ENUME Project A

No. 32

Fengxi Zhao

317342

November, 2023

Task 1:

1. Write a program finding *macheps* in the MATLAB environment on a lab computer or your computer.

Machine epsilon:

According to IEEE 754, 64-bits floating number mantissa has 52 digits, which means the absolute values of numbers smaller than 2^-52 will not be able to detect.

Algorithm description:

I try to design a loop which decreases half of the initial value each time until mechesp is neglectable. In this program 1 + mecheps is no longer larger than 1. The results match eps read from the MATLAB, and the loop counter also matches -52 exponent.

Result:

```
Project_A_Task_1
eps =
     2.220446049250313e-16

cou =
    52

mecheps =
     2.220446049250313e-16
```

Task 2:

2. Write a general program solving a system of n linear equations $\mathbf{Ax} = \mathbf{b}$ using *the indicated method.* Using only elementary mathematical operations on numbers and vectors is allowed (command "A\b" cannot be used, except only for checking the results). Apply the program to solve the system of linear equations for given matrix $\mathbf{A}$ and vector $\mathbf{b}$, for increasing numbers of equations $n = 10,20,40,80,160,\ldots$ until the solution time becomes prohibitive (or the method fails), for:

a) $a_{ij} = \begin{cases} 6 & for\ i = j \\ -1 & for\ i = j-1\ or\ i = j+1 \\ 0 & other\ cases \end{cases}$, $\quad b_i = -2 + 0.3\,i, \quad i,j = 1,\ldots,n;$

b) $a_{ij} = 1/[4(i + j + 1)], \quad b_i = 7/(6\,i),\ i$ – even; $b_i = 0,\ i$ – odd, $\ i,j = 1,\ldots,n.$

For each case a) and b) calculate the solution error defined as the Euclidean norm of the vector of residuum $\mathbf{r} = \mathbf{Ax} - \mathbf{b}$, where $\mathbf{x}$ is the solution, and plot this error versus n. For $n = 10$ print the solutions and the solutions' errors, make the residual correction and check if it improves the solutions.

The indicated method: Gaussian elimination with partial pivoting.

Gaussian elimination with partial pivoting algorithm:

Step1: Implement the matrices A and B.

Step 2: Partial pivot the matrices. For the matrix AB, for each column, find the max absolute value below the current row, then exchange them.

Step 3: After each exchange on step 2, elimination pass on all rows i > k. Subtract (a_{ik}/a_{kk})(row k) for all other rows > k. Then we can get an upper triangular matrix.

Step 4: Backward substitution. Find the first answer in the bottom of the matrix, store it in the answer vector. Substitute the answer to the row next to the bottom one and find the next answer. Repeat until find them all.

Result:

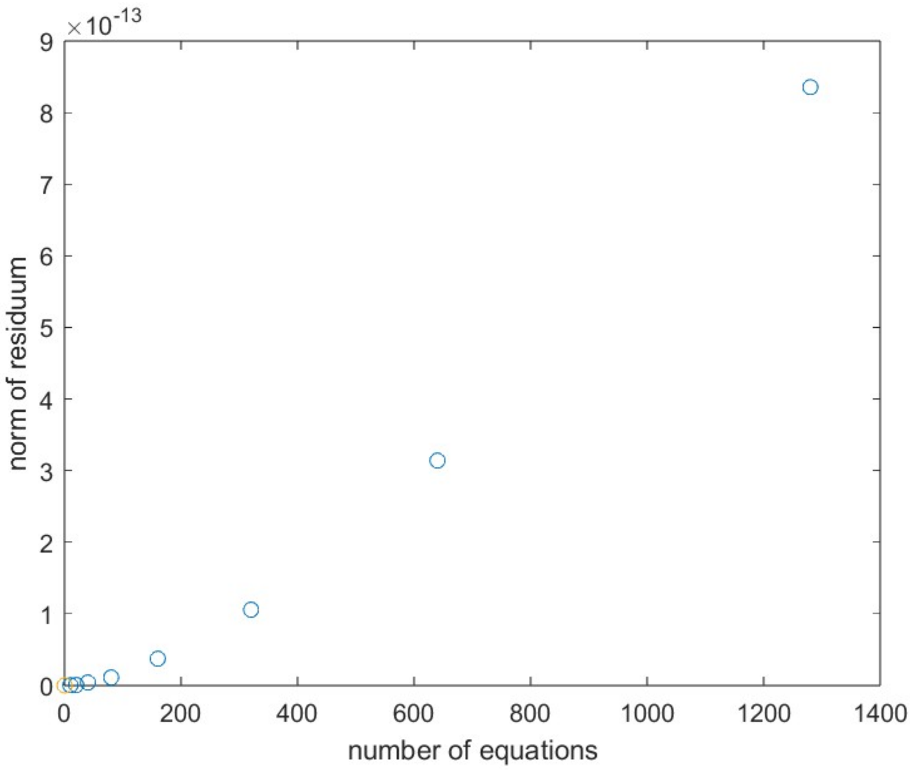
```
>> Project_A_Task_2
Task2a:
Matrix A:
     6    -1     0     0     0     0     0     0     0     0
    -1     6    -1     0     0     0     0     0     0     0
     0    -1     6    -1     0     0     0     0     0     0
     0     0    -1     6    -1     0     0     0     0     0
     0     0     0    -1     6    -1     0     0     0     0
     0     0     0     0    -1     6    -1     0     0     0
     0     0     0     0     0    -1     6    -1     0     0
     0     0     0     0     0     0    -1     6    -1     0
     0     0     0     0     0     0     0    -1     6    -1
     0     0     0     0     0     0     0     0    -1     6

Vector B:
   -1.7000
   -1.4000
   -1.1000
   -0.8000
   -0.5000
   -0.2000
    0.1000
    0.4000
    0.7000
    1.0000

Result by me:
   -0.3392
   -0.3353
   -0.2725
   -0.1996
   -0.1249
   -0.0500
    0.0247
    0.0984
    0.1654
    0.1942
```

```
Result by Matlab:
   -0.3392
   -0.3353
   -0.2725
   -0.1996
   -0.1249
   -0.0500
    0.0247
    0.0984
    0.1654
    0.1942

Norm of residuum:
   2.5448e-16

Norm of residuum after residual correction:
   2.4912e-16

Condition number:
    1.9404

Task2b:
Matrix A:
    0.0833    0.0625    0.0500    0.0417    0.0357    0.0312    0.0278    0.0250    0.0227    0.0208
    0.0625    0.0500    0.0417    0.0357    0.0312    0.0278    0.0250    0.0227    0.0208    0.0192
    0.0500    0.0417    0.0357    0.0312    0.0278    0.0250    0.0227    0.0208    0.0192    0.0179
    0.0417    0.0357    0.0312    0.0278    0.0250    0.0227    0.0208    0.0192    0.0179    0.0167
    0.0357    0.0312    0.0278    0.0250    0.0227    0.0208    0.0192    0.0179    0.0167    0.0156
    0.0312    0.0278    0.0250    0.0227    0.0208    0.0192    0.0179    0.0167    0.0156    0.0147
    0.0278    0.0250    0.0227    0.0208    0.0192    0.0179    0.0167    0.0156    0.0147    0.0139
    0.0250    0.0227    0.0208    0.0192    0.0179    0.0167    0.0156    0.0147    0.0139    0.0132
    0.0227    0.0208    0.0192    0.0179    0.0167    0.0156    0.0147    0.0139    0.0132    0.0125
    0.0208    0.0192    0.0179    0.0167    0.0156    0.0147    0.0139    0.0132    0.0125    0.0119
```

