111550113 謝詠晴

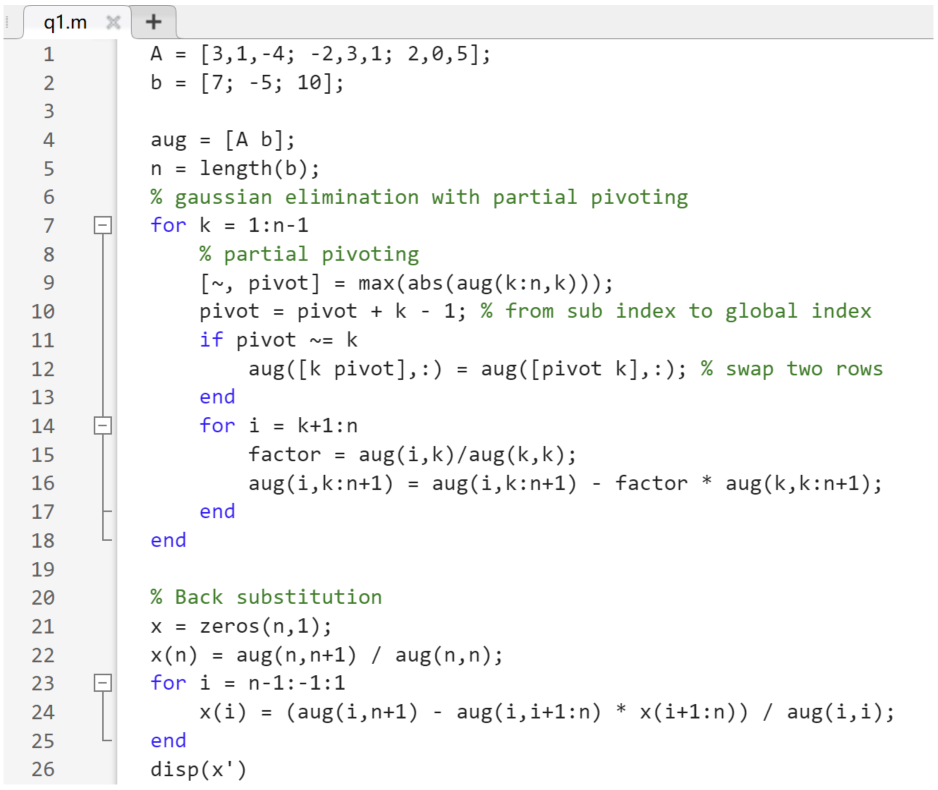
Numerical Methods_ Assignment 2

1. Use Gaussian elimination with partial pivoting to solve the equation.

   ANS:

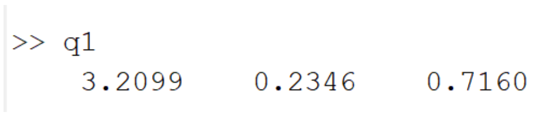
   x = 3.2099, y = 0.2346, z = 0.7160. There's no row interchange needed.

   - The implementation is written in "q1.m"
   - It iterates through each column and performs partial pivoting to find the maximum absolute value in that column. Then swap the rows if necessary.
   - Next, it iterates through the rows below the current row and eliminates the elements below the pivot in the current column.
   - Finally, use back substitution to find the solution vector x.



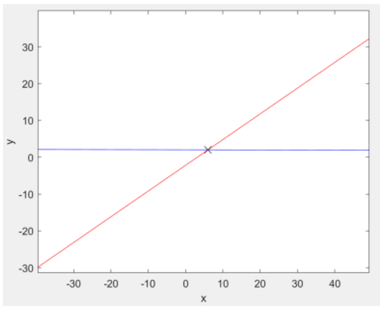
```matlab
A = [3,1,-4; -2,3,1; 2,0,5];
b = [7; -5; 10];

aug = [A b];
n = length(b);
% gaussian elimination with partial pivoting
for k = 1:n-1
    % partial pivoting
    [~, pivot] = max(abs(aug(k:n,k)));
    pivot = pivot + k - 1; % from sub index to global index
    if pivot ~= k
        aug([k pivot],:) = aug([pivot k],:); % swap two rows
    end
    for i = k+1:n
        factor = aug(i,k)/aug(k,k);
        aug(i,k:n+1) = aug(i,k:n+1) - factor * aug(k,k:n+1);
    end
end

% Back substitution
x = zeros(n,1);
x(n) = aug(n,n+1) / aug(n,n);
for i = n-1:-1:1
    x(i) = (aug(i,n+1) - aug(i,i+1:n) * x(i+1:n)) / aug(i,i);
end
disp(x')
```

Result:



```
>> q1
    3.2099    0.2346    0.7160
```

2. Solve a system of two equations and the intersections should be at (6,2).
   - First, I graph the two lines to see where's the intersection.
   - The implementation of graphing is written in "q2_graph.m".



