AMATH 584: Numerical Linear Algebra: Exercise 10.1
Sid Meka

Determine the eigenvalues, determinant, and singular values of a Householder reflector. For the eigenvalues, give a geometric argument as well as an algebraic proof.

Introduction to Householder Reflector: A Householder reflector is a linear transformation represented by a matrix H that reflects a vector across a hyperplane. It is given by:

$$H = I - 2\frac{vv^*}{v^*v}$$

where

- v is a nonzero vector
- I is the Identity Matrix

A property of H to keep in mind is that it is Hermitian and orthogonal meaning:

- $H^* = H$
- $HH^* = I$
- $H^*H = I$

Geometric Argument for Eigenvalues of a Householder Reflector: A Householder reflector reflects vectors across the hyperplane perpendicular to the vector v. Any vector that lies in the hyperplane, or in other words, any vector orthogonal to v is unchanged by the reflection, meaning it is an eigenvector with eigenvalue 1. The vector v itself, which is reflected in the opposite direction, has eigenvalue -1 because it flips its direction across the hyperplane.

Thus, geometrically, we can say that the eigenvalues of the Householder reflector are:

- -1 corresponding to the direction of v
- 1 for all directions orthogonal to v

Since the Householder matrix is of size $n \times n$ and reflects across an $(n-1)$ dimensional hyperplane, we have the eigenvalues:

- 1 with multiplicity $n-1$
- -1 with multiplicity 1

Algebraic Proof: We will now compute the eigenvalues of H algebraically.

Proof. Suppose $Hx = \lambda x$ meaning x is an eigenvector with eigenvalue λ. Remember,

$$H = I - 2\frac{vv^*}{v^*v}$$

So, that means we have the following equations to work with:

$$Hx = \lambda x$$

$$Hx = \left(I - 2\frac{vv^*}{v^*v}\right) x$$

$$\left(I - 2\frac{vv^*}{v^*v}\right) x = \lambda x$$

From the following equations, we can say:

$$Ix - 2\frac{vv^*}{v^*v}x = \lambda x$$

$$x - 2\frac{vv^*}{v^*v}x = \lambda x$$

$$v^*x - 2v^*\frac{vv^*}{v^*v}x = \lambda v^*x$$

Let's define $\hat{v}$ as $\hat{v} = \frac{v}{||v||}$, where $||\hat{v}|| = 1$. We can continue:

$$v^*x - 2v^*\hat{v}\hat{v}^*x = \lambda v^*x$$

We have defined v as a nonzero

$$x - 2\hat{v}\hat{v}^*x = \lambda x$$

Next, we apply $\hat{v}\hat{v}^*$. We will define μ such that $\lambda_i = 1 - 2\mu_i$. We have $\mu_1 \neq 0$, but we have $\mu_j = 0$, where $j = 2, \ldots, n$. This means $\hat{v}\hat{v}^*x = \mu_1\hat{v}$ and where $j = 2, \ldots, n$, $\hat{v}\hat{v}^*x = 0$.
When $\mu \neq 0$, such when we have μ_1, we have that

$$\hat{v}\hat{v}^*\vec{u} = \mu_1\vec{u}$$

$$\hat{v}^*\hat{v}\hat{v}^*\vec{u} = \hat{v}^*\mu_1\vec{u}$$

$$||\hat{v}|| = 1, \text{ so } \hat{v}^*\vec{u} = \hat{v}^*\mu_1\vec{u}$$

$$\lambda_1 = -1$$

In the case that $\mu = 0$, we have defined μ_j to be $j = 2, \ldots, n$. Thus,

$$\lambda_j = 1 - 2(\mu_j) \text{ for } j = 2, \ldots, n$$

$$\lambda_j = 1 - 2(0) \text{ for } j = 2, \ldots, n$$

$$\lambda_j = 1 \text{ for } j = 2, \ldots, n$$

Thus, we have that $\lambda_1 = -1$ and that $\lambda_j = 1$ for $j = 2, \ldots, n$ for our eigenvalues. Thus, we have the eigenvalues:

- 1 with multiplicity $n - 1$
- -1 with multiplicity 1

□

Determinant of a Householder Reflector: Note that the determinant of a matrix is equal to the product of the matrix's eigenvalues. Thus,

$$\det(H) = \prod_{i=1}^{n} \lambda_i$$

That means:

$$\det(H) = \underbrace{(-1)}_{\text{as this eigenvalue has a multiplicity 1}} \times \underbrace{1 \times 1 \times \ldots \times 1}_{\text{multiplied } n-1 \text{ times as the eigenvalue 1 has a multiplicity } n-1}$$

$$\det(H) = (-1) \times 1^{n-1}$$
$$\det(H) = -1$$

Singular Values: Notice that H is orthogonal meaning that $H^*H = I$ and $HH^* = I$. Thus, the square roots of the eigenvalues of H^*H are all 1 meaning that the singular values of H are all 1.
Thus, the singular values of a Householder reflector are:

$$\underbrace{1, 1, \ldots, 1}_{\text{with multiplicity } n}$$

```
% Parameters
m = 50;   % Number of grid points
n = 12;   % Polynomial degree + 1
% Define t as an m-vector of linearly spaced points from 0 to 1
t = linspace(0, 1, m)';   % The grid, column vector
% Define the Vandermonde matrix A, flipping columns to match polynomial form
A = fliplr(vander(t));    % This gives an m x n matrix (flipped Vandermonde)
% Resize A to be m x n (since vander produces m x m by default)
A = A(:, 1:n);
% Define the vector b as cos(4t) evaluated on the grid
b = cos(4 * t);
% Method 1: Normal Equations
x_ne = (A' * A) \ (A' * b);
% Method 2: QR factorization
[Q, R] = qr(A, 0);   % Economy size QR factorization
x_qr = R \ (Q' * b);
% Method 3: Backslash operator (MATLAB's direct least squares)
x_backslash = A \ b;
% Method 4: SVD
[U, S, V] = svd(A, 0);  % Economy size SVD
x_svd = V * (S \ (U' * b));
% Print results with 16-digit precision
fprintf('Least Squares Solution (Normal Equations):\n');
fprintf('%.16f\n', x_ne);
fprintf('\nLeast Squares Solution (QR Factorization):\n');
fprintf('%.16f\n', x_qr);
fprintf('\nLeast Squares Solution (Backslash):\n');
fprintf('%.16f\n', x_backslash);
fprintf('\nLeast Squares Solution (SVD):\n');
fprintf('%.16f\n', x_svd);
```

```
Least Squares Solution (Normal Equations):
1.0000000011219337
-0.0000007580689982
-7.9999639497938979
-0.0006257006405706
10.6721062674470861
-0.0271272354516007
-5.6063608389560518
-0.1546121175613065
1.7921902801316751
-0.0697124495911037
-0.3414744074935626
0.0819372856585112

Least Squares Solution (QR Factorization):
1.0000000009966066
-0.0000004227430514
-7.9999812356848450
-0.0003187632419408
10.6694307959524490
-0.0138202881434843
-5.6470756271279656
-0.0753160244113009
1.6936069632861610
0.0060321090871414
-0.3742417036488793
0.0880405761173453

Least Squares Solution (Backslash):
1.0000000009966066
-0.0000004227434611
-7.9999812356724274
-0.0003187633979249
10.6694307970110316
-0.0138202924897002
-5.6470756157103219
-0.0753160440478328
1.6936069853005031
0.0060320935859139
-0.3742416974239821
0.0880405750298363

Least Squares Solution (SVD):
1.0000000009966055
-0.0000004227430369
-7.9999812356850351
-0.0003187632389944
10.6694307959244146
-0.0138202879897087
```

```
-5.6470756276353793
-0.0753160233669467
1.6936069619342609
0.0060321101573674
-0.3742417041223360
0.0880405762070285
```

