CHANNEL EQUALIZATION USING GA FAMILY

CHANNEL EQUALIZATION USING GA FAMILY

Debidutta Mohanty

Department of Electronics and Communication Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Orissa, India

Channel Equalization using GA Family

Thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Technology

(Research)

in

Engineering

by

Debidduta Mohanty

under the guidance of

Dr. Ganapati Panda

Department of Electronics and Communication Engineering

National Institute of Technology Rourkela

Rourkela-769 008, Orissa, India

Department of Electronics and Communication Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Orissa, India.

Certificate

This is to certify that the work in the thesis entitled *Channel Equalization Using GA Family* by *Mr Debidutta Mohanty* in partial fulfillment of the requirements for the award of Master of Technology (Research) Degree in Electronics and communication Engineering with specialization in "Telematics and Signal Processing" at the National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by her 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.

Ganapati Panda
Professor
ECE department of NIT Rourkela

Place: NIT Rourkela
Date: 24 January 2008

Acknowledgment

First of all, I would like to express my deep sense of respect and gratitude towards my advisor and guide **Prof (Dr.) Ganapati Panda**, who has been the guiding force behind this work. I want to thank him for introducing me to the field of Signal Processing and giving me the opportunity to work under him. I am greatly indebted to him for his constant encouragement and invaluable advice in every aspect of my academic life. His presence and optimism have provided an invaluable influence on my career and outlook for the future. I consider it my good fortune to have got an opportunity to work with such a wonderful person.

I express my respects to **Prof (Dr.) K. K. Mahapatra**, **Prof (Dr.) G. S. Rath**, **Prof Dr. S.K. Patra**, and **Dr. S. Meher** for teaching me and also helping me how to learn. They have been great sources of inspiration to me and I thank them from the bottom of my heart.

I would like to thank all faculty members and staff of the Department of Electronics and Communication Engineering, N.I.T. Rourkela for their generous help in various ways for the completion of this thesis.

I would also like to mention the name of **S.K.Nanda** and **B.Majhi** for helping me a lot during the thesis period.

I would like to thank all my friends and especially my classmates for all the thoughtful and mind stimulating discussions we had, which prompted us to think beyond the obvious. I have enjoyed their companionship so much during my stay at NIT Rourkela.

I am especially indebted to my parents (**Mr. Ajit Kumar Mohanty** and **Mrs. Bidutlata Mohanty**) and sisters (**Luna dei** and **Lima dei**) for their love, sacrifice, and support. They are my first teachers after I came to this world and have set great examples for me about how to live, study, and work.

Debidutta Mohanty

Abstract

High speed data transmissions over communication channels distort the transmitted signals in both amplitude and phase due to presence of Inter Symbol Interference (ISI). Other impairments like thermal noise, impulse noise and cross talk also cause further distortions to the received symbols. Adaptive equalization of the digital channels at the receiver removes/reduces the effects of such ISIs and attempts to recover the transmitted symbols. Basically an equalizer is an inverse filter which is placed at the front end of the receiver. Its transfer function is inverse to the transfer function of the associated channel.

The Least-Mean-Square (LMS), Recursive-Least-Square (RLS) and Multilayer perceptron (MLP) based adaptive equalizers aim to minimize the ISI present in the digital communication channel. These are gradient based learning algorithms and therefore there is possibility that during training of the equalizers, its weights do not reach to their optimum values due to the mean square error (MSE) being trapped to local minimum. In other words true Weiner solution is not achieved because of gradient based training. The bit-error-rate (BER) performance of the equalizer further degrades when data transmission takes place through nonlinear channels.

The standard derivative based algorithms suffer from local minima problem while obtaining the solution of the weights. To prevent the premature settling of the weights, evolutionary computing based update algorithm is proposed which is essentially a derivative free technique. Equalization is basically an iterative process of minimization of mean square error. Thus equalization can be viewed as optimization problem. The minimization of squared error is achieved iteratively using GA. Thus GA based approach is an efficient method to achieve adaptive channel equalization. In the present thesis classes of new adaptive channel equalizers are proposed using derivative free evolutionary computing tools such as Genetic Algorithm (GA) and Particle swarm optimization (PSO). These algorithms are suitably used to update the weights of the proposed equalizers. The performance of these

equalizers is evaluated in terms of speed of convergence, computational time and bit-error-rate (BER) and is compared with its LMS based counter part. It is observed that the new set of adaptive equalizers offer improved performance so far as the accuracy of reception is concerned. However, in order of increasing training time the equalizers may be arranged as the adaptive Genetic Algorithm (AGA), Particle Swarm Optimization (PSO), Real coded Genetic Algorithm (RCGA), Binary coded Genetic Algorithm (BGA) based equalizer.

However being a population based algorithm, standard Genetic Algorithm (SGA) suffers from slower convergence rate. To minimize the training time three different adaptive GAs (AGAs) are proposed in the thesis and their convergence times have been compared. The thesis also investigates on the new equalizers using Real coded Genetic Algorithm (RCGA) and Binary coded Genetic Algorithm (BGA). Their performances are also evaluated.

In the conventional FLANN (Functional Link Artificial Neural Network) [1] structure the complexity increases due to incorporation of more number of paths after functional expansions. To reduce the structural complexity some pruning of structure is essential. Keeping this in mind the GA based pruning strategy is used in the FLANN identifier. It is observed that about 50% of the total signal paths can be pruned keeping the performance identical to that of original FLANN structure.

Acronyms

ISI	Inter Symbol Interference
LMS	Least Mean Square
RLS	Recursive Least Square
MLP	Multi Layer Perceptron
MSE	Mean Square Error
BER	Bit Error Rate
GA	Genetic Algorithm
PSO	Particle Swarm Optimization
AGA	Adaptive Genetic Algorithm
RCGA	Real Coded Genetic Algorithm
BGA	Binary Coded Genetic Algorithm
SGA	Standard Genetic Algorithm
FLANN	Functional Link Artificial Neural Network
DCR	Digital Cellular Radio
LAN	Local Area Network
DSP	Digital Signal Processing
RBF	Radial Basis Function
SI	System Identification
GD	Gradient Descent
FIR	Finite Impulse Response
BP	Back Propagation
BPSK	Binary Phase Shift Keying
pdf	Probability Density Function
CDF	Cumulative Distribution Function
MLSE	Maximum Likelihood Sequence Estimator
TE	Transversal Equalizer
DFE	Decision Feedback Equalizer
FFMLP	Feed Forward Multi Layer Perceptron
SV	Support Vector
MAP	Maximum a Posteriori
FPGA	Field Programmable Gate Array
IIR	Infinite Impulse Response
ANN	Artificial Neural Network

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 BACKGROUND

In the modern day there is a burgeoning growth of digital transmission across communication networks for voice, video and data traffic. The Internet and mobile telecommunications have recently dominated social and business interaction across the world. The telephone networks were originally designed for voice communication but, in recent times, the advances in digital communications using ISDN, data communications with computers, fax, video conferencing etc. have pushed the use of these facilities far beyond the scope of their original intended use. Similarly, introduction of digital cellular radio (DCR) and wireless local area networks (LAN's) have stretched the limited available radio spectrum capacity to the limits it can offer. Bandwidth efficiency has become a growing concern with the rise in data rates within an expanding communication network.

The improvements made in communication technology throughout the last number of decades have been facilitated by significant hardware and digital signal processing advances. Computing power has grown with improving silicon technology as governed by Moore's Law. This will enable the implementation of larger and more complex signal processing algorithms. Consequently, there will come a time when the computational cost of DSP algorithms is not an issue rather their effectiveness.In this thesis transmission dispersion issues due to finite bandwidth of the channel are considered. Finite bandwidth and multiple propagation paths can degrade the digital transmission resulting in intersymbol interference (ISI) [2].The

addition of noise during propagation also degrades the quality of the received signal. The process of reversing the effect of ISI is defined as equalization, which has been described as the most heavily exploited area for adaptive filtering in digital communication systems, speed and efficiency for economic bandwidth utilization.

1.2 A REVIEW ON CHANNEL EQUALIZATION

To counter the effect of multipath propagation there are several techniques available. The most widely used include frequency diversity, space diversity, amplitude equalization and channel equalization (amplitude and delay correction) [3]. The first two of these require a bandwidth overhead. But, in general, bandwidth is costly. These signal diversity techniques were used in analogue radio and have been easily adapted to digital systems that undergo highly selective interference. The amplitude equalizers are designed to flatten the received spectrum to correct the spectral shape. An amplitude equalizer is often used in conjunction with frequency or space diversity, which can provide sufficient equalization for specific channels (minimum phase). However, to adequately characterize the effects of all channel types, (minimum and non-minimum phase), the channel equalizer is adopted [3].

The channel equalizer reconstructs or estimates the corrupted data sequence from a set of received symbols. Equalizers have been adopted in telephone and mobile communication systems to improve the symbol error rates and the linear FIR filter has been used within equalization, which dates back to the time when loading coils were used to improve voice transmission in telephone networks [2].

The FIR approach classifies the received class sets to their desired output using a linear function of the filter inputs. However, this does not always provide ideal separation of the input data points. It has been shown through Bayesian analysis that the ideal classification of a non-minimum phase channel should have non-linear characteristics [4].

1.2.1 Non-linear Equalization

Research conducted in the field of channel equalization during 1990s has shown that there are performance advantages using non-linear classification. Neural network structures have been implemented to achieve this, such as Multi-Layer Perceptron (MLP) [5, 6] and Radial-Basis Function (RBF) networks [7, 8].

The Bayesian equalisation solution can be implemented by an RBF neural network [9], and can be described as a universal approximator [10] . However, the RBF has a high computation cost. The network structure size can increase exponentially as the problem difficulty increases [9] . The RBF model uses radial distribution functions to approximate the symbol centers of each problem class where the classification boundary between the classes depends upon the interaction of these radial function sets [11] . If an insufficient number of these centers are given, or are misplaced, the equalisation can be severely affected and poor initialization is seen to hinder the solution [11] . Work in RBF equalisation is detailed by Chang et al and Chen et al [12, 13].

The MLP can be compared to the FIR filter in that when the MLP is reduced to its most basic form, a single perceptron with a linear activation function; it is identical to the FIR filter [11]. As the MLP structure is enlarged, its filtering capacity becomes increasingly non-linear, which allows for a complexity/performance trade-off, if so desired. A three layer MLP with sufficient perceptron units is able to create arbitrary classification of the input vector [14] ; therefore the MLP neural network can also be described as a universal approximator [15] . Gibson et al first applied the MLP structure to the channel equalisation problem [16] and studies into its training have been made by Siu and Sweeney [6, 17].

In 1999, Patra et.al utilized functional link artificial neural networks (FLANN) [18] to build the nonlinear channel equaliser. The basic principle of FLANN is to expand the dimensionality of the input signal space by using a set of linearly independent functions. The expansion can produce fairly complicated decision boundaries at the output space, so the FLANN is capable of dealing with linear inseparable problems. As FLANN has a two-layer structure its circuit is generally

simpler than MLP, and thus faster processing speed can be achieved.

Although FLANN exhibits better performance than the MLP, it still has some potential drawbacks [19]. Specifically, to further improve the BER performance one needs to enlarge the dimensionality of its input signal space. This will significantly increase the number of nodes in the input layer, and thus the circuit may become too complicated to be practical.

A Genetic Algorithm (GA) is a stochastic training scheme that need not have a derivation that requires knowledge of the local error gradient [18] , which gradient-descent training relies on. A GA consists of an evolutional process that raises the fitness of a population using the Darwinian survival of the fittest criterion [18]. A GA relies upon the use of a solution population. Each solution within the population has to generate a cost value in each training iteration, which is based on the equalisation error. GAs has proven to be useful in training and search applications that suffer from stability problems, locating solutions that have previously been unobtainable [20].

Swarm intelligence (SI) is an artificial intelligence technique based around the study of collective behavior in decentralized, self-organized, systems. The expression "swarm intelligence" was introduced by Beni and Wang in 1989, in the context of cellular robotic systems. SI systems are typically made up of a population of simple agents interacting locally with one another and with their environment. Although there is normally no centralized control structure dictating how individual agents should behave, local interactions between such agents often lead to the emergence of global behavior. Examples of systems like this can be found in nature, including ant colonies, bird flocking, animal herding, bacteria molding and fish schooling.

Since GA and SI can be viewed as an optimization algorithm and channel equalization can also be treated as a squared error minimization problem, so these algorithms can be employed to solve the problem effectively.

1.3 MOTIVATION

Adaptive filtering has proven to be useful in many contexts such as linear prediction, channel equalization, noise cancellation, and system identification. For system identification, which is arguably the most general adaptive filtering paradigm shown in Fig1.1, the adaptive filter attempts to iteratively determine an optimal model for the unknown system, or "plant", based on some function of the error between the output of the adaptive filter and the output of the plant. The optimal model or solution is attained when this function of the error is minimized. The adequacy of the resulting model depends on the structure of the adaptive filter, the algorithm used to update the adaptive filter parameters, and the characteristics of the input signal.

When the error surface is multimodal, local optimization techniques that work well for FIR adaptive filters, such as versions of gradient descent (GD) including the least mean squares (LMS) algorithm and back propagation for neural networks, are not suitable because they are likely to become trapped in the local minimum and never converge to the global optimum. Since swarm intelligence and Genetic algorithm techniques differs from traditional methods and are not fundamentally limited by restrictive assumptions about the search space, such as assumptions concerning continuity, existence of derivatives, unimodality, etc, they have great potential in providing better results than conventional techniques.

Hence the main motivation behind the proposed thesis work is to explore the use of the evolutionary computing tools such as GA and its variants as well as Particle Swarm Optimization (PSO) in adaptive channel equalization of nonlinear channel and compare its performance with these obtained from standard methods such that the LMS, RLS and MLP .

1.4 THESIS OUTLINE

The complete outline of the present thesis proceeds as follows:

Chapter 1 gives an introduction to channel equalization and reviews various learning algorithms such as the Least-Mean-Square (LMS) algorithm, Recursive-

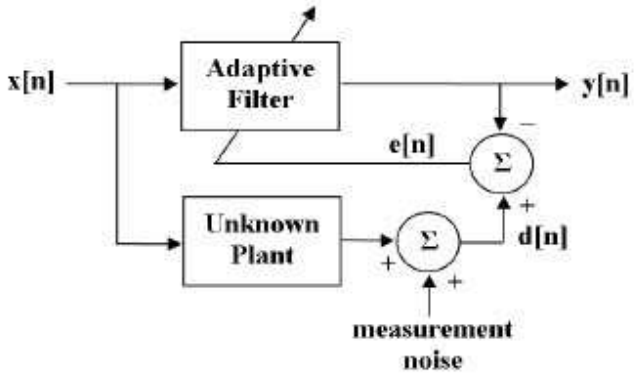


Figure 1.1: Adaptive System Identification

Least-Square (RLS) algorithm, Back-Propagation (BP) algorithm, Radial-Basis Function (RBF), Functional Link Artificial Neural Network (FLANN), Genetic Algorithm (GA), and Particle Swarm Optimization (PSO) used to train the equalizer parameters. It also includes the motivation behind undertaking the thesis work.

Chapter 2 discusses various channel equalization techniques of different communication channels.It also deals about inter symbol interference (ISI) and linear and nonlinear adaptive equalizer structures.

Chapter 3 presents an overview of adaptive filter and its structures. It describes various adaptive filter structures and their applications. Three gradient based training methods such as the LMS, RLS and Back Propagation (BP) algorithm are also explained in this chapter. The performances of adaptive equalizers with LMS, RLS and Back propagation training are obtained through simulation study and are presented and the findings are compared in this chapter.

Chapter 4 provides an introduction to evolutionary computing technique and discusses in details about standard genetic algorithm (SGA) and its operators. The drawbacks of gradient based algorithms are also discussed. It also focuses various GAs such as real coded GA (RCGA) and three forms of Adaptive GA (AGA).

Chapter 5 provides a comprehensive evaluation of all types of new equalizers proposed in Chapter-4. The performance of SGA, AGA, and RCGA and the LMS based equalizers are compared using exhaustive simulations. The performance in-

cludes CPU time, convergence and bit-error-rates.

Chapter 6 introduces the concept of swarm intelligence and discusses in details about Particle Swarm Optimization (PSO) which is one of the tools of evolutionary computing. The performance of the PSO equalizer is obtained and compared with that of SGA and LMS based equalizer. These results are presented in this chapter.

The structural complexity of the FLANN structure increases due to incorporation of more number of paths after functional expansions. But certain signal path does not contribute to the performance. Hence pruning of the structure is required without sacrificing the performance. This issue is investigated and solved in **Chapter 7** using Binary coded GA (BGA) algorithm. It is observed that about 50% pruning of the structure is possible.

Chapter 8. deals with the conclusion of the investigation made in the thesis. This chapter also suggests some future research related to the topic.

1.5 THESIS CONTRIBUTIONS

The major contribution of the thesis is outlined below:

- The MLP based channel equalizers perform better than the LMS based equalizers and in some cases better than RLS based equalizers in terms of minimum mean square error and bit-error rate.

- The GA based approach for channel equalization is introduced. The GA based approach is found to be more efficient than other standard derivative based learning. In addition the AGA and RCGA based equalizers have been proposed and shown to have better performance and involve less computational complexity.

- The PSO is also used for updating the weights of the equalizers during training. This derivative free training algorithm offers faster convergence performance and less BER. It also involves less computation.

- Finally the FLANN structure is pruned using GA to obtain reduced identifier structure without sacrificing the quality. About 50% reduction in structure is possible.

Chapter 2

Basic Principles of Channel Equalization

2.1 INTRODUCTION

In an ideal communication channel, the received information is identical to that transmitted. However, this is not the case for real communication channels, where signal distortions take place. A channel can interfere with the transmitted data through three types of distorting effects: power degradation and fades, multi-path time dispersions and background thermal noise [2] . Equalisation is the process of recovering the data sequence from the corrupted channel samples. A typical baseband transmission system is depicted in Fig2.1, where an equalizer is incorporated within the receiver [21] .

The equalisation approaches investigated in this thesis are applied to a BPSK (binary phase shift keying) baseband communication system. Each of the transmitted data belongs to a binary and 180 out of phase alphabet $\{-1, +1\}$.

Within this chapter channel baseband models are explained. A transversal equaliser structure is also examined [22]. -

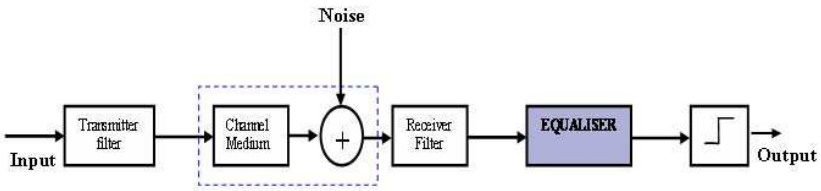


Figure 2.1: A Baseband Communication System

2.1.1 MULTIPATH PROPAGATION

Within telecommunication channels multiple paths of propagation commonly occur. In practical terms this is equivalent to transmitting the same signal through a number of separate channels, each having a different attenuation and delay [2] . Consider an open-air radio transmission channel that has three propagation paths, as illustrated in Fig2.2 [23]. These could be direct, earth bound and sky bound.

