DIGITAL SIGNAL PROCESSING

ECE – 2003

PROJECT REPORT

ACTIVE NOISE CANCELLATION USING ADAPTIVE FILTERS

Under the Guidance of

Prof. Dr. Malaya Kumar Hota

School of Electronics and Communication Engineering

Vellore Institute of Technology (VIT)

Vellore, TamilNadu - 632014

Team Members

Ayushman Mookherjee – 18BEC0888

Shatakshi Singh – 18BEC0879

Hemant Mangal – 18BEC0582

Table of Content

ABSTRACT

This project contains the basic principles and background mechanism of Adaptive Noise Cancellation and its applications. Adaptive Noise Cancellation is a unique process of detecting additive noise in the signal and subtract it from the signal. It is a simple method that can attain some remarkable results in the field of noise cancellation. It needs two inputs - a primary input containing the corrupted signal and a reference input containing noise correlated in some unknown way with the primary noise. The reference input is adaptively filtered and subtracted from the primary input to obtain the signal estimate.

The effect of uncorrelated noises in primary and reference inputs, and presence of signal components in the reference input on the ANC performance is investigated. It is shown that in the absence of uncorrelated noises and when the reference is free of signal, noise in the primary input can be essentially eliminated without signal distortion. Along with this the usage of the Least mean square algorithm as an adaptive line enhancer, notch filter and lastly as a bias/drift removal is also studied.

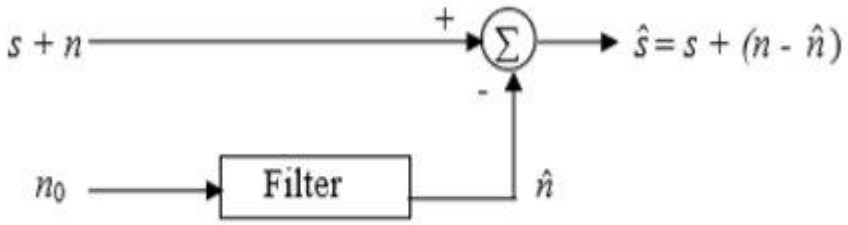
In the field of Adaptive Noise Cancelling, there are various algorithms which are formulated and used over the years. In this project we have taken the three widely used algorithms namely LMS, NLMS and LLMS algorithms and have done a comparative study by cancelling out echo using a given dataset. Echo return Loss enhancement performance values of each were calculated and hence a result was reached to.

Computer simulations for all cases are carried out using MatLab software and experimental results are presented that illustrate the usefulness of Adaptive Noise Cancellation Techniques.

INTRODUCTION

The usual method of estimating a signal corrupted by additive noise is to pass it through a filter that tends to suppress the noise while leaving the signal relatively unchanged i.e. direct filtering

Noise Cancellation is a variation of optimal filtering that involves producing an estimate of the noise by filtering the reference input and then subtracting this noise estimate from the primary input containing both signal and noise.



Subtracting noise from a received signal involves the risk of distorting the signal and if done improperly, it may lead to an increase in the noise level. This requires that the noise estimate n ˆ should be an exact replica of n. If it were possible to know the relationship between n and n ^, or the characteristics of the channels transmitting noise from the noise source to the primary and reference inputs are known, it would be possible to make n ˆ a close estimate of n by designing a fixed filter. However, since the characteristics of the transmission paths are not known and are unpredictable, filtering and subtraction are controlled by an adaptive process. With adaptive control, noise reduction can be accomplished with little risk of distorting the signal.

PRINCIPLES AND BACKGROUND METHODS

- **ANC (ADAPTIVE NOISE CONCELLATION)**

An Adaptive Noise Canceller (ANC) has two inputs – primary and reference. The primary input receives a signal s from the signal source that is corrupted by the presence of noise n uncorrelated with the signal. The reference input receives a noise n0 uncorrelated with the signal but correlated in some way with the noise n. The noise no passes through a filter to produce an output n ˆ that is a close estimate of primary input noise. This noise estimate is subtracted from the corrupted signal to produce an estimate of the signal at s ˆ, the ANC system.

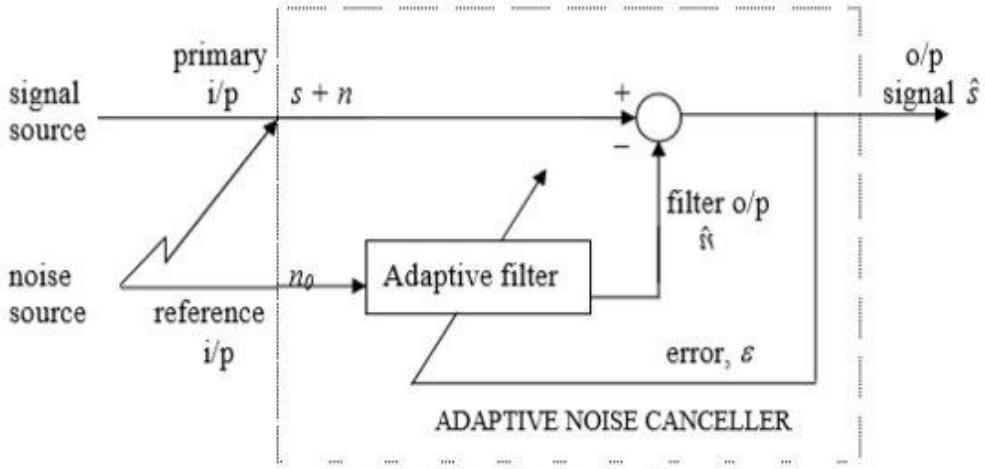


Fig. 1 Adaptive Noise Canceller

In noise canceling systems a practical objective is to produce a system output sˆ= s + n – nˆ that is a best fit in the least squares sense to the signals.

- **EFFECT OF UNCORRELATED NOISE**

The adaptive noise canceller works on the principle of correlation cancellation i.e., the ANC output contains the primary input signals with the component whose correlated estimate is available at the reference input, removed. Thus the ANC is capable of removing only that noise which is correlated with the reference input. Presence of uncorrelated noises in both primary and reference inputs degrades the performance of the ANC. Thus, it is important to study the effect of these uncorrelated noises.

Uncorrelated noise in primary input

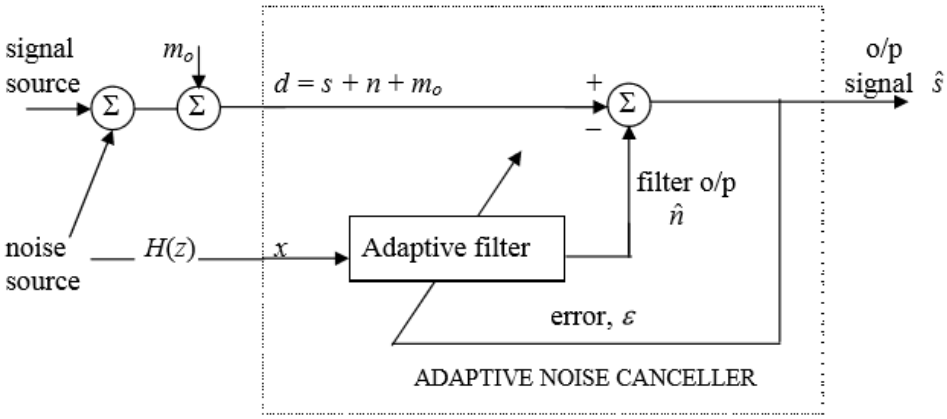


Fig. 2 ANC with uncorrelated noise m_0 in primary input

Uncorrelated noise in the reference input

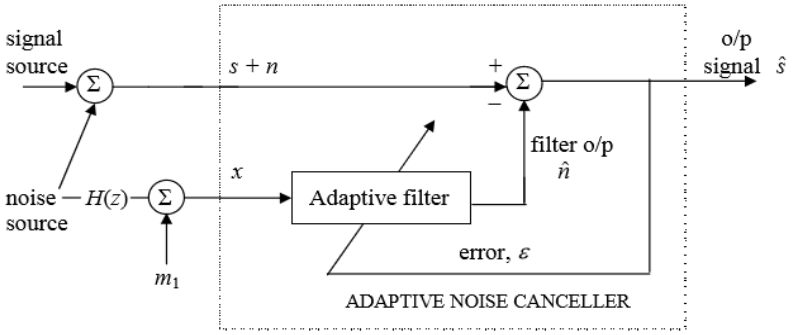


Fig. 3 ANC with uncorrelated noise in reference input

We see that the filter transfer function now cannot equalize the effect of the channel and the filter output is only an approximate estimate of primary noise n.

- **EFFECT OF SIGNAL COMPONENT**

Often low-level signal components may be present in the reference input. The adaptive noise canceller is a correlation canceller, as mentioned previously and hence presence of signal components in the reference input will cause some cancellation of the signal also. This also causes degradation of the ANC system performance. Since the reference input is usually obtained from points in the noise filed where the signal strength is small, it becomes essential to investigate whether the signal distortion due to reference signal components can outweigh the improvement in the signal-to noise ratio provided by the ANC.

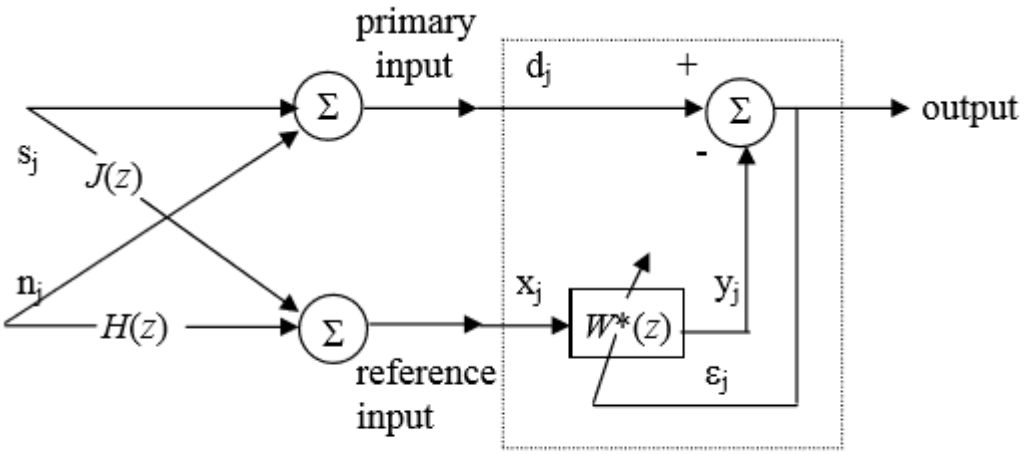


Fig. 4 ANC with signal components in reference input

If the signal-to-noise density ratio at the reference input is low, the output noise will be low, i.e. the smaller the signal components in the reference input, the more perfectly the noise will be cancelled.

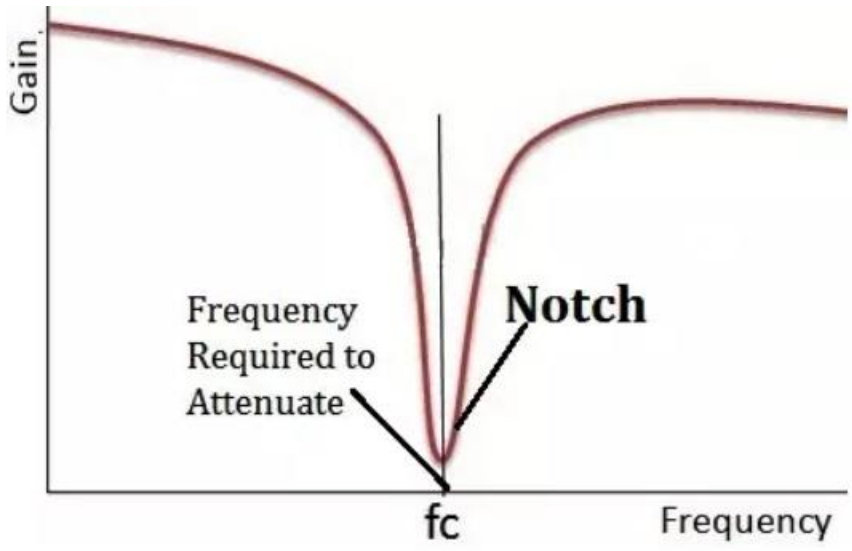
If the signal-to-noise density ratio in the primary input is low, the filter will be trained most effectively to cancel the noise rather than the signal and consequently output noise will be low.

- **WORKING OF A NOTCH FILTER**

A Notch filter is nothing but a narrow band-stop filter. If the stopband of band-stop filter is very narrow and highly attenuated over a few hertz, then that special type of band-stop filter is known as Notch filter. Since the stop band of Notch filter is narrow up to few Hz so the Notch filter is also known as Narrow band stop filter in some cases. It is a highly selective (High Q value) form of band-stop filter which can be used to block a single or very small band of frequencies rather than whole bandwidth of different frequencies. It acts as a gain for one frequency component and attenuator for all other frequencies.

- **Notch Filter Frequency Response**

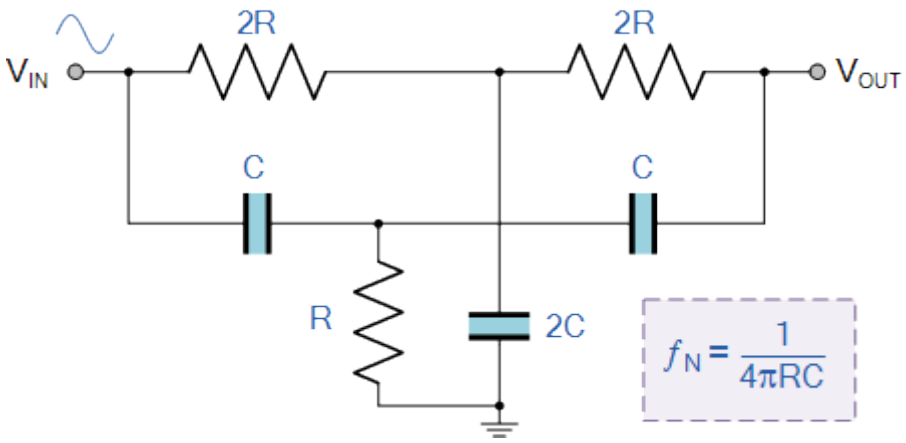
The ideal response of any notch filter would be a completely flat response over the usable range with the exception of notch frequency as shown in the figure below.



- **Notch Filter Circuit**

The most common Notch filter design is the twin-T notch filter network. In its basic form, the twin-T also called a parallel-T configuration. It consists of two RC branches in the form of two tee sections connected in parallel.