Sid Meka AMATH 584

Commenting on Findings for Exercise 11.3

Normal Equations exhibit significant discrepancies in several digits compared to the other methods: QR Factorization, Backslash, and SVD. Specifically, the solutions obtained through the Normal Equations method differ notably in several places while the solutions from QR, Backslash, and SVD are much closer to one another though the Backslash method does seem a little off compared to QR Factorization and SVD. This suggests that the Normal Equations are more sensitive to rounding errors. Therefore, the Normal Equations method does exhibit numerical instability, particularly when compared to the QR Factorization, Backslash, and SVD methods.

```
function compare_qr_methods(epsilons)
   % Set default epsilon values if not provided
   if nargin < 1
       epsilons = [1, 1e-2, 1e-4, 1e-6, 1e-8];
   end

   % Initialize arrays to store the errors
   error_house = zeros(size(epsilons));
   error_classical = zeros(size(epsilons));
   error_modified = zeros(size(epsilons));

   % Loop through each epsilon value
   for i = 1:length(epsilons)
       eps = epsilons(i);
       % Define the matrix A with the current epsilon
       A = [1 1 1; eps 0 0; 0 eps 0; 0 0 eps];
       % Householder QR
       [Q_house, ~] = householderQR(A);
       % Classical Gram-Schmidt
       [Q_classical, ~] = classical_gram_schmidt(A);
       % Modified Gram-Schmidt
       [Q_modified, ~] = modified_gram_schmidt(A);

       % Compute ||Q^*Q - I||_2 for each method (using correct identity size)
       error_house(i) = norm(Q_house' * Q_house - eye(size(Q_house, 2)), 2);
       error_classical(i) = norm(Q_classical' * Q_classical -
eye(size(Q_classical, 2)), 2);
       error_modified(i) = norm(Q_modified' * Q_modified - eye(size(Q_modified,
2)), 2);

       % Print the results
       fprintf('For epsilon = %e\n', eps);
       fprintf('Householder QR: ||Q^*Q - I||_2 = %.16f\n', error_house(i));
       fprintf('Classical Gram-Schmidt: ||Q^*Q - I||_2 = %.16f\n',
error_classical(i));
       fprintf('Modified Gram-Schmidt: ||Q^*Q - I||_2 = %.16f\n',
error_modified(i));
       fprintf('\n');
   end

   % Plotting the errors on a log-log scale
   figure;
   loglog(epsilons, error_house, '-o', 'DisplayName', 'Householder QR',
'LineWidth', 2);
   hold on;
   loglog(epsilons, error_classical, '-x', 'DisplayName', 'Classical
Gram-Schmidt', 'LineWidth', 2);
   loglog(epsilons, error_modified, '-s', 'DisplayName', 'Modified
Gram-Schmidt', 'LineWidth', 2);
```

```
   hold off;

   % Label the axes and the plot
   xlabel('Epsilon (\epsilon)');
   ylabel('Error ||Q^*Q - I||_2');
   title('Comparison of QR Methods');
   legend('Location', 'best');
   grid on;
end
% Function for Householder QR (Algorithm 10.1)
function [Q, R] = householderQR(A)
   [m, n] = size(A);
   Q = eye(m);  % Start Q as an identity matrix
   R = A;       % Start R as A

   for k = 1:n
       x = R(k:m, k);
       v_k = x + sign(x(1)) * norm(x) * eye(length(x), 1);
       v_k = v_k / norm(v_k);  % Normalize

       % Apply the Householder reflection to R
       R(k:m, k:n) = R(k:m, k:n) - 2 * v_k * (v_k' * R(k:m, k:n));

       % Update Q
       Q(k:m, :) = Q(k:m, :) - 2 * v_k * (v_k' * Q(k:m, :));
   end

   % Since Q is formed by reflections, transpose it
   Q = Q';
end
% Classical Gram-Schmidt
function [Q, R] = classical_gram_schmidt(A)
   [m, n] = size(A);
   Q = zeros(m, n);
   R = zeros(n, 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
% Modified Gram-Schmidt
function [Q, R] = modified_gram_schmidt(A)
   [m, n] = size(A);
   Q = zeros(m, n);
   R = zeros(n, n);
   v = zeros(m, n);

   for i = 1:n
       v(:, i) = A(:, i);
   end

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

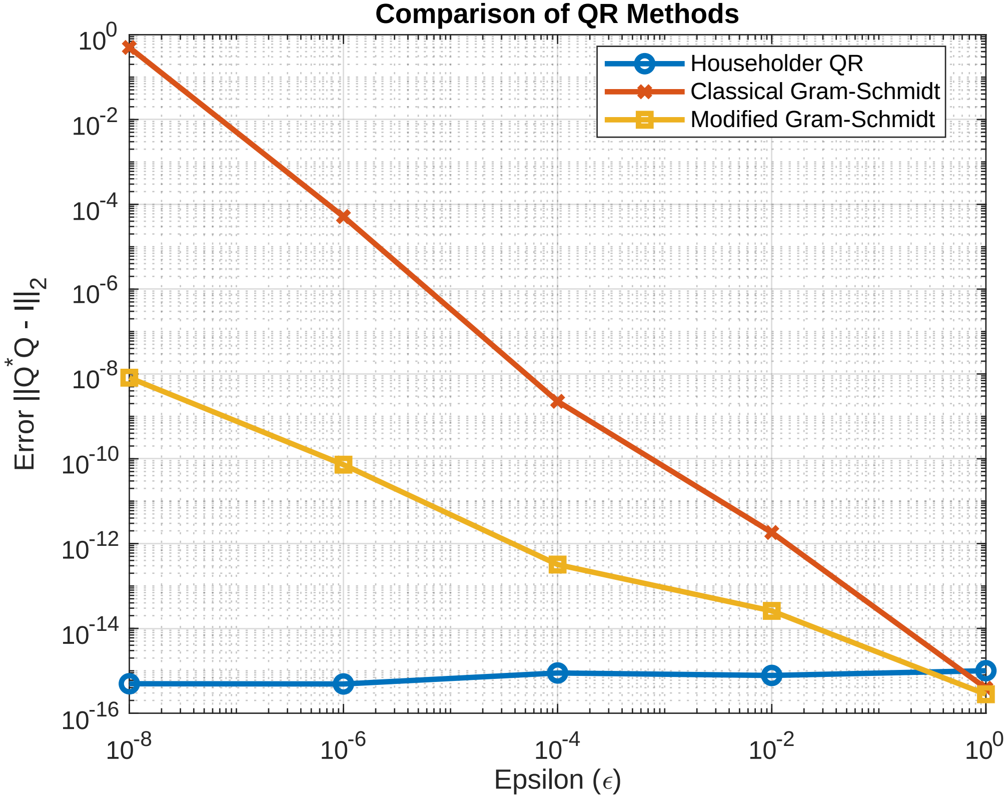
```
>> AMATH584Homework4A1
For epsilon = 1.000000e+00
Householder QR: ||Q^*Q - I||_2 = 0.0000000000000010
Classical Gram-Schmidt: ||Q^*Q - I||_2 = 0.0000000000000004
Modified Gram-Schmidt: ||Q^*Q - I||_2 = 0.0000000000000003

