DSPLab Final Project

SPRING 2017

This is a short report of Implementation of LMS adaptive filter on DSP board dsk6713, Using Codecomposer.

Note: Matlab and CodeComposer codes are attached to the report

University of Tehran, School of Electrical and Computer Engineering

Hossein Souri	810192398	hsouri@ut.ac.ir
MohammadTaghi Badakhshan	810192323	mtbadakhshan@ut.ac.ir

Contents

Table of Figures

"The LMS algorithm has established itself as an important functional block of adaptive signal processing."

Preface

Introduction

As the final project of DSP Lab course we choose LMS adaptive filters. For design and implementing an adaptive filter we should first know how an adaptive filter works. And after that it's necessary to study about LMS algorithm. In this report we first introduce adaptive filters and after that we will focus on LMS algorithm. At the end we make a schedule for doing this project.

Filter structure

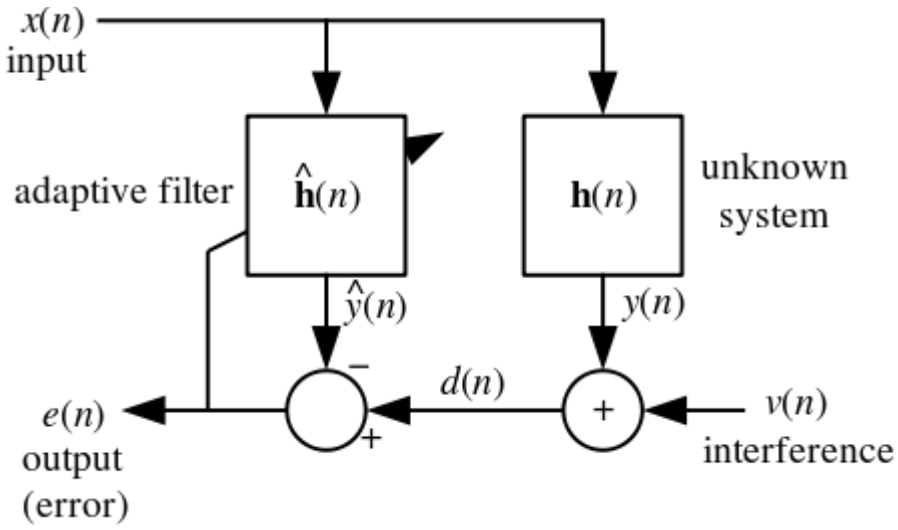

Figure 1 LMS adaptive filter structure

As shown in Fig 1 our purpose in designing a LMS filter is to reach the coefficient of unknown filter .i.e. h(n).

Problem Formulation

This project deals with the LMS algorithm, which is derived from the method of steepest descent by replacing $R = E\{u(n)u^H(n)\}$ and $p = E\{u(n)d^*(n)\}$ with the instantaneous estimates $\hat{R}(n) = u(n)u^H(n)$ and $\hat{p}(n) = u(n)d^*(n)$, respectively.

The recursive equations for the error and the filter coefficients of the least mean square algorithm are given by

$$e(n) = d(n) - w(n) * u(n) \rightarrow e(n) = d(n) - \hat{w}^H(n)u(N-n)$$

$$\hat{w}(n+1) = \hat{w}(n) + \mu e^*(n)u(N-n)$$

Convergence and stability in the mean

As the LMS algorithm does not use the exact values of the expectations, the weights would never reach the optimal weights in the absolute sense, but a convergence is possible in mean. That is, even though the weights may change by small amounts, it changes about the optimal weights. However, if the variance with which the weights change, is large, convergence in mean would be misleading. This problem may occur, if the value of step-size μ is not chosen properly.

If μ is chosen to be large, the amount with which the weights change depends heavily on the gradient estimate, and so the weights may change by a large value so that gradient which was negative at the first instant may now become positive. And at the second instant, the weight may change in the opposite direction by a large amount because of the negative gradient and would thus keep oscillating with a large variance about the optimal weights. On the other hand if μ is chosen to be too small, time to converge to the optimal weights will be too large.