The basic twin-T Notch filter circuit is shown in the figure below as:

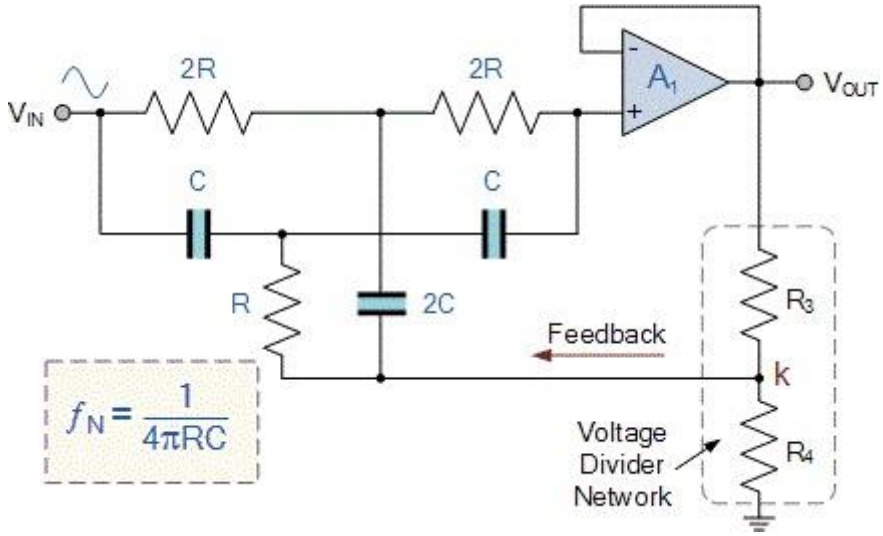


The upper T-configuration of resistor 2R and capacitor 2C form the low pass filter section, whereas the lower T-configuration of resistor R and capacitor C form the high pass filter section of the design.

The frequency at which this basic twin-T notch filter design offers maximum attenuation is called **Notch frequency**. So notch frequency is formulated as:

$$f_N = \frac{1}{4\pi RC}$$

For obtaining a high level of attenuation and narrow notch, an operational amplifier is used to design a single Op-Amp twin-T notch filter circuit. The single Op-Amp twin-T notch filter circuit is shown in the below figure as:



- **Notch Filter Applications**

In different technologies, Notch filters are used in different ways.

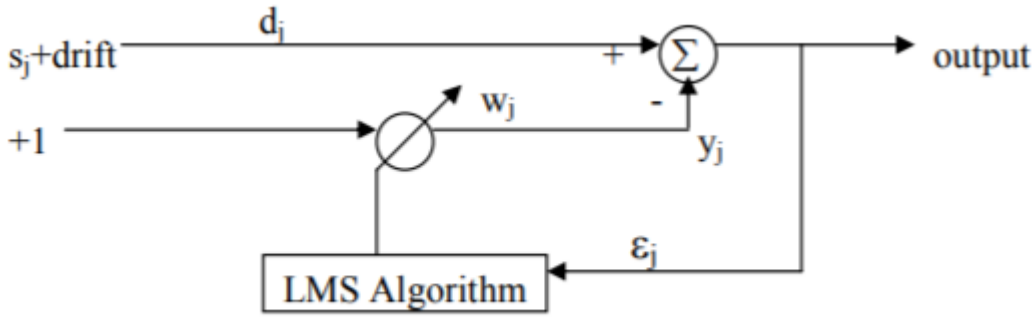
- In communication electronics, the signal is distorted due to some harmonics (Noise) which makes the original signal to interfere with noise signal which leads to error in the output. Thus notch filters are used to eliminate these unwanted frequencies of harmonics.
- These filters are used by musicians in high-quality audio applications such as graphic equalizer, synthesizer and PA system.
- In telephone technology, Notch filters are also used as the telephone line noise (Harmonics) reducer and DSL performance of internet service.
- Also the notch filters are widely used in the electric guitar amplifiers. Actually, the electric guitar produces a ‘hum’ at 60 Hz frequency. Then this filter is used to reduce that ‘hum’ by rejecting 60 Hz in order to amplify the signal produced by the guitar amplifier and make it the best equipment.

These are also used in some of the acoustic applications like Mandolin, Bass instrument amplifiers, etc.

- It is also used in Optical communication technologies. The best example is Raman Spectroscopy. At the end of optical fiber, there may be some interfering frequencies of light that make the distortions in the light beam of optical fiber. Then these distortions are eliminated by the narrow band-stop filters.
- In image and signal processing, these filters are highly preferred to eliminate noise.
- It is also used to reduce the static on the radio, which is commonly used in our daily life.
- These filters are also used in medical field applications, i.e., in biomedical instruments like ECG for removing lines.

- **BIAS / DRIFT REMOVAL**

The use of a bias weight in an adaptive filter to cancel low-frequency drift in the primary input is a special case of notch filtering with the notch at zero frequency. A bias weight is incorporated to cancel dc level or bias and hence is fed with a reference input set to a constant value of one. The value of the weight is updated to match the dc level to be cancelled. Because there is no need to match the phase of the signal, only one weight is needed.



The transfer function from the primary input to the noise canceller output is now derived. The expression of the output of the adaptive filter yj is given by

$$y_j = w_j \,.1 = w_j$$

The bias weight w is updated according to the LMS update equation

$$w_{j+1} = w_j + 2\mu(\varepsilon_j . 1)$$
$$y_{j+1} = y_j + 2\mu(d_j - y_j)$$
$$= (1-2\mu)y_j + 2\mu d_j$$

Taking the z-transform of both the sides yields the steady-state solution:

$$Y(z) = \frac{2\mu}{z - (1 - 2\mu)} D(z)$$

Z-transform of the error signal is

$$E(z) = D(z) - Y(z)$$
$$= \frac{z - 1}{z - (1 - 2\mu)} D(z)$$

The transfer function is now

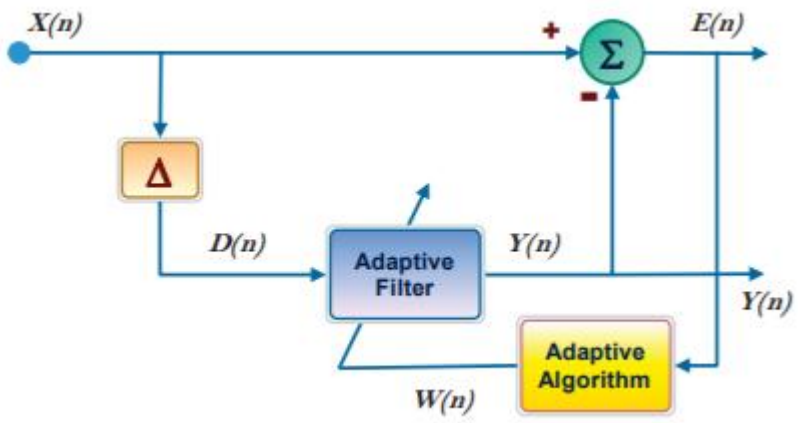
$$H(z) = \frac{E(z)}{D(z)} = \frac{z - 1}{z - (1 - 2\mu)}$$

This shows that the bias-weight filter is a high pass filter with a zero on the unit circle at zero frequency and a pole on the real axis at a distance 2μ to the left of the zero. The smaller the μ, the closer is the location of the pole and the zero, and hence the notch is precisely at zero frequency i.e. only dc level is removed. The single-weight noise canceller acting as a high-pass filter is capable of removing not only a constant bias but also slowly varying drift in the primary input. If the bias level drifts and this drift is slow enough, the bias weight adjusts adaptively to track and cancel the drift. Using a bias weight alongwith the normal weights in an ANC can accomplish bias or drift removal simultaneously with cancellation of periodic or stochastic interference.

- **ADAPTIVE LINE ENHANCER**

Adaptive line enhancement (ALE) refers to the case where a noisy signal, *x(n)* consisting of a sinusoidal component, s*(n)* is available and the requirement is to remove the noise part of the signal, *n(n)*. This may be achieved by the arrangement shown below. It consists of a de-correlation stage, symbolized and an adaptive predictor. The de-correlation stage attempts to remove any correlation that may exist between the samples of noise, by shifting them samples Δ apart. As a result, the predictor can only make a prediction about the sinusoidal component of *u(n)*, and when adapted to minimize the instantaneous squared error output, *e(n)*, the line enhancer will be a filter optimized (the Wiener solution) or tuned to the sinusoidal component.

There are several techniques used to process biomedical signals but these systems analyse the input signal, the noise signal and the output signals. ALE is similar to the ANC architecture but ANC uses two input signals whereas the ALE uses the input signal as the reference signal so the output signal is nearly error-free. The proposed ALE – LMS architecture was implemented to process and separate the signal at the same time in order to analyse the real time input signals as shown in Figure.



In this architecture, the input signal is taken to be the reference signal to obtain the minimum error rate of the output signal. The adaptive algorithms provide feedback to the system to update the FIR filter coefficient. The adaptive algorithms are classified into Least Mean Square (LMS. The LMS algorithms

are specific to different applications. For noise cancellation we proposed LMS algorithm which update the adaptive FIR filter coefficients. The proposed system processes the signals in an iterative manner to ensure minimum error and the output of the adaptive filter is separated from the reference signals. The output can be analysed by varying the adaptive filter order, for instance, N = 2, 4, 8, 16 and 32 in order to minimise errors, Signal-to-Noise Ratio (SNR) and Mean Square Error (MSE) in the shortest response time.

In this approach the ALE-LMS was implemented in MATLAB and the output was obtained from varying the filter order to analyse the signal-to-noise ratio (SNR), Mean square error (MSE) and the overall processing time taken to obtain the output of the system.

Error & Performance Analysis of ALE – LMS					
Filter Order *(N)*	**N=2**	**N=4**	**N=8**	**N=16**	**N=32**
SNR	99.13	42.001	112.33	90.98	101.74
MSE	16.54	11.69	8.2	5.8	4.1
Comp. Time *(s)*	5.2	5.78	5.28	5.92	8.48
Error *e(n)*	-0.00096	0.00075	0.009352	0.043618	0.129049

- **WIENER FILTER THEORY**

In signal processing, the **Wiener filter** is a filter used to produce an estimate of a desired or target random process by linear time-invariant (LTI) filtering of an observed noisy process, assuming known stationary, signal and noise spectra, and additive noise. The goal of the Wiener filter is to compute a statistical estimate of an unknown signal using a related signal as an input and filtering that known signal to produce the estimate as an output. For example, the known signal might consist of an unknown signal of interest that has been corrupted by additive noise. The Wiener filter can be used to filter out the noise from the corrupted signal to provide an estimate of the underlying signal of interest.

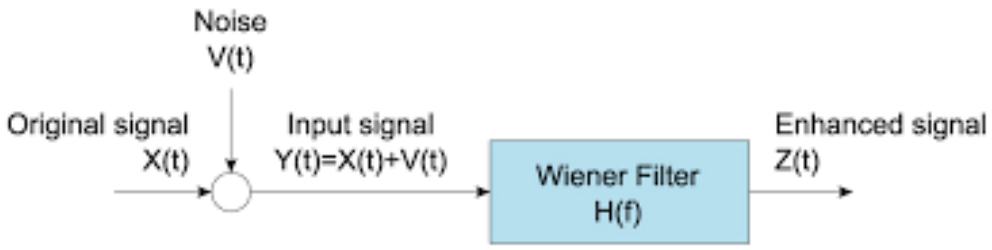


Figure 1. Optimal (Wiener) filtering signal flow

- **ECHO RETURN LOSS ENHANCEMENT**

Estimating Echo Return Loss and Echo Return Loss Enhancement.
Acoustic echo cancellation (AEC) is a signal processing technique that is used to achieve echo-free full-duplex communication in a telecommunications system that has acoustic coupling between the loudspeaker and microphone.

Echo return loss enhancement (ERLE) is measured in dB.
ERLE is the ratio of send-in power and the power of a residual error signal (E) immediately after the cancellation. ERLE measures the amount of loss introduced by the adaptive filter alone.

It is essentially, the ratio of the power of the desired signal over the power of the residual signal.

To obtain an estimate of convergence or the Echo Return Loss Enhancement (ERLE), one must first estimate the coupling factor or the Echo Return Loss (ERL) of the loudspeaker-microphone enclosure. An estimate of the ERL is required to determine how much attenuation can be attributed to the echo path and how much can be attributed to the echo canceller. The coupling factor determines the attenuation or possible gain in the path.

- **LEAST MEAN SQUARE ALGORITHM**

In the LMS algorithm, the coefficients are adjusted from sample to sample in such a way as to minimize the Mean Square Error (MSE).The LMS is based on the steepest descent algorithm where the weight vector is updated from sample to sample as follows

$$W_{k+1} = W_k - \mu_k \qquad (1)$$

Where Wk and µk are the weight and the true gradient vectors, respectively at the kth sampling instant. µ controls the stability and rate of convergence.

1. Initially, set each weight to an arbitrary fixed value as 0. For each subsequent sampling instant, k, carryout steps from 2 to 4 below.

2. Compute filter output

$$n'_k = \Sigma w\,(i)\, x_{k-i} \qquad (2)$$

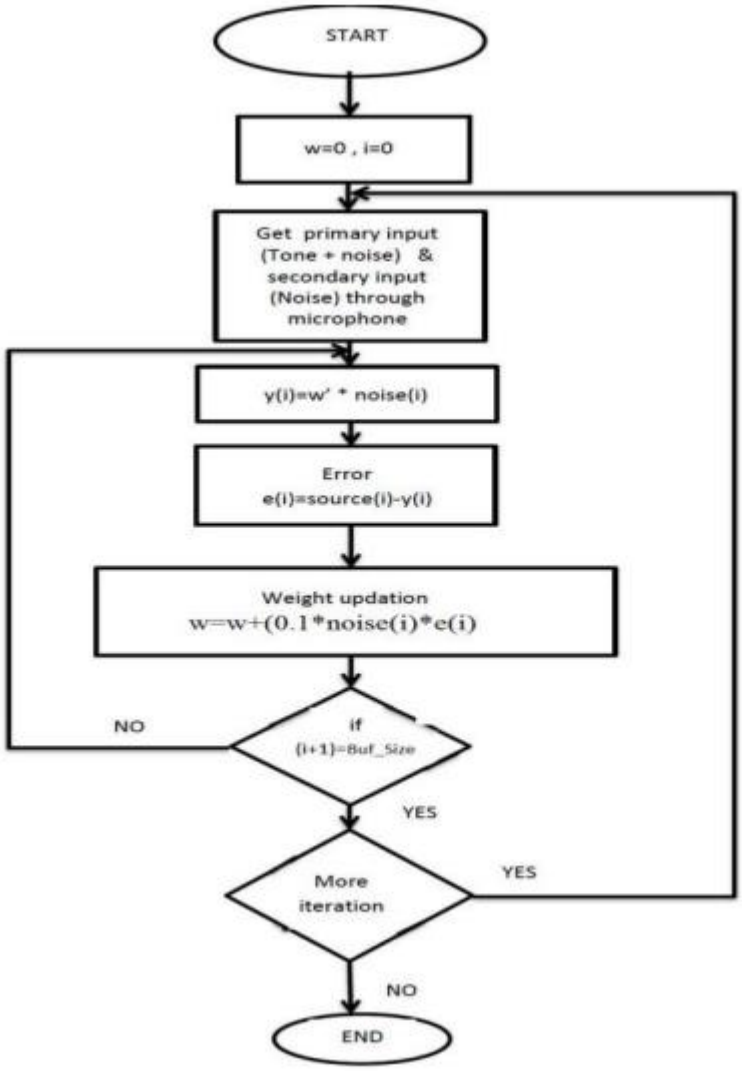
3. Compute the error estimate

$$e_k = y_k - n'_k \qquad (3)$$

4. Update the next filter weights

$$w_{k+1}(i) = w_k(i) + 2\mu e_k x_{k-i} \qquad (4)$$

Because of its simplicity, the LMS algorithm is one of the popular adaptive algorithms. However, the LMS algorithm is very slow and data dependent convergence behaviour. One of the primary disadvantages of the LMS algorithm is having a fixed step size parameter for every iteration. This requires an understanding of the statistics of the input signal prior to commencing the adaptive filtering operation. In practice this is rarely achievable.



