VLSI DPS HW#2

電機四 **4109064119** 陳柏淳

Q1. LMS filter design

```
clear all
close all
clc

%% parameters setting
Iters = 200;                    % Iteration Times
M = 15;                         % Filter Size
M_end = M + Iters;              % Final index
W = zeros(M,1);                 % Filter Coefficients
W_hist = zeros(M, M_end); % History of Filter Coefficients
mu = 1e-2;                      % Step size
n = 1 : M_end;                  % Time index
s = sin(2*pi*n/16) + cos(2*pi*n/4); % Input
d = sin(2*pi*n/16);             % Desired
e = zeros(M_end);               % Error
r = zeros(M_end);               % RMS
rms_step = 16;

% Converged Condition
converge_value = 0.05/sqrt(2);
converged = false;

%% LMS Iteration
for i = M : M_end
    U = s(i:-1: i-M+1)';            % Extract input
    d_hat = W' * U;                 % Compute filter output
    e(i) = d(i) - d_hat;            % Compute error
    W = W + mu*e(i)*U;              % Update filter coefficients
    W_hist(:,i+1) = W;              % Add coefficients to history
    r(i) = sqrt(mean(e(max(M, i-rms_step+1):i).^2)); % RMS
    % Determine convergence
    if r(i) < converge_value && ~converged
        disp(['Convergence achieved at ' num2str(i-M+1) ' iterations.']);
        converged = true;
    end
end

disp(['Min RMS value ' num2str(min(r(M:Iters)))]);


```

```
%% Plot
% Plot the RMS Error versus time
figure('Name','RSM Plot');
plot(M:M_end, r(M:M_end));
xlim([M M_end]);
xlabel('n');
ylabel('RMS Error');
title('RSM versus n');

% Plot the filter coefficients versus time
figure('Name','Filter Coefficients');
plot(M:M_end, W_hist(:, M:M_end)');
xlim([M M_end]);
xlabel('n');
ylabel('Coefficients');
title('Filter Coefficients');

% Compute and plot the frequency response
fft_resp = fft(W, 64);
f = (0:63); % frequency axis
figure('Name','Frequency response');
plot(f, abs(resp));
xlim([0 63]);
xlabel(' Sample points(FFT)');
ylabel('Magnitude');
title(' 64?point FFT to the impulse response with low pass filter');


%% Determine the minimum M
Iters = 5000;              % Maximum Iteration Times
M_max = 100;               % Maximum Filter Size
mu = 1e-2;                 % Initial Step size
rms_step = 16;

% Converged Condition
converge_value = 0.05/sqrt(2);
convergence_flag = false;

for M = 1:M_max
    M_end = M + Iters;          % Final index
    W = zeros(M,1);             % Filter Coefficients
    n = 1 : M_end;              % Time index
    s = sin(2*pi*n/16) + cos(2*pi*n/4); % Input
    d = sin(2*pi*n/16);         % Desired
    e = zeros(M_end, 1);        % Error
    r = zeros(M_end, 1);        % RMS

```

```
88      % LMS Iteration
89      for i = M : M_end
90          U = s(i:-1: i-M+1)';            % Extract input
91          d_hat = W' * U;                 % Compute filter output
92          e(i) = d(i) - d_hat;             % Compute error
93          W = W + mu*e(i)*U;              % Update filter coefficients
94          r(i) = sqrt(mean(e(max(M, i-rms_step+1):i).^2)); % RMS
95          % Determine convergence
96          if r(i) < converge_value
97              disp(['Converged at M = ' num2str(M) ', Iteration = ' num2str(i-M+1)]);
98              min_M = M;
99              min_iter_count = i - M + 1;
100             convergence_flag = true;
101             break;
102         end
103     end
104     if convergence_flag
105         break;
106     end
107 end
108
109 disp(['Minimum iterations for M converged : ' num2str(min_M)]);
110 disp(['Iterations required: ' num2str(min_iter_count)]);
111
```

- Result

a. ($\mu=10^{-2}$, Iter=200)

Converged at 95 iterations

Min RMS = 0.0017354

```
Converged at 95 iterations.
Min RMS value : 0.0017354
```

b. The minimum filter length m so that the adaptation can converge in no more than 5000 iterations.

b1. m = 4 when the step size = 10^{-2}

```
Converged at M = 4, Iteration = 4511
Minimum iterations for M converged : 4
Iterations required: 4511
```

