ECSE 343: Numerical Methods for Engineers- Assignement 2

Due Date: 9th March 2021

Student Name: Hongtao Xu

Student ID:

Please type your answers and write you code in this .mlx script. If you choose to provide the handwritten answers, please scan your answers and include those in SINGLE pdf file.

Please submit this **.mlx** file along with the **PDF** copy of this file.

Question 1:

a) Prove that $\kappa(A^T A) = \kappa(A)2$.

Ax = 0

A^T Ax = 0, so X belongs to N(A^T A)

hence, N(A) belongs to N(A^T A); also x belongs to N(A^T A)

so A^T Ax = 0

$x^T A^T$ Ax = 0 -> $(\mathrm{Ax})^T$ Ax =0

Ax = 0, so x belongs to N(A)

because N(A^T A) belongs to N(A)

so N(A^T A) = N(A) and thier dimensions are the sane

finally $\kappa(A^T A) = \kappa(A)2$

b) Given two orthonormal matrices $U$and Q, show that product of these matrices, UQ, is also orthonormal.

UU^T = U^T U = I

QQ^T = Q^T Q = I

so $(\mathrm{UQ})^T$ (UQ) = $Q^T U^T$ UQ = I

in this way, it is orthonormal

c) If A is an invertible matrix and Q is an orthonormal matrix, show that $\kappa(\mathrm{QA}) = \kappa(A)$.

Hint: $\kappa(A) = \frac{\sigma 1}{\sigma n}$ (the ratio of the largest and smallest eigen values).

A is a intvertiable matrix and ! is an orthonormal matrix, in this way, det(A) n != 0 also det(A) is the product of eigenvalues

we know that eigenvalues of A are non-zero

QQ^T = Q^T Q = I

det (QQ^T)=det(Q^T) = 1

(detQ)^2 = 1 so that det(Q) = +1/-1

det(QA) = det(Q)(detA) = (+/- 1)*(product of eigenvalues of A)

K(QA) = $\frac{\sigma 1'}{\sigma n'}$ =$\frac{\sigma 1}{\sigma n}$ =K(A)

Question 2:

(a) **Cholesky factorization** is a popular matrix factorization algorithm analogous to LU decompostion for positive definite matrices. In this case a matrix $\boldsymbol{M} \in \Re^{N \times N}$ is factored as

$$\boldsymbol{M} = \boldsymbol{L}\,\boldsymbol{L}^T$$

Where $\boldsymbol{L} \in \Re^{N \times N}$ is a lower triangular matrix with real and positive diagonal enteries.

Note that there are a number of tests/definitions for positive definite matrices. A matrix, $\boldsymbol{M}$is positive definite if and only if it satisfies the following two conditions,

1. The matrix $\boldsymbol{M}$must be sysmterical, i.e. $\boldsymbol{M} = \boldsymbol{M}^{\boldsymbol{T}}$
2. $\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{M}\,\boldsymbol{x} > 0$ for all $\boldsymbol{x} \in \Re^{N \times 1}$; $\|\boldsymbol{x}\| \neq 0$.

An alternative definition/test is that a matrix $\boldsymbol{M} \in \Re^{N \times N}$is positive definite if and only if it can be written as

$$\boldsymbol{M} = \boldsymbol{A}\boldsymbol{A}^{\boldsymbol{T}}$$

It can be shown (similarly to how we derived the Doolittle algorithm) that the entries of matrix L are given by the following expression

$$\mathbf{L}_{i,j} = \begin{cases} \sqrt{\mathbf{M}_{j,j} - \sum_{k=1}^{j-1} \mathbf{L}_{j,k}^2}, & \text{if } i = j \\ \frac{1}{\mathbf{L}_{j,j}} \left(\mathbf{M}_{i,j} - \sum_{k=1}^{j-1} \mathbf{L}_{i,k} \mathbf{L}_{j,k} \right), & \text{if } i > j \\ 0, & \text{otherwise} \end{cases}$$