Fig2.1b describes how a receiver picks up the transmitted data. The direct signal is received first whilst the earth and sky bound are delayed. All three of the signals are attenuated with the sky path suffering the most.

Multipath interference between consecutively transmitted signals will take place if one signal is received whilst the previous signal is still being detected [2]. In Fig2.1 this would occur if the symbol transmission rate is greater than $1/\tau$ where, τ represents transmission delay. Because bandwidth efficiency leads to high data rates, multi-path interference commonly occurs.

Channel models are used to describe the channel distorting effects and are given as a summation of weighted time delayed channel inputs d(n-i) .

$$H(z) = \sum_{i=0}^{m} d(n-i)z^{-i} = d(n) + d(n-1)z^{-1} + d(n-2)z^{-2} + ... \qquad (2.1)$$

The transfer function of a multi-path channel is given in (2.1). The model coefficients d(n-i) describe the strength of each multipath signal.

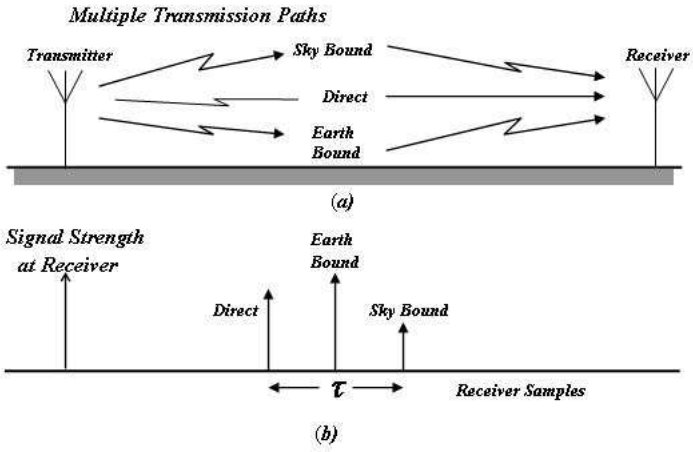


Figure 2.2: Impulse Response of a transmitted signal in a channel which has 3 modes of propagation, (a) The signal transmitted paths, (b) The received samples

2.2 MINIMUM AND NONMINIMUM PHASE CHANNELS

When all the roots of the H(Z) lie within the unit circle, the channel are termed minimum phase. The inverse of a minimum phase [24] channel is convergent, illustrated by (2.2) :

$$H(z) = \begin{cases} 1.0 + 0.5z^{-1}\frac{1}{H(z)} \\ \frac{1}{1.0+0.5z(-1)} \\ \sum_{i=0}^{\infty}(\frac{-1}{2})^i z^{-i} \\ 1 - 0.5z^{-1} + 0.25z^{-2} - 0.125z^{-3} + ... \end{cases} \tag{2.2}$$

where as the inverse of non-minimum phase channels are not convergent, as shown in (2.3)

$$H(z) = \begin{cases} 0.5 + 1.0z^{-1}\frac{1}{H(z)} \\ \frac{z}{1.0+0.5z} \\ z.[\sum_{i=0}^{\infty}(\frac{-1}{2})^i z^{-i}] \\ z.[1 - 0.5z + 0.25z^2 - 0.125z^3] \end{cases} \tag{2.3}$$

Since equalisers are designed to invert the channel distortion process they will in effect model the channel inverse. The minimum phase channel has a linear inverse model therefore a linear equalisation solution exists. However, limiting the inverse model to m-dimensions will approximate the solution and it has been shown that non-linear solutions can provide a superior inverse model in the same dimension [21] .

A linear inverse of a non-minimum phase channel does not exist without incorporating time delays. A time delay creates a convergent series for a non-minimum phase model, where longer delays are necessary to provide a reasonable equaliser. (2.4) describes a non-minimum phase channel with a single delay inverse and a four sample delay inverse. The latter of these is the more suitable form for a linear filter.

$$H(z) = \begin{cases} 0.5 + 1.0z^{-1}z^{-1}\frac{1}{H(z)} \\ \frac{1}{1+0.5z} \\ 1 - 0.5z + 0.25z^2 - 0.125z^3 + ...(noncausal)z^{-4}\frac{1}{H(z)} \\ z^{-3} - 0.5z^{-2} + 0.25z^{-1} - 0.125z + ...(truncatedandcausal) \end{cases} \tag{2.4}$$

The three-tap non-minimum phase channel $H(z) = 0.3410+0.8760z^{-1}+0.3410z^{-2}$ is used throughout this thesis for simulation purposes. A channel delay, D, is included to assist in the classification so that the desired output becomes $u(n-D)$.

2.3 INTERSYMBOL INTERFERENCE

Inter-symbol interference (ISI) has already been described as the overlapping of the transmitted data [2]. It is difficult to recover the original data from one channel sample dimension because there is no statistical information about the multipath propagation. Increasing the dimensionality of the channel output vector helps characterize the multipath propagation. This has the affect of not only increasing the number of symbols but also increases the Euclidean distance between the output classes.

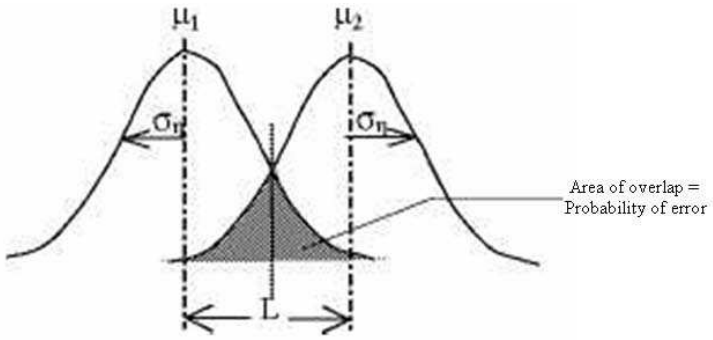


Figure 2.3: Interaction between two neighboring symbols.

When additive Gaussian noise, η is present within the channel, the input sample will form Gaussian clusters around the symbol centers. These symbol clusters can be characterized by a probability density function (pdf) with a noise variance σ_η^2 , where the noise can cause the symbol clusters to interfere. Once this occurs, equalisation filtering will become inadequate to classify all of the input samples. Error control coding schemes can be employed in such cases but these often require extra bandwidth [25].

2.3.1 SYMBOL OVERLAP

The expected number of errors can be calculated by considering the amount of symbol interaction, assuming Gaussian noise. Taking any two neighboring symbols, the cumulative distribution function (CDF) can be used to describe the overlap between the two noise characteristics. The overlap is directly related to the probability of error between the two symbols and if these two symbols belong to opposing classes, a class error will occur.

Fig2.3 shows two Gaussian functions that could represent two symbol noise distributions. The Euclidean distance, L, between symbol canters and the noise variance, σ^2, can be used in the cumulative distribution function of (2.5) to calculate the area of overlap between the two symbol noise distributions and therefore the probability of error, as in (2.6).

$$CDF(x) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\Pi\sigma}} \exp[-\frac{x^2}{2\sigma^2}]dx \tag{2.5}$$

$$P(e) = 2CDF[\frac{L}{2}] \tag{2.6}$$

Since each channel symbol is equally likely to occur [22], the probability of unrecoverable errors occurring in the equalisation space can be calculated using the sum of all the CDF overlap between each opposing class symbol. The probability of error is more commonly described as the BER. (2.7) describes the BER based upon the Gaussian noise overlap, where N_{SP} is the number of symbols in the positive class, N_m is the number of number of symbols in the negative class and $\triangle_i$, is the distance between the i^{th}positive symbol and its closest neighboring symbol in the negative class.

$$BER(\sigma_n) = \log[\frac{2}{N_{sp} + N_m} \sum_{i=1}^{N_{sp}} CDF(\frac{\Delta_i}{2\sigma_n})] \tag{2.7}$$

2.4 CHANNEL EQUALIZATION

The optimal BER equalisation performance is obtained using a maximum likelihood sequence estimator (MLSE) on the entire transmitted data sequence [26] . A more practical MLSE would operate on smaller data sequences but these can still be computationally expensive, they also have problems tracking time-varying channels and can only produce sequences of outputs with a significant time delay. Another equalisation approach implements a symbol-by-symbol detection procedure and is based upon adaptive filters [2]. The symbol-by-symbol approach to equalisation applies the channel output samples to a decision classifier that separates the symbol into their respective classes. Two types of symbol-by-symbol equalisers are examined in this thesis, the transversal equalizer (TE) and decision feedback equaliser (DFE). Traditionally these equalisers have been designed using linear filters, LTE and LDFE, with a simple FIR structure. The ideal equaliser will model the inverse of the channel model but this does not take into account the effect of noise within the channel.

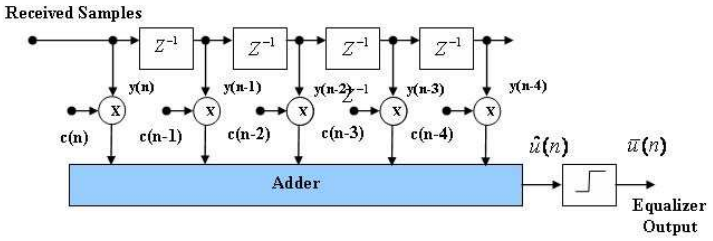


Figure 2.4: Linear Transversal Equalizer

2.4.1 TRANSVERSAL EQUALIZER

The transversal equaliser uses a time-delay vector, Y(n)(2.8), of channel output samples to determine the symbol class. The $\{m\}$ TE notation used to represent the transversal equaliser specifies m inputs. The equaliser filter output will be classified through a threshold activation device (Fig2.4) so that the equaliser decision will belong to one of the BPSK states $\overline{u}(n) \in \{-1, +1\}$.

$$Y(n) = [y(n), y(n-1), ..., y(n-(m-1))] \tag{2.8}$$

Considering the inverse of the channel $H(z) = 1.0 + 0.5z^{-1}$ that was given in (2.3), this is an infinitely long convergent linear series: $\frac{1}{H(z)} = \sum_{i=1}^{m}(-1/2)^i z^{-i}$. Each coefficient of this inverse model can be used in a linear equaliser as a FIR tap-weight. Each tap-dimension will improve the accuracy; however, high input dimensions leave the equaliser susceptible to noisy samples. If a noisy sample is received, this will remain within the filter affecting the output from each equaliser tap. Rather than designing a linear equaliser, a non-linear filter can be used to provide the desired performance that has a shorter input dimension; this will reduce the sensitivity to noise.

2.4.2 Decision Feedback Equalizer

A basic structure of the decision feedback equalizer (DFE) is shown in Fig2.5. The DFE consists of a transversal feed forward and feedback filter. In the case when the communication channel causes severe ISI distortion, the LTE could not be provide satisfactory performance. Instead, a DFE is required. The DFE uses past corrected samples, w(n), from a decision device to the feedback filter and

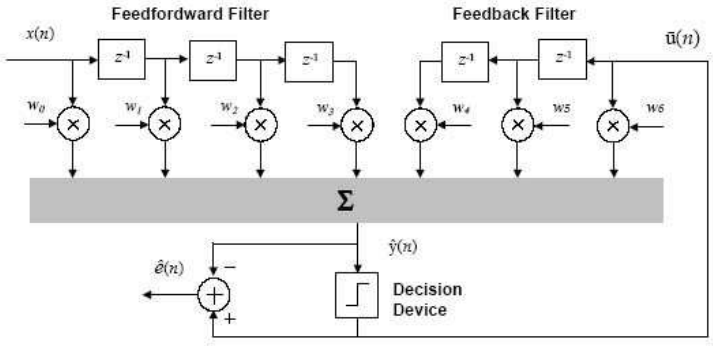


Figure 2.5: Decision Feedback Equalizer

combines with the feed forward filter. In effect, the function of the feedback filter is to subtract the ISI produced by previously detected symbols from the estimates of future samples . Consider that the DFE is updated with a recursive algorithm; the feed forward filter weights and feedback filter weights can be jointly adapted by the LMS algorithm on a common error signal $\hat{e}(n)$ as shown in (2.9).

$$W(n+1) = W(n) + \mu\hat{e}(n)V(n) \tag{2.9}$$

where $\hat{e}(n) = u(n) - y(n)$ and $V(n) = [x(n), x(n-1), ..., x(n-k1-1), u(n-k2-l), ...u(n)]^T$. The feed forward and feedback filter weight vectors are written in a joint vector as $W(n) = [w_0(n), w_1(n), ..., w_{k1+k2-1}(n)]^T$. k1 and k2 represent the feed forward and feedback filter tap lengths respectively. Suppose that the decision device causes an error in estimating the symbol u(n). This error can propagate into subsequent symbols until the future input samples compensate for the error. This is called the error propagation which will cause a burst of errors . The detrimental potential of error propagation is the most serious drawback for decision feedback equalization. Traditionally, the DFE is described as being a non-linear equalizer because the decision device is non-linear. However, the DFE structure is still a linear combiner and the adaptation loop is also linear. It has therefore been described as a linear equalizer structure .

2.4.3 NON-LINEAR EQUALISER STRUCTURES

Recently there has been interest into the application of non-linear architectures to the equalization problem, with the aim of enhancing the noise performance as well as the channel non-linearity. Both LTF and DFE architectures can benefit from the implementation of these structures, by showing an enhanced Bit Error Rate (BER) performance when compared to conventional linear architectures . Three particular types of non-linearity have been investigated, the Gaussian radial basis function (RBF), the feed forward multilayer perceptron (FFMLP), and the Volterra Kernel.

The Gaussian RBF equalizer has been suggested as a solution to the fast fading time varying mobile telecommunications systems , where its adaptation to the non-stationary channel model has been shown to surpass the performance of a more conventional maximum likelihood sequence estimator MLSE. The RBF model also is surprisingly parsimonious when compared to the MLSE. However, as the dimensionality of the input increases, the number of indicated kernels also increases. If the kernel centers are not identified with a high degree of accuracy the system can be over specified.

The Volterra kernel (third order) has also been utilized in satellite communication channels, and as such it can be trained utilizing a least squares training algorithm. However like the RBF kernel the Volterra series suffers from the curse of dimensionality caused by the proliferation of the cross coefficients. This problem can be alleviated by a careful choice of the desired polynomial , which will result in the polynomial structure being both parsimonious and trainable using the Support Vector (SV) approach. The FFMLP was the first multilayer neural network structure to be implemented after a method of training was discovered . Work by Siu (1990) has shown the feasibility of using these non-linear structures to equalize time delayed non-minimum phase channels; however, as it seems with all non-linear architectures, training difficulties tend to limit their effectiveness. It has been shown that the non-linear boundaries could be close to the optimal maximum a posteriori (MAP) boundary, which is formed by utilizing a Gaussian

RBF network with centers at all of the possible signal centers . It is possible to train a FFMLP with fewer processing units than that generated by the MAP criterion, and thus have a more parsimonious structure. There has, however, been a tendency to train to linear solutions that do not truly reflect the non-linear nature of the decision surface. The primary reason for this was that the gradient descent training schemes employed tend to cause premature convergence to local minima, as well as algorithmic instability, due primarily to the topology of the error surface. It has been shown that gradient descent can fail even when the FFMLP structure itself is sufficient to deal with the problem.

This chapter discussed the background of channel equalization and highlights some of the most common equalizer structures, the LTF and the DFE. Both the linear and non-linear methods have been discussed with the aim of highlighting the necessity of the non-linear architecture, even though we have used a linear equalizer as the test problem.

2.5 SUMMARY

This chapter explains the needs and different methods of channel equalization. The natures of minimum and non-minimum phase channels are described. It is seen that the equalizer dimension is large for non-minimum channels. Various interferences in communication channels are addressed. Multipath interference is explained briefly. A transversal equalizer and decision feed back equalizer is briefly explained. Finally the nonlinear equalizer structures are explained briefly.

Chapter 3

Comparison of Channel Equalization Performance using Different Adaptive Algorithms

Introduction

An adaptive filter is a computational device that attempts to model the relationship between two signals in real time in an iterative manner. Adaptive filters are often realized either as a set of program instructions running on an arithmetical processing device such as a microprocessor or DSP chip, or as a set of logic operations implemented in a field-programmable gate array (FPGA) or in a semi-custom or custom VLSI integrated circuit. However, ignoring any errors introduced by numerical precision effects in these implementations, the fundamental operation of an adaptive filter can be characterized independently of the specific physical realization that it takes. For this reason, we shall focus on the mathematical forms of adaptive filters as opposed to their specific realizations in software or hardware.

An adaptive filter is defined by four aspects:

1. the *signals* being processed by the filter
2. the *structure* that defines how the output signal of the filter is computed from its input signal
3. the *parameters* within this structure that can be iteratively changed to alter the filter's input-output relationship

4. the *adaptive algorithm* that describes how the parameters are adjusted from one time instant to the next.

By choosing a particular adaptive filter structure, one specifies the number and type of parameters that can be adjusted. The adaptive algorithm used to update the parameter values of the system can take on a myriad of forms and is often derived as a form of *optimization procedure* that minimizes an *error criterion* that is useful for the task at hand.

In this section, we present the general adaptive filtering problem and introduce the mathematical notation for representing the form and operation of the adaptive filter. We then discuss several different structures that have been proven to be useful in practical applications. We provide an overview of the many and varied applications in which adaptive filters have been successfully used. We give a simple derivation of the *least-mean-square (LMS) algorithm*[27, 28], which is perhaps the most popular method for adjusting the coefficients of an adaptive filter, and we discuss some of this algorithm's properties and shortcomings. Finally, we discuss new algorithms and techniques which can be applied in place of conventional methods.

As for the mathematical notation used throughout this section, all quantities are assumed to be real-valued. Scalar and vector quantities shall be indicated by lowercase (e.g., x) and uppercase-bold (e.g., X) letters, respectively. We represent scalar and vector sequences or signals as x(n) and X(n), respectively, where n denotes the discrete time or discrete spatial index, depending on the application. Matrices and indices of vector and matrix elements shall be understood through the context of the discussion.

3.1 THE ADAPTIVE FILTERING PROBLEM

Figure 3.1 shows a block diagram in which a sample from a digital *input signal x(n)* is fed into a device, called an *adaptive filter*, that computes a corresponding *output signal* sample y(n) at time n. For the moment, the structure of the adaptive filter is not important, except for the fact that it contains adjustable parameters

whose values affect how y(n) is computed. The output signal is compared to a second signal d(n), called the *desired response signal*, by subtracting the two samples at time n. This difference signal, given by

$$e(n) = d(n) - y(n) \tag{3.1}$$

is known as the *error signal*. The error signal is fed into a procedure which alters or *adapts* the parameters of the filter from time n to time (n + 1) in a well-defined manner. This process of adaptation is represented by the oblique arrow that pierces the adaptive filter block in the figure. As the time index n is incremented, it is hoped that the output of the adaptive filter becomes a better and better match to the desired response signal through this adaptation process, such that the magnitude of e(n) decreases over time. In this context, what is meant by "better" is specified by the form of the adaptive algorithm used to adjust the parameters of the adaptive filter.

