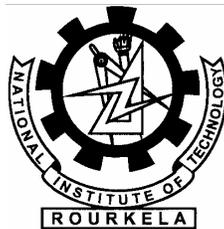


FINITE IMPULSE RESPONSE FILTER IMPLEMENTATION USING LMS ALGORITHM

A Thesis Submitted In Partial Fulfillment of the Requirements

For the Degree of

**Bachelor of Technology
In electrical engineering**



**DEPARTMENT OF ELECTRICAL ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY ROURKELA
2007**

**~~~Submitted
by:
SAMPRIT
10302064
ELECTRICAL
ENGINEERING**



**National Institute of Technology
Rourkela**

CERTIFICATE

This is to certify that the thesis entitled, “**FINITE IMPULSE RESPONSE FILTER IMPLEMENTATION USING LMS ALGORITHM** ” submitted by Mr. Samprit in partial fulfillment of the requirements of the award of Bachelor of Technology Degree in Electrical Engineering at the National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by them under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other university / institute for the award of any Degree or Diploma.

Date -

Mrs. K.R. SUBHASHINI
Dept. of Electrical Engg.
National Institute of Technology
Rourkela - 769008

A C K N O W L E D G E M E N T

I wish to express my profound sense of deepest gratitude to my guide and motivator Mrs. K. R. Subhashini, Lecturer, Electrical Engineering Department, National Institute of Technology, Rourkela for her valuable guidance, sympathy and co-operation and finally help for providing necessary facilities and sources during the entire period of this project.

I wish to convey my sincere gratitude to all the faculties of Electrical Engineering Department who have enlightened me during my studies. The facilities and co-operation received from the technical staff of Electrical Engg. Dept. is thankfully acknowledged.

I express my thanks to my friends who helped me lot in completing my project.

Last, but not least, I would like to thank the authors of various research articles and book that referred to.

**Samprit
10302064
National Institute
Of Technology
Rourkela**

Table of content

Certificate	i
Acknowledgement	ii
Abstract	vi
List of figures	vii
CHAPTER 1 Introduction	1
1.1 OBJECTIVE	2
1.2 ADAPTIVE FILTER	2
1.3 IDEAL LOW PASS FILTER	3
1.4 FIR (finite impulse response)	3
CHAPTER 2 CHARACTERISTICS OF FIR AND DISCUSSION	5
2.1 FIR LOW PASS FILTERS	6
2.2 DISCUSSION	6
2.3 FREQUENCY RESPONSE IN FIR/ Z TRANS FORM IN FIR	8
2.4 FREQUENCY RESPONSE FORMULA OF FIR	8
2.5 DC GAIN OF FIR	9
2.6 TERMS USED IN FIR	9
CHAPTER 3 : DESIGN AND PROPERTIES	10
3.1 FILTER DESIGN	11
3.2 SPECIAL TYPES OF FILTER	11
3.3 PROPERTIES	13
3.4 NUMERIC PROPERTIES	14
3.5 MULTIRATE (DECIMATING AND INTERPOLATING) SYSTEMS	
PROPERTIES	14
CHAPTER 4: IMPLEMENTATION	15
4.1 IMPLEMENTATION PRINCIPLE	16
4.2 TESTS FOR IMPLEMENTATION OF FIR	16
4.3 TRICKS OF IMPLEMENTING FIR	16
4.4 HOW FIR SORT OUT NEEDLESS CALCULATION IN	
IMPLEMENTATION	16
CHAPTER 5: LMS ALGORITHM	18

5.1 LEAST MEAN SQUARES ALGORITHM	19
5.2 IDEA	19
5.3 Simplifications	20
5.4 LMS algorithm summary	21
5.5 Normalized least mean squares filter (NLMS)	22
5.6 Optimal learning rate	22
5.7 Proof	22
5.8 Example: Plant Identification	24
CHAPTER 6: CHANNEL EQUALISATION	25
6.1 CHANNEL EQUALISATION	26
6.2 DISCRPTION ON CHANNEL EQUALISATION	26
6.3 DESCRIPTION OF THE RELATED ART	27
6.4 SUMMARY OF THE INVENTION	27
CHAPTER 7: SYSTEM IDENTIFICATION, BIT ERROR RATE	
AND SIGNAL TO NOISE RATIO	30
7.1SYSTEM IDENIFICATION	31
7.2BIT ERROR RATE	32
7.3SIGNAL TO NOISE RATIO	33
CHAPTER 8: MATLAB PROGAM, RESULT AND GRAPH	34
8.1PROGRAM 1 AND RESULT	35
8.2 PROGRAM 2 AND RESULT	36
8.3 PROGRAM 3 AND RESULT	38
8.4 PROGRAM 4 AND RESULT	40
8.5 PROGRAM 5 AND RESULT	43
8.6 PROGRAM 6 AND RESULT	47
CHAPTER 9 CONCLUSION AND REFERENCE	53
CONCLUSION	54
REFERENCE	54

ABSTRACT

The principle objective this project is to determine the coefficients of the FIR filters that met desired specifications. The determinations of coefficients involve channel equalization, system identification and SNR vs. BER plot using LMS algorithm. This project is to investigate the performance of an FIR filter equalizer for data transmission over a channel that causes intersymbol interference.

FIR filter removes unwanted parts of the signal, such as random noise, or extracts the useful parts of the signal, such as the components lying within a certain frequency range. In signal processing, there are many instances in which an input signal to a system contains extra unnecessary content or additional noise which can degrade the quality of the desired portion. In such cases we may remove or filter out the useless samples using FIR filters. Here by using LMS algorithm in channel equalization we determined coefficients in Matlab programming. In project by inducing white Gaussian signal or random signal (noise) with data signal we equalize for data transmission over a channel.

They can easily be designed to be "linear phase" (and usually are). Put simply, linear-phase filters delay the input signal, but don't distort its phase.

They are simple to implement. On most DSP microprocessors, the FIR calculation can be done by looping a single instruction.

They are suited to multi-rate applications. By multi-rate, we mean either "decimation" (reducing the sampling rate), "interpolation" (increasing the sampling rate), or both. Whether decimating or interpolating, the use of FIR filters allows some of the calculations to be omitted, thus providing an important computational efficiency. In contrast, if IIR filters are used, each output must be individually calculated, even if it that output will discard (so the feedback will be incorporated into the filter).

They have desirable numeric properties. In practice, all DSP filters must be implemented using "finite-precision" arithmetic, that is, a limited number of bits. The use of finite-precision arithmetic in IIR filters can cause significant problems due to the use of feedback, but FIR filters have no feedback, so they can usually be implemented using fewer bits, and the designer has fewer practical problems to solve related to non-ideal arithmetic.

In the implementation of FIR in system identification estimated channel parameters are almost same as channel parameter. So FIR filter provides effective way to remove unwanted signals, channel equalization and system identification.

LIST OF FIGURES

Fig-1.1 Adaptive filter diagram	2
Fig-1.2 block diagram of simple FIR	4
Fig3-showing typical deviations in low pass filters with FIR approximations	6
Fig 2.1: Tapped Delay Line fir Filter	8
Fig 5.1 showing adaptive filter adjustment using LMS algorithm	19
Fig. 5.2 Direct form adaptive FIR filter.	21
Fig 5.3 showing plant identification using LMS	24
Fig. 5.4 showing plant identification graph	24
Fig 6.1 showing application of adaptive filtering to adaptive Channel equalization.	28
Fig 6.2 showing channel equalization application	29
Fig 7.1 showing block diagram of system identification or system modeling problem	32
Fig 8.1 showing plot of the square and plot of MSE	42
Fig 8.2 system identification graph output	46
Fig 8.3 showing SNR vs. BER graph	52

Chapter 1

OBJECTIVE

INTRODUCTION

1.1 OBJECTIVE

Objective of this project is to determine the coefficients of the FIR filters that met desired specifications. The determinations of coefficients involve channel equalization, system identification and SNR vs. BER plot using LMS algorithm.

This project is to investigate the performance of an FIR filter equalizer for data transmission over a channel that causes intersymbol interference.

1.2 ADAPTIVE FILTER

An **adaptive filter** is a filter which *self-adjusts* its transfer function according to an optimizing algorithm. Because of the complexity of the optimizing algorithms, most adaptive filters are digital filters that perform digital signal processing and adapt their performance based on the input signal. By way of contrast, a non-adaptive filter has static filter coefficients (which collectively form the transfer function).

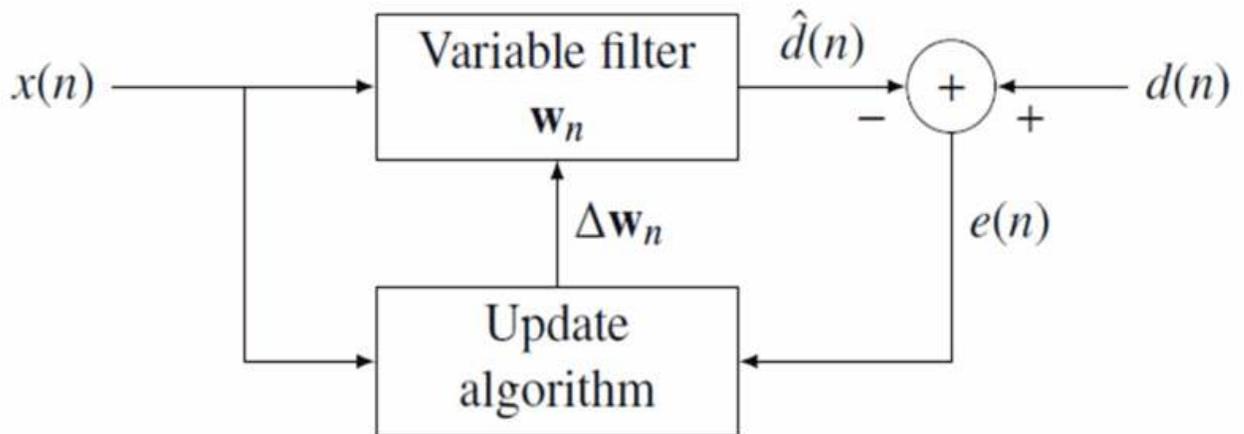


Fig-1.1 Adaptive filter diagram

1.3 IDEAL LOW PASS FILTER

The ideal low pass filter is one that allows through all frequency components of a signal below a designated cutoff frequency ω_c , and rejects all frequency components of a signal above ω_c .

Its frequency response satisfies

$$HLP(e^{j\omega}) = \begin{cases} 1, & 0 \leq \omega \leq \omega_c \\ 0, & \omega_c < \omega \leq \pi \end{cases}$$

The impulse response of the ideal low pass filter can easily be found to be $HLP[n] = \text{sinc}(\omega_c n) \pi n, -\infty < n < \infty$.

1.4 FIR (finite impulse response)

The impulse response is “finite” because there is no feedback in the filter; if you put in an impulse (that is, a single “1” sample followed by many “0” samples), zeroes will eventually come out after the “1” sample has made its way in the delay line past all the coefficients.