(a) Using three significant digits of precision and no row interchanges. [1.99,10.0]

(a)
$$\begin{bmatrix} 0.1 & 51.7 & 104 \\ 5.1 & -7.3 & 16 \end{bmatrix}$$

$$= \begin{bmatrix} 0.1 & 51.7 & 104 \\ 0 & -2650 & -5280 \end{bmatrix}$$

$$x_2 = \frac{-5280}{-2650} = 1.99$$

$$x_1 = \frac{104 - 51.7 \times 1.99}{0.1} = \frac{104 - 103}{0.1} = 10.0$$

(b) Using partial pivoting. [2.01,6.02]

(b)
$$\begin{bmatrix} 5.1 & -7.3 & 16 \\ 0.1 & 51.7 & 104 \end{bmatrix} \big) -0.0196$$

$$= \begin{bmatrix} 5.1 & -7.3 & 16 \\ 0 & 51.8 & 104 \end{bmatrix}$$

$$x_2 = \frac{104}{51.8} = 2.01$$

$$x_1 = \frac{16 - (-7.3) \times 2.01}{5.1} = \frac{16 + 14.7}{5.1} = 6.02$$

(c) Using scaled partial pivoting. The result doesn't exactly match part(a) or (b) but is much closer to part(b). [2.00,5.99]

(c)
$$\begin{bmatrix} 0.1 & 51.7 & 104 \\ 5.1 & -7.3 & 16 \end{bmatrix} \Rightarrow \begin{bmatrix} 51.7 \\ 7.3 \end{bmatrix}$$

$$= \begin{bmatrix} 0.00193 & 1 & 2.01 \\ 0.699 & -1 & 2.19 \end{bmatrix} \Rightarrow \begin{bmatrix} 0.699 & -1 & 2.19 \\ 0.00193 & 1 & 2.01 \end{bmatrix}$$

$$\begin{bmatrix} 0.699 & -1 & 2.19 \\ 0 & 1 & 2.00 \end{bmatrix} \big) 0.00276$$

$$x_2 = \frac{2}{1} = 2.00$$

$$x_1 = \frac{2.19 - (-1) \times 2.00}{0.699} = 5.99 \quad \#$$

3. Find the LU equivalent of matrix A that has 2's in each diagonal position of L rather than 1's.

3.

$$A = \begin{bmatrix} 2 & -1 & 3 & 2 \\ 2 & 2 & 0 & 4 \\ 1 & 1 & -2 & 2 \\ 1 & 3 & 4 & -1 \end{bmatrix}$$

$R_2 = R_2 - R_1$, $R_3 = R_3 - \frac{1}{2}R_1$, $R_4 = R_4 - \frac{1}{2}R_1$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & 1 & 0 \\ \frac{1}{2} & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 2 & -1 & 3 & 2 \\ 0 & 3 & -3 & 2 \\ 0 & \frac{3}{2} & \frac{-7}{2} & 1 \\ 0 & \frac{7}{2} & \frac{5}{2} & -2 \end{bmatrix}$$

$R_3 = R_3 - \frac{1}{2}R_2$, $R_4 = R_4 - \frac{7}{6}R_2$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 1 & 0 \\ \frac{1}{2} & \frac{7}{6} & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 2 & -1 & 3 & 2 \\ 0 & 3 & -3 & 2 \\ 0 & 0 & -2 & 0 \\ 0 & 0 & 6 & \frac{-13}{3} \end{bmatrix}$$

$R_4 = R_4 - (-3)R_3$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 1 & 0 \\ \frac{1}{2} & \frac{7}{6} & -3 & 1 \end{bmatrix} \quad U = \begin{bmatrix} 2 & -1 & 3 & 2 \\ 0 & 3 & -3 & 2 \\ 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & \frac{-13}{3} \end{bmatrix}$$

$$2 \times L = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 \\ 1 & 1 & 2 & 0 \\ 1 & \frac{7}{3} & -6 & 2 \end{bmatrix} \quad \frac{1}{2} \times U = \begin{bmatrix} 1 & \frac{-1}{2} & \frac{3}{2} & 1 \\ 0 & \frac{3}{2} & \frac{-3}{2} & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & \frac{-13}{6} \end{bmatrix}$$

$$A = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 \\ 1 & 1 & 2 & 0 \\ 1 & \frac{7}{3} & -6 & 2 \end{bmatrix} \begin{bmatrix} 1 & \frac{-1}{2} & \frac{3}{2} & 1 \\ 0 & \frac{3}{2} & \frac{-3}{2} & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & \frac{-13}{6} \end{bmatrix}$$ #

