HW2 LMS Filter and DCT Filter design

VLSI DSP HW2

Shun-Linag Yeh, NCHU Lab612

3/18 2023

INDEX

Adaptive FIR Low pass filter

Problem

Q1. LMS filter design

For a least mean square (LMS) adaptive filter, assume the filter is of the form finite impulse response (FIR) and 15-tap long (i.e., with 15 coefficients b_0~b_{14} for x(n)~x(n-14)). Given an input signal consisting of 2 frequency components

$s(n) = \sin(2\pi*n/12)+\cos(2\pi*n/4)$

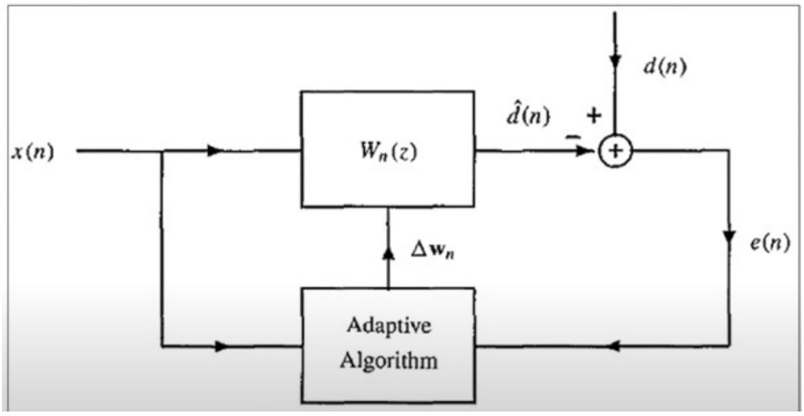
develop an adaptive low pass filter design

Set the target as a low pass filter to remove the high frequency component $\cos(2\pi*n/4)$ and use $\sin(2\pi*n/12)$ as the desired (or training) signal for LMS adaptation. Assume the step size μ is 10^{-2}.

- write a Matlab code to simulate the LMS based adaptive filtering. Calculate the RMS (root mean square) value of the latest 16 prediction errors (i.e., $r = \text{sqrt}((e^2(n) + e^2(n-1) + \ldots + e^2(n-15))/16)$ and the adaptation is considered being converged if this value is less than 10% of RMS (root mean square) value of the desired signal, which equals 0.1/sqrt(2).
- Show the plot of "r" versus "n" and indicate when the filter converges, i.e. how many training samples are required
- Show the plot of filter coefficients $b_i(n)$, for i = 0~14, versus "n" and see if 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 is indeed a low pass one. Note that the input vector to the 64-point FFT is (b_0, b_1, ..., b_{14}, 0,0,.....,0) with 49 trailing zeros.
- Change the step size μ to 10^{-4} and see how the behavior of the adaptive filter changes.
- Conduct simulation with a sufficiently large number of samples to see how small the value of "r" can be (the convergence bias)

Derivation steps

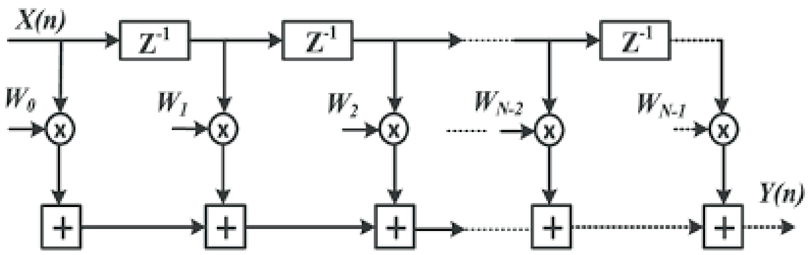
Adaptive Filter specification



1. x(n) is the input signal, wn(z) is the adaptive filter block with coefficients of wn.
2. d_hat(n) is the generated system response and d(n) is the desired signal.
3. e(n) is the error between d_hat(n) and d(n)
4. The adaptive algorithm block determines which kind of policy we should use to find the suitable filter coefficients. In this HW, LMS algorithm is chosen.

The adaptive FIR filter

$$\hat{d}(n) = \sum_{k=0}^{p} \omega_n(k)x(n-k) = \mathbf{w}_n^T \boldsymbol{X}(n)$$



- The desired output is genereated through the p-tap FIR filter design, where wn is the coefficients that gets updated on the fly.

