DESIGN OF ADAPTIVE FILTER AND IT'S SIMULATION ON MATLAB

By

KANHAIYA KUMAR-021068
PRASHANT KUMAR-021057

MAY-2006

Submitted in partial fulfillment of the degree of bachelor of Technology

DEPARTMENT OF ELECTRONICS
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY-WAKNAGHAT

CERTIFICATE

This is to certify that the work entitled , "adaptive filter " submitted by Kanhaiya Kumar in partial fulfillment for the award of degree of bachelor Of technology in electronics and communication of Jaypee University Of Information Technology Has been carried out under my supervision. this work has not been submitted Partially or wholly to any other university or institute for the award of this or any other degree or diploma.

Prof.(Dr) S.V.Bhooshan.

Acknowledgement

Many people have contributed to this project in a variety of ways over the past few months.. We also acknowledge the many helpful comments received from other teachers of the different departments and visualization courses and seminars. I are indebted to all those who provided reviews and suggestions for improving the materials and topics covered in our package, and I extend our apologies to anyone we may have failed to mention.

Thank You: Kanhaiya Kumar.
Prashant Kumar.

TABLE OF CONTENTS

LIST OF FIGURES

List of Abbreviations

1. μ = step size
2. f(X[n]) = function of input
3. f(e[n]) =function of error
4. X[n] = primary signal
5. S[n] = voice
6. Y[n] =noise

ABSTRACT

Introduction –The Evolution of Convolution

In a general sense, adaptive filters are systems that vary through time because the characteristics of its inputs may be varying. That is what separates it from classical digital signal processing - the digital system itself changes through time. In a sense, its convolution properties are evolving.

Therefore, adaptive filters must be non-linear because superposition does not hold. But when their adjustments are held constant after adaptation, then some can become linear, and belong under the well-named class linear adaptive filters. We will be working with those in this project.

Whenever there is a requirement to process signals in an environment of unknown statistics, adaptive filters will more often than not do the job better than a fixed filter.

An adaptive filter can track that noise, follow its characteristics, and knock it out so that you may have a good, clean conversation.

Other applications besides noise cancellation include system identification, signal prediction, source separation, channel equalization, and more.

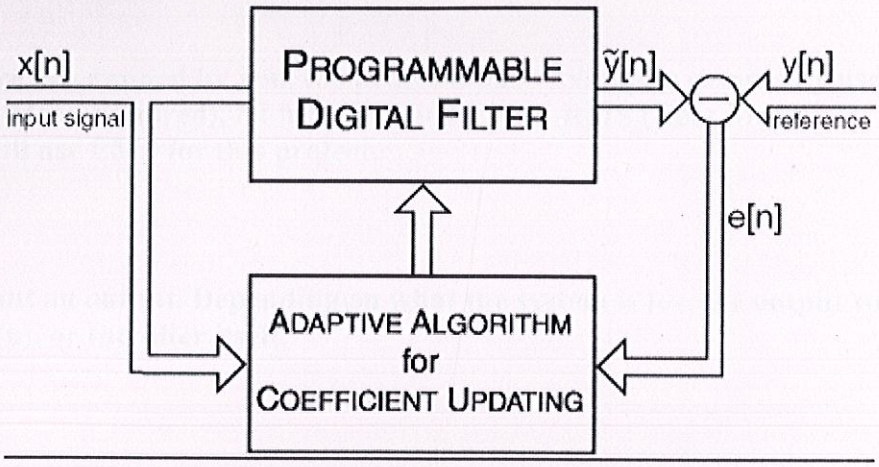
PURPOSE

This project is an introduction to adaptive filters. A summation of what we learned and put together:

1. **General theory**
    - **What's the math involved**
    - **What's the algorithm**
    - **What's it for**
2. **Simulation in MATLAB with Least Mean Squared algorithm**
    - **Noise cancellation**
    - **FIR identification**
3. **Implementation of noise cancellation in real-time with TI DSP board (Research Work)**

DESIGN & SIMULATION

General Form of Adaptive Filters



Pictured here is the general structure of an adaptive filter. The main points to be noted here are

1. **The system takes in two inputs**
2. **The top box has one input**
3. **The bottom box has that same input plus an error input**
4. **The top box is being changed by the bottom box**
5. **What's the output?**

If the bottom box is the brain, then the top box is the body. The brain is using what it knows and putting it through an algorithm in which to control the body. The algorithm is always of the form

General Form of Algorithm for Updating Coefficients

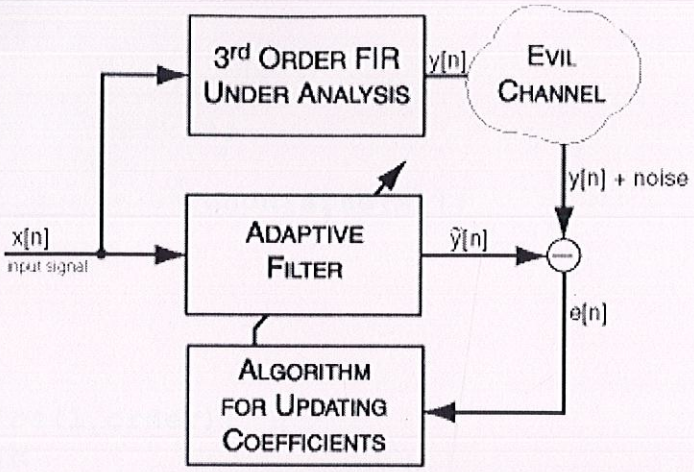
$$\begin{bmatrix}\text{New}\\ \text{Parameter}\end{bmatrix} = \begin{bmatrix}\text{Old}\\ \text{Parameter}\end{bmatrix} + \underbrace{\begin{bmatrix}\text{Step}\\ \text{Size}\end{bmatrix}}_{\mu} \bullet \underbrace{\begin{bmatrix}\text{Function}\\ \text{of Input}\end{bmatrix}}_{f(x[n])} \bullet \underbrace{\begin{bmatrix}\text{Function}\\ \text{of Error}\end{bmatrix}}_{f(e[n])}$$

The functions are determined by you, the programmer. Examples commonly used are LMS (Least Mean Squared), NLMS (Normed LMS), RMS (Root Mean Squared). We will use LMS for this project.

The user will want an output. Depending on what the system is for, the output will be either y(n), e(n), or the filter itself.

SYSTEM IDENTIFICATION

Adaptive Filter for FIR Filter Identification



The bottom two boxes are our adaptive system. It is figuring out what the top box is. The top box is a Finite Impluse Response (FIR) filter programmed by MATLAB with the magic command "filter(B,A,signal)." The filter is a "Direct Form II Transposed" implementation of the standard difference equation:

```
a(1)*y(n) = [b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-
nb)] - [a(2)*y(n-1) + ... + a(na+1)*y(n-na)]
```

We simulated this filter with the command "filter(rand(3,1), 1, signal)" This sets a(1) in the difference equations to one. The first three B coefficients are given random numbers. That's what our adaptive filter is going to figure out. The rest of the coefficients are set to zero.

```
% LMS identification of FIR filter

clear
close all

N = 50;
x = randn(N,1);
t = [1:N];
order = 10;
h = order*rand(order,1);

y = filter(h,1,x)+.5*randn(size(x));

plot(y)

mu = .1;
L = order-1;
w(L,:) = zeros(1,order);
for i=order:N

        z = x(i-L:i);
        w(i,:) = w(i-1,:)-mu*(w(i-1,:)*z-y(i))*z';

end

%plot(t,w(:,3),'red',t,w(:,2),'blue',t,w(:,1),'green')
plot(t,w(:,10),t,w(:,9),t,w(:,8),t,w(:,7),t,w(:,6),t,w(:,5)
,t,w(:,4),t,w(:,3),t,w(:,2),t,w(:,1))
legend('b1','b2','b3','b4','b5','b6','b7','b8','b9','b10')
xlabel('time')
ylabel('values of B coefficients')
title('Adapting B coefficients vs. time')
h
```

Now to put our boxes into action. As seen by the block diagram, any random is signal is fed into our system and the unknown filter. The signal is filtered through the unknown, and our initial guess of the unknown. The difference of the outputs (plus some noise) e(n) is taken and put through our magical coefficient updating algorithm to get our programmable digital filter closer to the unknown. As more signal is fed through, our digital filter will start to mimic the unknown. It's learning

