ECSE 343: Numerical Methods for Engineers- Assignement 2

Due Date: 11 November 2021

Student Name:Lawi Mwirigi

Student ID:260831614

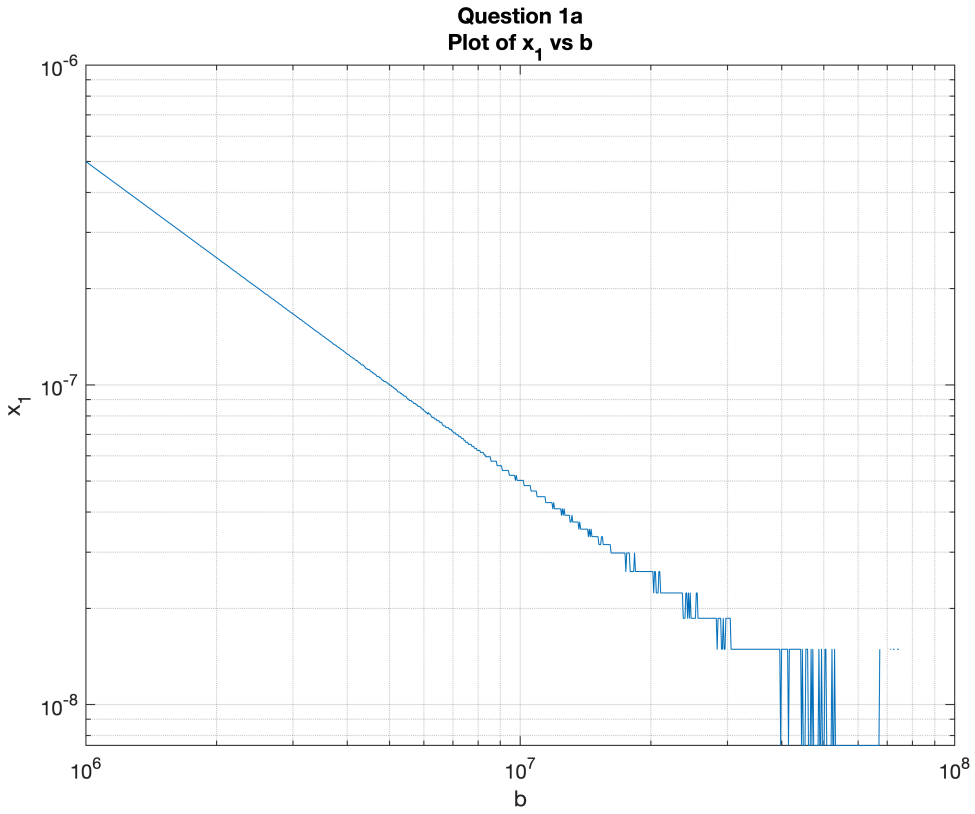
Please type your answers and write you code in this .mlx script. If you choose to provide the handwritten answers, please scan your answers and include those in SINGLE pdf file.

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

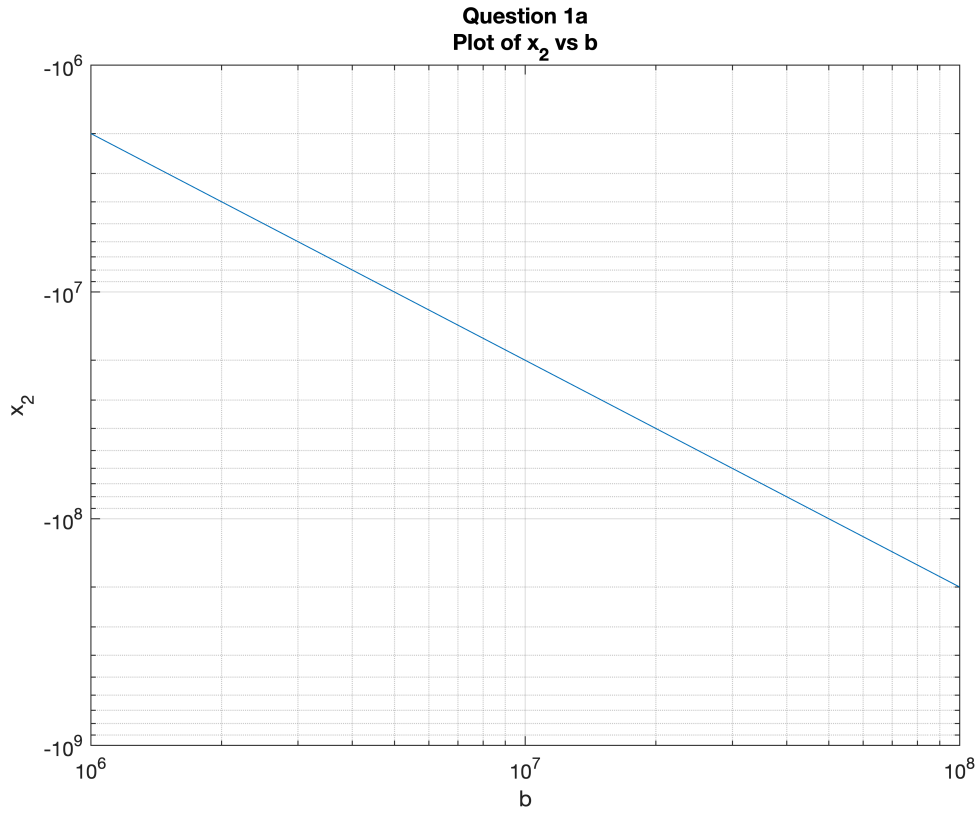
Question 1:

a) Given the equation $x^2 + 2bx - 1 = 0$ where b is a parameter chosen such that $b > 0$. Using the quadratic formula the roots x_1 and x_2 of this equation are given by $x = -b \pm \sqrt{b^2 + 1}$. The following MATLAB program computes and plots the roots x_1 and x_2 as a function of b for 1000 logarithmically spaced values of b between 10^6 and 10^8. Run the code and explain the reason why there are discontinuities.

```
b = logspace(6,8,1000);

x1 = -b+sqrt(b.^2 +1);
x2 = -b-sqrt(b.^2 +1);



%plot x1 on a loglog scale

figure
loglog(b,x1)
title({'Question 1a';'Plot of x_1 vs b'})
xlabel('b')
ylabel('x_1')
grid on
```




```
%plot x2 on a loglog scale

figure
loglog(b,x2)
xlabel('b')
ylabel('x_2')
title({'Question 1a';'Plot of x_2 vs b'})
grid on
```




```
%The reason there are discontinuities is due to catastrophic cancellation.
%We are  subtracting two numbers that are very close
%together: the relative error of the result can be much larger than
% the relative errors of the inputs.
%This is called "catastrophic cancellation and causes discontinuities
```

b) For the part a) one of the obtained roots is $x_1 = -b + \sqrt{b^2 + 1}$. When using 64-bit IEEE number representation (double precision) what is the smallest value of b the result of the computation $x_1 = -b + \sqrt{b^2 + 1}$ produces an answer $x_1 = 0$? Explain.

% type up your solution here

=2^53

```
%what is happening here is that 2^53 is an overflow that cannot be
%represented.
y=2^53+1
```

```
y = 9.0072e+15
```

```
x =2^53
```

```
x = 9.0072e+15
```

```
x-y
```

```
ans = 0
```

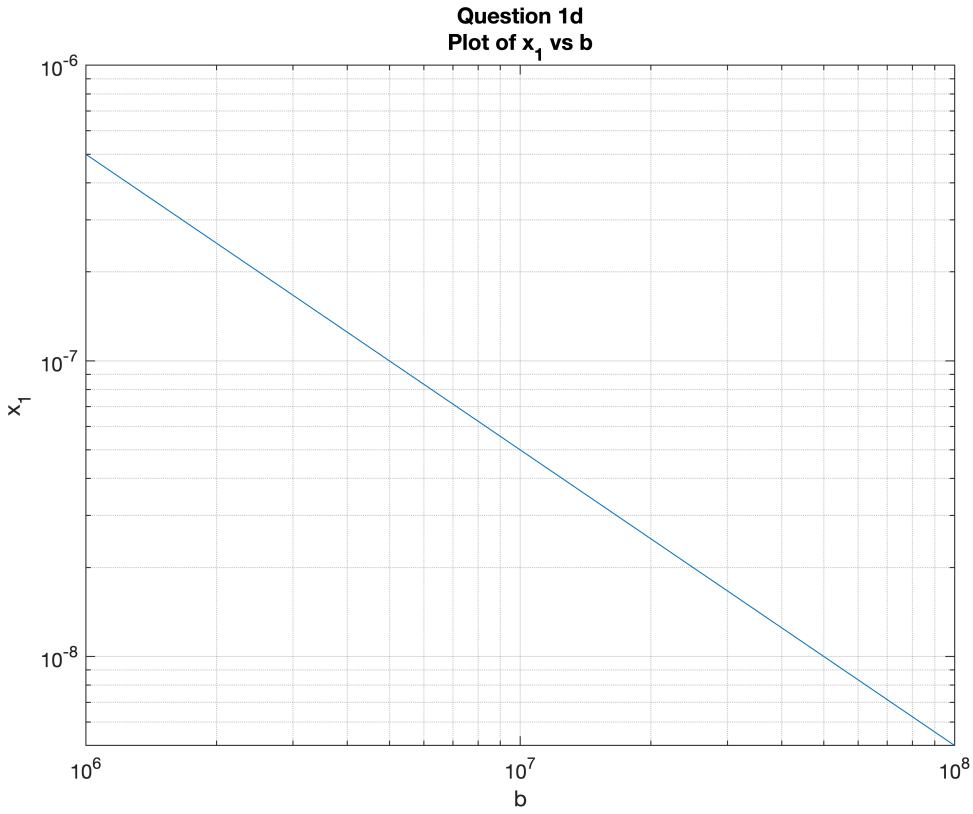
c) Prove that the product of roots $x_1x_2 = (-b + \sqrt{b^2+1})(-b - \sqrt{b^2+1}) = -1$ for all $b > 0$.

% type up your solution here

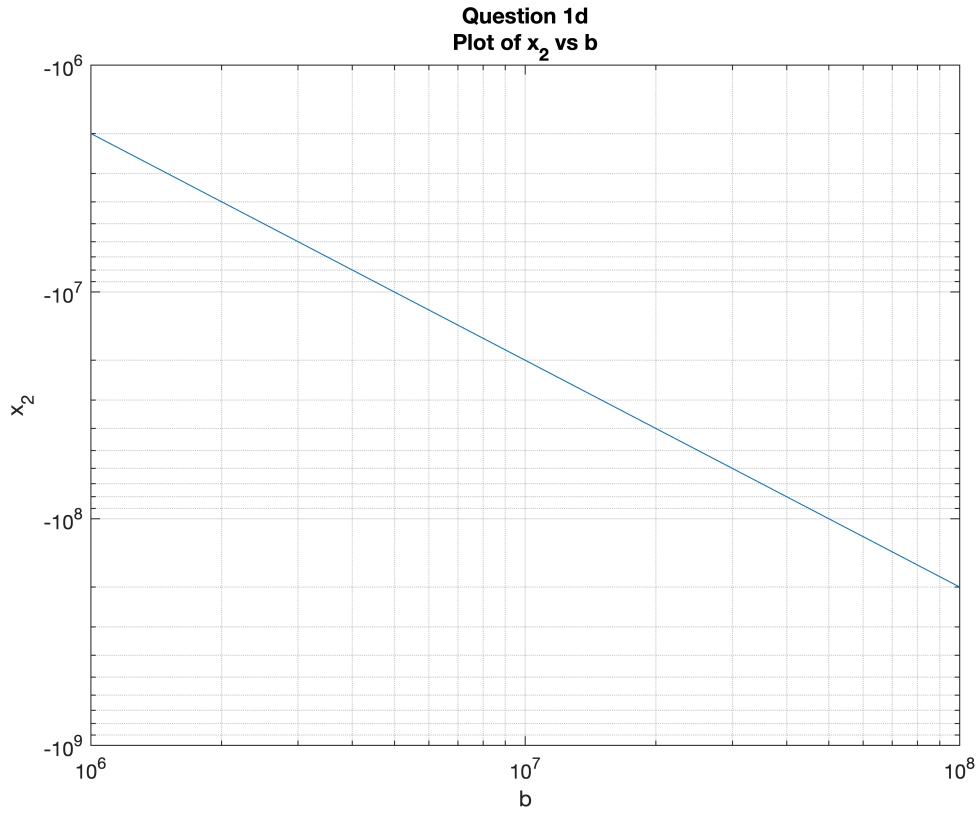
-b(-b-sqrt(b^2+1))+sqrt(b^2+1)(-b-sqrt(b^2+1))=b^2+b*sqrt(b^2+1)-b*sqrt(b^2+1) - sqrt(b^2+1)*sqrt(b^2+1)=-1

