+1:m
        m_j = A(j, i) / A(i, i);   % m(j) = a(j)/a(i)
        L(j, i) = m_j;
        A(j, :) = A(j, :) - m_j * A(i, :);
    end

    fprintf("After Elimination:\n");
    disp(A);
    disp(L);
end

L = L.*2;
A = A./2;
disp(A);
disp(L);
```

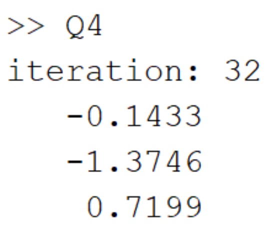


```
L:
     2.0000         0         0         0
     2.0000    2.0000         0         0
     1.0000    1.0000    2.0000         0
     1.0000    2.3333   -6.0000    2.0000

U:
     1.0000   -0.5000    1.5000    1.0000
          0    1.5000   -1.5000    1.0000
          0         0   -1.0000         0
          0         0         0   -2.1667
```

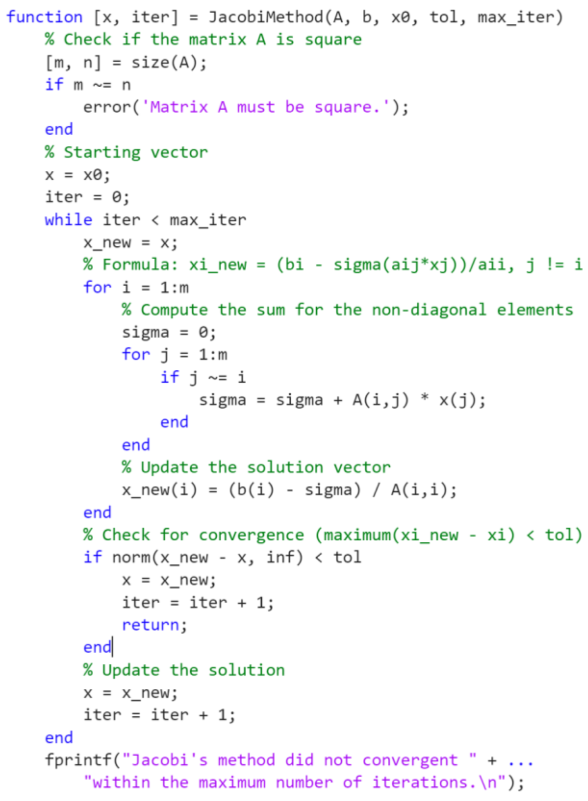
4.



```
A = [7,-3, 4;
     2, 5, 3;
    -3, 2, 6];
b = [6; -5; 2];

x0 = [0; 0; 0]; % starting vector
tol = 1e-5;     % five significant digits
max_iter = 100;

[x, iter] = JacobiMethod(A, b, x0, tol, max_iter);
fprintf("iteration: %d\n", iter);
disp(x);
```

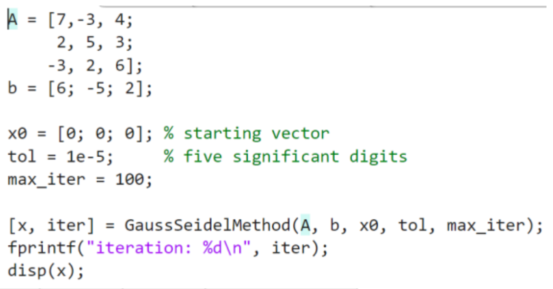


```
>> Q4
iteration: 32
   -0.1433
   -1.3746
    0.7199
```

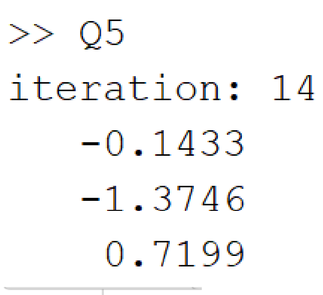


