**VLSI DPS HW#2**

**電機四 4109064119 陳柏淳**

Q1. LMS filter design

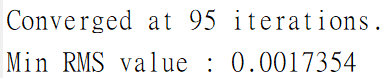
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100101102103104105106107108109110111 | 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    % 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  r(i) = sqrt(mean(e(max(M, i-rms\_step+1):i).^2)); % RMS  % Determine convergence  if r(i) < converge\_value  disp(['Converged at M = ' num2str(M) ', Iteration = ' num2str(i-M+1)]);  min\_M = M;  min\_iter\_count = i - M + 1;  convergence\_flag = true;  break;  end  end  if convergence\_flag  break;  end  end    disp(['Minimum iterations for M converged : ' num2str(min\_M)]);  disp(['Iterations required: ' num2str(min\_iter\_count)]); |

* Result

a. (μ=10-2, Iter=200)

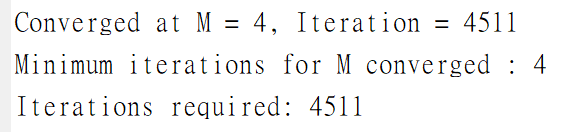
Converged at 95 iterations

Min RMS = 0.0017354

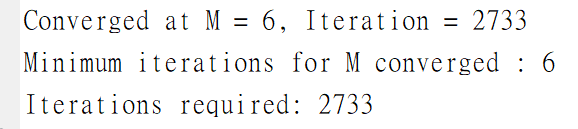


b. The minimum filter length m so that the adaptation can converge in no more than 5000 iterations.