d) Write a script that computes and plots the roots x_1 and x_2 as a function of b for 1000 logarithmically spaced values of b between 10^6 and 10^8 (same as in part a. above) which does not suffer from numerical illconditioning.

```
% write your program here

b = logspace(6,8,1000);

%% Put your code for computing x1 and x2 below
x1 = -1./(-b-sqrt(b.^2 +1)) ;
x2 = -b-sqrt(b.^2 +1);


%plot x1 on a loglog scale
figure(1)
loglog(b,x1)
xlabel('b')
ylabel('x_1')
grid on
title({'Question 1d';'Plot of x_1 vs b'})
```




```
%plot x2 on a loglog scale
figure(2)
loglog(b,x2)
xlabel('b')
ylabel('x_2')
grid on
title({'Question 1d';'Plot of x_2 vs b'})
```



Question 2:

(a) **Cholesky factorization** is an alternative to LU decompostion which can be applied on symmetric positive definite matrices. A symmetric positive definite matrix $\boldsymbol{M}$ must satisfy following two conditions:

1. The matrix $\boldsymbol{M}$ must be sysmetric, i.e. $\boldsymbol{M} = \boldsymbol{M}^{\boldsymbol{T}}$

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

If the above two conditons are met then the matrix $\boldsymbol{M}$ can be factored as

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

where $\boldsymbol{L}$ is a $N \times N$ the lower triangular matrix with real and positive diagonal enteries. The enteries of matrix $\boldsymbol{L}$ 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, the use the outline of the function named *CholeskyDecomposition* provided in the appendix.

Use the cell below to test your Cholesky decompostion code by computing the Frobenius norm of $(\boldsymbol{L}\,\boldsymbol{L}^T - M)$.

```
M= [2 -1 0; -1 2 -1; 0 -1 2];

L = CholeskyDecompositon(M)
```

```
L = 3×3
    1.4142         0         0
   -0.7071    1.2247         0
         0   -0.8165    1.1547
```

```
W= chol(M)
```

```
W = 3×3
    1.4142   -0.7071         0
         0    1.2247   -0.8165
         0         0    1.1547
```

```


Error =  norm(L*L'-M)
```

```
Error = 4.4409e-16
```

```

```

b) In regression problems we ofthen need to solve the normal eqautions, $\boldsymbol{A}^T\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}$. Show that the matrix $(\boldsymbol{A}^T\boldsymbol{A})$ is symmetric positive definite, and thus can be factored using Cholesky Decomposition.

answer=

to check the matrix $(\boldsymbol{A}^T\boldsymbol{A})$ symetry :

$(\boldsymbol{A}^T\boldsymbol{A})^T = \boldsymbol{A}^T(\boldsymbol{A}^T)^T = \boldsymbol{A}^T\boldsymbol{A}$

$(x^T)\boldsymbol{A}^T\boldsymbol{A})\,\boldsymbol{x} = x^T\boldsymbol{A}^T(\boldsymbol{A}\,\boldsymbol{x}) = (\boldsymbol{A}\,\boldsymbol{x})^T(\boldsymbol{A}\,\boldsymbol{x}) = (\boldsymbol{A}\,\boldsymbol{x})\,(\boldsymbol{A}\,\boldsymbol{x})$ cos (0) >0 for all x != 0. therefore it is positive definite.

c) Using Cholesky facotrization scheme $(A^T A)$ can be factored as $A^T A = L L^T$. This transforms the normal equation into the form,

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

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 function named *LeastSquareSolver_Cholesky* for least squares in the function in the appendix.

*Note: for the forward and backward substitution use the functions named **ForwardSubs** and **BackwardSubs** provided in the appendix.*

Test the your Cholesky Least Saquare solver below.

```
A = rand(30,15); % random over-determined system
b = ones(30,1)
```

```
b = 30×1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     ⋮
```

```
x = LeastSquareSolver_Cholesky(A,b);
```

```
L = 15×15
    3.1537         0         0         0         0         0         0         0 ···
    2.6989    1.8376         0         0         0         0         0         0
    2.0185    1.2209    1.7904         0         0         0         0         0
    2.2267    0.3402    0.9519    1.9110         0         0         0         0
    2.7092    0.6629    1.1544    0.5496    1.4163         0         0         0
    2.3950    1.2635    0.5565    0.8680    0.9862    1.7211         0         0
    2.1968    0.1669    1.0324    0.8317   -0.1668    0.3205    1.6154         0
    2.5258    0.1232    0.4657    0.8905    0.0250    0.2875    0.4333    1.4046
    2.4783    0.7806    1.0981    1.0558   -0.1162    0.6524   -0.0180    0.0364
    2.5006    0.5996    0.8504    0.8777    0.0751    0.4471    0.2779    0.5350
     ⋮
y = 15×1
    3.1485
   -4.1682
  -23.8594
```

```
   20.8373
   25.9799
    2.3113
  -49.8154
   -8.9612
   30.8020
  -21.6218
      ⋮
x = 15×1
    0.9984
   -2.2683
  -13.3262
   10.9039
   18.3440
    1.3429
  -30.8377
   -6.3798
   20.0325
  -15.8187
      ⋮
```

```
residual = A*x-b;

Norm_residual = norm(residual);

disp(['Question 2c: The norm of the residual after solving using Cholesky based least s
```

```
Question 2c: The norm of the residual after solving using Cholesky based least sqaure solver is 1708.768
```

Question 3:

a) Show that a good estimate for the condition number of A^TA is $\kappa\left(A^TA\right) \approx \kappa(A)^2$.