```
function [x, iter] = JacobiMethod(A, b, x0, tol, max_iter)
    % Check if the matrix A is square
    [m, n] = size(A);
    if m ~= n
        error('Matrix A must be square.');
    end
    % Starting vector
    x = x0;
    iter = 0;
    while iter < max_iter
        x_new = x;
        % Formula: xi_new = (bi - sigma(aij*xj))/aii, j != i
        for i = 1:m
            % Compute the sum for the non-diagonal elements
            sigma = 0;
            for j = 1:m
                if j ~= i
                    sigma = sigma + A(i,j) * x(j);
                end
            end
            % Update the solution vector
            x_new(i) = (b(i) - sigma) / A(i,i);
        end
        % Check for convergence (maximum(xi_new - xi) < tol)
        if norm(x_new - x, inf) < tol
            x = x_new;
            iter = iter + 1;
            return;
        end
        % Update the solution
        x = x_new;
        iter = iter + 1;
    end
    fprintf("Jacobi's method did not convergent " + ...
        "within the maximum number of iterations.\n");
```

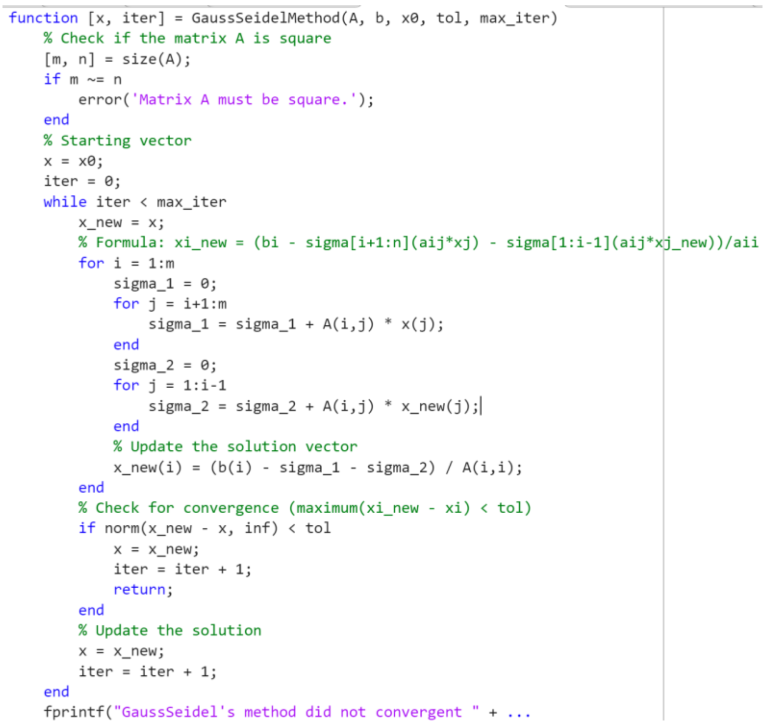
5. The Gauss-Seidel method requires approximately half the number of iterations compared to the Jacobi method.



```
A = [7,-3, 4;
     2, 5, 3;
    -3, 2, 6];
b = [6; -5; 2];

x0 = [0; 0; 0]; % starting vector
tol = 1e-5;     % five significant digits
max_iter = 100;

[x, iter] = GaussSeidelMethod(A, b, x0, tol, max_iter);
fprintf("iteration: %d\n", iter);
disp(x);
```

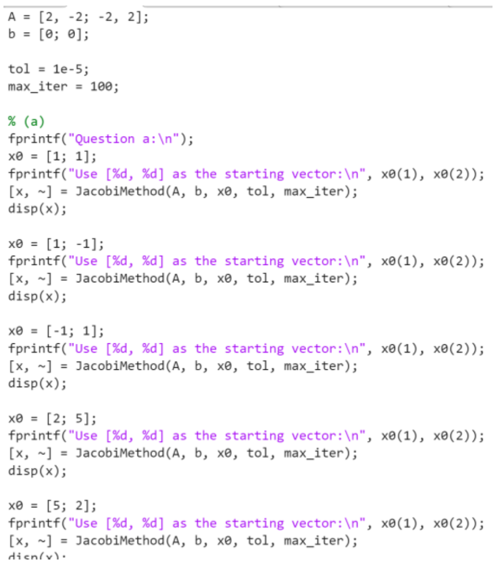