Finite impulse response (FIR) filters are the most popular type of filters implemented in software. This paper will help you understand them both on a theoretical and a practical level.

Filters are signal conditioners. Each functions by accepting an input signal, blocking prespecified frequency components, and passing the original signal minus those components to the output. For example, a typical phone line acts as a filter that limits frequencies to a range considerably smaller than the range of frequencies human beings can hear. That’s why listening to CD-quality music over the phone is not as pleasing to the ear as listening to it directly.

A *digital filter* takes a digital input, gives a digital output, and consists of digital components. In a typical digital filtering application, software running on a digital signal processor (DSP) reads input samples from an A/D converter, performs the mathematical manipulations dictated by theory for the required filter type, and outputs the result via a D/A converter.

An *analog filter*, by contrast, operates directly on the analog inputs and is built entirely with analog components, such as resistors, capacitors, and inductors.

There are many filter types, but the most common are lowpass, highpass, bandpass, and bandstop. A *low pass filter* allows only low frequency signals (below some specified cutoff) through to its output, so it can be used to eliminate high frequencies. A lowpass filter is handy, in that regard, for limiting the uppermost range of frequencies in an audio signal; it's the type of filter that a phone line resembles.

A *highpass filter* does just the opposite, by rejecting only frequency components below some threshold. An example highpass application is cutting out the audible 60Hz AC power "hum", which can be picked up as noise accompanying almost any signal in the U.S.

The designer of a cell phone or any other sort of wireless transmitter would typically place an analog *bandpass filter* in its output RF stage, to ensure that only output signals within its narrow, government-authorized range of the frequency spectrum are transmitted.

Engineers can use bandstop filters, which pass both low and high frequencies, to block a predefined range of frequencies in the middle.

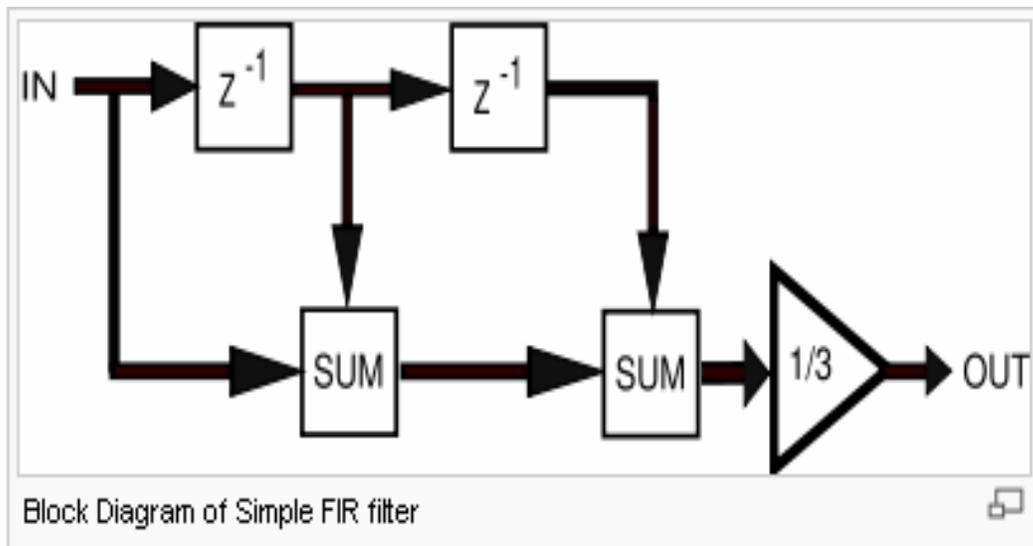


Fig-1.2 block diagram of simple FIR

Chapter 2

CHARACTERISTICS OF FIR DISCUSSION

2.1 FIR LOW PASS FILTERS

the pass band Because the impulse response required to implement the ideal low pass filter is infinitely long, it is impossible to design an ideal FIR low pass filter.

Finite length approximations to the ideal impulse response

Lead to the presence of ripples in both the pass band ($\omega < \omega_c$) and the stop band ($\omega > \omega_c$) of the filter, as well as to a nonzero transition width between and stop band of the filter.

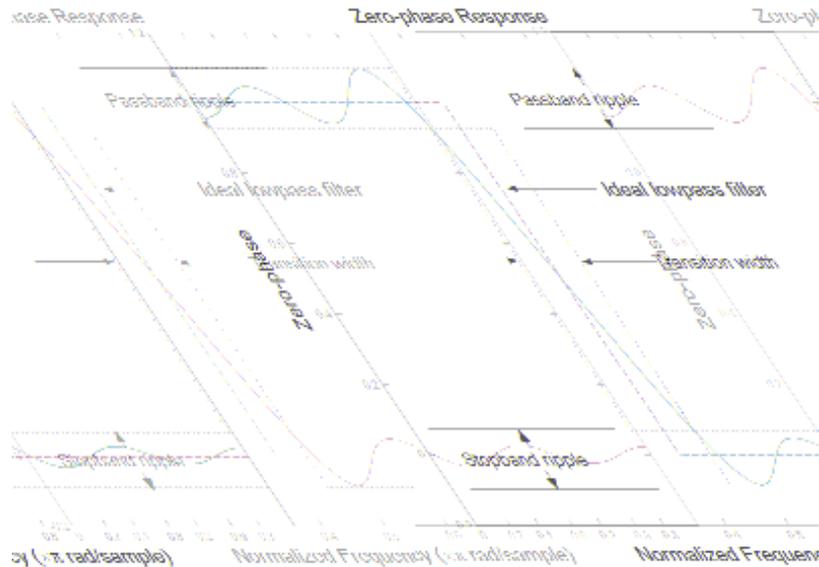


Fig3-showing typical deviations in low pass filters with FIR approximations

2.2 Discussion

We start the discussion by stating the difference equation which defines how the input signal is related to the output signal

$$y[n] = b_0x[n] + b_1x[n - 1] + \dots + b_Px[n - P]$$

Where P is the filter order, $x[n]$ is the input signal, $y[n]$ is the output signal and b_i are the filter coefficients. The previous equation can also be expressed

As

$$y[n] = \sum_{i=0}^P b_i x[n-i]$$

To find the impulse response we set

$$x[n] = \delta[n]$$

where $\delta[n]$ is the Kronecker delta impulse. The impulse response for an FIR filter follows as

$$\begin{aligned} h[n] &= \sum_{i=0}^P b_i \delta[n-i] \\ &= b_n \end{aligned}$$

The Z-transform of the impulse response yields the transfer function of the FIR filter

$$\begin{aligned} H(z) &= Z\{h[n]\} \\ &= \sum_{n=-\infty}^{\infty} h[n] z^{-n} \\ &= \sum_{i=0}^P b_i z^{-i} \end{aligned}$$

The transfer function allows us to judge whether or not a system is BIBO stable. To be specific the BIBO stability criterion requires all poles of the transfer function to have an absolute value smaller than one. In other words all poles must be located within a unit circle in the z -plane. To find the poles of the transfer function we have to extend it with $\frac{z^P}{z^P}$ and arrive at

$$H(z) = \frac{\sum_{i=0}^P b_i z^{P-i}}{z^P}$$

The FIR transfer function contains P poles at $z = 0$. Since all poles are at the origin, all poles are located within the unit circle of the z -plane; therefore all FIR filters are stable.

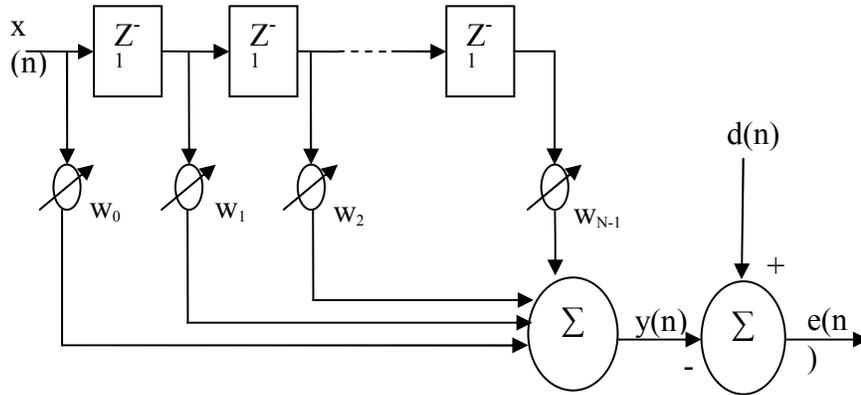


Fig 2.1: Tapped Delay Line fir Filter

2.3 FREQUENCY RESPONSE IN FIR/ Z TRANS FORM IN FIR

For an N-tap FIR filter with coefficients $h(k)$, whose output is described by:

$$y(n) = h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) + \dots + h(N-1)x(n-N+1),$$

the filter's Z transform is:

$$H(z) = h(0)z^0 + h(1)z^{-1} + h(2)z^{-2} + \dots + h(N-1)z^{-(N-1)}, \text{ or}$$

$$H(z) = \sum_{n=0}^{N-1} h(n)z^{-n}$$

2.4 FREQUENCY RESPONSE FORMULA OF FIR

The variable z in $H(z)$ is a continuous complex variable, and we can describe it as: $z = r e^{j\omega}$, where r is a magnitude and ω is the angle of z . If we let $r=1$, then $H(z)$ around the unit circle becomes the filter's frequency response $H(j\omega)$. This means that substituting $e^{j\omega}$ for z in $H(z)$ gives us an expression for the filter's frequency response $H(\omega)$, which is:

$$H(j\omega) = h(0)e^{-j0\omega} + h(1)e^{-j1\omega} + h(2)e^{-j2\omega} + \dots + h(N-1)e^{-j(N-1)\omega}, \text{ or}$$

Using Euler's identity, $e^{-ja} = \cos(a) - j\sin(a)$, we can write $H(\omega)$ in rectangular form as:

$$H(j\omega) = h(0)[\cos(0\omega) - j\sin(0\omega)] + h(1)[\cos(1\omega) - j\sin(1\omega)] + \dots + h(N-1)[\cos((N-1)\omega) - j\sin((N-1)\omega)], \text{ or}$$

$$H(j\omega) = \sum_{n=0}^{N-1} h(n)[\cos(n\omega) - j\sin(n\omega)]$$

2.5 DC GAIN OF FIR

Consider a DC (zero Hz) input signal consisting of samples which each have value 1.0. After the FIR's delay line had filled with the 1.0 samples, the output would be the sum of the coefficients. Therefore, the gain of a FIR filter at DC is simply the sum of the coefficients.