Initially, the weight parameter (w) and also the loop variable (i) are set to zero. Then obtained input signals from the microphone. In the next step the filter output is calculated and is further used to compute the error estimate signal. Then the filter weights are updated by fixing the step size value (μ) to be a constant. This procedure is repeated until the loop parameter becomes equal to the buffer size. This implementation is depicted in figure.

- **NORMALIZED LEAST MEAN SQUARE ALGORITHM**

The NLMS algorithm is equally simple, but more robust variant of the LMS algorithm and exhibits a better balance between simplicity and performance compared to LMS algorithm. Due to its better properties the NLMS has been largely used in real-time applications. The normalized least mean square algorithm (NLMS) is an extension of the LMS algorithm which bypasses this issue by computing maximum step size value. Step size value is computed by using the following formula.

Step size=1/dot product (input vector, input vector)

This step size is proportional to the inverse of the total expected energy of the instantaneous values of the coefficients of the input vector. NLMS algorithm is having the advantage over the LMS algorithm in case of Mean square error and Average attenuation

1. The NLMS algorithm is written as follows

$$W_{k+1} = W_k + 2\mu e_k X_k$$

$$\mu_k = \alpha / NP'$$

Where, μ_k : time varying step size factor normalised

N : filter length

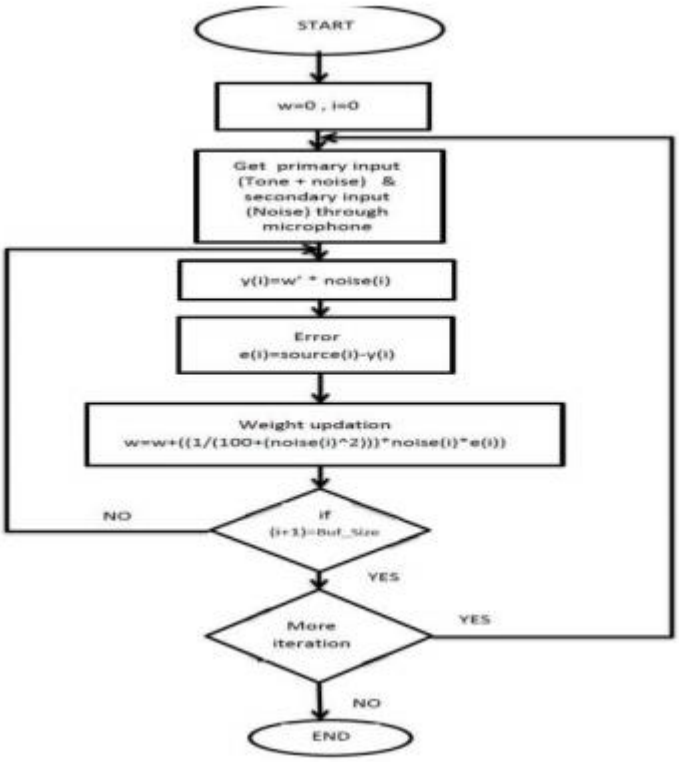
α : constant ($0 < \alpha < 2$)

P'_k: power estimate of x_k

2. Power estimation is upgraded by following equation,

$$P'_{k+1} = (1-\beta)\, P'_k + N\beta x^2_{k-i}$$

Where, β: smoothing parameter



Initially, the weight parameter (w) and the loop variable (i) are set to zero. The obtained input signals from the microphone. In the next step the filter output is calculated and is further used to calculate the error estimate signal. The filter weights are updated depending on the calculated step size value (μ). This procedure is repeated until the loop parameter becomes equal to the buffer size. This implementation is depicted in figure.

COMPARITIVE STUDY OF DIFFERENT ALGORITHMS

- LMS algorithm is very simple in implementation and stable under different conditions and slow convergence.

- NLMS algorithm is of less computational and with good convergence speed makes this algorithm for good echo cancellation. It shows greater stability with unknown input signals. The noise amplification becomes smaller or less size due to the presence of normalized step size. It has minimum study state error and faster convergence.

- RLS algorithm converges faster than LMS in stationary environment but in non stationary LMS algorithm is better than RLS. Sensitivity to computer round off error this leads to instability. Grater computational complexity.

Table I
Performance Comparison of Adaptive Algorithms

S. No.	Algorithms	MSE	Complexity	Stability
1.	LMS	$1.5*10^{-2}$	2N+1	Less Stable
2.	NLMS	$9.0*10^{-3}$	3N+1	Stable
3.	RLS	$6.2*10^{-3}$	$4N^2$	High Stable

LMS Algorithm	RLS Algorithm
Simple and can be easily applied.	Increased complexity and computational cost.
Takes longer to converge.	Faster convergence.
Adaptation is based on the gradient-based approach that updates filter weights to converge to the optimum filter weights.	Adaptation is based on the recursive approach that finds the filter coefficients that minimize a weighted linear least squares cost function relating to the input signals.
Larger steady state error with respect to the unknown system.	Smaller steady state error with respect to unknown system.
Does not account for past data.	Accounts for past data from the beginning to the current data point.
Objective is to minimize the current mean square error between the desired signal and the output.	Objective is to minimize the total weighted squared error between the desired signal and the output.
No memory involved. Older error values play no role in the total error considered.	Has infinite memory. All error data is considered in the total error. Using the forgetting factor, the older data can be de-emphasized compared to the newer data. Since $0 \leq \lambda < 1$, applying the factor is equivalent to weighting the older error.

The correlation that is applied in updating the old estimate of the coefficient vector is based on the instantaneous sample value of the input tap-vector and error signal in the LMS algorithm. In RLS algorithm it utilizes the past available information. The correlation applied to the previous estimate consists of three factors: The step size, error signal, tap input vector. But in RLS algorithm it consists of two factors that is estimation error and gain vector. RLS algorithm is self-orthogonalizing and independent of Eigen value spread of the correlation matrix of the filter input. LMS algorithm requires approximately 20 M iterations to converge in mean square whereas RLS algorithm converge in mean square with in less than 2M iterations. So RLS algorithm converges faster than the LMS algorithm. RLS algorithm exhibits zero mis-adjustment and LMS algorithm exhibits non zero misadjustments.

The LLMS algorithm has proved to be much more tactful, precise and adept to vast real time usage compared to the LMS, NLMS and RLS algorithms as would be evident in the matlab study done as well.

- **COMPUTATIONAL COMPLEXITY COMPARISION OF DIFFERENT ALGORITHM**

For each set of input output samples **LMS** algorithm requires 2M+1 additions and multiplications.

NLMS requires more number of computations for evaluation when compared to LMS algorithm due the presence of the reference signal power which will have leads to more computations in NLMS algorithm. NLMS algorithm requires 3M+1multiplications which is M times more than the LMS algorithm

The computational complexity of **RLS** is proportional (M+1) ^2 the convergence is insensitive to Eigen value spread and misadjustment is zero theoretically.

MATLAB CODES AND RESULTS

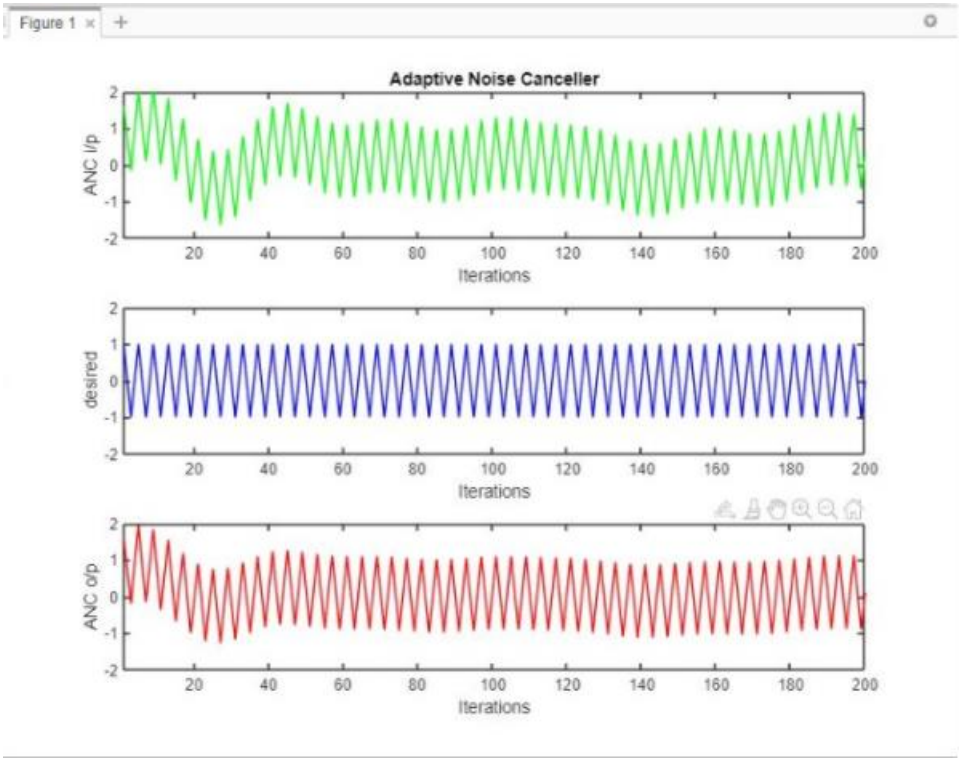
ANC using simple LMS algorithm

Code:

```
clear;
M=16; %order of filter
mu=0.04; %step-size
N=200; %Iterations
f=750;
Ts=1/(4*f); %fs=4 times the freq of the signal
noise=(rand(N,1)-0.5);
n=zeros(M,1);
w=zeros(M,1);
Wn=[0.1 0.5]; %see w/o filter
[B,A]=butter(2,Wn);
x=filter(B,A,n);
for i=1:N
t=(i-1)*Ts;
for k=M:-1:2
n(k)=n(k-1);
end
s(i)=cos(2*pi*f*t);
n(1)=0.2*(cos(2*pi*50*t)+sin(2*pi*100*t)+cos(2*pi*60*t)+sin(2*pi*80*t)+cos(2*p
i*30*t)+ sin(2*pi*20*t)+sin(2*pi*10*t)+ sin(2*pi*90*t)); %noise(i);
d(i)=s(i)+n(1);
x=filter(B,A,n);
d_out(i)=w'*x;
```

```
e(i)=d(i)-d_out(i);
w=w+mu*e(i)*x;
end
i=1:N;
subplot(3,1,1);
plot(i,d,'g');
title('Adaptive Noise Canceller');
xlabel('Iterations');
ylabel('ANC i/p');
axis([1 N -2 2]);
subplot(3,1,2);
plot(i,s,'b');
xlabel('Iterations');
ylabel('desired');
axis([1 N -2 2]);
subplot(3,1,3);
plot(i,e,'r');
xlabel('Iterations');
ylabel('ANC o/p');
axis([1 N -2 2]);
```