```
>> Q5
iteration: 14
   -0.1433
   -1.3746
    0.7199
```

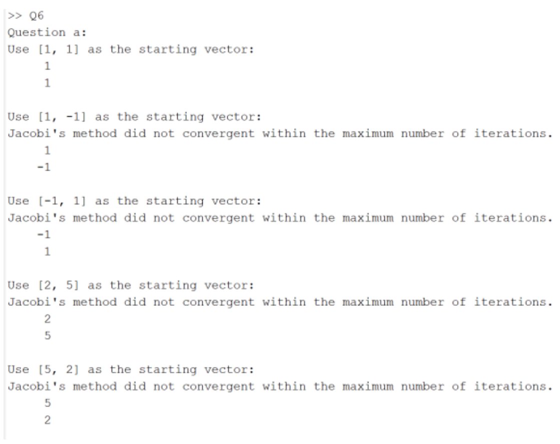


```
function [x, iter] = GaussSeidelMethod(A, b, x0, tol, max_iter)
    % Check if the matrix A is square
    [m, n] = size(A);
    if m ~= n
        error('Matrix A must be square.');
    end
    % Starting vector
    x = x0;
    iter = 0;
    while iter < max_iter
        x_new = x;
        % Formula: xi_new = (bi - sigma[i+1:n](aij*xj) - sigma[1:i-1](aij*xj_new))/aii
        for i = 1:m
            sigma_1 = 0;
            for j = i+1:m
                sigma_1 = sigma_1 + A(i,j) * x(j);
            end
            sigma_2 = 0;
            for j = 1:i-1
                sigma_2 = sigma_2 + A(i,j) * x_new(j);
            end
            % Update the solution vector
            x_new(i) = (b(i) - sigma_1 - sigma_2) / A(i,i);
        end
        % Check for convergence (maximum(xi_new - xi) < tol)
        if norm(x_new - x, inf) < tol
            x = x_new;
            iter = iter + 1;
            return;
        end
        % Update the solution
        x = x_new;
        iter = iter + 1;
    end
    fprintf("GaussSeidel's method did not convergent " + ...
```

6. (a) Assuming the vector x is [a, b], during the Jacobi method iteration, it will cycle between [b, a] and [a, b]. It cannot convergent.

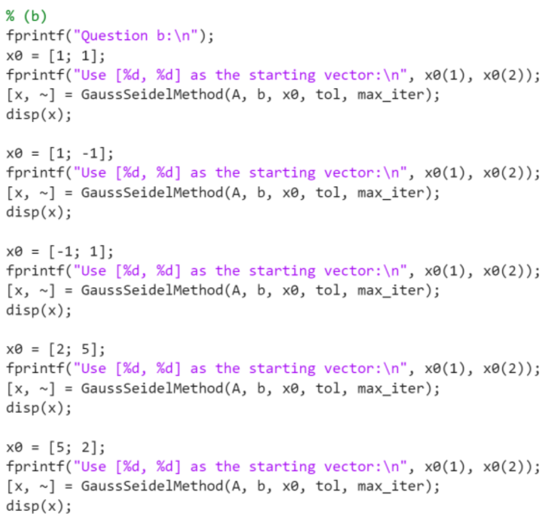


```
A = [2, -2; -2, 2];
b = [0; 0];

tol = 1e-5;
max_iter = 100;