In the adaptive filtering task, adaptation refers to the method by which the parameters of the system are changed from time index n to time index (n + 1). The number and types of parameters within this system depend on the computational structure chosen for the system. We now discuss different filter structures that have been proven useful for adaptive filtering tasks.

3.2 FILTER STRUCTURES

In general, any system with a finite number of parameters that affect how y(n) is computed from x(n) could be used for the adaptive filter in Fig.3.1. Define the *parameter* or *coefficient vector W(n)*

$$W(n) = \quad [w_0(n) \quad w_1(n)... \quad w_{L-1}(n)]^T \tag{3.2}$$

where $\{w_i(n)\}$, $0 < i < L - 1$ are the L parameters of the system at time n. With this definition, we could define a general input-output relationship for the adaptive filter as

$$y(n) = f(W(n), y(n-1), y(n-2), ...y(n-N), x(n), x(n-l), ...x(n-M+l)) \tag{3.3}$$

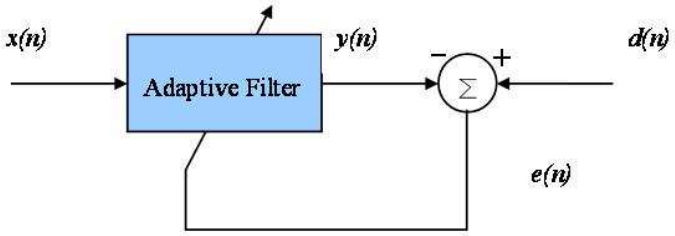


Figure 3.1: Adaptive Filtering Algorithm

where f ( ) represents any well-defined linear or nonlinear function and M and N are positive integers. Implicit in this definition is the fact that the filter is *causal*, such that future values of are not needed to be computed. While *non-causal* filters can be handled in practice by suitably buffering or storing the input signal samples, we do not consider this possibility.

Although (3.3) is the most general description of an adaptive filter structure, we are interested in determining the best *linear relationship* between the input and desired response signals for many problems. This relationship typically takes the form of a *finite-impulse-response* (FIR) or *infinite-impulse-response* (IIR) filter. Fig3.2 shows the structure of a direct-form FIR filter, also known as a *tapped-delay-line* or *transversal filter*, where z^{-1} denotes the unit delay element and each $w_i(n)$ is a multiplicative gain within the system. In this case, the parameters in W(n) correspond to the impulse response values of the filter at time n. We can write the output signal y(n) as

$$y(n) = \sum_{i=0}^{L-1} w_i(n)x(n-i) \tag{3.4}$$

$$= W^T(n)X(n) \tag{3.5}$$

where $X(n) = [x(n), x(n-1)...x(n-L+l)]^T$ denotes the *input signal vector* and T denotes vector transpose. Note that this system requires L multiplies and L - 1 adds to implement and these computations are easily performed by a processor or circuit so long as L is not too large and the sampling period for the signals is not too short. It also requires a total of 2L memory locations to store the L input signal samples and the L coefficient values, respectively.

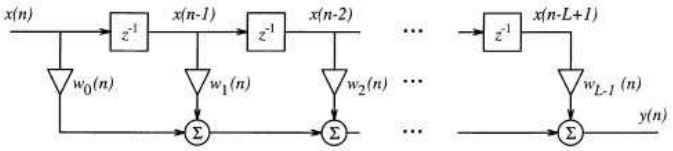


Figure 3.2: Structure of an FIR filter

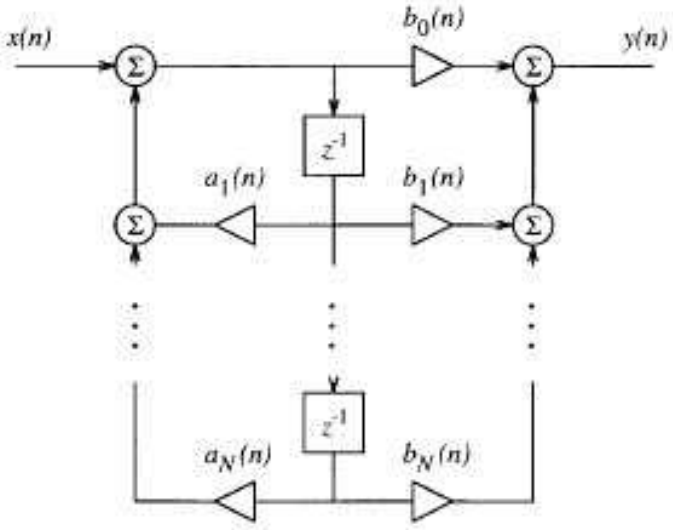


Figure 3.3: Structure of an IIR filter.

The structure of a direct-form IIR filter is shown in Fig. 3.3. In this case, the output of the system can be mathematically represented as

$$y(n) = \sum_{i=1}^{N} a_i(n)y(n-i) + \sum_{j=0}^{N} b_j(n)x(n-j) \tag{3.6}$$

Thus, for purposes of computing the output signal y(n), the IIR structure involves a fixed number of multiplies, adds, and memory locations not unlike the direct-form FIR structure. A third structure that has proven useful for adaptive filtering tasks is the *lattice filter*. A lattice filter is an FIR structure that employs L - 1 stages of preprocessing to compute a set of auxiliary signals $b_i(n), 0 < i < L-1$ known as *backward prediction errors*. These signals have the special property that they are *uncorrelated,* and they represent the elements of X(n) through a *linear transformation*. Thus, the backward prediction errors can be used in place of the delayed input signals in a structure similar to that in Fig.3.2 , and the uncorrelated nature of the prediction errors can provide improved convergence performance of

the adaptive filter coefficients with the proper choice of algorithm. Details of the lattice structure and its capabilities are discussed in [29].

A critical issue in the choice of an adaptive filter's structure is its computational complexity. Since the operation of the adaptive filter typically occurs in real time, all of the calculations for the system must occur during one sample time. The structures described above are all useful because y(n) can be computed in a finite amount of time using simple arithmetical operations and finite amounts of memory.

3.3 THE TASK OF AN ADAPTIVE FILTER

When considering the adaptive filter problem as illustrated in Fig.3.1 for the first time, a reader is likely to ask, "If we already have the desired response signal, what is the point of trying to match it using an adaptive filter?" In fact, the concept of "matching" y(n) to d(n) with some system obscures the subtlety of the adaptive filtering task. Consider the following issues that pertain to many adaptive filtering problems:

- ***In practice, the quantity of interest is not always d(n)***. Our desire may be to represent in y(n) a certain component of d(n) that is contained in x(n), or it may be to isolate a component of d(n) within the error e(n) that is *not* contained in x(n). Alternatively, we may be solely interested in the values of the parameters in W(n) and have no concern about x(n), y(n), or d(n) themselves. Practical examples of each of these scenarios are provided later in this chapter.

- ***There are situations in which d(n) is not available at all times***. In such situations, adaptation typically occurs only when d(n) is available. When d(n) is unavailable, we typically use our most-recent parameter estimates to compute y (n) in an attempt to *estimate* the desired response signal d(n).

- ***There are real-world situations in which d(n) is never available***. In such cases, one can use additional information about the characteristics of

a "hypothetical" d(n), such as its predicted statistical behavior or amplitude characteristics, to form suitable estimates of d(n) from the signals available to the adaptive filter. Such methods are collectively called *blind adaptation algorithms*. The fact that such schemes even work is a tribute both to the ingenuity of the developers of the algorithms and to the technological maturity of the adaptive filtering field.

It should also be recognized that the relationship between x(n) and d(n) can vary with time. In such situations, the adaptive filter attempts to alter its parameter values to follow the changes in this relationship as "encoded" by the two sequences x(n) and d(n). This behavior is commonly referred to as *tracking*.

3.4 APPLICATIONS OF ADAPTIVE FILTERS

Perhaps the most important driving forces behind the developments in adaptive filters throughout their history have been the wide range of applications in which such systems can be used. We now discuss the forms of these applications in terms of more-general problem classes that describe the assumed relationship between d(n) and x(n). Our discussion illustrates the key issues in selecting an adaptive filter for a particular task.

3.4.1 DIRECT MODELLING (SYSTEM IDENTIFICATION)

Consider Fig. 3.4, which shows the general problem of *system identification*. In this diagram, the system enclosed by dashed lines is a "black box," meaning that the quantities inside are not observable from the outside. Inside this box is

1. an unknown system which represents a general input-output relationship and

2. the signal $\eta_i(n)$, called the observation noise signal because it corrupts the observations of the signal at the output of the unknown system.

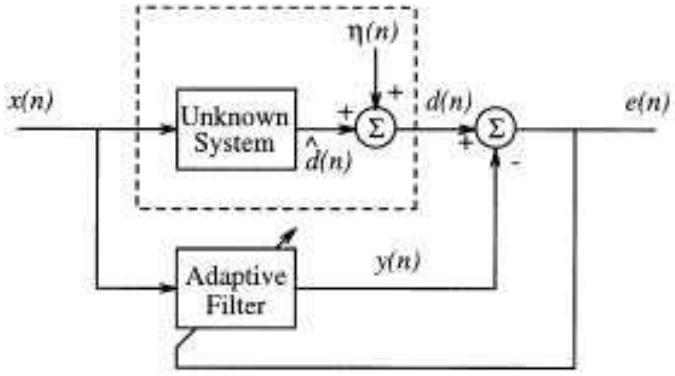


Figure 3.4: System Identification.

Let d(n) represent the output of the unknown system with x(n) as its input. Then, the desired response signal in this model is

$$d(n) = \hat{d}(n) + \eta(n) \tag{3.7}$$

Here, the task of the adaptive filter is to accurately represent the signal d(n) at its output. If y(n) = d (n), then the adaptive filter has accurately modeled or identified the portion of the unknown system that is driven by x(n).

Since the model typically chosen for the adaptive filter is a linear filter, the practical goal of the adaptive filter is to determine the best linear model that describes the input-output relationship of the unknown system. Such a procedure makes the most sense when the unknown system is also a linear model of the same structure as the adaptive filter, as it is possible that y(n) = d(n) for some set of adaptive filter parameters. For ease of discussion, let the unknown system and the adaptive filter both be FIR filters, such that

$$d(n) = W_{OPT}^{T}(n)X(n) + \eta(n) \tag{3.8}$$

where $W_{OPT}(n)$ is an optimum set of filter coefficients for the unknown system at time n. In this problem formulation, the ideal adaptation procedure would adjust W(n) such that $W(n) = W_{OPT}$ (n) as $n \longrightarrow \infty$. In practice, the adaptive filter can only adjust W(n) such that y(n) closely approximates d(n) over time. The system identification task is at the heart of numerous adaptive filtering

applications. We list several of these applications here [30].

- Channel Identification [31]
- Plant Identification [32]
- Echo Cancellation for Long-Distance Transmission [33]
- Acoustic Echo Cancellation [34]
- Adaptive Noise Cancelling [35]

3.4.2 INVERSE MODELLING

We now consider the general problem of *inverse modelling*, as shown in Fig.3.5 In this diagram, a *source signals* s(n) is fed into an unknown system that produces the input signal x(n) for the adaptive filter. The output of the adaptive filter is subtracted from a desired response signal that is a delayed version of the source signal, such that

$$d(n) = s(n - \triangle) \tag{3.9}$$

where $\triangle$ is a positive integer value. The goal of the adaptive filter is to adjust its characteristics such that the output signal is an accurate representation of the delayed source signal.

3.4.3 CHANNEL EQUALIZATION

Channel equalization is an alternative to the technique of channel identification described previously for the decoding of transmitted signals across non-ideal communication channels. In both cases, the transmitter sends a sequence s(n) that is known to both the transmitter and receiver. However, in equalization, the received signal is used as the input signal x(n) to an adaptive filter, which adjusts its characteristics so that its output closely matches a delayed version $s(n - \triangle)$ of the known transmitted signal. After a suitable adaptation period, the coefficients of the system either are fixed and used to decode future transmitted messages or

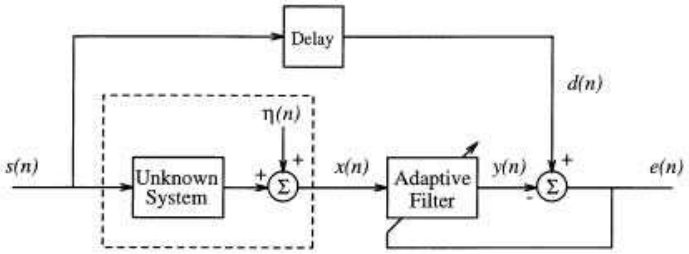


Figure 3.5: Inverse Modelling

are adapted using a crude estimate of the desired response signal that is computed from y(n). This latter mode of operation is known as *decision-directed adaptation*.

Channel equalization was one of the first applications of adaptive filters and is described in the pioneering work of Lucky [36]. Today, it remains as one of the most popular uses of an adaptive filter. Practically every computer telephone modem transmitting at rates of 9600 *baud* (bits per second) or greater contains an adaptive equalizer. Adaptive equalization is also useful for wireless communication systems. Qureshi [2] provides a tutorial on adaptive equalization. A related problem to equalization is *deconvolution*, a problem that appears in the context of geophysical exploration [37].

3.5 GRADIENT-BASED ADAPTIVE ALGORITHMS

An adaptive algorithm is a procedure for adjusting the parameters of an adaptive filter to minimize a cost function chosen for the task at hand. In this section, we describe the general form of many adaptive FIR filtering algorithms and present a simple derivation of the LMS adaptive algorithm. In our discussion, we only consider an adaptive FIR filter structure, such that the output signal y(n) is given by (2.8). Such systems are currently more popular than adaptive IIR filters because

1. the input-output stability of the FIR filter structure is guaranteed for any set of fixed coefficients, and

2. the algorithms for adjusting the coefficients of FIR filters are simpler in general than those for adjusting the coefficients of IIR filters.

3.5.1 GENERAL FORM OF ADAPTIVE FIR ALGORITHMS

The general form of an adaptive FIR filtering algorithm is

$$W(n+1) = W(n) + \mu(n)G(e(n), X(n), \phi(n)) \tag{3.10}$$

where G(-) is a particular vector-valued nonlinear function, $\mu(n)$ is a *step size* parameter, e(n) and X(n) are the error signal and input signal vector, respectively, and $\Phi(n)$ is a vector of states that store pertinent information about the characteristics of the input and error signals and/or the coefficients at previous time instants. In the simplest algorithms, $\Phi(n)$ is not used, and the only information needed to adjust the coefficients at time n are the error signal, input signal vector, and step size.

The step size is so called because it determines the magnitude of the change or "step" that is taken by the algorithm in iteratively determining a useful coefficient vector. Much research effort has been spent characterizing the role that $\mu(n)$ plays in the performance of adaptive filters in terms of the statistical or frequency characteristics of the input and desired response signals. Often, success or failure of an adaptive filtering application depends on how the value of $\mu(n)$ is chosen or calculated to obtain the best performance from the adaptive filter.

3.5.2 THE MEAN-SQUARED ERROR COST FUNCTION

The form of G(-) in (3.9) depends on the cost function chosen for the given adaptive filtering task. We now consider one particular cost function that yields a popular adaptive algorithm. Define the *mean-squared error* (MSE) cost function as

$$J_{MSE}(n) = \frac{1}{2}\int_{-\infty}^{\infty} e^2(n)P_n(e(n))de(n) \tag{3.11}$$

$$= \frac{1}{2}Ee^2(n) \tag{3.12}$$

where $p_n(e)$ represents the probability density function of the error at time n and E- is shorthand for the *expectation integral* on the right-hand side of (3.12).

The MSE cost function is useful for adaptive FIR filters because

- $J_{MSE}(N)$ has a well-defined minimum with respect to the parameters in W(n)
- the coefficient values obtained at this minimum are the ones that minimize the power in the error signal e(n), indicating that y(n) has approached d(n); and
- J_{MSE} is a smooth function of each of the parameters in W(n), such that it is differentiable with respect to each of the parameters in W(n).

The third point is important in that it enables us to determine both the optimum coefficient values given knowledge of the statistics of d(n) and x(w) as well as a simple iterative procedure for adjusting the parameters of an FIR filter.

3.5.3 THE WIENER SOLUTION

For the FIR filter structure, the coefficient values in W(n) that minimize $J_{MSE}(n)$ are well-defined if the statistics of the input and desired response signals are known. The formulation of this problem for continuous-time signals and the resulting solution was first derived by Wiener [38]. Hence, this optimum coefficient vector $W_{MSE}(n)$ is often called the *Wiener solution* to the adaptive filtering problem. The extension of Wiener's analysis to the discrete-time case is attributed to Levinson [39]. To determine $W_{MSE}(N)$ we note that the function $J_{MSE}(N)$ in (3.12) is quadratic in the parameters $\{w_i(n)\}$, and the function is also differentiable. Thus, we can use a result from optimization theory that states that the derivatives of a smooth cost function with respect to each of the parameters is zero at a minimizing point on the cost function error surface. Thus, $W_{MSE}(n)$ can be found from the solution to the system of equations

$$\frac{\delta J_{MSE}(n)}{\delta w_i(n)} = 0, 0 \leq (i) \leq (L) \tag{3.13}$$

Taking derivatives of $J_{MSE}(N)$ in (3.12) and noting that e(n) and y(n) are given by (3.1) and (3.5), respectively, we obtain

$$\frac{\delta J_{MSE}(n)}{\delta w_i(n)} = E[e(n)\frac{\delta(e(n))}{\delta w_i(n)}] \tag{3.14}$$

$$= -E[e(n)\frac{\delta y(n)}{\delta w_i(n)}] \tag{3.15}$$

$$= -E[e(n)x(n-i)] \tag{3.16}$$

$$\begin{aligned} = -(E[d(n)x(n-i)] - \sum_{j=0}^{L-1} \\ E[x(n-i)x(n-j)w_j(n)] \end{aligned} \tag{3.17}$$

where we have used the definitions of e(n) and of y(n) for the FIR filter structure in (3.1) and (3.5), respectively, to expand the last result in (3.17). By defining the matrix $R_{XX}(n)$ and vector $P_{dx}(n)$ as

$$\begin{aligned} R_{XX} = \quad & E[X(n)X^T(n)] \\ and & \\ P_{dx}(n) = \quad & E[d(n).X(n)] \end{aligned} \tag{3.18}$$

respectively, we can combine (3.13) and (3.17) to obtain the system of equations in vector form as

$$R_{XX}(n)W_{MSE}(n) - P_{dx}(n) = 0 \tag{3.19}$$

where 0 is the zero vector. Thus, so long as the matrix $R_{XX}(n)$ is invertible, the optimum Wiener solution vector for this problem is

$$W_{MSE}(n) = R_{XX}^{-1}(n)P_{dx}(n) \tag{3.20}$$

3.5.4 THE METHOD OF STEEPEST DESCENT

The method of steepest descent is a celebrated optimization procedure for minimizing the value of a cost function J(n) with respect to a set of adjustable parameters W(n). This procedure adjusts each parameter of the system according

to

$$w_i(n+1) = w_i(n) - \mu(n)\frac{\delta J(n)}{\delta(w_i(n))} \tag{3.21}$$

In other words, the i^{th} parameter of the system is altered according to the derivative of the cost function with respect to the i^{th} parameter. Collecting these equations in vector form, we have

$$W(n+1) = W(n) - \mu(n)\frac{\delta J(n)}{\delta W(n)} \tag{3.22}$$

where $\frac{\delta J(n)}{\delta W(n)}$ is a vector of derivatives $\frac{dJ(n)}{dW_i(n)}$.