% type your answer here

in the pdf file.

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

% type your answer here

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

% type your answer here

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 = rand(15,6); % A is a 15x6 matrix of random numbers
[Qgs,Rgs] = gschmidt(A)
```

```
Qgs = 15×6
    0.1423    0.1833    0.2901    0.3630    0.1950    0.1849
    0.3750   -0.1143    0.1529   -0.0700   -0.3056   -0.0128
    0.3674    0.1119    0.0299   -0.1177   -0.6310   -0.0268
    0.1359   -0.0438    0.0066    0.2598   -0.2437    0.5614
    0.3650    0.1775   -0.3046    0.1185   -0.0369   -0.2279
    0.0117    0.1661    0.4041   -0.1637    0.0888    0.5286
    0.3754   -0.1923    0.0197    0.1918    0.1692    0.0470
    0.2194   -0.0160   -0.0640    0.0410    0.4655    0.0151
    0.0460    0.5417    0.1914   -0.1854    0.1982   -0.1008
    0.2595   -0.0779    0.2888    0.2031    0.0782   -0.4044
     ⋮

Rgs = 6×6
    2.2305    1.2913    1.5683    2.1670    1.8107    1.5210
         0    1.6819    1.0439    0.7144    0.6116    0.4055
         0         0    1.2777    0.3676    0.7931    0.6013
         0         0         0    1.2081    0.1992    0.4233
         0         0         0         0    1.0128    0.4373
         0         0         0         0         0    1.3144
```

```
%use the norm to check
normGS = norm(A-Qgs*Rgs,'fro')
```

```
normGS = 4.4664e-16
```

```
%produces a relatively small frobenius norm
```

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 = rand(15,6); % A is a 15x6 matrix of random numbers
[Qmgs,Rmgs] = mgschmidt(A);

normMGS = norm(A-Qmgs*Rmgs,'fro')
```

```
normMGS = 3.7898e-16
```

```
%produces a relatively small frobenius norm
```

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 = rand(15,6); % A is a 15x6 matrix of random numbers
[Q,R] = householder(A);

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

```
normHH = 2.0920e-15
```

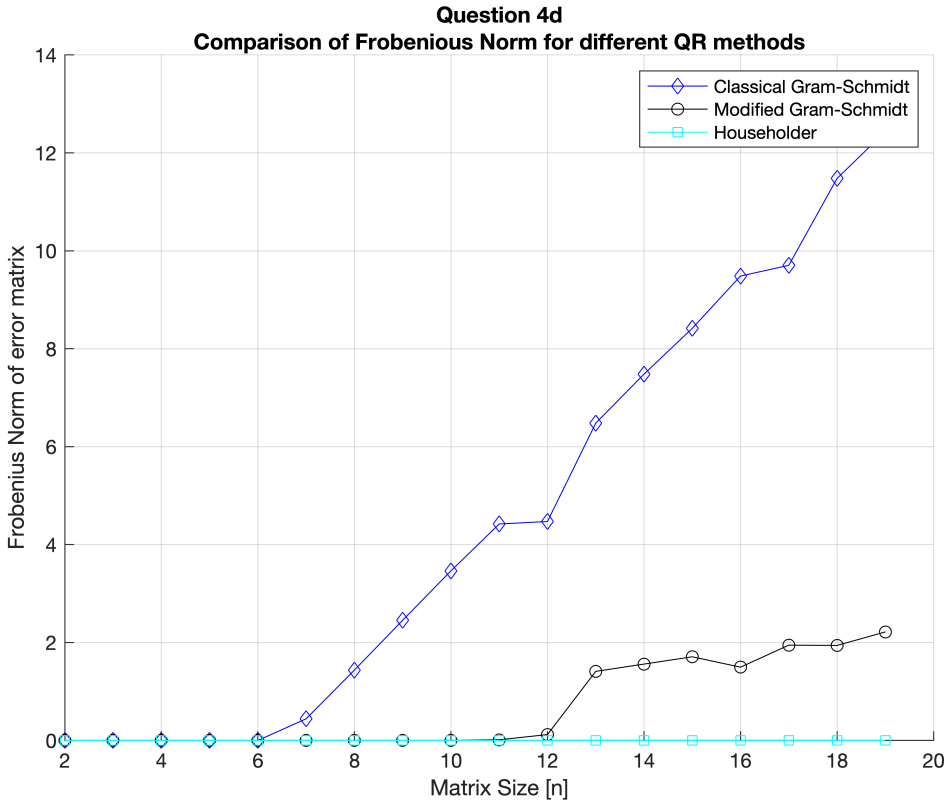
```
%produces a relatively small frobenius norm
```

d) We know that $\boldsymbol{Q}$ matrix obtained using QR decompostion should have orthonormal columns i.e. $\boldsymbol{Q^TQ = I}$. 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

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

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 $(\boldsymbol{Q^TQ - I})$ to access the performance of the lorithms written in part (a), (b), and (c). Comment on the results obtained, which method performs better.

```
for n=2:19
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


figure
hold on
grid on
plot(2:19,err_gs,'b-d')
plot(2:19,err_mgs,'k-o')
plot(2:19,err_hh,'c-s')
ylabel('Frobenius Norm of error matrix');
xlabel('Matrix Size [n]')
legend('Classical Gram-Schmidt','Modified Gram-Schmidt','Householder');
title({'Question 4d'; 'Comparison of Frobenious Norm for different QR methods'})
```




```
%From the graph we can conclude that as matrix size n >6 classical
%gram-schmidt performs worst has the highest frebenius norm of error matrix
%while Householder remains constant thoughout regardless of the matrix
%size.modified gram-schmidt performs worse than householder but better than
```

```
%classical gram-schmidt.
```

Question 5:

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

$$\begin{array}{|l} Input,,x \\ Output,y \end{array} \begin{array}{|cccccc} x_0 & x_1 & x_2 & x_3 & \dots & x_{m-1} & x_m \\ y_0 & y_1 & y_2 & y_3 & \dots & y_{m-1} & y_m \end{array}$$

we can fit a n^{th}degree polynomial, $p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n$. To find the coefficicients, $\boldsymbol{A} = [a_0,\ a_1,\ a_2, \cdots a_n]$, we can solve the linear system shown below.

