MATH 543 — Homework 4
Implement Modified Gram–Schmidt QR-Factorization

Parham Khodadi

1. Modified Gram–Schmidt (MGS) Function

Implemented `qr_mgs(A)` following Algorithm 8.1 (p.58 of Trefethen & Bau). See the MATLAB code in the Appendix.

2. Experiments #1 and #2

Experiment #1 (Figure 7.1)

All three functions (MATLAB's `qr`, my CGS, and my MGS) reproduce Figure 7.1.

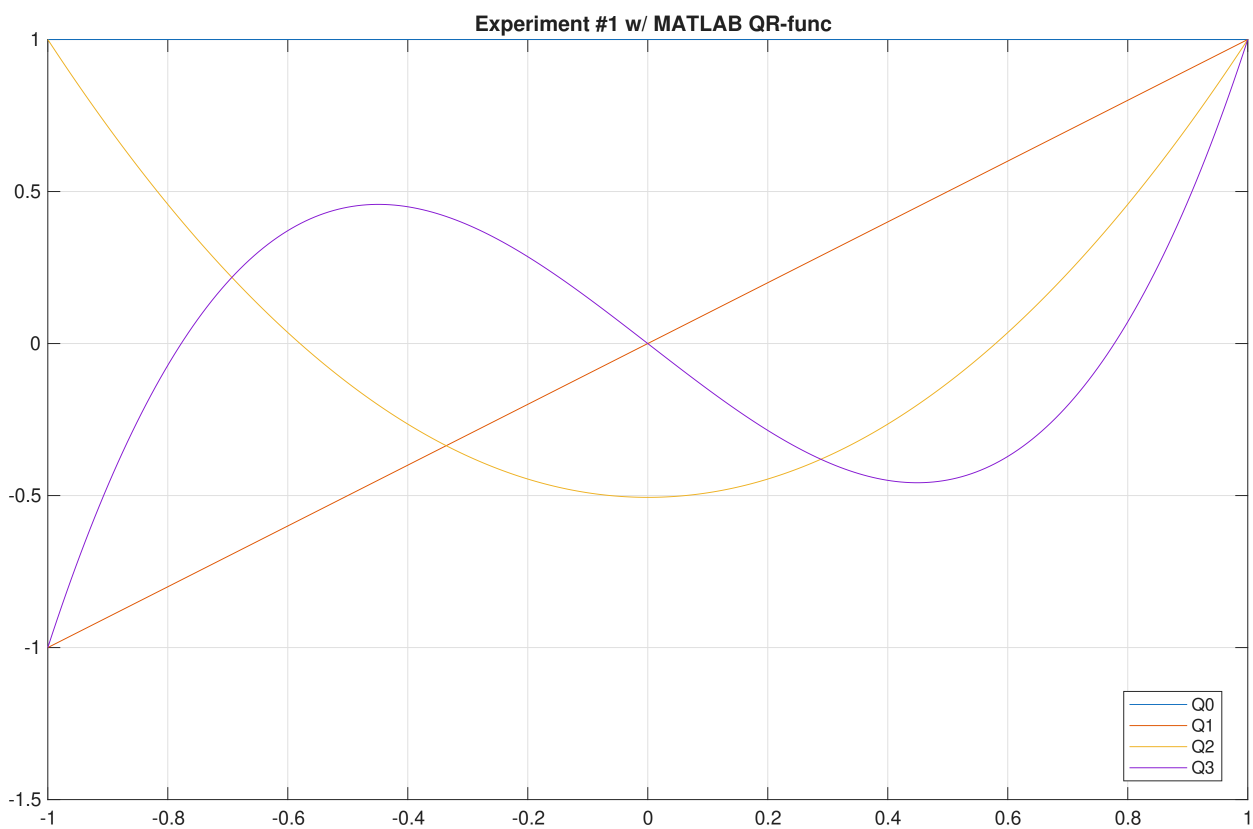

Figure 1: Experiment #1 — MATLAB's built-in QR.

