1. (a) Determine the eigenvalues, determinant, and singular values of a Householder reflector $H = I - 2\frac{vv^T}{v^Tv}$. For the eigenvalues, give a geometric argument as well as an algebraic proof.

Solution. Let us begin with a geometric interpretation of the Householder transformation. We would like to find a transformation that maps x to $\|x_1\|e_1$. Consider the vector $v = \|x_1\|e_1 - x$. The projection of x onto the hyperplane orthogonal to the vector v is $\text{proj}(x) = x - v\left(\frac{v^Tx}{v^Tv}\right)$. However, if we are to project to $\|x_1\|e_1$, we need to go twice this distance. Thus, our Householder reflection is characterized by $x - 2v\left(\frac{v^Tx}{v^Tv}\right)$ and we say that our Householder reflect $H = (I - \frac{vv^T}{v^Tv})$. This geometric interpretation immediately motivates many properties such as symmetry and orthogonality. One can also see eigenvalues from this interpretation. Since our projection is a reflection across a hyperplane (of dimension $m-1$), all but 1 direction is fixed. That is, $\lambda_i = 1$ for $\lambda = 1, \dots, m-1$ and $\lambda_m = -1$.

For the algebraic solutions, note that if $y \perp v$. Then $Hy = y - 2\frac{v^Ty}{v^Tv} = y - 2 \cdot 0 = y$. Since $\dim\text{span}(y) = 1$, $\dim\text{span}(y)^\perp = m-1$. Thus, $\lambda = 1$ is an eigenvalue with multiplicity $m-1$. Noting that $Hv = -v$ shows that the remaining eigenvalue is -1. The determinant is simply the product of all of these eigenvalues. Thus, $\det H = 1^{m-1} \cdot -1 = -1$. Since H is symmetric (because vv^T is symmetric), we know that the singular values are the absolute value of the eigenvalues. That is, $\sigma_i = 1$ for all $i = 1, \dots, m$.

(b) Consider the Givens rotation $G = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$. Given a geometric interpretation of the action of G on a vector in $\mathbb{R}^2$. Do the same analysis as part (a) for G, but no geometric interpretation is needed for eigenvalues.

Solution. Recall that the matrix $R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$ rotates a vector in $\mathbb{R}^2$ counterclockwise by an angle of θ. Substituting $\theta = -\theta$, then we get the Givens matrix $G = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$. If θ is the angle between a point x and the x-axis, then Gx rotates x to the x-axis. We accomplish this by setting θ to be the angle such that $\cos(\theta) = \frac{x_1}{\sqrt{x_1^2+x_2^2}}$ and $\sin(\theta) = \frac{x_2}{\sqrt{x_1^2+x_2^2}}$

Algebraically, we can compute the eigenvalues as the roots of the equation

$$\cos^2(\theta) + \lambda^2 - 2\lambda\cos(\theta) + \sin^2(\theta) = 0$$

Letting $x = (a, b)^T$, this becomes

$$1 + \lambda - \frac{2\lambda a}{\sqrt{a^2+b^2}} = 0$$

after substituting θ. By the quadratic formula, we obtain

$$\lambda_{1,2} = \frac{\frac{2a}{\sqrt{a^2+b^2}} \pm \sqrt{\frac{2a}{\sqrt{a^2+b^2}} - 4}}{2} = \frac{2a}{\sqrt{a^2+b^2}} \pm \frac{2b}{\sqrt{a^2+b^2}}i = \cos(\theta) \pm i\sin(\theta)$$

as our eigenvalues. The singular values can be found through construction of a singular value decomposition. First note that for any α, β, the vectors $u = (-\alpha, \beta)^T, v = (\beta, \alpha)^T$ are orthogonal since

$$\begin{bmatrix} -\alpha & \beta \end{bmatrix} \begin{bmatrix} \beta \\ \alpha \end{bmatrix} = -\alpha\beta + \beta\alpha = 0$$

Thus,

$$\begin{bmatrix} -\cos(\theta) & \sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} I_2 = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} = G$$

is a valid SVD for G. From this decomposition, we immediately read off $\sigma_{1,2} = \pm 1$.

