**A**

**Project Report**

**On**

**“SYSTEM IDENTIFICATION USING ADAPTIVE SIGNAL PROCESSING”**



ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

**Submitted to: Submitted by:**

Dr. Kulbir Singh Deepti Singh

Roll No: 801361007

Harkiranjeet Kaur

Roll No: 801361009

**CONTENTS**

**1. INTRODUCTION TO THE PROJECT**

**2. MATHEMATICAL MODELLING OF AN ADAPTIVE PROCESS (SYSTEM IDENTIFICATION)**

**3. MATLAB CODE AND RESULTS**

**INTRODUCTION TO THE PROJECT**

**ABSTRACT**

**In this project we aim to identify the communication system through an adaptive algorithm, where a BPSK modulated audio signal transmission is corrupted by AWGN noise. Corrupted signal is therefore passed through an unknown system which adaptively modifies itself resulting in the output of identified system approximately equal to the desired signal transmitted and computes the optimum coefficients of a filter resulting in the minimum error.**

**Results using LMS (Least Mean Square) and RLS (Recursive Least Squares) Adaptive Algorithms have been presented. Also the reduction of MSE (Mean Square Error) with the number of iterations is illustrated with the help of a graph with LMS. Both the algorithms are analyzed respectively.**

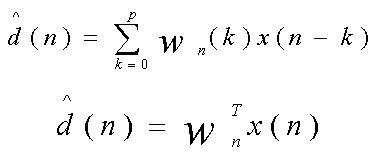
**ADAPTIVE FIR FILTER**

FIR filters are usually used in adaptive filtering applications that range from adaptive equalization in digital communication systems to adaptive noise control systems. The reasons for the popularity of FIR adaptive filters are (i) The filter coefficients control the stability easily (ii)There are simple and efficient algorithms for adjusting the filter coefficients (iii)The performance of these algorithms is well understood in terms of their convergence and stability.



Fig 1.a.) General Adaptive Filter Model b.) Adaptive Filter with its coefficients

An FIR adaptive filter for estimating a desired signal d (n) from a related signal x (n)



here it is assumed that x(n) and d(n) are non stationary Random process and the goal is to find the coefficient vector wn at the time n that minimizes the mean-square error,

ξ(n) =E{|e(n)|2}

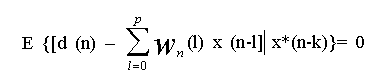
where, e(n) = d(n)-d’(n)

= d(n)-wnT x(n)

As in the derivation of the FIR wiener filter, the solution to this minimization problem may be found by setting the derivative of ξ(n) with respect to wn\*(k) equal to zero for k = 0,1,2,…….,p. the result is:

E{e(n)x\*(n-k)} = 0;

Substituting e(n) into above equation we have

k = 0,1,…..,p

Above equation is a set of p+1 linear equations in the p+1 unknowns wn (l). However, unlike the case of an FIR Wiener filter where it was assumed that x(n) and d(n) are jointly wide sense stationary, the solution to these equations depends on n.

**MATHEMATICAL MODELING OF AN ADAPTIVE PROCESS (SYSTEM IDENTIFICATION)**

**Step 1**: **Load a .wav extension file**

Consider an audio file having .wav extension and load it in the program.

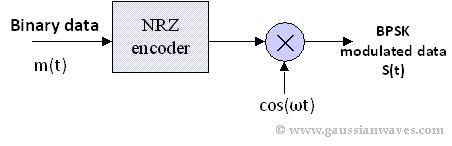
Wavread: It is a matlab function to read .wav sound file. Wavread (wavfile) loads a sound file specified by the string wavfile, returning the sampled data in y. Amplitude values are in the range [-1, +1].

**Step 2**: **Convert wave file into a binary file**

Convert the waveread file into single precision unsigned integer form, which is then converted into binary data (0’s and 1’s) using dec2bin after typecasting command in matLAB.

**Step 3**:

Apply **BPSK modulation** to the binary data.