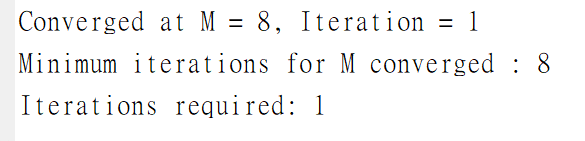
b1. m = 4 when the step size = 10-2



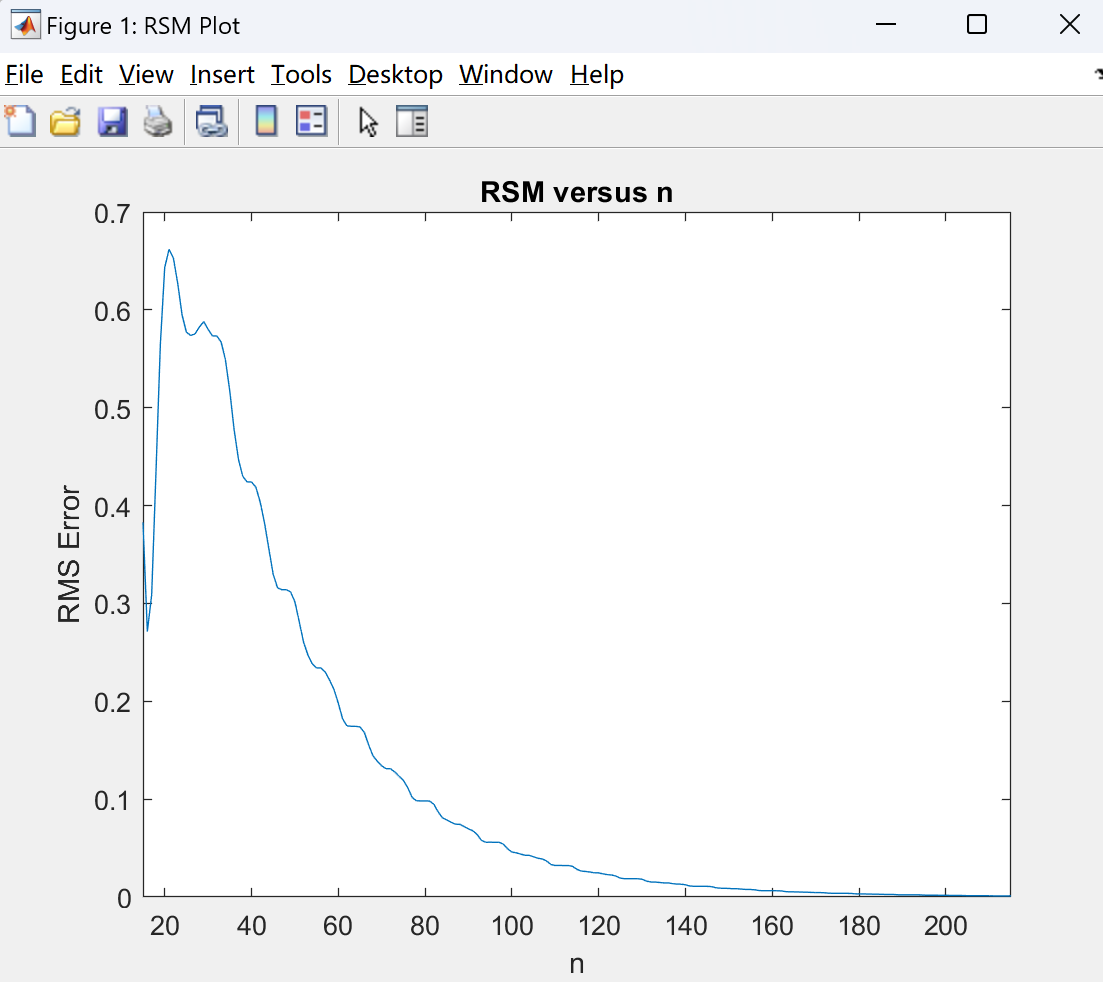