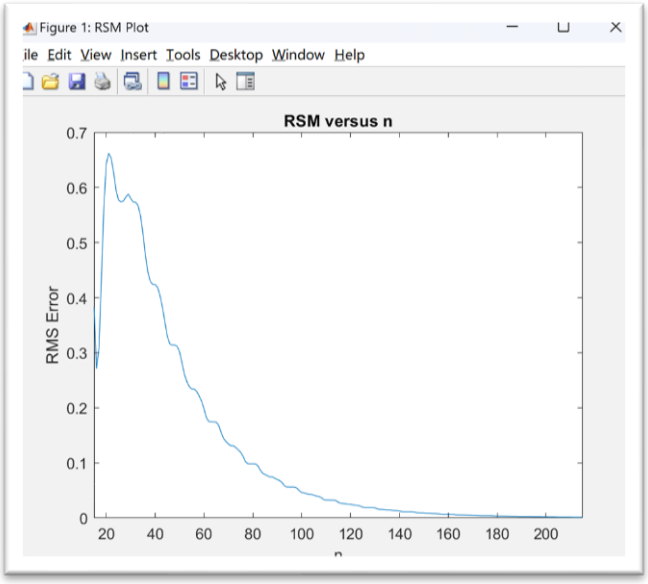
b2. m = 6 when the step size = 10^{-3}

```
Converged at M = 6, Iteration = 2733
Minimum iterations for M converged : 6
Iterations required: 2733
```

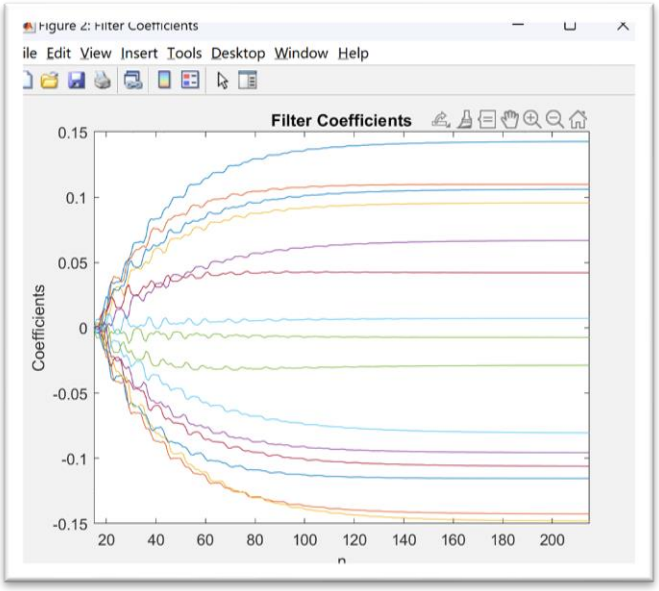
b3. m = 8 when the step size = 10^{-4}

```
Converged at M = 8, Iteration = 1
Minimum iterations for M converged : 8
Iterations required: 1
```

C. Plot

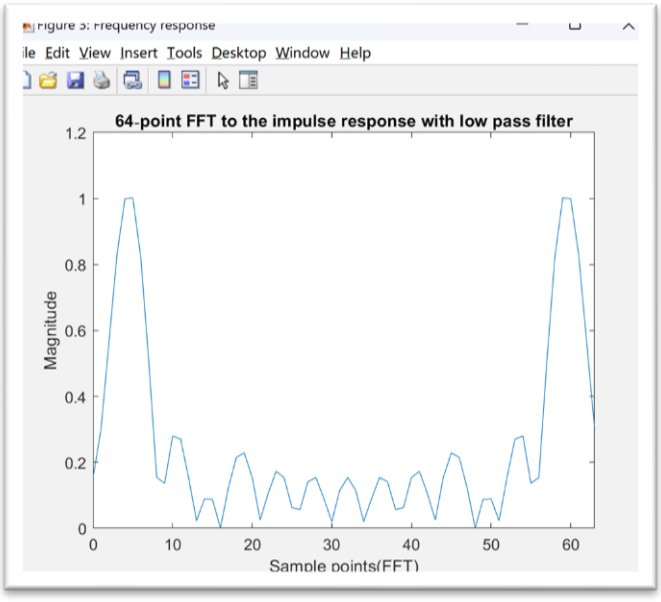


Show the plot of “r(n)” versus “n” and indicate when the filter converges.
95 training samples are required.



Show the plot of filter coefficients bi(n), for i = 0 ~ m-1, versus “n”.

The values of filter coefficients remain mostly unchanged after convergence.