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m-1} & x_{m-1}^2 & \cdots & x_{m-1}^n \\ 1 & x_m & x_m^2 & \cdots & x_m^n \end{bmatrix}}_{\boldsymbol{M}} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}}_{\boldsymbol{a}} = \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{m-1} \\ y_m \end{bmatrix}}_{\boldsymbol{Y}}$$

The system $\boldsymbol{M}\,\boldsymbol{a} = \boldsymbol{Y}$ can then be solved to compute the coefficients in vector $\boldsymbol{a}$.

a) Your first task is to write a function that takes the input data vector $\boldsymbol{x}$ and degree of the polynomial, then returns the polynomial matrix $\boldsymbol{M}$. Complete the function *PolynomialMatrix* in the appendix.

```
% test your code here by inputing X= [ 1 2 3] for degree 3.

% You can manually verify if this matrix is implemented correctly.

X= [ 1; 2; 3];
Mtest = PolynomialMatrix(X,3)
```

```
Mtest = 3×4
     1     1     1     1
     1     2     4     8
     1     3     9    27
```

```

```

b) The least squares method can be used to solve the overdetermined linear system $\boldsymbol{M}\,\boldsymbol{a} = \boldsymbol{Y}$ to obtain the polynomial coefficients. This can be done by directly solving the Normal equations (using cholesky

decomposition for example) or by using QR decomposition. In the code below, use the **MATALB's QR decomposition function** to find the polynomial coefficients if we assume a polynomial model of order 1.

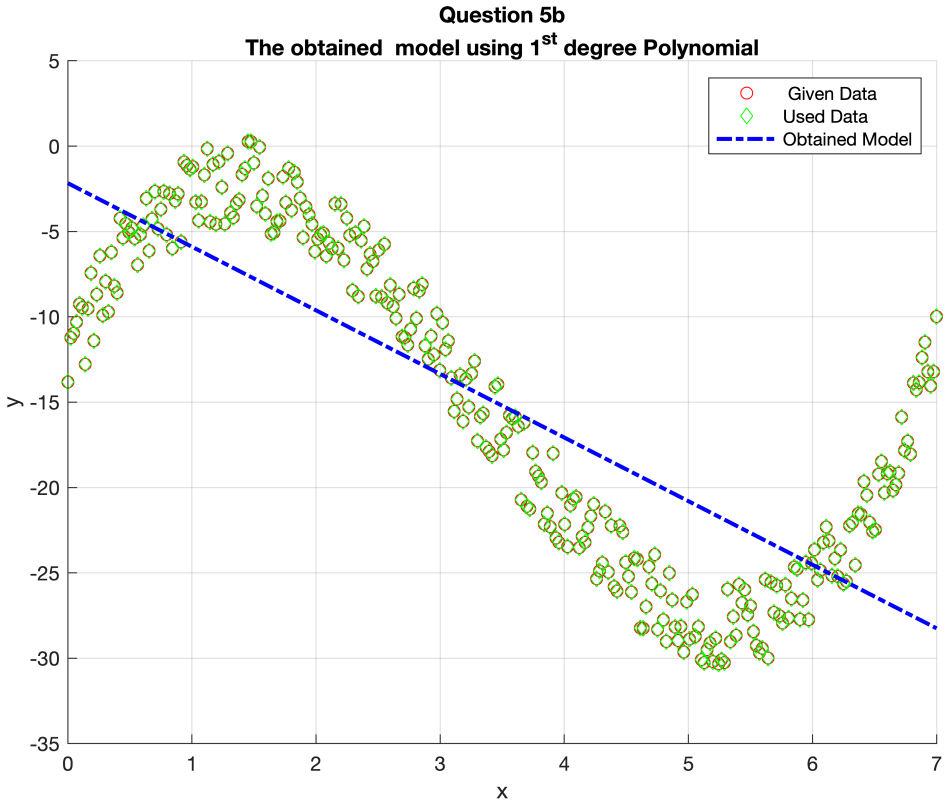
*Note: for the forward and backward substitution use the functions named **ForwardSubs** and **BackwardSubs** provided in the appendix.*

```
% Load the Input Data from the provided .mat file

load('Polynomial_Fitting_Data.mat');

x = x_given;
y = y_given;
deg = 1;

% get Polynomial Matrix
M = PolynomialMatrix(x,deg);



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

   % Find the polynomial coefficients using Q & R matrices obtained obove.
   % For forward and backward substitution functions use following
   % functions ForwardSubs(), BackwardSubs



   % Name the Coefficeints obtained QR as Ar

            % write your code here

   Ar = (R) \ (Q'*y);
```

Now that you have obtained the coefficients for the first order polynomial model, $p(x) = a_0 + a_1 x$, we can test this model by evaluating $p(x)$ at different values of x between 0 to 7. **Run the code below and explain the results obtained in the figure. Is this order sufficient?**

```
   xtest = x;
   powers = 0:deg;
   px = zeros(length(xtest),1);

   for I = 1:length(powers)
       px = px + Ar(I)*xtest.^powers(I);
   end
   %plots
    figure
   hold on
   plot(x,y,'ro')
 plot(x,y,'gd')
```

```
    plot(xtest,px,'b-.','Linewidth',2)
    xlabel('x')
    ylabel('y')
    legend(' Given Data','Used Data','Obtained Model')
    grid on
    title({'Question 5b';'The obtained  model using 1^{st} degree Polynomial'})
```




```
    %this order is not suffiecient this is because it is using only few
    %data points therefore it does not give a close accurate approximation
    %it is underfitting- the model performs poorly on the training data
    %because it is unable to capture the relationship between the input
    %examples and the output examples.This small degree of fredom will give
    %us false hypothesis.
```

