Numerical Methods

Project Number: A9

Name Surname: Beste Baydur

Index Number: 295597

Winter Semester 2020

TASK 1:

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

Theory:

Machine epsilon is an important general term for machine accuracy of a computer. It is the smallest non-negative unit that can change when added to a number.

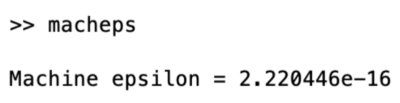
$$\varepsilon + 1 \neq 1.$$

For the t-bit mantissa, the machine precision is eps = 2^{-t}

IEEE 754 Representation can have 32 bits or 64 bits and, in both cases, first bit shows whether the number is positive or negative. For negative, the first bit is 1 and for positive it is 0. In 32 bits, 8 bits are reserved for shifted exponent and 32-8=24 bits will be left but as I mentioned before the first bit is for the sign of number so there are 23 bits left for mantissa. In 64 bits, 11 bits are reserved for shifted exponent and 64-11=53 bits are left and since the first bit representing sign implies here as well, 52 bits are there for mantissa.

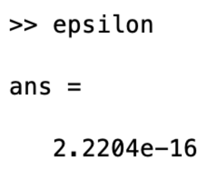
Since know we know how many bits there are for mantissa with 64 bits, we can calculate the eps for 52.

When we operate the program, we get the result:



```
>> macheps

Machine epsilon = 2.220446e-16
```

The result matches with the result of the default eps function:



```
>> epsilon

ans =

   2.2204e-16
```

The result also matches with the reference value of the IEEE 754-20008 standard (eps ≈ 2.22e^{-16} ≈ 2^{-52})

TASK 2:

Write a general program solving a system of n linear equations Ax = 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 A and vector b, for increasing numbers of equations n = 10,20,40,80,160... until the solution time becomes prohibitive (or the method fails), for:

$$\text{a) } a_{ij} = \begin{cases} 8 & \text{for } i = j \\ 3 & \text{for } i = j-1 \text{ or } i = j+1, \\ 0 & \text{other cases} \end{cases} \qquad b_i = 4.4 + 0.6\,i, \qquad i,j = 1,\ldots,n;$$

$$\text{b) } a_{ij} = 1/[3(i+j+1)], \qquad b_i = 5/(3\,i),\ i-\text{even};\ b_i = 0,\ i-\text{odd},\ \ i,j = 1,\ldots,n.$$

For each case a) and b) calculate the solution error defined as the Euclidean norm of the vector of residuum r = Ax–b, where 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.

There are 2 different types of methods for solving system of linear equations:

1- Finite methods
2- Iterative methods

- In finite methods, in order to get the solution, we have to apply a finite number of elementary arithmetical operations defined by the method and the dimensions of the problem. Gauss elimination method is an example to this type of methods.

- In iterative methods, the initial point that we start from is improved by the method in subsequent iterations. There are no certain number of iterations, it is related to assumed solution accuracy. They are used in sparse matrices which means that most of the elements of the matrix are zeros.

For an iterative method to work, 2 conditions should be met:

1- Convergence condition

$$\text{sr}(M) < 1$$

where sr(M) stands for spectral radius.

2- Coincidence condition

$$\widehat{\mathbf{x}} = \mathbf{M}\widehat{\mathbf{x}} + \mathbf{w}$$

Gaussian Elimination

The Gauss elimination algorithm is divided into two steps:

1- The Gaussian elimination phase
2- Back-substitution phase

- Gaussian elimination phase:

In order to proceed to the back substitution, we need this phase. The objective is to get is to convert the system of linear equations Ax=b to an equivalent system with an upper-triangular matrix. The name of the form that we obtain is also called row echelon form. First, we assume that $a_{11}^{(1)} \neq 0$ and we define the row multipliers by:

$$l_{i1} \overset{\text{df}}{=} \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}, \qquad i = 2, 3, ..., n.$$

By multiplying first row by l_{i1} and subtracting from the i-th row, we get the first row. By changing the value of i till n:

$$\mathbf{w}_i = \mathbf{w}_i - l_{i1}\mathbf{w}_1 \quad \Longleftrightarrow \quad \begin{aligned} a_{ij}^{(2)} &= a_{ij}^{(1)} - l_{i1}a_{1j}^{(1)},\ j = 1, 2, ..., n, \\ b_i^{(2)} &= b_i^{(1)} - l_{i1}b_1^{(1)}, \qquad\qquad i = 2, 3, ..., n. \end{aligned}$$

we obtain the result:

$$\begin{array}{ccccccccc} a_{11}^{(1)}x_1 & + & a_{12}^{(1)}x_2 & + & \cdots & + & a_{1n}^{(1)}x_n & = & b_1^{(1)}, \\ & & a_{22}^{(2)}x_2 & + & \cdots & + & a_{2n}^{(2)}x_n & = & b_2^{(2)}, \\ & & \vdots & & & & \vdots & & \vdots \\ & & a_{n2}^{(2)}x_2 & + & \cdots & + & a_{nn}^{(2)}x_n & = & b_n^{(2)}, \end{array}$$

$$\text{i.e.,} \quad \mathbf{A}^{(2)}\mathbf{x} = \mathbf{b}^{(2)}.$$

To be more general compared to the given example, it can be said that:

$$\mathbf{A}^{(n)}\mathbf{x} = \mathbf{b}^{(n)}$$

Where $A^{(n)}$ stands for an upper-triangular matrix. As I mentioned in the beginning, we fulfill the requirements to proceed to next step: Back substitution.

- Back substitution:

In this method, we proceed from the last equation and we move to the equation before it, until we reach the first one.

Gaussian elimination with partial pivoting:

We use pivoting in order to increase the numerical stability which can decrease due to operations with very small numbers. There are two types of pivoting: Partial and full pivoting.
In both cases, the pivot row is chosen. But afterwards, in full pivoting, we choose also the pivot column. It is better for getting more accurate results, but it also is more complex than partial pivoting. Therefore, we prefer using the partial pivoting.

Partial pivoting:

In partial pivoting first, we choose the central element as that from elements

$$a_{jk}^{(k)} \ (k \le j \le n)$$

With the biggest absolute value from a row by doing:

$$\left|a_{ik}^{(k)}\right| = \max_{j}\left\{\left|a_{kk}^{(k)}\right|, \left|a_{k+1,k}^{(k)}\right|, \ldots, \left|a_{nk}^{(k)}\right|\right\}$$

Next, we swap the i-th -which is pivot row- and k-th row. Pivoting should be implemented at every step to obtain smaller numerical errors.

Residual correction (iterative improvement):

The solution that we obtain can be not as precise as we want sometimes. Residuum correction helps us to get a more accurate solution. In order to apply this method, we solve the system of linear equations with respect to δx:

$$A\delta x=r$$

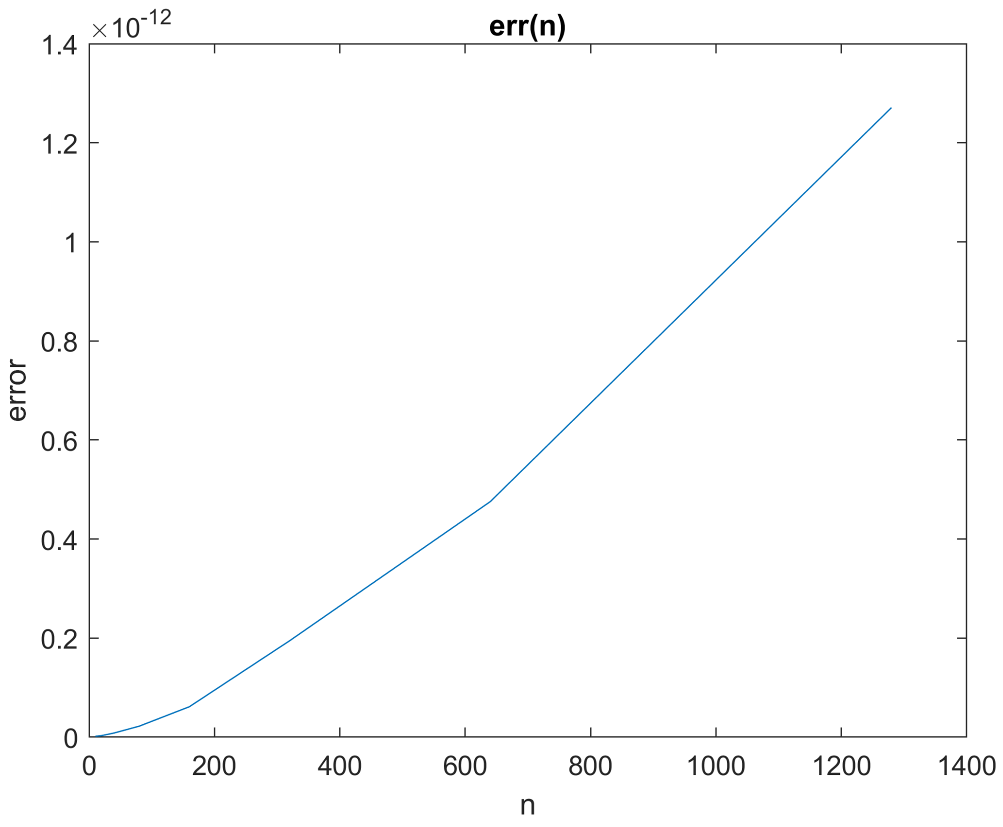
Then we calculate the error, which can be called residuum:

$$x^{(2)} = x^{(1)} - \delta x.$$

If we still can't obtain a result that is accurate enough, we repeat the procedure.