For an FIR adaptive filter that minimizes the MSE cost function, we can use the result in (3.17) to explicitly give the form of the steepest descent procedure in this problem. Substituting these results into (3.21) yields the update equation for W(n) as

$$W(n+1) = W(n) + \mu(n)(P_{dx}(n) - R_{XX}(n)W(n)) \tag{3.23}$$

However, this steepest descent procedure depends on the statistical quantities $E\{d(n)x(n-i)\}$ and $E\{x(n-i)x(n-j)\}$ contained in $P_{dx}(n)$ and $R_{xx}(n)$, respectively. In practice, we only have measurements of both d(n) and x(n) to be used within the adaptation procedure. While suitable estimates of the statistical quantities needed for (3.23) could be determined from the signals x(n) and d(n), we instead develop an approximate version of the method of steepest descent that depends on the signal values themselves. This procedure is known as the *LMS algorithm.*

3.6 THE LMS ALGORITHM

The cost function J(n) chosen for the steepest descent algorithm of (3.21) determines the coefficient solution obtained by the adaptive filter. If the MSE cost function in (3.12) is chosen, the resulting algorithm depends on the statistics of x(n) and d(n) because of the expectation operation that defines this cost function. Since we typically only have measurements of d(n) and of x(n) available to us, we substitute an alternative cost function that depends only on these measurements.

One such cost function is the least-squares cost function given by

$$J_{LS}(n) = \sum_{k=0}^{n} \alpha(k)(d(k) - W^T(n)X(k))^2 \tag{3.24}$$

where a(n) is a suitable weighting sequence for the terms within the summation. This cost function, however, is complicated by the fact that it requires numerous computations to calculate its value as well as its derivatives with respect to each W(n), although efficient recursive methods for its minimization can be developed. Alternatively, we can propose the simplified cost function $J_{LMS}(N)$ given by

$$J_{LSM}(n) = \frac{1}{2}e^2(n) \tag{3.25}$$

This cost function can be thought of as an instantaneous estimate of the MSE cost function, as $J_{MSE}(n) = E_{JLMS}(n)$. Although it might not appear to be useful, the resulting algorithm obtained when $J_{LMS}(N)$ is used for J(n) in (3.21) is extremely useful for practical applications. Taking derivatives of $J_{LMS}(n)$ with respect to the elements of W(n) and substituting the result into (3.21), we obtain the LMS adaptive algorithm given by

$$W(n+1) = W(n) + \mu(n)e(n)X(n) \tag{3.26}$$

Note that this algorithm is of the general form in (3.10). It also requires only multiplications and additions to implement. In fact, the number and type of operations needed for the LMS algorithm is nearly the same as that of the FIR filter structure with fixed coefficient values, which is one of the reasons for the algorithm's popularity.

The behavior of the LMS algorithm has been widely studied, and numerous results concerning its adaptation characteristics under different situations have been developed. For now, we indicate its useful behavior by noting that the solution obtained by the LMS algorithm near its convergent point is related to the Wiener solution. In fact, analyses of the LMS algorithm under certain statistical

assumptions about the input and desired response signals show that

$$\lim_{n\to\infty} E[W(n)] = W_{MSE} \tag{3.27}$$

when the Wiener solution $W_{MSE}(n)$ is a fixed vector. Moreover, the average behavior of the LMS algorithm is quite similar to that of the steepest descent algorithm in (3.23) that depends explicitly on the statistics of the input and desired response signals. In effect, the iterative nature of the LMS coefficient updates is a form of time-averaging that smoothes the errors in the instantaneous gradient calculations to obtain a more reasonable estimate of the true gradient.

The problem is that gradient descent is a local optimization technique, which is limited because it is unable to converge to the global optimum on a multimodal error surface if the algorithm is not initialized in the basin of attraction of the global optimum.

Several modifications exist for gradient based algorithms in attempt to enable them to overcome local optima. One approach is to simply add noise or a momentum term [30] to the gradient computation of the gradient descent algorithm to enable it to be more likely to escape from a local minimum. This approach is only likely to be successful when the error surface is relatively smooth with minor local minima, or some information can be inferred about the topology of the surface such that the additional gradient parameters can be assigned accordingly. Other approaches attempt to transform the error surface to eliminate or diminish the presence of local minima [40], which would ideally result in a unimodal error surface. The problem with these approaches is that the resulting minimum transformed error used to update the adaptive filter can be biased from the true minimum output error and the algorithm may not be able to converge to the desired minimum error condition. These algorithms also tend to be complex, slow to converge, and may not be guaranteed to emerge from a local minimum. Some work has been done with regard to removing the bias of equation error LMS [40,41] and Steiglitz-McBride [42] adaptive IIR filters, which add further complexity with varying degrees of success.

Another approach [43] , attempts to locate the global optimum by running several LMS algorithms in parallel, initialized with different initial coefficients. The notion is that a larger, concurrent sampling of the error surface will increase the likelihood that one process will be initialized in the global optimum valley. This technique does have potential, but it is inefficient and may still suffer the fate of a standard gradient technique in that it will be unable to locate the global optimum if none of the initial estimates is located in the basin of attraction of the global optimum. By using a similar congregational scheme, but one in which information is collectively exchanged between estimates and intelligent randomization is introduced, structured stochastic algorithms are able to hill-climb out of local minima. This enables the algorithms to achieve better, more consistent results using a fewer number of total estimates. These types of algorithms provide the framework for the algorithms discussed in the following sections.

3.7 THE RLS ALGORITHM

The RLS (recursive least squares) algorithm is another algorithm for determining the coefficients of an adaptive filter. In contrast to the LMS algorithm, the RLS algorithm uses information from all past input samples (and not only from the current tap-input samples) to estimate the (inverse of the) autocorrelation matrix of the input vector. To decrease the influence of input samples from the far past, a weighting factor for the influence of each sample is used. This weighting factor is introduced in the cost function

$$J[n] = \sum_{i=1}^{n} \rho^{n-i} |e[i,n]|^2 \tag{3.28}$$

where the error signal $e_i[i,n]$ is computed for all times $1 \leq$ i $\leq$ n using the current filter coefficients $c[n]$:$e[i,n] = d[i] - c^T[n]x[i]$, where $x[i]$ and c^T represents input signal and transpose of the channel coefficient vector respectively.

When ρ =1, the squared error for all sample times i up to current time n is considered in the cost function J equally. If $0 < \rho < 1$ the influence of past error values decays exponentially: method of *exponentially weighted least squares.* is

called the *forgetting factor*.

Analogous to the derivation of the LMS algorithm we find the gradient of the cost function with respect to the current weights

$$\Delta_c J[n] = \sum_{i=1}^{n} \rho^{n-i}(-2E(d[i]x[i]) + 2E(x[i]x^T[i])c[n])) \tag{3.29}$$

where x^T represents the transpose of the input signal vector.We now, however, d_o trust in the ability to estimate the expected values $E(dx) = p$ and $E(x, x^T)$=R with sufficient accuracy using all past samples, and do not use a gradient descent method, but immediately search for the minimum of the cost function by setting its gradient to zero $\nabla_c J[n] = 0$. The resulting equation for the optimum filter coefficients at time n is

$$\begin{aligned} \phi(n)c[n] &= z[n] \\ c[n] &= \phi^{-1}[n]z[n] \end{aligned} \tag{3.30}$$

with $\phi[n] = \sum_{i=1}^{n} \rho^{n-i}x[i]x^T[i]$, and $z[n]$=$\sum_{i=1}^{n} \rho^{n-i}d^*x^T[i]$ Both $\phi[n]$ and $z[n]$ can be computed recursively:

$$\phi[n] = \rho\phi[n-1] + x[n]x^T[n] \tag{3.31}$$

and

$$z[n] = \rho z[n-1] + d^\dagger[n]x[n] \tag{3.32}$$

To find c[n] the coefficient vector we, however, need the inverse matrix $\phi^{-1}[n]$. Using a matrix inversion lemma [44] a recursive update equation for P[n]=$\phi^{-1}[n]$ is found as:

$$\begin{aligned} P[n] &= \rho^{-1}P[n-1] + \rho^{-1}k[n]x[n] \\ &\qquad with \\ k[n] &= \frac{\rho^{-1}P[n-1]x[n]}{1 + \rho^{-1}x^T[n]P[n-1]x[n]} \end{aligned} \tag{3.33}$$

Finally, the weights update equation is

$$c[n] = c[n-1] + k[n](d^\dagger[n] - x^T[n]c[n-1]) \tag{3.34}$$

The equations to solve in the RLS algorithm at each time step are (3.33) and (3.34). The RLS algorithm is computationally more complex than the LMS algorithm.

Note, however, that due the recursive updating the inversion of matrix $\phi[n]$ is not necessary (which would be a considerably higher computational load). The RLS algorithm typically shows a faster convergence compared to the LMS algorithm.

3.8 ARTIFICIAL NEURAL NETWORK (ANN)

Artificial neural network (ANN) takes their name from the network of nerve cells in the brain. Recently, ANN has been found to be an important technique for classification and optimization problem [45, 46] . McCulloch and Pitts have developed the neural networks for different computing machines. There are extensive applications of ANN in the field of channel equalization, estimation of parameters of nonlinear systems , pattern recognition , etc. ANN is capable of performing nonlinear mapping between the input and output space due to its large parallel interconnection between different layers and the nonlinear processing characteristics. An artificial neuron basically consists of a computing element that performs the weighted sum of the input signal and the connecting weight. The sum is added with the bias or threshold and the resultant signal is then passed through a nonlinear function of sigmoid or hyperbolic tangent type. Each neuron is associated with three parameters whose learning can be adjusted; these are the connecting weights, the bias and the slope of the nonlinear function. For the structural point of view a NN may be single layer or it may be multilayer. In multilayer structure, there is one or many artificial neurons in each layer and for a practical case there may be a number of layers. Each neuron of the one layer is connected to each and every neuron of the next layer. The Functional link ANN is another type of single layer NN. In this type of network the input data is allowed to pass through a functional expansion block where the input data are nonlinearly mapped to more number of points. This is achieved by using trigonometric functions, tensor products or power terms of the input. The output of the functional expansion is then passed through a single neuron.

The learning of the NN may be supervised in the presence of the desired signal or it may be unsupervised when the desired signal is not accessible. Rumelhart

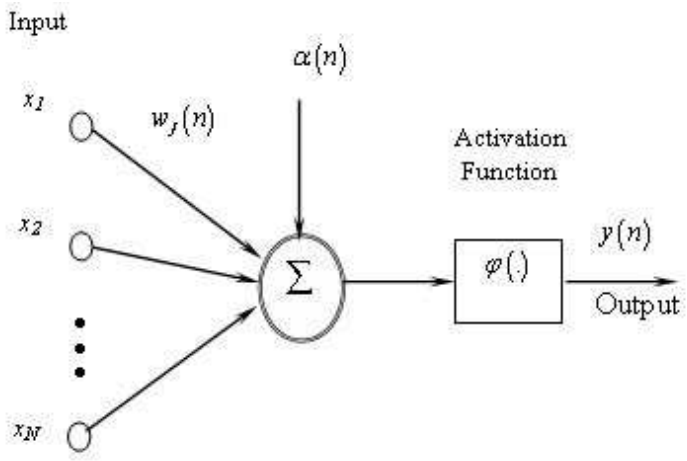


Figure 3.6: Structure of a Single Neuron

developed the Back propagation algorithm, which is central to much work on supervised learning in multilayer NN. A feed forward structure with input, output, hidden layers and nonlinear sigmoid functions are used in this type of network. In recent years many different types of learning algorithm using the incremental back propagation algorithm , evolutionary learning using the nearest neighbor MLP and a fast learning algorithm based on the layer-by-layer optimization procedure are suggested in literature. In case of unsupervised learning the input vectors are classified into different clusters such that elements of a cluster are similar to each other in some sense. The method is called competitive learning , because during learning sets of hidden units compete with each other to become active and perform the weight change. The winning unit increases its weights on those links with high input values and decreases them on those with low input values. This process allows the winning unit to be selective to some input values. Different types of NNs and their learning algorithms are discussed below.

3.8.1 SINGLE NEURON STRUCTURE

The basic structure of an artificial neuron is presented in Fig. 3.6. The operation in a neuron involves the computation of the weighted sum of inputs and threshold. The resultant signal is then passed through a nonlinear activation function. This is also called as a perceptron, which is built around a nonlinear neuron;

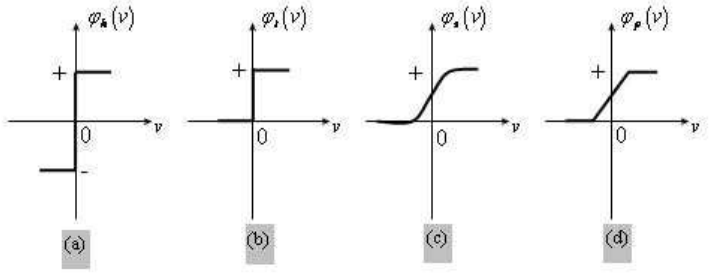


Figure 3.7: Different Types of Non-Linear Activation Function

whereas the LMS algorithm described in the preceding sections is built around a linear neuron. The output of the neuron may be represented as,

$$y(n) = \phi\Big[\sum_{j=1}^{N} w_j(n)x_j(n) \quad + \quad \alpha(n)\Big] \tag{3.35}$$

where $\alpha(n)$ is the threshold to the neurons at the first layer, $w_j(n)$ is the weight associated with the j^{th} input, N is the no. of inputs to the neuron and $\phi(.)$ is the nonlinear activation function. Different types of nonlinear function are shown in Fig.(3.7)[1]. Signum Function: For this type of activation function, we have

$$\phi(v) = \begin{cases} 1, if\ v > 0 \\ 0, if\ v = 0 \\ -1, if\ v < 0 \end{cases} \tag{3.36}$$

Threshold Function: This function is represented as,

$$\phi(v) = \begin{cases} 1, if\ v \geq 0 \\ 0, if\ v < 0 \end{cases} \tag{3.37}$$

Sigmoid Function: This function is s-shaped, is the most common form of the activation function used in artificial neural network. It is a function that exhibits a graceful balance between linear and nonlinear behaviour.

$$\phi(v) = \frac{1}{1 + e^{-av}} \tag{3.38}$$

[1](a) Signum function or hard limiter, (b) Threshold function, (c) Sigmoid function, (d) Piece-wise Linear

where v is the input to the sigmoid function and a is the slope of the sigmoid function. For the steady convergence a proper choice of a is required.

Piecewise-Linear Function: This function is

$$\phi(v) = \begin{cases} 1, v \geq +1/2 \\ v, +1/2 > v > +1/2 \\ 0, v \leq +1/2 \end{cases} \tag{3.39}$$

where the amplification factor inside the linear region of operation is assumed to be unity. This can be viewed as an approximation to a nonlinear amplifier.

3.9 MULTILAYER PERCEPTRON (MLP)

In the multilayer neural network or multilayer perceptron (MLP), the input signal propagates through the network in a forward direction, on a layer-by-layer basis. This network has been applied successfully to solve some difficult and diverse problems by training in a supervised manner with a highly popular algorithm known as the error back-propagation algorithm [47–50]. The scheme of MLP using four layers is shown in Fig3.8. $x_i(n)$ represents the input to the network, f_j and f_k represent the output of the two hidden layers and $y_l(n)$ represents the output of the final layer of the neural network. The connecting weights between the input to the first hidden layer, first to second hidden layer and the second hidden layer to the output layers are represented w_{ij},w_{jk} and w_{kl} by respectively.

If P_1 is the number of neurons in the first layer, each element of the output vector may be calculated as,

$$f_j = \varphi_j \sum_{i=1}^{N} [w_{ij} x_i(n) + \alpha_j], j = 1, 2, 3...P_1 \tag{3.40}$$

where α_j is the threshold to the neurons at the first layer, N is the no. of inputs and φ. is the nonlinear activation function. The time index n has been dropped to make the equations simpler. Let P_2 be the number of neurons in the second layer. The output of this layer is represented as, f_k and may be written as

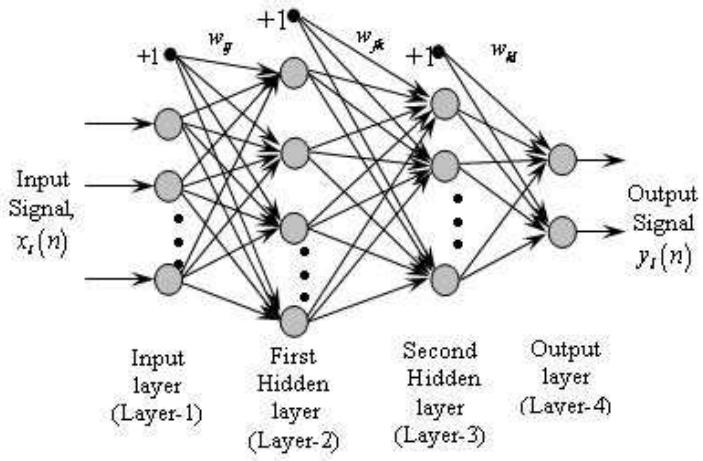


Figure 3.8: Structure of Multilayer perceptron (MLP)

$$f_k = \varphi_k \sum_{j=1}^{P_1} [w_{jk} f_j + \alpha_k], j = 1, 2, 3...P_2 \tag{3.41}$$

where, α_k is the threshold to the neurons at the second layer. The output of the final layer can be calculated as

$$y_l(n) = \varphi_l \sum_{k=1}^{P_2} [w_{kl} f_k + \alpha_l], j = 1, 2, 3...P_3 \tag{3.42}$$

where, α_l is the threshold to the neuron at the final layer and P_3 is the no. of neurons in the output layer. The output of the MLP may be expressed as

$$y_l(n) = \varphi_n[\sum_{k=1}^{P_2} w_{kl} \varphi_k [\sum_{j=1}^{P_1} w_{jk} \varphi_j [\sum_{i=1}^{N} w_{ij} x_i(n) + \alpha_j] + \alpha_k] + \alpha_l] \tag{3.43}$$

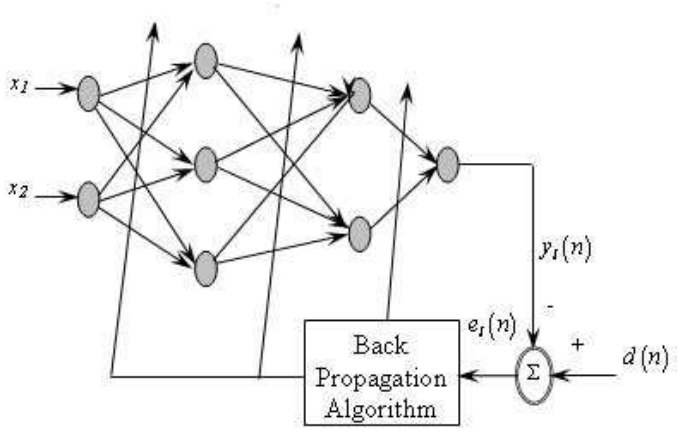


Figure 3.9: Neural Network Training Using BP Algorithm

3.10 Back-propagation (BP) Algorithm

An MLP network with 2-3-2-1 neurons (2, 3, 2 and 1 denote the number of neuron in the input layer, the first hidden layer, the second hidden layer and the output layer respectively) with the back-propagation (BP) learning algorithm, is depicted in Fig.3.9. The parameters of the neural network can be updated in both sequential and batch mode of operation. In BP algorithm, initially the weights and the thresholds are initialized as very small random values. The intermediate and the final outputs of the MLP are calculated by using (3.40), (3.41), and (3.42).

