COURSEWORK 1

1. First, from problem sheet 2, we derived an algorithm in order to get QR-decomposition using householder reflectors of any (mxn) matrix. We will implement this function into our bigger algorithm that does the parallel version of the QR-decomposition. See below the normal householder QR-decomposition:

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

    % Initialization : Find size of A
        % m = number of rows
        % n = number of columns
    [m, n] = size(A);


    % STEP 1 : Householder-based QR decomposition

        % STEP 1.1 : Initialization of F, householder reflector into zeros
            and A,
        % matrix we want to decomposoe

    F_3D = zeros(m, m, n);
    A_1 = A;

        % STEP 1.2 : Loop (as per lecture notes)
    for k = 1:n

            % STEP 1.2.1 : Pick kth column vector
        x = A(k:m, k);

            % STEP 1.2.2 : Give identity matrix size (m-k+1) for reflector
            % computation below
        I = eye(m-k+1);

            % STEP 1.2.3 : Define basis vector e1
        e = I(:,1);

            % STEP 1.2.4 : Find sign of first entry of vector x
        if sign(x(1)) ~= -1
            s = 1;
        else
            s = -1;
        end

            % STEP 1.2.5 : Compute v
        v = s * norm(x) * e + x;

            % STEP 1.2.6 : Normalize v
        v = v/norm(v);

            % STEP 1.2.7 : Update A & F
        A(k:m, k:n) = A(k:m, k:n) - 2 * v * (v' * A(k:m, k:n));
```

```
        F = eye(m-k+1) - 2 * v * v';

            % STEP 1.2.8 : Update each layer of reflector
        F_3D(:, :, k) = blkdiag(eye(k-1), F);

    end


    % STEP 2 : Reverse engineer reflectors

        % STEP 2.1 : Initialization of our transposed Q
    Q_T = F_3D(:, :, 1);

        % STEP 2.2 : Hit Q_T with updated reflector each time
    for i = 2:n
        Q_T = F_3D(:, :, i) * Q_T;
    end

        % STEP 2.3 : Finally, compute QR from Full QR-decomposition
    R = Q_T * A_1;

    Q = Q_T';


    % STEP 3 : Reduction of QR decomposition (for purposes of function
    % parallel_qr)

        % STEP 3.1 : Cut null space from Q ((m-n) columns on right of Q)
    Q = Q( : , 1:n);

        % STEP 3.2 : Cut unnecessary (m-n) rows at bottom of R
    R = R(1:n , :);


end
```

Now, look at the computer program implementing the parallel version of the QR-decomposition:

```
function [Q_final, R_final] = parallel_qr(A,p)

    % Initialization : Find size of A
        % m = number of rows
        % n = number of columns
    [m, n] = size(A);


    % STEP 1 : Partition A into p submatrices

        % STEP 1.1 : Use m = qp + r to compute q & r
    r = rem(m,p);
    q = floor(m/p);

        % STEP 1.2 : Initialization of equal size submatrices with zeros
    A_sub = zeros(q,n,p-1);

        % STEP 1.3 : Create the  first (p-1) submatrices of size (qxn)
    for i = 1:p-1
```

```
20          A_sub(:,:,i) = A((i-1)*q + 1 : i*q , :);
21      end
22
23          % STEP 1.4 : Create the last submatrix of size ((q+r)xn)
24      A_sub_last = A((p-1)*q + 1 : m , :);
25
26
27      % STEP 2 : QR decomposition for each A_i p submatrices, i=1:p
28
29          % STEP 2.1 : Initialization of Q_1 (first (p-1) Q matrices), Q_1_last
                (last Q
30          % matrix) & R_1 (p R matrices) with zeros
31      Q_1 = zeros(q, n, p-1);
32      Q_1_last = zeros(q+r, n);
33      R_1 = zeros(n, n, p);
34
35          % STEP 2.2 : 1st QR decomposition for the first (p-1) submatrices
36      for i=1:p-1
37          [Q_temp0, R_temp0] = qr_factor(A_sub(:, :, i)) ;
38          Q_1(:, :, i) = Q_temp0;
39          R_1(:, :, i) = R_temp0;
40      end
41
42          % STEP 2.3 : 1st QR decomposition for the last submatrix p
43      [Q_temp1, R_temp1] = qr_factor(A_sub_last) ;
44      Q_1_last(:, :) = Q_temp1;
45      R_1(:, :, p) = R_temp1;
46
47          % STEP 2.4 : Concatenate matrices together to form R'
48      R_prime = R_1(:, :, 1);
49      for i = 2:p
50          R_prime = [R_prime ; R_1(:, :, i)];
51      end
52
53
54      % STEP 3 : QR decomposition for R'
55
56          % STEP 3.1 : Initialization of Q_2 with zeros
57      Q_2 = zeros(n,n,p);
58
59          % STEP 3.2 : 2nd QR decomposition for the matrix R_prime
60      [Q_prime , R_final] = qr_factor(R_prime);
61
62          % STEP 3.3 : Partition matrix Q_2 into p submatrices
63      for i = 1 : p
64          Q_2(:, :, i) = Q_prime((i-1)*n + 1 : i*n, :);
65      end
66
67
68      % STEP 4 : Multiplication to find final Q of big QR decomposition
69
70          % STEP 4.1 : Initialization of Q with zeros
71      Q = zeros(q, n, p-1);
72
73          % STEP 4.2 : Multiplication of Q_1's and Q_2's to form Q for first
74          % (p-1) submatrices
75      for i = 1:p-1
76          Q(:, :, i) = Q_1(:, :, i) * Q_2(:, :, i);
77      end
```

