```
format long e;
```

Question 1

Using a random matrix to be orthonormalized by CGS, cgs(V) is defined in the end and we compare with matlab QR.

```
n = 5;
m = 4;
V = randn(n,m);

[Q,R] = cgs(V);
[Q_matlab,R_matlab] = qr(V,0);
```

Q and R obtained are as follows

```
Q
```

```
Q = 5×4
     1.245351810891765e-01    -5.972066574795062e-01    -6.073743057305413e-01 ···
     8.892988065121665e-01    -6.723769389220595e-02    -7.468357431315895e-02
    -2.516023700070126e-03     3.826326009368674e-01    -3.038226754159869e-02
     1.052026615829701e-01     6.860815969442283e-01    -6.233901047431105e-01
    -4.272758956783337e-01    -1.473351139985281e-01    -4.857782535206349e-01
```

```
R
```

```
R = 4×4
     8.850418673616114e-01     1.406541382993273e-01     4.113537355541331e-01 ···
                         0     2.512015699849316e+00     1.118203340181799e+00
                         0                         0     1.664134059853937e+00
                         0                         0                         0
```

Checking the difference in norms

```
norm(V - Q*R)
```

```
ans =
     2.289773196936369e-16
```

```
norm(Q*R - Q_matlab*R_matlab)
```

```
ans =
     1.531236527388275e-15
```

```
norm(eye(m) - Q'*Q)
```

```
ans =
     5.964509741611854e-16
```

The norms are small, so the CGS algorithm is equivalent to the QR decomposition of the matrix for a random matrix. Q here is orthogonal matrix.

We check the instability of CGS for hilbert matrix.

```
% hilb matrix
```

```
V = hilb(14);

[Q, R] = cgs(V);
[Q_matlab, R_matlab] = qr(V);

norm(eye(14) - Q'*Q)
```

```
ans =
     6.998568323542483e+00
```

```
norm(eye(14) - Q_matlab'*Q_matlab)
```

```
ans =
     8.156161935422079e-16
```

Question 2

mgs(V) is defined in the end by slightly modifying cgs(V).

```
n = 5;
m = 4;
V = randn(n,m);

[Q,R] = mgs(V);
[Q_matlab,R_matlab] = qr(V,0);
```

Q and R obtained are as follows

```
Q
```

```
Q = 5×4
     2.500102545777628e-01     1.341616507191819e-01     6.443636994411766e-01 ⋯
     1.085834675150164e-01    -2.073371362133280e-01     5.939279254901677e-01
    -5.590381113180657e-01     6.380308559282208e-01    -4.538954518450944e-02
     6.111624421804116e-01    -1.845299511066945e-02    -4.731695976111240e-01
     4.895522061541372e-01     7.291008690696422e-01     7.807280736528614e-02
```

```
R
```

```
R = 4×4
     1.271353527464341e+00     1.161413143389559e+00    -6.072641204284448e-01 ⋯
                         0     2.661085988085706e+00    -3.102783487872157e-01
                         0                         0     1.950620981490806e+00
                         0                         0                         0
```

Checking the difference in norms

```
% hilb matrix
norm(V - Q*R)
```

```
ans =
     2.366564341261224e-16
```

```
norm(Q*R - Q_matlab*R_matlab)
```

```
ans =
     1.031488772583170e-15
```

```
norm(eye(m) - Q'*Q)
```

```
ans =
     2.535444519466362e-15
```

The norms are small, so the MGS algorithm is equivalent to the QR decomposition of the matrix for a random matrix. Q here is orthogonal matrix.