In Binary Phase Shift Keying (BPSK) only one sinusoid is taken as basis function modulation. Modulation is achieved by varying the phase of the basis function depending on the message bits. The following equation outlines BPSK modulation technique.

i.e. if So(t) = A sin (ω\*t) represents 0,

then So(t) = A sin (ω\*t + π) represents 1.

The constellation diagram of BPSK has all points lying entirely on the x axis. It has no projection on the y axis. That is because the BPSK modulated signal will have an in-phase component (I) but no quadrature component (Q). It is having only one basis function.

A BPSK modulator can be implemented by [**NRZ coding**](http://www.gaussianwaves.com/2010/01/matlab-simulation-psd-of-line-codes/)the message bits (1 represented by + ve voltage and 0 represented by -ve voltage).

Sign operator can be used to implement BPSK in matLAB.

**Step 4: AWGN CHANNEL**

Pass the modulated signal through AWGN channel. An AWGN channel adds white Gaussian noise to the signal that passes through it.

**awgn** is the MatLAB command which adds this noise according to the SNR specified by the programmer.

x(n)=s(n) + ,

where x(n) is a noisy signal and s(n) is a transmitted signal, represents additive Gaussian noise.

**Step 5:**

**BPSK Demodulation**

The received noisy signal after awgn channel is demodulated to get the wave in binary format again (i.e. 0’s and 1’s). Demodulation is performed through decision making process i.e.

if received value is **< 0**, it is **0**,

else if it is **>0**, it is considered to be **1.**

**Step 6:**

**Reconvert the binary sequence back to wave file** in the format same as, in step 1, using bin2dec operator after typecasting.

**Step7:**

**ADAPTIVE FILTERING PROBLEM (Mathematical Analysis)**

d(n) is the desired wave file as in step 1

****

Either LMS or RLS algo

X(n) is the noisy wave file obtained in step 6

Thus, from the figure we have,

At n th iteration,

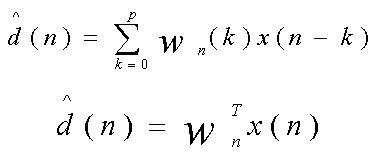
input signal to filter = x(n) ; noisy wave

desired signal is the original wave file = d(n).

Assumed filter coefficients at first iteration w(n)=0.

Filter length=32.

Calculated filter output is:



Error calculated at every iteration is:

e(n) = d(n)-d’(n)

= d(n)- w n T x(n)

**WEIGHT UPDATION BASED ON THE ERROR**

**Adaptive Filtering Algorithm**

Many proposed algorithms are available for cancelling the added noise. The algorithms are, Least mean square algorithm, Normalized LMS algorithm and Recursive least square algorithm

**LMS Algorithm**

We developed the steepest descent adaptive filter, which has a weight vector update equation given by

wn+1 = wn + μE{e(n) x\*(n)}

A practical limitation with this algorithm is that the expectation E{e(n)x\*(n)} is generally unknown. Therefore, it must be replaced with an estimate such as the sample mean.

A special case above equation occurs if we use a one point sample mean (L=1),

Ê{e(n)x\*(n)}= e(n)x\*(n)

In this case, the weight vector update equation assumes a particularly simple form

wn+1 =wn + μ e(n)x\*(n)

and is known as the LMS algorithm. The simplicity of the algorithm comes from the fact that the update of the kth coefficient

***wn+1(k) =wn (k) + μ e(n)x\*(n)***,

requires only one multiplication and one addition (the value for μ e(n) need only the computed once and may be used for all of the coefficients).

**RLS Algorithm**

Recursive least squares (RLS) algorithm is used to find the filter coefficients that relate to producing the recursively least squares of the error signal (difference between the desired and the actual signal). We have considered gradient descent algorithms for the minimization of the mean-square error.

