

Numerical Methods, project A No.9

Name: Gaurav Chauhan

Date:1/11/2022

ID:- 309602

Introduction

This report contains the solutions of the questions presented in Numerical Methods, project A Number 9. Each question is solved using a different Matlab script file. Re-usable functions are written as saved as separate function files. There are a total of 4 main questions which are outline the following part of this report.

Problem 1

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

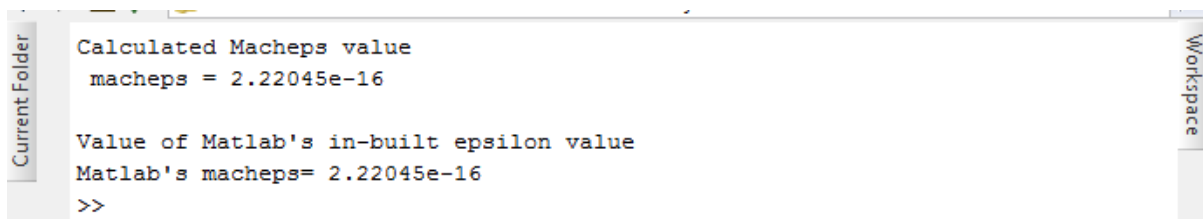
Solution to Problem 1

In digital computers, the distance between 1.0 and the next larger double-precision number is called machine epsilon. The actual machine epsilon depends on the specific machine or computer. In Matlab, the implemented algorithm starts with an initial value of 1 and reduces it by half iteratively in a loop. The current machine epsilon in each loop is compared to 1. The loop stops when the computer cannot distinguish the difference between 1 and (1+machine epsilon). Twice the last value of the loop is registered as the machine epsilon.

Matlab code

```
% This scripts implements a program to find macheps in the MATLAB
% environment on a lab computer or personal computer.
% Clear and clean the Matlab workspace
clc; clear all; close all
macheps = 1.0;
% The condition to stop is: 1+macheps <= 1,
% thus the condition to continue is 1+macheps > 1
while (1+macheps) > 1
    % Decrease macheps by a factor of 2
    macheps = macheps/2;
end
fprintf('Calculated Macheps value\n macheps = %g\n',macheps*2)
fprintf('\rValue of Matlab's in-built epsilon value\n')
fprintf('Matlab's macheps= %g\n',eps)
```

Matlab result



```
Current Folder | Calculated Macheps value  
macheps = 2.22045e-16  
Workspace  
Value of Matlab's in-built epsilon value  
Matlab's macheps= 2.22045e-16  
>>
```

The macheps determined by the implemented algorithm is equal to the macheps in-built and stored in Matlab to a variable named *eps*.

Problem 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 “ $\mathbf{A} \backslash \mathbf{b}$ ” 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 $n = 10, 20, 40, 80, 160, \dots$ until the solution time becomes prohibitive (or the method fails), for:

$$\text{a) } a_{ij} = \begin{cases} 7 & \text{for } i = j \\ -2 & \text{for } i = j-1 \text{ or } i = j+1, \\ 0 & \text{other cases} \end{cases}, \quad b_i = -3 + 0.5 i, \quad i, j = 1, \dots, n;$$

$$\text{b) } a_{ij} = 3/[7(i+j+1)], \quad b_i = 9/(7 i), i - \text{odd}; b_i = 0, i - \text{even}, \quad i, j = 1, \dots, n.$$

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.

Solution to Problem 2

A system of n linear equations is formed and solved using Gaussian elimination with partial pivoting. Since many different systems of equations need to be formed with a different number of equations, a separate function to generate the systems is created. It takes as inputs the number of equations and a choice of formulas in part ‘a’ or part ‘b’ and returns the system matrix \mathbf{A} and output vector \mathbf{b} .

Another separate function to implement Gaussian elimination with partial pivoting is created.

A call to this function returns a solution vector of the system. For each system of n linear equations solved, the error is computed as $\|\mathbf{Ax}-\mathbf{b}\|$ and plotted as a function of n .

For each case in (a) and (b), the solution vector and errors corresponding to $n = 10$ are shown.