% (a)
fprintf("Question a:\n");
x0 = [1; 1];
fprintf("Use [%d, %d] as the starting vector:\n", x0(1), x0(2));
[x, ~] = JacobiMethod(A, b, x0, tol, max_iter);
disp(x);

x0 = [1; -1];
fprintf("Use [%d, %d] as the starting vector:\n", x0(1), x0(2));
[x, ~] = JacobiMethod(A, b, x0, tol, max_iter);
disp(x);

x0 = [-1; 1];
fprintf("Use [%d, %d] as the starting vector:\n", x0(1), x0(2));
[x, ~] = JacobiMethod(A, b, x0, tol, max_iter);
disp(x);

x0 = [2; 5];
fprintf("Use [%d, %d] as the starting vector:\n", x0(1), x0(2));
[x, ~] = JacobiMethod(A, b, x0, tol, max_iter);
disp(x);

x0 = [5; 2];
fprintf("Use [%d, %d] as the starting vector:\n", x0(1), x0(2));
[x, ~] = JacobiMethod(A, b, x0, tol, max_iter);
disp(x);
```

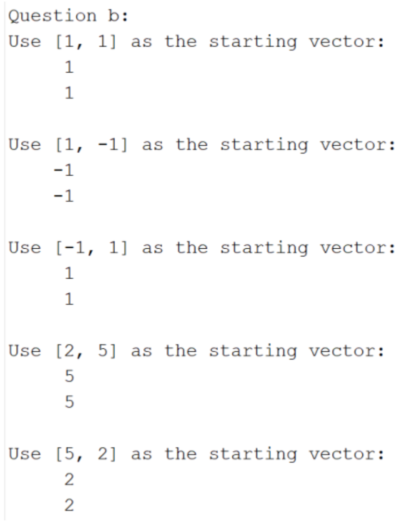


```
>> Q6
Question a:
Use [1, 1] as the starting vector:
     1
     1

Use [1, -1] as the starting vector:
Jacobi's method did not convergent within the maximum number of iterations.
     1
    -1

Use [-1, 1] as the starting vector:
Jacobi's method did not convergent within the maximum number of iterations.
    -1
     1

Use [2, 5] as the starting vector:
Jacobi's method did not convergent within the maximum number of iterations.
     2
     5

Use [5, 2] as the starting vector:
Jacobi's method did not convergent within the maximum number of iterations.
     5
     2
```

(b) Assuming the vector x is [a, b], in the Gauss-Sedial method, it will converge immediately, and the result will be [b, b].

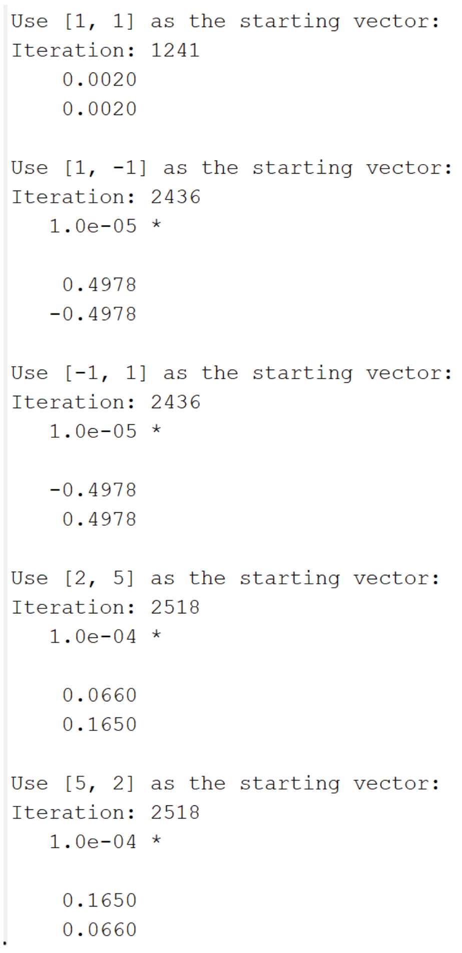