Results:

A)



From the result graph, we can see that there is a proportional relation between error and n. But the increase varies for different values of n. Between 0-200, the increase is comparably less than the rest of the graph. Between 200-600 increase is more rapid. After 640, as we can see the increase gets much faster.

n=10 Elapsed time is 0.009087 seconds.
n=20 Elapsed time is 0.002742 seconds.
n=40 Elapsed time is 0.000760 seconds.
n=80 Elapsed time is 0.001496 seconds.
n=160 Elapsed time is 0.015208 seconds.
n=320 Elapsed time is 0.077582 seconds.
n=640 Elapsed time is 1.086093 seconds.
n=1280 Elapsed time is 15.424609 seconds. ->Here elapsed time rapidly increases

For n = 10 result before correction was:

```
A =

    0.4988
    0.3365
    0.4704
    0.4757
    0.5278
    0.5835
    0.5829
    0.7289
    0.5401
    1.0975

err =

   1.7764e-15
```

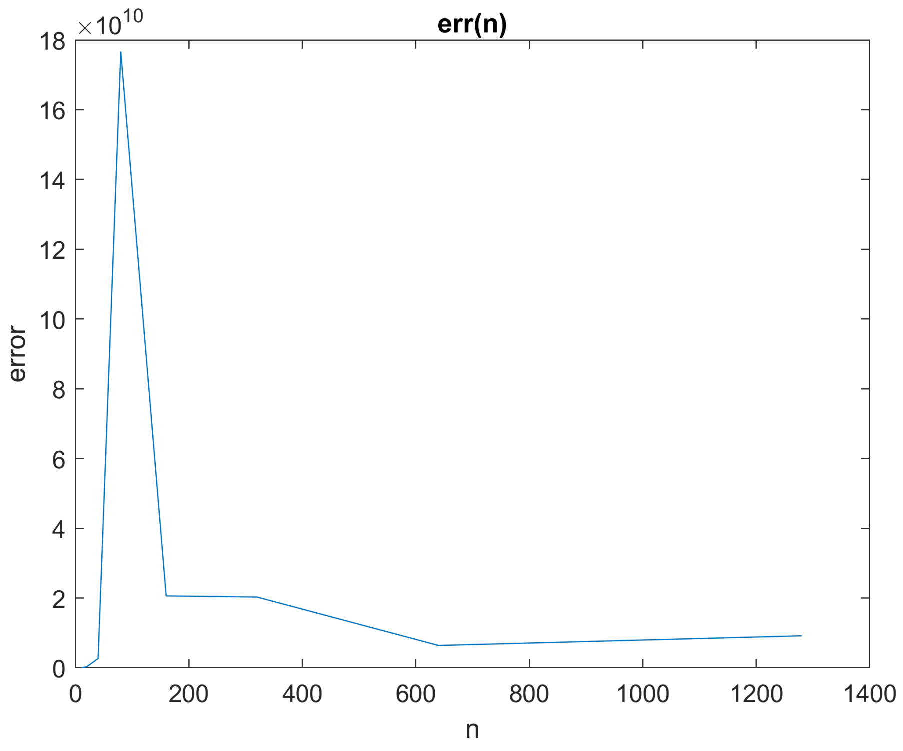
and after correction:

A has not changed

```
err=

    5.32907e-15
```

As we can see, the error before and after the correction has a very minor difference. Also considering that the value is already very low, there was not a considerable effect of the residual correction.

B)



```
n=10 Elapsed time is 0.000058 seconds.
n=20 Elapsed time is 0.000062 seconds.
n=40 Elapsed time is 0.000189 seconds.
n=80 Elapsed time is 0.001014 seconds.
n=160 Elapsed time is 0.011116 seconds.
n=320 Elapsed time is 0.079543 seconds.
n=640 Elapsed time is 1.185258 seconds.
n=1280 Elapsed time is 15.655202 seconds.
```
->here elapsed time rapidly increases

For n = 10 result before correction was

```
A =

  1.0e+14 *

  -0.0000
   0.0012
  -0.0155
   0.0990
  -0.3647
   0.8206
  -1.1468
   0.9713
  -0.4565
   0.0914

err =

  2.0144e+04
```

After correction:

A has not changed

```
err=

  0.00447301
```

The program has broken down when n became 80, because of ill-conditioned matrix, which means that small perturbations can result into large change of the value in the final result. Residual corrections behavior is similar. After correction A has not changed because error is too small to have any impact on the numbers we have obtained. same as before and after correction residuum didn’t influence the results.