For epsilon = 1.000000e-02
Householder QR: ||Q^*Q - I||_2 = 0.0000000000000008
Classical Gram-Schmidt: ||Q^*Q - I||_2 = 0.0000000000018275
Modified Gram-Schmidt: ||Q^*Q - I||_2 = 0.0000000000000259

For epsilon = 1.000000e-04
Householder QR: ||Q^*Q - I||_2 = 0.0000000000000009
Classical Gram-Schmidt: ||Q^*Q - I||_2 = 0.0000000022646732
Modified Gram-Schmidt: ||Q^*Q - I||_2 = 0.0000000000003203

For epsilon = 1.000000e-06
Householder QR: ||Q^*Q - I||_2 = 0.0000000000000005
Classical Gram-Schmidt: ||Q^*Q - I||_2 = 0.0000513267756554
Modified Gram-Schmidt: ||Q^*Q - I||_2 = 0.0000000000725871

For epsilon = 1.000000e-08
Householder QR: ||Q^*Q - I||_2 = 0.0000000000000005
Classical Gram-Schmidt: ||Q^*Q - I||_2 = 0.5000000000000006
Modified Gram-Schmidt: ||Q^*Q - I||_2 = 0.0000000081649659
```


Comparison of QR Methods
Householder QR
Classical Gram-Schmidt
Modified Gram-Schmidt
Error $||Q^*Q - I||_2$
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
10^{-8}
10^{-6}
10^{-4}
10^{-2}
10^{0}
Epsilon (ϵ)

AMATH 584: Numerical Linear Algebra: Homework 4 A1 Explanation
Sid Meka

In the results, we see and analyze the numerical stability of the three methods with respect to a decreasing ϵ value:

- Classical Gram-Schmidt: This method performs well when $\epsilon = 1$, but the accuracy actually gets worse and worse rapidly for smaller values of ϵ. We see that the error becomes worse due to this particular method's sensitivity to ill conditioning.
- Modified Gram-Schmidt: We see that the modified version of the Gram-Schmidt is a little better than its classical counterpart when given significantly lower values of ϵ. However, there is still some error that shows up as ϵ decreases.
- Householder QR: This method actually delivers pretty high accuracy. Even referencing the plot, we see pretty low error and high accuracy for significantly small ϵ. Thus, we can say that on the whole, Householder QR is the most numerically stable method for QR Decomposition.

Thus, while the Classical Gram Schmidt is fast, it profoundly lacks stability especially for ill conditioned matrices. The modified counterpart is better for numerical stability, but shows some error in numerical computing for significantly small ϵ. The Householder QR method is the most accurate choice on the whole especially concerning significantly small ϵ.