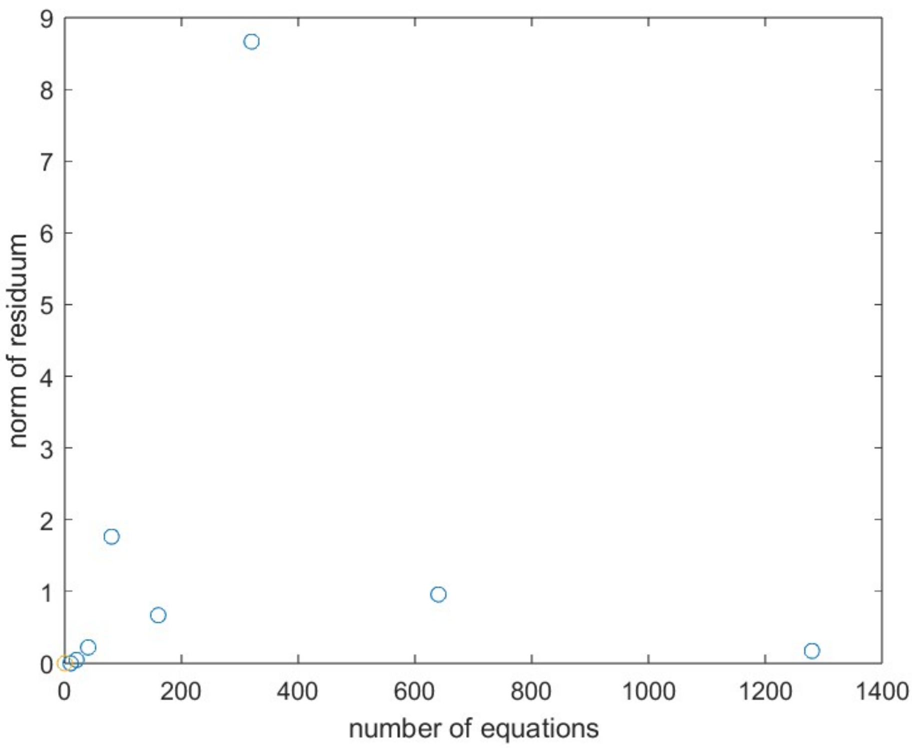
```
Vector B:
         0
    0.5833
         0
    0.2917
         0
    0.1944
         0
    0.1458
         0
    0.1167

Result by me:
   1.0e+14 *

  -0.0000
   0.0011
  -0.0144
   0.0924
  -0.3401
   0.7652
  -1.0694
   0.9058
  -0.4257
   0.0852

Result by Matlab:
   1.0e+14 *

  -0.0000
   0.0011
  -0.0145
   0.0925
  -0.3407
   0.7665
  -1.0712
   0.9073
  -0.4264
   0.0853

Norm of residuum:
   1.0242e-04


```

```
Norm of residuum after residual correction:
   1.4074e-04

Condition number:
   2.4236e+14
```

Task_2_a:



Task_2_b:



Comments on residual correction:

As the result shows, on task a it works. The solution error is smaller after residual correction method. However, in task b it is even larger. The main reason is caused by the larger condition number in task b. The matrix is ill-conditioned.

Task 3:

3. Write a general program for solving the system of n linear equations $\mathbf{Ax} = \mathbf{b}$ using the Gauss-Seidel and Jacobi iterative algorithms. Apply it for the system:

$$
\begin{aligned}
12x_1 + 2x_2 + x_3 - 6x_4 &= 6 \\
4x_1 - 15x_2 + 2x_3 - 5x_4 &= 8 \\
2x_1 - x_2 + 8x_3 - 2x_4 &= 20 \\
5x_1 - 2x_2 + x_3 - 8x_4 &= 2
\end{aligned}
$$

and compare the results of iterations plotting norm of the solution error $||\mathbf{Ax}_k - \mathbf{b}||_2$ versus the iteration number k=1,2,3,… until the assumed accuracy $||\mathbf{Ax}_k - \mathbf{b}||_2 < 10^{-10}$ is achieved. Try to solve the equations from problem 2a) and 2b) for n=10 using a chosen iterative method.

Algorithm:

Jacobi method:

Step 1: Divide matrix A into L, D, U three parts.

Step 2: Set a tolerance.

Step 3: Set a loop according to the book. After the solution error is smaller than tolerance, break the iteration.

$$\mathbf{D}\mathbf{x}^{(i+1)} = -\left(\mathbf{L} + \mathbf{U}\right)\mathbf{x}^{(i)} + \mathbf{b}, \qquad i = 0, 1, 2, \ldots$$

Gauss-Seidel method:

Step 1: Divide matrix A into L, D, U three parts.

Step 2: Set a tolerance.

Step 3: Set a loop according to the book. After the solution error is smaller than tolerance, break the iteration.