This intuitive result can be checked against the formula above. If we set ω to zero, the cosine term is always 1, and the sine term is always zero, so the frequency response becomes:

$$H(j\omega) = \sum_{n=0}^{N-1} h(n) \quad \text{even more.}$$

2.6 TERMS USED IN FIR:-

- *Impulse Response* - The "impulse response" of a FIR filter is actually just the set of FIR coefficients. (If you put an "impulse" into a FIR filter which consists of a "1" sample followed by many "0" samples, the output of the filter will be the set of coefficients, as the 1 sample moves past each coefficient in turn to form the output.)
- *Tap* - A FIR "tap" is simply a coefficient/delay pair. The number of FIR taps, (often designated as "N") is an indication of 1) the amount of memory required to implement the filter, 2) the number of calculations required, and 3) the amount of "filtering" the filter can do; in effect, more taps means more stop band attenuation, less ripple, narrower filters, etc.)
- *Multiply-Accumulate (MAC)* - In a FIR context, a "MAC" is the operation of multiplying a coefficient by the corresponding delayed data sample and accumulating the result. FIRs usually require one MAC per tap. Most DSP microprocessors implement the MAC operation in a single instruction cycle.

- *Transition Band* - The band of frequencies between pass band and stop band edges. The narrower the transition band, the more taps are required to implement the filter. (A "small" transition band results in a "sharp" filter.)
- *Delay Line* - The set of memory elements that implement the " Z^{-1} " delay elements of the FIR calculation.
- *Circular Buffer* - A special buffer which is "circular" because *incrementing* at the end causes it to wrap around to the beginning, or because *decrementing* from the beginning causes it to wrap around to the end. Circular buffers are often provided by DSP microprocessors to implement the "movement" of the samples through the FIR delay-line without having to literally move the data in memory. When a new sample is added to the buffer, it automatically replaces the oldest one.

Chapter 3

DESIGN

PROPERTIES

3.1 FILTER DESIGN

To design a filter means to select the coefficients such that the system has specific characteristics. The required characteristics are stated in filter specifications. Most of the time filter specifications refer to the frequency response of the filter.

The three most popular design methods are (in order):

- **Parks-McClellan:** The Parks-McClellan method (inaccurately called "Remez" by Matlab) is probably the most widely used FIR filter design method. It is an iteration algorithm that accepts filter specifications in terms of pass band and stop band frequencies, pass band ripple, and stop band attenuation. The fact that you can directly specify all the important filter parameters is what makes this method so popular. The PM method can design not only FIR "filters" but also FIR "differentiators" and FIR "Hilbert transformers".

Windowing: In the windowing method, an initial impulse response is derived by taking the Inverse Discrete Fourier Transform (IDFT) of the desired frequency response. Then, the impulse response is refined by applying a data window to it.

- **Direct Calculation:** The impulse responses of certain types of FIR filters (e.g. Raised Cosine and Windowed Sinc) can be calculated directly from formulas.

3.2 SPECIAL TYPES OF FILTER

Aside from "regular" and "extra crispy" there are:

- *Boxcar* - Boxcar FIR filters are simply filters in which each coefficient is 1.0. Therefore, for an N-tap boxcar, the output is just the sum of the past N samples. Because boxcar FIRs can be implemented using only adders, they are of interest primarily in hardware implementations, where multipliers are expensive to implement.
- *Hilbert Transformer* - Hilbert Transformers shift the phase of a signal by 90 degrees. They are used primarily for creating the imaginary part of a complex signal, given its real part.
- *Differentiator* - Differentiators have an amplitude response which is a linear function of frequency. They are not very popular nowadays, but are sometimes used for FM demodulators.
- *Lth-Band* - Also called "Nyquist" filters, these filters are a special class of filters used primarily in multirate applications. Their key selling point is that one of every L coefficients is zero--a fact which can be exploited to reduce the number of multiply-accumulate operations required to implement the filter. (The famous "half-band" filter is actually an Lth-band filter, with $L=2$.)

- *Raised-Cosine* - This is a special kind of filter that is sometimes used for digital data applications. (The frequency response in the pass band is a cosine shape which has been "raised" by a constant.)

3.3 PROPERTIES

A FIR filter has a number of useful properties which sometimes make it preferable to an infinite impulse response filter. FIR filters:

- Are inherently stable. This is due to the fact that all the poles are located at the origin and thus are located within the unit circle.
- Require no feedback. This means that any rounding errors are not compounded by summed iterations. The same relative error occurs in each calculation.
- Can have linear phase

3.3 a) LINEAR PHASE IN FIR:-

Most FIR is linear-phase filters; when a linear-phase filter is desired, a FIR is usually used.

"Linear Phase" refers to the condition where the phase response of the filter is a linear (straight-line) function of frequency (excluding phase wraps at +/- 180 degrees). This results in the *delay* through the filter being the same at all frequencies. Therefore, the filter does not cause "phase distortion" or "delay distortion". The lack of phase/delay distortion can be a critical advantage of FIR filters over IIR and analog filters in certain systems, for example, in digital data modems.

3.3 b) DELAY IN LINEAR PHASE

The formula is simple: given a FIR filter which has N taps, the delay is: $(N - 1) / (2 * F_s)$, where F_s are the sampling frequency. So, for example, a 21 tap linear-phase FIR filter operating at a 1 kHz rate has delay: $(21 - 1) / (2 * 1 \text{ kHz}) = 10$ milliseconds.

3.3 c) ALTERNATIVE TO LINEAR PHASE

Non-linear phase - Actually, the most popular alternative is "minimum phase". Minimum-phase filters (which might better be called "minimum delay" filters) have less delay than linear-phase filters with the same amplitude response, at the cost of a non-linear phase characteristic, a.k.a. "phase distortion".

A low pass FIR filter has its largest-magnitude coefficients in the center of the impulse response. In comparison, the largest-magnitude coefficients of a minimum-phase filter are nearer to the beginning.

3.4 NUMERIC PROPERTIES:-

Again, the key is the lack of feedback. The numeric errors that occur when implementing FIR filters in computer arithmetic occur separately with each calculation; the FIR doesn't "remember" its past numeric errors. In contrast, the feedback aspect of IIR filters can cause numeric errors to compound with each calculation, as numeric errors are fed back.

The practical impact of this is that FIRs can generally be implemented using fewer bits of precision than IIRs. For example, FIRs can usually be implemented with 16 bits, but IIRs generally require 32 bits, or even more.

3.5 MULTIRATE (DECIMATING AND INTERPOLATING) SYSTEMS PROPERTIES

because only a fraction of the calculations that would be required to implement a decimating or interpolating FIR in a literal way actually needs to be done.

Since FIR filters do not use feedback, only those outputs which are actually going to be used have to be calculated. Therefore, in the case of decimating FIRs (in which only 1 of N outputs will be used), the other N-1 outputs don't have to be calculated. Similarly, for interpolating filters (in which zeroes are inserted between the input samples to raise the sampling rate) you don't actually have to multiply the inserted zeroes with their corresponding FIR coefficients and sum the result; you just omit the multiplication-additions that are associated with the zeroes (because they don't change the result anyway.)

In contrast, since IIR filters use feedback, every input must be used, and every input must be calculated because all inputs and outputs contribute to the feedback in the filter.

Chapter 4

IMPLEMENTATION

4.1 IMPLEMENTATION PRINCIPLE:-

Structurally, FIR filters consist of just two things: a sample delay line and a set of coefficients. To implement the filter:

1. Put the input sample into the delay line.
2. Multiply each sample in the delay line by the corresponding coefficient and accumulate the result.
3. Shift the delay line by one sample to make room for the next input sample.

4.2 TEST FOR IMPLEMENTATION OF FIR

Here are a few methods:

- **Impulse Test:** A very simple and effective test is to put an impulse into it (which is just a "1" sample followed by at least $N - 1$ zeroes.) You can also put in an "impulse train", with the "1" samples spaced at least N samples apart. If all the coefficients of the filter come out in the proper order, there is a good chance your filter is working correctly. (You might want to test with non-linear phase coefficients so you can see the order they come out.) We recommend you do this test whenever you write a new FIR filter routine.
- **Step Test:** Input N or more "1" samples. The output after N samples, should be the sum (DC gain) of the FIR filter.
- **Sine Test:** Input a sine wave at one or more frequencies and see if the output sine has the expected amplitude.
- **Swept FM Test:** *From Eric Jacobsen:* "My favorite test after an impulse train is to take two identical instances of the filter under test, use them as I and Q filters and put a complex FM linear sweep through them from DC to $F_s/2$. You can do an FFT on the result and see the complete frequency response of the filter, make sure the phase is nice and continuous everywhere, and match the response to what you'd expect from the coefficient set, the precision, etc."

4.3 TRICKS OF IMPLEMENTING FIR

FIR tricks center on two things

- 1) No need to calculate things that don't need to be calculated, and
- 2) "Faking" circular buffers in software.

4.4 HOW FIR SORT OUT NEEDLESS CALCULATION IN IMPLEMENTATION

First, if your filter has zero-valued coefficients, you don't actually have to calculate those taps; you can leave them out. A common case of this is "half-band" filter, which have the property that every-other coefficient is zero.

Second, if your filter is "symmetric" (linear phase), you can "pre-add" the samples which will be multiplied by the same coefficient value, prior to doing the multiply. Since this technique essentially trades an add for a multiply, it isn't really useful in DSP microprocessors which can do a multiply in a single instruction cycle. However, it is useful in ASIC implementations (in which addition is usually much less expensive than multiplication); also, some newer DSP processors now offer special hardware and instructions to make use of this trick.

Chapter 5

LMS ALGORITHM

5.1 LEAST MEAN SQUARES ALGORITHM

Least mean squares (LMS) algorithms are used in adaptive filters to find the filter coefficients that relate to producing the least mean squares of the error signal (difference between the desired and the actual signal). It is a stochastic gradient descent method in that the filter is only adapted based on the error at the current time.

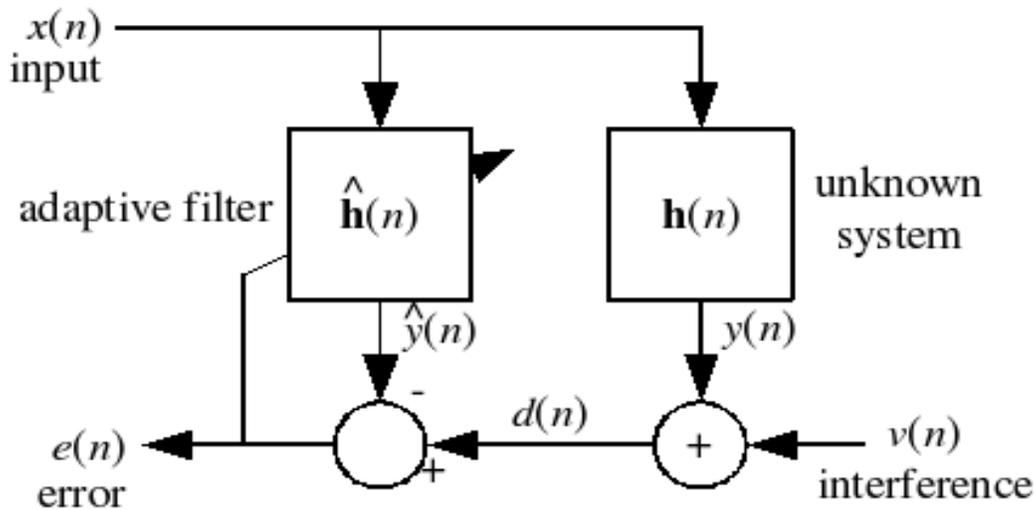


Fig 5.1 showing adaptive filter adjustment using LMS algorithm

Most linear adaptive filtering problems can be formulated using the block diagram above. That is, an unknown system $\mathbf{h}(n)$ is to be identified and the adaptive filter attempts to adapt the filter $\hat{\mathbf{h}}(n)$ to make it as close as possible to $\mathbf{h}(n)$, while using only observable signals $x(n)$, $d(n)$ and $e(n)$ ($y(n)$, $v(n)$ and $h(n)$ are not directly observable).