4. Solve the system with the Jacobi method and use [0, 0, 0] as the starting vector.

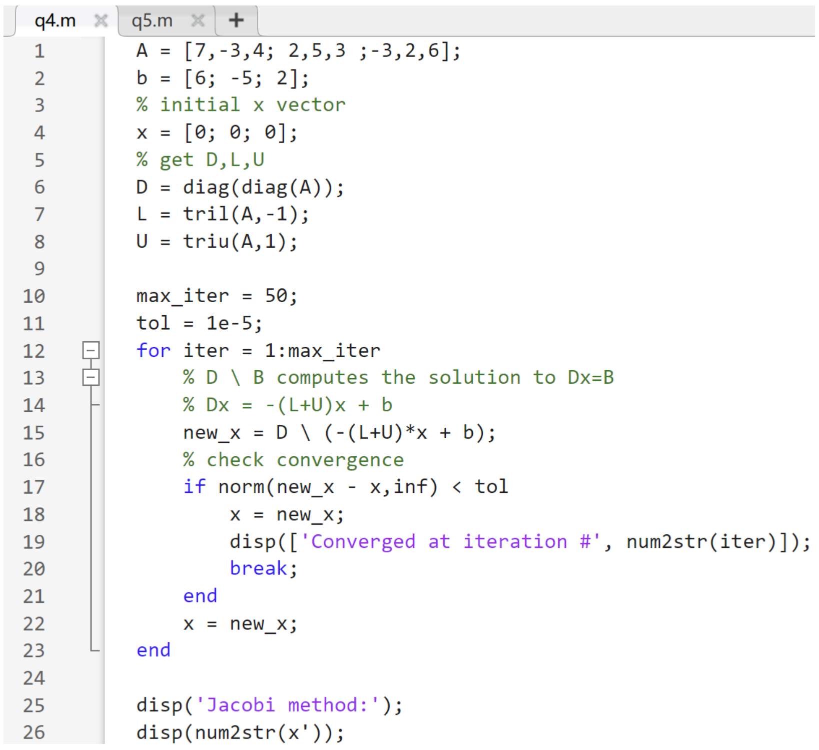
   (1) Exchange row 2 and row 3, since $5 \geq 2 + 3, \quad 6 \geq 3 + 2$

$$\begin{bmatrix} 7 & -3 & 4 \\ -3 & 2 & 6 \\ 2 & 5 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 7 & -3 & 4 \\ 2 & 5 & 3 \\ -3 & 2 & 6 \end{bmatrix}$$

   (2) The implementation is written in "q4.m"

   - I use 'diag()', 'tril()' and 'triu()' to get the decomposition of matrix A, which are diagonal (D), lower triangular (L), and upper triangular (U) matrices.

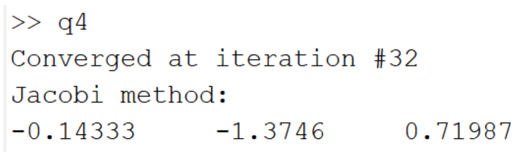
- In the iteration, compute the new estimate for x using the Jacobi iteration formula: ‘new_x = D \ (-(L+U)*x + b)’.
- Here, D \ B symbol means the solution to Dx = B.
- Then, check for convergence using the condition “norm(new_x - x, inf) < tol”.



```
A = [7,-3,4; 2,5,3 ;-3,2,6];
b = [6; -5; 2];
% initial x vector
x = [0; 0; 0];
% get D,L,U
D = diag(diag(A));
L = tril(A,-1);
U = triu(A,1);

max_iter = 50;
tol = 1e-5;
for iter = 1:max_iter
    % D \ B computes the solution to Dx=B
    % Dx = -(L+U)x + b
    new_x = D \ (-(L+U)*x + b);
    % check convergence
    if norm(new_x - x,inf) < tol
        x = new_x;
        disp(['Converged at iteration #', num2str(iter)]);
        break;
    end
    x = new_x;
end

disp('Jacobi method:');
disp(num2str(x'));
```

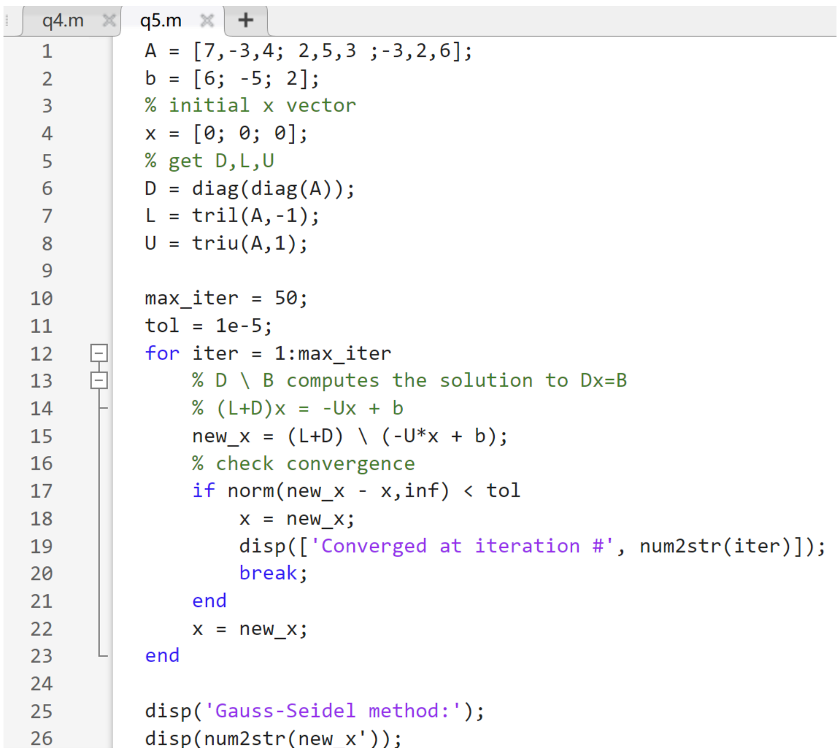
(3) Result:

- It takes <u>32 iterations</u> to get the solution accurate to five significant digits.
- And <u>the answer is [-0.14332, -1.3746, 0.71987]</u>.



```
>> q4
Converged at iteration #32
Jacobi method:
-0.14333      -1.3746      0.71987
```

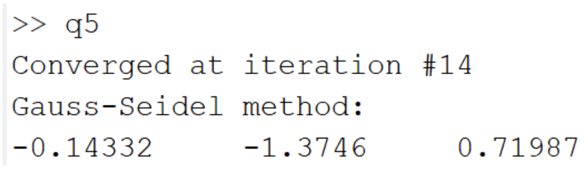
5. Repeat Problem 4 with the Gauss-Seidel method.
   - The implementation is written in “q5.m” and is quite similar to q4. The only difference between them is the formula to compute new_x.
   - Here, we use “new_x = (L+D) \ (-U*x + b)”.



```
A = [7,-3,4; 2,5,3 ;-3,2,6];
b = [6; -5; 2];
% initial x vector
x = [0; 0; 0];
% get D,L,U
D = diag(diag(A));
L = tril(A,-1);
U = triu(A,1);

max_iter = 50;
tol = 1e-5;
for iter = 1:max_iter
    % D \ B computes the solution to Dx=B
    % (L+D)x = -Ux + b
    new_x = (L+D) \ (-U*x + b);
    % check convergence
    if norm(new_x - x,inf) < tol
        x = new_x;
        disp(['Converged at iteration #', num2str(iter)]);
        break;
    end
    x = new_x;
end

disp('Gauss-Seidel method:');
disp(num2str(new_x'));
```

- Result:

  Only 14 iterations are required, which is less than Jacobi method.

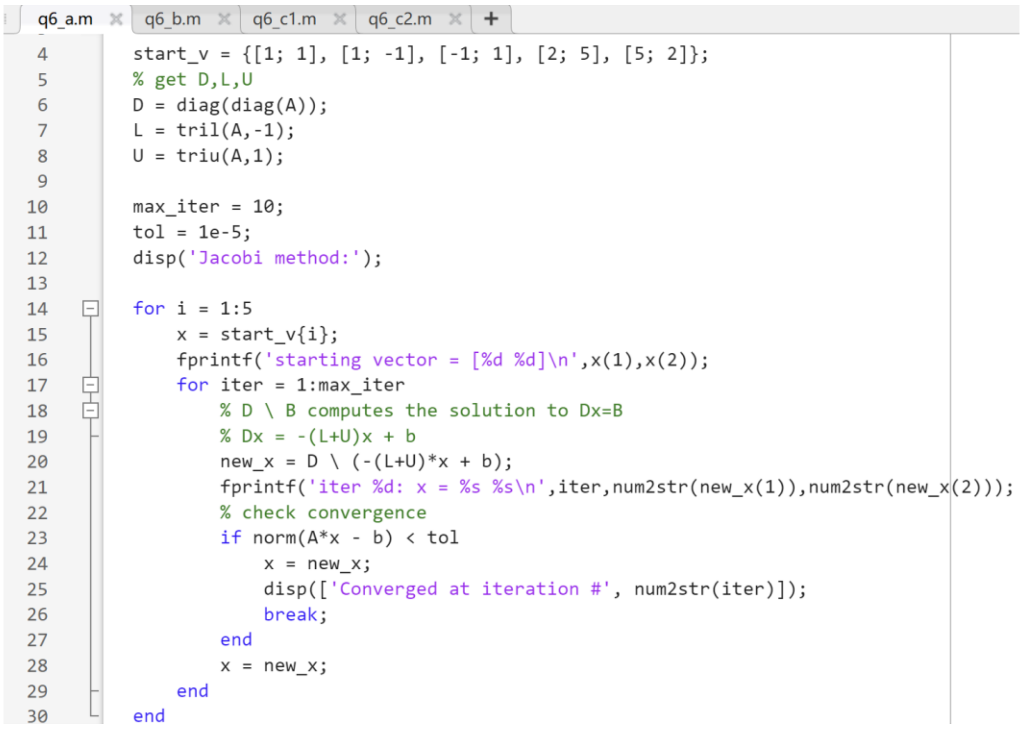


```
>> q5
Converged at iteration #14
Gauss-Seidel method:
-0.14332      -1.3746      0.71987
```

6. This 2 × 2 matrix is obviously singular and is almost diagonally dominant. If the right-hand-side vector is [0, 0], the equations are satisfied by any pair where x = y.