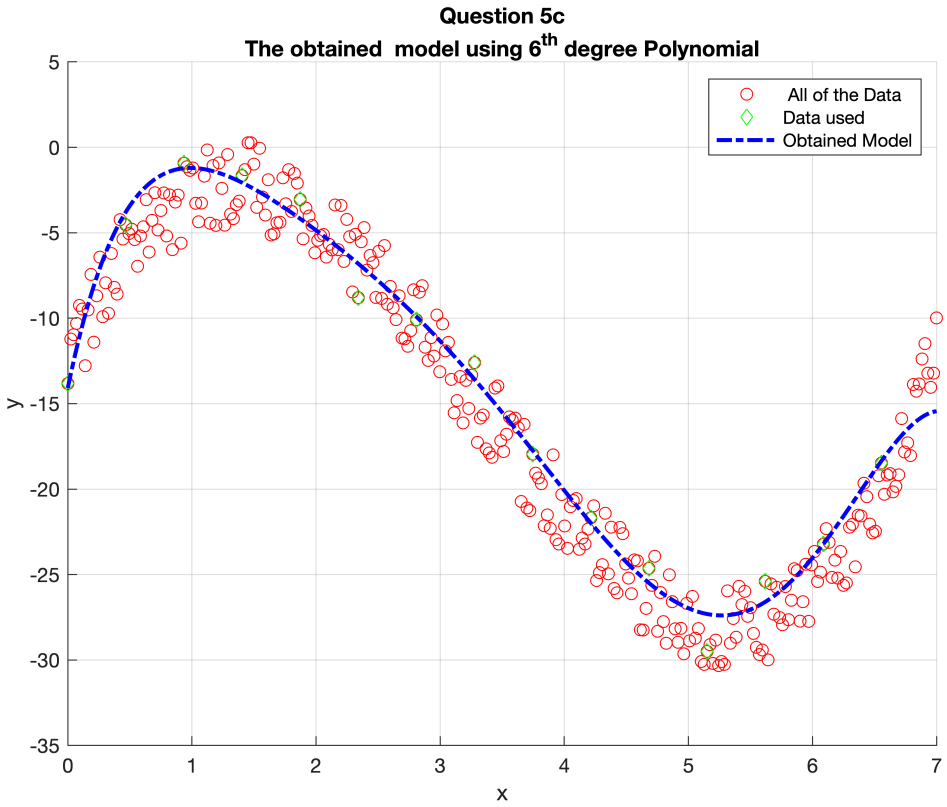
c) In this part we will use a higher order model (a polynomial of order 6), but only a limited amount of input data points to generate this model (we will use only **15 data points)**. Use QR decomposition function to find the polynomial coefficients for the **6th degree polynomial**.

```
% Load the Input Data from the provided .mat file

load('Polynomial_Fitting_Data.mat');

deg = 6; %polynomial degree
x = x_given(1:20:end);  % x has 15 data points!!
y = y_given(1:20:end);  % y has 15 data points!!
```

```
% get Polynomial Matrix
Mp= PolynomialMatrix(x,deg);


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

    %-------------------------------------------------------------------------------
    % Find the polynomial coefficients using  Q & R matrices obtained above.



    % Name the Coefficeints obtained QR as Ar

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

We can test this model by evaluating $p(x) = a_0 + a_1 x + \ldots + a_6 x^6$ at different values of x between 0 to 7. **Is this a good model? Comment on the results obtained in figure Question 5c, and explain the cause of what you see.**

```
    xtest = linspace(0,7,701);
    xtest = xtest';
    powers = 0:deg;
    px = zeros(length(xtest),1);

    for I = 1:length(powers)
        px = px + Ar(I)*xtest.^powers(I);

    end
    %plots
    figure
    hold on
    plot(x_given,y_given,'ro') % plot all Data
    plot(x,y,'gd') % plot Used Data
    plot(xtest,px,'b-.','Linewidth',2)
    xlabel('x')
    ylabel('y')
    legend(' All of the Data','Data used','Obtained Model')
    title({'Question 5c';'The obtained  model using 6^{th} degree Polynomial'})
    grid on
```




```
% Yes this is a good model.It is a balanced model .The model is able to
% capture the behaviour of input  to output.A relatively good selection of
% degree of freedom
```

d) In this part we will use a polynomial of order **14,** again this will be used with limited amount of data (we will use only **15 data points, same as part c**). Use QR decomposition function to find the polynomial coefficients for the **14th degree polynomial**.

```
% Load the Input Data from the provided .mat file

load('Polynomial_Fitting_Data.mat');