The error implicitly depends on the filter coefficients through the estimate:

e(n)=d(n)-\hat{d}(n)

The weighted least squares error function C—the cost function we desire to minimize—being a function of e(n) is therefore also dependent on the filter coefficients:

C(\mathbf{\mathbf{w}_n})=\sum_{i=0}^{n}\lambda^{n-i}e^{2}(i)

The RLS algorithm for a *p*-th order RLS filter can be summarized as

|  |  |
| --- | --- |
| Parameters: | p= filter order |
|  | \lambda= forgetting factor |
|  | \delta= value to initialize \mathbf{P}(0) |
| Initialization: | \mathbf{w}(n)=0 |
|  | x(n)=noisy signal wave.  d(n)=Original signal |
|  |  |
|  | \mathbf{P}(0)=\delta^{-1}I where I is the [identity matrix](http://en.wikipedia.org/wiki/Identity_matrix) of rank p+1 |
| Computation: | For n=1,2,\dots |
|  | \mathbf{x}(n) =  \left[ \begin{matrix} x(n)\\ x(n-1)\\ \vdots\\ x(n-p) \end{matrix} \right] |
|  | \alpha(n) = d(n)-\mathbf{x}^T(n)\mathbf{w}(n-1) |
|  | \mathbf{g}(n)=\mathbf{P}(n-1)\mathbf{x}^*(n)\left\{\lambda+\mathbf{x}^{T}(n)\mathbf{P}(n-1)\mathbf{x}^*(n)\right\}^{-1} |
|  | \mathbf{P}(n)=\lambda^{-1}\mathbf{P}(n-1)-\mathbf{g}(n)\mathbf{x}^{T}(n)\lambda^{-1}\mathbf{P}(n-1) |
|  |  |

Where g(n) is the Kalman Gain and is the error vector.

Weight updation is carried out as:

 \mathbf{w}(n) = \mathbf{w}(n-1)+\,\alpha(n)\mathbf{g}(n).

**CALCULATION OF MSE (Mean Square Error)**

Mean Square error tells us about the deviation of actual output from the desired output, thus it is calculated at each iteration to check whether algorithm is approaching towards the desired response or not.

It is calculated by squaring all the elements of the error vector and thus taking the average of their summation.

If \hat{Y} is a vector of n predictions, and Y is the vector of the true values, then the (estimated) MSE of the predictor is:

\operatorname{MSE}=\frac{1}{n}\sum_{i=1}^n(\hat{Y_i} - Y_i)^2.

**CONCLUSION**

With the suitable number of iterations system adapts itself to behave nearly as the system to be identified with a minimum possible error. Final coefficients give the desired response.

**MATLAB CODE AND RESULTS**

%LMS ALGORITHM

%%%wave file

clc

clear all

nsec = 1;

[wave,fs]=wavread('C:/Users/Harkiran/Desktop/BassClarinet\_model1.wav');

figure(1)

subplot(3,1,1);

plot(wave);

title('Original Sound Wave');

%%%%%% DECIMAL TO BINARY

wavbinary = dec2bin( typecast( single(wave(:)), 'uint8'), 8 ) - '0';

[rows\_ori columns\_ori] = size(wave)

[rows columns]=size(wavbinary);

%%%%%%BPSK

bpsk\_bits=sign(wavbinary);

%%%%%AWGN NOISE CHANNEL

Noisy\_wave=awgn(bpsk\_bits,10);

Noisy\_wave1=awgn(wave,10);

subplot(3,1,2);

plot(Noisy\_wave1);

title('Noisy Sound Wave');

%%%% NOISY SIGNAL RECEPTION

for i=1:rows

for j= 1:columns

if (Noisy\_wave(i,j)>0)

Noisy\_wave(i,j)=1;

else

Noisy\_wave(i,j)=0;

end

end

end

%%%%% BINARY TO DECIMAL

wavdata = reshape( typecast( uint8(bin2dec( char(Noisy\_wave + '0') )), 'single' ) ,rows\_ori, columns\_ori );

z=sum(wave~=Noisy\_wave1)

wavwrite(Noisy\_wave1,fs,'C:/Users/Harkiran/Desktop/Noisy');

%%%%% LMS MODULE %%%%%%

L=32;

mu=0.08;