Function to generate a system of n linear Equations

```
function [A,b] = System_AB(n,System)
% System_AB(n,Matrix) generates a system of n linear equations
%
% INPUTS: n = number of linear equations
%         System = 'a' or 'b' selects the system type
%
% OUTPUTS: A = The System Matrix of the selected system
%         b = The output vector of the selected system
%
A = zeros(n,n); % Initialize the System Matrix with zeros
b = zeros(n,1); % Initialize the System output matrix with zeros

if System~='a'&&System~='b'% Set Default to case 'a'
    fprintf('System type selected not available.System of case (a) is
used by Default\n')
    System='a';
end
%% System Case 'a'
if System=='a'
    % Apply the Given formulas
    for i=1:n
        for j=1:n
            if i==j
                A(i,j)= 7;
            elseif i==j-1 || i==j+1
                A(i,j) = -2;
            end
        end
        % Generate the b vector
        b(i) = -3+0.5*i;
    end
end
%% System Case 'b'
if System=='b'
    % Apply the given formulas
    for i=1:n
        for j=1:n
            A(i,j)=3/(7*(i+j+1));
        end
        if mod(i,2) == 1 % for i is odd
            b(i) = 9/(7*i);
        else
            b(i) = 0; % for i is even
        end
    end
end
end
end
```

Function to implement Gaussian elimination with partial pivoting

```
function xi = GaussPP(A,b)
% Numerical Methods, project A No. 9
% The indicated method: Gaussian elimination with partial pivoting
% GaussPP(A,b) find the solution of a system described by a system
matrix
% A and an output vector b.
% INPUTS: A = System matrix
%         b = Output column vector
%
% OUTPUTS x = solution of the system
%
% The System matrix must be square
if diff(size(A))~=0, error('Matrix A IS NOT square!'); end

% Form the augmented matrix using A and b
Ab=[A b];

[~,n] = size(A); % Get the Dimensions of matrix A
n1 = n+1;
% Gaussian forward elimination
for k = 1:n-1
    % Partial pivoting
    [~,Pi] = max(abs(Ab(k:n,k)))); % Pi is the index of Pivot Row
    Si = Pi+k-1; % Index used to switch rows
    if Si~=k
        % Switch the rows
        Ab([k, Si],:)=Ab([Si, k],:);
    end
    % Forward elimination
    for Pi = k+1:n
        % Calculate Multiplication Factor
        MF = Ab(Pi,k)/Ab(k,k);
        Ab(Pi,k:n1)= Ab(Pi,k:n1)-MF*Ab(k,k:n1);% Elimination
    end
end

% Apply back substitution
xi = zeros(n,1);
% Start by solving the the last value of x
xi(n) = Ab(n,n1)/Ab(n,n);
for Pi = n-1:-1:1
    xi(Pi) = (Ab(Pi,n1)-Ab(Pi,Pi+1:n)*xi(Pi+1:n))/Ab(Pi,Pi);
end
end
```

Main script for problem 2 (a)

```
% Numerical Methods, project A No. 9
% Question 2
% This Script contains a general program for solving a system of n linear
% equations  $Ax = b$  using Gaussian elimination with partial pivoting

% Clear and clean up the works space
clc;clear;close all
format shortg
format compact
% Set the value of n
n0 = 10; % Initial number of equations

%% Problem 2 case (a)
disp('Problem 2 case (a)')
System_Matrix = 'a';
% Iterate for increasing values of
% n = 10,20,40,80,160,... until the solution time becomes prohibitive
n(1) = n0; % Start at n=10
ErrorNorm_a = zeros(1,20); % Initialize the Error norm
for i=1:4%8
    % Generate the system matrix and output Vector b
    [A,b] = System_AB(n(i),System_Matrix);
    x = GaussPP(A,b);
    % Calculate the Euclidean norm of residuum  $r = Ax - b$ 
    ErrorNorm_a(i) = norm(A*x - b);
    n(i+1) = 2*n(i);
end
ErrorNorm_a = ErrorNorm_a(1:i);
% plot Error versus n.
plot(n(1:end-1),ErrorNorm_a)
grid on
xlabel('Number of equation [n]')
ylabel('Error norm')
title('Solution error vs n')
% For n = 10 print the solutions and the solutions' errors
fprintf('Solutions and the solutions' errors for n = 10\n')
[A,b] = System_AB(10,'a'); % Get matrix A and vector b
Solution_x = GaussPP(A,b); % System Solution
solutions_errors = A*Solution_x - b; % Solution Errors
T = table(Solution_x,solutions_errors)

% Making the residual correction and checking if it improves the
solutions.
fprintf('\rSolutions and the solutions' errors for n = 10 after residual
correction\n')
err = GaussPP(A,solutions_errors);
Solution_x = Solution_x - err;
solutions_errors = A*Solution_x - b;
T = table(Solution_x,solutions_errors)
```