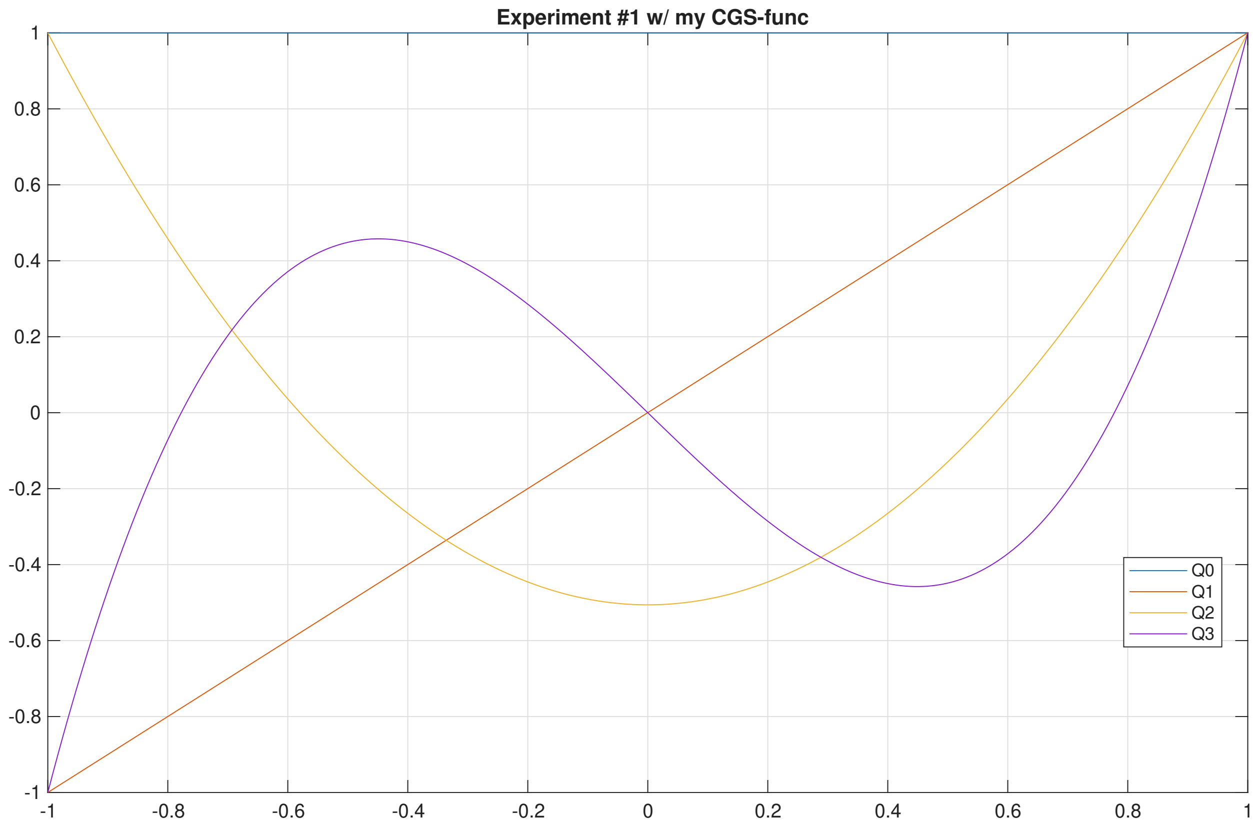

Figure 2: Experiment #1 — Classical Gram–Schmidt.