Output

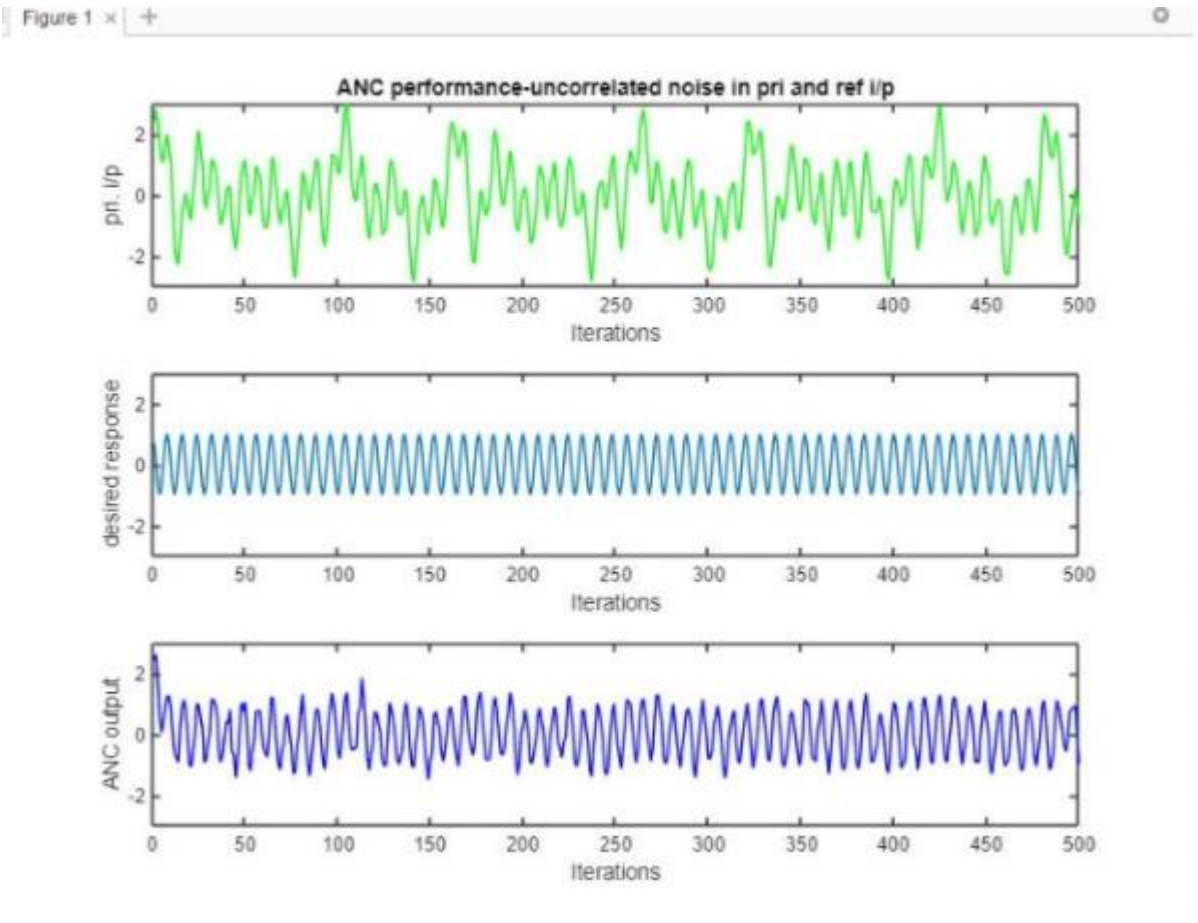


ANC with uncorrelated noise in primary and reference input

Code

```
clc
clear all;
close all;
sampling_time=1/(8*200);
mu=0.04;
M=16;
Iterations=500;
u=zeros(M,1);
x=zeros(M,1);
w=zeros(M,1);
e=zeros(Iterations,1);
Wn=[0.1 0.5];
[B,A]=butter(4,Wn);
pri_n=0.5*(rand(Iterations,1)-0.5);
ref_n=0.5*(rand(Iterations,1)-0.5);
for n=0:Iterations-1
t=n*sampling_time;
for i=M:-1:2
u(i)=u(i-1);
end
u(1)=0.5*(cos(2*pi*50*t)+sin(2*pi*100*t)+cos(2*pi*60*t)+sin(2*pi*80*t)+cos(2*p
i*30*t)+sin(2*pi*20*t)); %rand(1);
d(n+1)=cos(2*pi*200*n*sampling_time)+u(1)+pri_n(n+1);
x=filter(B,A,u)+ref_n(n+1);
d_out=conj(w')*x;
e(n+1)=d(n+1)-d_out;
w=w+mu*x*conj(e(n+1));
end
n=1:Iterations;
subplot(3,1,1);
plot(n,d,'g');
title('ANC performance-uncorrelated noise in pri and ref i/p');
axis([0,500,-3,3]);
xlabel('Iterations');
ylabel('pri. i/p');
subplot(3,1,3);
plot(n,e','b');
axis([0,500,-3,3]);
xlabel('Iterations');
ylabel('ANC output');
subplot(3,1,2);
plot(n,cos(2*pi*200*n*sampling_time));
ylabel('desired response');
xlabel('Iterations');
axis([0,500,-3,3]);
```

Output



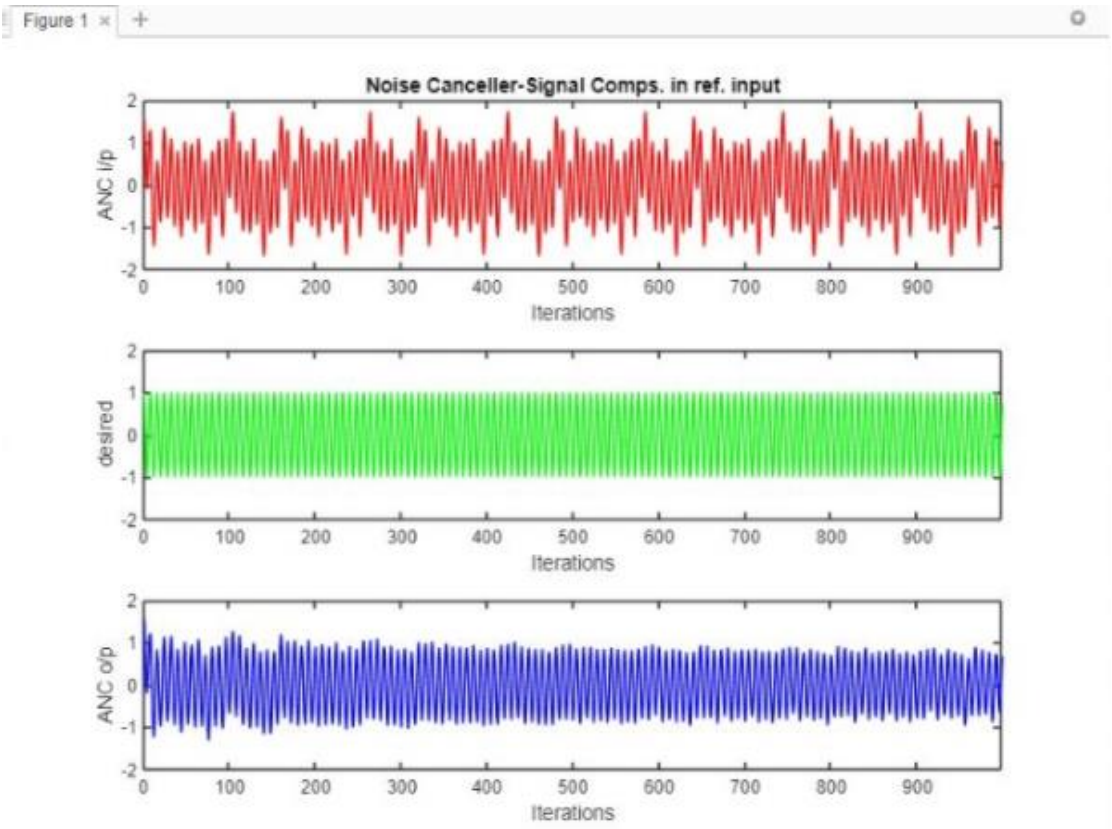
ANC with signal components in reference input

Code

```
clc;
clear all;
close all;
sampling_time=1/(8*200);
mu=0.05;
M=16;
Iterations=1000;
u=zeros(M,1);
x=zeros(M,1);
w=zeros(M,1);
s=zeros(M,1);
```

```
e=zeros(Iterations,1);
Wn=[0.1 0.5];
[Bn,An]=butter(2,Wn);
Ws=0.5;
[Bs,As]=butter(2,Wn);
for n=0:Iterations-1
t=n*sampling_time;
for i=M:-1:2
u(i)=u(i-1);
s(i)=s(i-1);
end
u(1)=
0.2*(cos(2*pi*50*t)+sin(2*pi*100*t)+cos(2*pi*60*t)+sin(2*pi*80*t)+cos(2*pi*30*
t)+sin(2*pi*20*t)); %rand(1)-0.5;
s(1)=cos(2*pi*200*n*sampling_time);
sig(n+1)=s(1);
noi(n+1)=u(1);
d(n+1)=s(1)+u(1);
x=filter(Bn,An,u)+0.04*filter(Bs,As,s);
d_out=conj(w')*x;
e(n+1)=d(n+1)-d_out;
w=w+mu*x*conj(e(n+1));
end
n=0:Iterations-1;
subplot(3,1,1);
plot(n,d,'r');
title('Noise Canceller-Signal Comps. in ref. input');
axis([0 Iterations-1 -2 2]);
xlabel('Iterations');
ylabel('ANC i/p');
subplot(3,1,2);
plot(n,cos(2*pi*200*n*sampling_time),'g');
axis([0 Iterations-1 -2 2]);
xlabel('Iterations');
ylabel('desired');
subplot(3,1,3);
plot(n,e','b');
axis([0 Iterations-1 -2 2]);
xlabel('Iterations');
ylabel('ANC o/p');
ZOOM XON;
```

Output



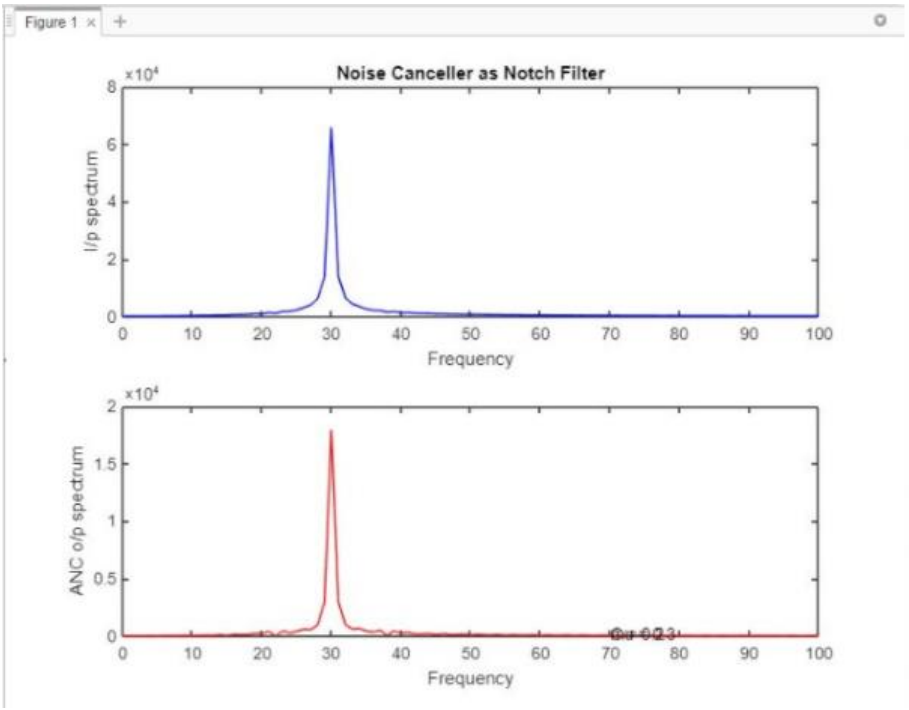
ANC as notch

Code

```
clear;
mu=0.3;
M=2;
Iterations=512;
C=0.2;
w0=2*pi*30;
phi=pi/4;
phi1=phi;
phi2=phi+pi/2;
w=zeros(M,1);
e=zeros(Iterations,1);
fs=(20:1:40)';
ws=2*pi*fs;
A=rand(size(ws));
theta=2*pi*rand(size(ws));
```

```
Ts=1/(8*max(fs));
for n=1:Iterations
t=(n-1)*Ts;
s(1:size(ws),n)=(cos(ws*t));
signal(n)=sum(s(1:size(ws),n));
pri_noise(n)=n*cos(w0*t+2*pi*n);
d(n)=signal(n)+pri_noise(n);
x(1,1)=C*cos(w0*t+phi1);
x(2,1)=C*cos(w0*t+phi2);
y=conj(w')*x; e(n)=d(n)-y;
w=w+2*mu*conj(e(n))*x;
end
f=0:100;
Se=zeros;
Sd=zeros;
Ssignal=zeros;
for n=1:Iterations
Se=Se+e(n)*exp(-sqrt(-1)*2*pi*f*(n-1)*Ts);
Sd=Sd+d(n)*exp(-sqrt(-1)*2*pi*f*(n-1)*Ts);
Ssignal=Ssignal+signal(n)*exp(-sqrt(-1)*2*pi*f*(n-1)*Ts);
end;
subplot(2,1,1);
xlabel('Iterations');
plot(f,abs(Sd),'b');
title('Noise Canceller as Notch Filter');
ylabel('I/p spectrum');
xlabel('Frequency');
subplot(2,1,2);
plot(f,abs(Se),'r');
ylabel('ANC o/p spectrum');
xlabel('Frequency');
text(70,250,'mu = 0.3');
text(70,210,'C = 0.2');
```

Output

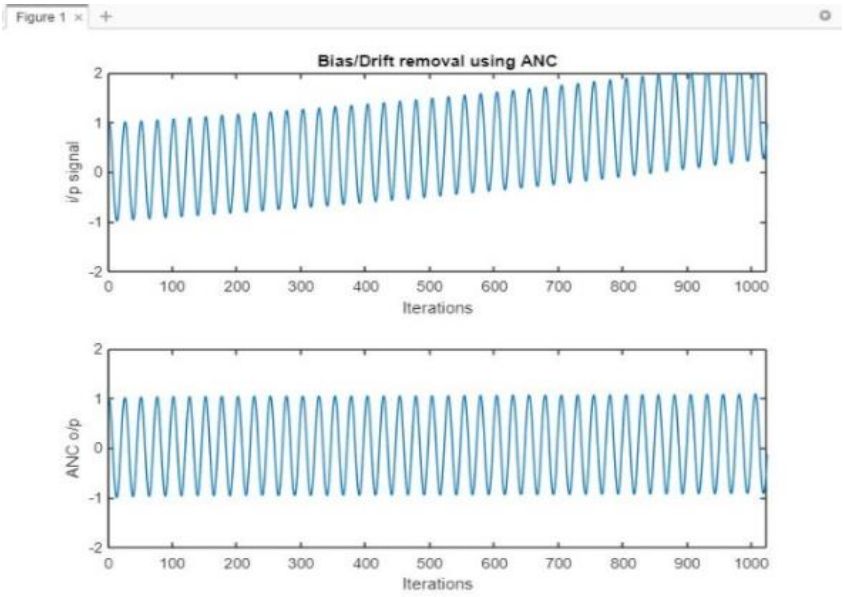


ANC for drift/bias removal

Code

```
clear;
N=1024;
mu=0.01; %change mu=0.001 and see the result-mmse vs. speed of adaptation
ws=200;
Ts=1/(4*ws);
%drift=1.2; %constant bias;
w(1)=0;
for n=1:N
t=(n-1)*Ts;
s(n)=cos(ws*t);
drift(n)=(-1+exp(0.0008*n));
d(n)=s(n)+drift(n);
drift_=w(n);
s_(n)=d(n)-drift_;
e(n)=s_(n);
w(n+1)=w(n)+2*mu*e(n);
end
subplot(2,1,1);
plot(d);
title('Bias/Drift removal using ANC');
ylabel('i/p signal');
xlabel('Iterations');
axis([0 N -2 2]);
subplot(2,1,2);
plot(s_);
ylabel('ANC o/p');
xlabel('Iterations');
axis([0 N -2 2]);
```