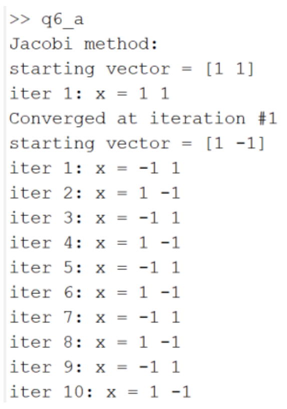
   (a) use Jacobi method with starting vectors: [1, 1],[1, -1], [-1, 1], [2, 5], [5, 2]

   - The implementation is written in "q6_a.m"
   - Using the code same as q4.m and I set 'max_iter' to 10, 'tol' to 1e-5.

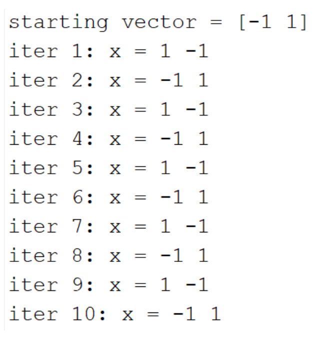


```
start_v = {[1; 1], [1; -1], [-1; 1], [2; 5], [5; 2]};
% get D,L,U
D = diag(diag(A));
L = tril(A,-1);
U = triu(A,1);

max_iter = 10;
tol = 1e-5;
disp('Jacobi method:');

for i = 1:5
    x = start_v{i};
    fprintf('starting vector = [%d %d]\n',x(1),x(2));
    for iter = 1:max_iter
        % D \ B computes the solution to Dx=B
        % Dx = -(L+U)x + b
        new_x = D \ (-(L+U)*x + b);
        fprintf('iter %d: x = %s %s\n',iter,num2str(new_x(1)),num2str(new_x(2)));
        % check convergence
        if norm(A*x - b) < tol
            x = new_x;
            disp(['Converged at iteration #', num2str(iter)]);
            break;
        end
        x = new_x;
    end
end
```

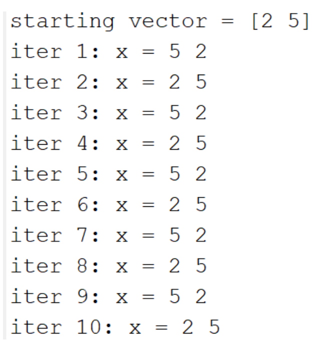
   - The result show that only when starting vector is [1,1]. The solution x vector is correct as [1,1].
   - The other starting vectors' result will infinitely iterate between themselves and the reversed. For example, the x vector of [2, 5] will oscillate between two different solutions [5, 2] and [2, 5].



```
>> q6_a
Jacobi method:
starting vector = [1 1]
iter 1: x = 1 1
Converged at iteration #1
starting vector = [1 -1]
iter 1: x = -1 1
iter 2: x = 1 -1
iter 3: x = -1 1
iter 4: x = 1 -1
iter 5: x = -1 1
iter 6: x = 1 -1
iter 7: x = -1 1
iter 8: x = 1 -1
iter 9: x = -1 1
iter 10: x = 1 -1
```



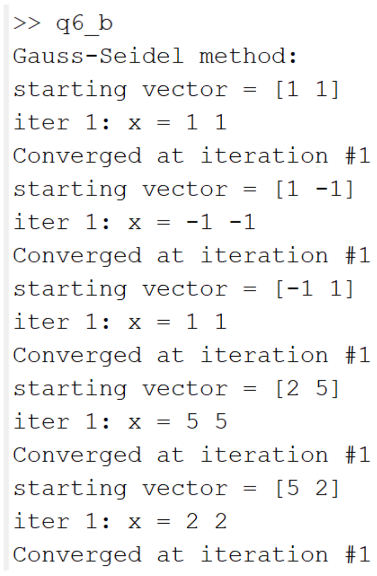
```
starting vector = [-1 1]
iter 1: x = 1 -1
iter 2: x = -1 1
iter 3: x = 1 -1
iter 4: x = -1 1
iter 5: x = 1 -1
iter 6: x = -1 1
iter 7: x = 1 -1
iter 8: x = -1 1
iter 9: x = 1 -1
iter 10: x = -1 1
```



```
starting vector = [2 5]
iter 1: x = 5 2
iter 2: x = 2 5
iter 3: x = 5 2
iter 4: x = 2 5
iter 5: x = 5 2
iter 6: x = 2 5
iter 7: x = 5 2
iter 8: x = 2 5
iter 9: x = 5 2
iter 10: x = 2 5
```

(b) Use Gauss-Seidel method with the same starting vectors:

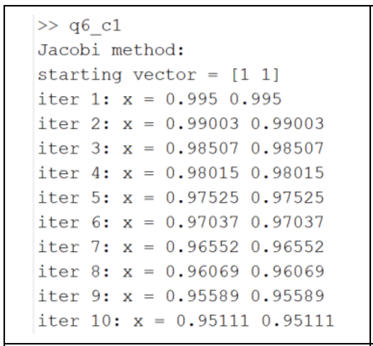
- The implementation is written in “q6_b.m”
- The result of using Gauss-Seidel method turn out to be correct.