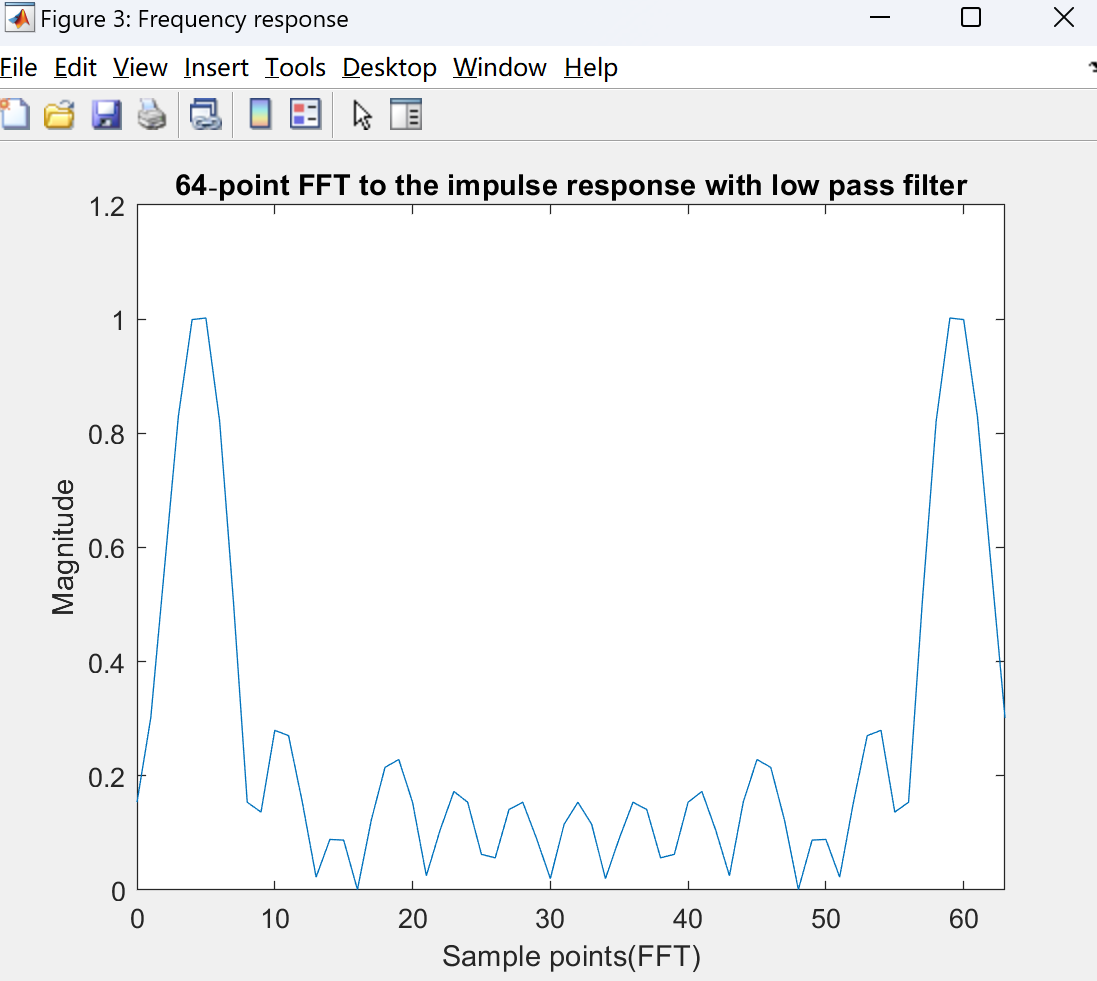
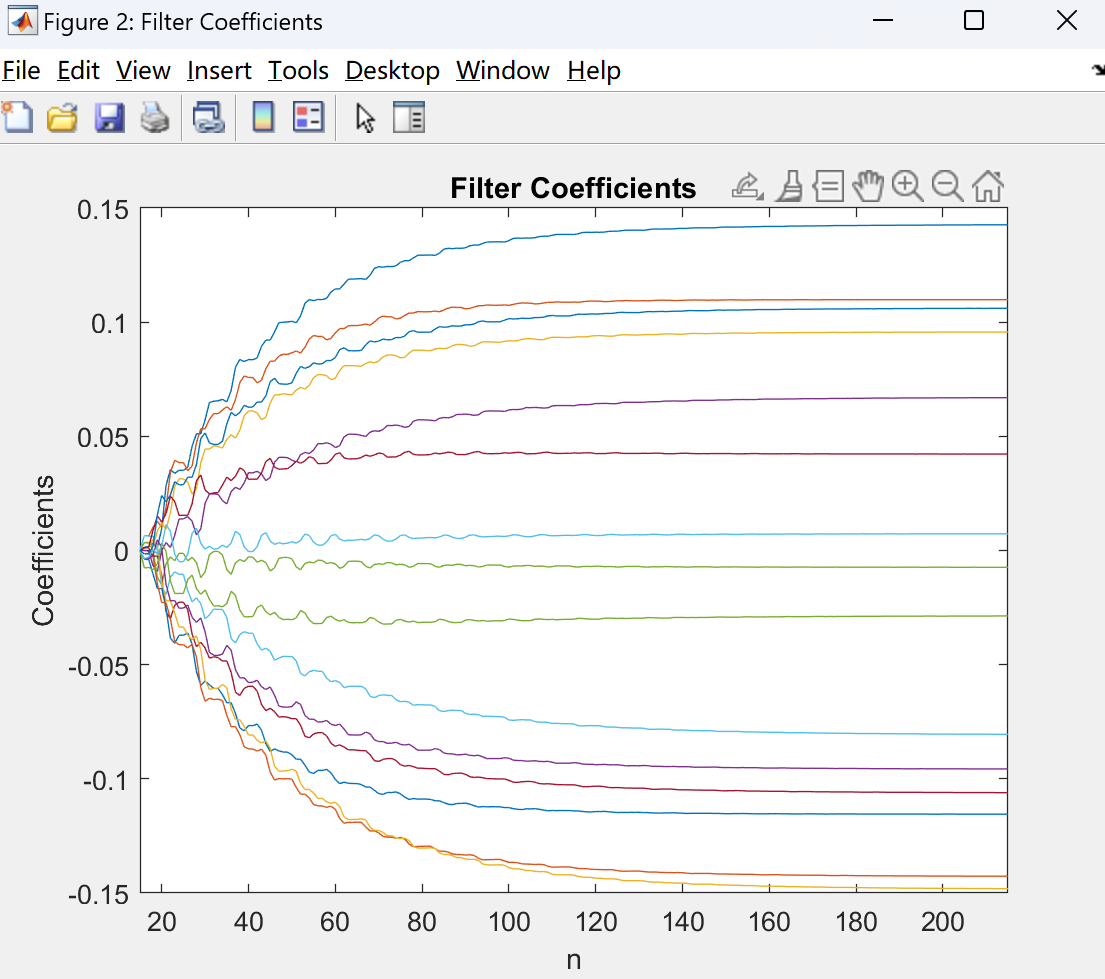
b2. m = 6 when the step size = 10-3