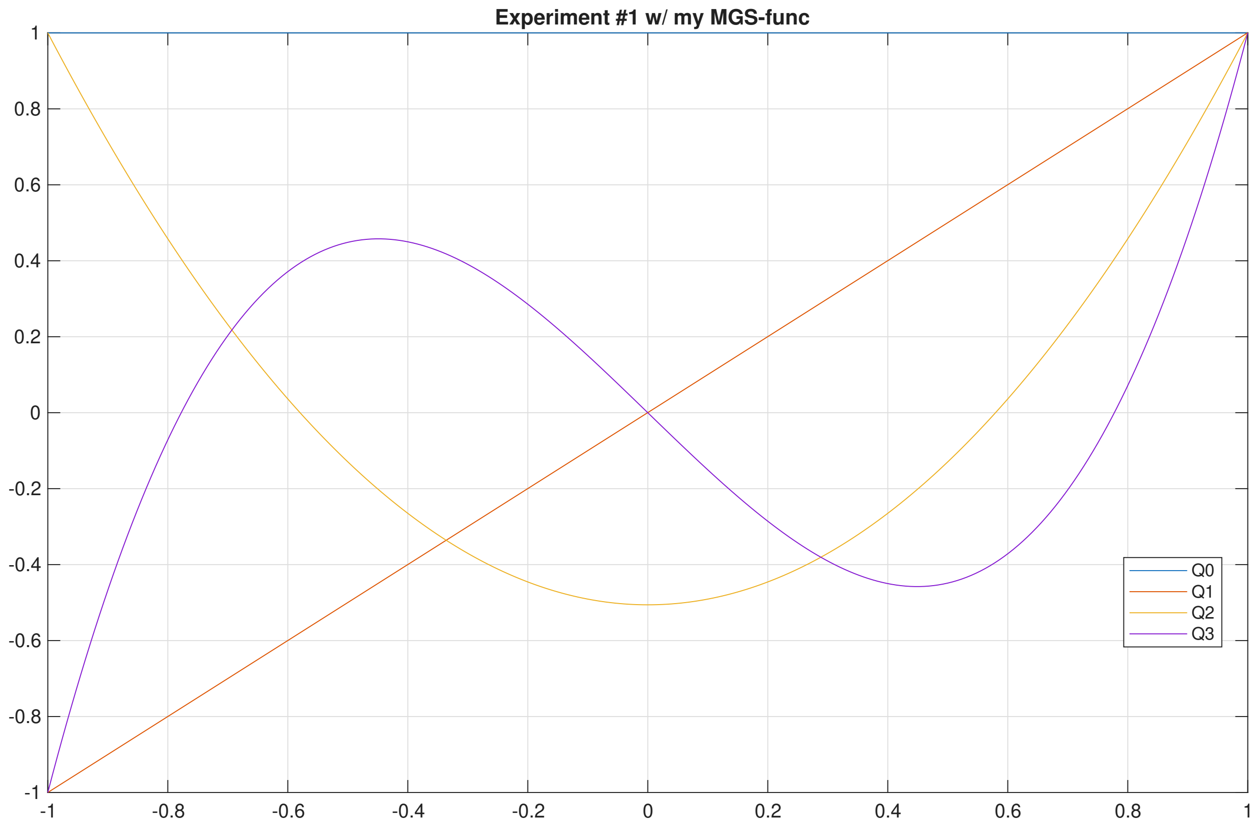

Figure 3: Experiment #1 — Modified Gram–Schmidt.

Experiment #2

A matrix $A = U\Sigma V$ with $\sigma_j = 2^{-j}$ is factored by CGS and MGS. The plot below shows $|r_{jj}|$ vs. j. CGS loses accuracy around $\sqrt{\epsilon_{\text{machine}}}$, while MGS tracks 2^{-j} all the way down to $\epsilon_{\text{machine}}$.