5.2 Idea

The idea behind LMS filters is to use the method of steepest descent to find a coefficient vector $\mathbf{h}(n)$ which minimizes a cost function. We start the discussion by defining the cost function as

$$C(n) = E \{ |e(n)|^2 \}$$

where $e(n)$ is defined in the block diagram section of the general adaptive filter and $E\{.\}$ denotes the expected value. Applying the steepest descent method means to take the partial derivatives with respect to the individual entries of the filter coefficient vector

$$\nabla C(n) = \nabla E \{e(n) e^*(n)\} = E \{\nabla e(n) e^*(n)\}$$

where ∇ is the gradient operator and with $\nabla e(n) = -\mathbf{x}(n)$ follows

$$\nabla C(n) = -E \{\mathbf{x}(n) e^*(n)\}$$

Now, $\nabla C(n)$ is a vector which points towards the steepest ascent of the cost function. To find the minimum of the cost function we need to take a step in the opposite direction of $\nabla C(n)$. To express that in mathematical terms

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) - \mu \nabla C(n) = \hat{\mathbf{h}}(n) + \mu E \{\mathbf{x}(n) e^*(n)\}$$

where μ is the step size. That means we have found a sequential update algorithm which minimizes the cost function. Unfortunately, this algorithm is not realizable until we know $E \{\mathbf{x}(n) e^*(n)\}$.

5.3 Simplifications

For most systems the expectation function $E \{\mathbf{x}(n) e^*(n)\}$ must be approximated. This can be done with the following unbiased estimator

$$\hat{E} \{\mathbf{x}(n) e^*(n)\} = \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{x}(n-i) e^*(n-i)$$

where N indicates the number of samples we use for that estimate. The simplest case is $N = 1$

$$\hat{E} \{\mathbf{x}(n) e^*(n)\} = \mathbf{x}(n) e^*(n)$$

For that simple case the update algorithm follows as

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \mu \mathbf{x}(n) e^*(n)$$

Indeed this constitutes the update algorithm for the LMS filter.

5.4 LMS algorithm summary

The LMS algorithm for a p th order algorithm can be summarized as

Parameters: $p =$ filter order

$\mu =$ step size

Initialization: $\hat{\mathbf{h}}(0) = \mathbf{0}$

Computation: For $n = 0, 1, 2, \dots$

$$\mathbf{x}(n) = [x(n), x(n-1), \dots, x(n-p+1)]^T$$

$$e(n) = d(n) - \hat{\mathbf{h}}^H(n)\mathbf{x}(n)$$

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \mu e^*(n)\mathbf{x}(n)$$

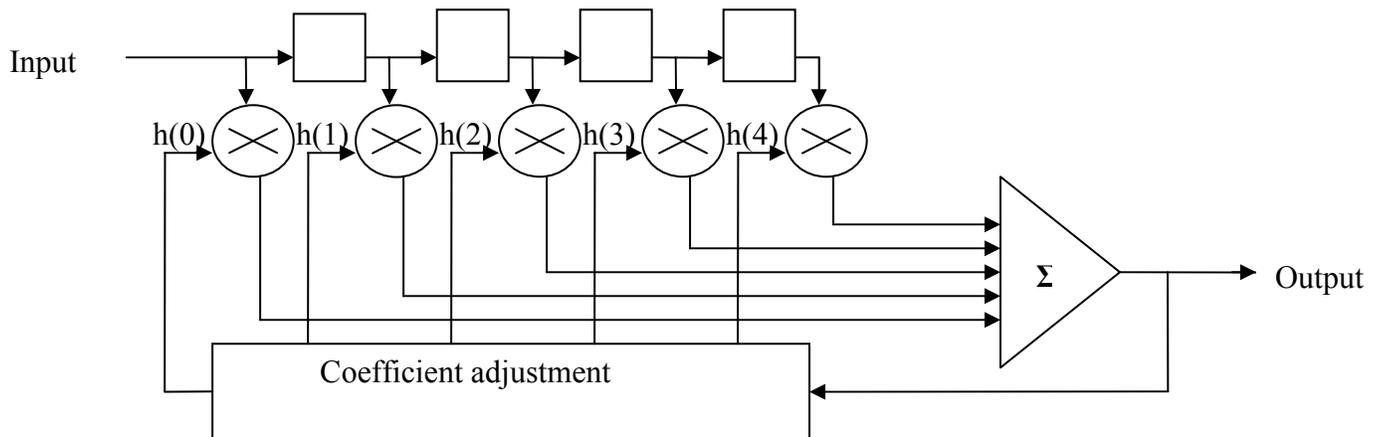


Fig. 5.2 Direct form adaptive FIR filter. Blank box above shows z^{-1} delay.

5.5 Normalized least mean squares filter (NLMS)

The main drawback of the "pure" LMS algorithm is that it is sensitive to the scaling of its input $x(n)$. This makes it very hard (if not impossible) to choose a learning rate μ that guarantees stability of the algorithm. The *Normalised least mean squares filter* (NLMS) is a variant of the LMS algorithm that solves this problem by normalising with the power of the input. The NLMS algorithm can be summarised as:

Parameters: $p = \text{filter order}$
 $\mu = \text{step size}$

Initialisation: $\hat{\mathbf{h}}(0) = \mathbf{0}$

Computation: For $n = 0, 1, 2, \dots$

$$\mathbf{x}(n) = [x(n), x(n-1), \dots, x(n-p)]^T$$

$$e(n) = d(n) - \hat{\mathbf{h}}^H(n)\mathbf{x}(n)$$

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \frac{\mu e^*(n)\mathbf{x}(n)}{\mathbf{x}^H(n)\mathbf{x}(n)}$$

5.6 Optimal learning rate

It can be shown that if there is no interference ($v(n) = 0$), then the optimal learning rate for the NLMS algorithm is

$$\mu_{opt} = 1$$

and is independent of the input $x(n)$ and the real (unknown) impulse response $\mathbf{h}(n)$. In the general case with interference ($v(n) \neq 0$), the optimal learning rate is

$$\mu_{opt} = \frac{E[|y(n) - \hat{y}(n)|^2]}{E[|e(n)|^2]}$$

The results above assume that the signals $v(n)$ and $x(n)$ are uncorrelated to each other, which is generally the case in practice.

5.7 Proof

Let the filter misalignment be defined as $\Lambda(n) = \left| \mathbf{h}(n) - \hat{\mathbf{h}}(n) \right|^2$, we can derive the expected misalignment for the next sample as:

$$E[\Lambda(n+1)] = E \left[\left| \hat{\mathbf{h}}(n) + \frac{\mu e^*(n) \mathbf{x}(n)}{\mathbf{x}^H(n) \mathbf{x}(n)} - \mathbf{h}(n) \right|^2 \right]$$

$$E[\Lambda(n+1)] = E \left[\left| \hat{\mathbf{h}}(n) + \frac{\mu (v^*(n) + y^*(n) - \hat{y}^*(n)) \mathbf{x}(n)}{\mathbf{x}^H(n) \mathbf{x}(n)} - \mathbf{h}(n) \right|^2 \right]$$

Let $\delta = \hat{\mathbf{h}}(n) - \mathbf{h}(n)$ and $r(n) = \hat{y}(n) - y(n)$

$$E[\Lambda(n+1)] = E \left[\left| \delta(n) - \frac{\mu (v(n) + r(n)) \mathbf{x}(n)}{\mathbf{x}^H(n) \mathbf{x}(n)} \right|^2 \right]$$

$$E[\Lambda(n+1)] = E \left[\left(\delta(n) - \frac{\mu (v(n) + r(n)) \mathbf{x}(n)}{\mathbf{x}^H(n) \mathbf{x}(n)} \right)^H \left(\delta(n) - \frac{\mu (v(n) + r(n)) \mathbf{x}(n)}{\mathbf{x}^H(n) \mathbf{x}(n)} \right) \right]$$

Assuming independence, we have:

$$E[\Lambda(n+1)] = \Lambda(n) + E \left[\left(\frac{\mu (v(n) - r(n)) \mathbf{x}(n)}{\mathbf{x}^H(n) \mathbf{x}(n)} \right)^H \left(\frac{\mu (v(n) - r(n)) \mathbf{x}(n)}{\mathbf{x}^H(n) \mathbf{x}(n)} \right) \right] - 2E \left[\frac{\mu |r(n)|^2}{\mathbf{x}^H(n) \mathbf{x}(n)} \right]$$

$$E[\Lambda(n+1)] = \Lambda(n) + \frac{\mu^2 E[|e(n)|^2]}{\mathbf{x}^H(n) \mathbf{x}(n)} - \frac{2\mu E[|r(n)|^2]}{\mathbf{x}^H(n) \mathbf{x}(n)}$$

$$\frac{dE[\Lambda(n+1)]}{d\mu} = 0$$

The optimal learning rate is found at $\frac{dE[\Lambda(n+1)]}{d\mu} = 0$, which leads to:

$$2\mu E[|e(n)|^2] - 2E[|r(n)|^2] = 0$$

$$\mu = \frac{E[|r(n)|^2]}{E[|e(n)|^2]}$$

5.8 Example: Plant Identification

The goal for a plant identification structure is to match the properties of an unknown system (plant) with an adaptive filter. The following figure shows the block diagram of a plant identification system. The Matlab source code is PlantIdent.

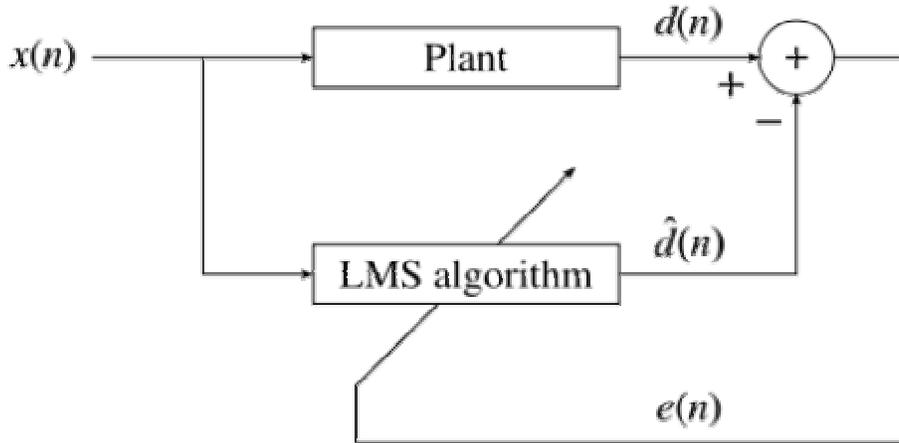


Fig 5.3 showing plant identification using LMS

In general any adaptive filter can be used for plant identification, however in this example we use an LMS structure. This example allows us to discuss the merits of the step size factor μ . The following figure shows the error signal for a near optimal value of μ . Clearly the error is minimized as the algorithm progresses.