```
78
79          % STEP 4.3 : Compute last Q (pth submatrix)
80      Q_final_last = Q_1_last*Q_2(:, :, p);
81
82          % STEP 4.4 : Concetenate final submatrices Q together to create
83          % Q_final
84      Q_final = Q(:, :, 1);
85
86      for i = 2:p-1
87          Q_final = [Q_final ; Q(:, :, i)];
88      end
89
90      Q_final = [Q_final ; Q_final_last];
91
92  end
```

Now, let's compare our two algorithm with a random matrix A and p: (output below)

```
A = randi(35,35,3);
[Q,R]=qr_factor(A)
[Q,R]=parallel_qr(A,5)
```

Clearly, these results validate the parallel version of the QR-decomposition as it varies by a multiple of -1 (which doesn't affect our result)

```
>> A=randi(35,35,3)

A =

    15     1    32
     1    33    11
    27    22    26
     3    32    12
    16    15    13
    19    15    21
    22    20    32
    16    21    10
     3    18    34
    10    21    26
     3     5     1
    26    27    30
    13    23    25
    23    13    20
    17    11    12
    14    10    12
    12     6     3
     4    12    17
    13    30    24
    14     9    21
    17     3    15
    28    18    33
    25     5    13
    30    15    34
    18     4     2
    30    26    16
    17    12    25
    10    20    25
    23    23    18
    20    23    22
    19    25    27
    14    11    29
    15     3    12
     5    23    20
    11    10    30
```

```
>> [Q,R]=qr_factor(A)

Q =

   -0.1437    0.1660    0.4035
   -0.0096   -0.4772   -0.1373
   -0.2587   -0.0007   -0.0613
   -0.0287   -0.4383   -0.1328
   -0.1533   -0.0295   -0.1005
   -0.1821    0.0066    0.0118
   -0.2108   -0.0313    0.1299
   -0.1533   -0.1185   -0.2161
   -0.0287   -0.2307    0.4150
   -0.0958   -0.1908    0.1527
   -0.0287   -0.0380   -0.0673
   -0.2491   -0.0869   -0.0250
   -0.1246   -0.1843    0.0782
   -0.2204    0.0845   -0.0347
   -0.1629    0.0418   -0.0912
   -0.1342    0.0205   -0.0452
   -0.1150    0.0557   -0.1482
   -0.0383   -0.1297    0.1475
   -0.1246   -0.2881   -0.0102
   -0.1342    0.0353    0.1317
   -0.1629    0.1604    0.0443
   -0.2683    0.0707    0.0965
   -0.2396    0.2272   -0.1087
   -0.2875    0.1392    0.1210
   -0.1725    0.1577   -0.2188
   -0.2875   -0.0238   -0.3226
   -0.1629    0.0270    0.1399
   -0.0958   -0.1760    0.1441
   -0.2204   -0.0637   -0.1716
   -0.1916   -0.0999   -0.0615
   -0.1821   -0.1416    0.0233
   -0.1342    0.0057    0.2601
   -0.1437    0.1363    0.0127
   -0.0479   -0.2807    0.0814
   -0.1054   -0.0157    0.3246

R =

 -104.3600  -84.8506 -113.1660
    0.0000  -67.4491  -36.2911
    0.0000    0.0000   53.9204
```

```
>> [Q,R]=parallel_qr(A,5)

