21960197 向平 Assignment 4

Ex 4.1.1

Form the normal equations, and compute the least squares solution and 2-norm error for the following inconsistent systems:

$$\text{(a)} \begin{bmatrix} 3 & -1 & 2 \\ 4 & 1 & 0 \\ -3 & 2 & 1 \\ 1 & 1 & 5 \\ -2 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \\ -5 \\ 15 \\ 0 \end{bmatrix}$$

```
A = [3 -1 2;
    4 1 0;
    -3 2 1;
    1 1 5;
    -2 0 3];
b = [10 10 -5 15 0]';
A'*A
```

```
ans = 3×3
    39    -4     2
    -4     7     5
     2     5    39
```

```
A'*b
```

```
ans = 3×1
   100
     5
    90
```

The normal equations are

$$\begin{bmatrix} 39 & -4 & 2 \\ -4 & 7 & 5 \\ 2 & 5 & 59 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 100 \\ 5 \\ 90 \end{bmatrix}$$

and can be solves as follows.

```
x = (A'*A)\(A'*b)
```

```
x = 3×1
    2.5246
    0.6616
    2.0934
```

The 2-norm error for the inconsistent systems are

```
norm_2_error = norm(A*x-b)
```

```
norm_2_error = 2.4135
```

$$
\text{(b)} \quad \begin{bmatrix} 4 & 2 & 3 & 0 \\ -2 & 3 & -1 & 1 \\ 1 & 3 & -4 & 2 \\ 1 & 0 & 1 & -1 \\ 3 & 1 & 3 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \\ 2 \\ 0 \\ 5 \end{bmatrix}
$$

```
A = [4 2 3 0;
    -2 3 -1 1;
    1 3 -4 2;
    1 0 1 -1;
    3 1 3 -2];
b = [10 0 2 0 5]';
A'*A
```

```
ans = 4×4
    31     8    20    -7
     8    23    -6     7
    20    -6    36   -16
    -7     7   -16    10
```

```
A'*b
```

```
ans = 4×1
    57
    31
    37
    -6
```

The normal equations are

$$
\begin{bmatrix} 31 & 8 & 20 & -7 \\ 8 & 23 & -6 & 7 \\ 20 & -6 & 36 & -16 \\ -7 & 7 & -16 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -7 \\ 7 \\ -16 \\ 10 \end{bmatrix}
$$

and can be solved as

```
x = (A'*A)\(A'*b)
```

```
x = 4×1
    1.2739
    0.6885
    1.2124
    1.7497
```

The resiual 2-norm error is

```
norm_2_error = norm(A*x-b)
```

```
norm_2_error = 0.8256
```

Ex 4.3.1

Write a Matlab program that implements classical Gram–Schmidt to find the reduced QR factorization. Check your work by comparing factorizations of the matrices in Exercise 1 with the Matlab qr(A,0) command or equivalent. The factorization is unique up to signs of the

entries of Q and R.

(a) $\begin{bmatrix} 4 & 0 \\ 3 & 1 \end{bmatrix}$

```
A = [4 0; 3 1];
[Q, R] = reduced_QR_factorization(A)
```

```
Q = 2×2
    0.8000   -0.6000
    0.6000    0.8000
R = 2×2
    5.0000    0.6000
         0    0.8000
```

```
Q*R-A
```

```
ans = 2×2
     0     0
     0     0
```

```
[Q_, R_] = qr(A, 0)
```

```
Q_ = 2×2
   -0.8000   -0.6000
   -0.6000    0.8000
R_ = 2×2
   -5.0000   -0.6000
         0    0.8000
```

(b) $\begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix}$

```
A = [1 2;1 1];
[Q, R] = reduced_QR_factorization(A)
```

```
Q = 2×2
    0.7071    0.7071
    0.7071   -0.7071
R = 2×2
    1.4142    2.1213
         0    0.7071
```

```
Q*R-A
```

```
ans = 2×2
     0     0
     0     0
```

```
[Q_, R_] = qr(A, 0)
```

```
Q_ = 2×2
   -0.7071   -0.7071
   -0.7071    0.7071
R_ = 2×2
```

```
   -1.4142   -2.1213
         0   -0.7071
```

(c) $\begin{bmatrix} 2 & 1 \\ 1 & -1 \\ 2 & 1 \end{bmatrix}$

```
A = [2 1;1 -1;2 1];
[Q, R] = reduced_QR_factorization(A)
```

```
Q = 3×2
    0.6667    0.2357
    0.3333   -0.9428
    0.6667    0.2357
R = 2×2
    3.0000    1.0000
         0    1.4142
```

```
Q*R-A
```

```
ans = 3×2
     0     0
     0     0
     0     0
```

```
[Q_, R_] = qr(A, 0)
```

```
Q_ = 3×2
   -0.6667    0.2357
   -0.3333   -0.9428
   -0.6667    0.2357
R_ = 2×2
   -3.0000   -1.0000
         0    1.4142
```