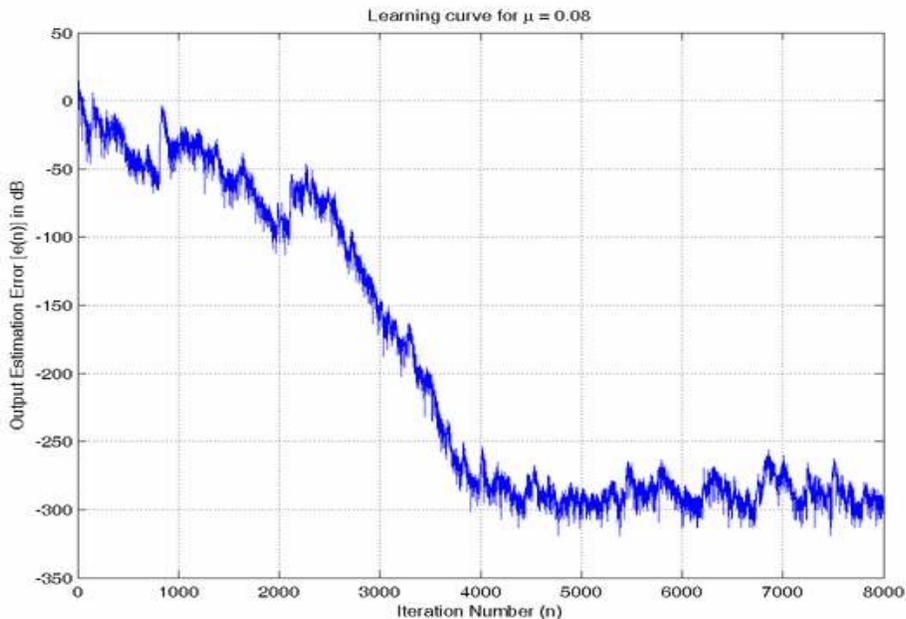


Fig. 5.4 showing plant identification graph

Chapter 6

CHANNEL EQUALISATION

6.1 CHANNEL EQUALISATION:-

The speed of data transmission over telephone channels is usually limited by channel distortion that causes intersymbol interference (ISI). An adaptive equalizer is employed in the modem to compensate for the channel distortion and thus allow for highly reliable high speed data transmission. The adaptive equalizers is basically an adaptive filter FIR filter with coefficients that are adjusted by means of the LMS algorithm to correct channel distortion, the process known as channel equalization.

A method and apparatus are provided for channel equalization with a digital finite impulse response (DFIR) filter using a pseudo random sequence. A read back signal of a pseudo random bit sequence is obtained. Samples are obtained from the read back signal of the pseudo random bit sequence. Tap gradients are calculated responsive to the obtained samples. The tap weights of the digital finite impulse response (FIR) filter are modified responsive to the calculated tap gradients. Debit samples and error samples are obtained from the read back signal of the pseudo random bit sequence and applied to a tap gradients calculator. Tap gradients are calculated by a bitwise multiplier and accumulation tap gradient calculation circuit. An attenuation function attenuates the calculated tap gradients by a programmable attenuation value.

6.2 DISCRPTION ON CHANNEL EQUALISATION

6.2 a)FIELD OF THE INVENTION

the present invention relates generally to the data processing field, and more particularly, relates to a method and apparatus for channel equalization with a digital finite impulse response (DFIR) filter using a pseudo random sequence in a direct access storage device (DASD) data channel.

6.3 DESCRIPTION OF THE RELATED ART

Direct access storage devices (DASDs) often incorporating stacked, commonly rotated rigid magnetic disks are used for storage of data in magnetic form on the disk surfaces. Data is recorded in concentric, radially spaced data information tracks arrayed on the surfaces of the disks. Transducer heads driven in a path toward and away from the drive axis write data to the disks and read data from the disks. Typically servo information is provided on one or more disk surfaces for reading by the transducer heads for accurately and reliably positioning transducer heads on the disk surfaces at a specific location to read and write data.

In today's disk drives the read back signal typically is equalized using a finite impulse response (FIR) filter. There are numerous algorithms used to find the FIR tap weights that will result in giving the optimum channel equalization. A need exists for an improved method for channel equalization with a digital FIR filter. In current DASD data

channels, channel equalization typically is achieved using extensive software algorithms. The known software algorithms are very time consuming, both during manufacture and use of the disk drive.

In earlier generations of DASD data channels, a common method of equalization used a method to reduce the Mean Square Error (MSE) of the equalized samples. Earlier designs suffered accuracy from noisy read back signals inherent in the channel.

A need exists for an improved method and apparatus for channel equalization with digital finite impulse response (DFIR) filter. A need exists for an improved equalization method that will eliminate the need for extensive software programs, as well as save time during manufacturing programming.

6.4 SUMMARY OF THE INVENTION

A principal object of the present invention is to provide a method and apparatus for channel equalization with a digital finite impulse response (DFIR) filter using a pseudo random sequence. Other important objects of the present invention are to provide such method and apparatus for channel equalization with a digital finite impulse response (DFIR) filter using a pseudo random sequence substantially without negative effect and that overcome many of the disadvantages of prior art arrangements.

In brief, a method and apparatus are provided for channel equalization with a digital finite impulse response (DFIR) filter using a pseudo random sequence. A read back signal of a pseudo random bit sequence is obtained. Samples are obtained from the read back signal of the pseudo random bit sequence. Tap gradients are calculated responsive to the obtained samples. The tap weights of the digital finite impulse response (DFIR) filter are modified responsive to the calculated tap gradients.

In accordance with features of the invention, dibit samples and error samples are obtained from the read back signal of the pseudo random bit sequence and applied to a tap gradients calculator. Tap gradients are calculated by a bitwise multiplier and accumulation tap gradient calculation circuit. An attenuation function attenuates the calculated tap gradients by a programmable attenuation value.

In channel equalization we use LMS algorithm,

$$h(\text{new}) = h(\text{old}) + \Delta * e * x(n)$$

Where $h(\text{new})$ = new weight coefficient

Δ = step size

e = error = $d(n) - y(n)$ = desired signal – output signal.

$X(n)$ = input signal.

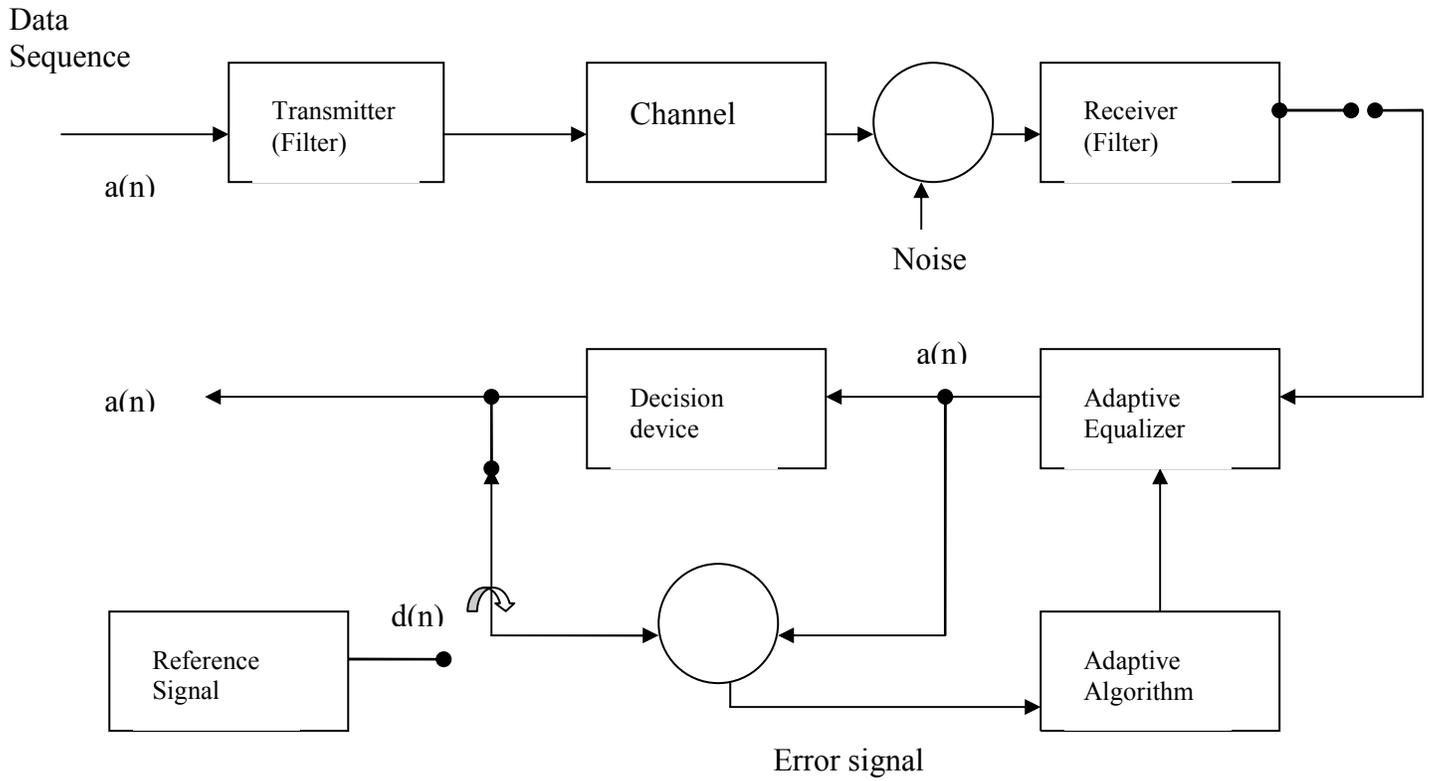


fig 6.1 showing application of adaptive filtering to adaptive channel equalization.