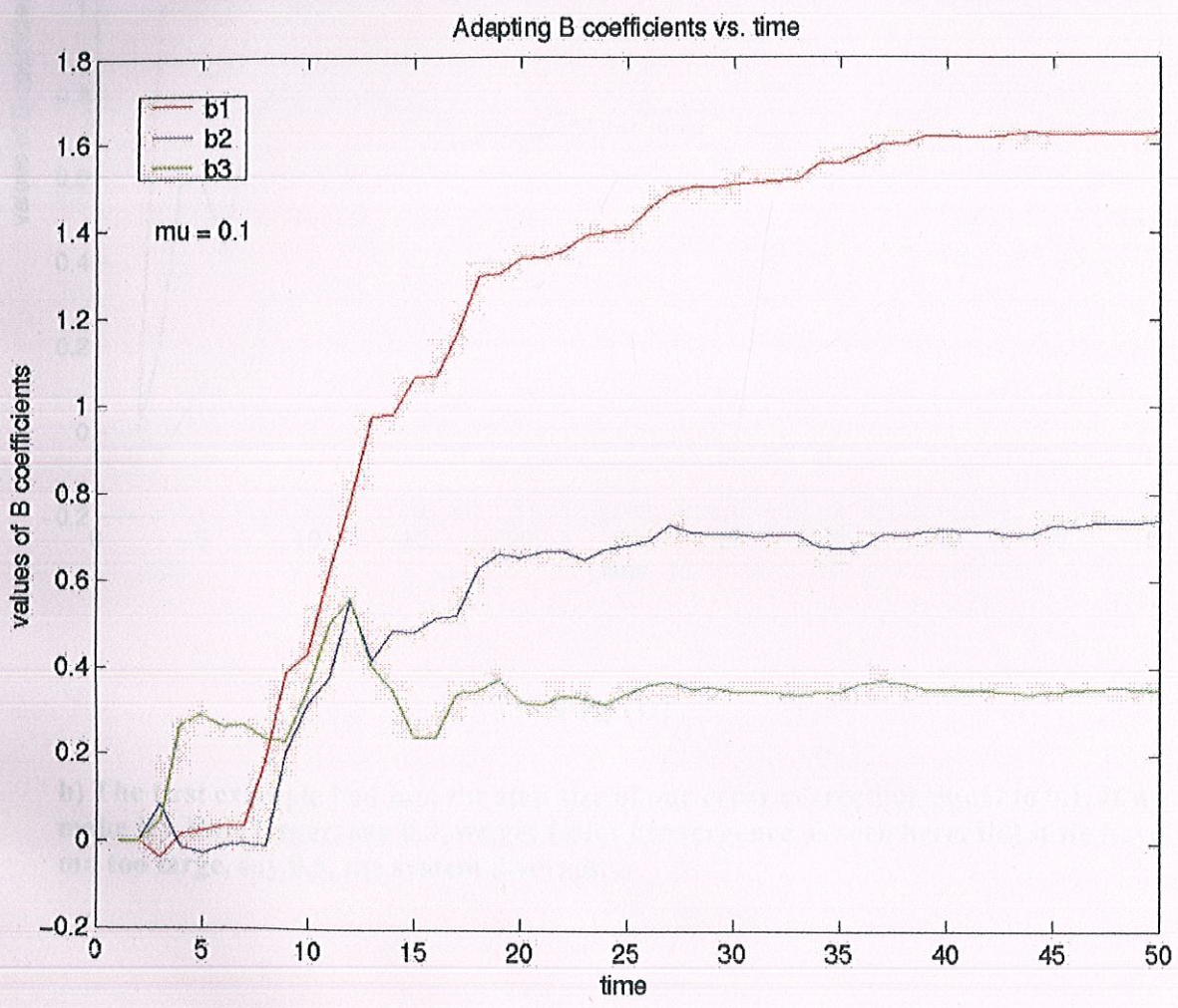


Fig (a)

a) This is a plot of the coefficients of our adaptive filter vs. time. You can see the values converge to the unknown values. So now we know b(1)=1.65, b(2)=0.72, b(3)=0.38. We have our system identified.

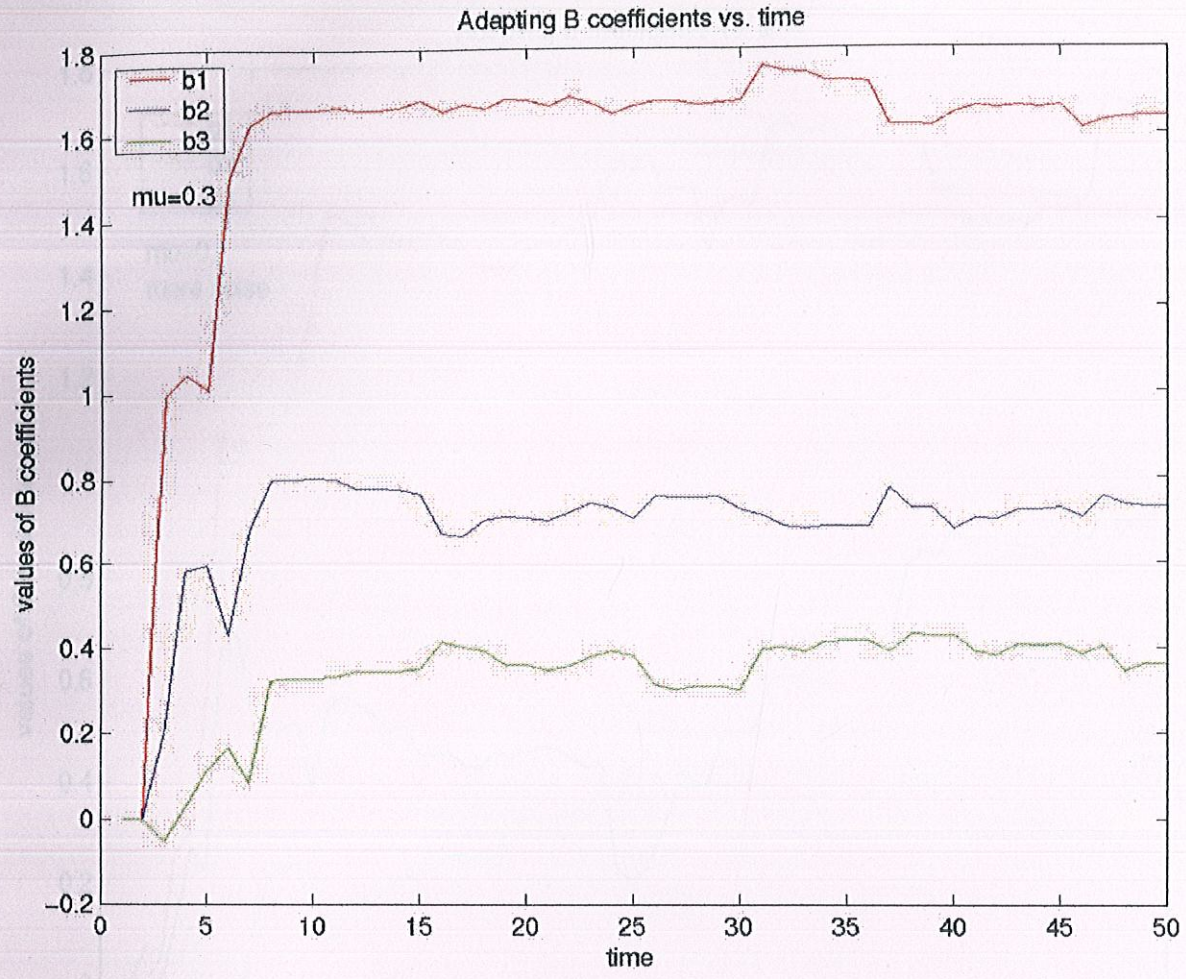


Fig (b)

b) The first example had mu, the step size of our error correction, equal to 0.1. If we make it a little larger, say 0.3, we get faster convergence as seen here. But it we have mu too large, say 0.5, the system diverges.