Error function

$$
\begin{aligned}
e(n) &= d(n) - \hat{d}(n) \\
&= d(n) - w_n^T \boldsymbol{X}(n)
\end{aligned}
$$

$$E\{e(n)x^*(n-k)\} = 0 \ ; \ k = 0,1,\dots,p$$

- Error function simply is the difference between the desired signal and the generated system response.
- Ultimate goal is to minimize the autocorrelation between error vector and input signal.

LMS algorithm

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \mu e(n)\boldsymbol{X}^*(n)$$

$$\omega_{n+1} = \omega_n(k) + \mu e(n)\boldsymbol{X}^*(n-k)$$

1. mu is the step sizes for the algorithm, which governs the variability of the coefficients in each iteration.
2. e(n)X*(n) is the factor of auto-correlation between the input signal and the error function.

RMS(Root mean square)

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}}$$

Code

Adaptive Filters

```matlab
function [wn_in_time, rn, wn, dn_hat, en, steps] = lms(xn, dn, mu, L)

    N = length(xn); %Length of input signals
    wn = zeros(1, L); %initialize filter coefficients
    dn_hat = zeros(1, N); % Initialize outputs
    rn = zeros(1, N);
    en = dn - dn_hat; % Error vectors, set as the difference of two signals
    en_initial = en(N - 16:N); % Latest 16 prediction error
    steps = 1; %Steps needed to reach the optimal value
    wn_in_time = zeros(L, N);

    desired_rn = RMS(en_initial, L) * 0.1;
    disp("Desired RMS value");
    disp(desired_rn);

    % FIR filter, constructed from the equation of FIR filter.
    % Starting from Lth signal all the way till length of the whole signals.
    for n = L:N
        % The L-tap coefficient vector
        x1 = xn(n:-1:n - L + 1);
        %Convolution
        dn_hat(n) = wn * x1';
        %Error vector
        en(n) = dn(n) - dn_hat(n);
        %LMS algorithm
        wn = wn + mu * en(n) * x1;

        wn_in_time(:, steps) = wn;

        % RMS calculation
        en_latest = en(n:-1:n - L + 1); % The latest 16 errors of error vector.
        % disp("Latest 16 errors");
        % disp(en_latest);
        rn(steps) = RMS(en_latest, L);

        if rn(steps) <= desired_rn
            break;
        else
            steps = steps + 1;
        end

    end

end
```

RMS

```matlab
function r = RMS(en, L)
    N = length(en); %Length of input signal
    sum = 0;

    for n = 1:N
        sum = sum + (en(n) ^ 2);
    end

    r = sqrt(sum / L);
end
```

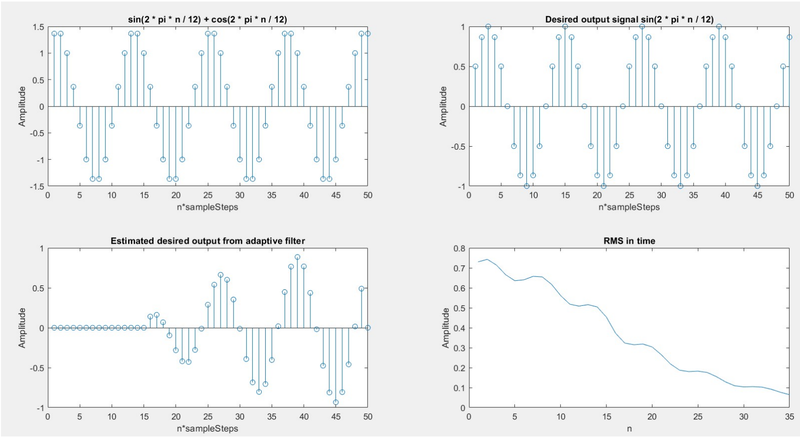
Main drivers

```
clear all;

% Note sequence starting from 1~60
Sample_size = 3000;
n = 1:Sample_size;
L = 15;
mu = 0.0001;

n_s = 1:100;

%Input signal xn(n)
xn = sin(2 * pi * n / 12) + cos(2 * pi * n / 12);
dn = sin(2 * pi * n / 12);

xn_s = sin(2 * pi * n_s / 12) + cos(2 * pi * n_s / 12);
dn_s = sin(2 * pi * n_s / 12);

%plot of original signal for 50 equally sampled value
figure(1);
sampleSteps = 100;
n_spaced = 1:sampleSteps:Sample_size;
subplot(2, 2, 1);
stem(xn(n_spaced));
title('sin(2 * pi * n / 12) + cos(2 * pi * n / 12)'); xlabel('n*sampleSteps'); ylabel('Amplitude');

subplot(2, 2, 2);
stem(dn(n_spaced));
title('Desired output signal sin(2 * pi * n / 12)'); xlabel('n*sampleSteps'); ylabel('Amplitude');

% Adaptive Filter, Look for the minimum samples to reach 10% of LMS
[wn_in_time, rn, wn, dn_hat, en, steps] = lms(xn, dn, mu, L);

% Plot of dn_hat
subplot(2, 2, 3);
stem(dn_hat(n_spaced));
title('Estimated desired output from adaptive filter'); xlabel('n*sampleSteps'); ylabel('Amplitude');

% Plot of rn
subplot(2, 2, 4);
plot(rn(n_spaced));
title('RMS in time'); xlabel('n*sampleSteps'); ylabel('Amplitude');

disp("RMS final value");
disp(rn(steps));
```