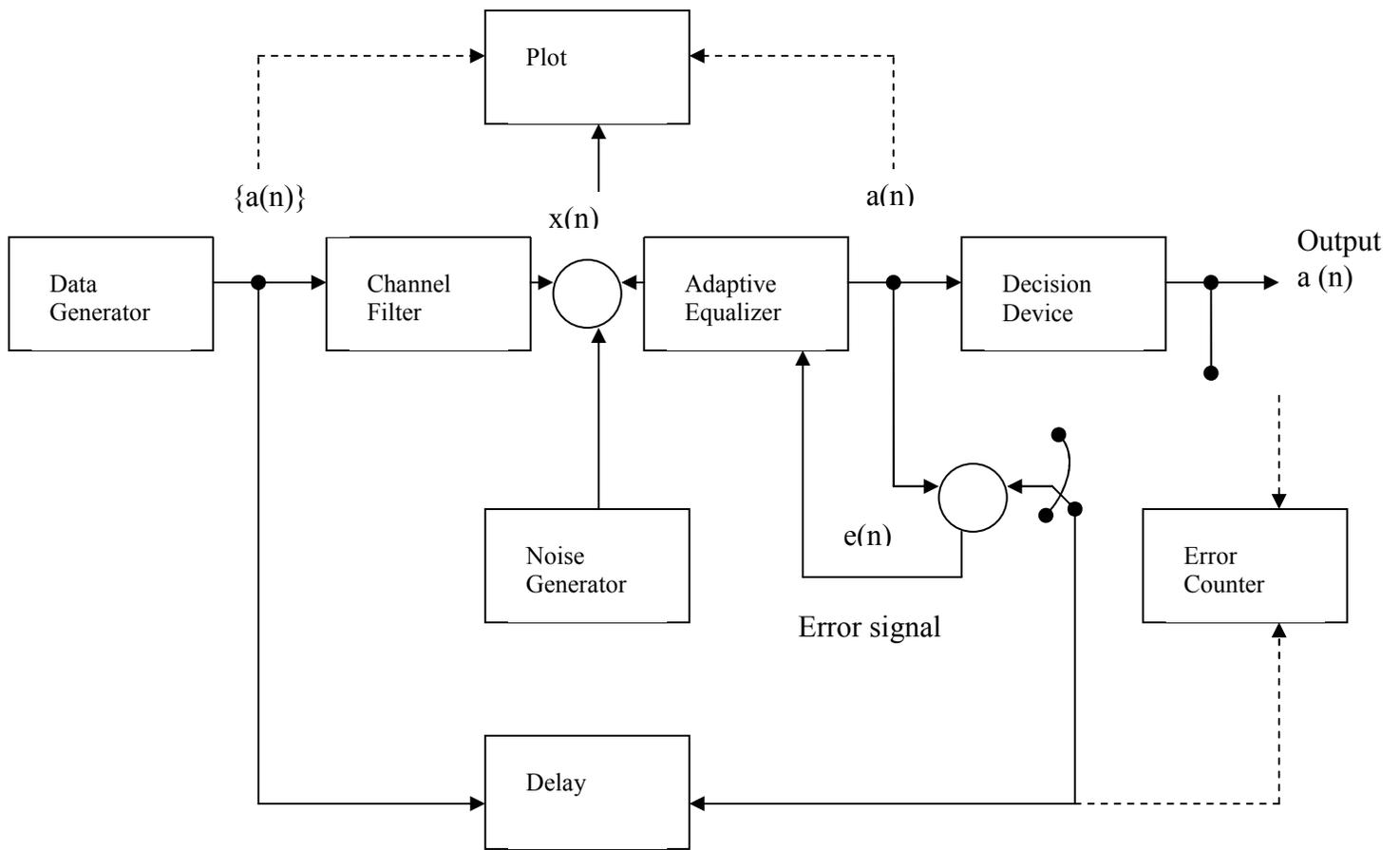


Fig 6.2 showing channel equalization application.

Chapter 7

SYSTEM IDENTIFICATION

BIT ERROR RATE

SIGNAL TO NOISE RATIO

7.1 SYSTEM IDENTIFICATION

System identification is a general term to describe mathematical tools and algorithms that build dynamical models from measured data. A dynamical model in this context is a mathematical description of the dynamic behavior of a system or process. Examples include:

- physical processes such as the movement of a falling body under the influence of gravity
- Economical processes such as stock markets that react to external influences.

One could build a so-called white-box model based on first principles, e.g. a model for a physical process from the Newton equations, but in many cases such models will be overly complex and possibly even impossible to obtain in reasonable time due to the complex nature of many systems and processes.

A much more common approach is therefore to start from measurements of the behavior of the system and the external influences (inputs to the system) and try to determine a mathematical relation between them without going into the details of what is actually happening inside the system. This approach is called system identification. Two types of models are common in the field of system identification:

- **Grey box model:** although the peculiarities of what is going on inside the system are not entirely known, a certain model is already available. This model does however still have a number of unknown free parameters which can be estimated using system identification.
- **Black box model:** No prior model is available. Most system identification algorithms are of this type.

System identification can be done in either the time or frequency domain.

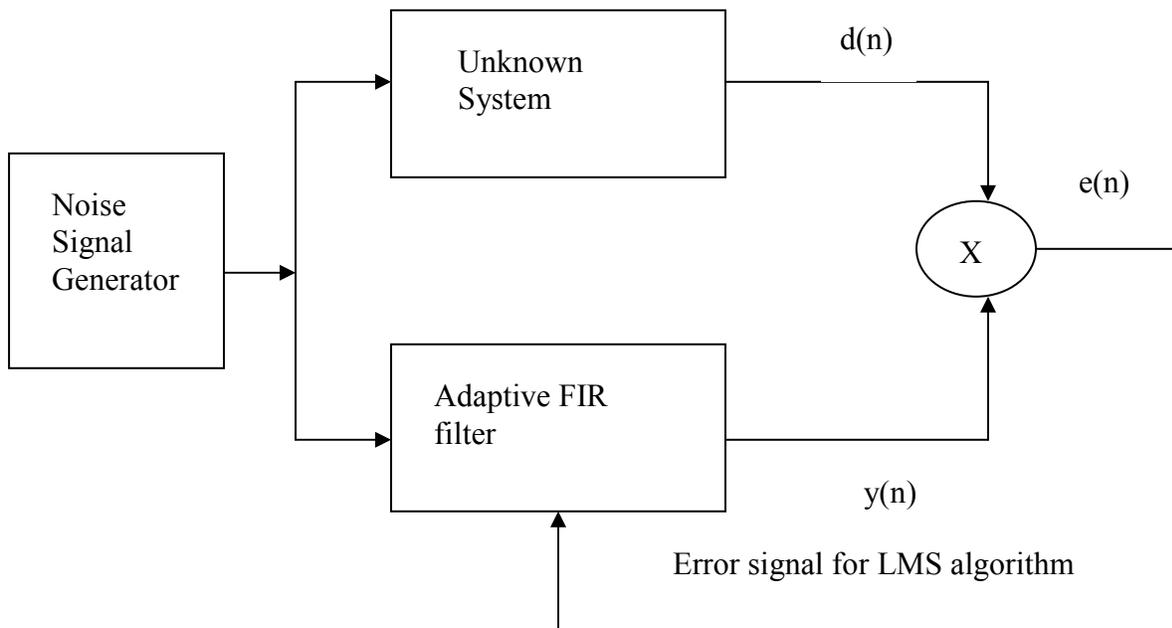


Fig 7.1 showing block diagram of system identification or system modeling problem

7.2 BIT ERROR RATE

In telecommunication, an **error ratio** is the ratio of the number of bits, elements, characters, or blocks incorrectly received to the total number of bits, elements, characters, or blocks sent during a specified time interval. The error ratio is usually expressed in scientific notation; for example, 2.5 erroneous bits out of 100,000 bits transmitted would be 2.5 out of 10^5 or 2.5×10^{-5} . Some software may display this as "2.5e-05".

The most commonly encountered ratio is the **bit error ratio** (BER) - also sometimes referred to as **bit error rate**.

For a given communication system, the bit error ratio will be affected by both the data transmission rate and the signal power margin.

Examples of bit error ratio are (a) transmission BER, *i.e.*, the number of erroneous bits received divided by the total number of bits transmitted; and (b) information BER, *i.e.*, the number of erroneous decoded (corrected) bits divided by the total number of decoded (corrected) bits.

On good connections the BER should be below 10^{-10}

The test time t can be calculated using Gaussian error distribution to:

$$t = -\frac{\ln(1 - c)}{b * r}$$

Where c is the degree of confidence level, b = upper bound of BER and r = bit rate. See also T-carrier

People usually plot the BER curves to describe the functionality of a digital communication system. In optical communication, BER (dB) vs. Received Power (dBm) is usually used; while in wireless communication, BER(dB) vs. SNR(dB) is used. The scale of Y axis (for BER) is usually in probability scale, so that the curve looks like straight line. (Examples of such figures needed, as well as the simple BER models)

Curve fitting for such BER curve is an interesting topic, attracting many research efforts.

7.3 SIGNAL TO NOISE RATIO

Signal-to-noise ratio is an engineering term for the power ratio between a signal (meaningful information) and the background noise:

$$\text{SNR} = \frac{P_{\text{signal}}}{P_{\text{noise}}} = \left(\frac{A_{\text{signal}}}{A_{\text{noise}}} \right)^2$$

Where P is average power and A is RMS amplitude. Both signal and noise power (or amplitude) must be measured at the same or equivalent points in a system, and within the same system bandwidth.

Because many signals have a very wide dynamic range, SNRs are usually expressed in terms of the logarithmic decibel scale. In decibels, the SNR is, by definition, 10 times the logarithm of the power ratio. If the signal and the noise is measured across the same impedance then the SNR can be obtained by calculating 20 times the base-10 logarithm of the amplitude ratio:

$$\text{SNR(dB)} = 10 \log_{10} \left(\frac{P_{\text{signal}}}{P_{\text{noise}}} \right) = 20 \log_{10} \left(\frac{A_{\text{signal}}}{A_{\text{noise}}} \right)$$

Chapter 8

MATLAB PROGRAM

RESULTS AND GRAPH

MATLAB PROGRAM OF FIR

8.1 PROGRAM 1

% Program to implement a FIR filter by Window technique.

% simple fir program for n no. of co-efficient.

Clear all;

clc;

% Program Code:

m=input('Enter the number of filter co-efficient: ');

x=input('Enter the input sample in matrix form: ');

z=input('Enter the input multiple form: ');

n=length(x);

a=0.95;

p=0: m-1;

b=a.^p;

xs=zeros(1,m);

xx=[x zeros(1,m-1)];

y=zeros(1, n+m-1);

for i=1:n+m-1

 xs(1)=xx(i);

 y(i)=sum(z.*xs.*b);

 for j=m:-1:2

 xs(j)=xs(j-1);

 end

end

y

8.1 a) EXAMPLE RESULT

Enter the number of filter co-efficient: [1 2 3 4 5 7 8 9 10 11 12]

Enter the input sample in matrix form: [2 3 5 7 8 9 20 12 3 4 6 9]

y =Columns 1 through 9

2 3 5 7 8 9 20 12 3

Columns 10 through 12

4 6 9 >>

8.2 PROGRAM 2

% Program by putting variable tap weights multiple factor
% Program to implement a FIR filter by Window technique.