2. Implement QR factorization in MATLAB based on

   - classical Gram-Schmidt (CGS)
   - modified Gram-Schmidt (MGS)
   - MGS with double orthogonalization
   - Householder reflectors

   Then we construct three matrix as follows.

```
A1 = randn(2^n20,15);
u = (-1:2/40:1)';
A2 = u.^(0:23);
A3 = u.^(0:40);
```

   For each matrix, run the algorithms, then compute $\frac{\|A-QR\|_F}{\|A\|_F}$ and $\|Q^TQ - I_n\|$. Draw conclusions about the backward stability of these algorithms, and the orthogonality of the computed Q factors, probably related to the condition numbers of the matrices.

   Solution. Figure 1 shows the errors $\|Q^TQ - I_n\|_2$ for each different algorithm as a function of the condition number of matrix A. I modified the code to produce 10 different matrices of increasing condition numbers. The condition numbers of these matrices are[1]

```
6.805817606385450e+00
5.137698938981403e+02
1.667468568998088e+04
6.117931220627653e+05
2.622543104797633e+07
1.387410733558138e+09
9.876123847277121e+10
1.097758126019078e+13
2.596372220698814e+15
3.498715070665783e+17
```

Algorithm	**Forward Error of A with large condition number**
CGS	$3.76e-16$
MGS	$6.36e-16$
MGS2	$3.49e-17$
Householder	$-1.72e-16$

Table 1: Forward Errors of QR Factorization

respectively. One can see from Figure 1 that MGS2 and Householder factorizations reliably produce orthogonal Q factors, even for matrices of incredibly large condition numbers. On the other hand, CGS and MGS fail to do. CGS shows its numerical instability by how much more quickly it produces completely unusable Q factors. However as shown in Table 1, all the algorithms produce respectable forward errors.