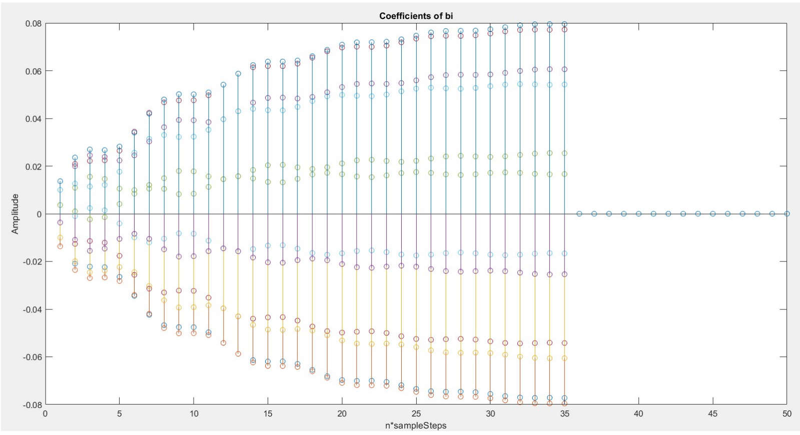
```
disp("Total Steps needed to reach 10% of RMS");
disp(steps);
% Plot of Coefficients v.s. steps
figure(2);
wn_in_time = wn_in_time';
stem(wn_in_time(n_spaced, :));
title('Coefficients of bi'); xlabel('n*sampleSteps'); ylabel('Amplitude');

% FFT for the impulse response of converged filters.
N = 64;
wn_padded = zeros(1, N);
wn_padded(1:L) = wn;

figure(3);

Y = fft(wn, N);
stem(Y); % Note must use stem.
```