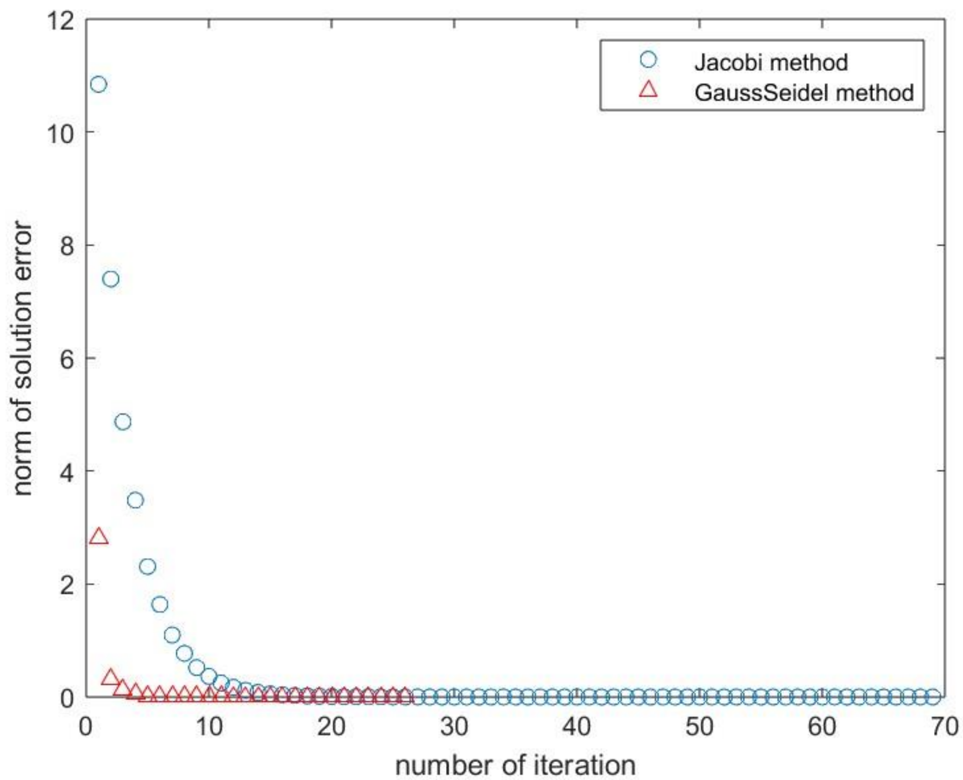
$$\mathbf{w}^{(i)} = \mathbf{U}\mathbf{x}^{(i)} - \mathbf{b}.$$

$$\mathbf{D}\mathbf{x}^{(i+1)} = -\mathbf{L}\mathbf{x}^{(i+1)} - \mathbf{U}\mathbf{x}^{(i)} + \mathbf{b}, \qquad i = 0, 1, 2, \dots$$

Result:

```
Task_3
Result from Jacobi method:
   0.560527423202912
  -0.210408191894976
   2.448309720860787
   0.458970402577874

Result from GaussSeidel method:
   0.560527423207681
  -0.210408191894173
   2.448309720859613
   0.458970402585795
```

Comparison of the results of iteration number and norm of the solution error:



Solve task2a:

```
Task 2a using Jacobi method:
  -0.339213569344186
  -0.335281416071006
  -0.272474927094786
  -0.199568146512340
  -0.124933951998973
  -0.050035565498079
   0.024720558991960
   0.098358919437181
   0.165432957619007
   0.194238826270512

Task 2a using GaussSeidel method:
  -0.339213569348042
  -0.335281416075601
  -0.272474927099257
  -0.199568146517268
  -0.124933952003346
  -0.050035565502469
   0.024720558988638
   0.098358919434324
   0.165432957617312
   0.194238826269552

Task 2a by Matlab:
  -0.339213569345671
  -0.335281416074028
  -0.272474927098497
  -0.199568146516955
  -0.124933952003231
  -0.050035565502430
   0.024720558988649
   0.098358919434327
   0.165432957617313
   0.194238826269552
```