Thus, an upper bound on μ is needed which is given as $0 < \mu < \frac{2}{\lambda_{max}}$.

Matlab Simulation

Before writing C code first we implement filter in Matlab and simulate it to validate the LMS algorithm. If it's true then we will implement it in Codecomposer.

```
clc
close all
clear all

N=input('length of sequence N = ');
t=linspace(1,N,N);
d=cos(2*pi*t/N);
x=sin(2*pi*t/N);
w=zeros(1,N);
mu=input('mu = ');

M=input('Number of irretate = ');
for j=1:M

for i=1:N
   e(i) = d(i) - w(i)' * x(N-i+1);
   w(i) = w(i) + mu * e(i) * x(N-i+1);
end

for i=1:N
yd(i) = w(i)' * x(N-i+1);
end

end
figure(1);
plot(t,d)
ylabel('Desired Signal')
xlabel('n')
grid on

figure(2);
plot(t,x)
ylabel('Input Signal+Noise')
xlabel('n')
grid on

figure(3)
plot(t,e)
ylabel('Error')
xlabel('n')
grid on
```

```
figure(4)
plot(t,yd)
ylabel('Adaptive Desired output')
xlabel('n')
grid on

figure(5)
plot(t,w)
ylabel('Filter coefficients')
xlabel('n')
grid on
```

The purpose of this code is to reach the desired signal $\cos\left(\frac{2\pi t}{N}\right)$ from input signal $\sin\left(\frac{2\pi t}{N}\right)$. In the following result we set N = 513, $\mu = 0.2$ and M = 20000.