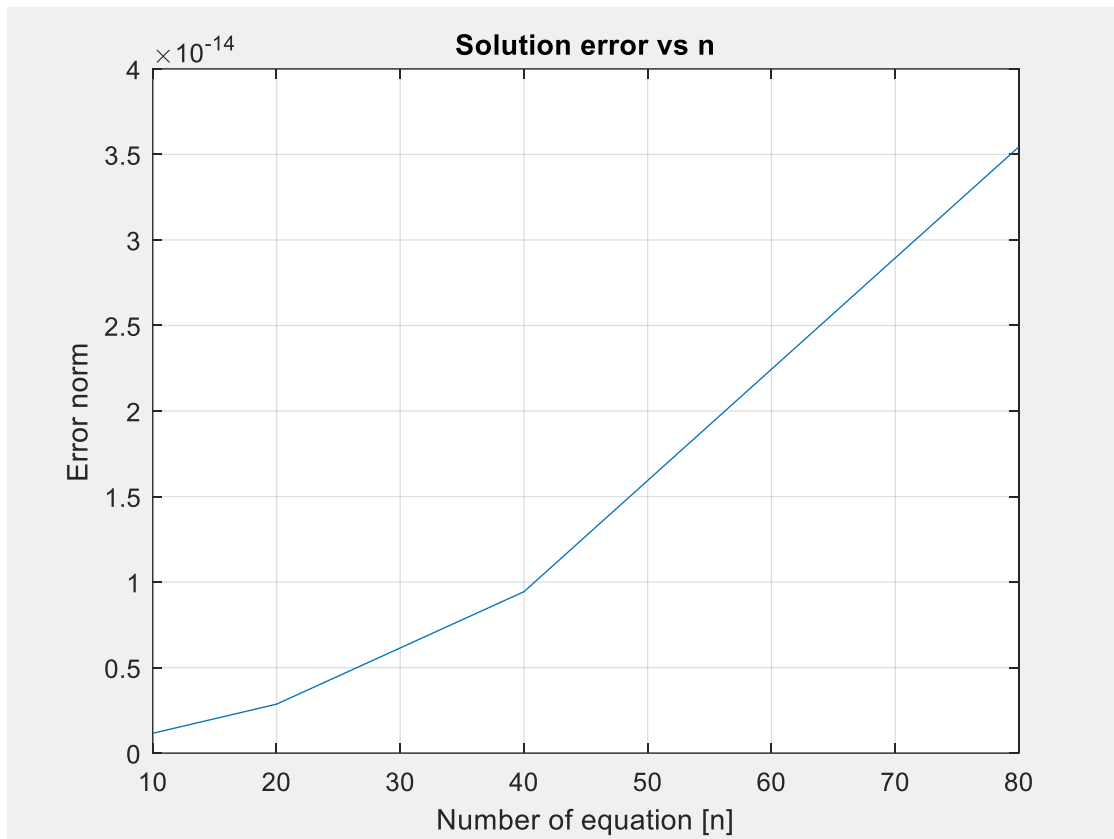
Matlab results 2(a)

```
Problem 2 case (a)
Solutions and the solutions' errors for n = 10
T =
10x2 table
    Solution_x    solutions_errors
    _____    _____
    -0.51948      -4.4409e-16
    -0.56818      -8.8818e-16
    -0.46916       4.4409e-16
    -0.32388      -2.2204e-16
    -0.16442       2.2204e-16
    -0.0015821      0
     0.15888       1.1102e-16
     0.30766       2.2204e-16
     0.41794       0
     0.40513       0

Solutions and the solutions' errors for n = 10 after residual correction
T =
10x2 table
    Solution_x    solutions_errors
    _____    _____
    -0.51948      0
    -0.56818      0
    -0.46916      0
    -0.32388      -2.2204e-16
    -0.16442      0
    -0.0015821    0
     0.15888      0
     0.30766      0
     0.41794      4.4409e-16
     0.40513      0
```

For $n = 10$, Gaussian elimination with partial pivoting solves the system with high accuracy.

This accuracy does not improve even after performing residual correction.



From the plot of errors vs n , we find that the solution error increases with an increasing number of equations. This happens because the round-off errors also increase when the number of equations increases.