```
% (b)
fprintf("Question b:\n");
x0 = [1; 1];
fprintf("Use [%d, %d] as the starting vector:\n", x0(1), x0(2));
[x, ~] = GaussSeidelMethod(A, b, x0, tol, max_iter);
disp(x);

x0 = [1; -1];
fprintf("Use [%d, %d] as the starting vector:\n", x0(1), x0(2));
[x, ~] = GaussSeidelMethod(A, b, x0, tol, max_iter);
disp(x);

x0 = [-1; 1];
fprintf("Use [%d, %d] as the starting vector:\n", x0(1), x0(2));
[x, ~] = GaussSeidelMethod(A, b, x0, tol, max_iter);
disp(x);

x0 = [2; 5];
fprintf("Use [%d, %d] as the starting vector:\n", x0(1), x0(2));
[x, ~] = GaussSeidelMethod(A, b, x0, tol, max_iter);
disp(x);

x0 = [5; 2];
fprintf("Use [%d, %d] as the starting vector:\n", x0(1), x0(2));
[x, ~] = GaussSeidelMethod(A, b, x0, tol, max_iter);
disp(x);
```

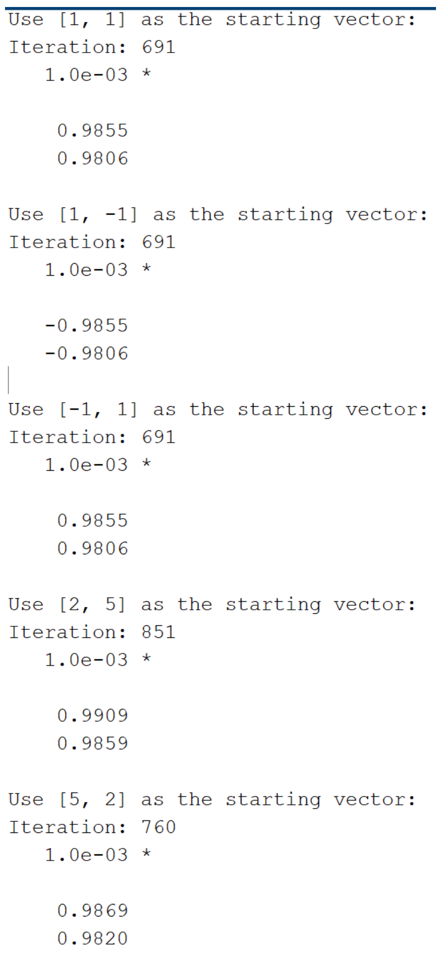


```
Question b:
Use [1, 1] as the starting vector:
     1
     1

Use [1, -1] as the starting vector:
    -1
    -1

Use [-1, 1] as the starting vector:
     1
     1

Use [2, 5] as the starting vector:
     5
     5

Use [5, 2] as the starting vector:
     2
     2
```

(c) The results in the Jacobi method are significantly influenced by the choice of the initial vector, whereas in the Gauss-Seidel method, the results tend to be approximately the same regardless of the initial vector.



```
Use [1, 1] as the starting vector:
Iteration: 1241
    0.0020
    0.0020

Use [1, -1] as the starting vector:
Iteration: 2436
   1.0e-05 *

    0.4978
   -0.4978

Use [-1, 1] as the starting vector:
Iteration: 2436
   1.0e-05 *

   -0.4978
    0.4978

Use [2, 5] as the starting vector:
Iteration: 2518
   1.0e-04 *

    0.0660
    0.1650

Use [5, 2] as the starting vector:
Iteration: 2518
   1.0e-04 *

    0.1650
    0.0660
```



```
Use [1, 1] as the starting vector:
Iteration: 691
   1.0e-03 *

    0.9855
    0.9806

Use [1, -1] as the starting vector:
Iteration: 691
   1.0e-03 *

   -0.9855
   -0.9806

Use [-1, 1] as the starting vector:
Iteration: 691
   1.0e-03 *

    0.9855
    0.9806

Use [2, 5] as the starting vector:
Iteration: 851
   1.0e-03 *

    0.9909
    0.9859

Use [5, 2] as the starting vector:
Iteration: 760
   1.0e-03 *

    0.9869
    0.9820
```