Output



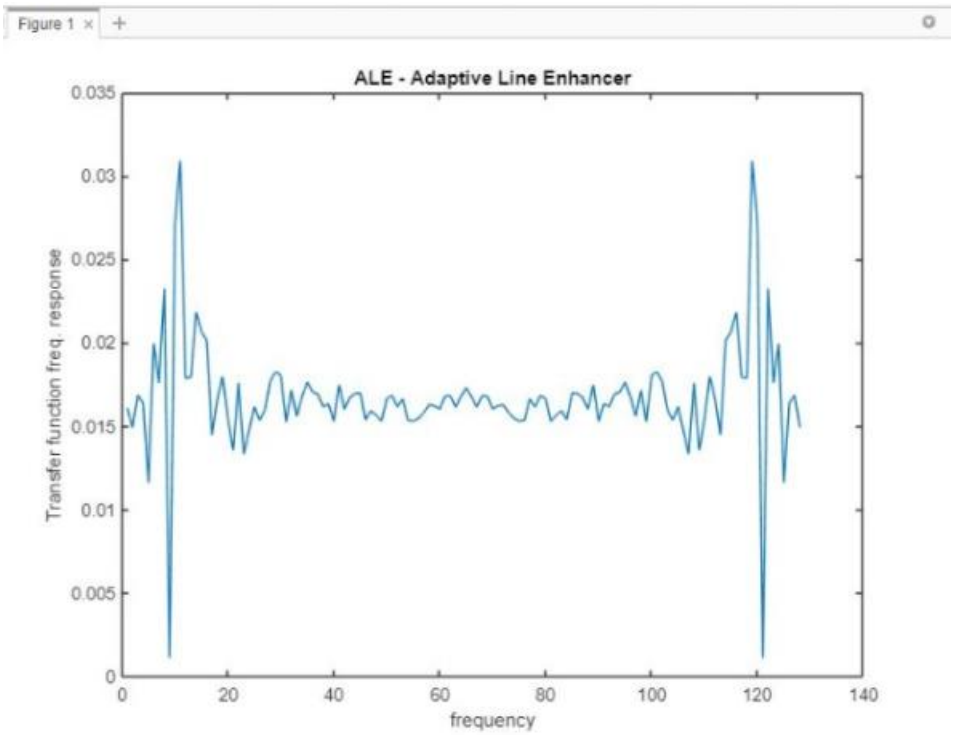
ANC as adaptive line enhancer

Code

```
clear;
f=200;
Ts=1/(16*f);
L=1024;
for l=1:L
t=(l-1)*Ts;
s(l,1)=0.1*cos(2*pi*f*t);
end
N=8;
for l=1:L
n(l,1:N)=normrnd(0,0.8,1,N);
end
mu=0.01;
M=128;
for i=1:N
w=zeros(M,N);
d=s+n(1:L,i);
ref=zeros(M,1);
for l=1:L
for k=M:-1:2
ref(k)=ref(k-1);
end
ref(1)=n(l,i);
d_out(l)=w(1:M,i)'*ref;
e(l)=d(l)-d_out(l);
```

```
w(1:M,i)=w(1:M,i)+2*mu*ref*conj(e(l));
end
end
w_avg=zeros(M,1);
for i=1:N
w_avg=w_avg+w(1:M,i);
end
w_ens=w_avg/N;
DFT=dftmtx(length(w_ens))*w_ens;
Power_spec=DFT.*conj(DFT);
plot(Power_spec);
title('ALE - Adaptive Line Enhancer');
xlabel('frequency');
ylabel('Transfer function freq. response');
```

Output



ANC using LMS with real time example

Code

```
clear all;
close all;
clc;
tic
% Storing data of the test signal in double-precision array
[desired,Fs] = audioread('near_end.wav');% Loading of test signal
desired = desired / rms(desired, 1);%Normalization of the signal
 m = length(desired);
t=(1:m)';


%Reference signal u(k)
%refer = wgn(m,1,-10); % Addition of noise directly in MATLAB
%refer = wgn(m,1,-5);
%refer = wgn(m,1,0);
%refer = wgn(m,1,5);
refer = wgn(m,1,10);


fil = fir1(11, 0.4);% Designing a FIR filter
u = filter(fil, 1, refer);%Filtering the reference signal


%Primary signal s(k)+n(k)
primary = desired+ u;


%Calculation of LMS filter weights
order = 11;
mu = 0.003642;
n = length(primary);
w = zeros(order,1);
E = zeros(1,m);
for k = 11:n
U = u(k-10:k);
y = U'*w;
E(k) = primary(k)-y;
w = w + mu*E(k)*U;
end


%Plotting of the signals
T=(1:m)';
figure(1);
subplot(4,1,1);
plot(t,desired);
title('Desired Signal in time domain');
xlabel('Time');
```

```
ylabel('desired signal');


subplot(4,1,2);
plot(t,u);
title('Reference Signal in time domain');
xlabel('Time');
ylabel('Reference signal');


subplot(4,1,3);
plot(t,primary);
xlabel('Time');
ylabel('primary signal');
title('Primary Signal in time domain');


subplot(4,1,4);
plot(T,E);
xlabel('Time');
ylabel('Denoised Signal');
title('Denoised signal in time domain');


figure(2);
freqz(desired);
title('Desired Signal in frequency domain');


figure(3);
freqz(u);
title('Reference Signal in frequency domain');


figure(4);
freqz(primary);
title('Primary Signal in frequency domain');


figure(5);
freqz(E);
title('Denoised signal in frequency domain');
%Signal to noise ratio(SNR) and varience of desired signal and denoised
signal(E)
disp(snr(desired));%Signal to noise ratio of the desired signal
disp(var(desired));%Varience of the desired signal
disp(snr(E));%Signal to noise ratio of the output signal E
disp(var(E));%Varience of the output signal E
toc;
```

Output

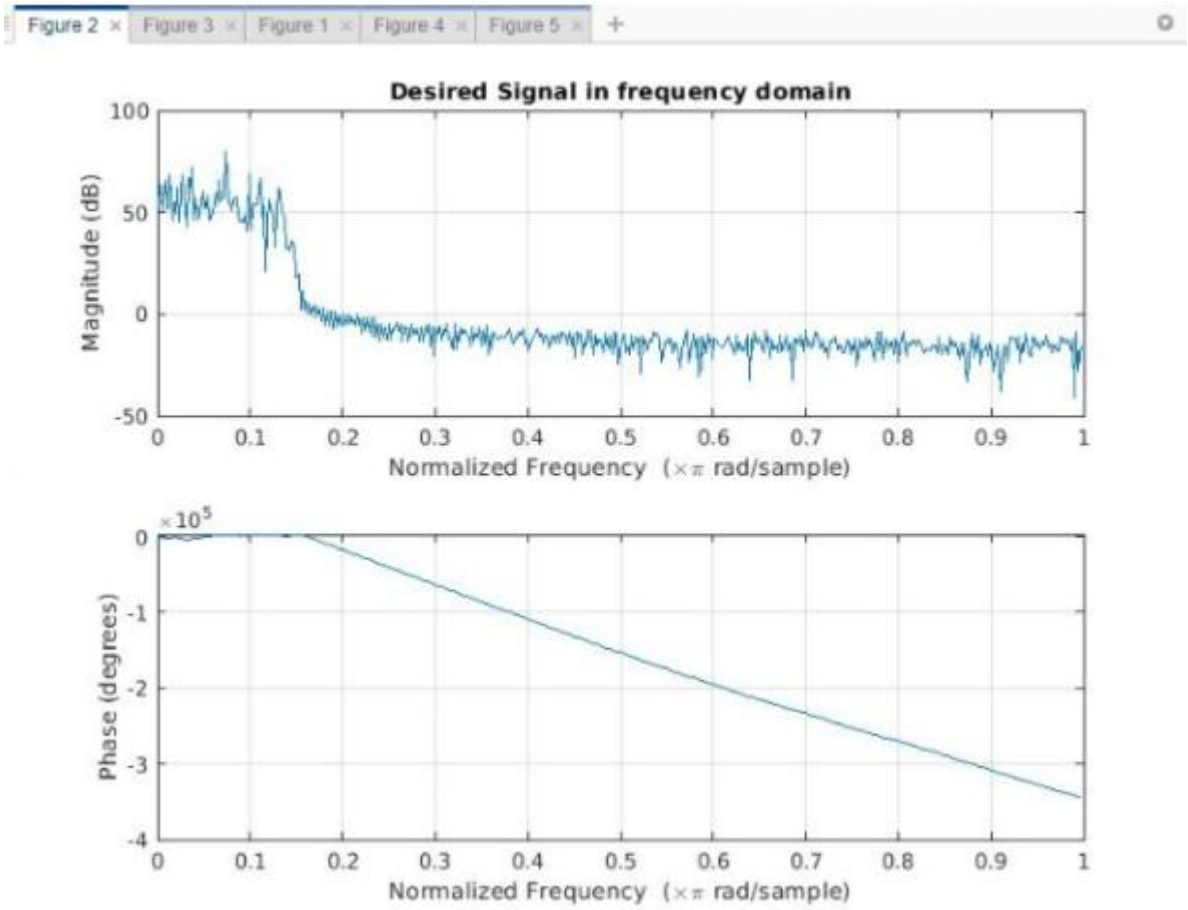


Figure 2
Figure 3
Figure 1
Figure 4
Figure 5
Desired Signal in frequency domain
Magnitude (dB)
100
50
0
-50
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Normalized Frequency (×π rad/sample)
×10^5
Phase (degrees)
0
-1
-2
-3
-4
Normalized Frequency (×π rad/sample)

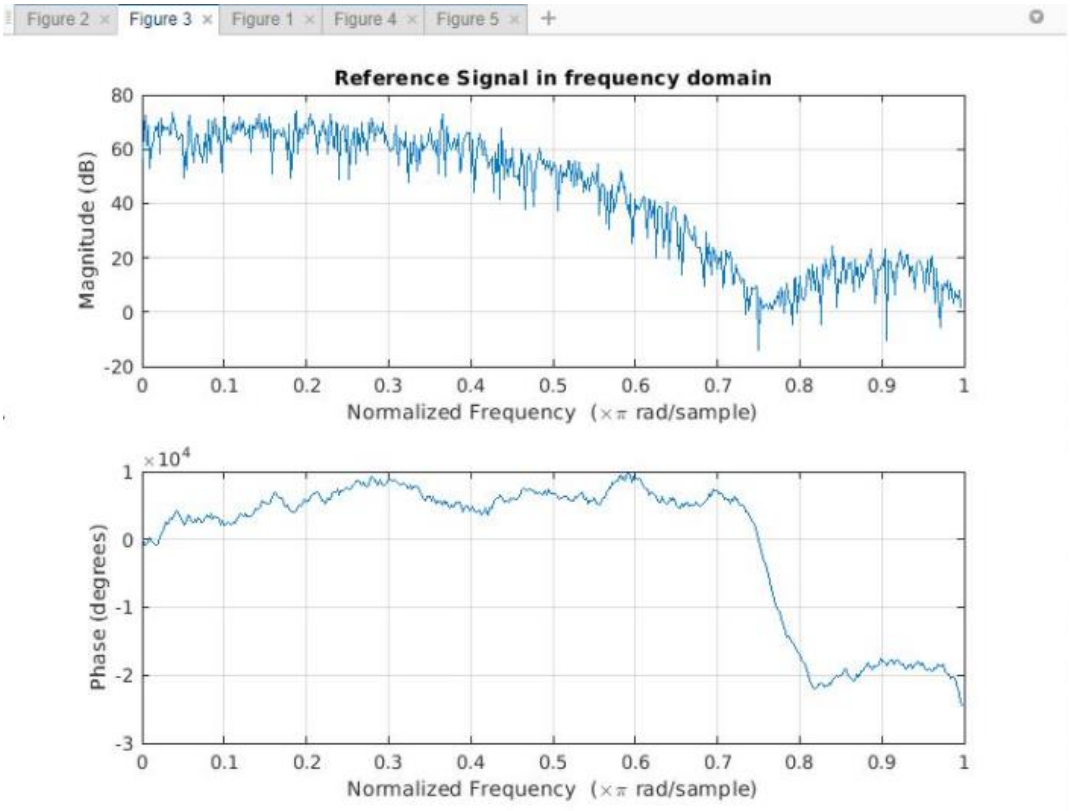


Figure 2
Figure 3
Figure 1
Figure 4
Figure 5
Reference Signal in frequency domain
Magnitude (dB)
Normalized Frequency (×π rad/sample)
×10^4
Phase (degrees)
Normalized Frequency (×π rad/sample)

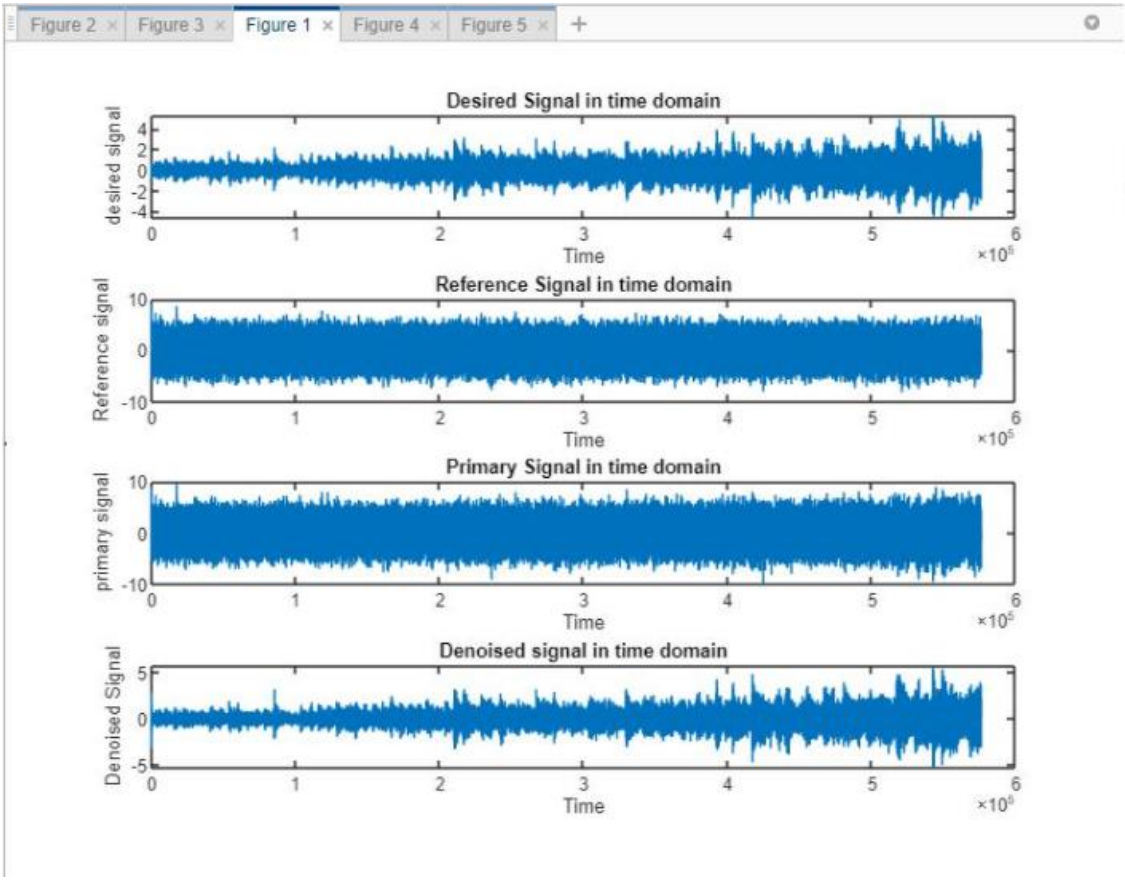


Figure 2
Figure 3
Figure 1
Figure 4
Figure 5
Desired Signal in time domain
desired signal
Time
Reference Signal in time domain
Reference signal
Time
Primary Signal in time domain
primary signal
Time
Denoised signal in time domain
Denoised Signal
Time