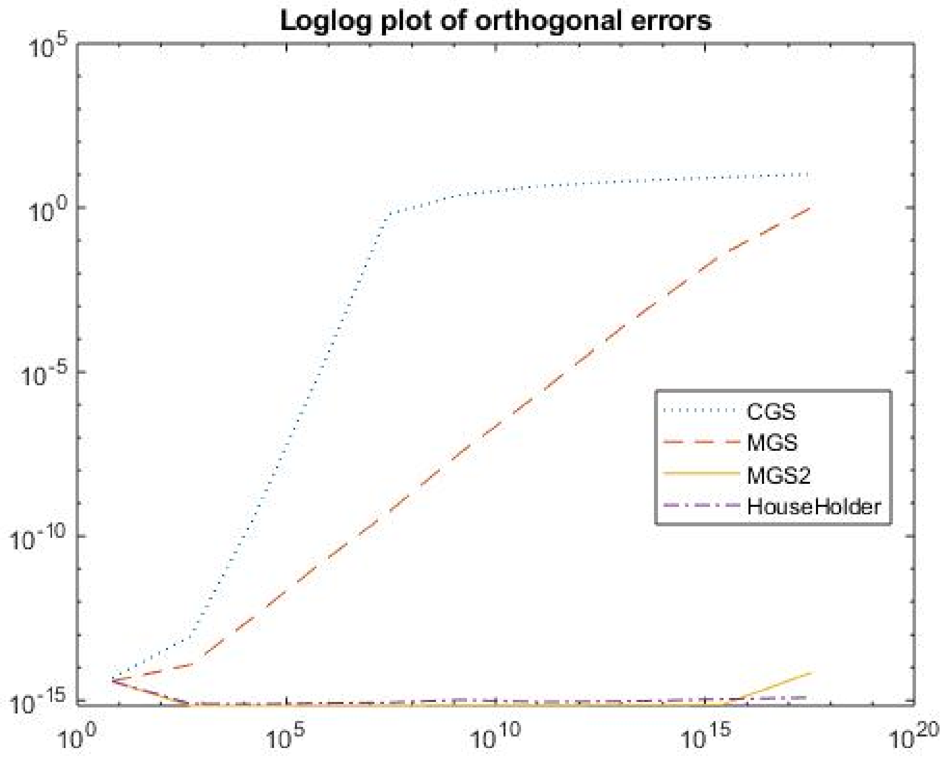


Figure 1: Factorization Errors

3. Evaluate the arithmetic work needed to retrieve the reduced factor Q_L from Householder and Givens reduction of A to R, respectively. Compare that with the cost for the first phase.

[1]Got lazy and just verbatim'd them

Solution. Recall that the full QR factorization in the first phase produces an R factor used in the QR factorization. However, for a reduced QR factorization, note that

$$A = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = [Q_L, Q_R] \begin{bmatrix} R \\ 0 \end{bmatrix} = Q_L R$$

Thus, it suffices to compute only Q_L. We can do this by applying $Q_n, \ldots Q_1$ (or $Q_n^T, \ldots, Q_1^T$ in the case of Givens) to the left of I_n. However, this almost the exact same cost as the phase 1 calculations. The only difference is that in phase 1, we applied the transformations to A instead of I_n. Thus, the cost of retrieving our Q_L is roughly the same, most definitely in the asymptotic sense, as phase 1. That is, explicitly computing Q_L is something that we do not want to do when we can avoid it as it doubles the computational cost.

4. Implement the algorithm for solving linear system $Ax = b$ or linear least squares problem $\min\|b - Ax\|_2$ based on Householder QR. Make sure that the reduced Q factor is not formed explicitly to save the cost of the second phase. Then, solve the linear least squares problem $\min\|b - Ax\|_2$ where $A = A_2$ and the linear system $Ax = b$, where $A = A_3$ in [Q2], and $b = [1, -1, 1, -1, \ldots,]^T$. Report $\frac{\|b - A\hat{x}\|_2}{\|A\|_2\|\hat{x}\|_2}$ for both solves, and compare this quantity associated with the solutions obtained by MATLAB's backslash.

   Solution. Rather than just run the experiments on two matrices, I solved linear least squares on $A = u.\hat{}(0 : 4i)$ where i ranged from 1 to 10. Figure 2 shows the differences in backward errors for both my solutions and MATLAB's backslash. I show this as a scatter plot because it appears as if my algorithm outperforms MATLAB's in almost every case[2]. However, after looking at the scaling, this is in fact a semilog plot, one with a very small scale. That is, the differences between the two backward errors are around machine precision. Furthermore, Figure 3 shows us that despite performing on par with MATLAB's backslash, my code has a long way to go before it is comparable in speed. Nonetheless, both exhibit backward errors on the magnitude of machine precision for solving linear least square problems.

5. Suppose the $m \times n$ matrix A has the form

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

   where A_1 is nonsingular matrix of dimension $n \times n$ and A_2 is an arbitrary matrix of appropriate dimension. Prove that $\|A^+\|_2 \leq \|A_1^{-1}\|_2$.

   Proof. Let $A = QR$ be a reduced QR factorization of A. Note that

$$\begin{aligned} \|A^\dagger\| = \|(A^T A)^{-1} A^T\| &= \|(R^T R)^{-1} R^T Q^T\| \\ &= \|R^{-1} Q^T\| \end{aligned}$$

[2]This isn't entirely true. I can generate plots in MATLAB, but it is much harder to generate tables. This is also the faster option to display data