Result

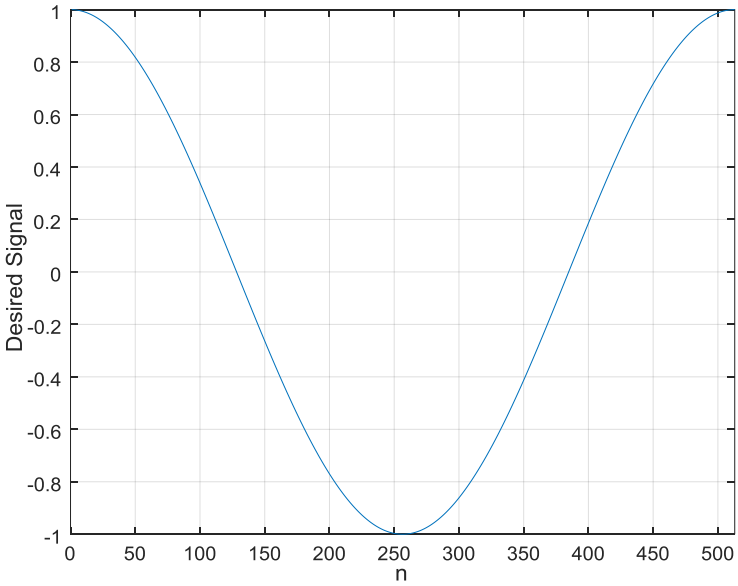

Figure 2 Desired signal $\cos\left(\frac{2\pi t}{N}\right)$

Matlab Simulation

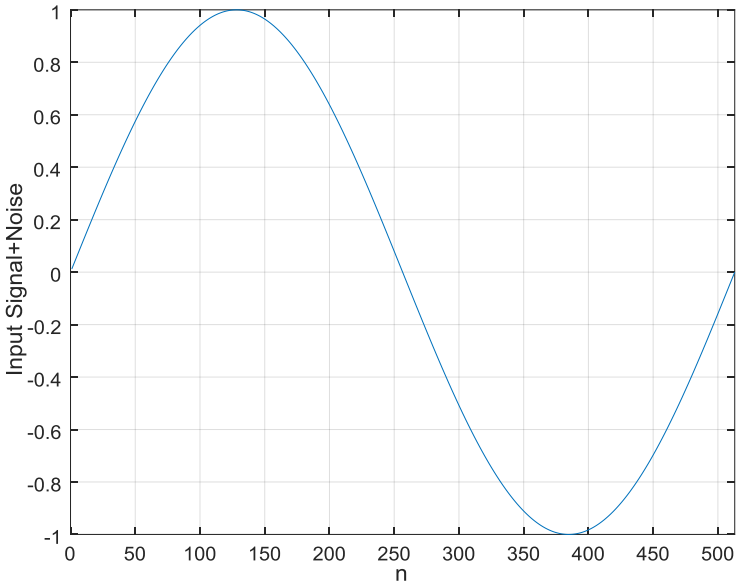

Figure 3 Input signal $sin\left(\frac{2\pi t}{N}\right)$

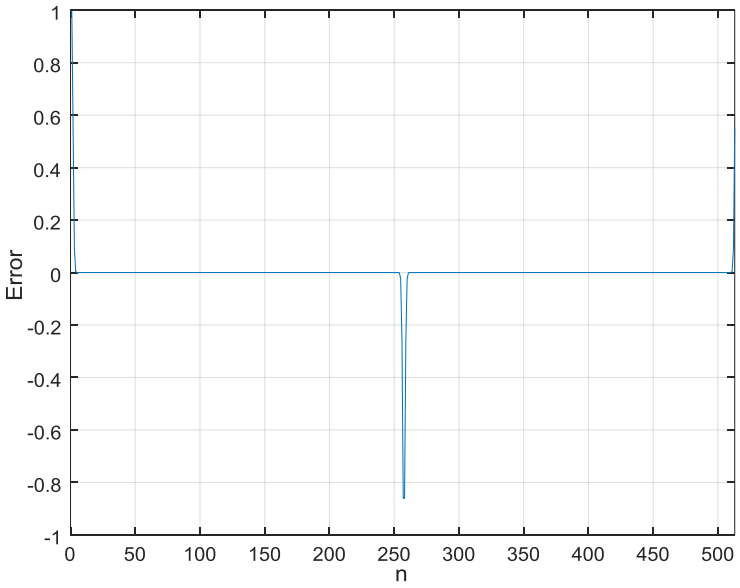

Figure 4 Error signal

Matlab Simulation

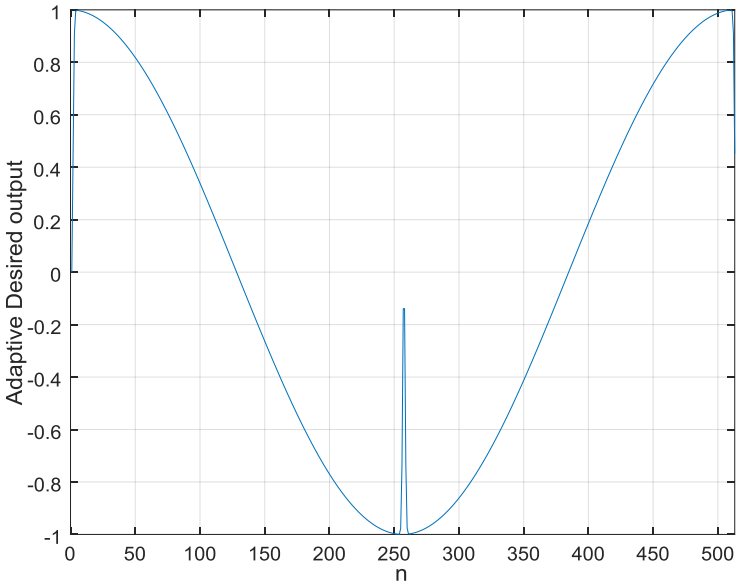

Figure 5 Filter output

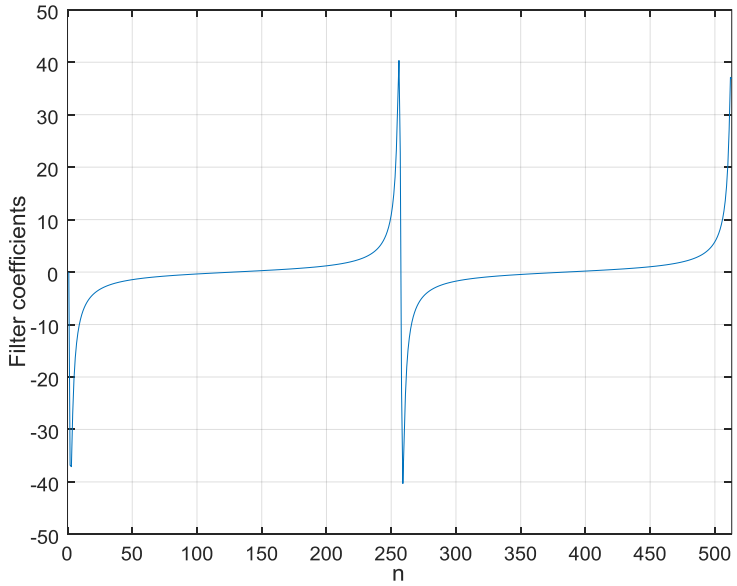

Figure 6 Filter coefficients

Matlab Simulation

As you see in Fig 6 the coefficients of adaptive filter are like a derivative filter, and its true base on theoretical knowledge because we know cos is derivation of sin.