TASK 3:

Write a general program for solving the system of n linear equations Ax = b using the Gauss Seidel and Jacobi iterative algorithms. Apply it for the system:

$$\begin{aligned} 4x_1 + x_2 + x_3 + x_4 &= 10 \\ x_1 + 5x_2 + 2x_3 + x_4 &= 20 \\ x_1 + 2x_2 + 6x_3 + x_4 &= 30 \\ x_1 + x_2 + 3x_3 + 7x_4 &= 40 \end{aligned}$$

Jacobi's Method

Jacobi's method is an iterative method, as mentioned on task 2. We start Jacobi's Method by decomposition of matrix A to:

$$A = L + D + U$$

Where:

- L = Sub diagonal matrix
- D = Diagonal matrix
- U = Matrix with entries above the diagonal

We can write the system of linear equations Ax = b as:
$Dx = -(L+U)x+b$
Assuming that matrix D is nonsingular, we can write an iterative method such as:

$$Dx^{(i+1)} = -(L + U)x^{(i)} + b, \quad i = 0,1,2,\ldots$$

Jacobi's method is also a parallel computational scheme.

Gauss-Seidel Method

Gauss-Seidel method starts the same way as Jacobi method does with the equation:

$$A = L + D + U$$

After this, we write the system of equations this way:

$$(L + D)x = -Ux + b$$

As we did on the last method, we assume D is nonsingular, we can write an iterative method such as:

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

Gauss Seidel method's computational structure is sequential, as the operations have to be performed sequentially opposed to Jacobi's method. Sufficient convergence condition is the strong row or column diagonal dominance. Usually Gauss-Seidel method is faster convergent than Jacobi's.

Euclidian norm

Formula:

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$$

where x is norm and x_i is residuum. Calculation of the Euclidian norm simplifies the understanding of the size of the matrix and vector. This allows us to accurately judge their compatibility and helps in error analysis

Stop Tests

We have to check the criteria before terminating iterative methods.
These criteria can be proposed:

1- Checking differences between two subsequent iteration points:

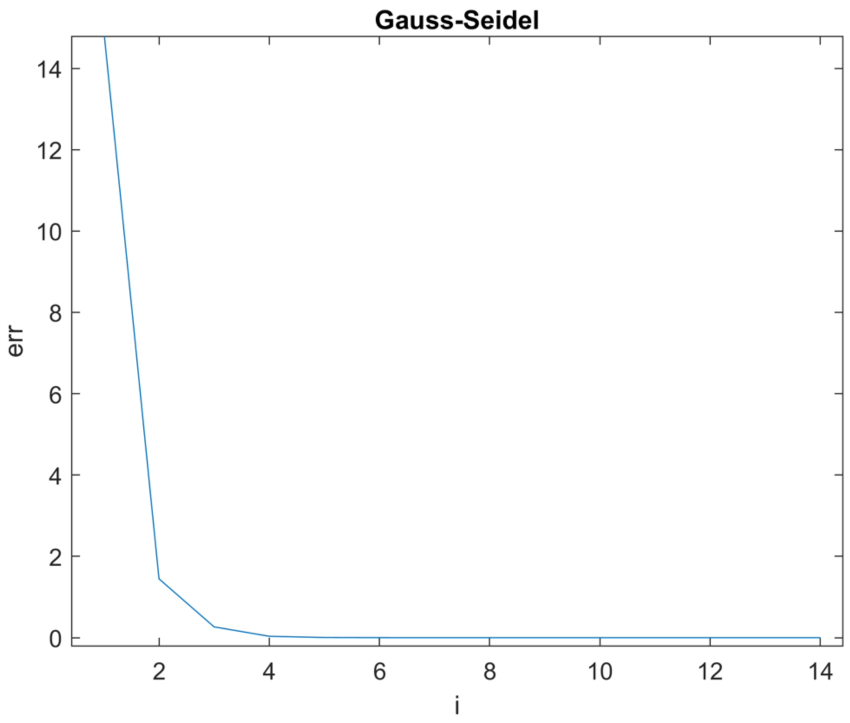
$$\left\|\mathbf{x}^{(i+1)} - \mathbf{x}^{(i)}\right\| \leq \delta,$$

where δ is an assumed tolerance. Since we are focused on the accuracy, even though the test above is fulfilled, we can perform additional tests with higher level.