Results

Adaptive Filter Response n=100, mu = 0.01, sampleSteps = 1 and RMS over time



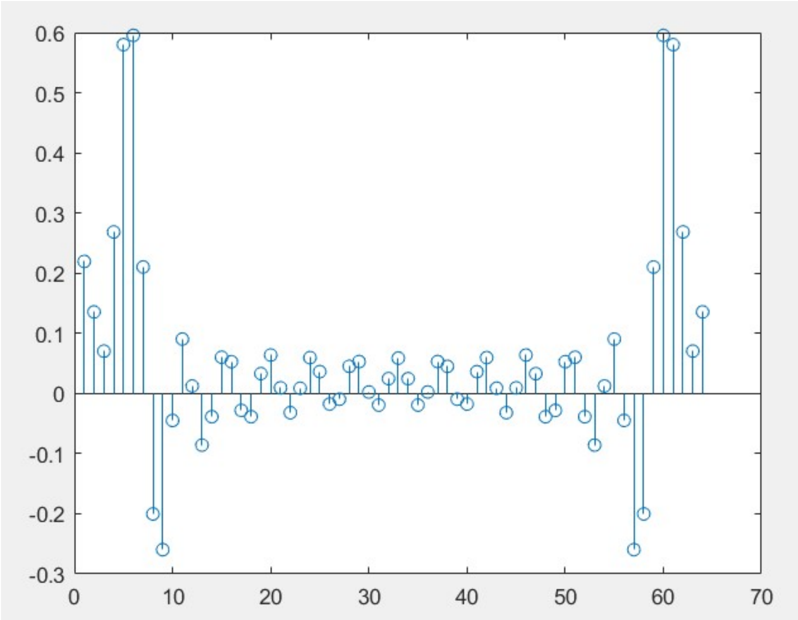
Filter Coefficients over time



Converged steps

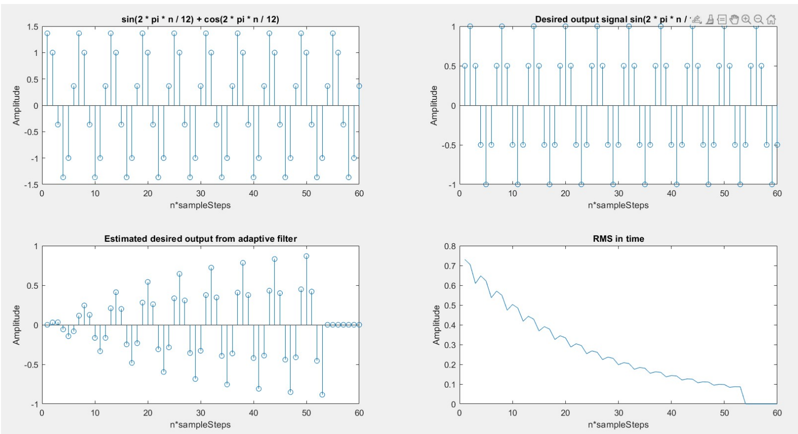
```
RMS final value
    0.0645

Total Steps needed to reach 10% of RMS
    35
```

64-point FFT spectrum

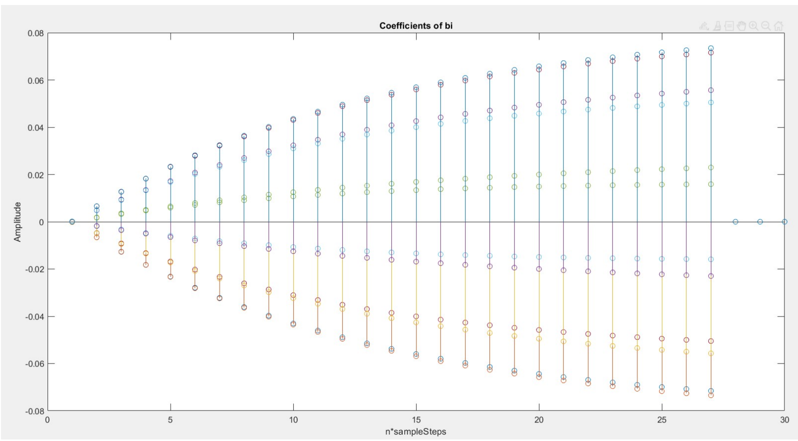


- The response is indeed a Low-Pass filter response.

Adaptive Filter Response n=3000, mu = 0.0001, samepleSteps = 100 and RMS over time



Filter Coefficients over time



Converged steps

```
RMS final value
    0.0762

Total Steps needed to reach 10% of RMS
        2615
```

- Notice the changing of the RMS value responses more drmatically due to smaller step difference. Also it takes longer and more sample points for it to converge.

RMS with large sample size n = 10000

```
RMS final value
   1.2020e-12

Total Steps needed to reach 10% of RMS
        2748
```

- The convergence bias is found, the filter cannot converge any further, it keeps on oscillating between within the convergence bias.

Note

- Due to the fact that sample sizes are large for smaller mu, plotting all of the signals makes analysis hard, thus samples_steps is defined s.t. only a certain multiple of signal sample_steps are selected for plotting.
- The latest 16 prediction errors should be selected for caculation, selecting more than that might yield the wrong results, and the filter would never converge.

Discrete Wavelet transform

Problem

Givens rotation

Tridiagnalization

1. After computing M' = Q'M using givens rotation through the linear transformation of rotation matrices where Q' is products of a series of Rotation matrices G, we yields an upper triangular matrix that is close to tridiagonal form.
2. Replace the upper symmetrical part of M' with the same result as the lower entries where i<j.
3. We can get triangularized matrix M'',where M'' = Q Q' M

QR Iterative Algorithm

Code

Main driver

Tridiagonalization and QR iterative

Givens Rotation

Result

Tridiagonalized matrix

Eigenvalue matrix after eigen-decomposition

Eigenvalue decompose using matlab

Note

1. The result of the Eigen-decomposition differs due to the ordering of the orthnormal basis and the ordering of eigenvectors when calculating the QR matrcies.
2. The eigenvalues are all the same but in different orderings.

Reference

[1] University of South California, Section 4.2.1: Givens Rotations, Math610 Jim Lambers