Obviously we see that the filter is a bit different with an ideal derivative filter. It can be explained by convergence coefficient i.e. step-size (μ) and the number of iterations (M).

As we decrease μ and increase M we see that the filter is becoming more similar to derivative filter.

In the following Matlab simulation we change the μ to 0.001 and M to 40000.

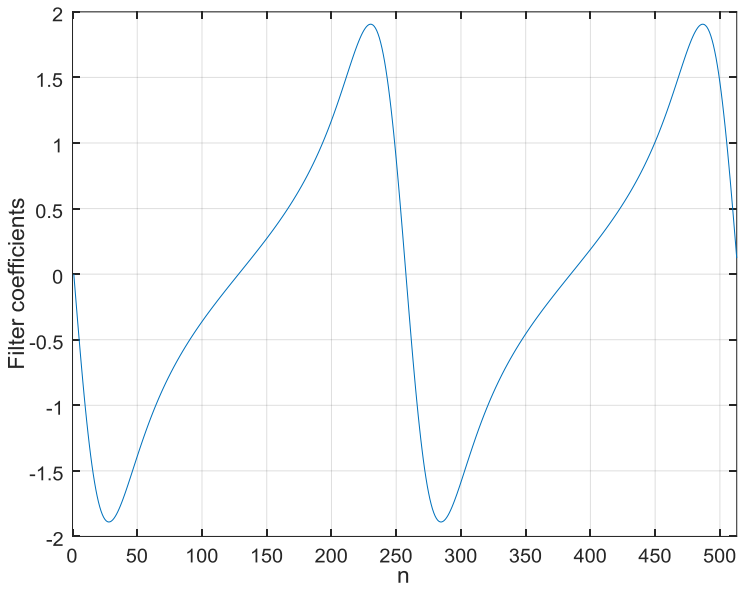

Figure 7 Filter coefficients after changing μ and M

The result validates our code.

CodeComposer Implementation

Now we use previous Matlab code and rewrite it in the C language in Codecomposer. The code is in the following.

```
#include "init.h"
#include <math.h>

#define M 500 // order of filter
#define mu 0.002 //convergnece rate
#define pi 3.14159265359
#define Len 513

float noise[513];
float noise_filter[513];

float err_nrt[513]={0}, w_nrt[Len]={0}, out_nrt[513]={0};


void main(){
	int i,j,Window;

	float mu_nrt = 0.2;


	for(i=0; i<513; i++){
		err_nrt[i] = 0;
		out_nrt[i] = 0;
	}
	for(i=0; i<Len; i++){
		w_nrt[i] = 0;
	}
	for(i=0; i<513; i++){
		noise[i] = sin(2*pi*i/513);
		noise_filter[i] = cos(2*pi*i/513);
	}


for(Window=0; Window<513;Window=Window+Len){
	for(i=0; i<20000; i++){
		for(j=0; j<Len; j++){
			err_nrt[Window+j] = noise_filter[Window+j] -
w_nrt[j]*noise[Len - Window+j];
			w_nrt[j] = w_nrt[j] +
mu_nrt*err_nrt[Window+j]*noise[Len - Window+j];
		}

		for(j=0; j<Len; j++){
```