deg =14
```

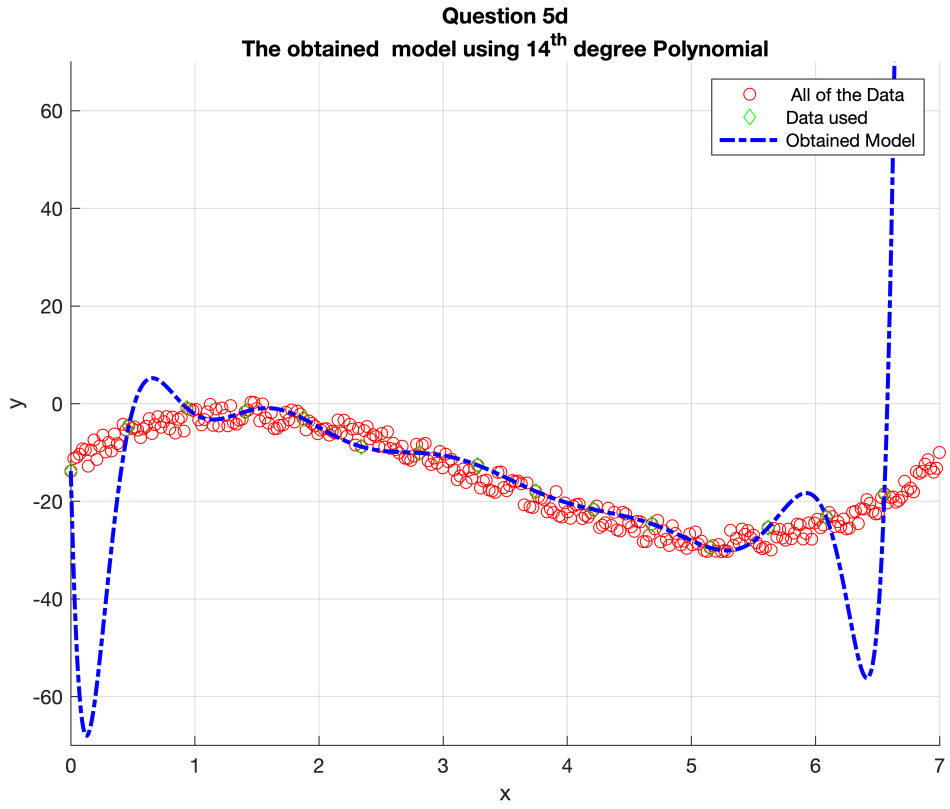
```
deg = 14
```

```
x = x_given(1:20:end);  % x has 15 data points!!
y = y_given(1:20:end);  % y has 15 data points!!

% get Polynomial Matrix
Mp= PolynomialMatrix(x,deg);


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

   %-------------------------------------------------------------------------------
```

```
% Find the polynomial coefficients using  Q & R matrices obtained obove.



% Name the Coefficeints obtained QR as Ar

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

We can test this model by evaluating $p(x)$ at different values of x between 0 to 7. **Is this a good model? Comment on the results obtained in figure Question 5d, and explain the cause of what you see.**

```
xtest = linspace(0,7,701);
xtest = xtest';
powers = 0:deg;
px = zeros(length(xtest),1);

for I = 1:length(powers)
    px = px + Ar(I)*xtest.^powers(I);

end
%plots
figure('Name','Question 5d','NumberTitle','off')
hold on
plot(x_given,y_given,'ro') % plot all Data
plot(x,y,'gd') % plot Used Data
plot(xtest,px,'b-.','Linewidth',2)
xlabel('x')
ylabel('y')
ylim([-70 70])
legend(' All of the Data','Data used','Obtained Model')
title({'Question 5d';'The obtained  model using 14^{th} degree Polynomial'})
grid on
```




```
% this is not a good model. This is because we used a very large degree of
% freedom therefore a large sample size which fails to generalize the unseen
% data and some seen data
```

e)Finally, now will use the various different versions of QR decompositon implemented in earlier questions. Use three versions of the QR decompostion that you implemented in Question 3. We want to fit the **polynomial of degree 59** through with **60 data points**. Clearly show all the plots (with legends) and comment on the results obtained.

```
% Load the Input Data from the provided .mat file

load('Polynomial_Fitting_Data.mat');

deg = 59
```

```
deg = 59
```

```
x = x_given(1:5:end);  % x has 60 data points!!
y = y_given(1:5:end);  % y has 60 data points!!

% get Polynomial Matrix
 Mp= PolynomialMatrix(x,deg);


   % Classical Gram-Schmidt QR
   [Qgs,Rgs] =  gschmidt(Mp);
```

```
% Modified Gram-Schmidt QR
 [Qmgs,Rmgs] =  mgschmidt(Mp);

% Householder QR
 [Qhh,Rhh] =  householder(Mp);
%-------------------------------------------------------------------------------
% Find the polynomial coefficients using  Q & R matrices obtained above.


% Name the Coefficeints obtained classical Gram-Schmidt QR as Ags
        ysol=ForwardSubs(Mp,x)
```

```
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =
 1.320904e-51.
ysol = 60×1
     0
     1
     0
     0
     0
     0
     0
     0
     0
     0
     ⋮
```

```
           % write your code here
         Ags = (Rgs) \ (Qgs'*ysol);
```

```
Warning: Unable to create personal MATLAB work folder:/Users/Mwirigi/Documents/MATLAB
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =
 3.396851e-50.
```

```
% Name the Coefficeints obtained Modified Gram-Schmidt QR as Amgs

           % write your code here
           Amgs= (Rmgs) \ (Qmgs'*ysol);
```

```
Warning: Unable to create personal MATLAB work folder:/Users/Mwirigi/Documents/MATLAB
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =
 6.173996e-34.
```

```
% Name the Coefficeints obtained Householder QR as Ahh

           % write your code here
          Ahh = (Rhh) \ (Qhh'*ysol);
```

