AMATH 584 HW 4 Pictures
Nate Whybra

Problem 11.3:

Editor - C:\Users\Nwhybra\Desktop\UW AMATH Masters\AMATH 584\HW\HW4\problem11_3.m

problem11_3.m | problemA1.m | house_qr.m

```
% Make our dimensions.
m = 50;
n = 12;

% Make our Vandermonde matrix.
t = linspace(0, 1, 50)';
A = fliplr(vander(t));
A = A(:, 1:n);

% Make fake data for us to fit.
b = cos(4 * t);

% Compute least squares solutions to Ax=b in different ways.
% Part (a):
x_a = (A' *A) \ (A' * b);

% Part (d) (Note: The second input in qr is to tell MATLAB to give me the reduced QR):
[Q, R] = qr(A, 0);
x_d = inv(R) * (Q' * b);

% Part (e):
x_e = A \ b;

% Part (f)  (Note: The second input in svd is to tell MATLAB to give me the reduced SVD):
[U, S, V] = svd(A, 0);
w = inv(S) * U' * b;
x_f = V * w;

% Part (g):
solutions = [x_a, x_d, x_e, x_f];
```

Our solutions are summarized in the following table:

	1	2	3	4
1	1.0000	1.0000	1.0000	1.0000
2	-1.7359e-06	-4.2274e-07	-4.2274e-07	-4.2274e-07
3	-7.9999	-8.0000	-8.0000	-8.0000
4	-0.0012	-3.1876e-04	-3.1876e-04	-3.1876e-04
5	10.6762	10.6694	10.6694	10.6694
6	-0.0457	-0.0138	-0.0138	-0.0138
7	-5.5539	-5.6471	-5.6471	-5.6471
8	-0.2507	-0.0753	-0.0753	-0.0753
9	1.9056	1.6936	1.6936	1.6936
10	-0.1531	0.0060	0.0060	0.0060
11	-0.3067	-0.3742	-0.3742	-0.3742
12	0.0757	0.0880	0.0880	0.0880

Column 1: The least square values from the normal equations.

Column 2: The least square values from the QR method.

Column 3: The least square values from just using backslash.

Column 4: The least square values from the SVD method.

It is clear from the table that the normal equations method seems to suffer from instability. The other 3 algorithms produce identical values different from the values produced by the normal equations (all except for the first row).

Problem A1:

Editor - C:\Users\Nwhybra\Desktop\UW AMATH Masters\AMATH 584\HW\HW4\problemA1.m

problem11_3.m | problemA1.m | house_qr.m

```
% Store our epsilon values of interest in a list.
eps = [1, 10^(-2), 10^(-4), 10^(-6), 10^(-8)];

% Make an empty matrix to store our results.
results = zeros(5, 3);

% Repeat this procedure for each epsilon.
for i=1:5
    % Create our matrix of interest A.
    A = [1 1 1; eps(i) 0 0; 0 eps(i) 0; 0 0 eps(i)];

    % Compute each version of the QR factorization.
    [Q1, R1] = cqr(A);
    [Q2, R2] = mqr(A);
    [Q3, R3] = house_qr(A);

    % Get the identity matrix.
    I = eye(3);

    % Store the norm errors as a row in our results matrix.
    results(i, :) = [norm(Q1' * Q1 - I, 2), norm(Q2' * Q2 - I, 2), norm(Q3' * Q3 - I, 2)];
end

% Make plots to summarize our findings.
figure;
loglog(eps, results, '-o');
xlabel('Epsilon');
ylabel('Error')
title('Error vs. Espilon')
legend('Classical QR', 'Modified QR', 'Householder QR')
```