CodeComposer Implementation

```
                    out_nrt[Window+j] = w_nrt[j] * noise[Len - Window +
j];

                }
        }

}

}
```

Result

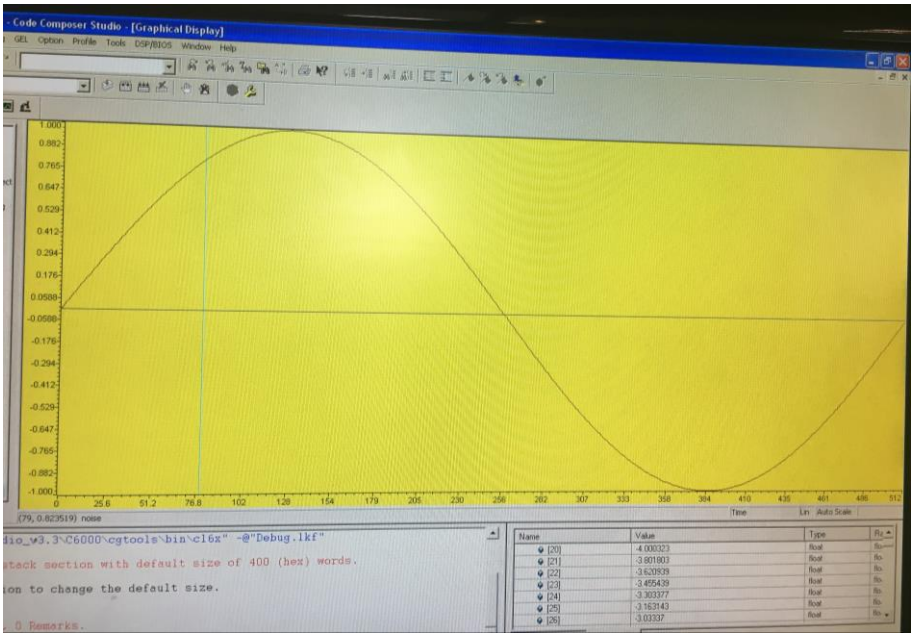

Figure 8 Input signal $\sin\left(\frac{2\pi t}{N}\right)$

CodeComposer Implementation

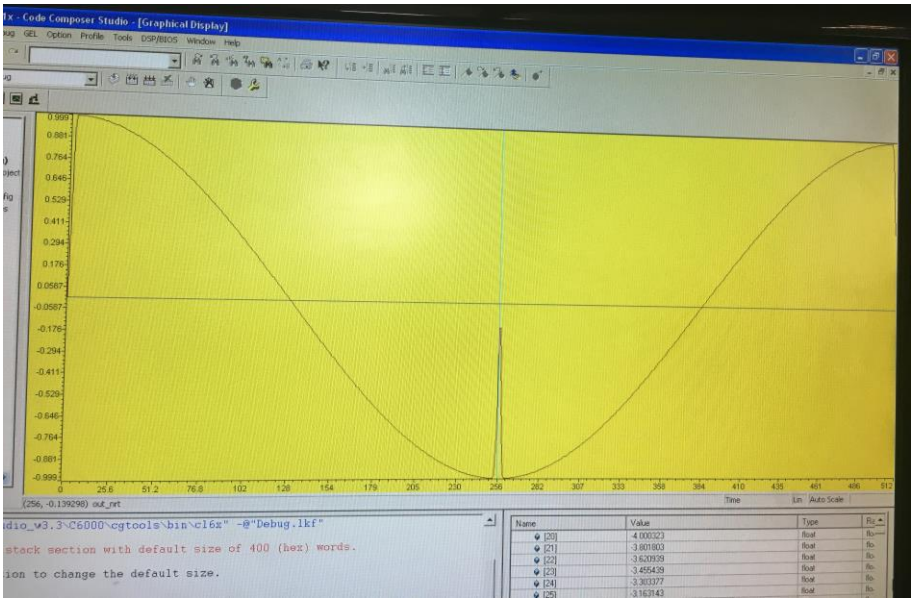

Figure 9 Output signal

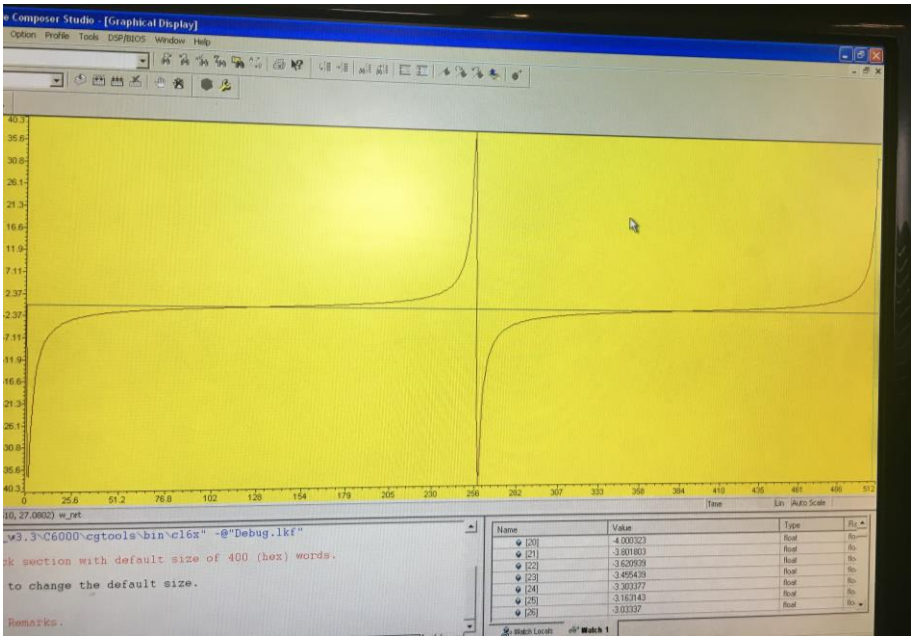

Figure 10 Filter coefficients

CodeComposer Implementation

As we can see in the result, output is like our desired signal $cos\left(\frac{2\pi t}{N}\right)$ and we should mention that differences is because of convergence coefficient and the number of iteration means that increasing the number of iterations and decreasing the step-size error will reach to zero.

Comparing Matlab and CodeComposer Results

Filter Coefficients

When you have a document that shows a lot of numbers, it's a good idea to have a little text that explains the numbers. You can do that here.