Main script for problem 2 (b)

```
% Numerical Methods, project A No. 9
% Question 2
% This Script contains a general program for solving a system of n linear
% equations Ax = b using Gaussian elimination with partial pivoting

% Clear and clean up the works space
clc;clear;close all
format shortg
format compact
% Set the value of n
n0 = 10; % Initial number of equations

%% Problem 2 case (b)
disp('Problem 2 case (b)')
System_Matrix = 'b';
% Iterate for increasing values of
% n = 10,20,40,80,160,... until the solution time becomes prohibitive
n(1) = n0; % Start at n=10
ErrorNorm_a = zeros(1,20); % Initialize the Error norm
for i=1:4%8
    % Generate the system matrix and output Vector b
    [A,b] = System_AB(n(i),System_Matrix);
    x = GaussPP(A,b);
```



```

    % Calculate the Euclidean norm of residuum  $r = Ax - b$ 
    ErrorNorm_a(i) = norm(A*x - b);
    n(i+1) = 2*n(i);
end
ErrorNorm_a = ErrorNorm_a(1:i);
% plot Error versus n.
plot(n(1:end-1),ErrorNorm_a)
grid on
xlabel('Number of equation [n]')
ylabel('Error norm')
title('Solution error vs n')
% For n = 10 print the solutions and the solutions' errors
fprintf('Solutions and the solutions' errors for n = 10\n')
[A,b] = System_AB(10,System_Matrix); % Get matrix A and vector b
Solution_x = GaussPP(A,b);           % System Solution
solutions_errors = A*Solution_x - b; % Solution Errors
T = table(Solution_x,solutions_errors)

% Making the residual correction and checking if it improves the
solutions.
fprintf('\rSolutions and the solutions' errors for n = 10 after residual
correction\n')
err = GaussPP(A,solutions_errors);
Solution_x = Solution_x - err;
solutions_errors = A*Solution_x - b;
T = table(Solution_x,solutions_errors)

```

Matlab results 2(b)

```

Problem 2 case (b)
Solutions and the solutions' errors for n = 10
T =
    10×2 table
      Solution_x      solutions_errors
    _____  _____
      2.1023e+09      -0.00010027
     -7.3326e+10       6.1035e-05
      9.2892e+11     -0.00037929
     -5.9396e+12     -0.00024414
      2.1871e+13       2.8774e-05
     -4.9212e+13       3.0518e-05
      6.8774e+13     -0.00026282
     -5.8249e+13     -0.00015259
      2.7378e+13     -0.00037057
     -5.479e+12      -0.00013733

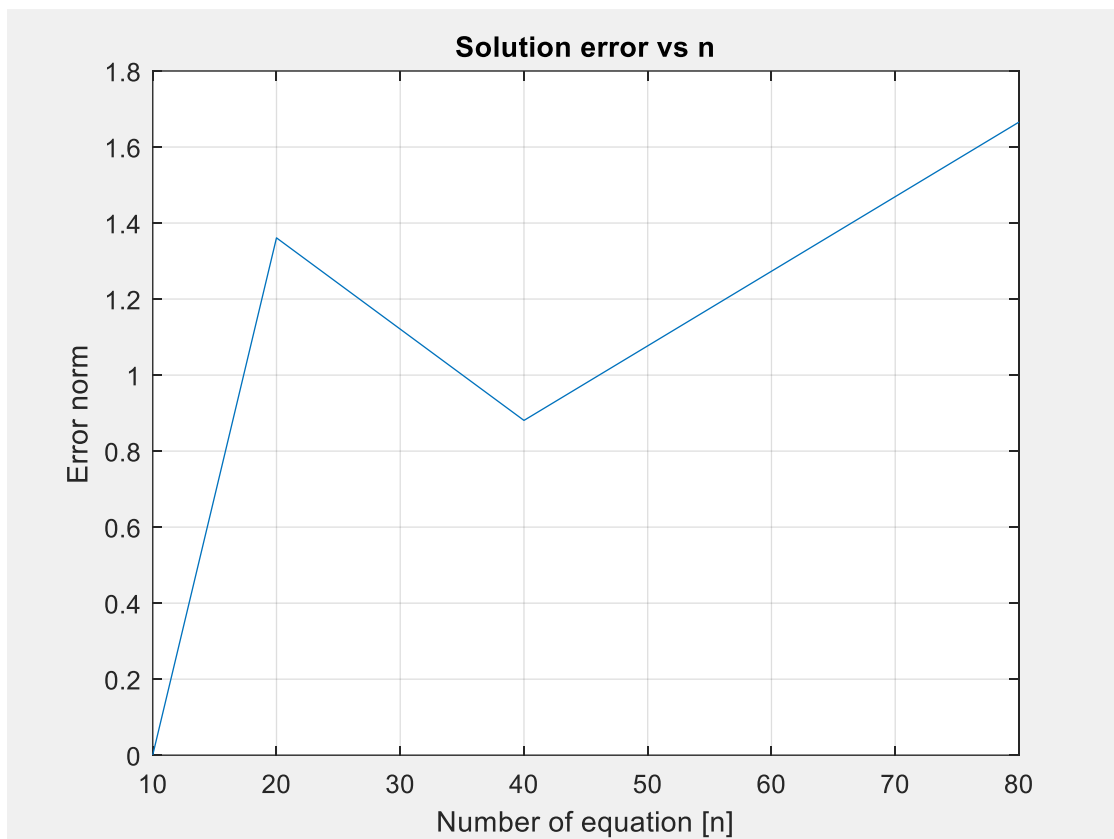
```

```

Solutions and the solutions' errors for n = 10 after residual correction
T =
10×2 table
    Solution_x    solutions_errors
    _____    _____
    2.104e+09      2.1798e-05
   -7.3384e+10     -3.0518e-05
    9.2966e+11     -0.00013515
   -5.9443e+12      0.00015259
    2.1888e+13     -0.00012381
   -4.9252e+13      6.1035e-05
    6.883e+13       5.761e-05
   -5.8297e+13      7.6294e-05
    2.74e+13        0.00031607
   -5.4835e+12      0.00015259
>>