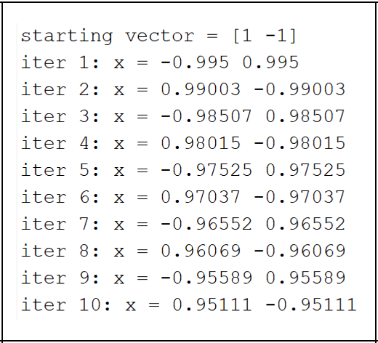
```
>> q6_b
Gauss-Seidel method:
starting vector = [1 1]
iter 1: x = 1 1
Converged at iteration #1
starting vector = [1 -1]
iter 1: x = -1 -1
Converged at iteration #1
starting vector = [-1 1]
iter 1: x = 1 1
Converged at iteration #1
starting vector = [2 5]
iter 1: x = 5 5
Converged at iteration #1
starting vector = [5 2]
iter 1: x = 2 2
Converged at iteration #1
```

(c) Change values -2 in the matrix to -1.99 and repeat parts (a) and (b).

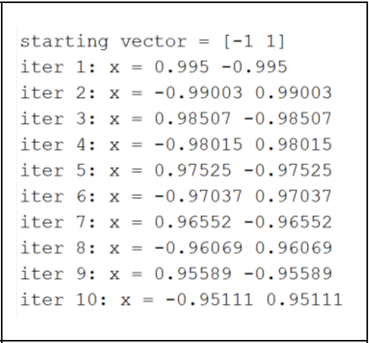
- Using Jacobi method, the implementation is written in “q6_c1.m” and I set ‘max_iter’ to 10.
- The result is still only correct when starting vector is [1, 1].
- The other starting vectors’ result is oscillated between two solutions, but the values are not the same in each iteration.



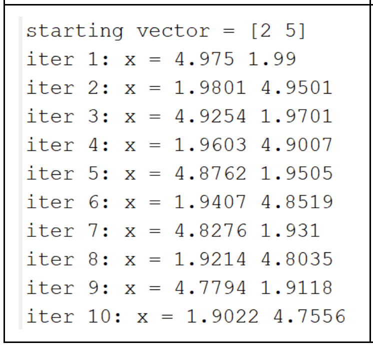
```
>> q6_c1
Jacobi method:
starting vector = [1 1]
iter 1: x = 0.995 0.995
iter 2: x = 0.99003 0.99003
iter 3: x = 0.98507 0.98507
iter 4: x = 0.98015 0.98015
iter 5: x = 0.97525 0.97525
iter 6: x = 0.97037 0.97037
iter 7: x = 0.96552 0.96552
iter 8: x = 0.96069 0.96069
iter 9: x = 0.95589 0.95589
iter 10: x = 0.95111 0.95111
```



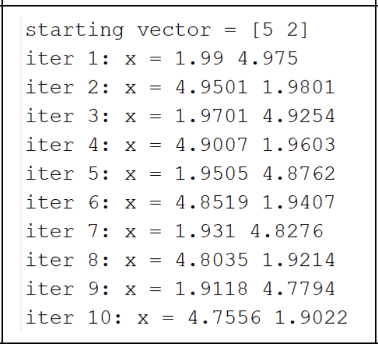
```
starting vector = [1 -1]
iter 1: x = -0.995 0.995
iter 2: x = 0.99003 -0.99003
iter 3: x = -0.98507 0.98507
iter 4: x = 0.98015 -0.98015
iter 5: x = -0.97525 0.97525
iter 6: x = 0.97037 -0.97037
iter 7: x = -0.96552 0.96552
iter 8: x = 0.96069 -0.96069
iter 9: x = -0.95589 0.95589
iter 10: x = 0.95111 -0.95111
```



```
starting vector = [-1 1]
iter 1: x = 0.995 -0.995
iter 2: x = -0.99003 0.99003
iter 3: x = 0.98507 -0.98507
iter 4: x = -0.98015 0.98015
iter 5: x = 0.97525 -0.97525
iter 6: x = -0.97037 0.97037
iter 7: x = 0.96552 -0.96552
iter 8: x = -0.96069 0.96069
iter 9: x = 0.95589 -0.95589
iter 10: x = -0.95111 0.95111
```