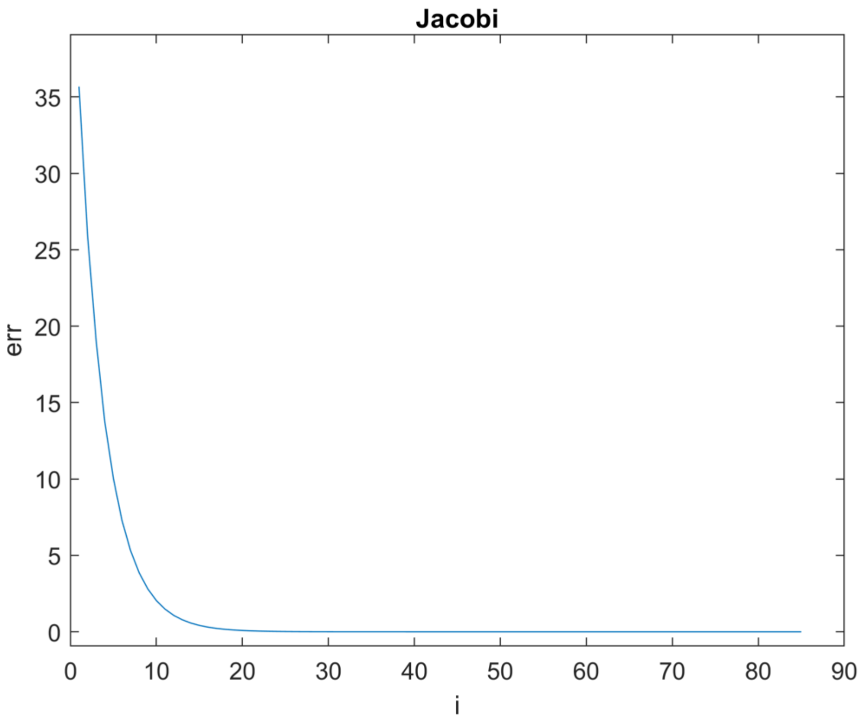
2- Checking a norm of the solution error vector:

$$\left\|\mathbf{A}\mathbf{x}^{(i+1)} - \mathbf{b}\right\| \leq \delta_2,$$

where δ_2 is an assumed tolerance. In case that the test is not satisfied, the value of δ in the equation of checking differences between two points, can be decreased and the process restarts. The value of δ is limited by the numerical errors so its value cannot be too low.



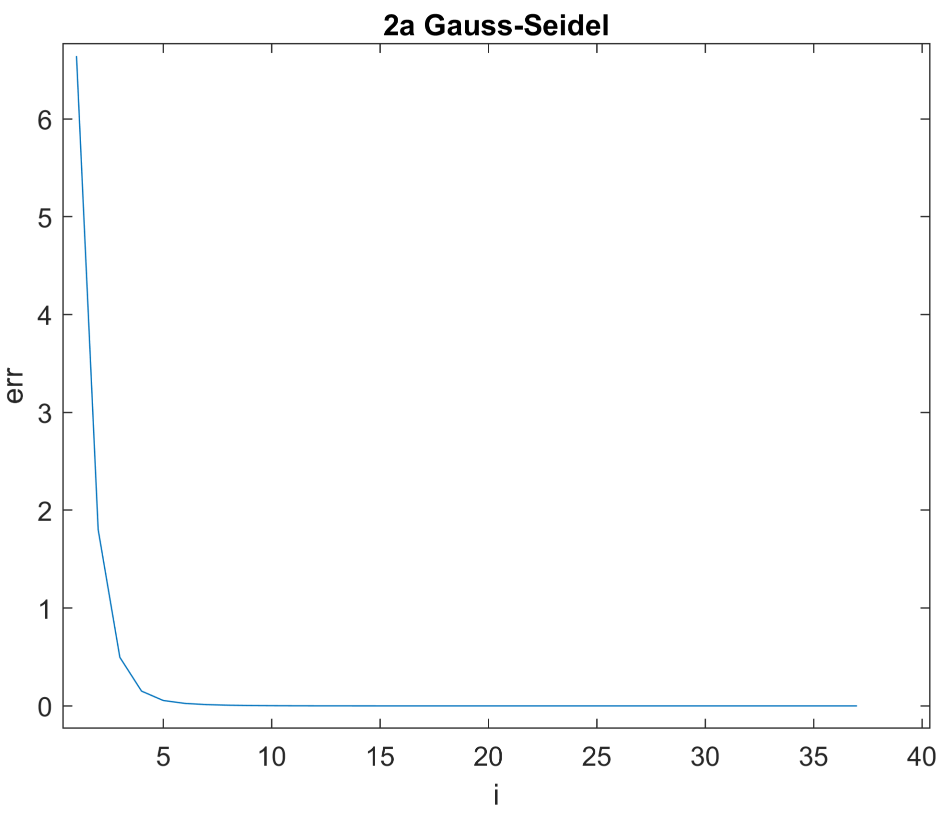
Gauss-Seidel iterations = 14



Jacobi iterations = 85

For problem to be solved using Gauss-Seidel method 14 iterations are needed, while Jacobi's method needs 85 iterations to solve the problem and to achieve needed accuracy. As we can see Gauss Seidel method for the given system has achieved result faster and with smaller error, than Jacobi's method that is why we will use this method for n=10 and solve systems given from tasks 2a and 2b