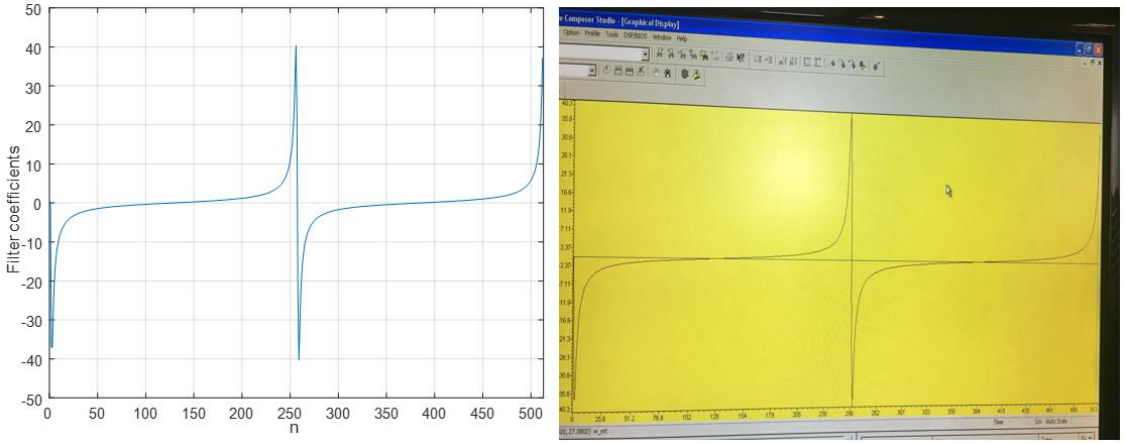

Figure 11 Comparison between filter coefficinets of Matlab and Codecomposer

The results of both Matlab and Codecomposer are the same and they are similar to ideal derivative filter. And as we explained before the difference is because of convergence coefficients and number of iterations.

In the following we show the ideal and calculated filter coefficients. With increasing the number of iterations (M) and decreasing the convergence coefficients (μ), the calculated filter will be equal to the ideal filter.

Comparing Matlab and CodeComposer Results

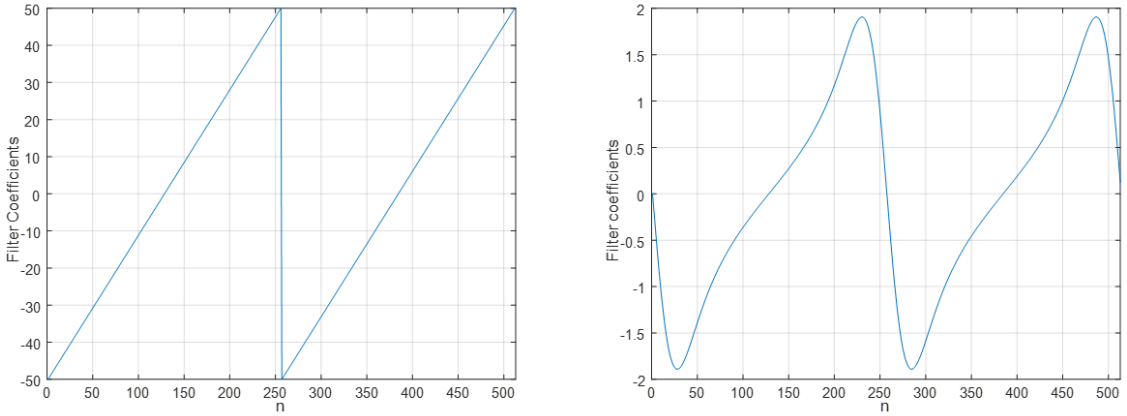

Figure 12 Comparison between ideal and calculated filter coefficients

Note: The ideal filter is calculated in discrete time domain it is periodic.

Conclusion

The LMS algorithm has established itself as an important functional block of adaptive signal processing. It offers some highly desirable features:

- Simplicity of implementation, be that in software or hardware form.
- Ability to operate satisfactorily in an unknown environment.
- Ability to track time variations of input statistics.

Indeed, the simplicity of the LMS algorithm has made it the standard against which other linear adaptive filtering algorithms are benchmarked.

Compare the final filter coefficients (w) obtained by the LMS algorithm with the filter that it should identify (h). If the coefficients are equal, your LMS algorithm is correct.

For reaching the above purpose we did 5 steps below:

1- Study the theory of problem.
2- Implement the filter in Matlab and test it.
3- Implement the filter in Codecomposer.
4- Comparing the results of Matlab and Codecomposer.
5- Validate the results with Ideal filter.

Contact Information

To replace a photo with your own, right-click it and then choose Change Picture.

Hossein Souri
EE student
Tel +98 918 213 25 30
hsouri@ut.ac.ir

MohammadTaghi Badakhshan
EE student
Tel +98 936 7844 177
mtbadakhshan@ut.ac.ir

University Information

University of Tehran, School of Electrical and Computer Engineering
North Kargar Ave., Tehran, Iran
Tel +98 21 8800 0939
Postal Code: 1417943966
http://ece.ut.ac.ir