We check the instability of MGS for hilbert matrix.

```
V = hilb(14);

[Q, R] = mgs(V);
[Q_matlab, R_matlab] = qr(V);

norm(eye(14) - Q'*Q)
```

```
ans =
     9.952960934174319e-01
```

```
norm(eye(14) - Q_matlab'*Q_matlab)
```

```
ans =
     8.156161935422079e-16
```

Question 3

mgsrep(V) is defined in the end, this performs modified gram schmidt with reorthogonalization. This basically means projecting the new basis vector onto the previously computed basis vectors.

```
n = 5;
m = 4;
V = randn(n,m);

[Q, R] = mgsrep(V);
[Q_matlab, R_matlab] = qr(V,0);
```

Q and R obtained are

```
Q
```

```
Q = 5×4
    -3.498151735326320e-01    -5.477449690431455e-02    -9.068355340959107e-01 ···
    -1.374952861756724e-01     7.312577984501045e-01    -1.473229577115254e-01
     3.498209069520731e-01    -2.386491224620673e-02    -8.053005820763216e-02
    -8.141313403245912e-01    -3.113838747760392e-01     2.978164002817654e-01
     2.711823821827634e-01    -6.039307370781242e-01    -2.465067173129355e-01
```

```
R
```

```
R = 4×4
     1.170125690948300e+00    -4.921286393512929e-01    -2.975718841800664e-01 ···
```

```
0    1.718573236033834e+00    6.911406140348373e-01
0                        0    1.600622063758596e+00
0                        0                        0
```

Checking the difference in norms

```
norm(V - Q*R)
```

```
ans =
     6.798699777552591e-17
```

```
norm(Q*R - Q_matlab*R_matlab)
```

```
ans =
     1.646596704845735e-15
```

```
norm(eye(m) - Q'*Q)
```

```
ans =
     4.480316447770626e-16
```

By the calculated norms, it can be observed that the mgsrep algorithm is giving the correct QR decomposition for a random matrix.

The calculated Q is indeed an orthogonal matrix.

We check the stability for hilbert matrix

```
% hilb matrix
V = hilb(14);

[Q, R] = mgsrep(V);
[Q_matlab, R_matlab] = qr(V);

norm(eye(14) - Q'*Q)
```

```
ans =
     3.939995737970531e-16
```

```
norm(eye(14) - Q_matlab'*Q_matlab)
```

```
ans =
     8.156161935422079e-16
```

We observe that Q is orthogonal for hilbert matrix with the mgsrep routine. Which performs better than the mgs and cgs routines.

Question 4 and 5

Reflect(x) and applreflect(u,gamma,x) is defined in the end. We check the working of reflect and applreflect and choosing sign of tau to avoid catastrophic cancellation.

```
n = 7;
x = randn(n,1);

[u, gamma, tau] = reflect(x)
```

```
u = 7×1
     1.000000000000000e+00
     1.612149548621438e-01
    -3.249656482640047e-01
    -4.446064317495189e-01
    -3.184715185050940e-01
    -4.106901505701231e-01
     1.728034680060622e-02
gamma =
     1.250268079640646e+00
tau =
     2.757250940066485e+00
```

```
y = applreflect(u,gamma,x);
```

Checking the maximum error (which should be of unit round off)

```
max(abs(y(2:end)))
```

```
ans =
     2.220446049250313e-16
```

```
[y(1) norm(x) x(1)]
```

```
ans = 1×3
    -2.757250940066485e+00     2.757250940066485e+00     6.900518978578063e-01
```

The absolute maximum error in entries of y is of the order of unit roundoff. The first element's (of y) sign is chosen to avoid catastrophic cancellation - opposite to the sign of first element of x and the first element of y has magnitude norm(x)

Question 6 and 7

Function reflectqr is defined at the end. We test the output of reflectqr for different randomly generated matrices.

```
N = 6:2:12;
m = 6;

for n = N
    fprintf('For n = %d and m = %d\n', n, m);
    A = randn(n,m);
    [Q,R] = reflectqr(A);
    [Qhat, Rhat] = qr(A,0);

    norm(Q*R - A)
    norm(Q'*Q - eye(m))
    norm(tril(R,-1))
    norm(R - Rhat)
    norm(Q - Qhat)
end
```

```
For n = 6 and m = 6
ans =
     1.329149084824830e-15
```

```
ans =
     1.083762340556522e-15
ans =
     0
ans =
     1.393321790493597e-15
ans =
     1.360523572706916e-15
For n = 8 and m = 6
ans =
     2.800115375619830e-15
ans =
     8.213224151494108e-16
ans =
     0
ans =
     1.676925721949480e-15
ans =
     1.223352457055724e-15
For n = 10 and m = 6
ans =
     1.674038307904540e-15
ans =
     5.044094373763046e-16
ans =
     0
ans =
     1.028864638660267e-15
ans =
     5.855292811951112e-16
For n = 12 and m = 6
ans =
     1.488459885949748e-15
ans =
     5.505132421595251e-16
ans =
     0
ans =
     1.609113524843091e-15
ans =
     9.204383474385828e-16
```