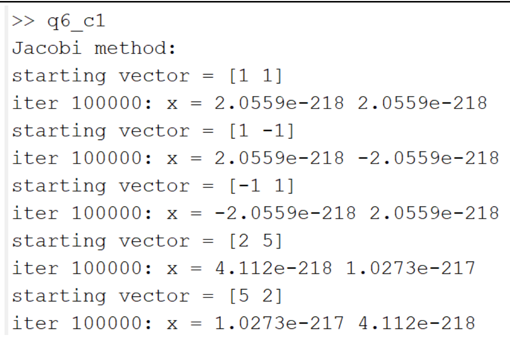
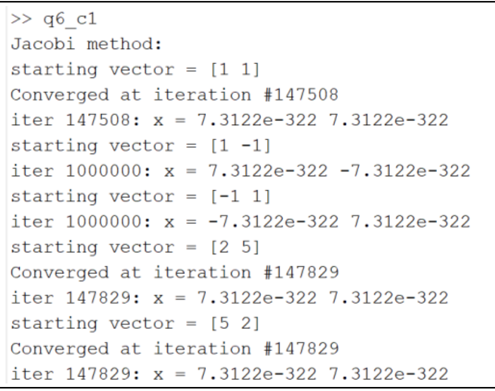


```
starting vector = [2 5]
iter 1: x = 4.975 1.99
iter 2: x = 1.9801 4.9501
iter 3: x = 4.9254 1.9701
iter 4: x = 1.9603 4.9007
iter 5: x = 4.8762 1.9505
iter 6: x = 1.9407 4.8519
iter 7: x = 4.8276 1.931
iter 8: x = 1.9214 4.8035
iter 9: x = 4.7794 1.9118
iter 10: x = 1.9022 4.7556
```



```
starting vector = [5 2]
iter 1: x = 1.99 4.975
iter 2: x = 4.9501 1.9801
iter 3: x = 1.9701 4.9254
iter 4: x = 4.9007 1.9603
iter 5: x = 1.9505 4.8762
iter 6: x = 4.8519 1.9407
iter 7: x = 1.931 4.8276
iter 8: x = 4.8035 1.9214
iter 9: x = 1.9118 4.7794
iter 10: x = 4.7556 1.9022
```

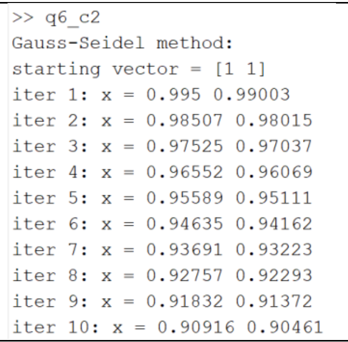
- If I set ‘max_iter’ to 100000 and 1000000:
- Starting vector [1,1] must have correct answer and starting vector [2,5], [5,2] will converge at around iteration #147829. While starting vector [1,-1] and [-1,1] won’t converge and it is probably because of the different signed value. (one positive and one negative)

max_iter = 100000	max_iter = 1000000
	

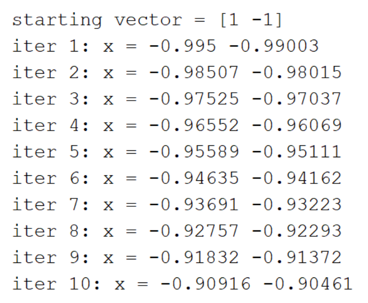
```
>> q6_c1
Jacobi method:
starting vector = [1 1]
iter 100000: x = 2.0559e-218 2.0559e-218
starting vector = [1 -1]
iter 100000: x = 2.0559e-218 -2.0559e-218
starting vector = [-1 1]
iter 100000: x = -2.0559e-218 2.0559e-218
starting vector = [2 5]
iter 100000: x = 4.112e-218 1.0273e-217
starting vector = [5 2]
iter 100000: x = 1.0273e-217 4.112e-218
```

```
>> q6_c1
Jacobi method:
starting vector = [1 1]
Converged at iteration #147508
iter 147508: x = 7.3122e-322 7.3122e-322
starting vector = [1 -1]
iter 1000000: x = 7.3122e-322 -7.3122e-322
starting vector = [-1 1]
iter 1000000: x = -7.3122e-322 7.3122e-322
starting vector = [2 5]
Converged at iteration #147829
iter 147829: x = 7.3122e-322 7.3122e-322
starting vector = [5 2]
Converged at iteration #147829
iter 147829: x = 7.3122e-322 7.3122e-322
```

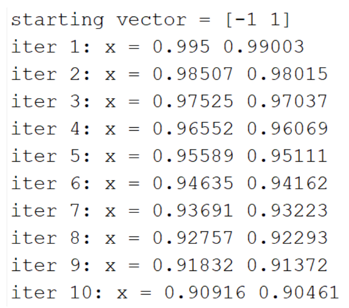
- Using <u>Gauss-Seidel method</u>, the implementation is written in “q6_c2.m”
- In each starting vector result, x1 and x2 are not exactly the same but quite similar. And the value of then are more and more close during each iteration.