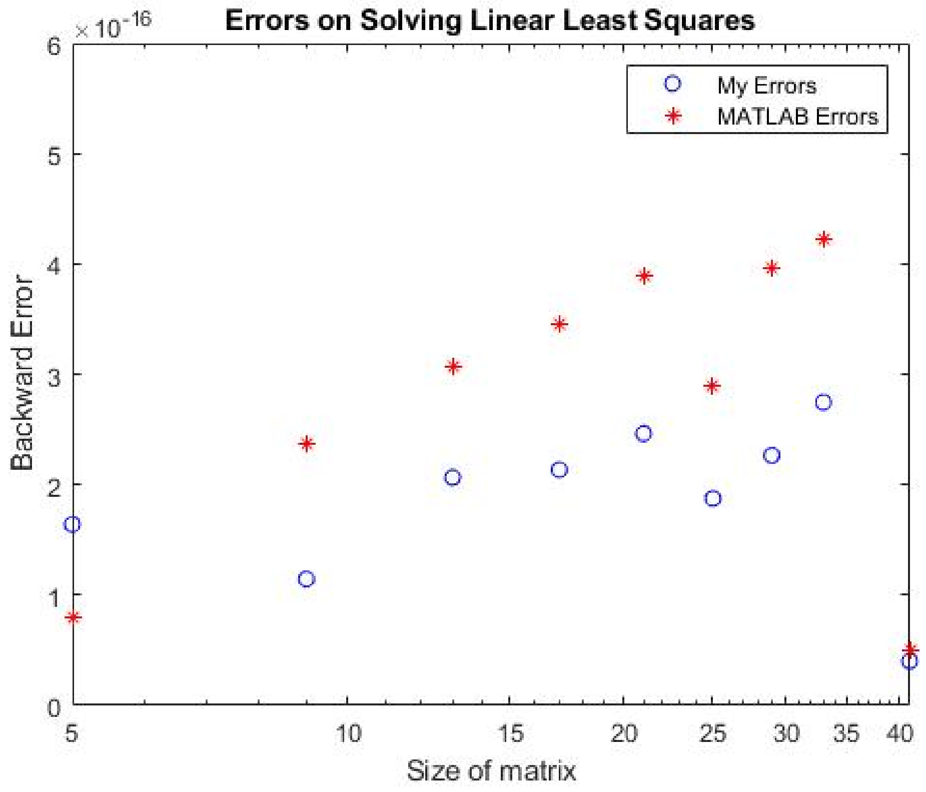
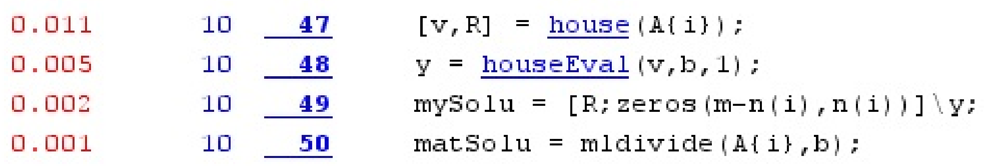


Figure 2: Backward Errors



```
0.011     10   47     [v,R] = house(A{i});
0.005     10   48     y = houseEval(v,b,1);
0.002     10   49     mySolu = [R;zeros(m-n(i),n(i))]\y;
0.001     10   50     matSolu = mldivide(A{i},b);
```

Figure 3: Execution Time

Now decompose Q into Q_1, Q_2 of the same size as the decomposition of A. That is,

$$\begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R$$

From the above, we conclude that $Q_1 R = A_1$ or equivalently, $R^{-1} = A_1^{-1} Q_1$. Substituting this into our previous equation,

$$\|A^{\dagger}\| = \|A_1^{-1} Q_1 Q^T\| \leq \|A_1^{-1}\| \|Q_1\| \|Q^T\|$$

We know that Q is orthogonal, so it only suffices to show that $\|Q_1\| \leq 1$. Let $x \in \mathbb{R}^n$ and suppose that $Q_1 x = z$. Then

$$Qx = \begin{bmatrix} z \\ a \end{bmatrix}$$

for some $a \in \mathbb{R}$. Thus, $\|Q_1 x\| \leq \|Qx\| \leq \|x\|$[3] and the inequality follows. $\square$

6. Take $m = 50, n = 12$. Using MATLAB's linspace, define t to be the m vector corresponding to linearly spaced grid points from 0 to 1. Using MATLAB's vander and fliplr, define A to be the $m \times n$ polynomial of degree $n-1$. Take b to be the function $\cos(4t)$ evaluated on the grid. Now calculate and print the least squares coefficient vector x by six methods.