The final output $y_l(n)$ at the output of neuron l , is compared with the desired output $d(n)$ and the resulting error signal $e_l(n)$ is obtained as

$$e_l(n) = d(n) - y_l(n) \tag{3.44}$$

The instantaneous value of the total error energy is obtained by summing all error signals over all neurons in the output layer, that is

$$\xi(n) = \frac{1}{2}\sum_{l=1}^{P_3} e_l^2(n) \tag{3.45}$$

where P_3 is the no. of neurons in the output layer.
This error signal is used to update the weights and thresholds of the hidden layers as well as the output layer. The reflected error components at each of the hidden

layers is computed using the errors of the last layer and the connecting weights between the hidden and the last layer and error obtained at this stage is used to update the weights between the input and the hidden layer. The thresholds are also updated in a similar manner as that of the corresponding connecting weights. The weights and the thresholds are updated in an iterative method until the error signal becomes minimum. For measuring the degree of matching, the mean square error (MSE) is taken as a performance measurement.

The updated weights are,

$$w_{kl}(n+1) = w_{kl}(n) + \Delta w_{kl}(n) \tag{3.46}$$

$$w_{jk}(n+1) = w_{jk}(n) + \Delta w_{jk}(n) \tag{3.47}$$

$$w_{ij}(n+1) = w_{ij}(n) + \Delta w_{ij}(n) \tag{3.48}$$

where, $\Delta w_{kl}(n)$,$\Delta w_{jk}(n)$ and $\Delta w_{ij}(n)$ are the change in weights of the output, hidden and input layer respectively. That is,

$$\Delta w_{kn}(n) = -2\mu \frac{d\xi(n)}{dw_{kl}(n)} = 2\mu e(n) \frac{dy_l(n)}{dw_{kl}(n)} = 2\mu e(n) \varphi_l'[\sum_{k=1}^{P_2} w_{kl} f_k + \alpha_l] f_k \tag{3.49}$$

Where, μ is the convergence coefficient ($0 \leq \mu \leq 1$). Similarly the can be computed .

The thresholds of each layer can be updated in a similar manner, that is

$$\alpha_l(n+1) = \alpha_l(n) + \Delta \alpha_l(n) \tag{3.50}$$

$$\alpha_k(n+1) = \alpha_k(n) + \Delta \alpha_k(n) \tag{3.51}$$

$$\alpha_j(n+1) = \alpha_j(n) + \Delta \alpha_j(n) \tag{3.52}$$

where, $\Delta\alpha_l(n)$,$\Delta\alpha_k(n)$ and $\Delta\alpha_j(n)$ are the change in thresholds of the output, hidden and input layer respectively. The change in threshold is represented as,

$$\Delta\alpha_l(n) = -2\mu\frac{d\xi(n)}{d\alpha_l(n)} = 2\mu e(n)\frac{dy_l(n)}{d\alpha_l(n)} = 2\mu e(n)\varphi_l'[\sum_{k=1}^{P_2} w_{kl}f_k + \alpha_l] \tag{3.53}$$

3.11 SIMULATION RESULTS

In the above sections the LT equalizer and its structure was described followed by its advantage and its training. The actual performance of equalizers was evaluated by computer simulation. During the simulation Bit Error Rate (BER) was used as the performance index. This section presents the BER performance of LT equalizers for a variety of parameters. The BER Vs SNR at receiver input was plotted for performance analysis.

Uniform random binary sequences of length 1000 were generated and transmitted through the channel. The channels were affected the ISI and AWGN. Output of the channel was fed to the equalizer and the detected samples at the equalizer were compared with suitable transmitted sample for BER evaluation.

The results of two different linear and nonlinear channels are used. While training, the additive noises used in the channel are -30dB (low noise), -10dB (medium noise) and 0dB (high noise) to test the performance of the three different algorithms in different noise conditions. Finally the performance of the equalizers is tested by plotting the Bit-error-rate (BER).
The following linear channel models are used [16]:

1. CH1: $H(z) = 0.2014 + 0.9586z^{-1} + 0.2014z^{-2}$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

2. CH2: $H(z) = 0.3040 + 0.9029z^{-1} + 0.3040z^{-2}$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

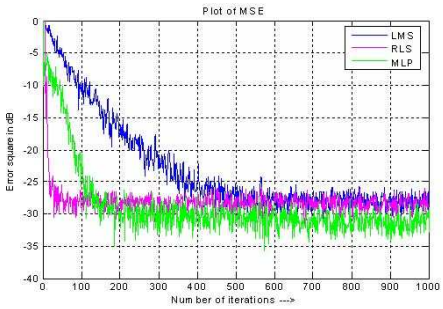
The following nonlinear channel models are used:

1. NCH1: $H(z) = 0.2014 + 0.9586z^{-1} + 0.2014z^{-2}$
   $b(k) = tanh[a(k)]$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

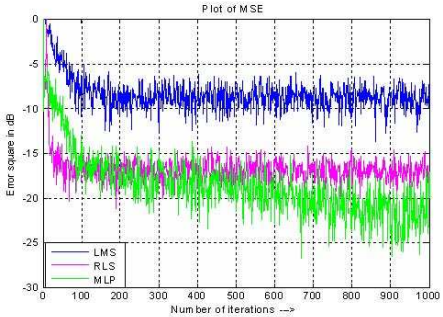
2. NCH1: $H(z) = 0.3040 + 0.9029z^{-1} + 0.3040z^{-2}$
   $b(k) = a(k) + 0.2a^2(k) - 0.1a^3(k)$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

Where b(k) is the output of the nonlinear channel.
The desired signal is generated by delaying the input binary sequence by m samples where $m = \frac{N}{2}$ or $\frac{(N+1)}{2}$ depending upon N is even or odd where N represents the order of the channel. In the simulation study N = 8 has been taken. For LMS algorithm, $\mu = 0.02$ and for RLS algorithm delta = 5000. Further a 3-6-1 MLP architecture is chosen for simulation. The learning rate for neural network architecture is 0.01.

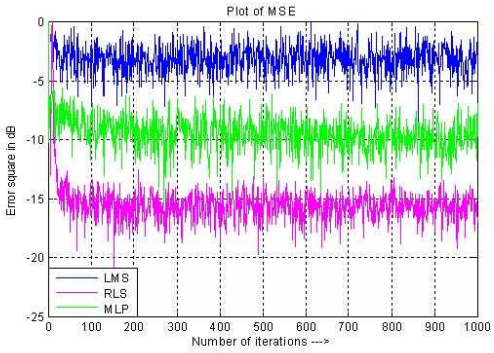
The convergence characteristics of LMS, RLS and Back propagation algorithm are obtained from simulation and is shown in Fig.3.10(a, b, c, d, e and f) and Fig. 3.11(a, b, c, d, e and f) for the linear channels and nonlinear channels respectively. Similarly the bit error plot (BER) for linear and nonlinear channels are shown in Fig. 3.12 (a, b, c, d, e and f) and Fig 3.13 (a, b, c, d, e and f) respectively. These results are used for comparison.



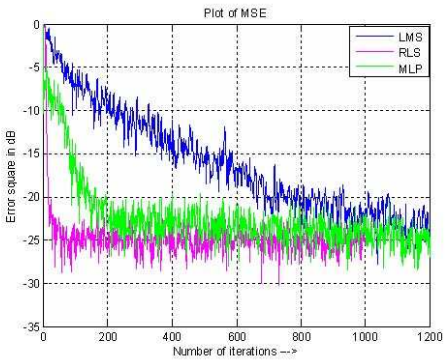
(a) CH1, SNR = 30dB



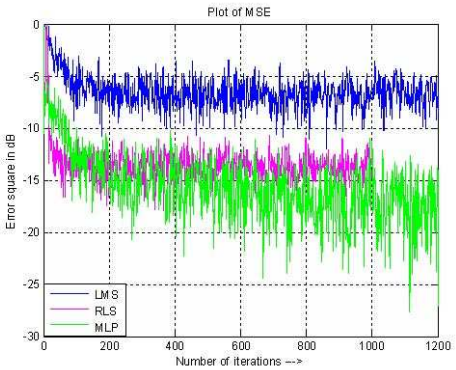
(b) CH1, SNR = 10dB



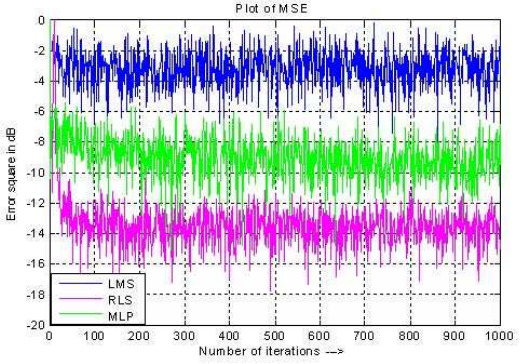
(c) CH1, SNR = 0dB



(d) CH2, SNR = 30dB

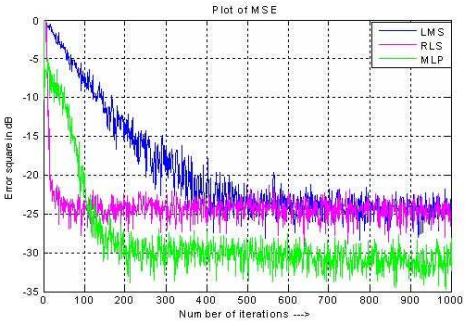


(e) CH2, SNR = 10dB

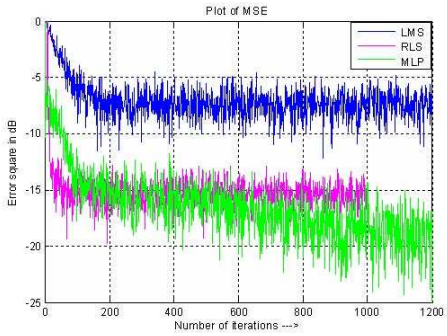


(f) CH2, SNR = 0dB

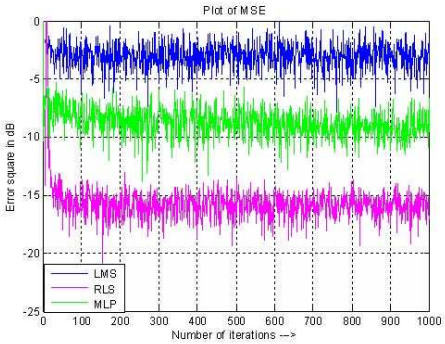
Figure 3.10: Plot of convergence characteristics of different linear channels at different noise conditions



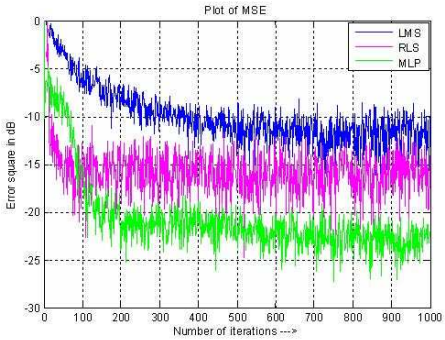
(a) NCH1, SNR = 30dB



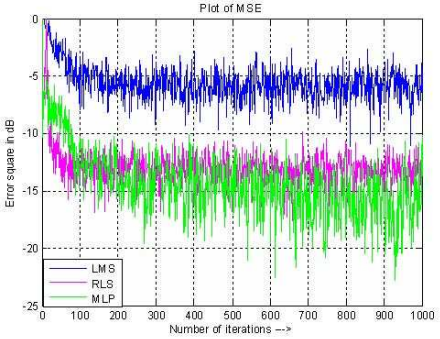
(b) NCH1, SNR = 10dB



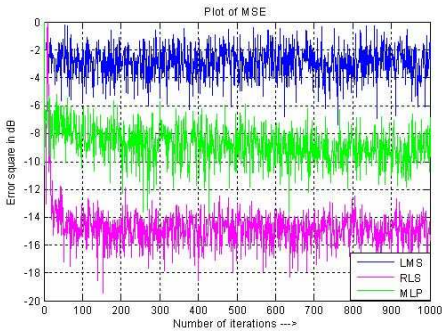
(c) NCH1, SNR = 0dB



(d) NCH2, SNR = 30dB

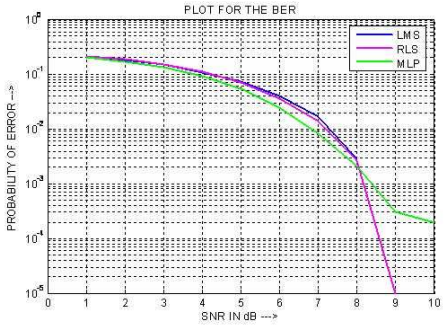


(e) NCH2, SNR = 10dB

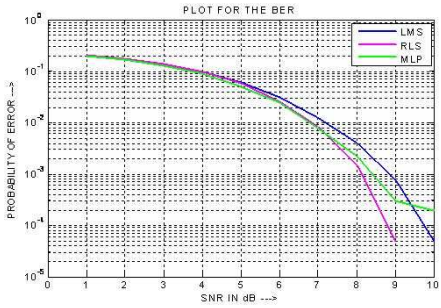


(f) NCH2, SNR = 0dB

Figure 3.11: Plot of convergence characteristics of different nonlinear channels at different noise conditions

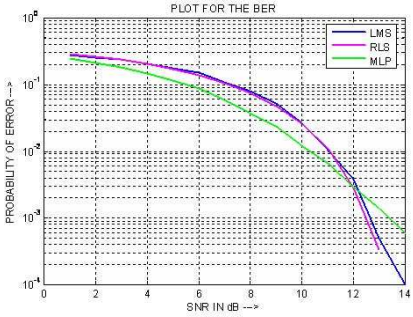


(a) CH1, SNR = 30dB

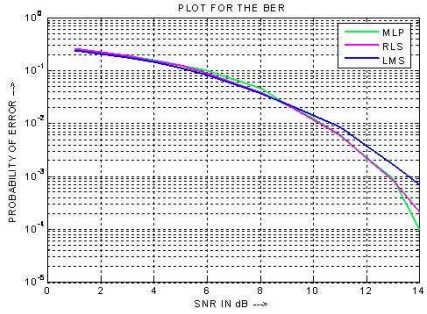


(b) CH1, SNR = 10dB

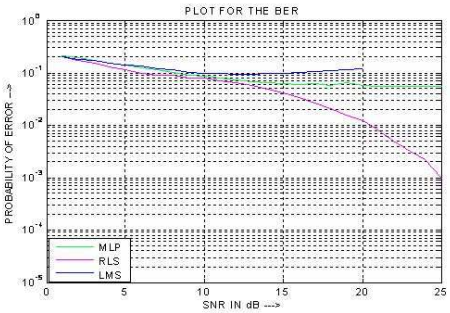
(c) CH1, SNR = 0dB



(d) CH2, SNR = 30dB

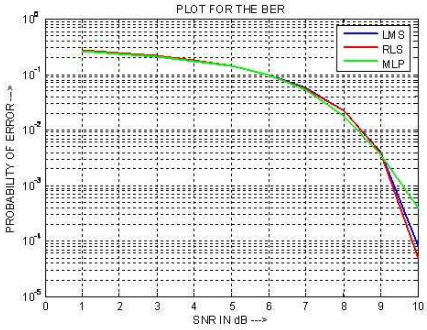


(e) CH2, SNR = 10dB

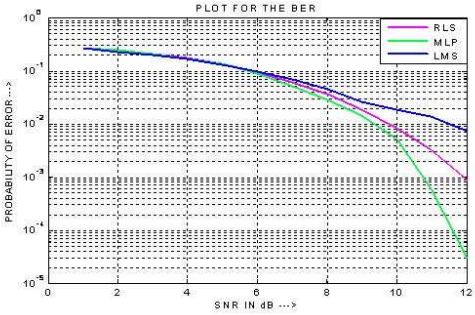


(f) CH2, SNR = 0dB

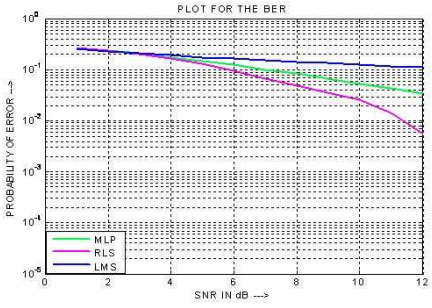
Figure 3.12: BER performance of LMS, RLS and MLP based equalizer for different linear channels at different noise conditions



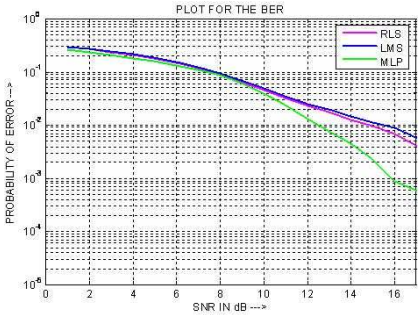
(a) NCH1, SNR = 30dB



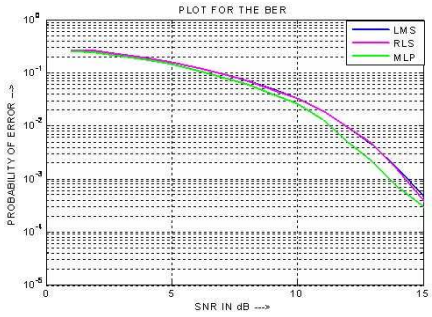
(b) NCH1, SNR = 10dB



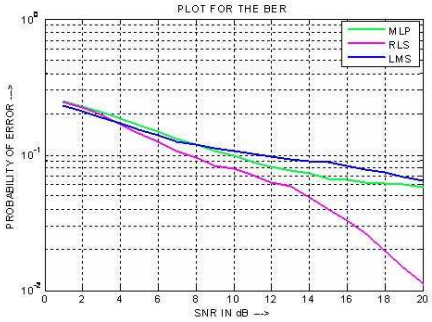
(c) NCH1, SNR = 0dB



(d) NCH2, SNR = 30dB



(e) NCH2, SNR = 10dB



(f) NCH2, SNR = 0dB

Figure 3.13: BER performance of LMS, RLS and MLP based equalizer for different nonlinear channels at different noise conditions

Table 3.1: Comparison of convergence rates of different algorithms

Algorithm	Number of samples consumed (To attain almost same MSE level)
RLS	20 - 30
MLP	200 - 300
LMS	500 - 1000

3.12 CONCLUSION

It is clear from the above section that RLS algorithm exhibits faster convergence rate compared to its counterparts.