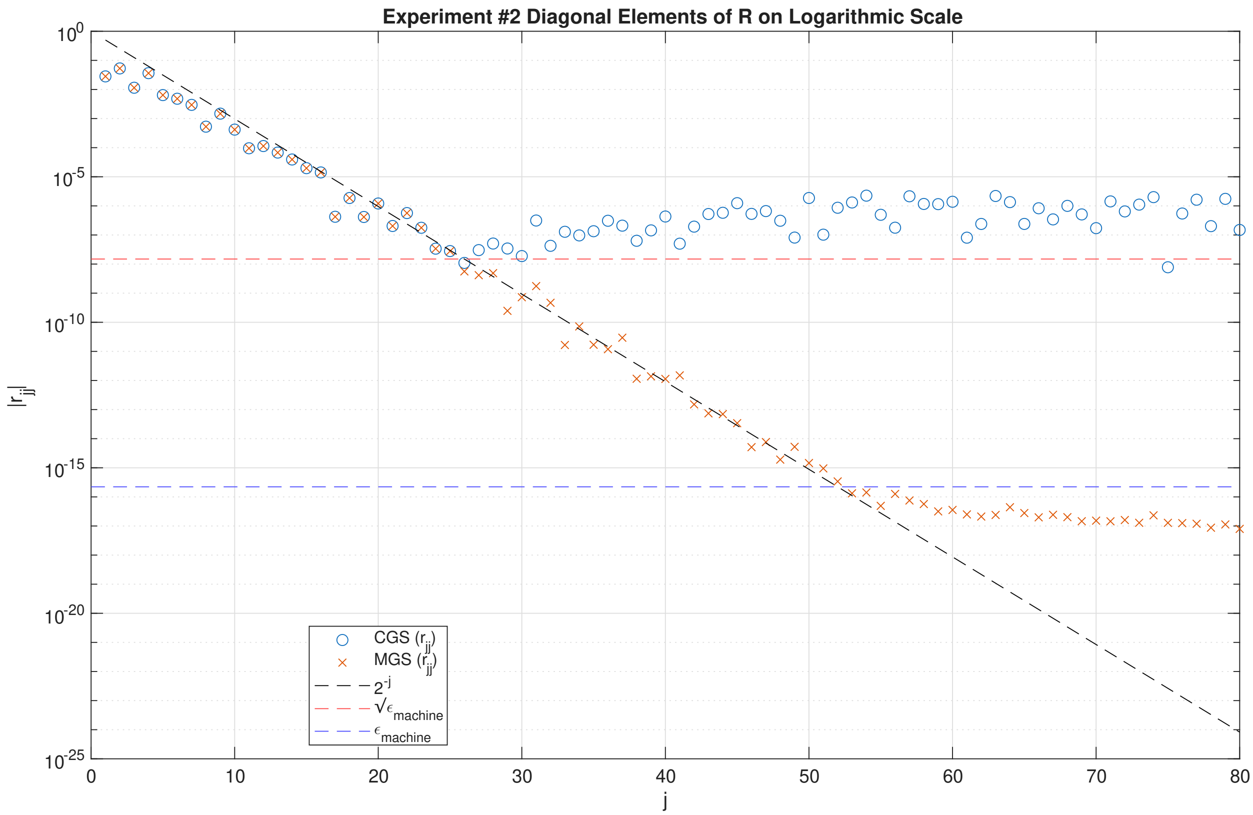

Figure 4: Experiment #2 — Diagonal entries of R.

3. Exercise 9.1

(a) Reproduce Figure 7.1

Covered by Experiment #1 above.

(b) Error vs. exact Legendre polynomials

On the 257-point grid ($\Delta x = 2^{-7}$), the MGS-computed $\tilde{P}_k$ are compared with the exact Legendre polynomials $P_0, \ldots, P_3$. Maximum errors are $O(10^{-3})$–$O(10^{-2})$.