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



C. Plot

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

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.

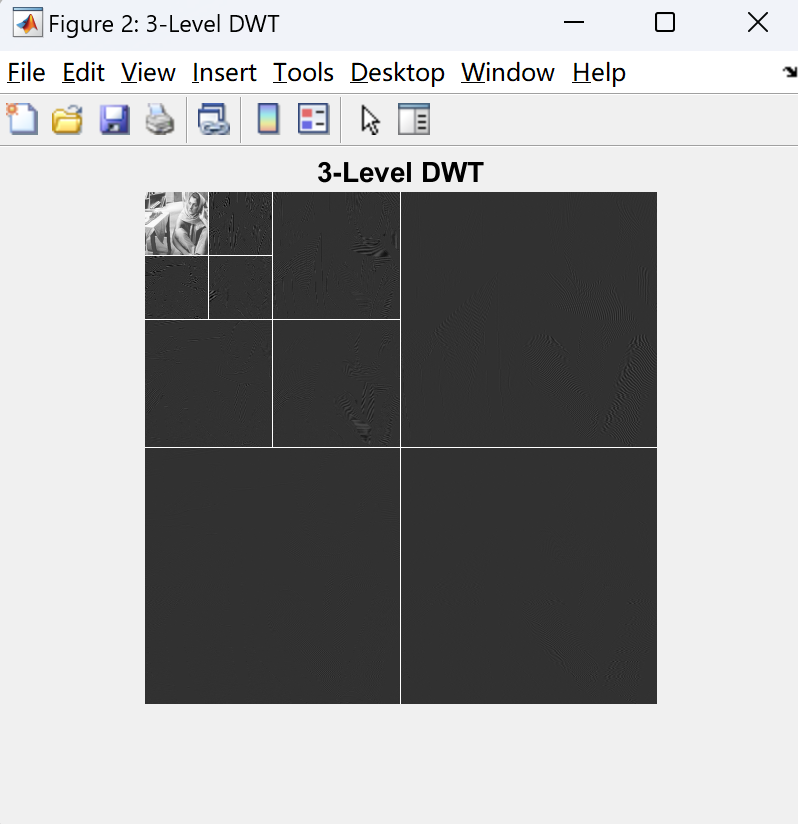
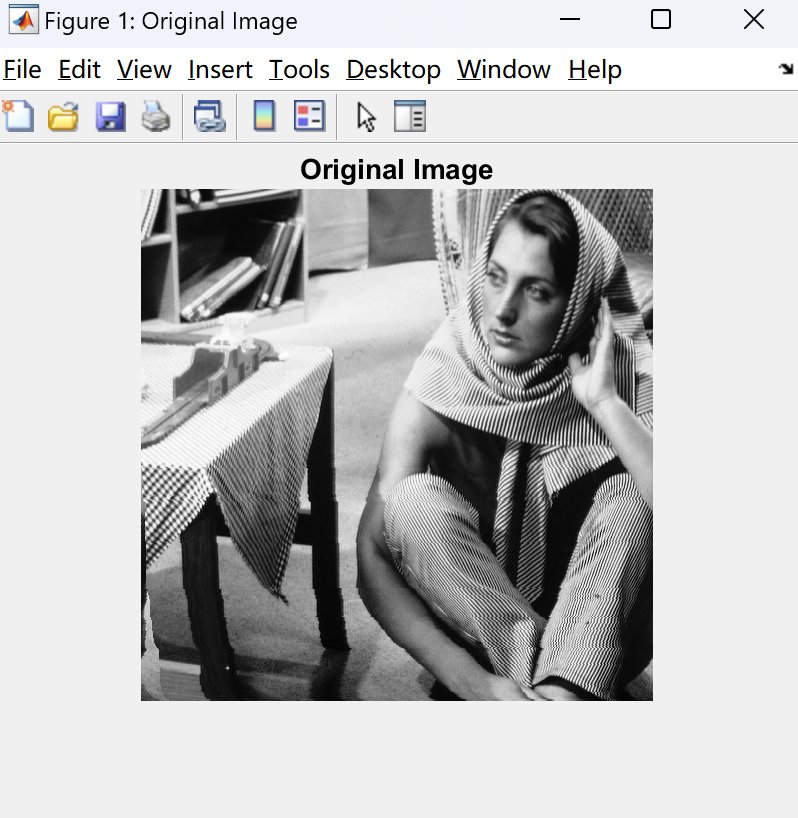
Show the plot of “r(n)” versus “n” and indicate when the filter converges.

95 training samples are required.