The cost of Cholesky factorization algorithm is rougly half than the LU decompostion. Your task is to implement the Cholesky decomposition alogorithm in the cell below. Note that for Cholesky decomposition pivoting is not required

Use the cell below to test the Cholesky decompostion code. To test this, compute the Frobenius norm of $(L\,L^T - M)$.

```
A = randi(100,5);
M =A'*A;
L = CholeskyDecomposition(M);
Error =  norm(L*L'-M);
```

b) Cholesky decompostion can be used to solve the normal eqautions, $A^T A\,x = A^T\,b$ as the matrix $(A^T A)$ is positive definite. Using Cholesky facotrization scheme $(A^T A)$ can be factored as $A^T A = L\,L^T$. Using this the normal equations become,

$$L\,L^T\,x = A^T\,b$$

The above equation consists of the triangular systems, the solution **x** can be obtained by first solving $L\,y = A^T\,b$ using the forward substitution, then by solving the $L^T\,x = y$ using backward subsitution. Implement the Cholesky solver for least squares in the function below.

You can use your Forward/Backward Substitution from Assignment 1 for bonus marks. Include your code for forward/backward substitution in the Appendix.

```
% in the end
```

d) To test the Cholesky Least Squares solver we numerical algorithm we will use a square matrix. Run the cell below to compare the solution.

```
A = [12    10     6     5    46;
     79    14    24    17    55;
     39    95    36    65    30;
     25    96    83    74    75;
     41    58     2    65    19];

B = [45; 35; 58; 12; 10];
x = LeastSquareSolver_Cholesky(A,B);
```

```
x = 5×1
    0.4453
    0.3109
    1.2031
    0.6448
    0.2666
```

```
% verify your solution
R = chol(A'*A,'lower');
Error2 = norm(R*R'-M);
X = R'\(R\(A'*B));
```

Question 3:

Polynomial fitting is one of the many applications of Least-Squares Approximation. Given the experimental data, shown in Table below,

$$\begin{array}{l|l} Input, , x & x0\ x1\ x2\ x3\ \ldots\ xm-1\ xm \\ Output, y & y0\ y1\ y2\ y3\ \ldots\ ym-1\ ym \end{array}$$

we can fit a n^{th}degree polynomial, $p(x) = a0 + a1x + a2\,x^2 + a3x^3 + \cdots + a_n\,x^n$. To find the coefficicients, $\boldsymbol{a} = [a0,\ a1,\ a2, \cdots an]$, we can solve the linear system shown below.

$$\underbrace{\begin{bmatrix} 1 & x0 & x0^2 & \cdots & x0^n \\ 1 & x1 & x1^2 & \cdots & x1^n \\ 1 & x2 & x2^2 & \cdots & x2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & xm-1 & xm-1^2 & \cdots & xm-1^n \\ 1 & xm & xm^2 & \cdots & xm^n \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} a0 \\ a1 \\ a2 \\ \vdots \\ an \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} y0 \\ y1 \\ y2 \\ \vdots \\ ym-1 \\ ym \end{bmatrix}}_{y}$$

Where M is a Vandermonde matrix (known to be ill-conditioned). The system $M\,a = y$ can then be solved to compute the coefficients in vector a.

a) Your first task is to write a function that takes the input data vector x and degree of the polynomial, then returns the Vadermonde matrix M. Complete the function in the cell below.