(d) $\begin{bmatrix} 4 & 8 & 1 \\ 0 & 2 & -2 \\ 3 & 6 & 7 \end{bmatrix}$

```
A = [4 8 1;0 2 -2;3 6 7];
tic;
[Q, R] = reduced_QR_factorization(A)
```

```
Q = 3×3
    0.8000         0   -0.6000
         0    1.0000         0
    0.6000         0    0.8000
R = 3×3
     5    10     5
     0     2    -2
     0     0     5
```

```
disp(['Reduced QR Factorization 运行时间: ',num2str(toc)]);
```

```
Reduced QR Factorization 运行时间: 0.011879
```

```
Q*R-A
```

```
ans = 3×3
```

```
     0     0     0
     0     0     0
     0     0     0
```

```
[Q_, R_] = qr(A, 0)
```

```
Q_ = 3×3
   -0.8000    0.0000   -0.6000
         0   -1.0000   -0.0000
   -0.6000   -0.0000    0.8000
R_ = 3×3
   -5.0000  -10.0000   -5.0000
         0   -2.0000    2.0000
         0         0    5.0000
```

From the above results, we can observe that the results obtained from the reduced QR factorization method in my own implementation are the same as the Matlab qr(A, 0) function, except the signs of the entries of *Q* and *R.*

Ex 4.3.3

Repeat Computer Problem 1, but implement Householder reflections.

(a) $\begin{bmatrix} 4 & 0 \\ 3 & 1 \end{bmatrix}$

```
A = [4 0; 3 1];
[Q, R] = QR_Householder_reflection(A)
```

```
Q = 2×2
   -0.8000    0.6000
   -0.6000   -0.8000
R = 2×2
   -5.0000   -0.6000
   -0.0000   -0.8000
```

```
Q*R-A
```

ans = 2×2

10^{-15} ×

```
   -0.4441         0
    0.4441         0
```

```
[Q_, R_] = qr(A)
```

```
Q_ = 2×2
   -0.8000   -0.6000
   -0.6000    0.8000
R_ = 2×2
   -5.0000   -0.6000
         0    0.8000
```

(b) $\begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix}$

```
A = [1 2;1 1];
[Q, R] = QR_Householder_reflection(A)
```

```
Q = 2×2
   -0.7071    0.7071
   -0.7071   -0.7071
R = 2×2
   -1.4142   -2.1213
         0    0.7071
```

```
Q*R-A
```

```
ans = 2×2
```
10^{-15} ×
```
   -0.2220   -0.4441
   -0.2220   -0.1110
```

```
[Q_, R_] = qr(A)
```

```
Q_ = 2×2
   -0.7071   -0.7071
   -0.7071    0.7071
R_ = 2×2
   -1.4142   -2.1213
         0   -0.7071
```

(c) $\begin{bmatrix} 2 & 1 \\ 1 & -1 \\ 2 & 1 \end{bmatrix}$

```
A = [2 1;1 -1;2 1];
[Q, R] = QR_Householder_reflection(A)
```

```
Q = 3×3
   -0.6667    0.2357   -0.7071
   -0.3333   -0.9428    0.0000
   -0.6667    0.2357    0.7071
R = 3×2
   -3.0000   -1.0000
   -0.0000    1.4142
    0.0000         0
```

```
Q*R-A
```

```
ans = 3×2
```
10^{-15} ×
```
   -0.2220         0
         0   -0.2220
         0         0
```

```
[Q_, R_] = qr(A)
```

```
Q_ = 3×3
   -0.6667    0.2357   -0.7071
   -0.3333   -0.9428   -0.0000
   -0.6667    0.2357    0.7071
R_ = 3×2
   -3.0000   -1.0000
         0    1.4142
         0         0
```

(d) $\begin{bmatrix} 4 & 8 & 1 \\ 0 & 2 & -2 \\ 3 & 6 & 7 \end{bmatrix}$

```
A = [4 8 1;0 2 -2;3 6 7];
% 计时
tic
[Q, R] = QR_Householder_reflection(A)
```

```
Q = 3×3
   -0.8000    0.0000    0.6000
         0   -1.0000    0.0000
   -0.6000   -0.0000   -0.8000
R = 3×3
   -5.0000  -10.0000   -5.0000
   -0.0000   -2.0000    2.0000
   -0.0000         0   -5.0000
```

```
disp(['QR with Householder Reflection 运行时间: ',num2str(toc)])
```

```
QR with Householder Reflection 运行时间: 0.016508
```

```
Q*R-A
```

```
ans = 3×3
10^-15 ×
   -0.4441   -0.8882         0
         0         0         0
    0.4441    0.8882    0.8882
```

```
[Q_, R_] = qr(A)
```

```
Q_ = 3×3
   -0.8000    0.0000   -0.6000
         0   -1.0000   -0.0000
   -0.6000   -0.0000    0.8000
R_ = 3×3
   -5.0000  -10.0000   -5.0000
         0   -2.0000    2.0000
         0         0    5.0000
```

```
abs(Q)-abs(Q_)
```

```
ans = 3×3
10^-15 ×
         0    0.1665         0
         0         0    0.2776
         0    0.2220         0
```

```
abs(R)-abs(R_)
```

```
ans = 3×3
10^-14 ×
         0         0         0
    0.0000         0   -0.1332
    0.0444         0    0.1776
```

From the above results, we can observe that the results obtained from the QR factorization with Householder reflection in my own implementation are the nearly same as the Matlab qr(A, 0) function, except the signs of the entries of *Q* and *R* and some residual errors due to calculation precision.