The norms are of unit roundoff so the function seems to be working as expected.

Helper functions

```
function [Q, R] = cgs(V)
   [~,m] = size(V);
   R = zeros(m,m);

   for k = 1:m
      R(1:k-1,k) = V(:,k)' * V(:,1:k-1);
      V(:,k) = V(:,k) - V(:,1:k-1)*R(1:k-1,k);

      R(k,k) = norm(V(:,k),2);
      if R(k,k) == 0
         error('Columns of V are linearly dependent');
```

```
      end
      V(:,k) = V(:,k) / R(k,k);
   end

   Q = V;
end

function [Q, R] = mgs(V)
   [~,m] = size(V);
   R = zeros(m,m);

   for k = 1:m
      for i = 1:k-1
         R(i,k) = V(:,k)'*V(:,i);
         V(:,k) = V(:,k) - V(:,i)*R(i,k);
      end
      R(k,k) = norm(V(:,k), 2);
      if R(k,k) == 0
         error('Columns of V are linearly dependent');
      end
      V(:,k) = V(:,k) / R(k,k);
   end

   Q = V;
end

function [Q, R] = mgsrep(V)
   [~,m] = size(V);
   R = zeros(m,m);

   for k = 1:m
      for i = 1:k-1
         R(i,k) = V(:,k)'*V(:,i);
         V(:,k) = V(:,k) - V(:,i)*R(i,k);
      end
      % Reorthogonalization
      for i = 1:k-1
         alpha = V(:,k)'*V(:,i);
         V(:,k) = V(:,k)-V(:,i)*alpha;
         R(i,k) = R(i,k)+alpha;
      end
      R(k,k) = norm(V(:,k),2);
      if R(k,k) == 0
         error('Columns of V are linearly dependent') ;
      end
      V(:,k) = V(:,k) / R(k,k);
   end

   Q = V;
end
```

```
function B = applreflect(u, gamma, A)
    B = A - (gamma*u)*(u'*A);
end

function [u, gamma, tau] = reflect(x)
    [~,m] = size(x);

    maxi = max(abs(x));
    if maxi == 0
        gamma = 0;
        tau = 0;
    else
        x = x ./ maxi;
        tau = norm(x,2);
        % Choosing the sign
        if x(1) < 0
            tau = -tau;
        end

        x(1) = x(1) + tau;
        gamma = x(1,1) / tau;
        x(2 : end) = x(2 : end) / x(1);
        x(1) = 1;
        tau = tau * maxi;
    end

    u = x;
end

function [Q, R] = reflectqr(A)
    [n, m] = size(A);
    gamma_val = zeros(n,1);
    Q = eye(n);

    cols = m;
    if m == n
        cols = m-1;
    end

    for k = 1:cols
        [u, gamma, tau] = reflect(A(k:n,k));
        A(k:n,k+1:m) = applreflect(u, gamma, A(k:n,k+1:m));
        A(k,k) = -tau;
        A(k+1:n,k) = u(2:end);
        gamma_val(k) = gamma;
    end

    R = triu(A);
```

check whether A(k+1:n,k) = 0.

```
    for k = 1 : cols
        Q_k = eye(n);
        u = A(k:n,k);
        u(1) = 1;

        Q_k(k:n,k:n) = eye(n-k+1)-gamma_val(k)*(u*u');
        Q = Q*Q_k;
    end
    Q = Q(:,1:m);
    R = R(1:m,:);
end
```

Computation of Q takes more flops than required.