b) The linear system can be solved using different approaches. In the code below, use the Cholesky decompositon and QR decomposition function to find the polynomial coefficients for the 1st degree polynomial. Explain the results obtained in figure 1.

```
% Load the Input Data from the provided .mat file
load('Polynomial_Fitting_Data.mat');
% set the degree
deg = 1
```

```
deg = 1
```

```
% Define powers
powers = 0:deg;
x = x_given;
y = y_given;
% get Polynomial Matrix
Mp = PolynomialMatrix(x,deg);


   % MATLAB QR Decomposition
   [Q,R] = qr(Mp);

   % Find the polynomial coefficients using L, and Q & R matrices obtained above.

   % Name the Coefficeints obtained Cholesky  Ac
        % Use the MATLAB's Forward Slash Algorithm to obtain Ac
        % by solving the normal equations (like we did in Tutorial)
            % Normal Eqautions: (Mp'*Mp)*Ac = Mp'*y
            % write your code here
            Ac = (Mp'*Mp)\(Mp' *y);

   % Name the Coefficeints obtained QR as Ar

            % write your code here
            Ar = (Q*R)\y;

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   x_new = 0:0.01:7;
   x_new = x_new';

   fr = zeros(length(x_new),1);
   fc = zeros(length(x_new),1);



   for I = 1:length(powers)
       fr = fr + Ar(I)*x_new.^powers(I);
       fc = fc + Ac(I)*x_new.^powers(I);
   end
   %plots
   figure(1)
```

```
    hold on
    plot(x,y,'ro')
    plot(x_new,fc,'m--','Linewidth',2)
    plot(x_new,fr,'r-.','Linewidth',2)
    xlabel('x')
    ylabel('y')
    legend(' Given Data','Cholesky','MATLAB QR')
    grid on
```

c) Now use the cell below to use the Cholesky decompositon and QR decomposition function to find the polynomial coefficients for the 14th degree polynomial. **Explain the results obtained in figure 2.**

```
% use the code from Part b as a starting point
% set the degree
deg = 14;
% Define powers
powers = 0:deg;
x = x_given;
y = y_given;
% get Polynomial Matrix
Mp = PolynomialMatrix(x,deg);


    % MATLAB QR Decomposition
    [Q,R] = qr(Mp);

    % Find the polynomial coefficients using L, and Q & R matrices obtained obove.

    % Name the Coefficeints obtained Cholesky  Ac
         % Use the MATLAB's Forward Slash Algorithm to obtain Ac
         % by solving the normal equations (like we did in Tutorial)
             % Normal Eqautions: (Mp'*Mp)*Ac = Mp'*y
             % write your code here
             Ac = (Mp'*Mp)\(Mp' *y);
```

```
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.397461e-29.
```

```

    % Name the Coefficeints obtained QR as Ar

             % write your code here
             Ar = (Q*R)\y;
```

```
Warning: Rank deficient, rank = 12, tol =  1.485972e-01.
```

```

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    x_new = 0:0.01:7;
    x_new = x_new';

    fr = zeros(length(x_new),1);
    fc = zeros(length(x_new),1);


    for I = 1:length(powers)
        fr = fr + Ar(I)*x_new.^powers(I);
        fc = fc + Ac(I)*x_new.^powers(I);
    end
    %plots
    figure(1)
    hold on
    plot(x,y,'ro')
    plot(x_new,fc,'m--','Linewidth',2)
    plot(x_new,fr,'r-.','Linewidth',2)
    xlabel('x')
    ylabel('y')
    legend(' Given Data','Cholesky','MATLAB QR')
    grid on
```

d) In this part we will use limited amount of data, we will use only **15 data points**. Use Cholesky decompositon and QR decomposition function to find the polynomial coefficients for the 14th degree polynomial. **Comment on the results obtained in figure 3, which algorithm performs better and why?**

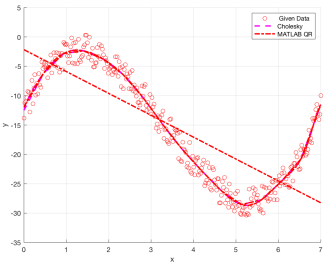
```
% Load the Input Data from the provided .mat file
load('Polynomial_Fitting_Data.mat');
% set the degree
deg =  14;
% Define powers
powers = 0:deg;
x = x_given(1:20:end);
y = y_given(1:20:end);
% get Polynomial Matrix
Mp14 = PolynomialMatrix(x,deg);


   % MATLAB QR Decomposition
   [Q,R] = qr(Mp14);

   % Find the polynomial coefficients using L, and Q & R matrices obtained obove.

   % Name the Coefficeints obtained Cholesky  Ac
        % Use the MATLAB's Forward Slash Algorithm to obtain Ac
        % by solving the normal equations (like we did in Tutorial)
            % Normal Eqautions: (Mp'*Mp)*Ac = Mp'*y
            % write your code here
            Ac14 = (Mp14'*Mp14)\(Mp14' *y);
```

```
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.471870e-30.
```

```
   % Name the Coefficeints obtained QR as Ar

            % write your code here
             Ar14 = (Q*R)\y;
```

```
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.968636e-17.
```

```
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   x_new = 0:7/15:7;
   x_new = x_new';

   fr = zeros(length(x_new),1);
   fc = zeros(length(x_new),1);


   for I = 1:length(powers)
       fr = fr + Ar(I)*x_new.^powers(I);
       fc = fc + Ac(I)*x_new.^powers(I);
   end
   %plots
   figure(1)
   hold on
   plot(x,y,'ro')
   plot(x_new,fc,'m--','Linewidth',2)
   plot(x_new,fr,'r-.','Linewidth',2)
   xlabel('x')
   ylabel('y')
   legend(' Given Data','Cholesky','MATLAB QR')
   grid on
```