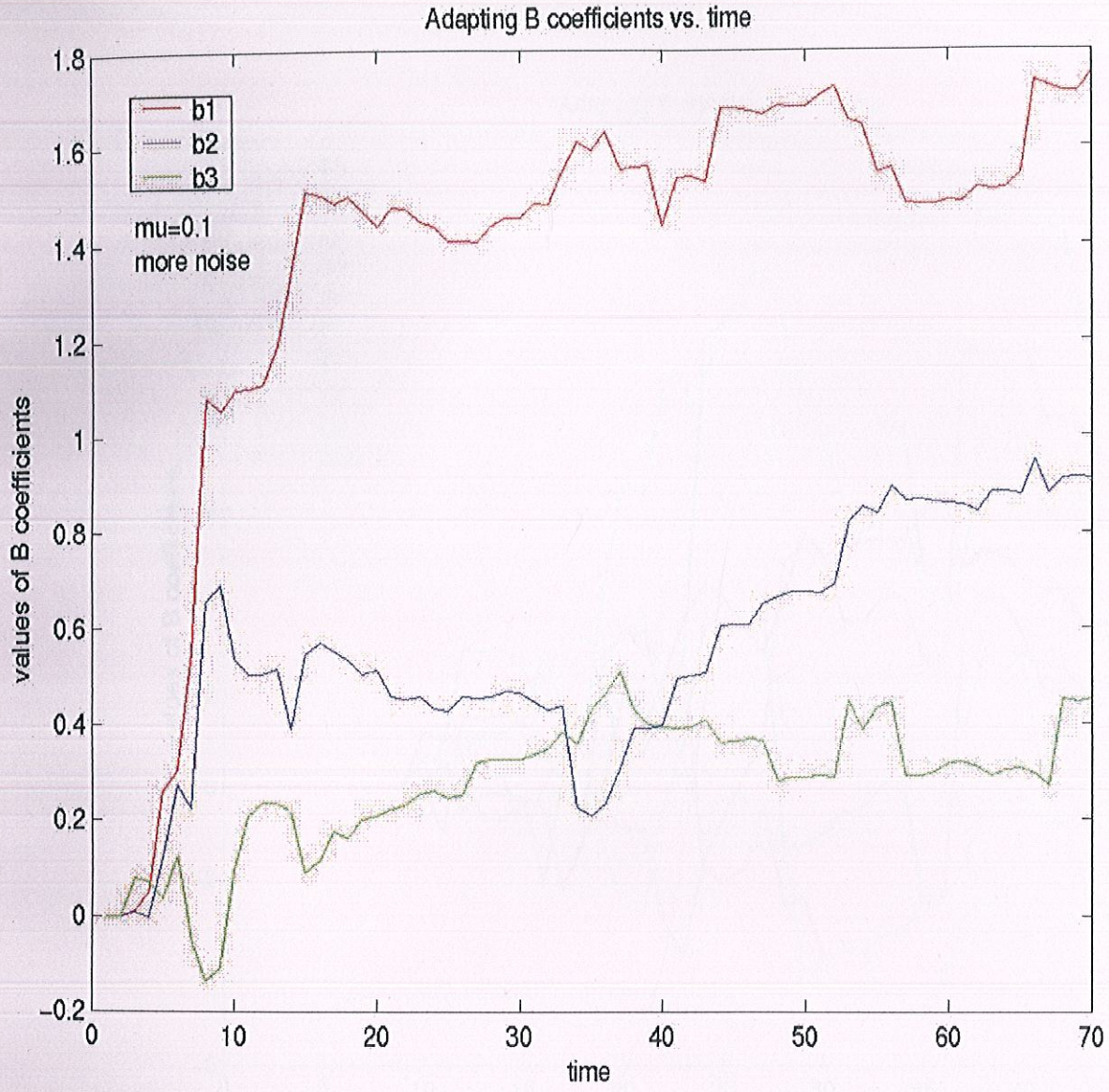


Fig (c)

c) If we have the noise turned up, the adaptive filter will oscillate more, but we can still make out that it converges to an estimate of the B coefficients.

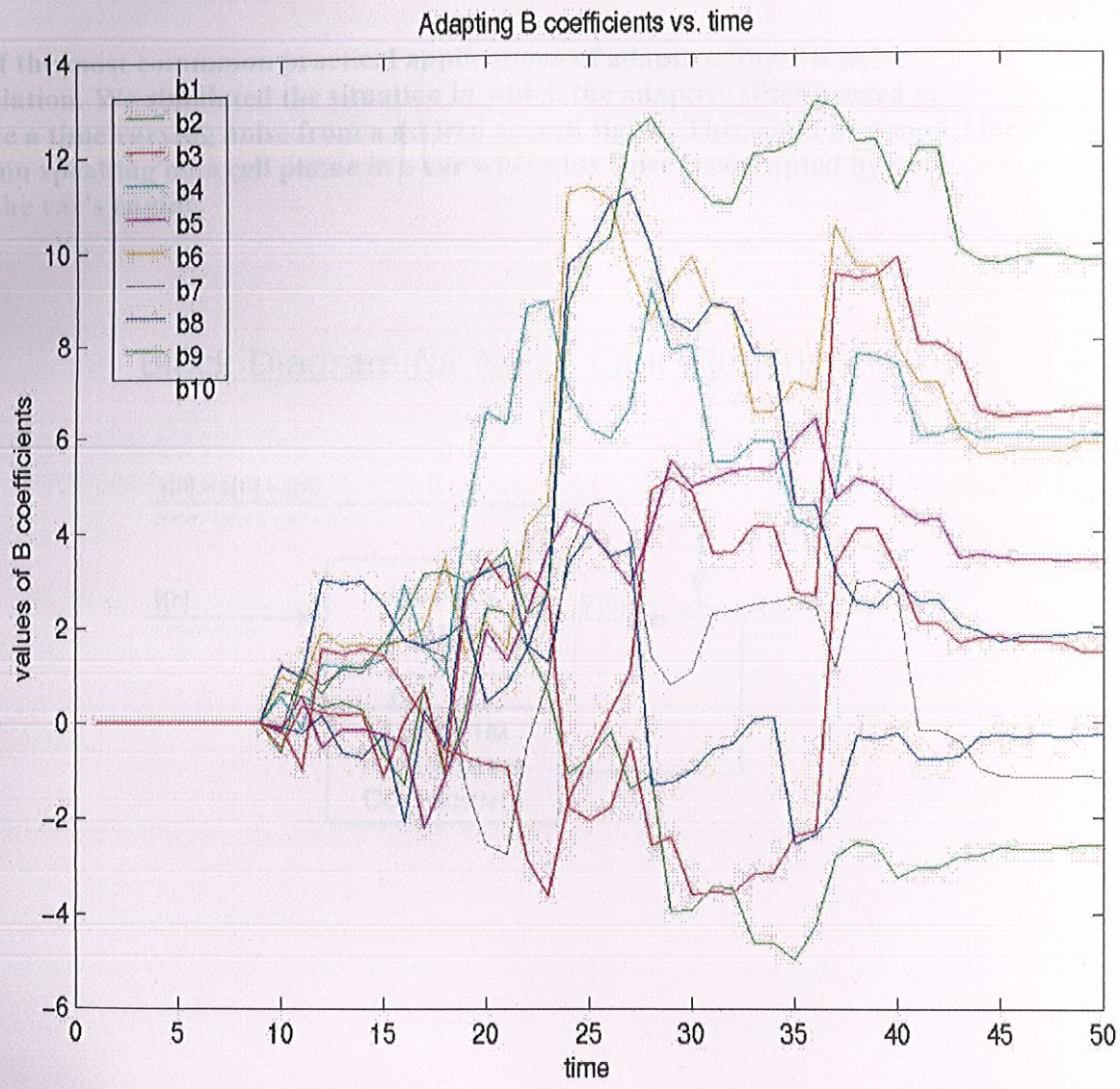


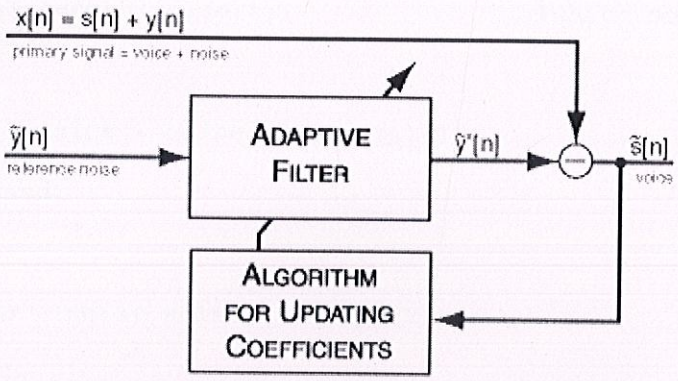
Fig (d)

d) We can also figure out any n-order FIR filter. Here's a 10th order filter.

NOISE CANCELLATION

One of the most commmon practical applications of adaptive filters is noise cancellation. We simulated the situation in which the adaptive filter needed to remove a time varying noise from a desired speech signal. This could be a model for a person speaking on a cell phone in a car where his voice is corrupted by noise from the car's engine.