Solve task2b:

```
Task 2b using Jacobi method:
   1.0e+91 *
  -1.571638092276401
  -2.253318771452647
  -2.781586734492064
  -3.206765474817626
  -3.558077853323158
  -3.854125493056358
  -4.107496189141665
  -4.327094039266652
  -4.519434721113141
  -4.689416679610037

Task 2b using GaussSeidel method:
   1.0e+03 *
  -0.272293087753558
   1.103005490542011
  -1.358268628970815
   1.080651030666311
  -1.151318748460034
   0.999251566837857
  -1.116907723151576
   0.947273981159989
  -1.102615473180369
   0.919580804149655

Task 2b by Matlab:
   1.0e+14 *
  -0.000032744878795
   0.001142096966142
  -0.014468591376950
   0.092514352126924
  -0.340655947827453
   0.766534492098572
  -1.071240098475689
   0.907306001844175
  -0.426441959200729
   0.085342829491853
```

We can see that for task2a it works properly. However in task2b the results are not correct. It is because of the ill-conditioned matrix in task2b.

Task 4:

4. Write a program of the QR method for finding eigenvalues of 5×5 matrices:
a) without shifts;
b) with shifts calculated on the basis of an eigenvalue of the 2×2 right-lower-corner submatrix.
Apply and compare both approaches for a chosen symmetric matrix 5×5 in terms of numbers of iterations needed to force all off-diagonal elements below the prescribed absolute value threshold 10^{-6}, print initial and final matrices. Elementary operations only permitted, commands "qr" or "eig" must not be used (except for checking the results).

Algorithm:

Without shift:

Step 1: QR factorize A, then iterate A with R*Q.

Step 2: Check if off-diagonal element is smaller than threshold. If so end the loop.

With shift:

Step 1: Extract 2*2 matrix from lower right corner, find eigenvalue closest to bottom right corner.

Step 2: Create new shifted matrix then QR factorize it.

Step 3: A = R*Q + shift.

Result:

```
>> Task_4
Matrix A to be factorize:
    19    13    15    18     3
     2    12     2    16     1
     3     2    18    11    19
     3    19    19     4     7
     4    15    20     8     6
```

```
Matrix Q:
    0.9512   -0.2542   -0.1537   -0.0767    0.0337
    0.1001    0.4422   -0.4782    0.7512    0.0370
    0.1502   -0.0425    0.8061    0.5050    0.2659
    0.1502    0.7074    0.0846   -0.4097    0.5495
    0.2003    0.4874    0.3011   -0.0830   -0.7904

Matrix R:
   19.9750   19.7247   24.0301   22.5782    8.0601
         0   22.6702   19.4976    8.7627    6.7497
         0         0   18.8785    1.1967   16.7763
         0         0         0   13.8927    6.7514
         0         0         0         0    4.2953

Matrix Q by matlab:
   -0.9512    0.2542    0.1537   -0.0767   -0.0337
   -0.1001   -0.4422    0.4782    0.7512   -0.0370
   -0.1502    0.0425   -0.8061    0.5050   -0.2659
   -0.1502   -0.7074   -0.0846   -0.4097   -0.5495
   -0.2003   -0.4874   -0.3011   -0.0830    0.7904

Matrix R by matlab:
  -19.9750  -19.7247  -24.0301  -22.5782   -8.0601
         0  -22.6702  -19.4976   -8.7627   -6.7497
         0         0  -18.8785   -1.1967  -16.7763
         0         0         0   13.8927    6.7514
         0         0         0         0   -4.2953

Eigenvalues without shift:
   49.6016
  -15.0785
   14.2019
   13.7642
   -3.4893

Iteration number without shift:
     1000000
```

```
Eigenvalues with shift:
  49.6016 - 0.0000i
 -15.0785 - 0.0000i
  14.2019 - 0.0000i
  13.7642 + 0.0000i
  -3.4893 + 0.0000i

Iteration number with shift:
     1

Eigenvalues by matlab:
   49.6016
  -15.0785
   -3.4893
   14.2019
   13.7642
```

Comparison:

We can see that without shifts the code is much easier. However, the iteration number is huge, and it took a lot of time to run in my computer. QR method with shifts only needs one iteration to get the result. I ran it several times with rand(5) matrices generated by matlab and found sometimes QR method without shift can fail.