Question 4:

In this question we will develop the different algorithms to implment QR factorization.

a) Use the cell below to implement a function that computes Gram-Schmidt based QR decompositon of a input matrix A.

Use the cell blow to test the your method by comparing the computing the $\|A - Q\,R\|F$. Use the Frobenius norm in the MATLAB, comment on your result.

```
A = randi(100,6); % A is a 6x6 matrix of random numbers
[Q,R] = gschmidt(A);

norm(A-Q*R,'fro');
```

b) Use the cell below to implement a function that uses the Modiefied Gram-Schmidt approach to compute the QR decompositon of the input matrix A.

Use the cell blow to test the your method by comparing the computing the $\|A - Q\,R\|F$. Use the Frobenius norm in the MATLAB, comment on your result.

```
 A = randi(100,6); % A is a 6x6 matrix of random numbers
[Q,R] = gschmidt(A);

norm(A-Q*R,'fro');
```

c) Use the cell below to implement a function that uses the Householder approach to compute the QR decompositon of the input matrix A.

Use the cell blow to test the your method by comparing the computing the $\|A - Q\,R\|F$. Use the Frobenius norm in the MATLAB, comment on your result.

```
A = randi(100,6); % A is a 6x6 matrix of random numbers
[Q,R] = householder(A);

norm(A-Q*R,'fro');
```

d) We know that Q matrix obtained using QR decompostion should have orthonormal columns i.e. $Q^TQ = I$. However, due to numerical errors, In this section we will test the functions written above to compare the error between the above implemeneted approaches. In oder to do this we will use a $n \times n$ HIlbert Matrix, the entries of Hibert Matrix are given by