Block Diagram for Noise Cancellation



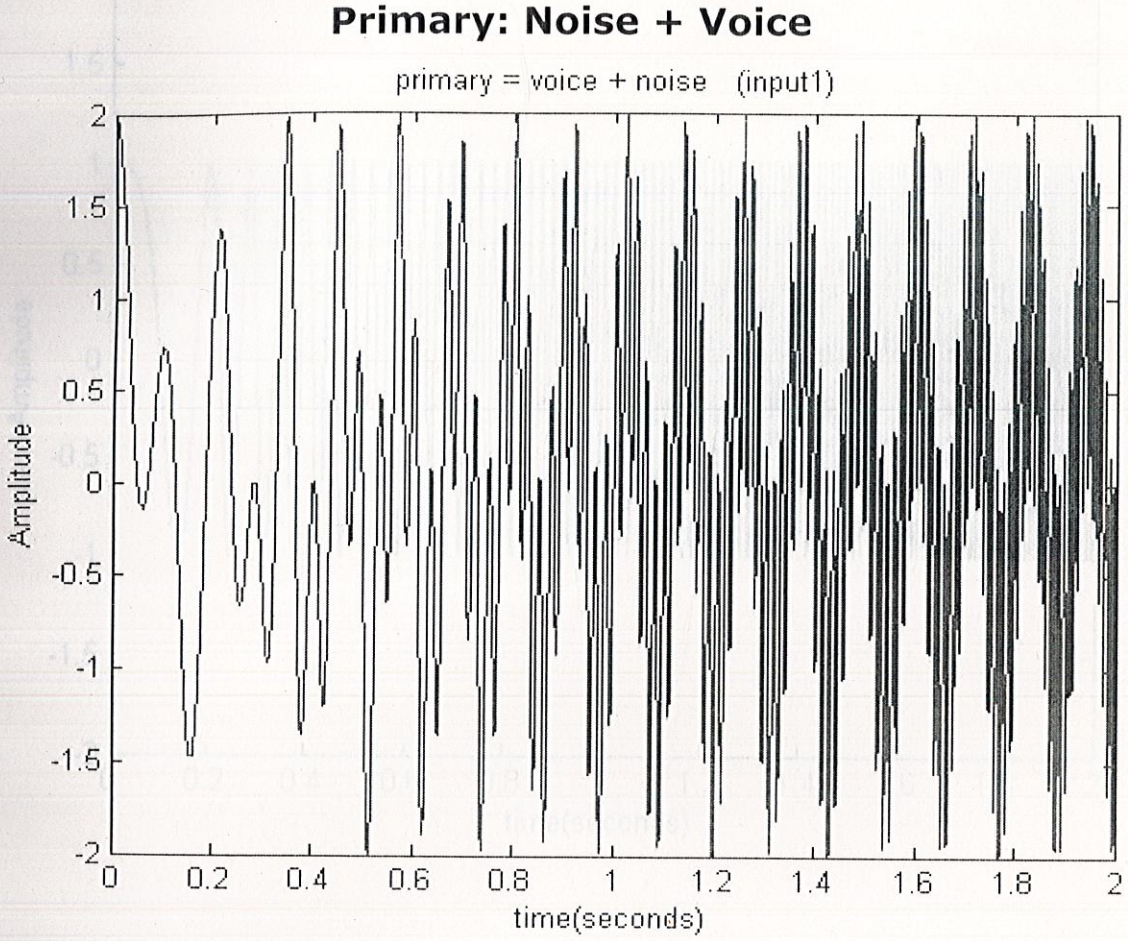
```
%noise cancellation

clear
close all

order=2;

size=2;                          %time duration of inputs
fs=8192;                           %digital sampling frequency
t=[0:1/fs:size];
N=fs*size;                       %size of inputs
f1=35/2;                           %frequency of voice
f2=99/2;                           %frequency of noise

voice=cos(2*pi*f1*t);
subplot(4,1,1)
plot(t,voice);
title('voice    (don''t have access to)')

noise=cos(2*pi*f2*t.^2);                                    %increasy
frequency noise
%noise=.1*rand(1,length(voice));                  %white noise
primary=voice+noise;
subplot(4,1,2)
plot(t,primary)
title('primary = voice + noise    (input1)')

ref=noise+.25*rand;                                                 %noisy
noise
subplot(4,1,3)
plot(t,ref)
title('reference  (noisy noise)    (input2)');

w=zeros(order,1);
mu=.006;
for i=1:N-order
   buffer = ref(i:i+order-1);
%current 32 points of reference
   desired(i) = primary(i)-buffer*w;                        %dot product
reference and coeffs
   w=w+(buffer.*mu*desired(i)/norm(buffer))';%update coeffs
end

subplot(4,1,4)
plot(t(order+1:N),desired)
title('Adaptive output  (hopefully it''s close to "voice")')
```

```
%noise cancellation

clear
close all

order=2;

size=2;                              %time duration of inputs
fs=8192;                                %digital sampling frequency
t=[0:1/fs:size];
N=fs*size;                           %size of inputs
f1=35/4;                                %frequency of voice
f2=99/4;                                %frequency of noise

%/-------------------------------------------------------------------------


%/-------------------------------------------------------------------------
voice=cos(2*pi*f1*t);
%plot(t,voice);
%axis([0  2  -2  2])
%title('voice    (don''t have access to)')
%xlabel('time(seconds)')
%ylabel('Amplitude')
%*-------------------------------------------------
noise=cos(2*pi*f2*t.^2);
%subplot(4,2,1)
%plot(t,noise)
%axis([0  2  -2  2])
%/-------------------------------------------------------------------------

%-------------------------------------------------------------------------

primary=voice+noise;
subplot(4,2,1)
plot(t,primary)
title('primary = voice + noise   (input1)')
xlabel('time(seconds)')
ylabel('Amplitude')

%/*-------------------------------------------------------------------------

ref = noise;
%ref=noise+0.25*rand;     %noisy noise
subplot(4,2,2)
plot(t,primary - ref)
axis([0  2  -2  2])
title('primary - reference noise(voice + noise - noise)')
xlabel('time(seconds)')
ylabel('Amplitude')

%/*-------
```

```
%/*-----------------------*/

%noise=.1*rand(1,length(voice));                    %white noise
ref=noise+0.25*rand;
                                                    %noisy noise

subplot(4,2,3)
plot(t,ref)
axis([0  2  -2  2])
title('reference  (noisy noise)    (input)');
xlabel('time(seconds)')
ylabel('Amplitude')

w=zeros(order,1);
mu=1.3;
for i=1:N-order
   buffer = ref(i:i+order-1);
%current 32 points of reference
   desired(i) = primary(i)-buffer*w;                %dot product
reference and coeffs
   w=w+(buffer.*mu*desired(i)/norm(buffer))';%update coeffs
end

subplot(4,2,4)
plot(t(order+1:N),desired)
axis([0  2  -2  2])
title(' output of filter with mu=0.0%)')
title('output signal[ voice ]')
xlabel('time(seconds)')
ylabel('Amplitude')
legend('mu = 0.00%')


subplot(4,2,5)
plot(t(order+1:N),desired)
axis([0  2  -2  2])
title('output of  filter  with  mu  ')
xlabel('time(seconds)')
ylabel('Amplitude')
legend('mu=1.3')

%/*-----------------------*/

% -----------

noise =cos((2*pi*f2*t.^2+pi));
primary=voice+noise;
ref = noise;

subplot(4,2,6)
plot(t,primary,'r')
axis([0  2  -2  2])
```