From BER plots it can be concluded that:-

1. In most of the cases MLP exhibits superior performance than LMS and RLS algorithms.
2. RLS equalizer outperforms its counterparts under high noise conditions (when SNR = 0 dB)
3. For nonlinear channels the performance of MLP equalizer is better than the rest equalizer.

3.13 SUMMARY

This chapter introduced the concept of adaptive filtering. The different filter structures like FIR and IIR are also dealt in this chapter. Several applications of adaptive filters were also discussed within this chapter. It is seen that the channel equalization falls under the category of inverse modelling.

Three gradient based training methods LMS, RLS and Back Propagation were also explained in this chapter. The performances of adaptive equalizers with LMS, RLS and Back propagation training are compared.

Chapter 4

Genetic Algorithm and its Variants for Optimization Operations

4.1 Introduction

Gradient-descent training algorithms are the most common form of training algorithms in signal processing today because they have a solid mathematical foundation and have been proven over the last five decades to work in many environments. However, Gradient-descent training has few limitations:

1. Derivative based algorithm so there are chances that the parameters may fall to local minima during training if the cost function other than squared error is taken into consideration.

2. Do not perform satisfactorily under high noise condition

3. In certain cases they do not perform satisfactorily if the order of the channel increases

4. Do not perform satisfactorily for nonlinear channels

5. LMS algorithm at times exhibit slower convergence

6. Rather than converging to the optimal solution the LMS algorithm normally rattles around it.

7. RLS algorithm suffers from instability problem

These limitations can be removed by using evolutionary algorithms (derivative free algorithms) such as Genetic algorithm, Particle swarm optimization etc.

Genetic Algorithms (GA) are based upon the process of natural selection and does not require error gradient statistics. As a consequence, a GA is able to find a global error minimum [51]. The acceptance of GA optimization across many fields has been slow due to the lack of a mathematical derivation. Published results have, however, demonstrated the advantage of the GA optimization and have aided in changing this perception in many disciplines [52–57].

4.2 THE GENETIC ALGORITHM

GAs are stochastic search mechanisms that utilize a Darwinian criterion of population evolution. The GA has robustness that allows its structural functionality to be applied to many different search problems [51, 58]. This effectively means that once the search variables are encoded into a suitable format, the GA scheme can be applied in many environments. The process of natural selection, described by Darwin, is used to raise the effectiveness of a group of possible solutions to meet an environmental optimum [59].

GAs have been applied to many applications that have previously used ineffective and unstable optimization techniques. The IIR filter is one such example. The IIR error surface is known to be multimodal, gradient learning algorithms become either unstable or stuck within a local minima [20]. These are the same observations that have been made in gradient-based training of the MLP. 'Evolutionary' approaches have been applied to the adaptive IIR filter to overcome these learning problems [20, 60] and can be applied to the MLP equaliser [61, 62] .

4.2.1 GA Operations

The GA operates on the basis that a population of possible solutions, called chromosomes, is used to assess the cost surface of the problem. The GA evolutionary process can be thought of as solution breeding in that it creates a new generation of solutions by crossing two chromosomes. The solution variables or

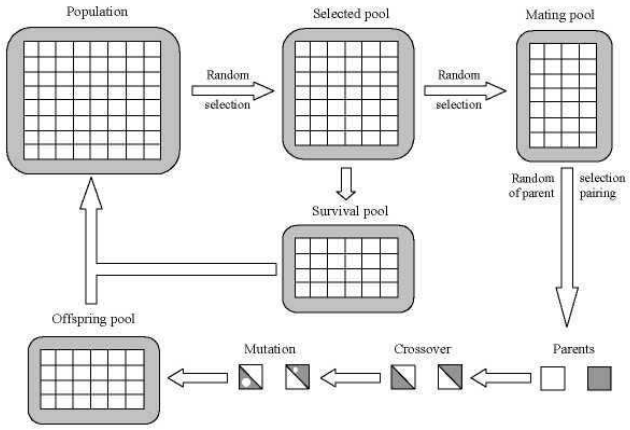


Figure 4.1: A GA iteration cycle. From the population a pool of individuals is randomly selected, some of these survive into the next iterations population. A mating pool is randomly created and each individual is paired off. These pairs undergo evolutionary operators to produce two new individuals that are added to the new population.

genes that provide a positive contribution to the population will multiply and be passed through each subsequent generation until an optimal combination is obtained.

The population is updated after each learning cycle through three evolutionary processes: *selection, crossover* and *mutation*. These create the new generation of solution variables.

The *selection* function creates a mating pool of parent solution strings based upon the *"survival of the fittest"* criterion. From the mating pool the *crossover* operator exchanges gene information. This essentially crosses the more productive genes from within the solution population to create an improved, more productive, generation. *Mutation* randomly alters selected genes, which helps prevent premature convergence by pulling the population into unexplored areas of the solution surface and adds new gene information into the population[1].

[1] 1 Chromosome - a single solution vector from the population. 2 Gene - a single variable from a solution vector. 3 Population - a number of solution vectors.

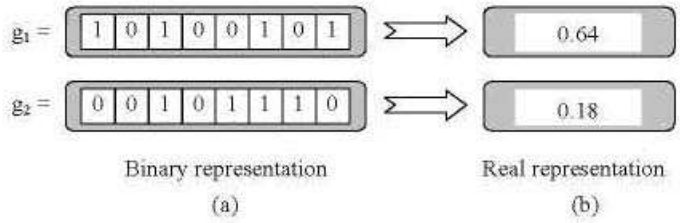


Figure 4.2: Representations of two single variable genes (a) 8-bit binary (b) real.

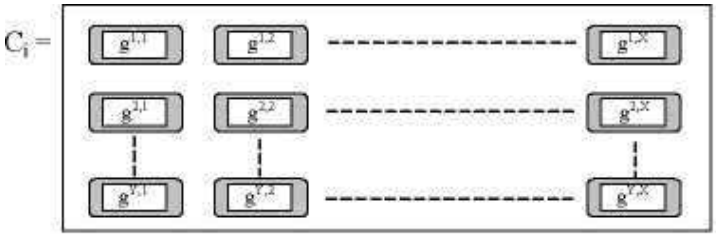


Figure 4.3: 3 A chromosome matrix of gene values $g_{y,x}$. C_i is the i_{th} solution chromosome within the population. .

4.2.2 POPULATION VARIABLES

A chromosome1 consists of the problem variables, where these can be arranged in a vector or a matrix. In the gene2 crossover process, corresponding genes are crossed so that there is no inter-variable crossing and therefore each chromosome uses the same fixed structure. An initial population3 that contains a diverse gene pool offers a better picture of the cost surface where each chromosome within the population is initialized independently by the same random process.

In the case of binary-genes each bit is generated randomly and the resulting bit-words are decoded into their real value equivalent.

The binary number is used in the genetic search process and the real value is used in the problem evaluation. This type of initialization results in a normally distributed population of variables across a specific range.

A GA population, P, consists of a set of N chromosomes $\{C_j...C_N\}$ and N fitness values $\{f_1...f_N\}$, where the fitness is some function of the error matrix.

$$P = [(c_1, f_1)(c_2, f_2)(c_3, f_3)...(c_N, f_N)] \tag{4.1}$$

The GA is an iterative update algorithm and each chromosome requires its fitness to be evaluated individually. Therefore, N separate solutions need to be assessed upon the same training set in each training iteration. This is a large evaluation overhead where population sizes can range between twenty and a hundred, but the GA is seen to have learning rates that evens this overhead out over the training convergence.

4.2.3 CHROMOSOME SELECTION

The selection process is used to weed out the weaker chromosomes from the population so that the more productive genes may be used in the production of the next generation. The chromosome finesses are used to rank the population with each individual assigned it own fitness value, f

$$E_i(n) = \frac{1}{M} \sum_{j=1}^{M} e_{ji}^2(n) \tag{4.2}$$

The solution cost value E_i of the f chromosome in the population is calculated from a training-block of M training signals (4.2) and from this cost an associated fitness is assigned:

$$f_i(n) = \frac{1}{(1 + E_i(n))} \tag{4.3}$$

The fitness can be considered to be the inverse of the cost but the fitness function in (4.3) is preferred for stability reasons, i.e.$E_i(n) = 0$.

When the fitness of each chromosome in the population has been evaluated, two pools are generated, a survival pool and a mating pool. The chromosomes from the mating pool will be used to create a new set of chromosomes through the evolutional processes of natural selection and the survival pool allows a number of chromosomes to pass onto the next generation. The chromosomes are selected randomly for the two pools but biased towards the fittest. Each chromosome may be chosen more than once and the fitter chromosomes are more likely to be chosen so that they will have a greater influence in the new generation of solutions.

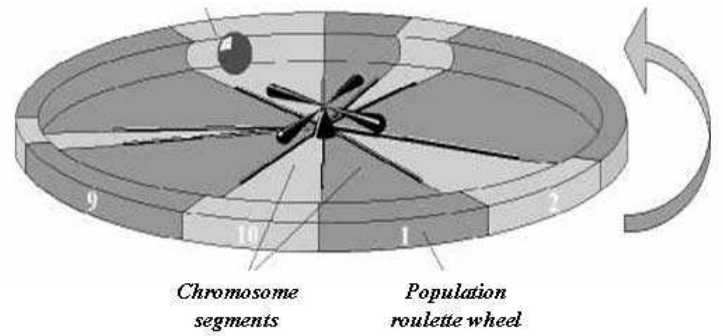


Figure 4.4: Biased roulette-wheel that is used in the selection of the mating pool.
.

The selection procedure can be described using a biased roulette wheel with the buckets of the wheel sized according to the individual fitness relative to the population's total fitness [51]. Consider an example population often chromosomes that have the fitness assessment of $f = 0.16,\ 0.16,\ 0.48,\ 0.08,\ 0.16,\ 0.24,\ 0.32,\ 0.08,\ 0.24,\ 0.16$ and the sum of the finesses are used to normalize these values, $f_{mm} = 2.08$.

Fig4.4 shows a roulette wheel that has been split into ten segments and each segment is in proportion to the population chromosomes relative fitness. The third individual has the highest fitness and nearly accounts for a quarter of the total fitness. The third segment therefore fills nearly a quarter of the roulette wheels area. The random selector points to a chosen chromosome, which is then copied into the mating pool because the third individual controls a greater proportion of the wheel, it has a greater probability of being selected.

As a procedural routine, the roulette wheel selection process is described Fig4.4. An individual is selected once the partial sum of fitness becomes greater than the random selector, which will be a value between zero and the sum of fitness.

After the GA crossover and mutation operators update the selected mating pool chromosomes, these supersede the old population and consequently the genes from the unselected chromosomes are lost.

$$\text{The sum of fitness} = \sum_{i=1}^{N_p} f_i \text{ - where } N_p \text{ is the population size.}$$

$i = 0$. $partial\_sum = 0$.

$rand\_no$ = Normally selected random number between zero and one.

Repeat until $partial\_sum >$ (randjao X the sum of fitness)

$i = i + 1$.

$partial\_sum = partial\_sum + fitness[i]$.

End.

Selected parent = i^{th} chromosome.

Figure 4.5: Selection routine used to create the GA mating pool.. .

4.2.4 GENE CROSSOVER

The crossover operator exchanges gene information between two selected chromosomes, (C_q, C_r), where this operation aims to improve the diversity of the solution vectors. The pair of chromosomes, taken from the mating pool, becomes the parents of two offspring chromosomes for the new generation.

In the case of a binary crossover operation the least significant bits are exchanged between corresponding genes within the two parents. For each gene-crossover a random position along the bit sequence is chosen and then all of the bits right of the crossover point is exchanged. When using an eight-bit word length there are nine positions the crossover selector can choose. In Fig4.6 the fifth crossover position is randomly chosen, where the first position corresponds to the left side with all the bits being exchanged and the ninth crossover position corresponding to the right side with no bit exchange. The original values in this example equated to 0.64 and 0.18 and the crossover produced two new values, 0.68 and 0.15.

Fig4.6shows a basic genetic crossover with the same crossover point chosen for both offspring genes. At the start of the learning process the extent of crossing over the whole population can be decided allowing the evolutionary process to randomly select the individual genes. The probability of a gene crossing, P(crossing), provides a percentage estimate of the genes that will be affected within each par-

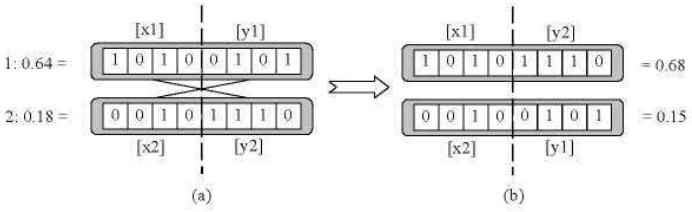


Figure 4.6: The basic genetic single point crossover (a) the original binary values (b) the new binary values.

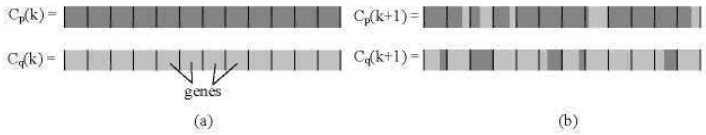


Figure 4.7: (a) Two chromosomes before crossover, (b) the chromosomes after crossover. The new genes contain splices from its mating partner. .

ent. $P(crossing) \leq 1$ allows all the gene values to be crossed and P(crossing)=0 leaves the parents unchanged, where a random gene selection value,$w \in \{1, 0\}$, is governed by this probability of crossing.

The crossover does not have to be limited to this simple operation. The crossover operator can be applied to each chromosome independently, taking different random crossing points in each gene. This operation would be more like grafting parts of the original genes onto each other to create the new gene pair. All of a chromosome's genes are not altered within a single crossover. A probability of gene-crossover is used to randomly select a percentage of the genes and those genes that are not crossed remain the same as one of the parents.

Fig4.7 describes a chromosome crossover. Each gene in both chromosomes are individually considered for crossover and those that are chosen are given a random amount of the corresponding gene from the matched chromosome.

4.2.5 CHROMOSOME MUTATION

The last operator within the breeding process is mutation. Each chromosome is considered for mutation with a probability that some of its genes will be mutated after the crossover operation.

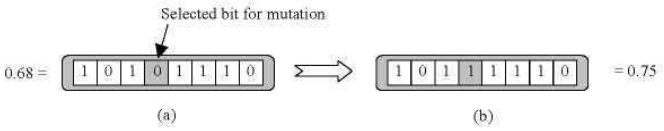


Figure 4.8: Binary chromosome (a) before mutation with a selected bit, (b) after the selected bit has been mutated.

A random number is generated for each gene, if this value is within the specified mutation selection probability, P(*mutation*), the gene will be mutated. The probability of mutation occurring tends to be low with around one percent of the population genes being affected in a single generation. In the case of a binary mutation operator, the state of the randomly selected gene-bits is changed, from zero to one or vice-versa.

4.3 REAL CODED GENETIC ALGORITHM (RCGA)

The GA crossover and mutation operators have been explained using a binary representation in section 4.1. These processes are easier to understand in the binary format, but the real number representation is more commonly used today. Real number genes enable a variety of different forms of crossover and mutation operators, where the binary genes are limited to the exchanging of bits [63, 64].

4.3.1 CROSSOVER

A common form of real number crossover involves an averaging of the two parent genes. The crossover used in this thesis is described in (4.4) and can be summarized as updating the variables by a percentage of its mating partner's value:

$$g_q^j(k+1) = g_q^j(k) + \alpha V_{qr}^j \varepsilon_q^j \omega_p g_r^j(k+1) = g_r^j(k) + \alpha V_{rq}^j \varepsilon_r^j \omega_p \tag{4.4}$$

where (g_q^j, g_r^j) are $j^{th}$the genes from the parent chromosomes (Cq, Cr). The amount of gene crossing is determined by a normal distributed random number, and this is applied to the crossing vector, The crossing vector in (4.5)describes

the difference between the two chromosome genes and will hold all of the gene information that will be shared.

$$V_r^q = g_r(k) - g_q(k) \tag{4.5}$$

The probability of a gene crossing, $F'(crossing)$, specifies the number of genes to be affected within each parent and the random gene selection values in the gene selection vector, $\{1, 0\}$, governs which genes are affected by the operation.

$$w = \begin{cases} 1if\sigma < P(crossing) \\ 0if\sigma \geq P(crossing) \end{cases} \tag{4.6}$$

(4.6) describes the generation of the selection vector elements, ω, where σ is a random number and $\sigma \in R[0, 1]$. The crossover range,α $(0 < \alpha < 1)$ is a scalar value that specifies the evolutionary step size and is equivalent to the learning rate in the LMS algorithm.

4.3.2 MUTATION

The real number mutation operator takes the selected genes and adds a random value from within a specified mutation range:

$$g^j(k+1)' = g^j(k+1) + \beta.\phi \tag{4.7}$$

where the j^{th} gene, *g(k+1)*, is selected and mutated by a random value β with in the mutation range $\beta\phi \in R[-1, 1]$. The mutation range is a difficult parameter to assign correctly, it can simply be set to a specific value or be some function of the population gene variance. In this thesis a specific value of fi is always assigned at the start of the training.

4.4 PARAMETERS OF GA

4.4.1 CROSSOVER and MUTATION PROBABILITY

There are two basic parameters of GA - crossover probability and mutation probability.