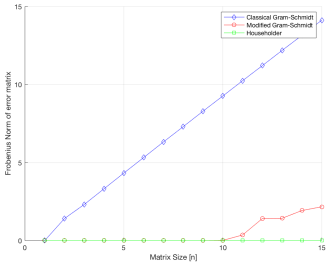
$$hi,j = \frac{1}{i + j - 1}$$

In the cell below we compute the QR decomposition of Hilbert matrices of size ranging from 2 to 16, then we compute the norm of error $(Q^TQ - I)$ to assess the performance of the lorithms written in part (a), (b), and (c). Comment on the results obtained.

```
for n=2:16
a= hilb(n); % n is the size of the matrix
[q_gs,r1] = gschmidt(a); %using algorithm classical Gram--Schmidt
[q_mgs,r2] = mgschmidt(a); %using algorithm modified Gram-Schmidt
[q_hh,r3] = householder(a); % using house holder

err_gs(n-1) = norm(q_gs'*q_gs - eye(n),'fro');
err_mgs(n-1) = norm(q_mgs'*q_mgs - eye(n),'fro');
err_hh(n-1) = norm(q_hh'*q_hh - eye(n),'fro');
end
```

```
    0.2000    0.0915   -0.0020   -0.0005    0.0000    0.0000   -0.0000    0.0000   -0.0000    0.
    0.1667    0.0739    0.0000   -0.0005   -0.0000    0.0000   -0.0000   -0.0000   -0.0000   -0.
    0.1489    0.0638    0.0019   -0.0002   -0.0000   -0.0000    0.0000   -0.0000   -0.0000    0.
    0.1359    0.0561    0.0027   -0.0002   -0.0000   -0.0000    0.0000    0.0000   -0.0000   -0.
    0.1250    0.0500    0.0034   -0.0001   -0.0000   -0.0000    0.0000    0.0000   -0.0000   -0.
    0.1151    0.0445    0.0038    0.0001   -0.0000   -0.0000   -0.0000    0.0000    0.0000   -0.
    0.1000    0.0315    0.0035    0.0001   -0.0000   -0.0000   -0.0000    0.0000    0.0000    0.
```

```
figure(4)
hold on
grid on
plot(err_gs,'b-d')
plot(err_mgs,'r-o')
plot(err_hh,'g-s')
ylabel('Frobenius Norm of error matrix');
xlabel('Matrix Size [n]')
legend('Classical Gram-Schmidt','Modified Gram-Schmidt','Householder');
```