   (a) Normal equations (using MATLAB's \)
   (b) QR factorization by MGS
   (c) QR factorization by Householder transformations
   (d) QR factorization computed by MATLAB's qr
   (e) $x = A\backslash b$ in MATLAB
   (f) SVD, using MATLABs svd

   The results are summarized in Table 2. To generate the table, I assumed that the solution $x = A\backslash b$ computed using MATLAB's backslash operator was the true solution. Each vector of coefficients were then compared to the solution generated with the backslash, and then normed. The results are a bit unsurprising. The normal equations and MGS look to be numerically unstable while Householder, MATLAB QR factorization, and MATLAB SVD all seem to be relatively close to the true solution. I am unsure as to how we only managed to get the square root of machine precision, but it is likely due to the fact that $x = A\backslash b$ is, in fact, not the true solution.

Algorithm	**Norm of Errors**
Normal Eq	$1.06e-1$
MGS	$1.526e-1$
Householder	$2.85e-8$
MATLAB QR	$3.31e-8$
MATLAB SVD	$3.20e-8$

Table 2: Norm of Coefficient Errors

[3]Note that these norms are not the same dimension wise

1 Appendix

1.1 Script files

1.1.1 Question 2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Spring 2018 Math 8610 w/ Xue
%   Homework 2
%
% Problem
%   2
%
% Function Dependencies
%   qrcgs.m
%   qrmgs.m
%   qrmgs2.m
%   house.m
%   formQ.m
%   houseEval.m
%
% Notes
%   None
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc
close all;

n = [15 9:4:41];
matrixCount = 10;
A = cell(matrixCount,1);
A{1} = rand(2^20,n(1));
u = (-1:2/40:1)';

condition= zeros(matrixCount,1);
condition(1:2) = [cond(A{1}) cond(A{2})];
for i = 2:matrixCount
    A{i} = u.^(0:4*i);
    condition(i) = cond(A{i});
end
```

```
cgs.forErr = zeros(matrixCount,1);
cgs.ortho = zeros(matrixCount,1);
mgs.forErr = zeros(matrixCount,1);
mgs.ortho = zeros(matrixCount,1);
mgs2.forErr = zeros(matrixCount,1);
mgs2.ortho = zeros(matrixCount,1);
hHolder.forErr = zeros(matrixCount,1);
hHolder.ortho = zeros(matrixCount,1);


%% Classical GS
for i = 1:matrixCount
    [Q,R] = qrcgs(A{i});
    cgs.forErr(i) = norm(A{i}-Q*R,'fro')/norm(A{i},'fro');
    cgs.ortho(i) = norm(Q'*Q-eye(n(i)),2);
end

%% Modified GS
for i = 1:matrixCount
    [Q,R] = qrmgs(A{i});
    mgs.forErr(i) = norm(A{i}-Q*R,'fro')/norm(A{i},'fro');
    mgs.ortho(i) = norm(Q'*Q-eye(n(i)),2);
end

%% Double Modified GS
for i = 1:matrixCount
    [Q,R] = qrmgs2(A{i});
    mgs2.forErr(i) = norm(A{i}-Q*R,'fro')/norm(A{i},'fro');
    mgs2.ortho(i) = norm(Q'*Q-eye(n(i)),2);
end

%% Householder Transformation
for i = 1:matrixCount
    [v,R] = house(A{i});
    Q = formQ(v);
    hHolder.forErr(i) = norm(A{i}-Q*R,'fro')/norm(A{i},'fro');
    hHolder.ortho(i) = norm(Q'*Q-eye(n(i)),2);
end

save('HW2Q2','cgs','mgs','mgs2','hHolder','condition')