Reference:

Task2:

https://youtu.be/c0i8hFsOV3A?si=whq0vviShpB1mZ0f

Task3:

Numerical Methods by Piotr Tatjewski

Task4:

https://www.youtube.com/watch?v=DAyGL_-XxkA

Numerical Methods by Piotr Tatjewski

Code:

Task 1:

```
clear all;

format long;

disp('eps = ');
disp(eps)%mecheps read from MATLAB

mecheps = 1; %
cou = 0; %count how many times the loop goes

while 1 + (mecheps/2) > 1
    mecheps = mecheps/2;
    cou = cou + 1;
end

disp('cou = ');
disp(cou);

disp('mecheps = ');
disp(mecheps);
```

Task 2:

Main:

```
format short;


% examples when n=10
n=10;

[A, B]=Task_2_a(n);
cond_a=cond(A);
[a, norm_r_1, norm_corrected_r_1]=Gauss(A, B);
b=A\B;
disp("Result by me:");
disp(a);
disp("Result by Matlab:");
disp(b);
disp("Norm of residuum:");
disp(norm_r_1);
disp("Norm of residuum after residual correction:");
disp(norm_corrected_r_1);
disp("Condition number:")
disp(cond_a);

[C, D]=Task_2_b(n);
cond_c=cond(C);
[c, norm_r_2, norm_corrected_r_2]=Gauss(C, D);
d=C\D;
disp("Result by me:");
disp(c);
disp("Result by Matlab:");
disp(d);
disp("Norm of residuum:");
disp(norm_r_2);
disp("Norm of residuum after residual correction:");
disp(norm_corrected_r_2);
disp("Condition number:")
disp(cond_c);

% plot norm of redsiduum and number of equations
plot_task('a',n);
plot_task('b',n);
```

Task2a:

```
% Task 2a Matrix Generation

function [A, B]=Task_2_a(n)

    % create matrix A
    A=zeros(n);
    for i=1:n
        for j=1:n
            if i==j
                 A(i,j)=6;
             elseif i==j-1||i==j+1
                 A(i,j)=-1;
             end
         end
     end

     % create vector B
     for i=1:n
         B(i)=-2+0.3*i;
         i=i+1;
     end
     B=B'; % transpose B to column vector



disp("Task2a:");
disp("Matrix A:");
disp(A);
disp("Vector B:");
disp(B);
end
```

Task2b:

```

% Task 2b Matrix Generation

function [A, B]=Task_2_b(n)

```

```
    % create matrix A&B
    A=zeros(n);
    B=zeros(n,1);
    for i=1:n
        for j=1:n
            A(i,j)=1/(4*(i+j+1));
        end
        if ~mod(i,2)
            B(i,1)=7/(6*i);
        else
            B(i,1)=0;
        end
     end

disp("Task2b:");
disp("Matrix A:");
disp(A);
disp("Vector B:");
disp(B);
end
```

Gaussian elimination:

```
% Gaussian Elimination with Partial Pivoting

function [x, norm_r, norm_corrected_r]= Gauss(A, B)

    n=length(B); % number of rows
    x=zeros(n,1);
    c=zeros(1,n);
    d1=0;

     for i=1:n-1

        max=abs(A(i,i)); % find the max valuse of first column
        m=i;

        for j=i+1:n
            if max<abs(A(j,i))
                max=abs(A(j,j));
                m=j;
```

```
            end
        end
        if(m~=i) % if pivoting is necessary then swap rows
            for k=i:n
                c(k)=A(i,k);
                A(i,k)=A(m,k);
                A(m,k)=c(k);
            end
            d1=B(i);
            B(i)=B(m);
            B(m)=d1;
        end
        % the elimination pass
        for k=i+1:n
            for j=i+1:n
                A(k,j)=A(k,j)-A(i,j)*A(k,i)/A(i,i);% choose the factor for division
            end
            B(k)=B(k)-B(i)* A(k,i)/A(i,i);
            A(k,i)=0;
        end
    end
    % backward substitution
    x(n)=B(n)/A(n,n);
    for i=n-1:-1:1
        sum=0;
        for j=i+1:n
            sum=sum+A(i,j)*x(j); % left side of the equation
        end
        x(i)=(B(i)-sum)/A(i,i);
    end
    %residuum
```

```
    r = A*x - B;
    norm_r = norm(r);
    % residual correction for once
    delta =A\r ;
    corrected_x = x-delta;
    corrected_r = (A*corrected_x)-B;
    norm_corrected_r = norm(corrected_r);
end
```