%%%% loop for updating coefficients %%%%

d=wave';

x=Noisy\_wave1';

start\_iter = 1;

end\_iter = min([length(x) length(d)]);

W = zeros(L,end\_iter);

dhat = zeros(end\_iter,1);

e = zeros(end\_iter,1);

W0=zeros(L,1);

W(:,1:start\_iter) = W0\*ones(1,start\_iter);

X = zeros(L,1);

W\_final=zeros(1,L);

mse\_value=zeros(1,end\_iter);

for n = start\_iter:end\_iter;

X(2:L) = X(1:L-1);

X(1) = x(n);

dhat(n) = X'\*W(:,n);

e(n) = d(n) - dhat(n);

mse\_value(n)=abs(mse(e(n)));

W(:,n+1) = W(:,n) + mu\*e(n)\*X;

W\_final=(W(:,n+1))';

end

y=filter(W\_final,1,x);

m=mse(y-d)

%%%%Initialize weight vectors for different iterations

W\_COEFF\_0=zeros(1,32);

W\_COEFF\_100=zeros(1,32);

W\_COEFF\_500=zeros(1,32);

W\_COEFF\_700=zeros(1,32);

W\_COEFF\_final=zeros(1,32);

%%%%% Value of weights at different iterations

W\_COEFF\_0=W(:,1)'

mse\_value(1)

W\_COEFF\_100=W(:,100)'

mse\_value(100)

W\_COEFF\_500=W(:,500)'

mse\_value(500)

W\_COEFF\_700=W(:,700)'

mse\_value(700)

W\_COEFF\_final=W\_final'

mse\_value(end\_iter)

subplot(3,1,3);

plot(y);

title('Sound Wave Recovered');

figure(2)

subplot(1,1,1);

plot(mse\_value);

figure(3);

subplot(2,1,1); plot(1:rows\_ori,[d;y;e']);

title('System Identification of an FIR Filter');

legend('Desired','Output','Error');

xlabel('Time Index'); ylabel('Signal Value');

subplot(2,1,2); stem(W\_final);

legend('Estimated');

xlabel('Coefficient #'); ylabel('Coefficient value'); grid on;

%%%%%%WAVEWRITE

wavwrite(y,fs,'C:/Users/Harkiran/Desktop/FinalWithAlgo');

**RESULTS: COEFFICIENTS AT DIFFERENT ITERATIONS (**after 0th, 100th , 500th , 700th and Final coefficients at the end**). And also the MSE is calculated at each iteration.**