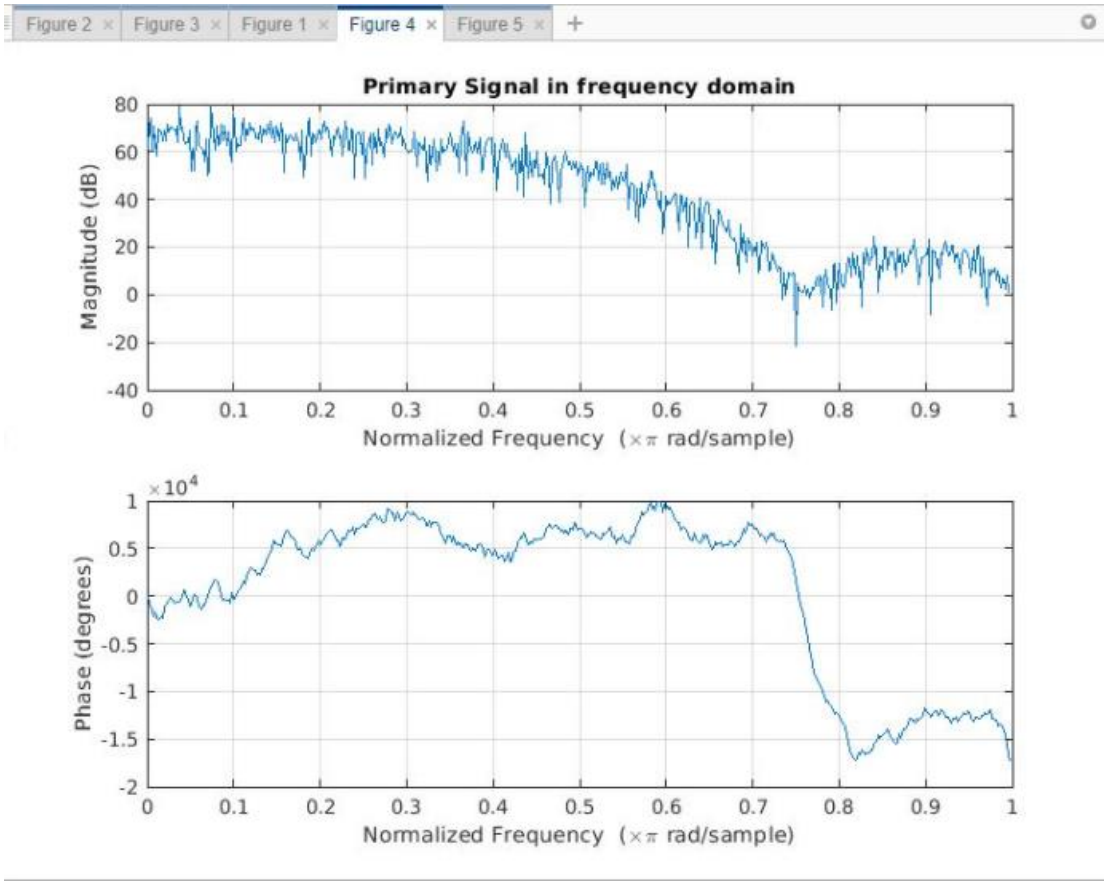


Figure 2
Figure 3
Figure 1
Figure 4
Figure 5
Primary Signal in frequency domain
Magnitude (dB)
Normalized Frequency (×π rad/sample)
×10^4
Phase (degrees)
Normalized Frequency (×π rad/sample)

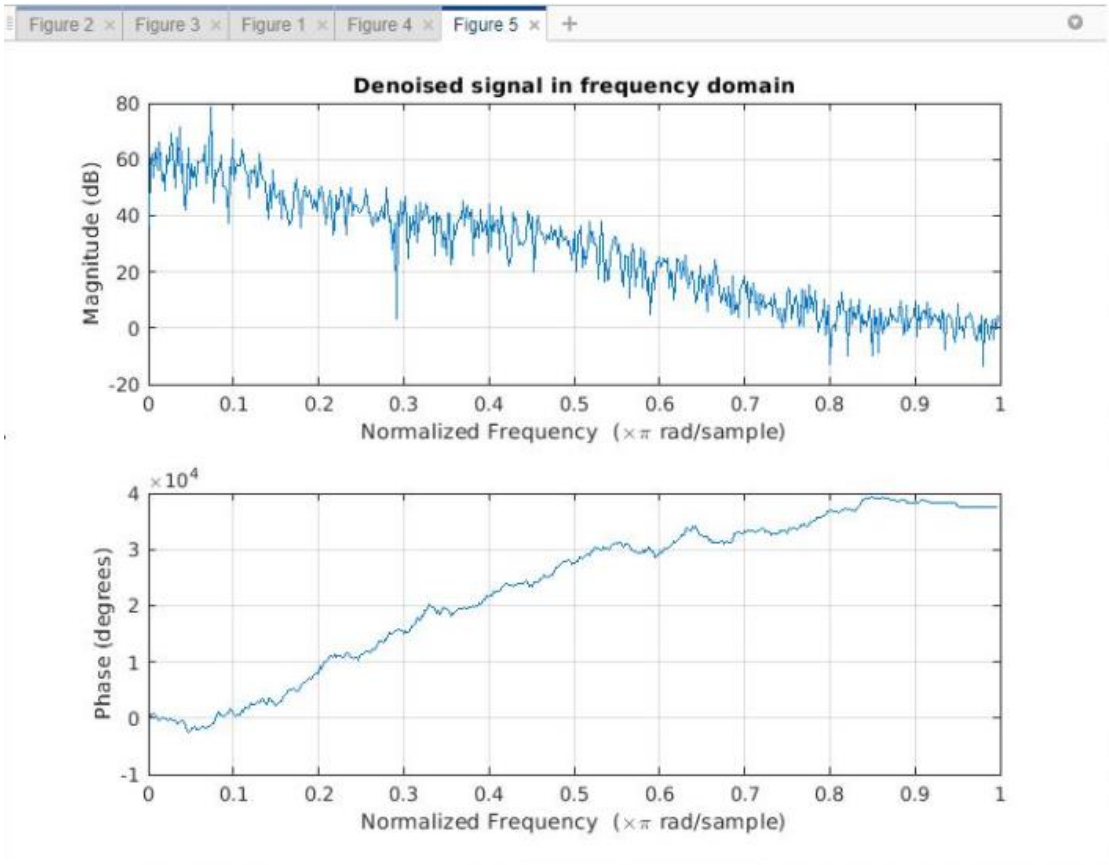


Figure 2
Figure 3
Figure 1
Figure 4
Figure 5
Denoised signal in frequency domain
Magnitude (dB)
Normalized Frequency (×π rad/sample)
×10^4
Phase (degrees)
Normalized Frequency (×π rad/sample)

ANC using NLMS with real time example

Code

```
 clc
clear all;
close all;
input = importdata('sample_data.mat');
fs = input.fs;
primary = input.reference;
reference = input.primary;
primary_size = size(primary,2);
Epsilon = 0.0001;
AllData = zeros(1,3);


for order = 30
W_1 = zeros(order,1);
W_2 = zeros(order,1);
performance_curve1 = zeros(46500,1);
performance_curve2 = zeros(18461,1);
primary_wrt_filter = primary(1 , order:end); %truncate primary
reference_wrt_filter = zeros((primary_size - order),order);
for update = (order) : primary_size %make reference_wrt_filter according to
filter
for update1=1:order
reference_wrt_filter((update-order+1),update1) = reference(update-update1+1);
end
end
disp(size(reference_wrt_filter,1));

%    for Nu = 0.001:(2 * 0.01):1
       for Nu = 0.05
%        for iterateReference = 1: size(reference_wrt_filter,1)
            for iterateReference = 1: size(reference_wrt_filter,1)
            MSE =0;
            Error = primary_wrt_filter(1, iterateReference) -
            (reference_wrt_filter(iterateReference,:) * W_2(:,1));
            X = reference_wrt_filter(iterateReference,:);
            Nu_by_Epsilon = Nu / (Epsilon + (X * X'));
            if iterateReference < 46501
                  Error = primary_wrt_filter(1, iterateReference) -
                  (reference_wrt_filter(iterateReference,:) * W_1(:,1));
                  W_1 = W_1 + (Nu_by_Epsilon * (Error * X)');

                  errorSquare = (primary_wrt_filter(1, 1:iterateReference)' -
                  (reference_wrt_filter(1:iterateReference, :) *
                  W_1(:,1))).^2;


                  MSE = sum(errorSquare)/(iterateReference);
                  performance_curve1(iterateReference,1) = MSE;

            end
```

```
            W_2 = W_2 + (Nu_by_Epsilon * (Error * X)');

            if iterateReference >= 46501
                errorSquare1 = (primary_wrt_filter(1,
                46501:iterateReference)' -
                (reference_wrt_filter(46501:iterateReference, :) *
                W_2(:,1))).^2;


                MSE = sum(errorSquare1)/(iterateReference-46500);
                performance_curve2(iterateReference-46500,1) = MSE;
            end

            end
%         MSE = sum(primary_wrt_filter - (reference_wrt_filter * W_1)')^2;
%         AllData(size(AllData,1)+1,1) = order;
%         AllData(size(AllData,1),2) = Nu;
%         AllData(size(AllData,1),3) = MSE;
       end

Out = (primary_wrt_filter(1, 1:46500) - (reference_wrt_filter(1:46500,:) *
W_1)');
Out1 = (primary_wrt_filter(1, 46501:end) - (reference_wrt_filter(46501:end,:)
* W_2)');
Out3 = vertcat(Out', Out1');


SNR_parameter = mean(primary_wrt_filter.^2)/mean(Out3.^2);
SNR_After = 10 * log10(SNR_parameter);
figure;
plot(performance_curve1);
title('Learning Curve For Filter Order = 50 and Iteration < 46.5K');
xlabel('Iteration -->');
ylabel('MSE -->');
legend('Nu = 0.05');


figure;
plot(performance_curve2);
title('Learning Curve For Filter Order = 50 and Iteration > 46.5K');
xlabel('Iteration -->');
ylabel('MSE -->');
legend('Nu = 0.05');
%
figure;
plot(Out3);
title('Error Signal After Applying NLMS For Filter Order = 50');
xlabel('Iteration -->');
ylabel('Error (Desired Output Signal) -->');
legend('Nu = 0.05');
%soundsc(Out3,fs);
end
```

Output

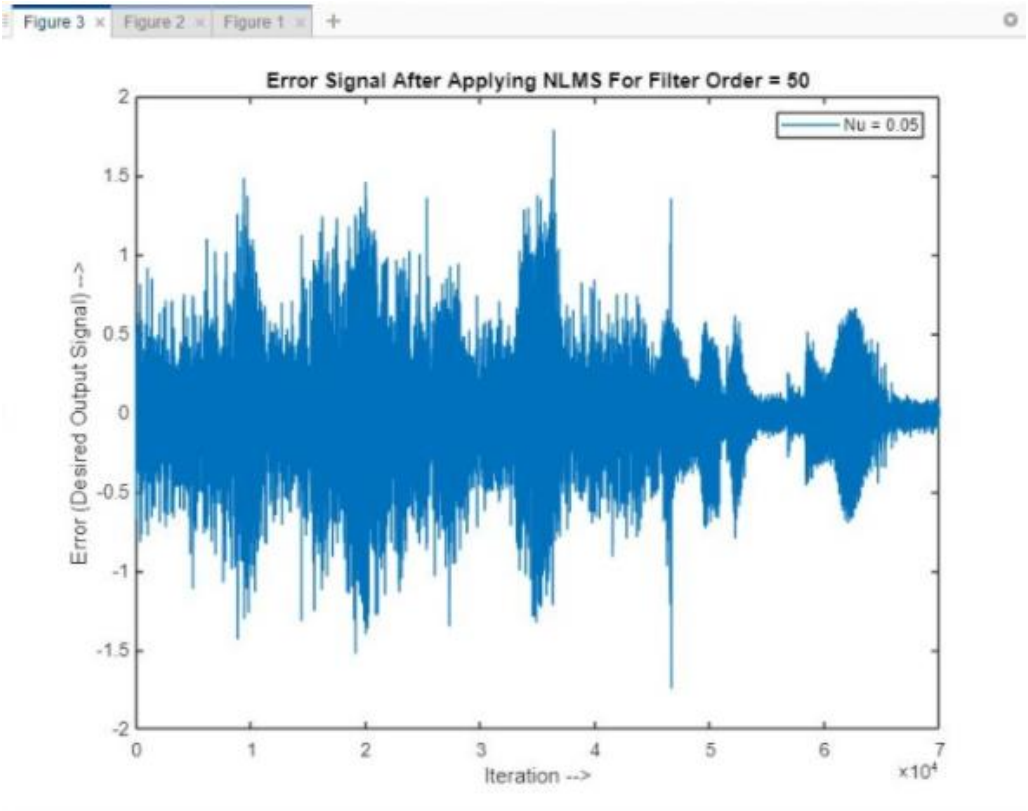


Figure 3
Figure 2
Figure 1
Error Signal After Applying NLMS For Filter Order = 50
Nu = 0.05
Error (Desired Output Signal) -->
Iteration -->
×10^4