Q2. Discrete Wavelet Transform

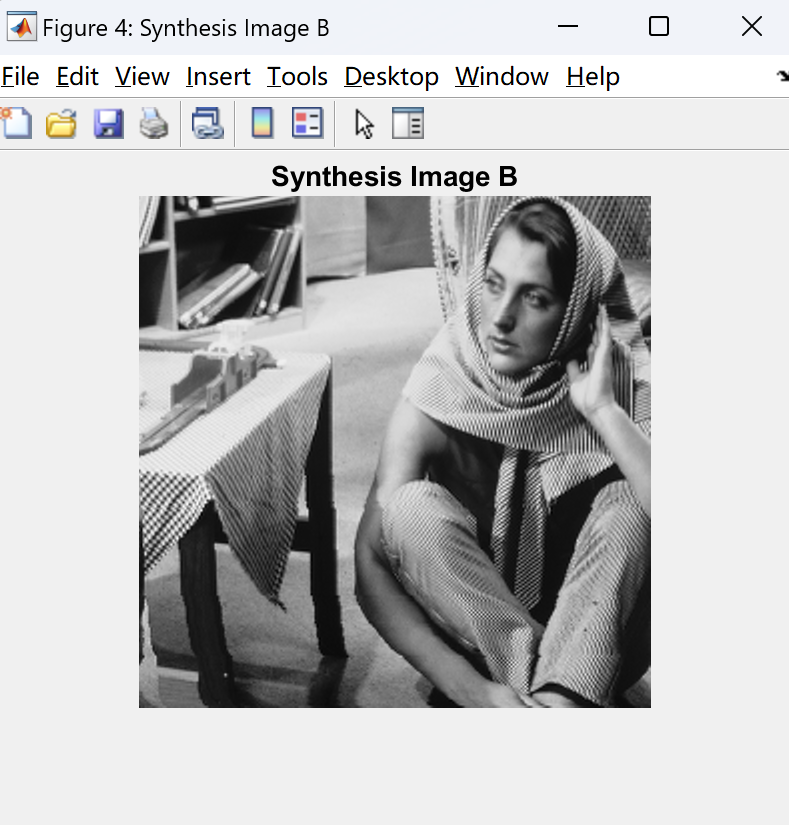
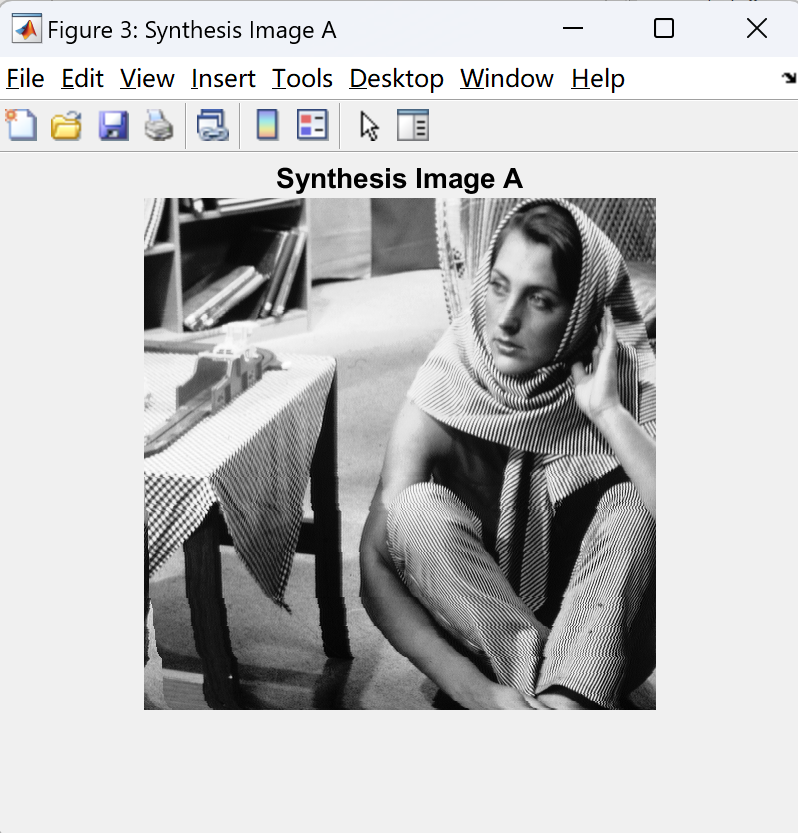
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100101102103104105106107108109110111112113114115116117118119120121122123124125126127128129130131132133134135136137138139140141142143144145146147148149150151152153154155156157158159160161162163164165166167168169170171172173174175176177178179180181182183184185186187188189190191192193194195196197198199200201202203 | 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



3-Level DWT Image

Original Image



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

IDWT synthesis image

Synthesis result (PSNR)

a) PSNR = 234.2033 dB

b) PSNR = 23.2903 dB