```

Comparing the solutions errors before and after residual correction we find that this system exhibit large errors.



The solution errors increase rapidly with an increasing number of equations in the system of equations. The coefficient matrix has a very large condition number hence it is very sensitive to round-off errors.

Problem 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:

$$\begin{aligned}14x_1 - x_2 - 6x_3 + 5x_4 &= 10 \\ x_1 - 8x_2 - 4x_3 - x_4 &= 0 \\ x_1 - 4x_2 - 12x_3 - x_4 &= -10 \\ x_1 - x_2 - 8x_3 - 16x_4 &= -20\end{aligned}$$

and compare the results of iterations plotting norm of the solution error $\|\mathbf{Ax}_k - \mathbf{b}\|_2$ versus the iteration number $k=1,2,3,\dots$ 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.

Solution to Problem 3

In this question, both Gauss-Seidel and Gauss-Jacobi iterative methods are applied to solve the given system of equations. They are separate functions for each of the algorithms and which are called in the main script. The two methods are compared in terms of the error of the solution at each iteration. The two functions were also applied to solve the system of $n=10$ from problem 2.

Function implementing Gauss-Seidel Algorithm

```
function [x, Rnorms, iter_K] = Gauss_Seidel(A, b, x)
% Gauss_Seidel(A,b, x) solves a system of linear equations by applying
% Gauss Seidel Iterative method
%
% INPUTS A = System matrix/Coefficient matrix
%         b = System Output column vector
%         x = Initial solution vector (same size as vector b)
%
% OUTPUTS: x = final solution of the system
%          Rnorms = a vector of error norms
%          iter_K = a list of iterations used to achieve the solution
%
n = size(A,1);
xnew = x;
Rnorms = [];% Set a vector of Error norms with initial zeros
iter_K = [];% Variable to store the iterations
iter = 1;% Iteration counter

% Applying the Gauss_Seidel iterative algorithm
while norm(A*x - b) >= 1e-10 % tolerance set to error less than 1e-10
    for i = 1:n
        x_temp = 0;
        for j = 1:n
            if j ~= i
                % Update solution
```

```

        x_temp = x_temp + xnew(j) * A(i,j);
    end
    end
    %Calculate a new solution
    xnew(i) = 1/A(i,i)*(b(i) - x_temp);
end
x = xnew; % save the new solution
% Calculate for Error norm
Rnorms(iter) = norm(A*x - b);
% Store the current iteration value in the variable iter_K
iter_K(iter) = iter;
% Increment the Iteration Counter by 1
iter = iter+1;
% Limit Iteration to a maximum of 10000
if iter>10000,break,end
end
end