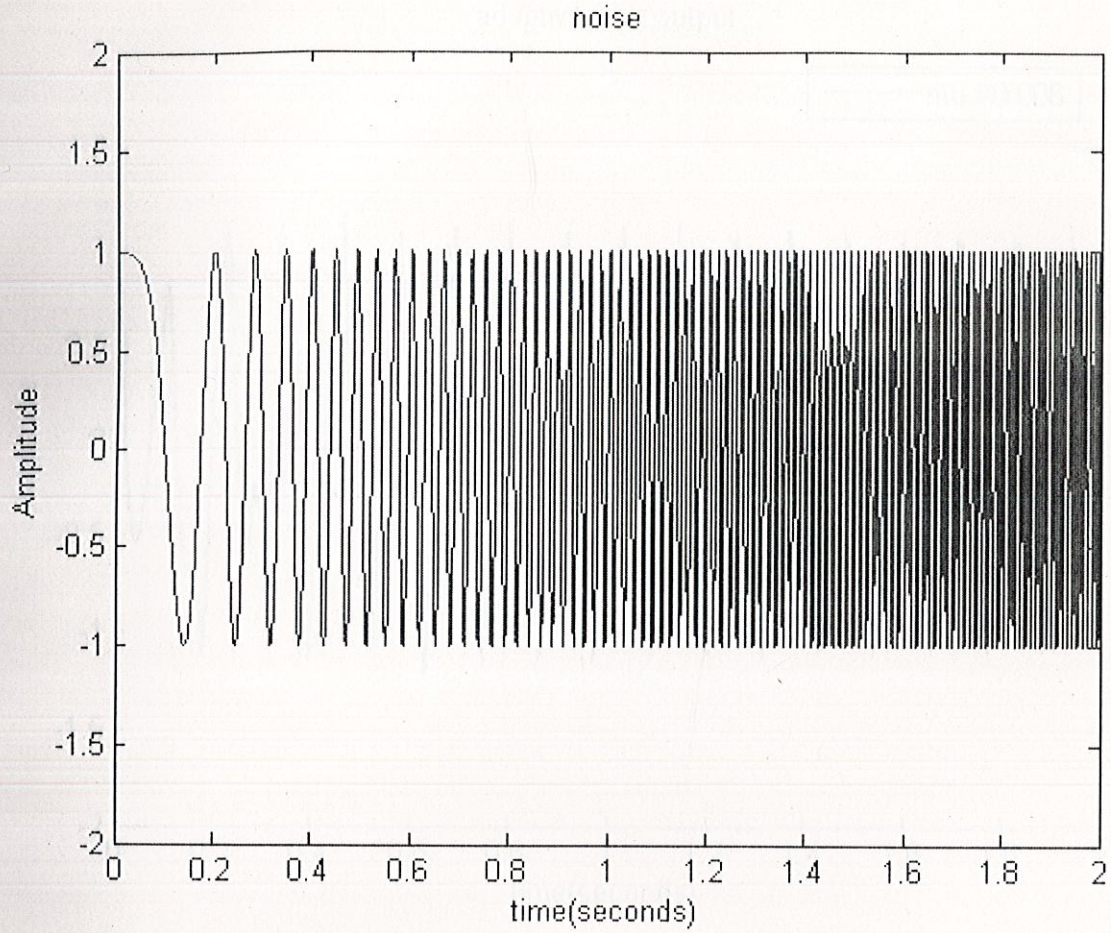
```
title('primary    reference  noise(ref noise  90 out of phase wi
noise)')
xlabel('time(seconds)')
ylabel('Amplitude')

w=zeros(order,1);
mu=0.006;
for i=1:N-order
   buffer = ref(i:i+order-1);
%current 32 points of reference
   desired(i) = primary(i)-buffer*w;                    %dot product
reference and coeffs
   w=w+(buffer.*mu*desired(i)/norm(buffer))';%update coeffs
end

subplot(4,2,7)
plot(t(order+1:N),desired)
%title(' output of filter with mu=0.006)')
title('adaptive filter output(ref noise 180 out of phase with  noise ')
xlabel('time(seconds)')
ylabel('Amplitude')
legend('mu =0.006')

%/*-------------------------------------------------------------------

%*/-------------------------------------------------------------------

%noise=.1*rand(1,length(voice));                  %white noise
noise1=.1*rand(1,length(voice));                  %white noise
ref = voice+noise1;


w=zeros(order,1);
mu=0.006;
for i=1:N-order
   buffer = ref(i:i+order-1);
%current 32 points of reference
   desired(i) = primary(i)-buffer*w;                    %dot product
reference and coeffs
   w=w+(buffer.*mu*desired(i)/norm(buffer))';%update coeffs
end

subplot(4,2,8)
plot(t(order+1:N),desired)
axis([0  2  -2  2])
%title('  output
title(
xlabel(
ylabel(
legend(
```

We begin with two signals, the primary signal and the reference signal. The primary signal contains both our desired voice signal and the noise signal from the car engine.



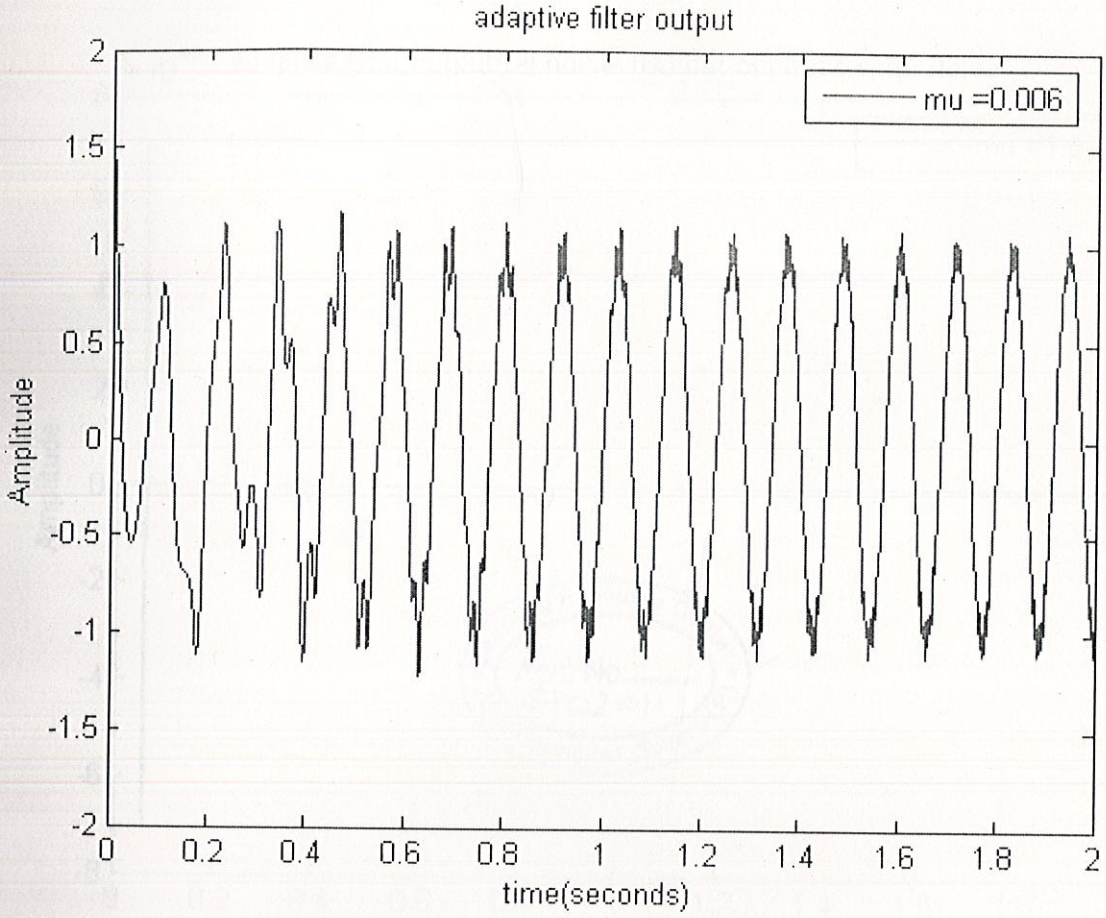
The reference signal is a tapped version of the noise in the primary signal, i.e. it must be correlated to the noise that we are trying to eliminate. In the case that we are trying to model, the primary signal may come from a microphone at the speaker's mouth which picks up both the speech signal and a noise signal from the car engine. The reference signal may come from another microphone that is placed away from the speaker and closer to the car engine, so the reference noise will be similar to the noise in primary but perhaps with a different phase and with some additional white noise added to it.