Crossover probability: This parameter decides how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is crossover, offspring are made from parts of both parent's chromosome. If crossover probability is 100 % then all offspring are made by crossover. If it is 0 % , whole new generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same). Crossover is made in hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better. However, it is good to leave some part of old population survives to next generation.

Mutation probability: This parameter decides how often parts of chromosome will be mutated. If there is no mutation, offspring are generated immediately after crossover (or directly copied) without any change. If mutation is performed, one or more parts of a chromosome are changed. If mutation probability is 100 % , whole chromosome is changed, if it is 0 %, nothing is changed. Mutation generally prevents the GA from falling into local extremes. Mutation should not occur very often, because then GA will in fact change to random search.

4.4.2 OTHER PARAMETERS

There are also some other parameters of GA. One another particularly important parameter is population size.

Population size: how many chromosomes are in population (in one generation). If there are too few chromosomes, GA has few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down. Research shows that after some limit (which depends mainly on encoding and the problem) it is not useful to use very large populations because it does not solve the problem faster than moderate sized populations.

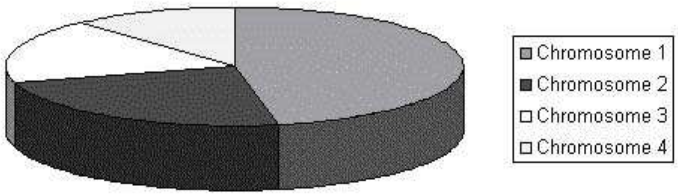


Figure 4.9: Roulette Wheel Selection

4.5 SELECTION METHODS

4.5.1 INTRODUCTION

As we already know from the GA outline, chromosomes are selected from the population to be parents for crossover. The problem is how to select these chromosomes. According to Darwin's theory of evolution the best ones survive to create new offspring. There are many methods in selecting the best chromosomes. Examples are roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection and some others.

Some of them will be described in this chapter.

4.5.2 Roulette Wheel Selection

Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have. Imagine a **roulette wheel** where all the chromosomes in the population are placed. The size of the section in the roulette wheel is proportional to the value of the fitness function of every chromosome - the bigger the value is, the larger the section is. See the following picture for an example.

A marble is thrown in the roulette wheel and the chromosome where it stops is selected. Clearly, the chromosomes with bigger fitness value will be selected more times. This process can be described by the following algorithm.

- [*Sum*] Calculate the sum of all chromosome fitness in population - sum S.
- [*Select*] Generate random number from the interval (0,S) - r.

- [*Loop*] Go through the population and sum the fitness from 0 - sum s. When the sum s is greater then r, stop and return the chromosome where you are.

Of course, the step 1 is performed only once for each population.

4.5.3 Steady-State Selection

This is not a particular method of selecting parents. The main idea of this type of selecting to the new population is that a big part of chromosomes can survive to next generation.

The stady-state selection GA works in the following way. In every generation a few good (with higher fitness) chromosomes are selected for creating new offspring. Then some bad (with lower fitness) chromosomes are removed and the new offspring is placed in their place. The rest of population survives to new generation.

4.5.4 Rank Selection

The previous type of selection will have problems when they are big differences between the fitness values. For example, if the best chromosome fitness is 90 % of the sum of all fitness then the other chromosomes will have very few chances to be selected. Rank selection ranks the population first and then every chromosome receives fitness value determined by this ranking. The worst will have the fitness 1, the second worst 2 etc. and the best will have fitness N (number of chromosomes in population). we can see in following picture, how the situation changes after changing fitness to the numbers determined by the ranking.

Now all the chromosomes have a chance to be selected. However this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones.

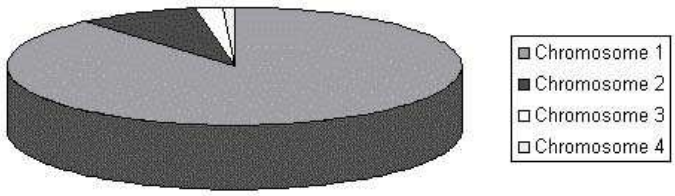


Figure 4.10: Situation before ranking (graph of fitness)

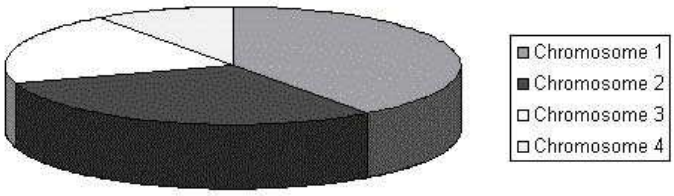


Figure 4.11: Situation after ranking (graph of order numbers)

4.5.5 Elitism

The idea of the elitism has been already introduced. When creating a new population by crossover and mutation, we have a big chance, that we will loose the best chromosome.

Elitism is the name of the method that first copies the best chromosome (or few best chromosomes) to the new population. The rest of the population is constructed in ways described above. Elitism can rapidly increase the performance of GA, because it prevents a loss of the best found solution.

4.6 ADAPTIVE BINARY CODED GENETIC ALGORITHM (AGA)

4.6.1 INTRODUCTION

The standard Genetic Algorithm (SGA) is slow i.e. it exhibits slower convergence rate. In other words in practice, SGA takes more time to train the adaptive filter. So to overcome this problem the fixed parameters of SGA are adapted [65, 66], there by accelerating the convergence rate of the algorithm.

4.6.2 ADAPTABLE PARAMETERS

- Probability of Crossover (P_c)
- Probability of Mutation (P_m)
- Population Size (Number of Chromosomes)
- Number of input samples

The above mentioned parameters are adapted in AGA to speed up its searching capabilities.

4.6.3 ADAPTIVE GENETIC ALGORITHM 1 (AGA1)

Here the P_c and P_m values are adapted according to some well defined rules:

MOTIVATIONS

It is essential to have two characteristics in GAs for optimizing multimodal functions. The first characteristic is the capacity to converge to an optimum (local or global) after locating the region containing the optimum. The second characteristic is the capacity to explore new regions of the solution space in search of the global optimum. The balance between these characteristics of the GA is dictated by the values of P_m, and P_c and the type of crossover employed. Increasing values of P_m and P_c promote exploration at the expense of exploitation. Moderately large values of P_c (0.5-1.0) and small values of P_m (0.001-0.05) are commonly employed in GA practice. In our approach, we aim at achieving this trade-off between exploration and exploitation in a different manner, by varying pm and pc adaptively in response to the fitness values of the solutions; pc and pm are increased when the population tends to get stuck at a local optimum and are decreased when the population is scattered in the solution space.

DESIGN OF ADAPTIVE P_C AND P_M

To vary P_c and P_m adaptively, for preventing premature convergence of GA to a local minimum it is essential to be able to identify whether the GA is converging to

an optimum. One possible way of detecting convergence is to observe the average fitness value f_{av} of the population in relation to the maximum fitness value f_{max} of the population. (f_{max}- f_{av}) is likely to be less for a population that has converged to an optimum solution than that for a population scattered in the solution space. The difference between in the average and maximum fitness values, (f_{max}- f_{av}) , is used as a yardstick for detecting the convergence of GA. The values of P_c and P_m are varied depending on the value of (f_{max}- f_{av}) . Since P_c and Pm have to be increased when the GA converges to a local minimum, i.e. when (f_{max}- f_{av}) decreases, P_c and P_m will have to be varied inversely with (f_{max}- f_{av}). Thus the expressions for Pc and Pm are:

$$\begin{aligned} P_c &= \frac{k_1}{(f_{max} - f_{av})} \\ P_m &= \frac{k_2}{(f_{max} - f_{av})} \end{aligned} \tag{4.8}$$

It has to be observed in the above expressions that P_c and P_m do not depend on the fitness value of any particular solution, and have the same values for all the solutions of the population. Consequently, solutions with high fitness values as well as solutions with low fitness values are subjected to the same levels of mutation and crossover. When a population converges to a globally optimal solution (or even a locally optimal solution), P_c and P_m increase and may cause the disruption of the near-optimal solutions. The population may never converge to the global optimum. Though we may prevent the GA from getting stuck at a local optimum, the performance of the GA (in terms of the generations required for convergence) will certainly deteriorate.

To overcome the above-stated problem, we need to preserve 'good' solutions of the population. This can be achieved by having lower values of P_c and P_m for high fitness solutions and higher values of P_c and P_m for low fitness solutions. While the high fitness solutions aid in the convergence of the GA, the low fitness solutions prevent the GA from getting stuck at a local optimum. The value of P_m should depend not only (f_{max}- f_{av}) but also the fitness value f of the solution. Similarly Pc should depend on the fitness values of both the parent solutions. The

closer f is to f_{max} ,the smaller $P_m$should be, i.e.. Pm should vary directly as (f_{max} - f). Similarly, Pc should vary directly as (f_{max} - f_p), where f_p is the larger of the fitness values of the solutions to be crossed. The expressions for P_c and P_m now take the forms

$$\begin{aligned} P_c &= k_1 \frac{(f_{max} - f_p)}{f_{max} - f_{av}}; \quad k1 \leq 1.0 \\ P_m &= k_2 \frac{(f_{max} - f)}{f_{max} - f_{av}}; \quad k2 \leq 1.0 \end{aligned} \tag{4.9}$$

k_1 and k_2 have to be less than 1.0 to constrain P_c and P_m to the range 0.0-1.0. Note that Pc and Pm are zero for the solution with the maximum fitness. Also $P_c = k_1$ for a solution with $f_p = f_{av}$ and $P_m = k_2$ for a solution with f = f_{av}. For solutions with sub average fitness values i.e., $f < f_{av}$, P_c and Pm might assume values larger than 1.0 . To prevent the overshooting of P_c and P_m beyond 1.0, we also have the following constraints,

$$\begin{aligned} P_c &= k_3, \quad f_p \leq f_{av} \\ P_m &= k_4, \quad f \leq f_{av} \end{aligned} \tag{4.10}$$

Where $k_3 \leq 1.0$ and $k_4 \leq 1.0$.

DEFAULT MUTATION

From the previous section it is clear that for a solution with the maximum fitness value P_c and P_m are both zero. The best solution in a population is transferred undisrupted into the next generation. Together with the selection mechanism, this may lead to an exponential growth of the solution in the population and may cause premature convergence. To overcome the above stated problem, we introduce a default mutation rate (of 0.005) for every solution in the AGA1.

CHOICE OF VALUES FOR K_1, K_2, K_3 AND K_4

The expressions for Pc and Pm are given as

$$P_c = \begin{cases} k_1 \frac{(f_{max} - f_p)}{(f_{max} - f_{av})}, & f_p \geq f_{av} \\ k_3, & f_p < f_{av} \end{cases} \tag{4.11}$$

and

$$P_m = \begin{cases} k_2 \frac{(f_{max}-f)}{(f_{max}-f_{av})}, f \geq f_{av} \\ k_4, f < f_{av} \end{cases} \tag{4.12}$$

Where k_1, k_2, k_3 and $k_4 \leq 1.0$

It is known that moderately large values of $P_c (0.5 < P_c < 1.0)$ and small values of $P_m (0.001 < P_m < 0.05)$ are essential for successful working of GAs. The moderately large value of Pc promote the extensive recombination of schemata where small values of Pm are necessary to prevent the disruption of the solutions. These guidelines are however relevant and useful when the values of Pc and Pm do not vary.

The main objective is to prevent the GA from getting stuck at a local optimum. To meet this goal AGA1 employs solutions with sub average fitness to search the search space for the region containing the global optimum. Such solutions need to be completely disrupted k_4 is assigned a value 0.5. Since solutions with a fitness value of f_{av} should also be disrupted completely k_2 is also assigned a value 0.5 as well.

Based on similar reasoning, k_1 and k_3 are set the value 1.0. This ensures that all solutions with a fitness value less than or equal to f_{av} compulsorily undergo crossover. The probability of crossover decreases as the fitness value (maximum of the fitness values of the parent populations) tends to f_{max} and is 0.0 for a solution with fitness value equal to f_{max}.

4.6.4 ADAPTIVE GENETIC ALGORITHM 2 (AGA2)

MOTIVATIONS

In AGA1 two mutations are carried out in one generation to prevent the premature convergence which is the source of delay. In other words the default mutation will consume some CPU time. According to AGA1, whether the fitness is more or less than the average, we can calculate the corresponding crossover probability and mutation probability. The closer the certain fitness to the optimum one, the less it's P_c and P_m is set. However, when they are equal to each other, P_c and P_m turn to be zero. This will make the better individuals (they may not be the

global optimum) stagnant at an early stage of evolution and drive the algorithm to local optimal solution. In other words, it will lead to premature convergence.

DESIGN OF ADAPTIVE P_C AND P_M

The improved P_c and P_m can be expressed as follows:

$$P_c = \begin{cases} P_{c1} \times (P_{c1} - P_{c2}) \times \frac{f_P - f_{av}}{f_{max} - f_{av}}, f_p \geq f_{av} \\ P_{c1}, f_p < f_{av} \end{cases} \tag{4.13}$$

$$P_m = \begin{cases} P_{m1} \times (P_{m1} - P_{m2}) \times \frac{f - f_{av}}{f_{max} - f_{av}}, f \geq f_{av} \\ P_{m1}, f < f_{av} \end{cases} \tag{4.14}$$

Where $P_{c1} = 0.9$, $P_{c2} = 0.6$, $P_{m1} = 0.1$ and $P_{m2} = 0.001$.

In this formula, the improvement guarantees the colony multiplicity and the convergence. As shown in the formula above, both crossover probability and mutation probability of the individual which has the maximum fitness are brought up to P_{c2} and P_{m2}.Corresponding, P_c and P_m of the better individuals increase at the same time. It solves the drawbacks of AGA1 successfully.

4.6.5 ADAPTIVE GENETIC ALGORITHM 3 (AGA3)

MOTIVATIONS

In SGA the population size (number of chromosomes) is kept fixed. At the start of the search process, the chromosomes are distributed randomly in the entire search space and gradually all the chromosomes tend to move in the direction of global optimum. Here instead of maintaining a constant population size (M) through out the generations, we can adaptively vary the population size. Similarly the number of input samples (n) can also be adaptively varied from generation to generation.

DESIGN OF ADAPTIVE POPULATION SIZE AND NUMBER OF INPUT SAMPLES

Initially (at generation = 1), the search process can be started with a large number of chromosomes to fill the entire search space. But as the number of

generations will increase, the chromosomes will gradually move towards the global optimum. In other words the search space gradually shrinks. So the number of potential solutions (chromosomes) can also be decreased proportionately. Here, the difference between the maximum (f_{max}) and average (f_{av}) fitness value (if the objective is to maximize the fitness function) of a particular generation is considered as a yardstick to solve the problem. For the similar reasons the number of input samples can be adapted judiciously. The adaptive equations can be expressed as follows:

$$\begin{aligned} M &= k^1 \times (f_{max} - f_{av}) \quad + \quad T_1 \\ n &= k^2 \times (f_{max} - f_{av}) \quad + \quad T_2 \end{aligned} \tag{4.15}$$

Where M and n represents the number of chromosomes and the number of input samples respectively of any particular generation. Again T_1 and T_2 represent the threshold values of M and n respectively.

The threshold values come in to picture because finally when all the chromosomes will attain the global optimum, $f_{max} = f_{av}$, so M = 0 and n = 0. In order to prevent this, the threshold values are added which will resolve the above problem.

4.7 SUMMARY

In this chapter GA is introduced and extensively explained. It has been stated that the GA can be applied to both unimodal and multimodal search surfaces for optimization where in the later case gradient descent algorithms face difficulties.

The GA operators such as Selection, Crossover and Mutation are also introduced. Crossover and mutation with binary as well as real numbers are discussed. The different parameters and selection procedures are also discussed.

The binary representation meets with difficulties when dealing with continuous search spaces with large dimensions and a great numerical precision is required. Since binary substrings representing each parameter with the desired precision are concatenated to form a chromosome for the GAs, the resulting chromosome encoding a large number of design variables would result in a huge string length.

Hence the Real coded GA was also introduced and the real crossover and real mutation were also explained.

Finally the fixed parameters of the SGA are adapted to gear up the search process. The adaptive probability of crossover, probability of mutation, population size and the number of input samples are also explained in this chapter.

Chapter 5

Application of GA based algorithms for channel equalization

5.1 Introduction

High speed data transmission over communication channels distorts the transmitted signals in both amplitude and phase due to presence of Inter Symbol Interference (ISI). Other impairments like thermal noise, impulse noise and cross talk also cause further distortions to the received symbols. Adaptive equalization of the digital channels at the receiver removes/reduces the effects of such ISIs and attempts to recover the transmitted symbols. Basically an equalizer is a filter which is placed in cascade with the transmitter and receiver with the aim to have an inverse transfer function of that of the channel in order to augment accuracy of reception. These issues are more elaborately discussed in the Chapter-2 and 3.

The Least-Mean-Square (LMS), Recursive-Least-Square (RLS) and Multilayer perceptron (MLP) based equalizers aim to minimize the ISI present in the channels particularly for nonlinear channels. However they suffer from long training time and undesirable local minima during training. Again the disadvantages or drawbacks of these derivative based algorithms have been discussed in Chapter-4.

In the present chapter we propose a new adaptive channel equalizer using Genetic Algorithm (GA) optimization technique which is essentially a derivative free optimization tool. This algorithm has been suitably used to update the weights

of the equalizer. The performance of the proposed equalizer has been evaluated and has been compared with its LMS based counter part.

However being a population based algorithm, the standard Genetic Algorithm (SGA) suffers from slower convergence rate. Hence the parameters of SGA are updated to improve the convergence rate without sacrificing the accuracy of reception. Further the Real-coded-GA (RCGA) is preferred to Binary-coded-GA (BGA) due to several reasons and these issues are discussed more elaborately later in this chapter.

5.2 STEPWISE REPRESENTATION OF GA BASED CHANNEL EQUALIZATION ALGORITHM:

The updating of the weights of the GA based equalizer is carried out using GA rule as outlined in the following steps:

1. The structure of the equalizer is a FIR system whose coefficients are initially chosen from a population of M chromosomes. Each chromosome constitutes NL number of random binary bits, each sequential group of L-bits represent one coefficient of the adaptive model, where N is the number of parameters of the model.

2. Generate K (500) number of input signal samples which are random binary in nature.

3. Each of the input samples is passed through the channel and then contaminated with the additive noise of known strength. The resultant signal is passed through the equalizer. In this way K numbers of desired signals are produced by feeding all the K input samples.

4. Each of the input sample is delayed which acts as desired signal.

5. Each of the desired output is compared with corresponding channel output and K errors are produced. The mean square error (MSE) for a given group

of parameters (corresponding to nth chromosome) is determined by using the relation $MSE(n) = \sum_{i=1}^{K} \frac{e_k^2}{K}$.This is repeated for N times.

6. Since the objective is to minimize MSE (n), n=1 to N, the GA based optimization is used.

7. The crossover, mutation and selection operator are sequentially carried out following the steps as given in Chapter-4.