Apply a 64-point FFT to the impulse response of the converged filter and verify the filter isindeed a low pass one.

Q2. Discrete Wavelet Transform

```
clear all
close all
clc

%% Define filter
h_coe = [     0.852698679009;        0.377402855613;        -0.110624404418;
               -0.023849465020;       0.037828455507];
g_coe = [     -0.788485616406;        0.418092273222;         0.040689417609;
              -0.064538882629];
q_coe = [     0.788485616406;         0.418092273222;        -0.040689417609;
              -0.064538882629];
p_coe = [     -0.852698679009;        0.377402855613;         0.110624404418;
              -0.023849465020;       -0.037828455507];

% Symmetric Extension
h = Symmetric_Extension(h_coe);
g = Symmetric_Extension(g_coe);
q = Symmetric_Extension(q_coe);
p = Symmetric_Extension(p_coe);

% Read img
ori_img = double(imread('HW2 test image.bmp'));

%% DWT Level 1
[L_1, H_1]     = DWT_ROW(ori_img, h, g);
[LL_1, LH_1] = DWT_COL(L_1, h, g);
[HL_1, HH_1] = DWT_COL(H_1, h, g);
% DWT Level 2
[L_2, H_2]     = DWT_ROW(LL_1, h, g);
[LL_2, LH_2] = DWT_COL(L_2, h, g);
[HL_2, HH_2] = DWT_COL(H_2, h, g);
% DWT Level 3
[L_3, H_3]     = DWT_ROW(LL_2, h, g);
[LL_3, LH_3] = DWT_COL(L_3, h, g);
[HL_3, HH_3] = DWT_COL(H_3, h, g);
% Combine
DWT_result = [[[LL_3 HL_3; LH_3 HH_3] HL_2;
                        LH_2 HH_2] HL_1;
                        LH_1 HH_1];
```

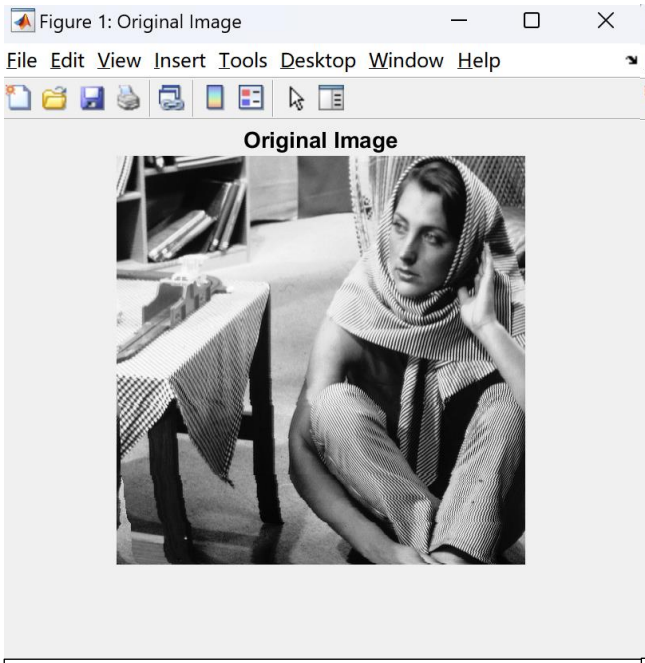
```
%% Reconstruction
% IDWT Level-3
Rec_L_3         = IDWT_COL(LL_3, LH_3, q, p);
Rec_H_3         = IDWT_COL(HL_3, HH_3, q, p);
Rec_LL_2        = IDWT_ROW(Rec_L_3, Rec_H_3, q, p);
% IDWT Level-2
Rec_L_2         = IDWT_COL(Rec_LL_2, LH_2, q, p);
Rec_H_2         = IDWT_COL(HL_2, HH_2, q, p);
Rec_LL_1        = IDWT_ROW(Rec_L_2, Rec_H_2, q, p);
% IDWT Level-1
Rec_L_1         = IDWT_COL(Rec_LL_1, LH_1, q, p);
Rec_H_1         = IDWT_COL(HL_1, HH_1, q, p);
Rec_img_a       = IDWT_ROW(Rec_L_1, Rec_H_1, q, p);

% IDWT Level-1 (setting HL1 LH1 HH1 to zero)
Rec_L_1_        = IDWT_COL(Rec_LL_1, zeros(size(LH_1)), q, p);
Rec_H_1_        = IDWT_COL(zeros(size(HL_1)), zeros(size(HH_1)), q, p);
Rec_img_b       = IDWT_ROW(Rec_L_1_, Rec_H_1_, q, p);

% Calaulate PSNR
PSNR_a = PSNR(ori_img, Rec_img_a, 8);
PSNR_b = PSNR(ori_img, Rec_img_b, 8);

disp(['PSNR a: ',num2str(PSNR_a) ,' dB']);
disp(['PSNR b: ',num2str(PSNR_b) ,' dB']);

%% Plot
% Original image
figure('Name','Original Image');
imshow(mat2gray(ori_img));
title('Original Image');

% 3-level DWT
figure('Name','3-Level DWT');
Plot_DWT = mat2gray(DWT_result);
% add lines
num_levels = 3;
for level = 1:num_levels
    region_size = 512 / (2^(level-1));
    Plot_DWT(1:region_size, round(region_size/2)) = 1; % Vertical line
    Plot_DWT(round(region_size/2), 1:region_size) = 1; % Horizontal line