Q =

    0.1437   -0.1660   -0.4035
    0.0096    0.4772    0.1373
    0.2587    0.0007    0.0613
    0.0287    0.4383    0.1328
    0.1533    0.0295    0.1005
    0.1821   -0.0066   -0.0118
    0.2108    0.0313   -0.1299
    0.1533    0.1185    0.2161
    0.0287    0.2307   -0.4150
    0.0958    0.1908   -0.1527
    0.0287    0.0380    0.0673
    0.2491    0.0869    0.0250
    0.1246    0.1843   -0.0782
    0.2204   -0.0845    0.0347
    0.1629   -0.0418    0.0912
    0.1342   -0.0205    0.0452
    0.1150   -0.0557    0.1482
    0.0383    0.1297   -0.1475
    0.1246    0.2881    0.0102
    0.1342   -0.0353   -0.1317
    0.1629   -0.1604   -0.0443
    0.2683   -0.0707   -0.0965
    0.2396   -0.2272    0.1087
    0.2875   -0.1392   -0.1210
    0.1725   -0.1577    0.2188
    0.2875    0.0238    0.3226
    0.1629   -0.0270   -0.1399
    0.0958    0.1760   -0.1441
    0.2204    0.0637    0.1716
    0.1916    0.0999    0.0615
    0.1821    0.1416   -0.0233
    0.1342   -0.0057   -0.2601
    0.1437   -0.1363   -0.0127
    0.0479    0.2807   -0.0814
    0.1054    0.0157   -0.3246

R =

  104.3600   84.8506  113.1660
    0.0000   67.4491   36.2911
   -0.0000    0.0000  -53.9204
```

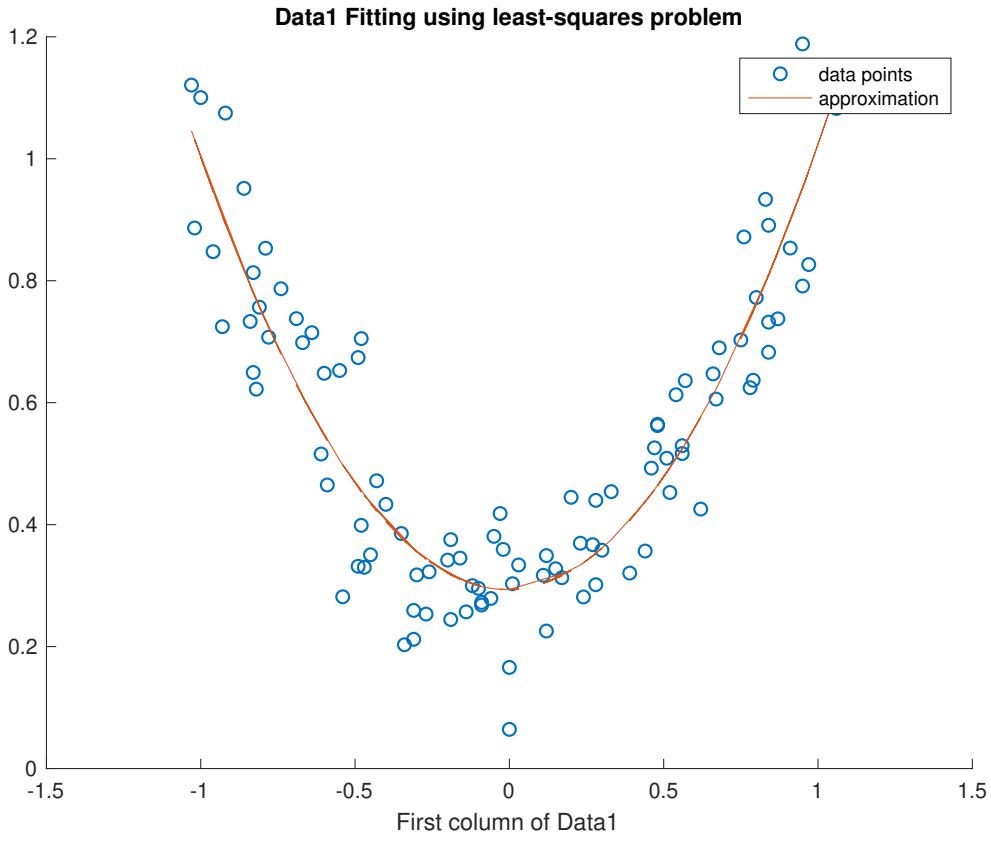
2. For the two given data sets, we will approximate the respective data using least-squares solver from problem sheet 2. Regarding Data1, we use a quadratic polynomial as follows:

```
% DATA 1 - Quadratic Polynomial

% STEP 1 : Initialization of values to solve Bx=b
    % STEP 1.1 : Load Data1
A = load('/Users/Hermine/Desktop/Data1.txt');
    % STEP 1.2 : Create matrix B
A1 = ones(100,1);
A2 = A(:,1);
A3 = A2.^2;
B = [A1,A2,A3];
    % STEP 1.3 : Define b
b = A(:,2);