Clear all;
clc;

% Program Code:

```
w=input('Enter the filter tap-weights (co-efficient): ');  
x=input('Enter the input sample in matrix form: ');  
a=input('Enter the input multiple factor form: ');  
n=length(x);  
m=length(w);
```

```
xs=zeros(1,m);  
xm=[x zeros(1,m-1)];  
y=zeros(1,n+m-1);
```

```
for i=1:n+m-1  
    xs(1)=xm(i);  
    y(i)=sum(a.*xs.*w);  
    for j=m:-1:2  
        xs(j)=xs(j-1);  
    end  
end
```

y

8.2 a) EXAMPLE RESULT

Enter the filter tap-weights (co-efficient): [1 2 3 4 5 6 7 8 9 10 11 23 45 67 98]
Enter the input sample in matrix form: [9 8 7 6 5 4 3 21 1 12 56 78 9 1 23]
Enter the input multiple factor form: [1 2 3 4 5 6 7 8 9 10 11 21 32 43 54]

y =

Columns 1 through 6

9 44 120 250 445 714

Columns 7 through 12

1064 1519 2101 2823 3753 8076

Columns 13 through 18

20877 45572 89483 80906 72774 65102

Columns 19 through 24

64348 75357 100109 170232 161230 343928

Columns 25 through 29

537296 451254 83629 71555 121716

>>

8.3 PROGRAM 3

% Program to implement a FIR filter by Window technique.

```
clear all;  
clc;
```

% Program Code:

```
w=input('Enter the filter tap-weights (co-efficients): ');  
x=input('Enter the input sample in matrix form: ');
```

```
n=length(x);  
m=length(w);
```

```
xs=zeros(1,m);  
xm=[x zeros(1,m-1)];  
y=zeros(1,n+m-1);
```

```
for i=1:n+m-1  
    xs(1)=xm(i);  
    y(i)=sum(xs.*w);  
    for j=m:-1:2  
        xs(j)=xs(j-1);  
    end  
end
```

```
y
```

8.3 a) EXAMPLE RESULT

Enter the filter tap-weights (co-efficients): [1 2 4 5 6 7 8 9 10 11]

Enter the input sample in matrix form: [1 2 3 8 9 4 5 7 2 7]

y =

Columns 1 through 9

1 4 11 27 53 88 129 172 218

Columns 10 through 18

273 311 341 350 296 222 199 160 92

Column 19

77

8.4 PROGRAM 4

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHANNEL EQUALIZATION USING LMS-ALGORITHM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all ;
close all ;
clc ;
rand ( 'state' , 0 ) ;
randn('state',0);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHANNELS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%ch = [0.4472, 0.8944];      % CH = 1
%ch = [0.2014, 0.9586, 0.2014];    % CH = 2
%ch = [0.2600, 0.9300, 0.2600];    % CH = 3
%ch = [0.3040, 0.9029, 0.3040];    % CH = 4
ch = [0.3410, 0.8760, 0.3410];    % CH = 5
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n = 2000 ;      % NUMBER OF SAMPLES

plant = ch;

snr = 30;

mu = 0.06 ;

N = 8 ;      % ORDER OF THE EQUALIZER

eq = zeros ( 1 , N ) ; % EQUALIZER

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% RANDOM SIGNAL = s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
s = hardlims ( rand ( 1 , n ) - 0.5 ) ; % SIGNAL

r1 = conv ( s , plant ) ;      % CONVOLUTION / CHANNEL OUTPUT

r = r1 ( 1 : n ) ;      % FOR SYMMETRY

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GENERATION OF THE WHITE GAUSSIAN NOISE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

    sp = 1 ;                                % SIGNAL POWER
    np = sp * 10 ^ ( - snr / 10 ) ;         % NOISE POWER
    noise = sqrt ( np ) * randn ( 1 , n ) ;
    x = r + noise ;                          % NOISY CHANNEL OUTPUT / INPUT TO THE
EQUALIZER

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DESIRED SIGNAL = d
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
delay = round ( N / 2 ) ;
d = [ zeros ( 1 , delay ) , s ] ;
xslide = zeros ( 1 , N ) ;
z = [] ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LMS - ALGORITHM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
y1=[];
for j = 1 : n

    xslide ( 2 : N ) = xslide ( 1 : N - 1 ) ;

    xslide ( 1 ) = x ( j ) ;

    y = xslide * eq' ;

    e = d(j) - y ;

    se(j) = e ^ 2 ;

    eq = eq + mu * e * xslide ;

    y1=[y1,y];

end

z = [ z , se ] ;
subplot(211) ; plot(se) ; title(' plot of the error square ') ; grid on ;
nmse = z / max(z);
subplot(212); plot(10*log10(nmse),'g'); title(' plot of MSE '); grid on;
clc
eq

```

8.4 a) RESULT

eq =

Columns 1 through 5

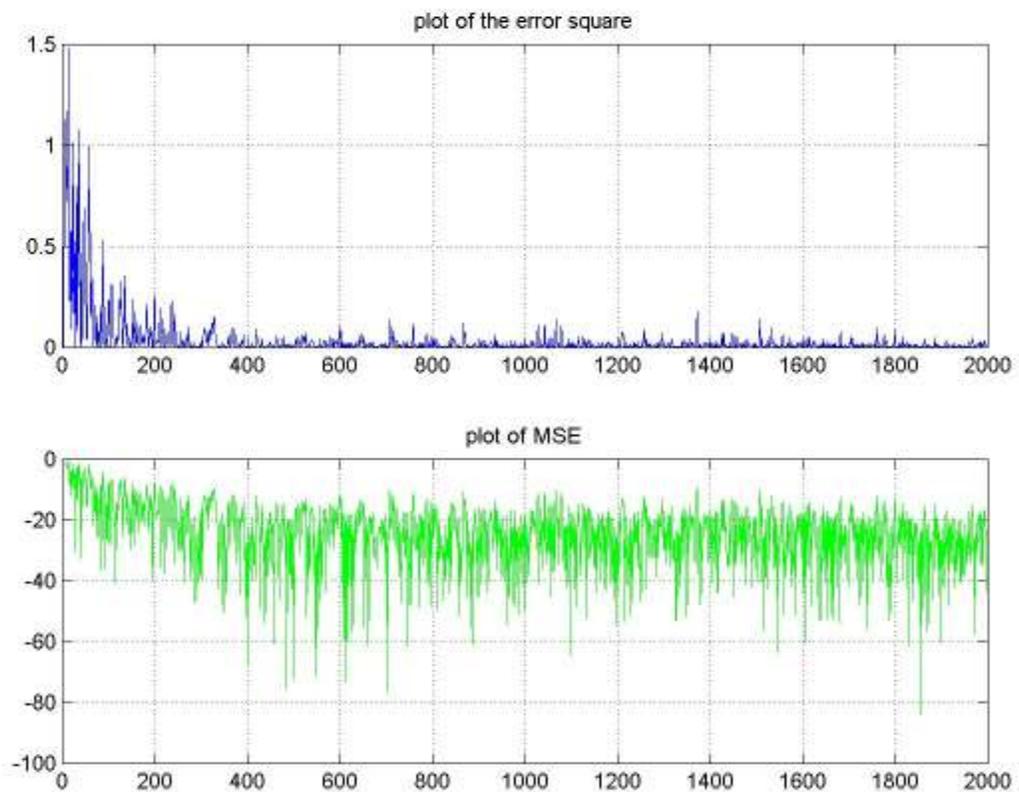
-0.1151 0.3693 -0.8293 1.7296 -0.8067

Columns 6 through 8

0.3874 -0.1861 0.0606

>>

Fig 8.1 showing plot of the square and plot of MSE



8.5 PROGRAM 5

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SYSTEM IDENTIFICATION USING LMS-ALGORITHM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear
clc
rand ( 'state' , 0 )
randn ( 'state' , 0 )

%ch = [0.4472, 0.8944];           % CH = 1
%ch = [0.2014, 0.9586, 0.2014]; % CH = 2
%ch = [0.2600, 0.9300, 0.2600]; % CH = 3
ch = [0.3040, 0.9029, 0.3040];  % CH = 4
%ch = [0.3410, 0.8760, 0.3410];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n = 3000 ;

plant = ch ;

snr = 10 ;

mu = 0.007 ;

N = length(plant) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% generation of the random signal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
z = [] ;
for i = 1 : 20

    s = (rand( 1 , n ) - 0.5)*sqrt(12) ;           % signal

    r1 = conv ( s , plant ) ;                     % convolution

    r = r1 ( 1 : n ) ;                            % for symmetry
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% generation of the (white gaussian noise / white noise) and addition with the
convolved output

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

sp = var(s); % signal power
np = sp * 10 ^ ( - snr / 10 ); % noise power
% noise = sqrt ( np ) * randn( 1 , n ); % white gaussian noise generation
noise = sqrt ( np ) * (rand( 1 , n)-0.5)*sqrt(12) ; % white uniform noise
x = r + noise ;

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

d = x ;

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

w = zeros ( 1 , N ); % intializing weights
xslide = zeros ( 1 , N );
se = [] ;

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% lms algorithm

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
y1 =[];

```

```

for j = 1 : n

```

```

xslide ( 2 : N ) = xslide ( 1 : N - 1 ) ;

xslide ( 1 ) = s ( j ) ;

y = xslide * w';

y1 = [ y1 y ] ;

e = d(j) - y ;

se( j ) = e ^ 2 ;

w = w + mu * e * xslide ;

end

z = [ z ; se ] ;

end

mse = mean( z ) ;
nmse = mse/ max(mse);
plot(10*log10(nmse))
grid on

%%%%%%%%%%
% show the results
%%%%%%%%%%
clc ;

disp ( ' channel parameters : ' )
disp ( plant )

disp ( ' estimated parameters : ' )
disp ( w )

```

8.5 a) RESULT

channel parameters :

0.3040 0.9029 0.3040

estimated parameters :