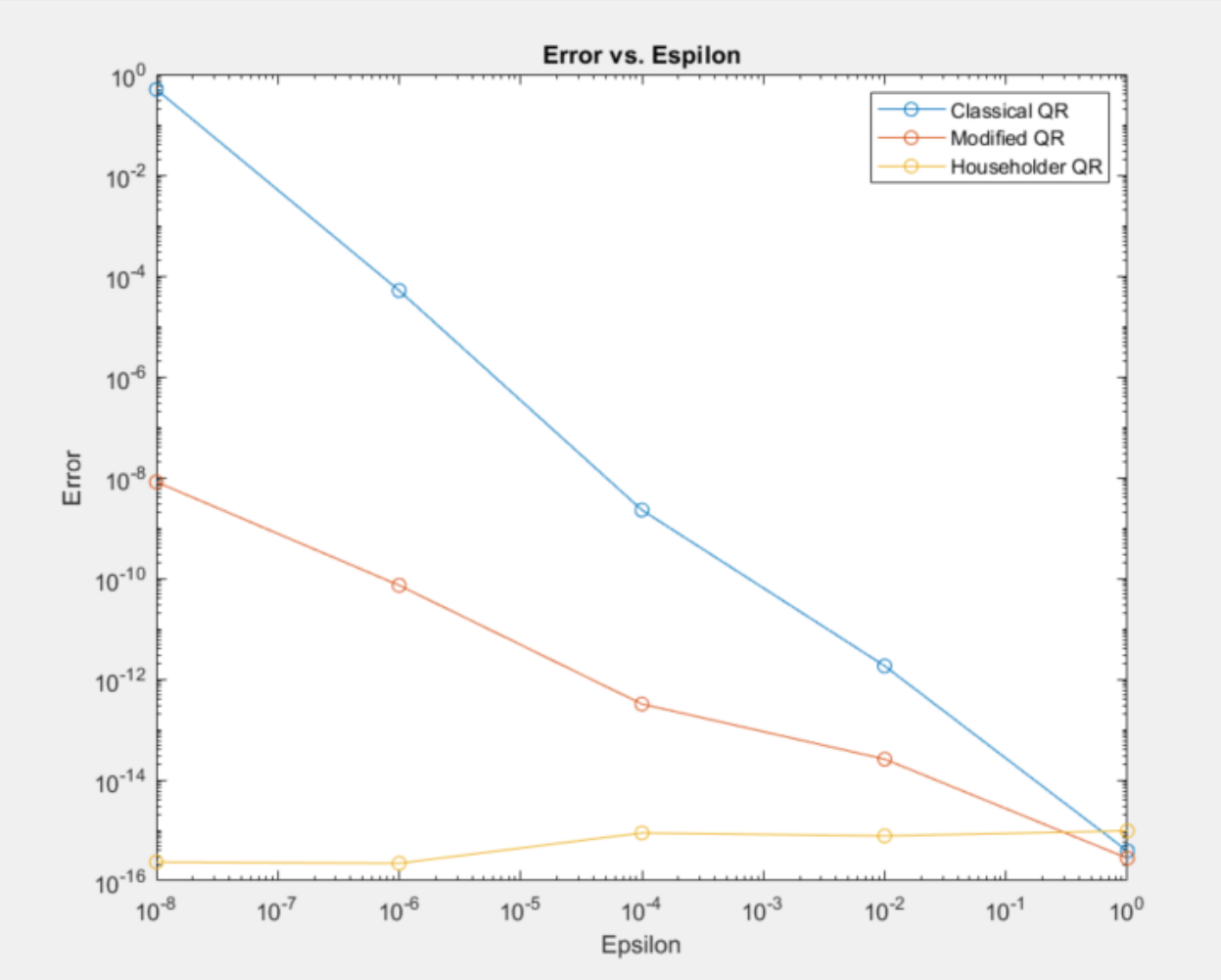
(Continues on next page)

Editor - C:\Users\Nwhybra\Desktop\UW AMATH Masters\AMATH 584\HW\HW4\house_qr.m

problem11_3.m | problemA1.m | house_qr.m

```
% Compute the reduced QR factorization of a matrix A using Householder Triangularization.
function [Q, R] = house_qr(A)
    % Get matrix dimensions.
    [m, n] = size(A);

    % Define a function for quickly getting standard basis vectors.
    % e(k, m) is a m x 1 column vector with a 1 in the kth row.
    e = @(k, m) [zeros(k-1,1); 1; zeros(m-k, 1)];

    % Make cell array for storing v_{k}'s.
    v_storage = cell(n, 1);

    % Algorithm (10.1).
    % Compute R and store it in A.
    for k = 1:n
        x = A(k:m, k);
        v = sign(x(1)) * norm(x, 2) * e(1, m-k+1) + x;
        v = v / norm(v, 2);
        A(k:m, k:n) = A(k:m, k:n) - 2 * v * (v' * A(k:m, k:n));
        v_storage{k} = v;
    end

    % Return R as a copy of A.
    R = A;
    % Make R a n x n matrix so that this is the reduced QR.
    R = R(1:n, :);

    % Make an empty matrix for Q.
    Q = zeros(m, n);

    % For each standard basis vector e_{i}, use Algorithm 10.3 to compute
    % Qe_{i} to get q_{i}.
    for i = 1:n
        q = e(i, m);
        for k = n:-1:1
            q(k:m) = q(k:m) - 2 * v_storage{k} * (v_storage{k}' * q(k:m));
        end
        Q(:, i) = q;
    end
end
```

(Continues on next page)



As we can see from the above plot, the “unitary matrix error” is larger for the Classical QR and Modified QR algorithms as compared to the Householder QR algorithm. The Householder algorithm error remains small and relatively constant as epsilon changes value, but the errors for the other algorithms are larger for small values of epsilon (the errors start high and decrease as epsilon grows larger). The error for each of the algorithms seems to be roughly similar when epsilon approaches 1.