Reference: Estimation of Noise



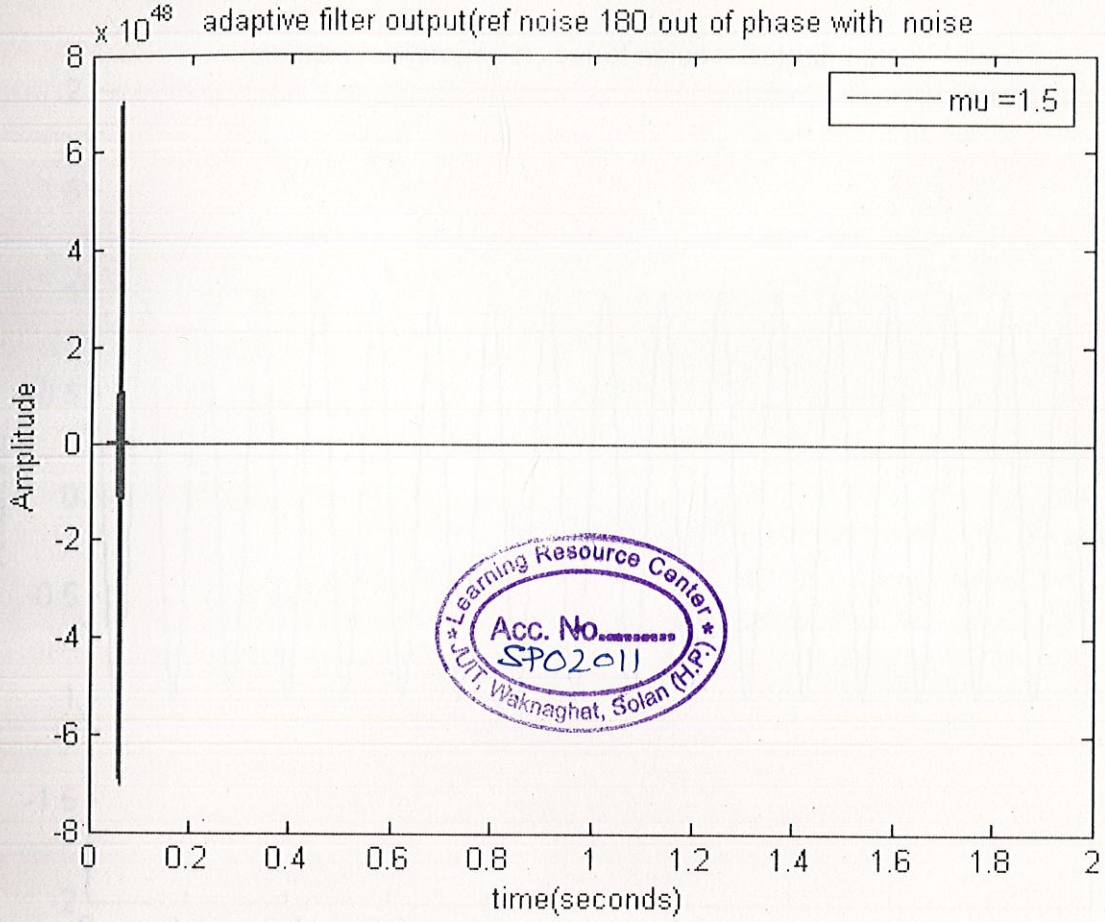
The LMS algotrithm updates the filter coefficients to minimize the error between the primary signal and the filtered noise. In the process of pouring through some Books DSP by Prokis & Manolokis ,adaptive filter theory by S.Haykin we came to the Ultimate program using matlab. the voice component of the primary signal is orthogonal to the reference noise.
Thus the minimum this error can be is just our desired voice signal.

Output: Voice



We then experimented with varying different parameters. It turns out that the output we get is very, very sensitive to mu. Apparently there is a very precise method for finding the most optimal mu, something to do with the eigenfuction of the correlation matrix between the primary and reference signals, but we used an educated trial and error technique. Basically we found that mu affects how fast a response we were able to get; a larger mu gives a faster response, but with a mu that is too large, the result will blow up.

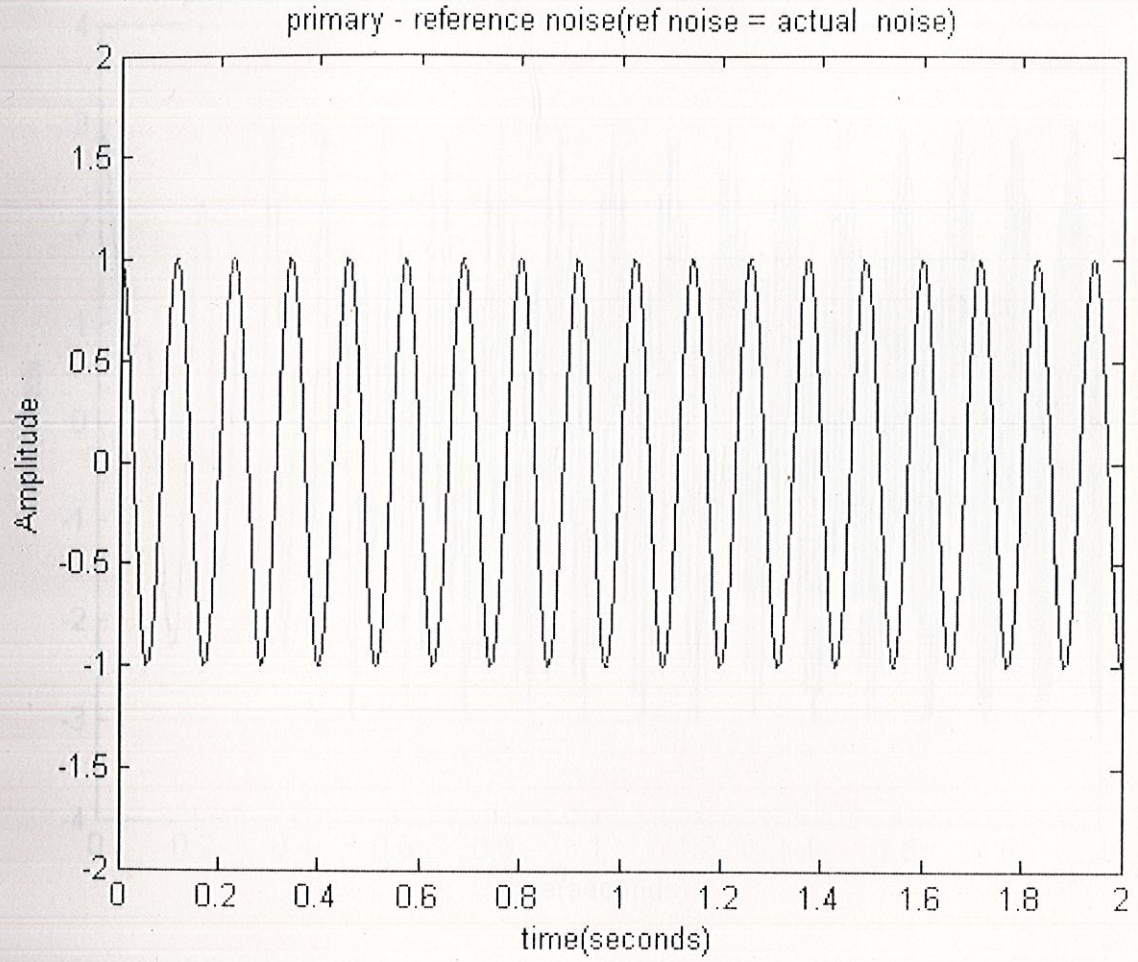
Result with Big mu



We also experimented with several different filter lengths.

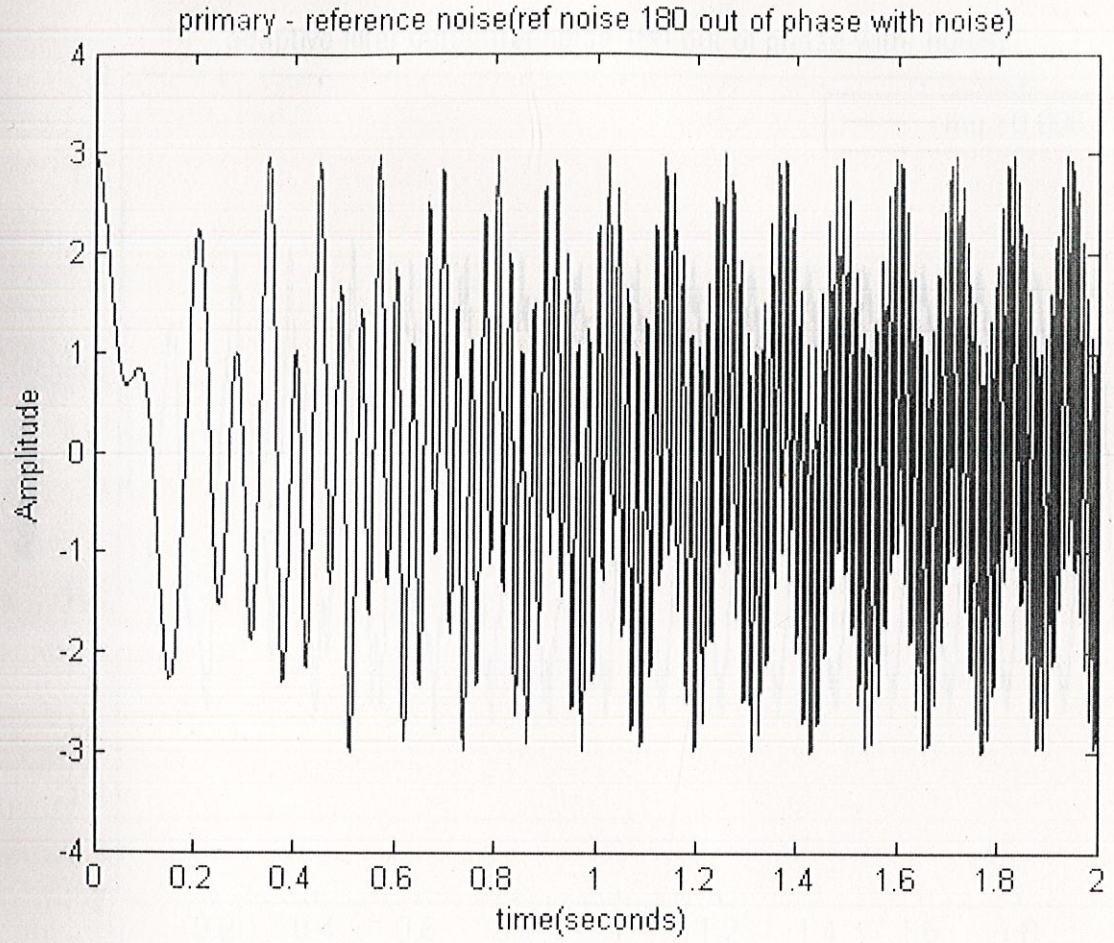
One question that is arises is why we cannot simply subtract the reference noise from the primary signal to obtain our desired voice signal. This method would work well if the reference noise was exactly the same as the actual noise in primary.