2a by Gauss seidel:

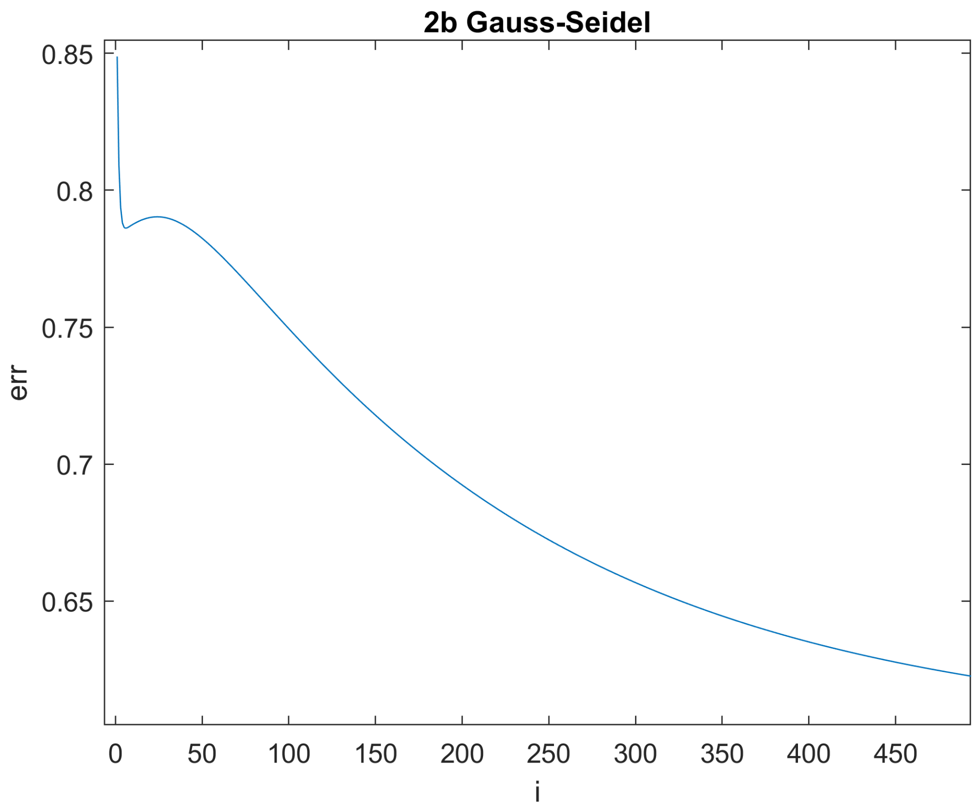


```
x =

   0.4989
   0.3363
   0.4710
   0.4744
   0.5305
   0.5775
   0.5961
   0.6996
   0.6049
   0.9539
```

```
iter =

    37
```

It took 37 iteration for Gauss-Seidel method to solve the problem. After we compared the result, we can clearly see that results didn't change, except for the error.

2b)



```
iter =

   500
```

(maximum)
Result was not achieved

```
SR =

    0.0457
```

SR < 1

System 2b is more sophisticated for iteration method than 2a. It has achieved max iterations but still hasn't achieved accuracy expected. Plot is attached above. To solve the problem, I calculated spectral radius for this system (2b). We have obtained SR = 0.0457, after calculating max(abs(eig(M))) which means that the problem will be solved after some time, but longer that we would want to.

TASK 4:

Write a program of the QR method for finding eigenvalues of 5x5 matrices:

a) **Without Shifts**

b) **With shifts calculated on the basis of an eigenvalue of the 2x2 right-lower-corner submatrix.**

Apply and compare both approaches for a chosen symmetric matrix 5x5 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).

QR method is for finding eigenvalues and eigenvectors of a matrix. Before we start the algorithm, it is suggested that we transform the matrix that we use in triadiagonal form(the Hessenberg form of symmetric matrices) since it increases the effectiveness of the calculations. The reason behind it is that in QR factorization triadiagonal form is sustained. In QR method we decompose the matrix A into QR, where Q stands for an orthogonal matrix and R stands for an upper triangular matrix.

