HW3, Dept.: 수리과학과, NAME: 국윤범

Problem 1 - Exercise 8.2

First, I implement “mgs.m” like below.

```
function [Q,R] = mgs(A)
% Input : m*n matrix A
% Output : [Q, R] where m*n matrix Q and n*n matrix R

[m ,n] = size(A);
Q = A;
R = zeros(n);

for i=1:n
    vi = Q(:,i);
    R(i,i) = norm(vi);
    Q(:,i) = vi / R(i,i);
    for j=i+1:n
        vj = Q(:,j);
       R(i,j) = Q(:,i)' * vj;
       Q(:,j) = vj - R(i,j) * Q(:,i);
    end
end
end
```

Then, I test the above function for two matrices and check Q*R = A.

```
%%%%% Problem 1 %%%%%
% Test mymgs
A1 = eye(3);
A2 = rand(3,2);

[q1,r1] = mgs(A1)
A1-q1*r1

[q2,r2] = mgs(A2)
A2-q2*r2
```

Below is the result of the above code. We can see Q*R completely matches with the original.

```
q1 =

     1     0     0
     0     1     0
     0     0     1

q2 =

    0.7202   -0.3493
    0.2851    0.9367
    0.6325   -0.0246

r1 =

     1     0     0
     0     1     0
     0     0     1

r2 =

    1.2285    0.9028
         0    0.7346

ans =

     0     0     0
     0     0     0
     0     0     0

ans =

     0     0
     0     0
     0     0
```

Problem 2 - Exercise 10.2

The following are two functions “house” and “formQ”

```
function [W, R] = house(A)
% Input : m*n matrix A
% Output : m*n lower triangular matrix W & n*n triangular matrix R

[m, n] =size(A);
W = zeros(m, n);

for k=1:n
   tmp = eye(m-k+1); e1 = tmp(:,1);
   x = A(k:m,k);
   vk = sign(x(1)) * norm(x) * e1 + x;
   W(k:m, k) = vk / norm(vk);

   A(k:m, k:n) = A(k:m, k:n) - 2 * W(k:m,k) * (W(k:m,k)'* A(k:m, k:n));
end
R = A(1:n,:);
end
```

```
function Q = formQ(W)
% Input : m*n matrix W produced by house
% Output : m*m orthogonal matrix Q

% Use Algorithm 10.3 for x = e_1, ..., e_n
[m, n] = size(W);
Q = eye(m);
for k=n:-1:1
    for i=1:m
       Q(k:m,i) = Q(k:m,i) - 2 * W(k:m,k) * (W(k:m,k)' * Q(k:m,i));
    end
end
end
```

I check the result of house, formQ to that of the built-in QR function “qr”.

```
%%%%% Problem 2 %%%%%
% Test my house (compare with bulit-in qr function)

A1 = rand(4,3);

% (a)
[Wh, Rh] = house (A1);
[Qm, Rm] = qr (A1);
Rh
Rm

% (b)
Qh = formQ(Wh)
Qm
```

```
Rh =

   -1.5279   -0.7451   -1.6759
    0.0000    0.4805    0.0534
         0   -0.0000    0.0580


Rm =

   -1.5279   -0.7451   -1.6759
         0    0.4805    0.0534
         0         0    0.0580
         0         0         0


Qh =

   -0.5332    0.4892    0.6519    0.2267
   -0.5928   -0.7162    0.1668   -0.3284
   -0.0831    0.4507   -0.0991   -0.8833
   -0.5978    0.2112   -0.7331    0.2462


Qm =

   -0.5332    0.4892    0.6519    0.2267
   -0.5928   -0.7162    0.1668   -0.3284
   -0.0831    0.4507   -0.0991   -0.8833
   -0.5978    0.2112   -0.7331    0.2462
```

As seen above, the results of QR factorization from “house” and “qr” are the same.

Problem 3 - Fig 9.1

First of all, I additionally implement classic Gram-schmidt method for comparison with other methods.

```
function [Q, R] = clgs(A)
% Input : m*n matrix A
% Output : [Q, R] where m*n matrix Q and n*n matrix R

[m ,n] = size(A);
Q = zeros(m,n);
R = zeros(n);

for j=1:n
    vj = A(:,j);
    for i=1:j-1
        R(i,j) = Q(:,i)' * A(:,j);
        vj = vj - R(i,j) * Q(:,i);
    end
    R(j,j) = norm(vj);
    Q(:,j) = vj / R(j,j);
end
end
```