**Graphs are given below.**

W\_COEFF\_0 =

Columns 1 through 15

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 16 through 30

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Columns 31 through 32

0 0

mse\_value =

0

W\_COEFF\_100 =

1.0e-004 \*

Columns 1 through 9

-0.0312 0.1604 -0.1768 0.1233 -0.0047 -0.1167 0.1718 -0.1105 0.0846

Columns 10 through 18

-0.0328 0.0312 -0.0519 0.0363 -0.0309 -0.0207 0.0526 -0.0883 0.1446

Columns 19 through 27

-0.1637 0.1510 -0.1048 0.0175 0.0441 0.0282 -0.0832 0.0416 -0.0070

Columns 28 through 32

0.0398 -0.0144 -0.0139 -0.0227 0.0886

mse\_value =

2.6373e-009

W\_COEFF\_500 =

Columns 1 through 9

0.0043 0.0034 0.0004 0.0000 0.0012 0.0009 0.0013 0.0026 0.0021

Columns 10 through 18

0.0036 0.0052 0.0070 0.0080 0.0088 0.0064 0.0053 0.0034 0.0020

Columns 19 through 27

-0.0001 0.0007 0.0002 0.0001 0.0001 -0.0005 0.0003 0.0001 0.0011

Columns 28 through 32

0.0012 0.0010 0.0009 -0.0002 0.0002

mse\_value =

0.0013

W\_COEFF\_700 =

Columns 1 through 9

0.0116 0.0115 0.0097 0.0090 0.0084 0.0085 0.0085 0.0077 0.0074

Columns 10 through 18

0.0073 0.0066 0.0068 0.0068 0.0074 0.0089 0.0085 0.0095 0.0110

Columns 19 through 27

0.0113 0.0110 0.0101 0.0108 0.0109 0.0104 0.0104 0.0107 0.0103

Columns 28 through 32

0.0097 0.0097 0.0084 0.0078 0.0066

mse\_value =

1.0534e-005

W\_COEFF\_final =

0.0008

0.0008

0.0005

0.0008

0.0008

0.0006

0.0005

0.0005

0.0005

0.0005

0.0005

0.0006

0.0009

0.0008

0.0006

0.0006

0.0005

0.0009

0.0011

0.0012

0.0015

0.0011

0.0012

0.0011

0.0013

0.0012

0.0012

0.0012

0.0012

0.0011

0.0012

0.0014

mse\_value =

1.9650e-005

**Graphs LMS:**

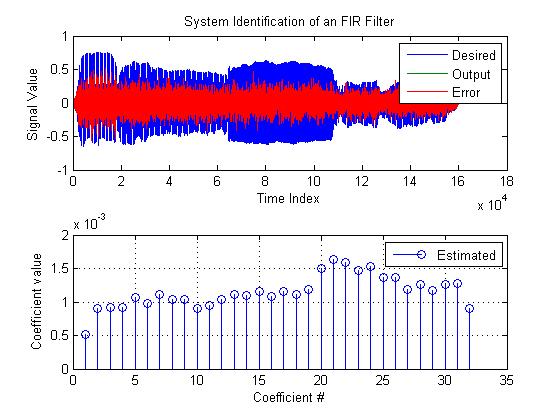


Figure a.): Output at every Iteration using LMS and desired output. Error between them is shown in red. b.) Coefficients final updated values.

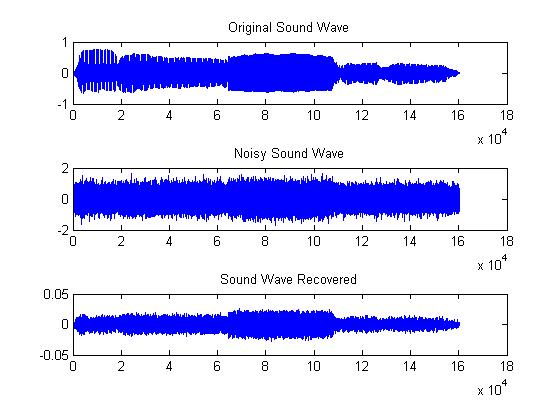


Figure: Comparison of Original sound wave, corrupted sound wave and recovered sound wave.

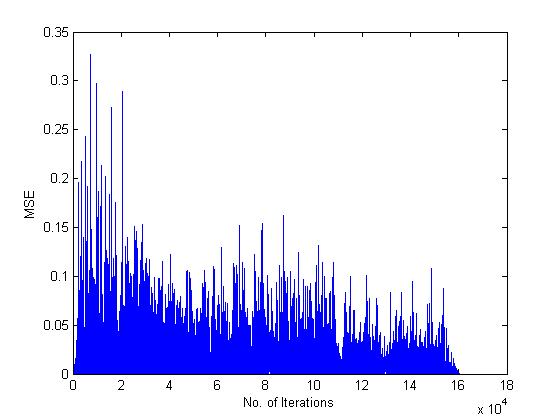


Figure: Variation of MSE with number of Iterations.