e) Finally, now we have implemented various different versions of QR decompositon. Use above versions of the the QR transform and use this to fit the polynomial in question 3. We want to fit the polynomial of degree 14 through the data with 15 points. Clearly show all the plots (with legends) and comment on the results obtained.

```
% Implement Question 4 part e here.Use the  code privided in Question 3 part (d) for inspitration.
load('Polynomial_Fitting_Data.mat');
% set the degree
deg =  14;
% Define powers
powers = 0:deg;
x = x_given(1:20:end);
y = y_given(1:20:end);
% get Polynomial Matrix
Mp = PolynomialMatrix(x,deg);


   % MATLAB QR Decomposition
   [q_gs,r1] = gschmidt(Mp); %using algorithm classical Gram--Schmidt
   [q_mgs,r2] = mgschmidt(Mp); %using algorithm modified Gram-Schmidt
```

```
v =
    0.0000   -0.0000         0         0         0         0         0         0         0
    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.
    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.
    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.
    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.
    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.
v = 0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.
    0.0000   -0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0001    0.
    0.0000   -0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0001    0.0004    0.
    0.0000   -0.0000   -0.0000    0.0000    0.0000    0.0000    0.0001    0.0002    0.0010    0.
    0.0000   -0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.
    0.0000   -0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.
    0.0000   -0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.
v = 0.0000   -0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.
    0.0000         0   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0001    0.
    0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0001    0.0004    0.
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0001    0.0002    0.0010    0.
```

```
    0.0000   -0.0000    0.0000   -0.0000    0.0000   -0.0000    0.0000   -0.0000    0.0000   -0.
    0.0000   -0.0000    0.0000   -0.0000   -0.0000    0.0000   -0.0000    0.0000   -0.0000    0.
    0.0000   -0.0000    0.0000    0.0000   -0.0000    0.0000   -0.0000   -0.0000    0.0000   -0.
v = 0.0000   -0.0000   -0.0000    0.0000   -0.0000    0.0000    0.0000   -0.0000    0.0000    0.
[illegible]
    0.0000   -0.0000   -0.0000    0.0000    0.0000   -0.0000    0.0000    0.0000   -0.0001   -0.
v = 0.0000   -0.0000   -0.0000    0.0000    0.0000   -0.0000   -0.0000    0.0000    0.0000   -0.
    0.0000   -0.0000   -0.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0000    0.0000   -0.
    0.0000   -0.0000   -0.0000   -0.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0000    0.
    0.0000   -0.0000   -0.0000   -0.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0001   -0.
    0.0000   -0.0000   -0.0000    0.0000   -0.0000    0.0000    0.0000   -0.0001    0.0000    0.
    0.0000   -0.0000   -0.0000    0.0000   -0.0000   -0.0000    0.0000   -0.0000   -0.0001    0.
    0.0000   -0.0000   -0.0000    0.0000    0.0000   -0.0000    0.0000    0.0000   -0.0001   -0.
v = 0.0000   -0.0000   -0.0000    0.0000    0.0000   -0.0000   -0.0000    0.0000    0.0000   -0.
    0.0000   -0.0000   -0.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0000    0.0000   -0.
    0.0000   -0.0000   -0.0000   -0.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0000    0.
    0.0000   -0.0000   -0.0000   -0.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0001   -0.
    0.0000   -0.0000   -0.0000    0.0000   -0.0000    0.0000    0.0000   -0.0001    0.0000    0.
    0.0000   -0.0000   -0.0000    0.0000   -0.0000   -0.0000    0.0000   -0.0000   -0.0001    0.
    0.0000   -0.0000   -0.0000    0.0000    0.0000   -0.0000    0.0000    0.0000   -0.0001   -0.
v = 0.0000   -0.0000   -0.0000    0.0000    0.0000   -0.0000   -0.0000    0.0000    0.0000   -0.
[illegible]
    0.0000   -0.0000   -0.0000    0.0001   -0.0001    0.0000    0.0003   -0.0006    0.0004    0.
    0.0000   -0.0000   -0.0000    0.0001   -0.0000   -0.0002    0.0003   -0.0001   -0.0007    0.
    0.0000   -0.0000   -0.0000    0.0001    0.0000   -0.0002    0.0001    0.0005   -0.0007   -0.
v = 0.0000   -0.0000   -0.0000    0.0000    0.0001   -0.0001   -0.0002    0.0005    0.0003   -0.
[illegible]
    0.0000   -0.0000   -0.0000    0.0001   -0.0001    0.0000    0.0003   -0.0006    0.0004    0.
    0.0000   -0.0000   -0.0000    0.0001   -0.0000   -0.0002    0.0003   -0.0001   -0.0007    0.
    0.0000   -0.0000   -0.0000    0.0001    0.0000   -0.0002    0.0001    0.0005   -0.0007   -0.
v = 0.0000   -0.0000   -0.0000    0.0000    0.0001   -0.0001   -0.0002    0.0005    0.0003   -0.
[illegible]
    0.0001   -0.0002   -0.0001    0.0007   -0.0012    0.0001    0.0030   -0.0059    0.0037    0.
    0.0001   -0.0001   -0.0002    0.0008   -0.0004   -0.0016    0.0034   -0.0006   -0.0074    0.
    0.0001   -0.0001   -0.0003    0.0006    0.0004   -0.0021    0.0009    0.0048   -0.0066   -0.
v = 0.0001   -0.0000   -0.0004    0.0003    0.0010   -0.0014   -0.0022    0.0048    0.0030   -0.
[illegible]
    1.0000   -1.8729   -0.5846    7.1446  -11.6015    0.9431   30.3482  -59.2989   37.4982   61.
    1.0000   -1.4047   -2.1193    7.5142   -4.1034  -16.0966   33.9693   -5.7386  -74.2799  117.
    1.0000   -0.9365   -3.2155    6.0360    4.1363  -21.4336    8.6217   47.8217  -65.6815  -44.
```

```
[q_hh,r3] = householder(Mp);
size(inv(q_gs*r1));
```

Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 1.968638e-17.

```
size(y);
A1 = (q_gs*r1)\y;
```

Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 1.968638e-17.

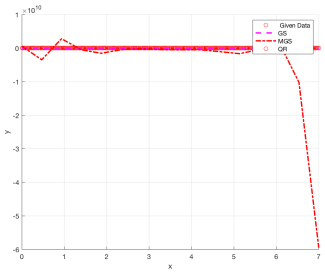
```
A2 = (q_gs*r2)\y;
```

Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 6.045334e-25.

```
A3 = (q_gs*r2)\y;
```

Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 6.045334e-25.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x_new = 0:7/15:7;
x_new = x_new';

fa = zeros(length(x_new),1);
fb = zeros(length(x_new),1);
fc = zeros(length(x_new),1);


for I = 1:length(powers)
    fa = fa + A1(I)*x_new.^powers(I);
    fb = fb + A2(I)*x_new.^powers(I);
    fc = fc + A3(I)*x_new.^powers(I);
end
%plots
figure(1)
hold on
plot(x,y,'ro')
plot(x_new,fa,'m--','Linewidth',2)
plot(x_new,fb,'r-.','Linewidth',2)
plot(x_new,fc,'r-.','Linewidth',2)
xlabel('x')
ylabel('y')
legend(' Given Data','GS','MGS', 'QR')
grid on
```



Appendix 1

```
function A = Hilbert_matrix(size)

A = zeros(size,size);

for i=1:size
    for j=1:size
        A(i,j) = 1/(i+j-1);
    end
end
end
% question 2 a
function L = CholeskyDecomposition(M);
if(size(M,1) ~= size(M,2))
    disp('Matrix M must be square!');
end
L = zeros(size(M));
m = size(M,1);
for i=1:m
  for j=1:m
    if i==j && j>=2
      for k=1:j-1
        M(j,j)=M(j,j)-L(j,k)*L(j,k);
        end
        L(i,j)=sqrt(M(j,j));
    elseif i==j
        L(i,j)=sqrt(M(j,j));
    elseif i>j && j>=2
        for k=1:j-1
            M(i,j)=M(i,j)-L(i,k)*L(j,k);
        end
```

```
        L(i,j)=M(i,j)/L(j,j);
    elseif i>j
      L(i,j)=1/L(j,j)*M(i,j);
    elseif i<j
      L(i,j)=0;
    end
  end
end

end

% question 2 b
function x = LeastSquareSolver_Cholesky(A,b)
M = A'*A;
L = CholeskyDecomposition(M);
y = forward_sub(L,b);
x = backward_sub(L,b);
end

function y = forward_sub(L,b)
l = length(b);
y = zeros(l,1);
y(1,1) = b(1)./L(1,1);
for j = 2:l
    y(j,1) = (b(j)-sum(L(j,1:j-1)*y(1:j-1,1)))./L(j,j);
end
end

function x = backward_sub(L,b)
l = length(b);
x = zeros(1,l);
x(1,l)=b(end)./L(l,l);
for i = l-1:-1:1
    temp = 1/L(i,i).*(b(i)-sum(L(i,i+1:end).*x(i+1:end)));
    x(1,i) = temp;
end
x=x'
end

%question 3a
function M = PolynomialMatrix(x,n)
powers = 0:n;
for i = 1:length(powers)
    M(:,i) = x.^powers(i);
end
end

%question 4a
function [Q,R] = gschmidt(A)
    [n,p] = size(A);
    Q = zeros(n,p);
    R = zeros(p,p);
    for i = 1:p
        Q(:,i) = A(:,i);
        if i ~= 1
            R(1:i-1,i) = Q(:,i-1)'*Q(:,i);
            Q(:,i) = Q(:,i) - Q(:,1:i-1)*R(1:i-1,i);
        end
        R(i,i) = norm(Q(:,i));
        Q(:,i) = Q(:,i)/R(i,i);
     end
end

 %question 4b
 function [Q,R] = mgschmidt(A)
   [n,p] = size(A);
    Q = zeros(n,p);
    R = zeros(p);
    v = zeros(p);
    for i = 1:n
```

```
        v(i,:) = A(i,:);
    end
        for i = 1:n
            R(i,i) = normest(v(:,i));
            Q(:,i) = v(:,i)/R(i,i);
       for j = (i+1):p
        R(i,j) = Q(:,i)'*v(:,j);
        v(:,j) = v(:,j)-R(i,j)*Q(:,i)
       end
        end
 end
% question 4c
    function [Q,R] = householder(A)
    [n,p] = size(A);
    Q = eye(n,n);
    for j = 1:p
        z = A(j:end,j);
        v = [-sign(z(1))*normest(z)-z(1);-z(2:end)];
        Qt = eye(size(z,1),size(z,1))-(2/(v'*v))*(v*v');
        Qtt = eye(n,n);
        Qtt(j:end,j:end) = Qt;
        Q = Q*Qtt;
        A = Qtt*A;
    end
    R = A;
    end
```