Reference Noise = Actual Noise

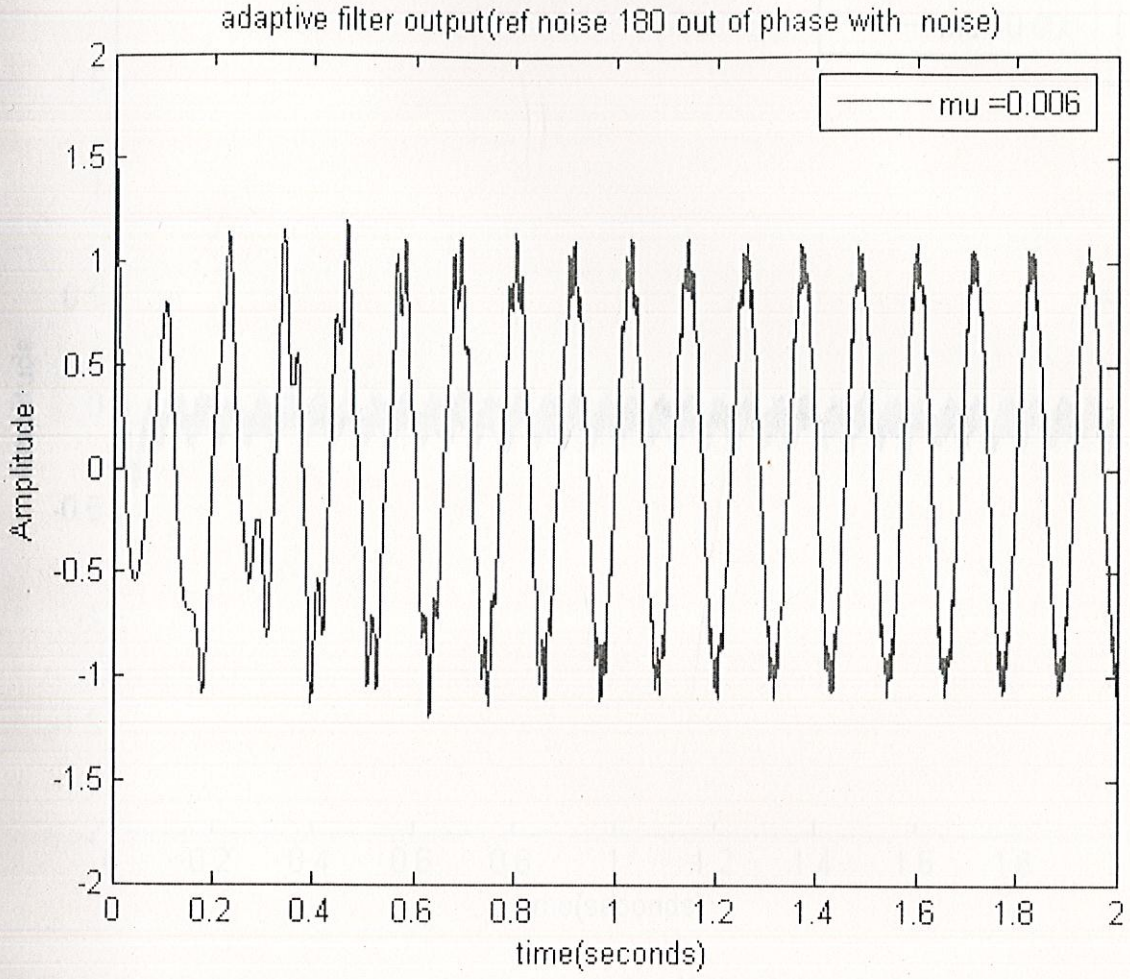


However, if the reference noise is exactly 180 degrees out of phase the with noise in primary, the noise will be doubled in the output. As can be seen in the figures below, the output from the adaptive filter is still able to sucessfully cancel out the noise.

Reference Noise 180 deg Out-of-Phase from Actual Noise

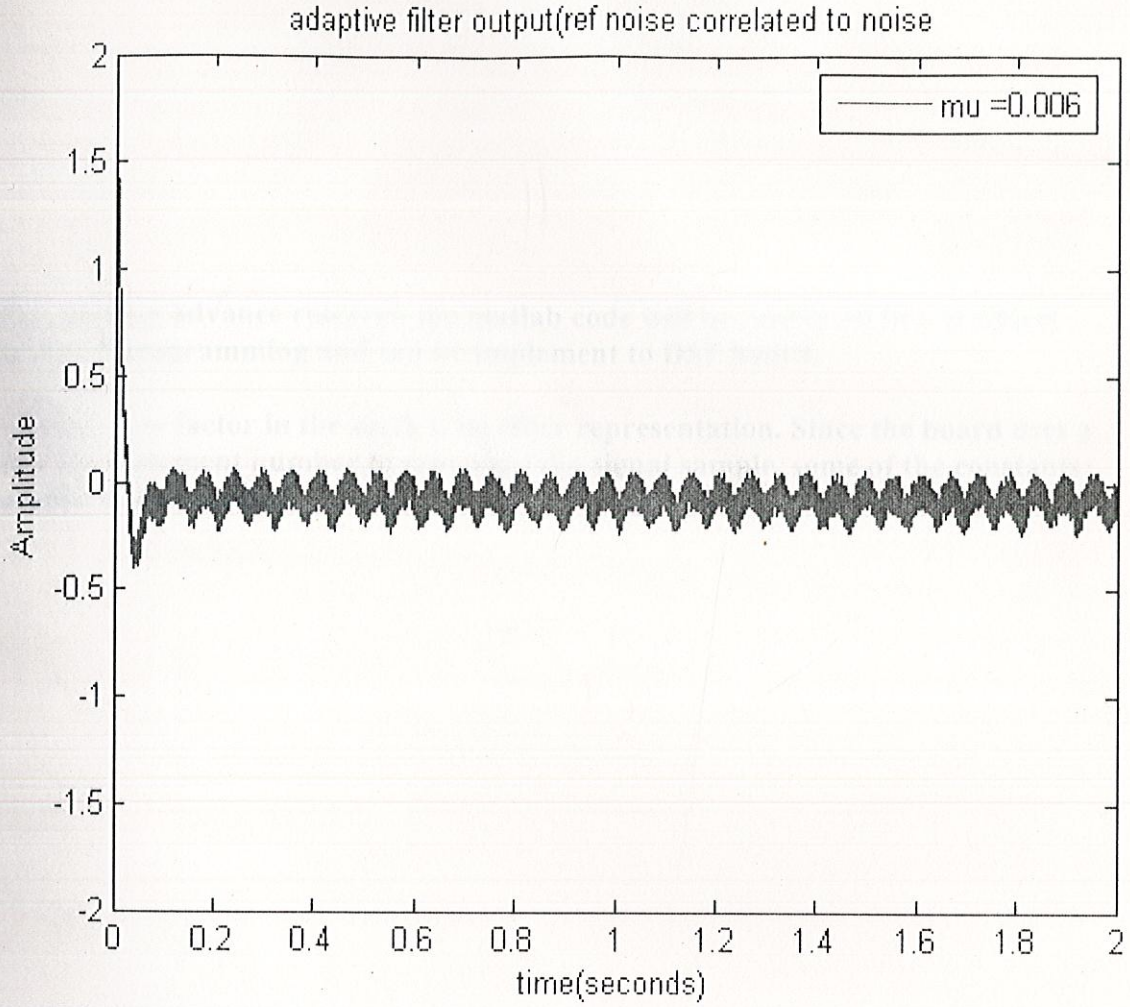


Filtered Output - It still works!