```

Function implementing Gauss Jacobi Algorithm

```

function [x, Rnorms,iter_K] = Gauss_Jacobi(A,b, x)
% Gauss_Jacobi(A,b, x) solves a system of linear equations by applying
% Gauss Jacobi Iterative method
%
% INPUTS A = System matrix/Coefficient matrix
%         b = System Output column vector
%         x = Initial solution vector (same size as vector b)
%
% OUTPUTS: x = final solution of the system
%          Rnorms = a vector of error norms
%          iter_K = a list of iterations used to achieve the solution
%
n = size(A,1);
xnew = x;
Rnorms = []; % Set a vector of Error norms with initial zeros
iter_K = []; % Variable to store the iterations
iter = 1; % Iteration counter

% Applying the Gauss_Jacobi iterative algorithm
while norm(A*x - b)>=1e-10% tolerance set to error less than 1e-10
    for i = 1:n
        x_temp = 0;
        for j = 1:n
            if j ~= i
                % Update solution
                x_temp = x_temp + x(j) * A(i,j);
            end
        end
        %Calculate a new solution
        xnew(i) = 1/A(i,i)*(b(i) - x_temp);
    end
    x = xnew; % save the new solution
    % Calculate for Error norm
    Rnorms(iter) = norm(A*x - b);
    % Store the current iteration value in the variable iter_K
    iter_K(iter) = iter;
    % Increment the Iteration Counter by 1
    iter = iter+1;
    % Limit Iteration to a maximum of 10000
end
end

```

```
        if iter>10000,break,end
    end
end
```

Matlab code main script

```
% Numerical Methods, project A No. 9
% Question 3
% This Script contains a general program for solving a system of n linear
% equations  $Ax = b$  using Gauss Seidel and Jacobi Iterative method

clc;clear; close all
%% System Description
% The System Matrix
A = [14,-1,-6,5;
     1,-8,-4,-1;
     1,-4,-12,-1;
     1,-1,-8,-16];
% The output vector
b = [10;0;-10;-20];
% Initialize the solution
x = zeros(length(b),1);
%% Solve the system using Gauss Seidel and Jacobi method
[xj, Rj,Kj] = Gauss_Jacobi(A,b, x);
[xs, Rs,Ks] = Gauss_Seidel(A, b, x);
%% Compare the results of iterations plotting norm of the solution error
figure()
hold on
plot(Kj,Rj, 'Linewidth',2)
plot(Ks,Rs, 'Linewidth',2)
grid on
xlabel('Iterations (n)')
ylabel('Error')
title('Norm of the solution error')
legend('Gauss-Jacobi','Gauss-Seidel')

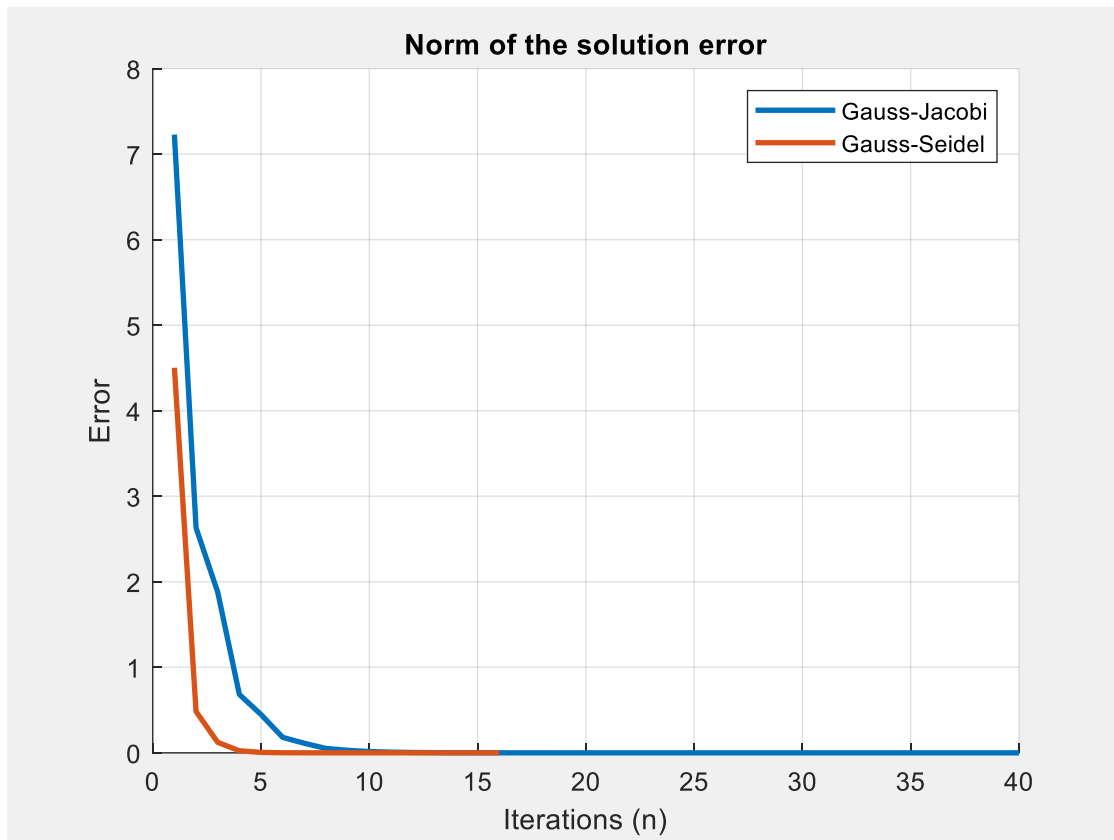
%% Solving System of Problem 2 using Gauss-Seidel and Jacobi methods
n = 10;      % Number of equations

% Solution of Problem 2(a) using Gauss_Seidel method
fprintf('\rSolution of Problem 2(a) using Gauss_Seidel method\n')
System_Matrix = 'a';
% Generate System Matrix A and Output vector b
[A,b] = System_AB(n,System_Matrix);
% Initialize the solution
x = zeros(length(b),1);
[x, Rs,Ks] = Gauss_Seidel(A, b, x);
Solution_xa = x
fprintf('Solution_error = %g \n',Rs(end))
fprintf('iterations = %g \n',Ks(end))

% Solution of Problem 2(b) using Gauss_Seidel method
fprintf('\rSolution of Problem 2(b) using Gauss_Seidel method\n')
System_Matrix = 'b';
% Generate System Matrix A and Output vector b
[A,b] = System_AB(n,System_Matrix);
% Initialize the solution
x = zeros(length(b),1);
[x, Rs,Ks] = Gauss_Seidel(A, b, x);
Solution_xb = x
fprintf('Solution_error = %g \n',Rs(end))
```

```
fprintf('iterations = %g \n',Ks(end))
```