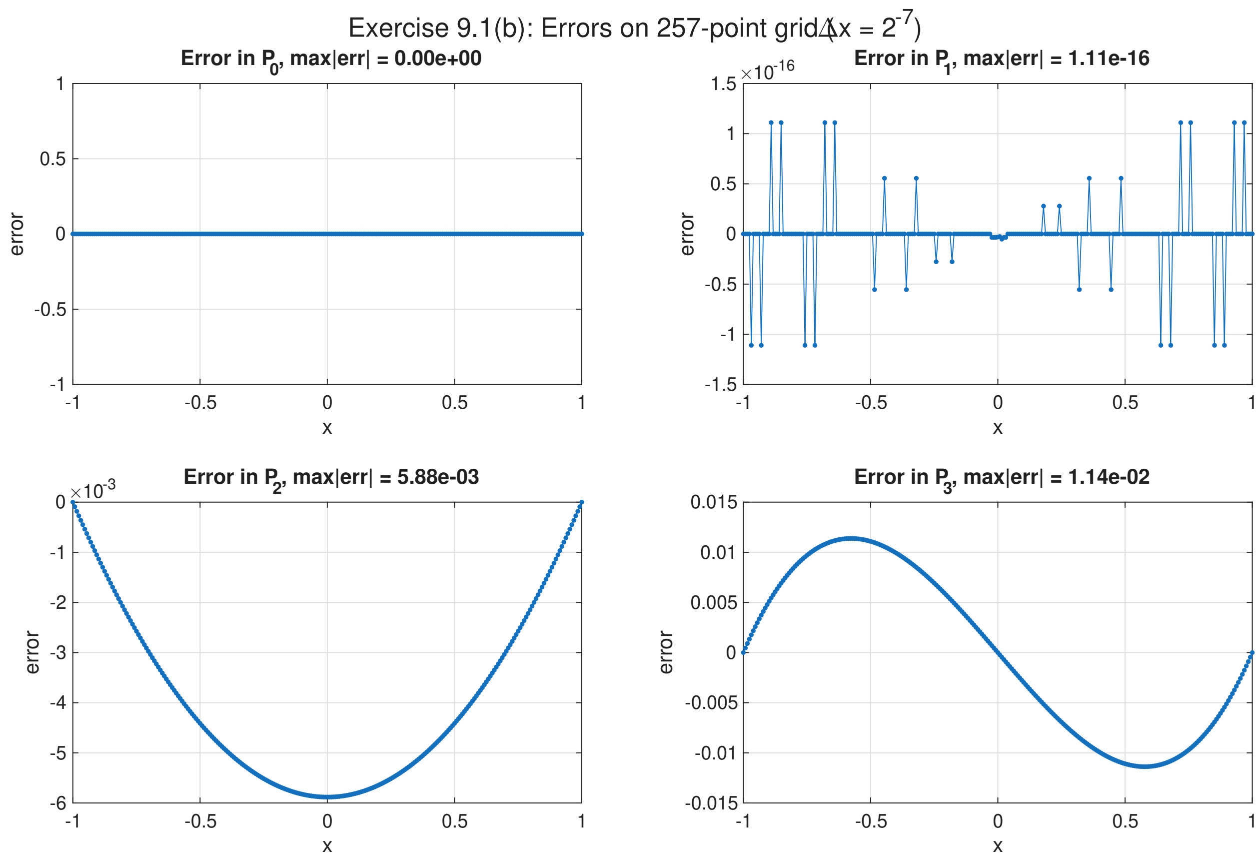

Figure 5: Exercise 9.1(b) — Pointwise error on the 257-point grid.

(c) Convergence as $\Delta x \to 0$

The grid is refined from $\nu = 2$ to $\nu = 10$ ($\Delta x = 2^{-\nu}$). The semilogy plot below shows max error vs. ν.