%% Analysis
load HW2Q2

loglog(condition,cgs.ortho,':')
hold on
```

```
loglog(condition,mgs.ortho,'--')
hold on
loglog(condition,mgs2.ortho,'-')
hold on
loglog(condition,hHolder.ortho,'-.')
title('Loglog plot of orthogonal errors')

legend('CGS','MGS','MGS2','HouseHolder','location','best')
```

1.1.2 Question 4

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Spring 2018 Math 8610 w/ Xue
%   Homework 2
%
% Problem
%   4
%
% Function Dependencies
%   house.m
%   formQ.m
%   houseEval.m
%
% Notes
%   None
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc
close all;

m = 41;
n = 5:4:m;

matrixCount = 10;
A = cell(matrixCount,1);
u = (-1:2/40:1)';
condition= zeros(matrixCount,1);

for i = 1:matrixCount
    A{i} = u.^(0:4*i);
    condition(i) = cond(A{i});
end

myBackErr = zeros(1,matrixCount);
matBackErr = zeros(1,matrixCount);


%% Evaluation
for i = 1:matrixCount
    powers = (2:n(i)+1)';
    solu = (-1*ones(n(i),1)).^powers;
    b = A{i}*solu;

    [v,R] = house(A{i});
    y = houseEval(v,b,1);
    mySolu = [R;zeros(m-n(i),n(i))]\y;
    matSolu = mldivide(A{i},b);

    myBackErr(i) = norm(b-A{i}*mySolu,2)/norm(A{i},2)/norm(mySolu,2);
    matBackErr(i) = norm(b-A{i}*matSolu,2)/norm(A{i},2)/norm(matSolu,2);
end

figure();
semilogx(n,myBackErr,'bo',n,matBackErr,'r*')
xlabel('Size of matrix')
ylabel('Backward Error')
title('Errors on Solving Linear Least Squares')
legend('My Errors','MATLAB Errors')
```

1.1.3 Question 6

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Spring 2018 Math 8610 w/ Xue
%   Homework 2
%
% Problem
%   6 - Trefethen 11.3
%
% Function Dependencies
%   house.m
%   formQ.m
%   houseEval.m
%
% Notes
%   None
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear
clc
close all;

m = 50;
n = 12;
t = linspace(0,1,m);

A = fliplr(vander(t));
A = A(:,1:12);
b = cos(4*t)';
problem = cond(A);
error = zeros(1,5);


%% Matlab Backslash (solution)
solu = A\b;

%% Normal Equations
R = cholDecomp(A'*A);
normalCoeff = backSub(R',forwardSub(R,A'*b));
error(1) = norm(solu-normalCoeff);

%% MGS
```

```
[Q,R] = qrmgs(A);
mgsCoeff = backSub(R,Q'*b);
error(2) = norm(solu-mgsCoeff);

%% Householder
[v,R] = house(A);
y = houseEval(v,b,1);
houseCoeff = [R; zeros(m-n,n)]\y;
error(3) = norm(solu-houseCoeff);

%% Matlab QR
[Q,R] = qr(A);
matqrCoeff = R\Q'*b;
error(4) = norm(solu-matqrCoeff);

%% Matlab SVD
[U,Sigma,V] = svd(A);
matsvdCoeff = V*(Sigma\(U'*b));
error(5) = norm(solu-matsvdCoeff);
```

1.2 Accompanying Functions

1.2.1 Classical GS

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% QRCGS.m
%
% DESCRIPTION
%   Decomposes a rectangular matrix A in rectangular matrix Q and square
%   upper triangular matrix R
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   A - m x n  matrix
%
% OUTPUT
%   Q - m x n orthonormal matrix
%   R - n x n upper triangular matrix
%
% NOTES
%   Issues a warning if matrix is rank deficient
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Q,R] = qrcgs(A)

[m,n] = size(A);

Q = zeros(m,n);
R = zeros(n);

for k = 1:n
    for i = 1:k-1
        R(i,k) = Q(:,i)'*A(:,k);
    end
    tmpAk = A(:,k);
    for i = 1:k-1
        tmpAk = tmpAk - Q(:,i)*R(i,k);
    end
    R(k,k) = norm(tmpAk,2);
    Q(:,k) = tmpAk/R(k,k);
end
```