Matlab Results



In the two methods, the solution error of the given system reduces drastically until the tolerance required is reached. We find that Gauss-Seidel converges faster than the Gauss-Jacobi. The Gauss-Seidel solves the equations sequentially using the results of the previous variable to solve the next equation. On the other side, Gauss-Jacobi uses the solutions of the previous iterations to solve all the equations of the next iteration.

```
Solution of Problem 2(a) using Gauss_Seidel method
Solution_xa =
    -0.51948
    -0.56818
    -0.46916
    -0.32388
    -0.16442
    -0.0015821
     0.15888
     0.30766
     0.41794
     0.40513
Solution_error = 3.83181e-11
iterations = 23
```

```
Solution of Problem 2(b) using Gauss_Seidel method
Solution_xb =
    3966.9
   -29963
    78282
   -87287
    59737
   -72148
    70304
   -56711
    79852
   -46228
Solution_error = 0.452434
iterations = 10000
>>
```

The system for $n=10$ in problem 2 is solved using the Gauss-Seidel method. We find that the solution of the system in case (a) converges with 59 iterations. The solution is very small. The system of the case (b) does not converge in 10000 iterations and the solution error is large. This system of the case (b) is ill-conditioned and that is why it is difficult to solve it numerically due to its sensitivity to round-off errors.

Problem 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).

Solution to Problem 4

QR decomposition method is applied to solve for eigenvalues of a 5×5 matrix. In one approach, QR without shifts is applied while in the second approach, QR with shifts calculated on the basis of an eigenvalue of the 2×2 right-lower-corner submatrix is applied. The two methods result in the same answers however require the different number of iterations to converge within the required tolerance.