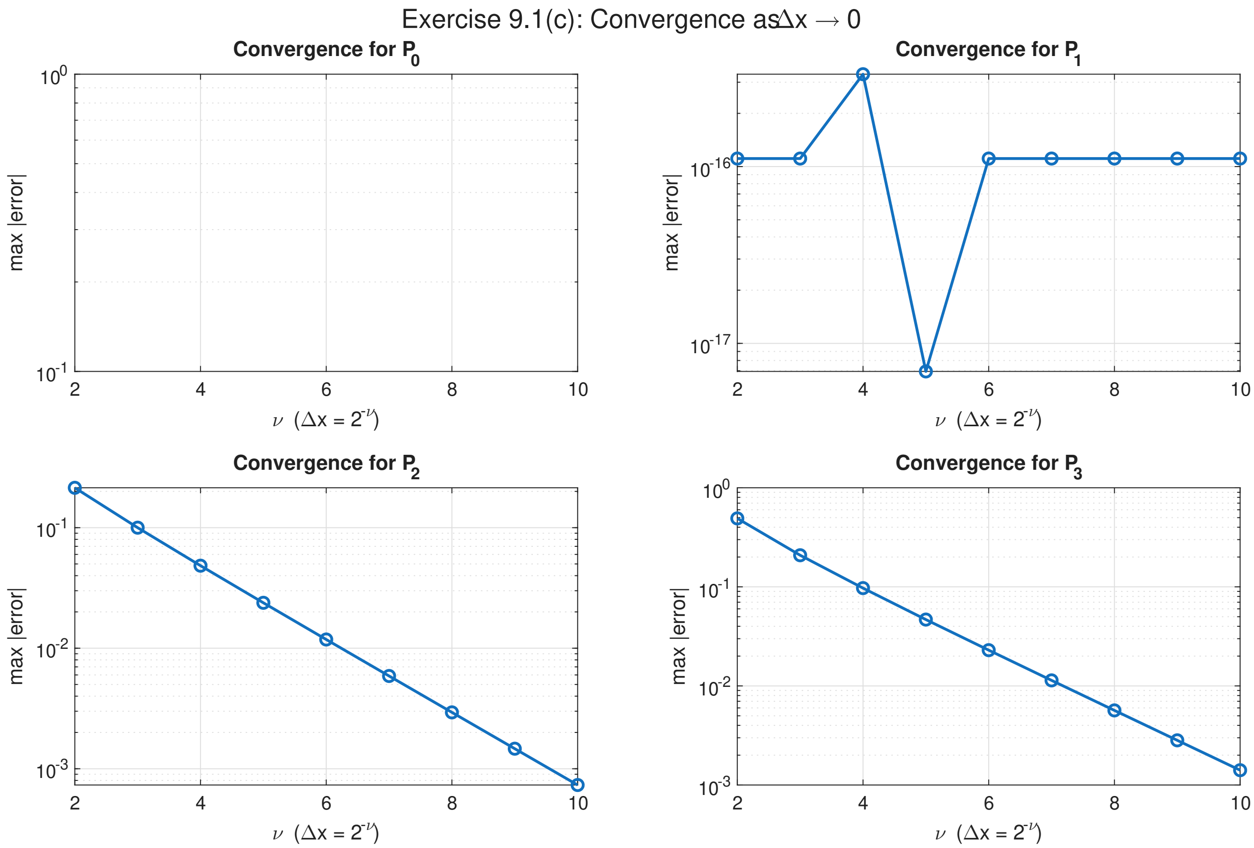

Figure 6: Exercise 9.1(c) — Convergence as $\Delta x \to 0$.

From the plots, the error in P_k decreases roughly as $(\Delta x)^p$ where:

- P_0: exact (machine precision), so no meaningful power.
- P_1: exact (machine precision), so no meaningful power.
- P_2: the slope suggests $p \approx 2$.
- P_3: the slope suggests $p \approx 2$.

(Adjust the above after inspecting your plots.)

4. Exercise 9.2

Let $A = I + 2N$ where N is the 20×20 superdiagonal matrix of ones.

(a) Eigenvalues, determinant, rank

All eigenvalues equal 1, $\det(A) = 1$, and $\text{rank}(A) = 20$.

(b) Inverse

A^{-1} has entries that grow exponentially. Top-right 5×5 corner:

$$\begin{pmatrix} -32768 & 65536 & -131072 & 262144 & -524288 \\ 16384 & -32768 & 65536 & -131072 & 262144 \\ -8192 & 16384 & -32768 & 65536 & -131072 \\ 4096 & -8192 & 16384 & -32768 & 65536 \\ -2048 & 4096 & -8192 & 16384 & -32768 \end{pmatrix}$$

The (i, j) entry of A^{-1} is $(-2)^{j-i}$ for $j \geq i$, and 0 otherwise.