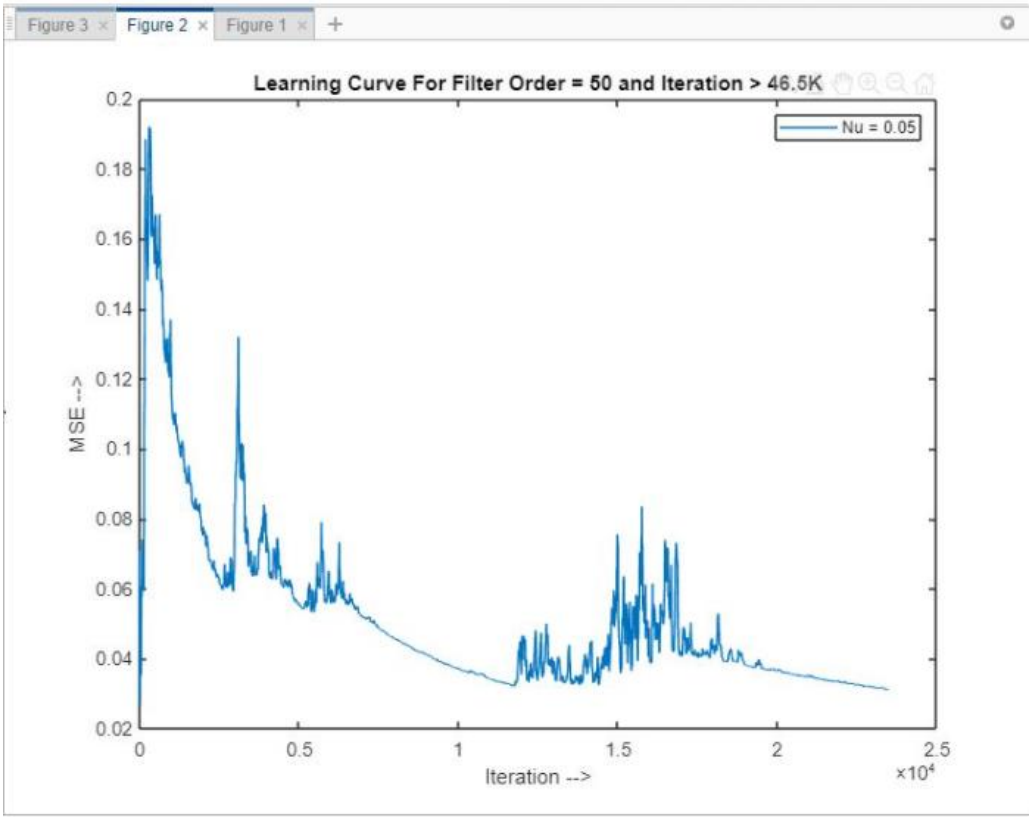
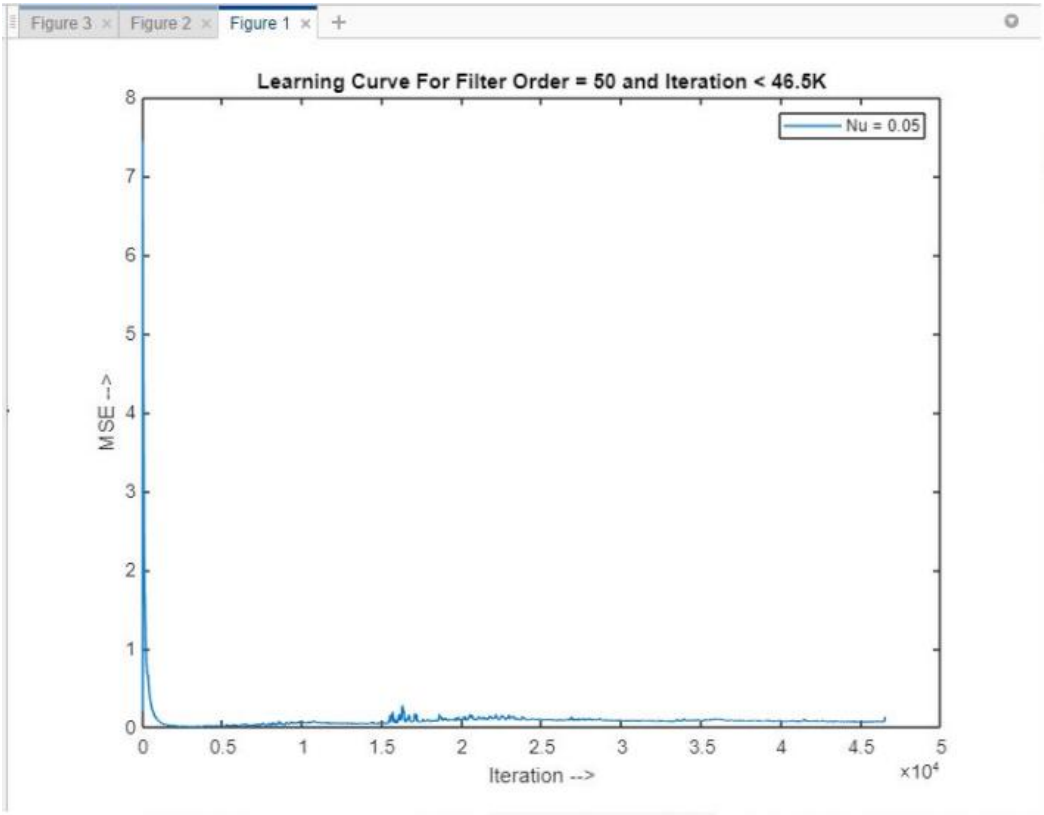


Figure 3
Figure 2
Figure 1
Learning Curve For Filter Order = 50 and Iteration > 46.5K
Nu = 0.05
MSE -->
Iteration -->
×10^4



Project with comparative study of three major algorithms

Code

```
clc;
clear all;
close all;


load('desnoi');
x2=x2(:);
s2=s2(:);
a = input('enter a value'); %initializing the values of a and c for step size
c = input('enter c value');
L = 512;
N=length(x2);
xin=zeros(L,1);
w=zeros(L,1);


%initialization of variables
y = 0*x2;
e = 0*x2;
powerS = 0*x2;
powerE = 0*x2;
%NLMS algorithm
```

```
for i=1:N
xin = [x2(i); xin(1:end-1)];
y(i)=w'*xin;
error=s2(i)-y(i); %ERROR
e(i)=error; %Store estimation error
mu=a/(c+xin'*xin); %Calculate Step-size
wtemp = w + 2*mu*error*xin; %Update filter
w = wtemp;
powerS(i) = abs(s2(i))^2; %Power of Microphone signal
powerE(i)=abs(e(i))^2; %power of Error signal
end

%LMS algorithm
for i=1:N
xin = [x2(i); xin(1:end-1)];
y1(i)=w'*xin;
error1=s2(i)-y1(i); %ERROR
e1(i)=error1; %Store estimation error
mu=0.01; %Calculate Step-size
wtemp = w + 2*mu*error1*xin; %Update filter
w = wtemp;
powerS1(i) = abs(s2(i))^2; %Power of Microphone signal
powerE1(i)=abs(e1(i))^2; %power of Error signal
end

%VL-LMS algorithm
for i=1:N
xin = [x2(i); xin(1:end-1)];
y2(i)=w'*xin;
error2=s2(i)-y2(i); %ERROR
e2(i)=error2; %Store estimation error
mu=0.01; %Calculate Step-size
gamma=0.05;
wtemp = (1-2*mu*gamma)*w + 2*mu*error2*xin; %Update filter
w = wtemp;
powerS2(i) = abs(s2(i))^2; %Power of Microphone signal
powerE2(i)=abs(e2(i))^2; %power of Error signal
end


for i=1:N-L
%Echo Return Loss Enhancement
ERLE(i)=10*log10(mean(powerS(i:i+L))/mean(powerE(i:i+L))); %Calculating the
ERLE
ERLE1(i)=10*log10(mean(powerS1(i:i+L))/mean(powerE1(i:i+L))); %Calculating the
ERLE
ERLE2(i)=10*log10(mean(powerS2(i:i+L))/mean(powerE2(i:i+L))); %Calculating the
ERLE
end

%-------Echo signal--------
figure
subplot(4,1,1)


plot(s2) %plotting the echo signal
```

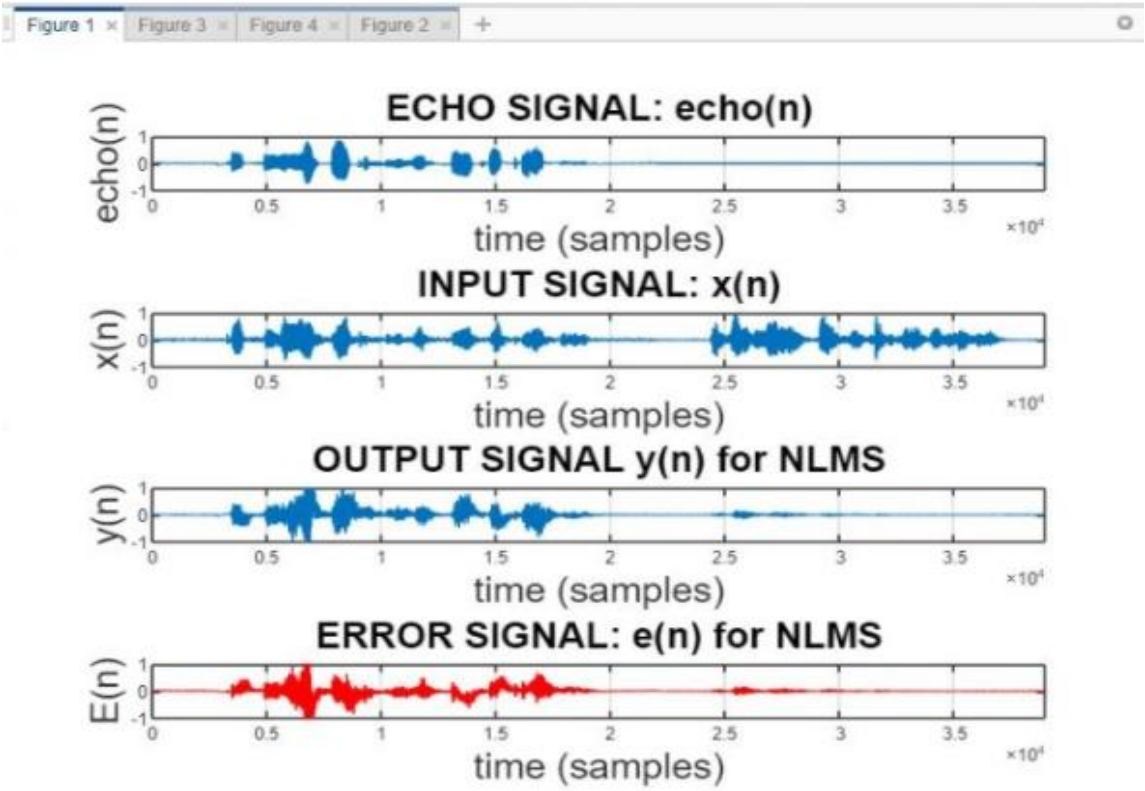
```
xlabel('time (samples)','FontSize', 18);
ylabel('echo(n)','FontSize', 18);
title('ECHO SIGNAL: echo(n)','FontSize', 18)
grid on
axis([0 N -1 1]);


%-------Input signal-------------------
subplot(4,1,2)
% figure*
plot(x2) %plotting the input signal
xlabel('time (samples)','FontSize', 18);
ylabel('x(n)','FontSize', 18);
title('INPUT SIGNAL: x(n)','FontSize', 18)
grid on
axis([0 N -1 1]);


%-------Output signal x(n)----------------
subplot(4,1,3)
% figure
plot(y) %plotting output signal
xlabel('time (samples)','FontSize', 18);
ylabel('y(n)','FontSize', 18);
title('OUTPUT SIGNAL y(n) for NLMS','FontSize', 18)
grid on
axis([0 N -1 1]);


%-------Error signal x(n)-----------------
subplot(4,1,4)
% figure
plot(e,'red') %plotting the error signal
xlabel('time (samples)','FontSize', 18);
ylabel('E(n)','FontSize', 18);
title('ERROR SIGNAL: e(n) for NLMS','FontSize', 18)
axis([0 N -1 1]);
grid on


%ERLE calculation
% ylabel('Desired signal/Error signal (dB)','FontSize', 20);
figure
subplot(311)
plot(ERLE)
hold on; %plotting the ERLE w.r.t desired signal
plot(s2,'r');
xlabel('Sample number (n)','FontSize', 20);
% ylabel('Desired signal/Error signal (dB)','FontSize', 20);
legend('ERLE', 'DESIRED SIGNAL')
title('ECHO RETURN LOSS ENHANCEMENT for NLMS','FontSize', 20)
axis([0 N -20 40]);
grid on


subplot(312)
```

```
plot(ERLE1)
hold on; %plotting the ERLE w.r.t desired signal
plot(s2,'r');
xlabel('Sample number (n)','FontSize', 20);
% ylabel('Desired signal/Error signal (dB)','FontSize', 20);
legend('ERLE', 'DESIRED SIGNAL')
title('ECHO RETURN LOSS ENHANCEMENT for LMS','FontSize', 20)
axis([0 N -20 40]);
grid on


subplot(313)
plot(ERLE2)
hold on; %plotting the ERLE w.r.t desired signal
plot(s2,'r');
xlabel('Sample number (n)','FontSize', 20);
% ylabel('Desired signal/Error signal (dB)','FontSize', 20);
legend('ERLE', 'DESIRED SIGNAL')
title('ECHO RETURN LOSS ENHANCEMENT for LLMS','FontSize', 20)
axis([0 N -20 40]);
grid on
%LMS algorithm


figure
subplot(4,1,1)


plot(s2) %plotting the echo signal
xlabel('time (samples)','FontSize', 18);
ylabel('echo(n)','FontSize', 18);
title('ECHO SIGNAL: echo(n)','FontSize', 18)
grid on
axis([0 N -1 1]);


%-------Input signal-------------------
subplot(4,1,2)
% figure*
plot(x2) %plotting the input signal
xlabel('time (samples)','FontSize', 18);
ylabel('x(n)','FontSize', 18);
title('INPUT SIGNAL: x(n)','FontSize', 18)
grid on
axis([0 N -1 1]);


% figure
subplot(4,1,3)
plot(y1)                                                    %plotting output
signal
xlabel('time (samples)','FontSize', 18);
ylabel('y(n)','FontSize', 18);
title('OUTPUT SIGNAL y(n) for LMS','FontSize', 18)
grid on
axis([0 N -1 1]);
```

```
%-------Error signal x(n)-----------------
subplot(4,1,4)
% figure
plot(e1,'red')                                          %plotting the error
signal
xlabel('time (samples)','FontSize', 18);
ylabel('E(n)','FontSize', 18);
title('ERROR SIGNAL: e(n) for LMS','FontSize', 18)
axis([0 N -1 1]);
grid on
% LLMS agorithm


figure
subplot(4,1,1)


plot(s2)                                            %plotting the echo signal
xlabel('time (samples)','FontSize', 18);
ylabel('echo(n)','FontSize', 18);
title('ECHO SIGNAL: echo(n)','FontSize', 18)
grid on
axis([0 N -1 1]);


%-------Input signal------------------
subplot(4,1,2)
% figure*
plot(x2)                                                %plotting the input
signal
xlabel('time (samples)','FontSize', 18);
ylabel('x(n)','FontSize', 18);
title('INPUT SIGNAL: x(n)','FontSize', 18)
grid on
axis([0 N -1 1]);


% figure
subplot(413)
plot(y2)                                                %plotting output
signal
xlabel('time (samples)','FontSize', 18);
ylabel('y(n)','FontSize', 18);
title('OUTPUT SIGNAL y(n) for LLMS','FontSize', 18)
grid on
axis([0 N -1 1]);


%-------Error signal x(n)-----------------
subplot(4,1,4)
% figure
plot(e2,'red')                                          %plotting the error
signal
xlabel('time (samples)','FontSize', 18);
```

```
ylabel('E(n)','FontSize', 18);
title('ERROR SIGNAL: e(n) for LLMS','FontSize', 18)
axis([0 N -1 1]);
grid on
```