Matlab code for QR without shifts

```
%This Scripts uses QR method for finding eigenvalues of 5x5 matrices
%without shifts.
clc;clear;close all

% Chosen symmetric matrix 5x5
A = [10    9    13    15    16
      9    10    9    13    15
     13    9    10    9    13
     15    13    9    10    9
     16    15    13    9    10];

[m,n] = size(A);% m = number of Rows,  n = number of columns

% Initialize various variables with zeros
Q = zeros(m,n);
R = zeros(n,n);
L = tril(A,-1);
Lv = L(:); B = Lv(Lv~=0);
Z = zeros(n,1);
r = 0; % Initialize iterations counter
tol = 1e-6; % Threshold Tolerance
while any(B > tol)
    % Initialization and Calculation for Q
    for i = 1:m
        Q(i,1) = A(i,1)/norm(A(:,1));
    end
    % Apply Gram-Schmidt algorithm
    for i = 2:n
        S = zeros(m,1);
        for k = 1:i-1
            S = S + (A(:,i)'*Q(:,k))*Q(:,k);
        end
    end
end
```



```

        Z = A(:,i) - S;
        for j = 1:m
            Q(j,i) = Z(j)/norm(Z);
        end
    end
    % Calculation for R
    for i = 1:m
        for j = i:n
            R(i,j) = Q(:,i)'*A(:,j);
        end
    end
    A = R*Q;% Set A=RQ
    % Check if threshold is achieved for all values
    L = tril(A,-1);Lv = L(:);
    B = abs(Lv(Lv~=0));
    % Increment the iteration
    r =r+1;
end
fprintf('Eigenvalues using QR method without shifts\n')
Eigenvalues = diag(A)
fprintf('Iterations = %g \n',r)
fprintf('\rEigenvalues obtained using built-in matlab function eig()\n')
Matlab_Eigenvalues = eig(A)

```

Matlab code for QR with shifts

```

%This Scripts uses QR method for finding eigenvalues of 5x5 matrices
%with shifts calculated on the basis of an eigenvalue of the 2x2
% right-lower-corner submatrix.
clc;clear;close all
% Chosen symmetric matrix 5x5
A = [10    9    13    15    16
      9    10    9    13    15
     13    9    10    9    13
     15    13    9    10    9
     16    15    13    9    10];
[m,n] = size(A);% m = number of Rows,  n = number of columns

% Initialize various variables with zeros
Q = zeros(m,n);
R = zeros(n,n);
L = tril(A,-1);
Lv = L(:); B = Lv(Lv~=0);
Z = zeros(n,1);
r = 0; % Initialize iterations counter
tol = 1e-6; % Threshold Tolerance

% Apply the algorithm for QR with shifts
while any(B > tol)
    A22 = A(end-1:end,end-1:end);% Extract right-lower-corner submatrix
    u = max(roots([1 -trace(A22) det(A22)]));% u = eigenvalue of A22
    A = A-u*eye(n); % Determine A - uI = QR
    for i = 1:m
        Q(i,1) = A(i,1) / norm(A(:,1)); % Initial matrix Q
    end
    % Apply Gram-Schmidt procedure to get Q
    for i = 2:n

```

```

        S = zeros(m,1);
        for k = 1:i-1
            S = S + (A(:,i)'*Q(:,k))*Q(:,k);
        end
        Z = A(:,i) - S;
        for j = 1:m
            Q(j,i) = Z(j)/norm(Z);
        end
    end
    % Calculating R
    for i = 1:m
        for j = i:n
            R(i,j) = Q(:,i)'*A(:,j);
        end
    end
    A = R*Q+u*eye(n); % A = RQ + uI
    % Check if threshold is achieved for all values
    L = tril(A,-1);Lv = L(:);
    B = abs(Lv(Lv~=0));
    % Increment the iteration
    r =r+1;
end
fprintf('Eigenvalues using QR method with shifts\n')
Eigenvalues = diag(A)
fprintf('Iterations = %g \n',r)
fprintf('\rEigenvalues obtained using built-in matlab function eig()\n')
Matlab_Eigenvalues = eig(A)

```

Matlab results

```

Eigenvalues using QR method without shifts
Eigenvalues =
    58.64
   -10.685
    3.203
   -2.8431
    1.6847
Iterations = 128

Eigenvalues obtained using built-in matlab function eig()
Matlab_Eigenvalues =
    58.64
   -10.685
    3.203
   -2.8431
    1.6847

```

```
Eigenvalues using QR method with shifts
Eigenvalues =
    58.64
   -10.685
   -2.8431
    1.6847
    11.037
Iterations = 32

Eigenvalues obtained using built-in matlab function eig()
Matlab_Eigenvalues =
    58.64
   -10.685
   -2.8431
    1.6847
    11.037
```

Using QR without shifts to solve the eigenvalues requires a higher number of iterations to converge to the solution when compared to QR with shifts. The use of shifts lowers the error by a large margin in each iteration hence overall reducing the total number of iterations required to converge.