% STEP 2 : Solve Bx=b
ans = least_squares(B,b);

% STEP 3 : Plot solutions
scatter(A2,b);
hold on;
plot(A2,ans(1)+ans(2)*A2+ans(3)*A3);
legend('data points', 'approximation')
title('Data1 Fitting using least-squares problem')
xlabel('First column of Data1')
hold off
```



Regarding Data2, we find with a scatter plot that it is a spiral, therefore, we have to transform to polar coordinates the system in order to solve it:

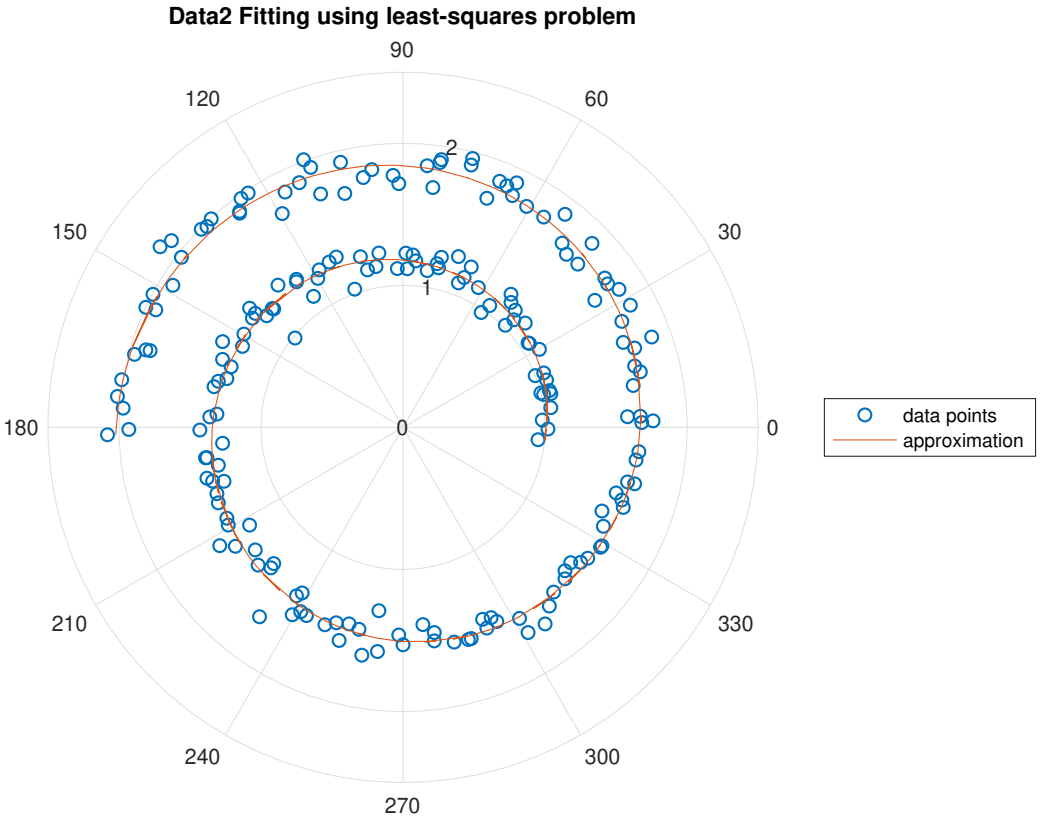
```
% DATA 2 - Polar 3rd degree polynomial

% STEP 1 : Initialization of values to solve Cx=rho
    % STEP 1.1 : load Data2
A = load('/Users/Hermine/Desktop/Data2.txt');
    % STEP 1.2 : Transform coordinates to polars
A1 = A(:,1);
A2 = A(:,2);
[theta,rho] = cart2pol(A1,A2);
    % STEP 1.3 : Adapt values of theta to their layers
theta(67:198,1) = theta(67:198,1) + 2*pi;
theta(199:200,1) = theta(199:200,1) + 4*pi;
    % STEP 1.4 : Create matrix C
B1 = ones(200,1);
B2 = theta;
B3 = B2.^2;
B4 = B2.^3;

C = [B1,B2,B3,B4];

% STEP 2 : Solve Cx=rho
ans = least_squares(C,rho);

% STEP 3 : Plot solutions
polarscatter(theta,rho);
hold on;
polarplot(theta, ans(1)+ ans(2)*B2 + ans(3)*B3 + ans(4)*B4);
legend('data points', 'approximation')
title('Data2 Fitting using least-squares problem')
hold off
```


Data2 Fitting using least-squares problem
90
120
60
150
30
180
0
210
330
240
300
270
2
1
0
data points
approximation