The one important condition on the use of adative filters for noise cancellation is that the noise can't be similar to the desired voice signal. If this is the case, the error that the filter is trying to minimize has the potential to go to zero, i.e. the filter also wipes out the voice signal. The figure below shows the output of the adaptive filter when the reference noise used was a sinusoid of the same frequency as that of the voice signal plus some white noise.

Filtered Output - No Good When Voice = Noise



FILTER IMPLEMENTATION

For further advance research the matlab code will be converted in c or object oriented programming and can be implement to DSP board.

Another key factor in the math is number representation. Since the board uses a two's complement number to represent the signal sample, some of the constants needed to be changed.

/*code to implement on dsp board

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <common.h>
#include <mcbspdrv.h>
#include <intr.h>
#include <board.h>
#include <codec.h>
#include <mcbsp.h>
#include <mathf.h>

#define PRINT_DBG 1
#define FILTERLENGTH 100


short buffer[FILTERLENGTH];
short w[FILTERLENGTH];

/* FILE LOCAL (STATIC) PROTOTYPES
*/

void hookint(void);
interrupt void serialPortRcvISR (void);


main()
{

        Mcbsp_dev dev;
        Mcbsp_config  mcbspConfig;
        int i;
        printf("Got to main!");

/* Initialize EVM


        for(i = 0; i < FILTERLENGTH; i++)
        {
                /* Initialize coefficients to 1's */
                w[i] = 1;
                /* Initialize buffer of previous inputs to 0's */
                buffer[i] = 0;
                /* printf("Wi: %d ",w[i]); */
        }

        evm_init();
```

```
/* Open MCBSP for subsequent Examples


        mcbsp_drv_init();

        dev = mcbsp_open(0);

        if (dev == NULL)
        {
                printf("Error opening MCBSP 0    \n  ");
                return(ERROR);
        }

/* configure McBSP


        memset(&mcbspConfig,0,sizeof(mcbspConfig));

        mcbspConfig.loopback                = FALSE;

        mcbspConfig.tx.update               = TRUE;
        mcbspConfig.tx.clock_mode           = CLK_MODE_EXT;
        mcbspConfig.tx.frame_length1        = 0;
        mcbspConfig.tx.word_length1         = WORD_LENGTH_32;

        mcbspConfig.rx.update               = TRUE;
        mcbspConfig.rx.clock_mode           = CLK_MODE_EXT;
        mcbspConfig.rx.frame_length1        = 0;
        mcbspConfig.rx.word_length1         = WORD_LENGTH_32;

        mcbsp_config(dev,&mcbspConfig);

        MCBSP_ENABLE(0, MCBSP_BOTH);

/* configure CODEC


        codec_init();

        /* link codec channels to audio inputs          */
        codec_adc_control(RIGHT,0.0,FALSE,LINE_SEL);
        codec_adc_control(LEFT,0.0,TRUE,MIC_SEL);

        /* mute (L/R)LINE input to mixer                */
        codec_line_in_control(LEFT,MIN_AUX_LINE_GAIN,TRUE);
        codec_line_in_control(RIGHT,MIN_AUX_LINE_GAIN,TRUE);


        /* D/A 0.0 dB atten, do not mute DAC outputs    */
        codec_dac_control(LEFT, 0.0, FALSE);
        codec_dac_control(RIGHT, 0.0, FALSE);
```

```
        /*
        sampleRate = 11025;
        actualrate = codec_change_sample_rate(sampleRate, TRUE);
        */

        codec_interrupt_enable();

        hookint();

/* Main Loop, wait for Interrupt


        while (1)
        {
        }

        /* mute DAC outputs (unused, as best we can tell)*/

        codec_dac_control(LEFT, 0.0, TRUE);
        codec_dac_control(RIGHT, 0.0, TRUE);


/* End Single Block Capture and Playback Example


        mcbsp_close(dev);

        return(OK);

}


/* FUNCTIONS


void hookint()
{
        intr_init();
        intr_map(CPU_INT15, ISN_RINT0);
        intr_hook(serialPortRcvISR, CPU_INT15);

        //INTR_ENABLE(1);
        INTR_ENABLE(15);
        INTR_GLOBAL_ENABLE();

        return;
```

```
}

/* Our Function - Where all the real work gets done */

interrupt void serialPortRcvISR (void)
{
        int i;
        int accum = 0;
        int sample_data;

        short chan1;
        short chan2;
        short primary;
        short ref;

        short next_output;
        short accumShort;

        /* read in left and right channels */

        sample_data = MCBSP_READ(0);

        /* Separate samples from left and right channels */
        chan1 = sample_data & 0xffff;
        chan2 = (sample_data & 0xffff0000) >> 16;

        primary = chan2;
        ref = chan1;


        /* Shift old buffered inputs right one position in array */
        for(i = FILTERLENGTH; i > 0; i--) {
                buffer[i] = buffer[i - 1];
                /*printf("b%d: %d",i,buffer[i]); */
        }


        /* Append new input value to front of buffer */
        buffer[0] = sample_data & 0xffff;


        /* Take dot product of buffered inputs and filter coefficients
*/
        for(i=0; i<FILTERLENGTH; i++)
        {
                accum += (buffer[i] * w[i]);
        }

        /* Right shift to convert to Q15 format (what the DSP wants) */
        accumShort = accum >> 16;

        /* Find "error" */
        next_output = primary - accumShort;
```

```
        /* Update coefficients using newest "error" calculation */
        for(i = 0; i < FILTERLENGTH; i++)
        {
                w[i] += (2 * buffer[i] * next_output)>>16;
        }

        /* Print to stdout if output sample gets too big */
        if(next_output > 32000)
                printf("\nAck! Out: %d",next_output);

        /* write out result to d/a converter*/
        /* multiply output by 2 to make it louder */
        MCBSP_WRITE(0, 2 * next_output);

        return;

}
```

CONCLUSIONS & FURTHER STUDY

Our implementation sucessfully achieved system idetification and noise cancellation. Specifically, time varying noise was reduced. The effects of varying different paramters in the algorithm were also observed. It was found that the stepsize mu affects the rate of convergence, a larger mu leads to faster convergence, but too large a mu can produce divergence. The order of the filter used afffects the distortion of the desired signal. Because of the adaptive filter updates its coefficients to minimize the error between the primary and reference siganls, we get poor performance if the desired signal is similar to the reference signal. This method of noise cancellation is most useful when the reference noise that we have access to is large or delayed relative to the noise that is actually in the primary signal.

The next area to explore would be to implement different algorithms for updating filter coefficients, for example the Normed Least Mean Squares algorithm was briefly experimented with. The system gave an acceptable output for white noise reduction, but its performance could be greatly improved. This noise was much more difficult to eliminate since there exists little correlation between the reference nosie and the noise in primary, given that they are both completely random. Finally, we can customize our system for various real world applications.

BIBLIOGRAPHY

Bellanger, M. ***Adaptive Digital Filters and Signal Analysis.*** **Marcel Dekker, Inc. 1987.**

Treichler, J., Johnson, C., Larimore, M. ***Theory and Design of Adaptive Filters.*** **John Wiley & Sons. 1987.**

Oppenheim, Schafer and Buck, *Discrete-Time Signal Processing*

Dgital Signal Processing , Proakis & Manolakis

B. Widrow and S.D. Stearns. (1985). *Adaptive Signal Processing*. [Good on applications, LMS]. Prentice-Hall.

C.F.N. Cowan and P.M. Grant. (1985). *Adaptive Filters*. [Good overview of lots of topics]. Prentice-Hall.

J.R. Treichler, C.R. Johnson and M.G. Larimore. (1987). *Theory and Design of Adaptive Filters*. [Good introduction to adaptive filtering, CMA; nice coverage of hardware]. Wiley-Interscience.

M.L. Honig and D.G. Messerschmidt. (1984). *Adaptive Filters: Structures, Algorithms, and Applications*. [Good coverage of lattice algorithms]. Kluwer.

S. Haykin. (1986). *Adaptive Filters Theory*. [Nice coverage of adaptive filter theory; Good reference]. Prentice-Hall.

WEBPAGES

http://cnx.org/content/m11829/latest/

http://www.cnel.ufl.edu/nsbook/chapter16_Getting_a_grip_on_adaptation.html

http://www.wiley.com/college/haykin/