1.2.2 Modified GS

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% qrmgs.m
%
% DESCRIPTION
%   Decomposes a rectangular matrix A in rectangular matrix Q and square
%   upper triangular matrix R using modified gram schmidt
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   A - m x n  matrix
%
% OUTPUT
%   Q - m x n orthonormal matrix
%   R - n x n upper triangular matrix
%
% NOTES
%   Issues a warning if matrix is rank deficient
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Q,R] = qrmgs(A)
[~,n] = size(A);

Q = A;
R = zeros(n);
for k = 1:n
    R(k,k) = norm(Q(:,k));
    Q(:,k) = Q(:,k)/R(k,k);

    for j = k+1:n
        R(k,j) = Q(:,k)'*Q(:,j);
        Q(:,j) = Q(:,j)-R(k,j)*Q(:,k);
    end
end
```

1.2.3 Double Modified GS

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% qrmgs2.m
%
% DESCRIPTION
%   Decomposes a rectangular matrix A in rectangular matrix Q and square
%   upper triangular matrix R using modified gram schmidt
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   A - m x n  matrix
%
% OUTPUT
%   Q - m x n orthonormal matrix
%   R - n x n upper triangular matrix
%
% NOTES
%   Issues a warning if matrix is rank deficient
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Q,R] = qrmgs2(A)

[Q1,R1] = qrmgs(A);
[Q,R2] = qrmgs(Q1);
R = R2*R1;
end
```

1.2.4 HouseHolder Factorization

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% HOUSE.m
%
% DESCRIPTION
%   Decomposes a rectangular matrix A in rectangular matrix Q and square
%   upper triangular matrix R householder transformations
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   A - m x n  matrix
%
% OUTPUT
%   v - matrix of vectors corresponding to Householder transformations
%   R - n x n upper triangular matrix
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [v,R] = house(A)
[m,n] = size(A);
v = zeros(m,n);

for k = 1:n
    x = A(k:m,k);
    v(k:m,k) = sign(x(1))*norm(x,2)*eye(m-k+1,1) + x;
    v(k:m,k) = v(k:m,k)/norm(v(k:m,k),2);
    A(k:m,k:n) = A(k:m,k:n) - 2*v(k:m,k)*(v(k:m,k)'*A(k:m,k:n));
end
R = A(1:n,:);
end
```

1.2.5 Evaluate Householder

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% HOUSEEVAL.m
%
% DESCRIPTION
%   Given a matrix of v factors of house.m and a vector b, computes Q'b or
%   Qb
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   v - matrix of factors from house.m
%   b - vector in Qb or Q'b
%   transpose - boolean variable that determines which calculation to
%   perform
%
% OUTPUT
%   b - either Q'b or Qb
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [b] = houseEval(v,b,transpose)
[m,n] = size(v);

if transpose
    for k = 1:n
        b(k:m) = b(k:m) - 2*v(k:m,k)*(v(k:m,k)'*b(k:m));
    end
else
    for k = n:-1:1
       b(k:m) = b(k:m) - 2*v(k:m,k)*(v(k:m,k)'*b(k:m));
    end
end
```

1.2.6 Form Q Function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FORMQ.m
%
% DESCRIPTION
%   Given an output v from house.m, formQ(v) produces the orthogonal Q
%   matrix in the reduced QR factorization such that A = QR
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   v - vector of Householder vectors computed in house.m
%
% OUTPUT
%   Q - m x n orthogonal matrix
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Q] = formQ(v)
[m,n] = size(v);

Q = zeros(m,n);

for i = 1:n
    x = zeros(m,1);
    x(i) = 1;

    Q(:,i) = houseEval(v,x,0);
end
```