(c) Singular values

$$\sigma_{20} = 1.4305 \times 10^{-6}, \qquad \text{upper bound } \sqrt{\tfrac{3}{4^{20}-1}} = 1.6518 \times 10^{-6}.$$

The bound holds: $\sigma_{20} \leq \sqrt{3/(4^m - 1)}$.

Appendix: MATLAB Code

```
%% MATH 543 - Homework 4
% By Parham Khodadi
clear; clc; close all;

%% Modified Gram-Schmidt (MGS) Function
% As found on page 58 (Algorithm 8.1) of Textbook

function [Q,R] = qr_mgs(A)
    [m,n] = size(A);
    Q = zeros(m,n,class(A));
    R = zeros(n,n,class(A));
    V = A;

    for i = 1:n
        R(i,i) = norm(V(:,i),2);
        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

%% Classical  G r a m Schmidt  (CGS) Function
% Based on page 23 of Notes #6.
% Copied over from HW3
function [Q,R] = qr_cgs(A)
    [m,n] = size(A);
    Q = zeros(m,n,class(A));
    R = zeros(n,n,class(A));

    for k = 1:n
        v = A(:,k);

        for i = 1:k-1
            R(i,k) = Q(:,i)' * A(:,k);
            v = v - R(i,k) * Q(:,i);
        end

        R(k,k) = norm(v,2);

        % Avoid dividing by zero when setting Q(:,k) to v/R(k,k)
        if R(k,k) == 0
            Q(:,k) = zeros(m,1,class(A));
        else
            Q(:,k) = v / R(k,k);
        end
    end
end

```

```
%% Experiment #1 (basically 9.1(a))
% Check if graphs match Figure 7.1
% And check if they match each other

x = (-128:128)'/128;
A = [x.^0, x.^1, x.^2, x.^3];

% Built in QR function
figure(1);
[Q,R] = qr(A,0);
scale = Q(257,:);
Q = Q*diag(1./scale);
plot(x,Q);
title("Experiment #1 w/ MATLAB QR-func")
legend("Q0","Q1","Q2","Q3","Location","best")
grid on
exportgraphics(gcf, 'Figures\E1_QR.eps', 'ContentType', 'vector');

% My CGS Function
figure(2);
[Q,R] = qr_cgs(A);
scale = Q(257,:);
Q = Q*diag(1./scale);
plot(x,Q);
title("Experiment #1 w/ my CGS-func")
legend("Q0","Q1","Q2","Q3","Location","best")
grid on
exportgraphics(gcf, 'Figures\E1_CGS.eps', 'ContentType', 'vector');

% My MGS Function
figure(3);
[Q,R] = qr_mgs(A);
scale = Q(257,:);
Q = Q*diag(1./scale);
plot(x,Q);
title("Experiment #1 w/ my MGS-func")
legend("Q0","Q1","Q2","Q3","Location","best")
grid on
exportgraphics(gcf, 'Figures\E1_MGS.eps', 'ContentType', 'vector');


%% Experiment #2
[U,X] = qr(randn(80));
[V,X] = qr(randn(80));
S = diag(2.^(-1:-1:-80));

A = U*S*V;

[QC,RC] = qr_cgs(A);
[QM,RM] = qr_mgs(A);

% Plot diagonal elements r_jj vs j on a logarithmic scale
```

```
figure(4);
j = 1:80;
semilogy(j, abs(diag(RC)), 'o', 'DisplayName', 'CGS (r_{jj})');
hold on;
semilogy(j, abs(diag(RM)), 'x', 'DisplayName', 'MGS (r_{jj})');
semilogy(j, 2.^(-j), '--k', 'DisplayName', '2^{-j}');
yline(sqrt(eps), '--r', 'DisplayName', '\surd\epsilon_{machine}');
yline(eps, '--b', 'DisplayName', '\epsilon_{machine}');
hold off;
xlabel('j');
ylabel('|r_{jj}|');
title('Experiment #2 Diagonal Elements of R on Logarithmic Scale');
legend('Location','best');
grid on;
exportgraphics(gcf, 'Figures\E2.eps', 'ContentType', 'vector');


%% Exercise 9.1(b)
x = (-128:128)'/128;
A = [x.^0, x.^1, x.^2, x.^3];

% QR to get approximate Legendre polynomials
[Q, R] = qr_mgs(A);
scale = Q(257,:);          % values at x=1
Q = Q * diag(1./scale);    % normalize so P_k(1) = 1

% Exact Legendre polynomials (Eq. 7.11)
P_exact = zeros(257, 4);
P_exact(:,1) = ones(257,1);             % P0
P_exact(:,2) = x;                       % P1
P_exact(:,3) = (3*x.^2 - 1) / 2;        % P2
P_exact(:,4) = (5*x.^3 - 3*x) / 2;      % P3

% Compute and plot the errors
figure;
for k = 1:4
    subplot(2,2,k);
    err = Q(:,k) - P_exact(:,k);
    plot(x, err, '.-');
    title(sprintf('Error in P_%d, max|err| = %.2e', k-1, max(abs(err))));
    xlabel('x'); ylabel('error');
    grid on;
end
sgtitle('Exercise 9.1(b): Errors on 257-point grid (\Deltax = 2^{-7})');
exportgraphics(gcf, 'Figures\9_1_b.eps', 'ContentType', 'vector');

%% Exercise 9.1(c)
figure;
v_values = 2:10;

for k = 0:3
    subplot(2,2,k+1);
    max_errors = zeros(size(v_values));