```

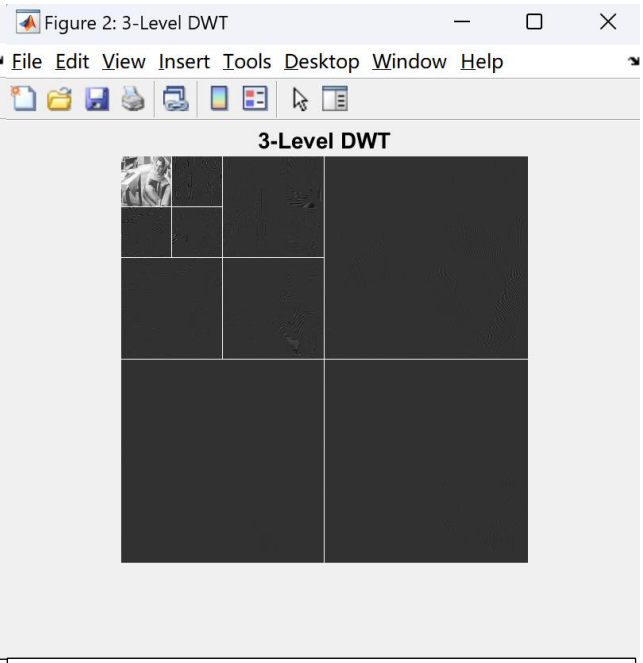
```
end
imshow(Plot_DWT);
title('3-Level DWT');

% Synthesis A image
figure('Name','Synthesis Image A');
imshow(mat2gray(Rec_img_a));
title('Synthesis Image A');

% Synthesis B image(se HL1 LH1 HH1 to zero)
figure('Name','Synthesis Image B');
imshow(mat2gray(Rec_img_b));
title('Synthesis Image B');

%% Function
% Symmetric Extension
function extended_data = Symmetric_Extension(data)
    Recersed_data = flipud(data);
    trimmed_data = data(2:end);
    extended_data = [Recersed_data; trimmed_data];
end

% Filter
function y = Filter(x, w)
    if iscolumn(x), x = x'; end
    if iscolumn(w), w = w'; end
    N = size(x, 2);
    M = size(w, 2);
    L = fix( M/ 2);
    temp = conv(w, [x(L+1:-1:2), x, x(N-1:-1:N-L)]);
    y = temp(M : M+N-1);
end

% ROW-wise DWT
function [L, H] = DWT_ROW(img, L_Filter, H_Filter)
    [row, col] = size(img);
    L = zeros(row, col);
    H = zeros(row, col);
    for i = 1: row
        L(i,:) = Filter(img(i,:), L_Filter);
        H(i,:) = Filter(img(i,:), H_Filter);

```

```
    end
    % Down Sample
    L = L(:, 1:2:end); % Keep Odd
    H = H(:, 2:2:end); % Keep Even
end

% COL-wise DWT
function [L, H] = DWT_COL(img, L_Filter, H_Filter)
    [row, col] = size(img);
    L = zeros(row, col);
    H = zeros(row, col);
    for i = 1: col
        L(:,i) = Filter(img(:,i), L_Filter)';
        H(:,i) = Filter(img(:,i), H_Filter)';
    end
    % Down Sample
    L = L(1:2:end, :); % Keep Odd
    H = H(2:2:end, :); % Keep Even
end

% ROW-wise IDWT
function img = IDWT_ROW(L, H, L_Filter, H_Filter)
    [row, col] = size([L H]);
    % up sample
    Ext_L = zeros(row, col);
    Ext_H = zeros(row, col);
    Ext_L(:, 1:2:end) = L; % keep odd
    Ext_H(:, 2:2:end) = H; % keep even
    for i = 1: row
        Ext_L(i,:) = Filter(Ext_L(i,:), L_Filter);
        Ext_H(i,:) = Filter(Ext_H(i,:), H_Filter);
    end
    img = Ext_L + Ext_H;
end

% COL-wise IDWT
function img = IDWT_COL(L, H, L_Filter, H_Filter)
    [row, col] = size([L; H]);
    % Up Sample
    Ext_L = zeros(row, col);
    Ext_H = zeros(row, col);