0.2932 0.9045 0.3168

>>

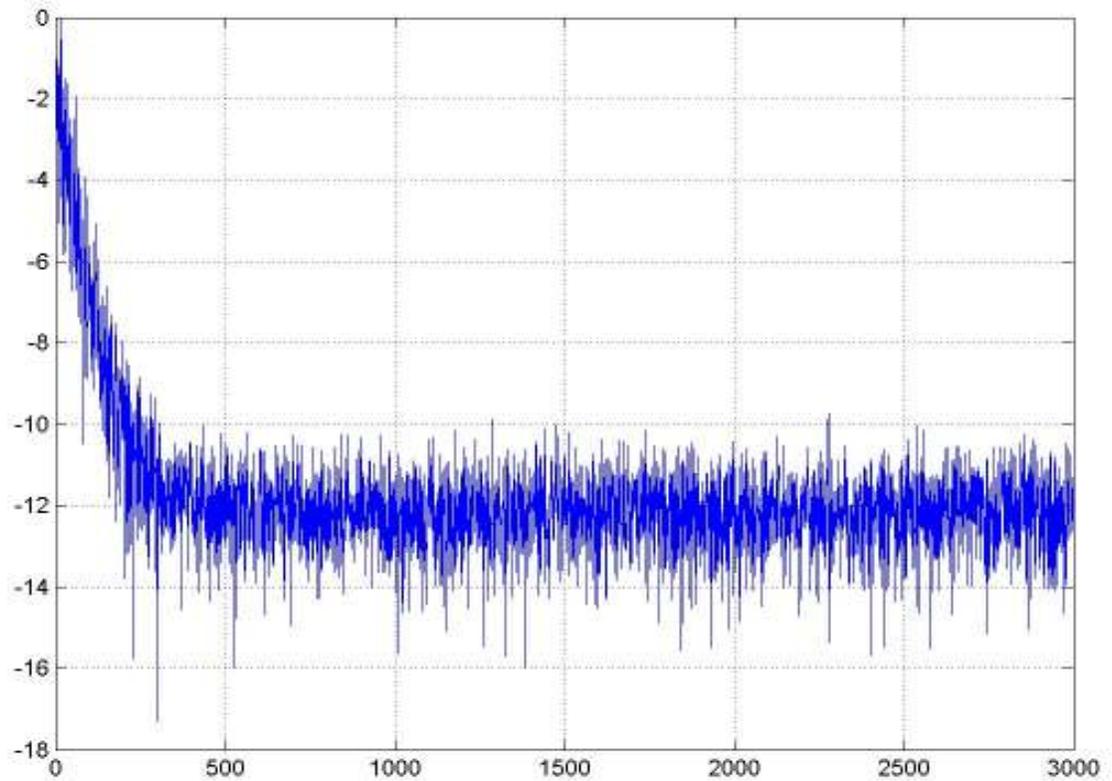


Fig 8.2 SYSTEM IDENTIFICATION GRAPH OUTPUT

8.6 PROGRAM 6

```
%%%%%%%%%%  
%% EQUALIZER TESTING : PLOT OF BER vs SNRdB  
%%%%%%%%%%  
clear all  
close all  
clc  
tic  
%%%%%%%%%%  
% NORMALISED CHANNELS  
%%%%%%%%%%  
%ch = [0.4472, 0.8944]; % CH = 1  
% ch = [0.2014, 0.9586, 0.2014]; % CH = 2  
%ch = [0.2600, 0.9300, 0.2600]; % CH = 3  
%ch = [0.3040, 0.9029, 0.3040]; % CH = 4  
ch = [0.3410, 0.8760, 0.3410]; % CH = 5  
eq = [-0.1151 0.3693 -0.8293 1.7296 -0.8067 0.3874 -0.1861 0.0606];  
%%%%%%%%%%  
% OTHER INITIALIZATIONS  
%%%%%%%%%%  
  
n = 10000 ; % NUMBER OF SAMPLES  
  
plant = ch;  
  
snrmin = 1 ;  
  
snrmax = 30 ;  
  
N = length ( eq ) ;  
  
ber = [] ;  
  
%%%%%%%%%%  
%%%%%%%%%%  
%%%%%%%%%%  
  
for snrdb = snrmin : snrmax  
snrdb  
rand ('state',0);  
randn('state',0);  
%%%%%%%%%%  
%%%%%%%%%%
```

```

% RANDOM SIGNAL = s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
s = hardlims ( rand ( 1 , n ) - 0.5 ); % SIGNAL

r1 = conv ( s , plant ); % CONVOLUTION / CHANNEL OUTPUT

r = r1 ( 1 : n ); % FOR SYMMETRY

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GENERATION OF THE WHITE GAUSSIAN NOISE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sp = 1 ; % SIGNAL POWER
np = sp * 10 ^ ( - snrdb / 10 ); % NOISE POWER
noise = sqrt ( np ) * randn( 1 , n );
x = r + noise ; % NOISY CHANNEL OUTPUT / INPUT TO THE
EQUALIZER

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DESIRED SIGNAL = d
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
delay = round ( N / 2 );
d = [ zeros( 1 , delay ) , s ];
xslide = zeros ( 1 , N );
count = 0 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for j = 1 : n

    xslide ( 2 : N ) = xslide ( 1 : N - 1 );

    xslide ( 1 ) = x ( j );

    y = xslide * eq' ;

    if y >= 0 ;

        y = 1 ;

    else

        y = -1 ;

    end

```

```

        if d(j)~= y
            count = count + 1;
        end

        if count == 100
            break
        end

    end

end    % END OF j = 1 : n

ber1 = count / j ;

ber = [ ber , ber1 ] ;

end % END FOR snrdb

semilogy( snrmin : snrdb , ber )
xlabel ('SNR IN dB --->')
ylabel ('PROBABILITY OF ERROR --->')
title(' PLOT FOR THE BER FOR CHANNEL : ');

grid on
toc

%%%%%%%%%%END%%%%%%%%%%
%%%%%%%%%%

clear all
close all
hold off
%channel system order
sysorder = 5 ;
% Number of system points
N=2000;
inp = randn(N,1);
n = randn(N,1);
[b,a] = butter(2,0.25);
Gz = tf(b,a,-1);

```

```

%This function is submitted to make inverse Z-transform (Matlab central file exchange)
%The first sysorder weight value
%h=ldiv(b,a,sysorder)';
% if you use ldiv this will give h :filter weights to be
h= [0.0976;
    0.2873;
    0.3360;
    0.2210;
    0.0964;];
y = lsim(Gz,inp);
%add some noise
n = n * std(y)/(10*std(n));
d = y + n;
totallength=size(d,1);
%Take 60 points for training
N=60 ;
%begin of algorithm
w = zeros ( sysorder , 1 ) ;
for n = sysorder : N
    u = inp(n:-1:n-sysorder+1) ;
    y(n)= w' * u;
    e(n) = d(n) - y(n) ;
% Start with big mu for speeding the convergence then slow down to reach the correct
weights
    if n < 20
        mu=0.32;
    else
        mu=0.15;
    end
    w = w + mu * u * e(n) ;
end
%check of results
for n = N+1 : totallength
    u = inp(n:-1:n-sysorder+1) ;
    y(n) = w' * u ;
    e(n) = d(n) - y(n) ;
end
hold on
plot(d)
plot(y,'r');
title('System output') ;
xlabel('Samples')
ylabel('True and estimated output')
figure
semilogy((abs(e))) ;
title('Error curve') ;

```

```
xlabel('Samples')
ylabel('Error value')
figure
plot(h, 'k+')
hold on
plot(w, 'r*')
legend('Actual weights','Estimated weights')
title('Comparison of the actual weights and the estimated weights') ;
axis([0 6 0.05 0.35])
```

8.6 a) RESULT

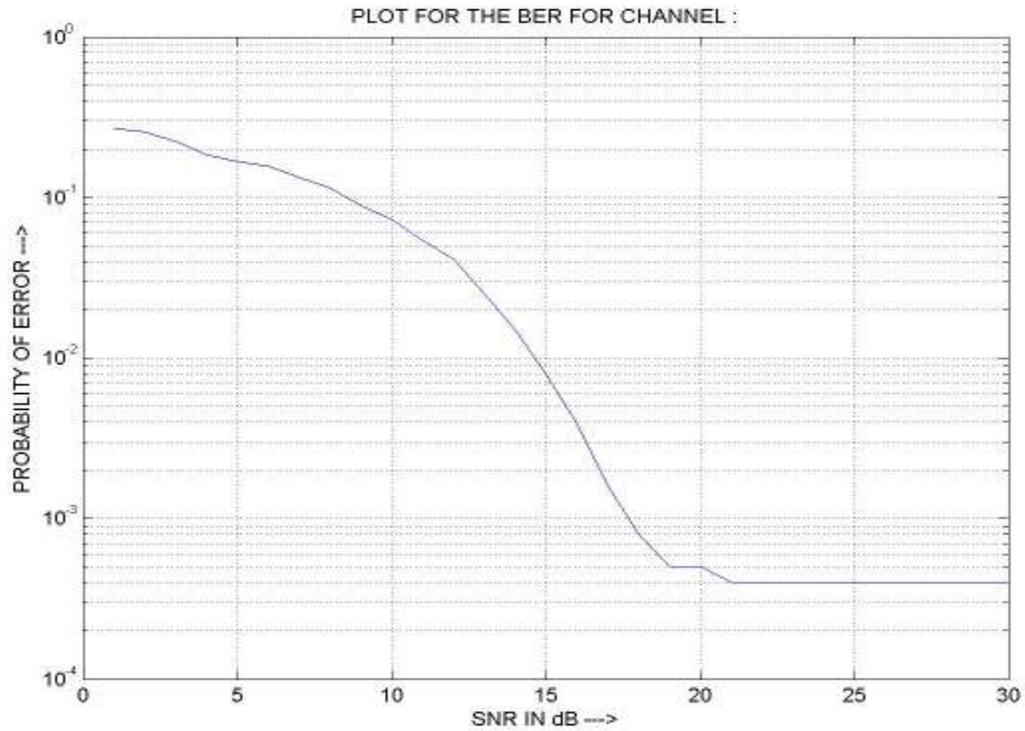


Fig 8.3 showing snr vs ber graph

snrdb = 1 , snrdb = 2 , snrdb = 3 , snrdb = 4 , snrdb = 5 , snrdb = 6 , snrdb = 7

snrdb = 8 , snrdb = 9 , snrdb = 10 , snrdb = 11 , snrdb = 12 , snrdb = 13 , snrdb = 14

snrdb = 15 , snrdb = 16 , snrdb = 17 , snrdb = 18 , snrdb = 19 , snrdb = 20 , snrdb = 21

snrdb = 22 , snrdb = 23 , snrdb = 24 , snrdb = 25 , snrdb = 26 , snrdb = 27 , snrdb = 28

snrdb = 29 , snrdb = 30

elapsed time =

5.7340

Chapter 9

**CONCLUSION
AND
REFERENCES**

CONCLUSION:-

- They can easily be designed to be "linear phase" (and usually are). Put simply, linear-phase filters delay the input signal, but don't distort its phase.
- They are simple to implement. On most DSP microprocessors, the FIR calculation can be done by looping a single instruction.
- They are suited to multi-rate applications. By multi-rate, we mean either "decimation" (reducing the sampling rate), "interpolation" (increasing the sampling rate), or both. Whether decimating or interpolating, the use of FIR filters allows some of the calculations to be omitted, thus providing an important computational efficiency. In contrast, if IIR filters are used, each output must be individually calculated, even if that output will be discarded (so the feedback will be incorporated into the filter).
- They have desirable numeric properties. In practice, all DSP filters must be implemented using "finite-precision" arithmetic, that is, a limited number of bits. The use of finite-precision arithmetic in IIR filters can cause significant problems due to the use of feedback, but FIR filters have no feedback, so they can usually be implemented using fewer bits, and the designer has fewer practical problems to solve related to non-ideal arithmetic.
- They can be implemented using fractional arithmetic. Unlike IIR filters, it is always possible to implement a FIR filter using coefficients with magnitude of less than 1.0. (The overall gain of the FIR filter can be adjusted at its output, if desired.) This is an important consideration when using fixed-point DSP's, because it makes the implementation much simpler.

REFERENCES

- 1) Digital signal processing by Simon hacking.
- 2) Digital signal processing by J G Provakis and D G Manolakis
- 3) Wikipedia.org
- 4) ccrma.stanford.edu
- 5) www.bores.com
- 6) www.logicdevices.com
- 7) <http://www.mathworks.com/products/filterdesign/demos.html?file=/products/demos/shipping/filterdesign/lpfirdemo.html>
- 8) Adaptive filters and equalizer by Bernard (Mulgrew, Colin F N Cowen)