Plot task:

```
function plot_task(task_num,n)
    % record the results
    record_n=zeros(n);
    record_norm_r=zeros(n);
    % for task a
     if task_num == ‘a’
         for i=1:8 % max i is 8 for my computer
             [E,F]=Task_2_a(n);
             [~,norm_r,~]=Gauss(E, F);
             record_n(i)=n;
             record_norm_r(i)=norm_r;
             n=n*2;
         end
     figure(1);
     plot(record_n, record_norm_r, ‘o’)
     xlabel(“number of equations”);
     ylabel(“norm of residuum”);
     % for task b
     elseif task_num == ‘b’
         for i=1:8 % max i is 8 for my computer
```

```
            [E,F]=Task_2_b(n);
            [~,norm_r,~]=Gauss(E, F);

            record_n(i)=n;
            record_norm_r(i)=norm_r;
            n=n*2;
        end
    end

    figure(2);
    plot(record_n, record_norm_r, ‘o’)
    xlabel("number of equations");
    ylabel("norm of residuum");

end
```

Task 3:

Main:

```

A=[12,2,1,-6;4,-15,2,-5;2,-1,8,-2;5,-2,1,-8];
b=[6;8;20;2];

% solve the equations in two methods
[x_1, n_1, norm_error_1]=Task_3_Jacobi(A,b);
disp("Result from Jacobi method:")
disp(x_1);
%disp(n);
%disp(norm_error);

[x_2, n_2, norm_error_2]=Task_3_GaussSeidel(A,b);
disp("Result from GaussSeidel method:")
disp(x_2);
%disp(n);
%disp(norm_error);

% compare the results of iteration number and norm of the solution error
%plot_task(A,b)

% solve 2a and 2b in both methods
[C, d] = Task_2_a(10);
[x_3, n_3, norm_error_3]=Task_3_Jacobi(C,d);
[x_4, n_4, norm_error_4]=Task_3_GaussSeidel(C,d);
disp("Task 2a using Jacobi method:")
disp(x_3);
disp("Task 2a using GaussSeidel method:")
disp(x_4);
disp("Task 2a by Matlab:")
disp(C\d);

[E, f] = Task_2_b(10);
[x_5, n_5, norm_error_5]=Task_3_Jacobi(E,f);
[x_6, n_6, norm_error_6]=Task_3_GaussSeidel(E,f);
disp("Task 2b using Jacobi method:")
disp(x_5);
disp("Task 2b using GaussSeidel method:")
disp(x_6);
```

```
disp("Task 2b by Matlab:")
disp(E\f);

```

Gauss-Seidel:

```

function [x, n, norm_error] = Task_3_GaussSeidel(A,b)

    x = zeros(size(A, 1), 1);

    % create L,D,U
    L=tril(A,-1);
    D=diag(diag(A));
    U=triu(A,1);

     % tolerance
     t=1e-10;

     for i=1:100

         w = U*x-b;

         n(i,1)=i;% count iteration times

         for j=1:size(A, 1)

             x(j) = -w(j);

             % -l*x
             for k=1:(j-1)
                  x(j) = -x(k)*L(j,k) + x(j);
             end

             % /d
             x(j) = inv(D(j, j)) * x(j);

             n(i,1)=i;% count iteration times

             norm_error(i,1)=norm(A*x - b);% record the norm of solution error

```

```
            end

            if norm(A*x - b) <= t % stop test
                break;
            end
        end
end
```

Jacobi's:

```
function [x, n, norm_error] = Task_3_Jacobi(A,b)

    x = zeros(size(A, 1), 1);

    % create L,D,U
    L=tril(A,-1);
    D=diag(diag(A));
    U=triu(A,1);

     % tolerance
     t=1e-10;

     for i=1:100

         x=D \ (b + (-L-U)*x);

         n(i,1)=i;% count iteration times

         norm_error(i,1)=norm(A*x - b);% record the norm of solution error

         if norm(A*x - b) <= t % stop test
             break;
         end
     end
end
```