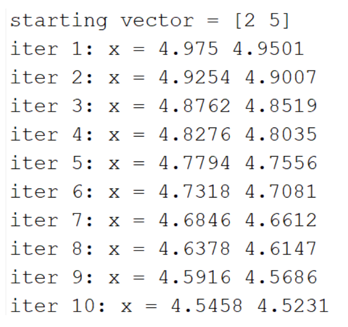
```
>> q6_c2
Gauss-Seidel method:
starting vector = [1 1]
iter 1: x = 0.995 0.99003
iter 2: x = 0.98507 0.98015
iter 3: x = 0.97525 0.97037
iter 4: x = 0.96552 0.96069
iter 5: x = 0.95589 0.95111
iter 6: x = 0.94635 0.94162
iter 7: x = 0.93691 0.93223
iter 8: x = 0.92757 0.92293
iter 9: x = 0.91832 0.91372
iter 10: x = 0.90916 0.90461
```



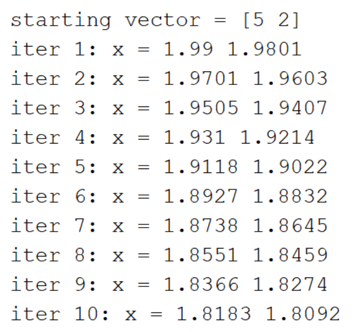
```
starting vector = [1 -1]
iter 1: x = -0.995 -0.99003
iter 2: x = -0.98507 -0.98015
iter 3: x = -0.97525 -0.97037
iter 4: x = -0.96552 -0.96069
iter 5: x = -0.95589 -0.95111
iter 6: x = -0.94635 -0.94162
iter 7: x = -0.93691 -0.93223
iter 8: x = -0.92757 -0.92293
iter 9: x = -0.91832 -0.91372
iter 10: x = -0.90916 -0.90461
```



```
starting vector = [-1 1]
iter 1: x = 0.995 0.99003
iter 2: x = 0.98507 0.98015
iter 3: x = 0.97525 0.97037
iter 4: x = 0.96552 0.96069
iter 5: x = 0.95589 0.95111
iter 6: x = 0.94635 0.94162
iter 7: x = 0.93691 0.93223
iter 8: x = 0.92757 0.92293
iter 9: x = 0.91832 0.91372
iter 10: x = 0.90916 0.90461
```

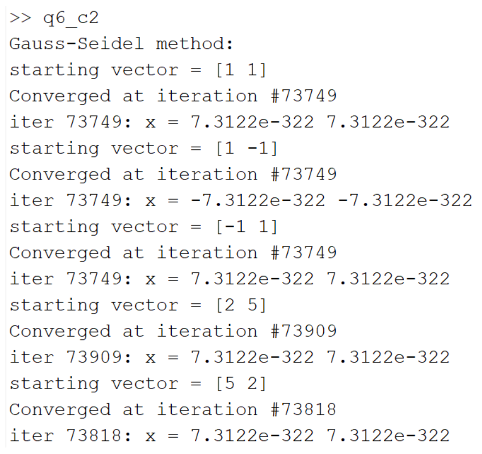


```
starting vector = [2 5]
iter 1: x = 4.975 4.9501
iter 2: x = 4.9254 4.9007
iter 3: x = 4.8762 4.8519
iter 4: x = 4.8276 4.8035
iter 5: x = 4.7794 4.7556
iter 6: x = 4.7318 4.7081
iter 7: x = 4.6846 4.6612
iter 8: x = 4.6378 4.6147
iter 9: x = 4.5916 4.5686
iter 10: x = 4.5458 4.5231
```



```
starting vector = [5 2]
iter 1: x = 1.99 1.9801
iter 2: x = 1.9701 1.9603
iter 3: x = 1.9505 1.9407
iter 4: x = 1.931 1.9214
iter 5: x = 1.9118 1.9022
iter 6: x = 1.8927 1.8832
iter 7: x = 1.8738 1.8645
iter 8: x = 1.8551 1.8459
iter 9: x = 1.8366 1.8274
iter 10: x = 1.8183 1.8092
```

- If I set ‘max_iter’ to 100000:
- All the starting vector will converge. And except [1,1], most of them converge around iteration #73700 to #73900.



```
>> q6_c2
Gauss-Seidel method:
starting vector = [1 1]
Converged at iteration #73749
iter 73749: x = 7.3122e-322 7.3122e-322
starting vector = [1 -1]
Converged at iteration #73749
iter 73749: x = -7.3122e-322 -7.3122e-322
starting vector = [-1 1]
Converged at iteration #73749
iter 73749: x = 7.3122e-322 7.3122e-322
starting vector = [2 5]
Converged at iteration #73909
iter 73909: x = 7.3122e-322 7.3122e-322
starting vector = [5 2]
Converged at iteration #73818
iter 73818: x = 7.3122e-322 7.3122e-322
```