In the given task I will use modified Gram-Schmidt algorithm. In this method, numerical properties are better thus there are less error issues compared to the classical version. In the classical version the columns are orthogonalized one after another but in the modified version, all the next columns are immediately orthogonalized with respect of this column.

a) Without shifts

$$A^{(k)} = Q^{(k)}R^{(k)}$$
$$A^{(k+1)}= Q^{(k)}R^{(k)}$$
$$Q^{(k)} \text{ is orthogonal so;}$$
$$R^{(k)} = (Q^{(k)})^{-1} A(k)= (Q^{(k)})^{T} A^{(k)}$$
$$A^{(k+1)}= (Q^{(k)})^{T} A^{(k)}Q^{(k)}$$
$$A^{(k)} \rightarrow V^{-1}AV = \text{diag}\{ \lambda i \}$$

Convergence rate:

$$\frac{|a_{i+1,i}^{(k+1)}|}{|a_{i+1,i}^{(k)}|} \approx \left|\frac{\lambda_{i+1}}{\lambda_i}\right|$$

b) With shifts

$$\begin{aligned} A^{(k)} - p_k I &= Q^{(k)} R^{(k)} \\ A^{(k+1)} &= R^{(k)} Q^{(k)} + p_k I \\ &= Q^{(k)T} (A^{(k)} - p_k I) Q^{(k)} + p_k I \\ &= Q^{(k)T} A^{(k)} Q^{(k)} \end{aligned}$$

Convergence rate:

$$\left|\frac{\lambda_{i+1} - p_k}{\lambda_i - p_k}\right|$$

The Random Matrix

A = [3 9 4 5 2; 9 3 7 2 4; 4 7 3 8 8; 5 2 8 3 1; 2 4 8 1 3];

3	9	4	5	2
9	3	7	2	4
4	7	3	8	8
5	2	8	3	1
2	4	8	1	3

The results that we obtain from this matrix is below:

Without Shifts

Iterations: 40

```
  23.6312
  -9.2635
  -5.4403
```

```
    3.6356
    2.4371
```

With Shifts

Iterations : 6

```
   23.6312   -9.2635   -5.4403    3.6356    2.4371
```

“Eig” function results:

```
   -9.2635
   -5.4403
    2.4371
    3.6356
   23.6312
```

It is clear that we obtain the exact same results in QR method with shifts and “Eig” function in Matlab. When we compare the number of iterations for both actions, it is obvious that QR Method with shifts is so much faster than without shifts.

Appendix

TASK 1:

```
macheps = 1;
while 1.0 + macheps/2.0 > 1.0
macheps = macheps / 2.0;
end
fprintf(1, 'Machine epsilon = %d\n', macheps);
```

TASK 2:

```
function X = gauss_elimination(A)
    tic; %Calculation of elapsed time starts
    n = length(A(:,1));
    for i=1:n %Eliminating variables
        a = A(i,i);
        b = i;
        for j=(i+1):n %Finding the biggest coefficient
            if(abs(A(j,i))>a)
                a = abs(A(i,i));
                b = j;
            end
        end
        help = A(i,:); %Row switch
        A(i,:) = A(b,:);
        A(b,:) = help;
        for j=(i+1):n %Row echelon form
            l=A(j,i)/A(i,i);
            A(j,:) = A(j,:) - l*A(i,:);
        end
    end
    for i = n:-1:1 %Back Substitution
        for j = 1:(i-1)
            l = A(j,i)/A(i,i);
            A(j,:) = A(j,:) - l*A(i,:);
        end
    end
    X=zeros(n,1);
    for i=1:n %Result vector
        X(i,1) = A(i,(n+1))/A(i,i);
    end
    toc; %Calculation of elapsed time ends
end
```

```
function result_matrix = matrix_generator(n)
% ----2a)
    A = zeros(n,n);
    B = zeros(n,1);
    for i=1:n-1
         A(i,i) = 8;
```

```
          A(i+1, i) = 3;
          A(i, i+1) = 3;
    end
    A(n,n) = 8;
    for i=1:n
        B(i,1) = 4.4 + 0.6*i;
    end
    result_matrix = [A,B];
% ----2b)
%
%     A = zeros(n,n);
%     B = zeros(n,1);
%     for i=1:n
%         for(j=1:n)
%             A(i,j) = 1/(3*(i+j+1));
%         end
%     end
%     for i=1:n
%         if ( mod(i,2) == 0)
%            B(i,1) = 5/(3*i);
%         else
%            B(i,1) = 0;
%         end
%     end
%     result_matrix = [A,B];
end
```

```
function res = normres(A, X) %Calculating the error
    n = length(A(:,1));
    B = A(:,n+1);
    Z = A(:,1:n);
    res = norm(B-Z*X);

%     if n == 10
%         for j=1:n
%           if j~=1
%             res = Z* X-B;
%           end
%           d=inv(Z)*res; %Inverting
%           X=X-d;
%            disp('X=');
%            disp(X);
%           eunorm = norm(res,1);
%           fprintf(1, 'Solution residuum after iteration: %g
\n\n\n', eunorm);
%         end
%     end
end
```

```
n = 10;
A = gauss_elimination(matrix_generator(n))
err = normres(matrix_generator(n), A)
```