**2. Using Inbuilt RLS Function**

% RLS Algorithm

%%%wave file

clc

clear all

nsec = 1;

[wave,fs]=wavread('C:/Users/Harkiran/Desktop/BassClarinet\_model1.wav');

figure(1)

subplot(3,1,1);

plot(wave);

xlabel('Time Index'); ylabel('Signal Value');

title('Sound Wave Original');

%%%%%% DECIMAL TO BINARY

wavbinary = dec2bin( typecast( single(wave(:)), 'uint8'), 8 ) - '0';

[rows\_ori columns\_ori] = size(wave)

[rows columns]=size(wavbinary);

%%%%%%BPSK

bpsk\_bits=sign(wavbinary);

%%%%%AWGN NOISE CHANNEL

Noisy\_wave=awgn(bpsk\_bits,10);

Noisy\_wave1=awgn(wave,10);

subplot(3,1,2);

plot(Noisy\_wave1);

xlabel('Time Index'); ylabel('Signal Value');

title('Noisy Sound Wave');

%%%% NOISY SIGNAL RECEPTION

for i=1:rows

for j= 1:columns

if (Noisy\_wave(i,j)>0)

Noisy\_wave(i,j)=1;

else

Noisy\_wave(i,j)=0;

end

end

end

%%%%% BINARY TO DECIMAL

wavdata = reshape( typecast( uint8(bin2dec( char(Noisy\_wave + '0') )), 'single' ) ,rows\_ori, columns\_ori );

z=sum(wave~=Noisy\_wave1)

wavwrite(Noisy\_wave1,fs,'C:/Users/Harkiran/Desktop/Noisy');

%%%%% RLS MODULE %%%%%%

L=32;

mu=0.08;

%%%% loop for updating coefficients %%%%

d=wave';

x=Noisy\_wave1';

b = fir1(31,0.5); % FIR system to be identified

P0 = 10\*eye(32); % Initial sqrt correlation matrix inverse

lam = 0.99; % RLS forgetting factor

ha = adaptfilt.rls(32,lam,P0);

[y,e] = filter(ha,x,d);

m=mse(e)

subplot(3,1,3);

plot(y);

xlabel('Time Index'); ylabel('Signal Value');

title('Sound Wave Recovered');

figure(2);

subplot(2,1,1); plot(1:rows\_ori,[d;y;e]);

title('System Identification of an FIR Filter');

legend('Desired','Output','Error');

xlabel('Time Index'); ylabel('Signal Value');

subplot(2,1,2); stem([b.',ha.Coefficients.']);

legend('Actual','Estimated');

xlabel('Coefficient #'); ylabel('Coefficient value'); grid on;

%%%%%%WAVEWRITE

wavwrite(y,fs,'C:/Users/Harkiran/Desktop/Final');

**RESULTS of RLS:**

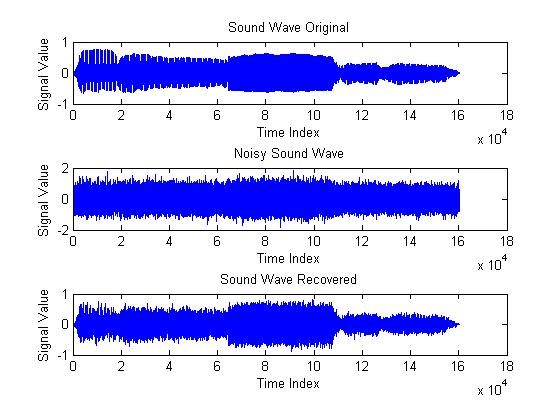


Figure: Comparison of Original sound wave, corrupted sound wave and recovered sound wave.

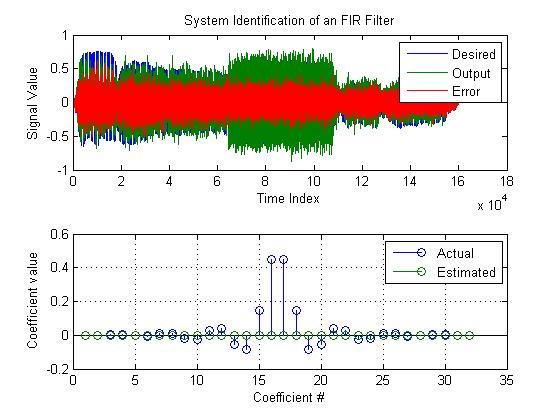


Figure a.): Output at every Iteration using RLS and desired output. Error between them is shown in red. b.) Coefficients final updated values.