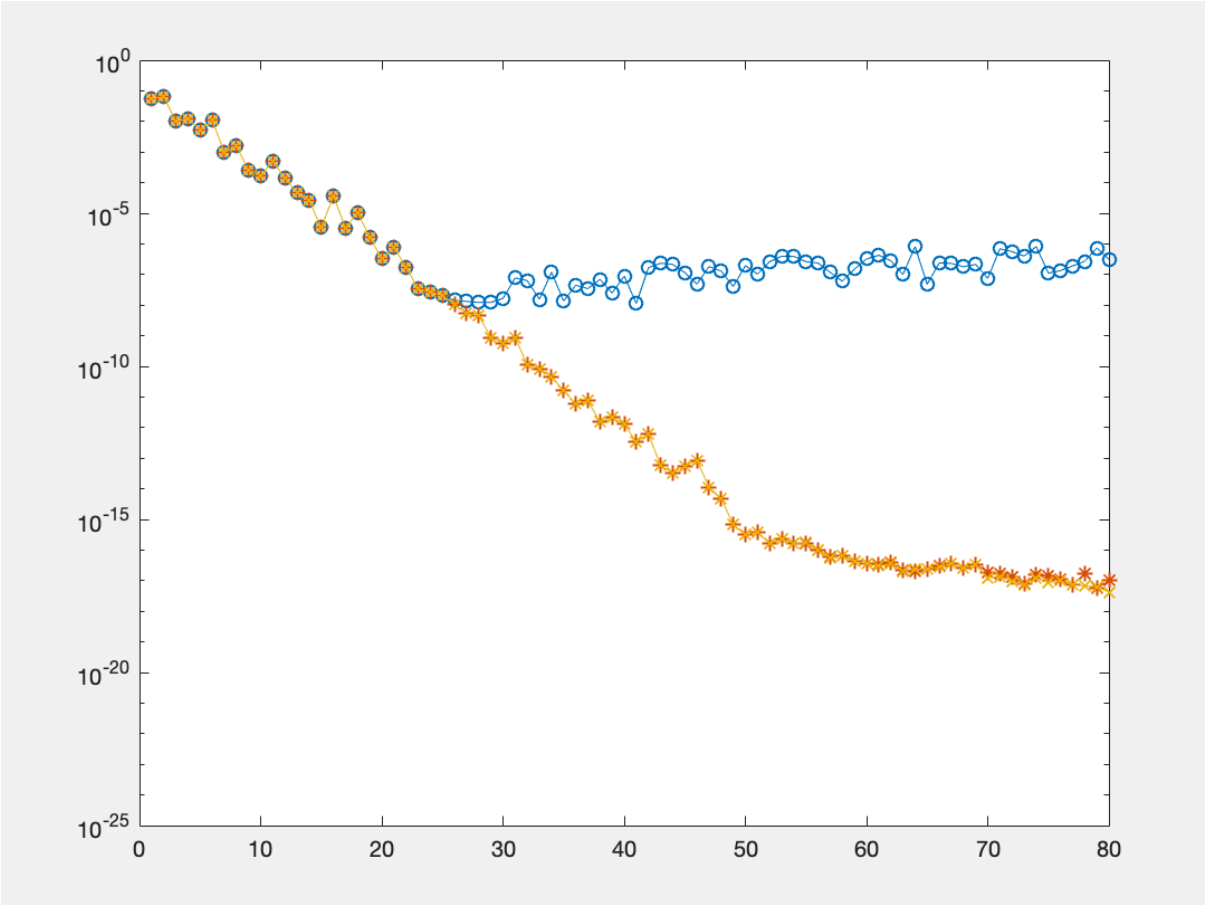
Then, plot diagonal entries r_{ii} of R from each methods. Since house method doesn't guarantee that the diagonal entries are nonnegative, so I plot absolute values of the entries just comparison with the other methods.

```
%%%%% Problem 3 %%%%%

[U,X] = qr(rand(80)); [V,X] = qr(rand(80));
S = diag(2.^(-1:-1:-80));
A = U*S*V;

[QC, RC] = clgs(A); rc = diag(RC);
[QM, RM] = mgs(A); rm = diag(RM);
[WH, RH] = house(A); rh = diag(abs(RH));

semilogy(rc, '-o');
ylim([1E-25 1]);
hold on;
semilogy(rm,'-*');
semilogy(rh,'-x');
hold off;
```



Circle = Classic Gram-schmidt

* = Modified Gram-schmidt

X = Household method

From the result above, mgs and house show almost same performance, which is significantly better than clgs. Hence, we can say house and mgs are the most stable methods.