TASK 3:

```
A = [4, 1, 1, 1; 1, 5, 2, 1; 1, 2, 6, 1; 1, 1, 3, 7];
b = [10; 20; 30; 40];
tol = 1e-10;% Error tolerance
x0 = zeros(4,1);% initial guess vector
[x,err,itr]= Gauss_Seidel(A,B,x0,tol);
itrt = 1:itr;
figure
plot(itrt,err(1:length(itrt)),'-.sr','MarkerSize',8);
set(gca,'FontSize',18)
xlabel('Iteration number')
ylabel('Error')
legend('Gauss Seidel Method')
grid

disp('     x  by Gauss Seidel Method')
fprintf('%d \n ', x);

function [x, Eerr, iter] = Gauss_Seidel(A, b, x, asacry)
U = A - tril(A);
L = inv(tril(A));
mi = 100;
Eerr = zeros(1,mi);
for iter = 1 : mi
x = L * (b - U * x); r = A * x - b;
err = norm(r,2); Eerr(iter) = err; if (err < asacry)
Eerr = Eerr(1:iter);
return; end
end

function [x,err,itr]= Gauss_Seidel(A,B,x,tol)
% Gauss Seidel Method
N=size(x,1);
ernrm=1e03;
err = zeros(1,100);
itr=0;

%-------------------------------------------------------------------
------
while ernrm>=tol
    x1=x;
    for i=1:N
        summ_R=0;
        for j=1:i-1
                summ_R=summ_R+A(i,j)*x(j);
        end
        for j=i+1:N
```

```
                summ_R=summ_R+A(i,j)*x1(j);
        end
        x(i)=(1/A(i,i))*(B(i)-summ_R);
    end
    itr=itr+1;
    ernrm=norm(x1-x);
    err(itr) = ernrm;
end




function [x,err,itr]= Jacobi_itr(A,B,x,tol)
% Jacobi Method
n=size(x,1);
ernrm=1e03;
err = zeros(1,100);
itr=0;

%-------------------------------------------------------------------
----
while ernrm>tol
    x1=x;
    for i=1:n
        summ_L=0;
        for j=1:n
            if j~=i
                summ_L=summ_L+A(i,j)*x(j);
            end
        end
        x(i)=(1/A(i,i))*(B(i)-summ_L);
    end
    itr=itr+1;
    ernrm=norm(x1-x);
    err(itr) = ernrm;
end
```

TASK 4:

```
function [Q R]= QR_GramSchmidt(A)

[m n] = size(A);
Q = zeros(m);
R = zeros(m);
R(1,1) = norm(A(:,1));% norm of the first column
Q(:,1) = A(:,1)/R(1,1);

    for j = 2:n
        Q(:,j) = A(:,j);
        for i = 1:j-1
            Q(:,j) = Q(:,j)-A(:,j)'*Q(:,i)*Q(:,i);
            R(i,j) = A(:,j)'*Q(:,i);
        end
        R(j,j) = norm(Q(:,j));
        Q(:,j) = Q(:,j)/norm(Q(:,j));
    end
end



function [d rr] = qr_shifted(A)
% Code for QR algorithm with shifts.
er = 5;
[m p] = size(A);
rr = 0;
 A = hessbrg(A);
for n = length(A):-1:2
  % QR iteration on Hessenberg
    while er > 1e-06% Error check------
          ald = zeros(1,m*(m-1)/2);
          s = 0;
         for i = 1:m
             for j = 1:p
                 if i>j
                    ald(s+1) = A(i,j);
                    s = s+1;
                 end
             end
         end
         er = max(abs(ald));%----------

         s = A(n,n);% H(n,n) is one of the eigenvalues of left end
corner 2*2 matrix
         [Q,R] = QR_GramSchmidt(A-s*eye(n));% Apply QR method on A-
sI
```

```
         A = R*Q + s*eye(n);% Estimated H
         rr = rr +1;% number of iterations
     end
  % Deflation
  d(n) = A(n,n);
  A = A(1:n-1,1:n-1);
end
d(1) = A(1,1);% final eigenvalue
end
```