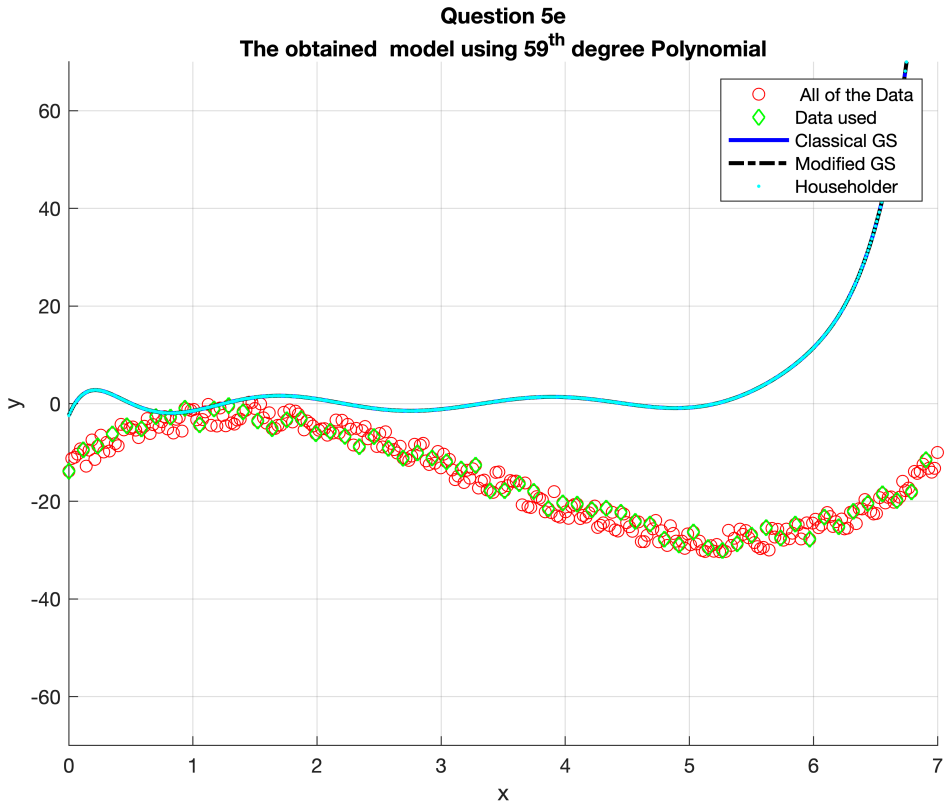
```
Warning: Unable to create personal MATLAB work folder:/Users/Mwirigi/Documents/MATLAB
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =
 3.542046e-58.
```

We can test this model by evaluating $p(x) = a_0 + a_1 x + \ldots + a_{59} x^{59}$ at different values of x between 0 to 7. Comment on the results obtained.

```
xtest = linspace(0,7,701);
```

```
Warning: Unable to create personal MATLAB work folder:/Users/Mwirigi/Documents/MATLAB
```

```
xtest = xtest';

pgs = zeros(length(xtest),1);
pmgs = zeros(length(xtest),1);
phh = zeros(length(xtest),1);
for I = 1:length(powers)
   pgs = pgs + Ags(I)*xtest.^powers(I);
   pmgs = pmgs + Amgs(I)*xtest.^powers(I);
   phh = phh + Ahh(I)*xtest.^powers(I);
end

%plots
figure

hold on
plot(x_given,y_given,'ro') % plot all Data
plot(x,y,'gd','LineWidth',3) % plot Used Data
plot(xtest,pgs,'b-','Linewidth',2)
plot(xtest,pgs,'k-.','Linewidth',2)
plot(xtest,pgs,'c.','Linewidth',1.2)
xlabel('x')
ylabel('y')
ylim([-70 70])
legend(' All of the Data','Data used','Classical GS','Modified GS','Householder')
  title({'Question 5e';'The obtained  model using 59^{th} degree Polynomial'})
grid on
```



Appendix

Implement the Cholesky Decompositon fucntion here.

```
function L =  CholeskyDecompositon(M)
if(size(M,1) ~= size(M,2))
    disp('Matrix M must be square!');

end
n = length( M );
L = zeros( n, n );
for i=1:n
   L(i, i) = sqrt(M(i, i) - L(i, :)*L(i, :)');
   for j=(i + 1):n
      L(j, i) = (M(j, i) - L(i,:)*L(j ,:)')/L(i, i);
   end
end
 L;
end
```

Implement Least Square Solver for Cholesky here

```
function x = LeastSquareSolver_Cholesky(A,b)
% get M = A^T
M = A'*A;
```

```
L = CholeskyDecompositon(M)

y = ForwardSubs(A,b)

x = BackwardSubs(L,y)

end
```

Implement Polynomial Matrix here

```
function M = PolynomialMatrix(x,n)
% write your code here
M = ones(size(x));
 deg = n;
 powers = 0:deg;
    for Y = 1:length(powers)
         M(:,Y) = x.^powers(Y);
    end
end
```

Implement QR Methods here

- *Classical GramSchmidt*

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

   % Write your code here.
   [p w] = size(A);
Q = zeros(p,w);
R = zeros(w);
for j=1:w
    v=A(:,j);
    for i=1:j-1
        R(i,j)=Q(:,i)'*A(:,j);
        v=v-R(i,j)*Q(:,i);
    end
    R(j,j)= norm(v);
    if R(j,j)~=0
        Q(:,j)=v/R(j,j);
    else
        Q(:,j)=v;
    end

end
end
```

- *Modified Gram-Schimdt*

```

```

```
function [Q,R] = mgschmidt(A)
[m,n]=size(A);
p=min(m,n);
R=zeros(p,n);
Q=zeros(m,p);
  for k=1:p-1
     R(k,k)=norm(A(:,k));
     Q(:,k)=A(:,k)/R(k,k);
     R(k,k+1:p)=A(:,k+1:p)'*Q(:,k);
     A(:,k+1:p)=A(:,k+1:p)-Q(:,k)*R(k,k+1:p);
end;
  R(p,p)=norm(A(:,p));
  Q(:,p)=A(:,p)/R(p,p);
end
```

- *Householder QR method*

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

[m,n] = size(A);
Q = eye(m);
R = A;
for j = 1:n

nox = norm(R(j:end,j));

s = -sign(R(j,j));

P = R(j,j) - s*nox;

w = R(j:end,j)/P;

w(1) = 1;

W = -s*P/nox;

R(j:end,:) = R(j:end,:)-(W*w)*(w'*R(j:end,:));
Q(:,j:end) = Q(:,j:end)-(Q(:,j:end)*w)*(W*w)';

end
end
```

Forward and Backward Substitution functions are provided here

```
function X = ForwardSubs(A,b)
%Inputs:
% A is a square matrix-
% b is the vector
%Outputs:
% X is the output vector
opt.LT = true;
```

```
opt.UT = false;
X= linsolve(A,b,opt);
end

function X = BackwardSubs(A,b)
%Inputs:
% A is a square matrix-
% b is the vector
%Outputs:
% X is the output vector
opts.LT = false;
opts.UT = true;
X= linsolve(A,b,opts);
end
```

Function to generate Hilbert Matrix is provided here

```
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
```