```

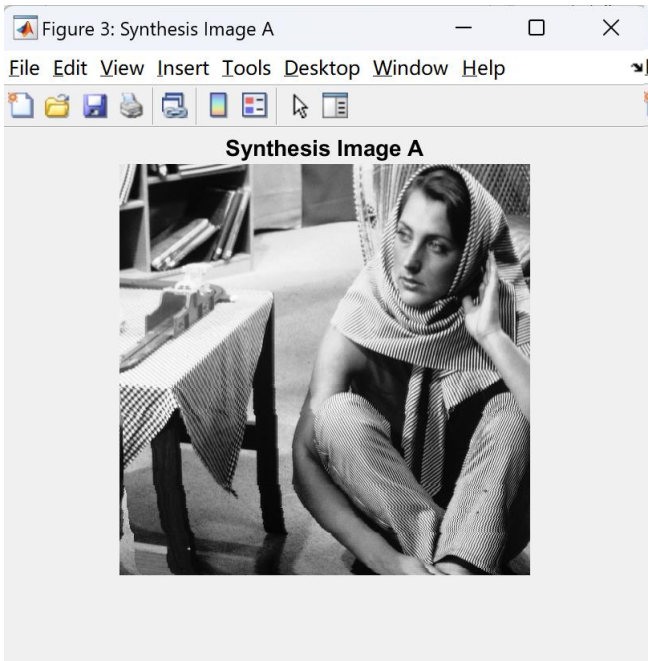
```
    Ext_L(1:2:end, :) = L; % Keep Odd
    Ext_H(2:2:end, :) = H; % Keep Even
    for i = 1: col
        Ext_L(:,i) = Filter(Ext_L(:,i), L_Filter)';
        Ext_H(:,i) = Filter(Ext_H(:,i), H_Filter)';
    end
    img = Ext_L + Ext_H;
end

% PSNR
function DWT_result = PSNR(ori_img, Rec_img, nbit)
    MSE = mean((Rec_img(:) - ori_img(:)).^2);
    MAXI = 2^nbit - 1;
    DWT_result = 10 * log10((MAXI^2) / MSE);
end

```

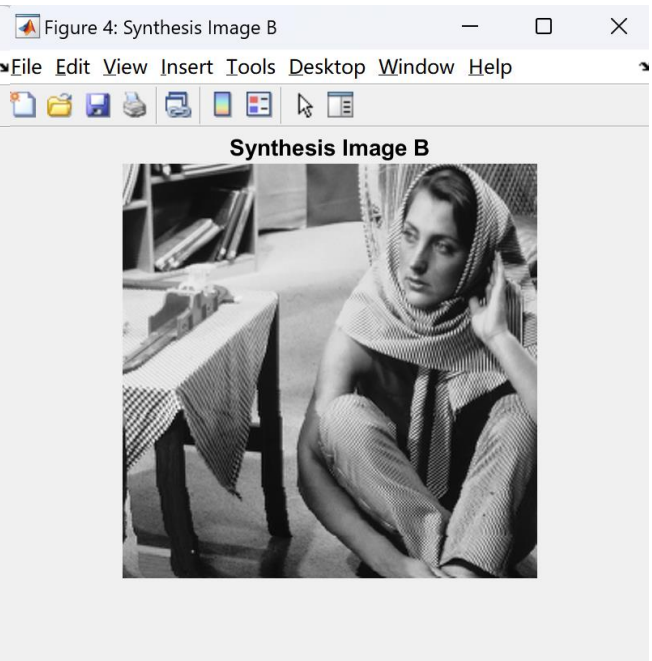
- Result





Original Image	3-Level DWT Image





IDWT synthesis image	IDWT synthesis image(set HL1, LH1, HH1 to 0)

Synthesis result (PSNR)

a) PSNR = 234.2033 dB

b) PSNR = 23.2903 dB