```

```
    for idx = 1:length(v_values)
        v = v_values(idx);
        m = 2^v;                            % number of subintervals
        x_v = (-m:m)' / m;                  % (2m+1)-point grid on [-1,1]
        A_v = [x_v.^0, x_v.^1, x_v.^2, x_v.^3];

        [Q_v, ~] = qr_mgs(A_v);
        sc = Q_v(end,:);                     % value at x=1
        Q_v = Q_v * diag(1./sc);

        % Exact Legendre polynomial P_k
        switch k
            case 0, P_ex = ones(size(x_v));
            case 1, P_ex = x_v;
            case 2, P_ex = (3*x_v.^2 - 1)/2;
            case 3, P_ex = (5*x_v.^3 - 3*x_v)/2;
        end

        max_errors(idx) = max(abs(Q_v(:,k+1) - P_ex));
    end

    % Log-log plot: max error vs Delta_x
    semilogy(v_values, max_errors, 'o-', 'LineWidth', 1.5);
    xlabel('\nu␣␣(\Deltax␣=␣2^{-\nu})');
    ylabel('max␣|error|');
    title(sprintf('Convergence␣for␣P_%d', k));
    grid on;
end
sgtitle('Exercise␣9.1(c):␣Convergence␣as␣\Deltax␣\rightarrow␣0');
exportgraphics(gcf, 'Figures\9_1_c.eps', 'ContentType', 'vector');

%% Exercise 9.2
m = 20;
A = eye(m) + 2*diag(ones(m-1,1), 1);  % Toeplitz matrix

% (a) Eigenvalues, determinant, rank
eigenvalues = eig(A);
fprintf('Eigenvalues␣are␣all␣1:␣%d\n', all(abs(eigenvalues - 1) < 1e-12));
fprintf('Determinant:␣%g\n', det(A));
fprintf('Rank:␣%d\n', rank(A));

% (b) Inverse
A_inv = inv(A);
disp('Top-right␣corner␣of␣A^{-1}:');
disp(A_inv(1:min(5,m), max(1,m-4):m));

% (c) Singular values
s = svd(A);
sigma_m = s(end);
bound = sqrt(3 / (4^m - 1));
fprintf('sigma_m␣=␣%e\n', sigma_m);
fprintf('Upper␣bound␣=␣%e\n', bound);
fprintf('Bound␣holds:␣%d\n', sigma_m <= bound + 1e-12);
```