Plot task:

```
function plot_task(A, b)

    [~, n_1, norm_error_1] = Task_3_Jacobi(A,b);
    [~, n_2, norm_error_2] = Task_3_GaussSeidel(A,b);

    % graph
    figure(3);
    plot(n_1, norm_error_1, 'o')
    hold on;
     plot(n_2, norm_error_2, '^r')
     hold off;

     xlabel("number of iteration");
     ylabel("norm of solution error");
     legend("Jacobi method","GaussSeidel method");


end
```

Task 4:

Main:

```

A = randi(10,5,5);

disp("Matrix A to be factorize:");
disp(A);

[Q, R] = QR(A);
[C, D] = qr(A);

disp("Matrix Q:");
disp(Q);
disp("Matrix R:");
disp(R);
disp("Matrix Q by matlab:");
disp(C);
disp("Matrix R by matlab:");
disp(D);

[e_1, i_1] = eigenvalue_no_shift(A);
disp("Eigenvalues without shift: ");
disp(e_1);
disp("Iteration number without shift: ");
disp(i_1);

[e_2, i_2] = eigenvalue_shift(A);
disp("Eigenvalues with shift: ");
disp(e_2);
disp("Iteration number with shift: ");
disp(i_2);

e_mat = eig(A);
disp("Eigenvalues by matlab: ");
disp(e_mat);
```

QR decomposition:

```
function [Q,R] = QR(A)

    [m, n] = size(A);

    Q = zeros(m,n);
    R = zeros(n,n);
    d = zeros(1,n);

     % factorization
     for i=1:n
         Q(:,i) = A(:,i);
         R(i,i) = 1;
         d(i) = Q(:,i)'*Q(:,i);

         for j=i+1:n
             R(i,j) = (Q(:,i)'*A(:,j))/d(i);
             A(:,j) = A(:,j)-R(i,j)*Q(:,i);
         end
     end

     % normalization
     for i=1:n
         dd = norm(Q(:,i));
         Q(:,i) = Q(:,i)/dd;
         R(i,i:n) = R(i,i:n)*dd;
     end
end
```

Eigenvalue without shift:

```
function [e, i] = eigenvalue_no_shift(A)

% iteration counter
i = 1;
```

```
% when the error is greater than threshold, do iteration
while max(max(A-diag(diag(A)))) > 1e-6 && i <1000000
    [Q, R] = QR(A);
    A = R * Q;
    i = i+1;
end

e = diag(A);

end
```

Find roots:

```
function [x1, x2] = quadpolynroots(a, b, c)

    delta1 = sqrt((b^2) - (4*a*c)) - b;
    delta2 = -sqrt((b^2) - (4*a*c)) - b;

    if abs(delta1) > abs(delta2)
        delta = delta1;
    else
        delta = delta2;
     end

     x1 = delta/(2*a);
     x2 = ((-b/a) - x1);
end
```

Eigenvalues with shift:

```
function [e, i] = eigenvalue_shift(A)

    n = size(A,1);
    e = diag(ones(n));

    initial_sub = A; % initial submatrix

    for k = n:-1:2
        DK = initial_sub; % initial matrix to calculate
```

```

        i = 0;
        while max(abs(DK(k,1:k-1)))>1e-6 && i<=1000000
            DD = DK(k-1:k,k-1:k); % bottom 2x2 right corner submatrix
            [ev1, ev2] = quadpolynroots(1,-(DD(1,1)+DD(2,2)),DD(2,2)*DD(1,1)-DD(2,1)*DD(1,2));

            if abs(ev1 - DD(2,2)) < abs(ev2 - DD(2,2))
                shift = ev1; % shift
            else
                shift = ev2;
            end

            DP = DK - eye(k)*shift; % shifted matrix
            [Q, R] = QR(DP); % QR factorization
            DK = R * Q + eye(k)*shift; % transformed matrix
            i = i+1;
        end

        e(k) = DK(k,k);

        if k > 2
            initial_sub = DK(1:k-1,1:k-1); % matrix deflation
        else
            e(1) = DK(1,1); % last eigenvalue
        end
    end
end
```