Output



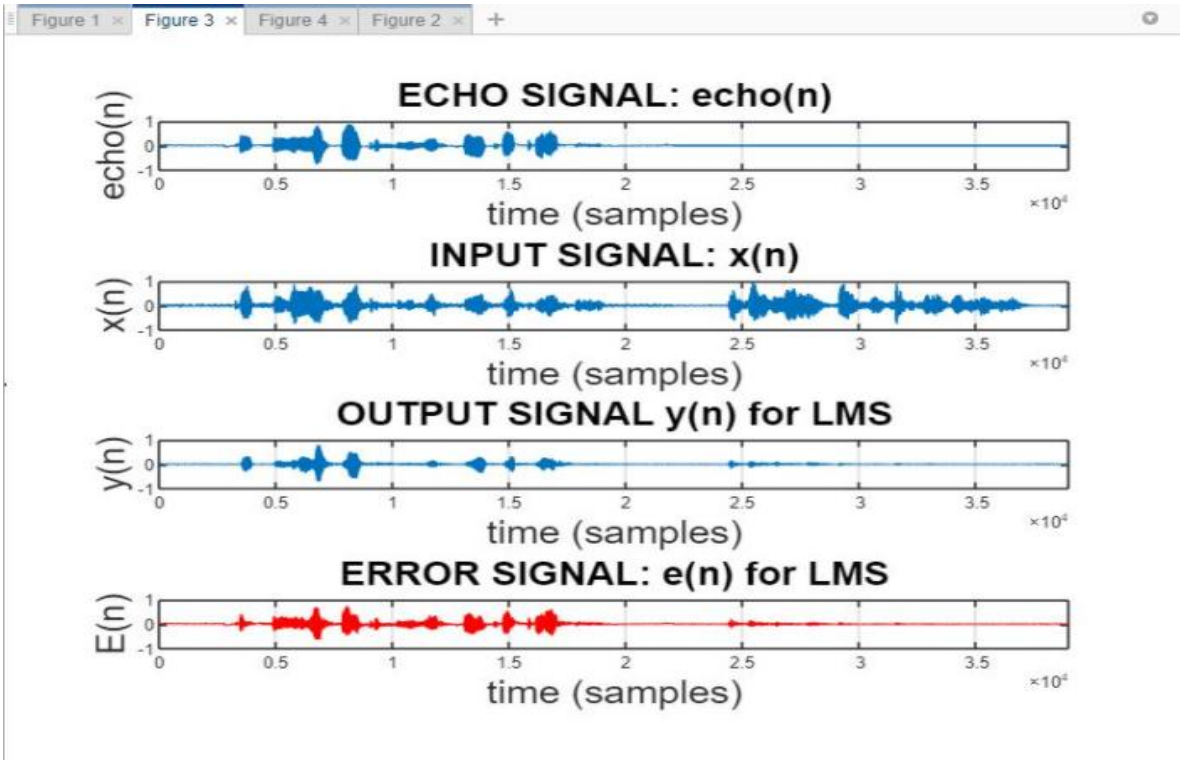


Figure 1
Figure 3
Figure 4
Figure 2
ECHO SIGNAL: echo(n)
echo(n)
time (samples)
INPUT SIGNAL: x(n)
x(n)
time (samples)
OUTPUT SIGNAL y(n) for LMS
y(n)
time (samples)
ERROR SIGNAL: e(n) for LMS
E(n)
time (samples)

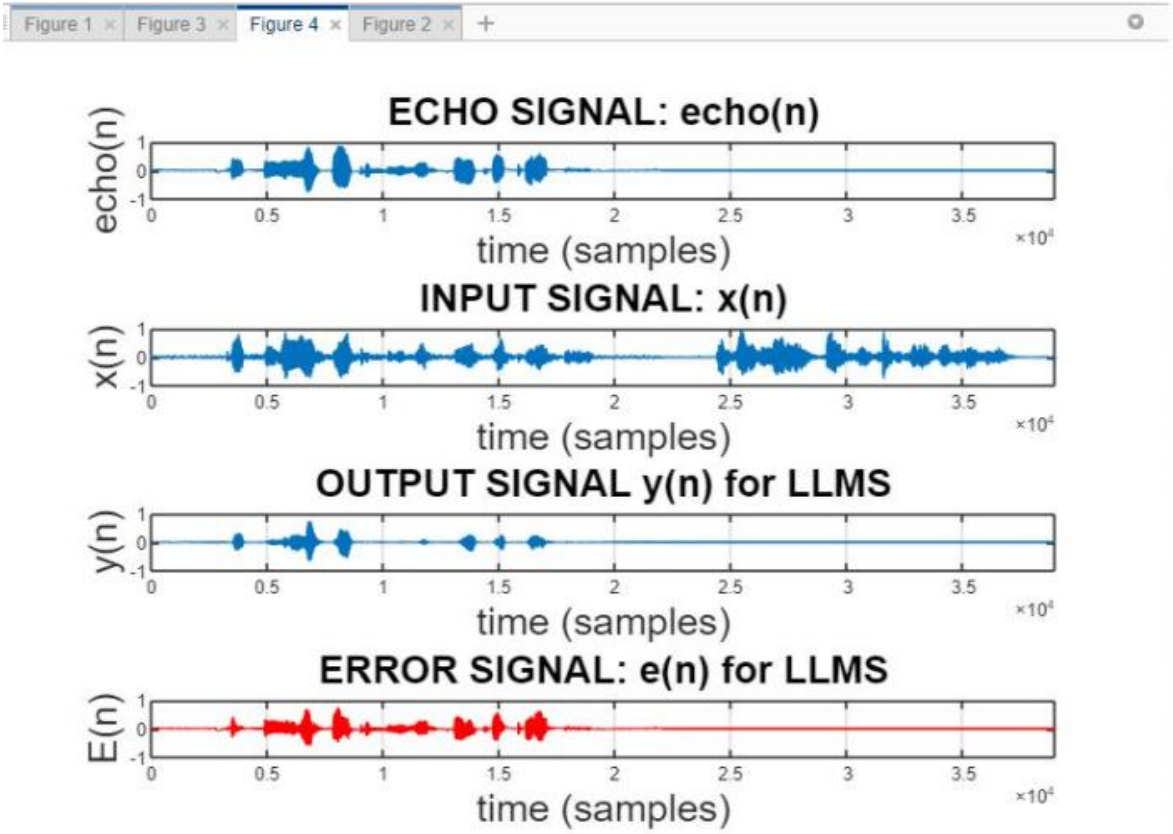


Figure 1
Figure 3
Figure 4
Figure 2
ECHO SIGNAL: echo(n)
echo(n)
time (samples)
INPUT SIGNAL: x(n)
x(n)
time (samples)
OUTPUT SIGNAL y(n) for LLMS
y(n)
time (samples)
ERROR SIGNAL: e(n) for LLMS
E(n)
time (samples)

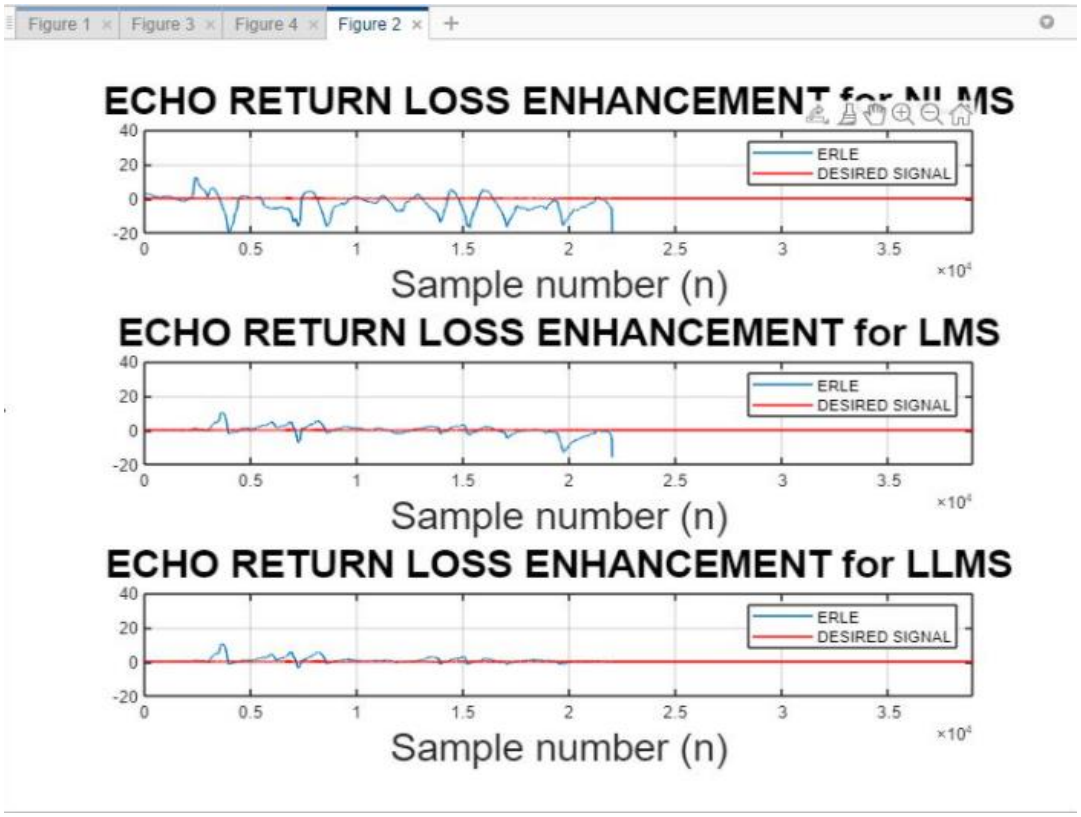


Figure 1
Figure 3
Figure 4
Figure 2
ECHO RETURN LOSS ENHANCEMENT for NLMS
ERLE
DESIRED SIGNAL
Sample number (n)
ECHO RETURN LOSS ENHANCEMENT for LMS
ERLE
DESIRED SIGNAL
Sample number (n)
ECHO RETURN LOSS ENHANCEMENT for LLMS
ERLE
DESIRED SIGNAL
Sample number (n)

APPLICATIONS

1. Low frequency drift cancelling using Adaptive Noise Filter

The use of a bias weight in an adaptive filter to cancel low-frequency drift in the primary input is a special case of notch filtering with the notch at zero frequency. A bias weight is incorporated to cancel dc level or bias and hence is fed with a reference input set to a constant value of one. The value of the weight is updated to match the dc level to be cancelled. Because there is no need to match the phase of the signal, only one weight is needed.

2. Adaptive Self Tuning Filter

The noise canceller without a reference input can be used for another important application. In many instances where an input signal consisting of mixed periodic and broadband components is available, the periodic rather than the broadband components are of interest. If the system output is taken from the adaptive filter in an adaptive noise canceller, the result is an adaptive self-tuning filter capable of extracting a periodic signal from broadband noise.

CONCLUSION

Adaptive Noise Cancellation is an alternative way of canceling noise present in a corrupted signal. The principal advantage of the method are its adaptive capability, its low output noise, and its low signal distortion. The adaptive capability allows the processing of inputs whose properties are unknown and, in some cases, non-stationary. Output noise and signal distortion are generally lower than can be achieved with conventional optimal filter configurations.

This Project indicates the wide range of applications in which Adaptive Noise Canceling can be used. The simulation results verify the advantages of adaptive noise cancellation. In each instance canceling was accomplished with little signal distortion even though the frequencies of the signal and interference overlapped. An overall study of the major algorithms were also done with real time examples

to find out that LLMS algorithm stands out to be more precise and accurate compared to the others.

REFERENCES

1. Active Noise Cancellation using Adaptive Filter Algorithms- BA Sujathakumari, Ravi S Barki, Lokesh AR, Pavithra CM / Published(Feb 2018)

2. Noise Cancellation Using Adaptive Filter Algorithms- Suman, Poonam Beniwal / Published (July-Aug 2015)

3. Comparative Study of Different Adaptive Filter Algorithms used for Effective Noise Cancellation- Nagaraju P, Gayathri S, Harshitha N, Meghna Prakash / Published (April 2014)

4. Normalized LMS Algorithm and it's uses- Mohit Mewara

5. Adaptive Filter Theory by Simen Haykin: 3rd edition, Pearson Education Asia.LPE.

6. Adaptive Signal Processing by John G Proakis, 3rd edition, Perntice Hall of India.

7. B. Widow, "Adaptive noise canceling: principles and applications", Proceedings of the IEEE, vol. 63, pp. 1692-1716, 1975. and Samuel D.Stearns; Pearson Education Asia, LPE

8. MUGDHA. M. DEWASTHALE AND R. D. KHARADKAR,(2014), "IMPROVED LMS ALGORITHM WITH FIXED STEP SIZE AND FILTER LENGTH USING ADAPTIVE WEIGHT UPDATION FOR ACOUSTIC NOISE CANCELLATION", 2014 ANNUAL IEEE INDIA CONFERENCE (INDICON), PP. 1-7.

9. JyothiDhiman, Shadab Ahmad, KuldeepGulia, „Comparison between Adaptive filter Algorithms" , International Journal of Science, Engineering and Technology Research (IJSETR), Volume 2, Issue 5, May 2013.

10. Komal R. Borisagar and Dr.G.R.Kulkarni : „Simulation and Comparative Analysis of LMS and RLS Algorithms Using Real Time Speech Input Signal“, 44 Vol.10 Issue 5 (Ver1.0)October2010,Global Journal of Researches in Engineering

11. Least Mean Squares filter- Wikipedia

12. Google.com