8. In each generation the minimum MSE(MMSE) (expressed in dB) is stored which shows the learning behavior of the adaptive model from generation to generation.

9. When the MMSE has reached a pre-specified level the optimization is stoped.

10. At this step all the chromosomes attend almost identical genes, which represent the desired filter coefficients of the equalizer.

5.3 COMPARISON BETWEEN LMS and GA BASED EQUALIZER:

5.3.1 COMPUTER SIMULATIONS:

In this section we carry out the simulation study of new channel equalizer. The coefficients of the equalizer are updated using both GA and LMS algorithm. The results of two different linear and nonlinear channels are used. While training, the additive noises used in the channel are -30dB (low noise), -10dB (medium noise) and 0dB (high noise) to test the performance of the three different algorithms in different noise conditions. Finally the performance of the equalizers is compared by plotting the Bit-error-rate (BER) graphs.
The following standard linear channels are used in the simulation study :

1. CH1: $H(z) = 0.2600 + 0.9300z^{-1} + 0.2600z^{-2}$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

2. CH2: $H(z) = 0.3410 + 0.8760z^{-1} + 0.3410z^{-2}$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

In addition the following nonlinear channels are also used in the simulation.

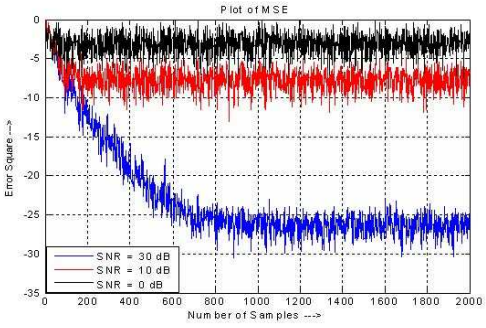
1. CH1: $H(z) = 0.2600 + 0.9300z^{-1} + 0.2600z^{-2}$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

2. CH2: $H(z) = 0.3410 + 0.8760z^{-1} + 0.3410z^{-2}$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

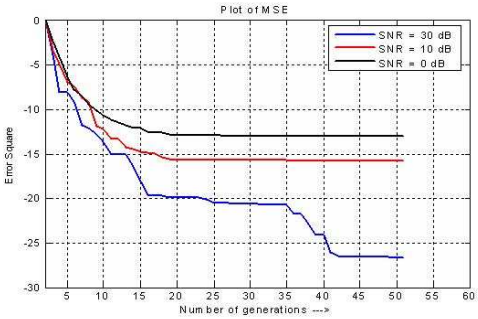
Where b (k) is the output of the nonlinear channel

The desired signal is generated by delaying the input binary sequence by m samples where or depending upon N is even or odd where N represents the order of the channel. In the simulation study N = 8 has been taken. For LMS algorithm, mu = 0.02. For binary coded GA (BGA), population size (M) = 40, total number of bits used to represent each chromosome = 120 (i.e. 15 bits per variable), $R_{min} = -2, R_{max} = 2$(where R_{min} and R_{max} represents the range or boundary values), Pc(Probability of crossover) = 0.9 and Pm(Probability of mutation) = 0.03. Again tournament selection is preferred which is followed by two-point crossover.

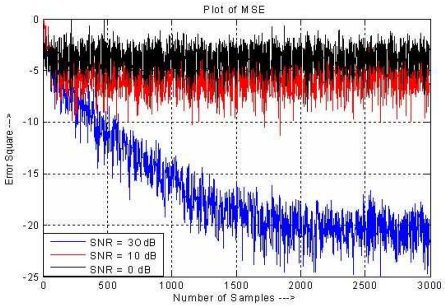
The convergence characteristics of BGA and LMS is obtained from simulation and is shown in Fig.5.1(a, b, c and d) and Fig. 5.2(a, b, c and d) for the linear channels and nonlinear channels respectively. Similarly the bit error plot (BER) of LMS and BGA equalizers for linear and nonlinear channels are shown in Fig. 5.3 (a, b, c, d, e and f) and Fig 5.4 (a, b, c, d, e and f) respectively. These results are used for performance comparison.



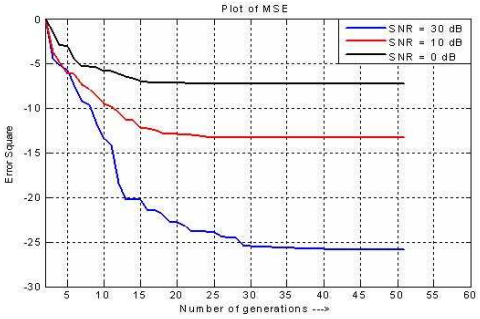
(a) CH1, LMS algorithm



(b) CH1, Genetic algorithm

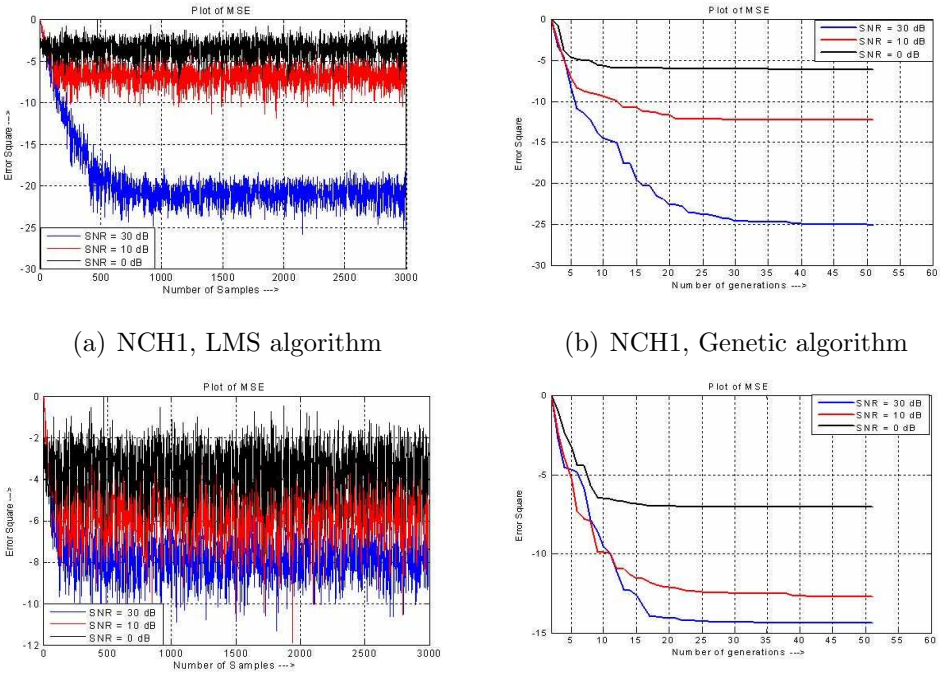


(c) CH2, LMS algorithm



(d) CH2, Genetic algorithm

Figure 5.1: Plot of convergence characteristics of various algorithms for different linear channels at 30dB, 10dB and 0dB



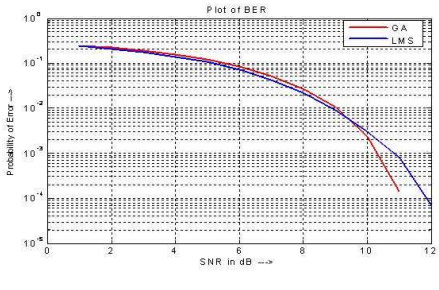
(a) NCH1, LMS algorithm

(b) NCH1, Genetic algorithm

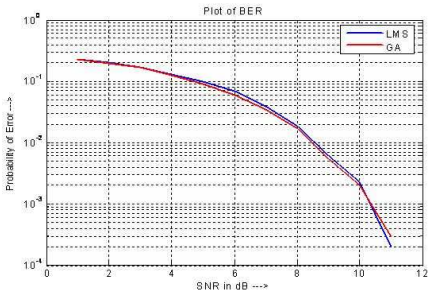
(c) NCH2, LMS algorithm

(d) NCH2, Genetic algorithm

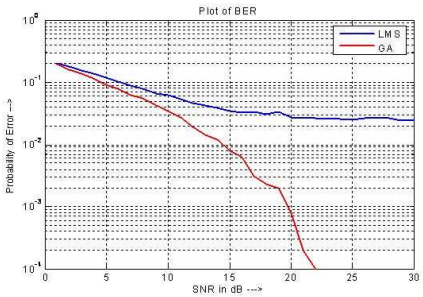
Figure 5.2: Plot of convergence characteristics of various algorithms for different nonlinear channels at 30dB, 10dB and 0dB



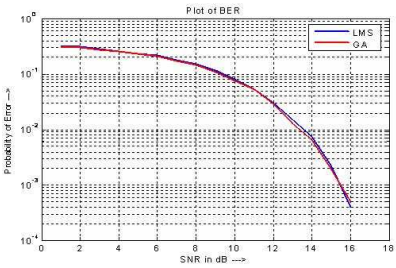
(a) CH1, SNR=30dB



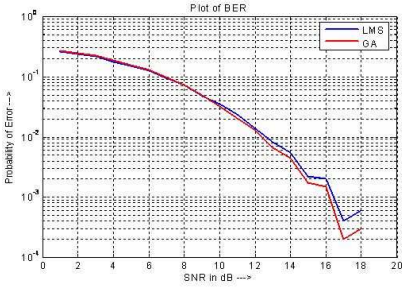
(b) CH1, SNR=10dB



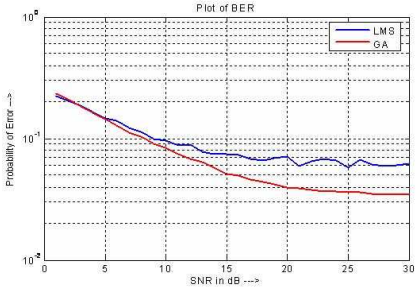
(c) CH1, SNR=0dB



(d) CH2, SNR=30dB

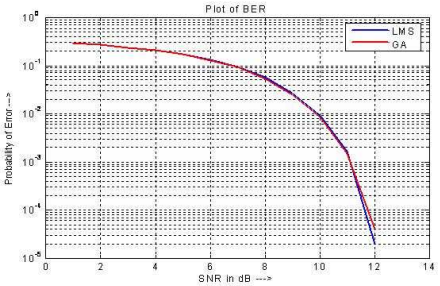


(e) CH2, SNR=10dB

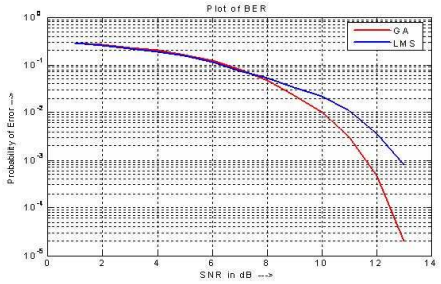


(f) CH2, SNR=0dB

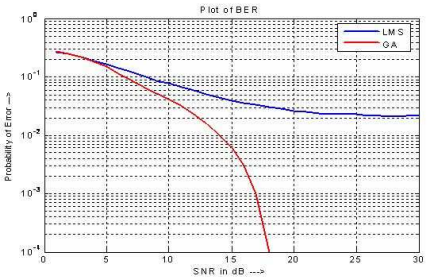
Figure 5.3: BER performance of LMS and GA based equalizer for different linear channels at different noise conditions



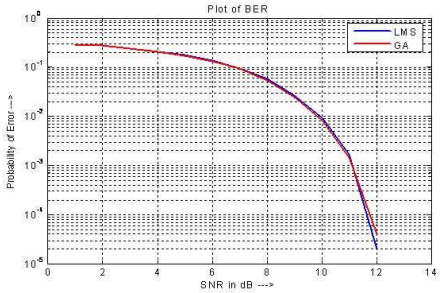
(a) NCH1, SNR = 30dB



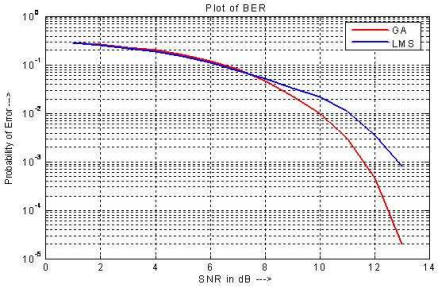
(b) NCH1, SNR = 10dB



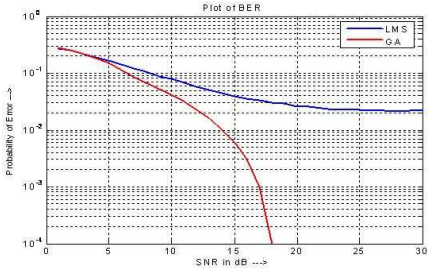
(c) NCH1, SNR = 0dB



(d) NCH2, SNR = 30dB



(e) NCH2, SNR = 10dB



(f) NCH2, SNR = 0dB

Figure 5.4: BER performance of LMS and GA based equalizer for different non-linear channels at different noise conditions

Table 5.1: Comparison of CPU Time

Algorithm used in equalization	Approximate CPU Time (in seconds)
LMS	1 - 1.5
SGA	20 - 23

* CPU times are measured under similar conditions

5.3.2 CONCLUSION:

Thus it can be concluded from the results that

For Linear channels:

1. For less noisy conditions, the LMS and GA equalizer perform almost similarly

2. Under high noise conditions, the GA equalizer outperforms its LMS counterpart.

For Nonlinear channels

For both low and high noise conditions, the performance of GA equalizer is better than the LMS equalizer.

5.3.3 COMPARISON OF CONVERGENCE SPEED:

It is clear from the Table5.1that the convergence speed of LMS algorithm is better than the GA i.e. is to achieve the convergence, the LMS algorithm consumes less time than the GA based approach.

Hence it is concluded that the SGA is slow but it exhibits superior bit-error-rate performance.

5.4 COMPARISON BETWEEN GA and AGA BASED EQUALIZER:

The basic GA is slow in training i.e. it exhibits slower convergence; it takes more time to train the equalizer parameters. So in this chapter various parameters of the GA are adapted to accelerate the convergence speed of the SGA.

5.4.1 COMPUTER SIMULATIONS:

In this section we carry out the simulation study of proposed channel equalizers. The coefficients of the equalizer are updated using GA and AGA algorithms. The results of four different linear and nonlinear channels are used. While training, a white uniform noise of strength 30dB is added to test the performance of the three different AGA based equalizers. Finally the performance of the equalizers is tested by plotting the Bit-error-rate (BER) graphs.

The following linear channel models are used:

1. CH1: $H(z) = 0.2014 + 0.9586z^{-1} + 0.2014z^{-2}$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

2. CH2: $H(z) = 0.2600 + 0.9300z^{-1} + 0.2600z^{-2}$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

3. CH3: $H(z) = 0.3040 + 0.9029z^{-1} + 0.3040z^{-2}$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

4. CH4: $H(z) = 0.3410 + 0.8760z^{-1} + 0.3410z^{-2}$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

The following nonlinear channel models are used:

1. NCH1: $H(z) = 0.2014 + 0.9586z^{-1} + 0.2014z^{-2}$
   $b(k) = a(k) + 0.2a^2(k) - 0.1a^3(k)$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

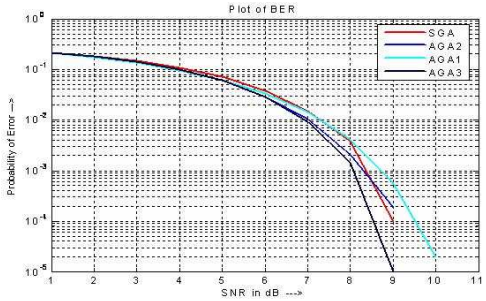
2. NCH2: $H(z) = 0.2600 + 0.9300z^{-1} + 0.2600z^{-2}$
   $b(k) = tanh[a(k)]$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

3. NCH3: $H(z) = 0.3040 + 0.9029z^{-1} + 0.3040z^{-2}$
   $b(k) = a(k) + 0.2a^(k) + -0.1a^3(k)$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

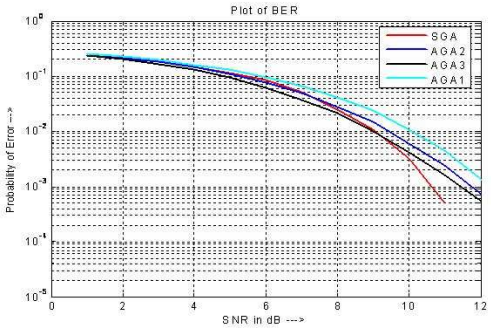
4. NCH4: $H(z) = 0.3410 + 0.8760z^{-1} + 0.3410z^{-2}$
   $b(k) = tanh[a(k)]$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

Where b (k) is the output of the nonlinear channel. For binary coded GA (BGA), population size (M) = 40, total number of bits used to represent each chromosome = 120 (that is 15 bits per variable), $R_{min} = -2, R_{max} = 2$ (where R_{min} and R_{max} represents the range or boundary values), Pc(Probability of crossover) = 0.9 and Pm(Probability of mutation) = 0.03. Again tournament selection is preferred which is followed by two point crossover.

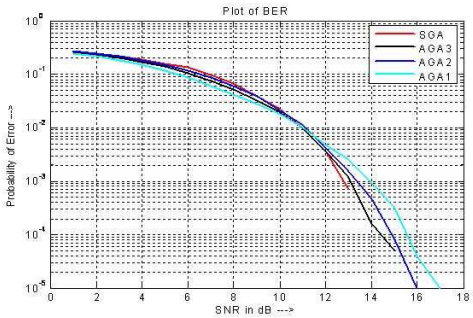
The bit error plot (BER) of BGA and three different AGA equalizers for linear and nonlinear channels are shown in Fig.5.5 (a, b, c and d) and Fig5.6(a, b, c and d) respectively. These results are used for comparison of performance.



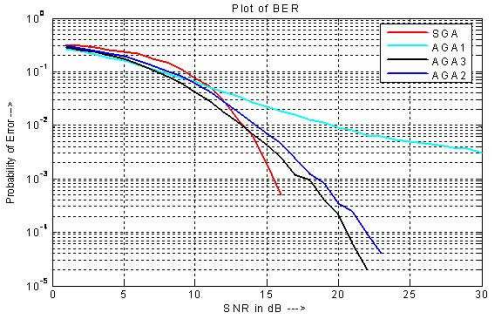
(a) CH1



(b) CH2

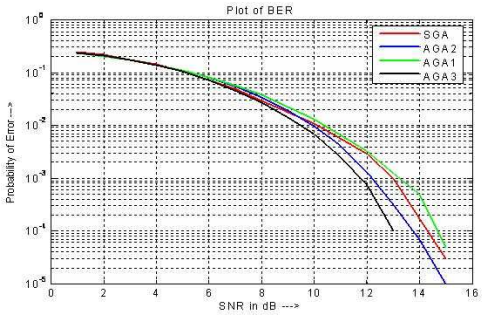


(c) CH3

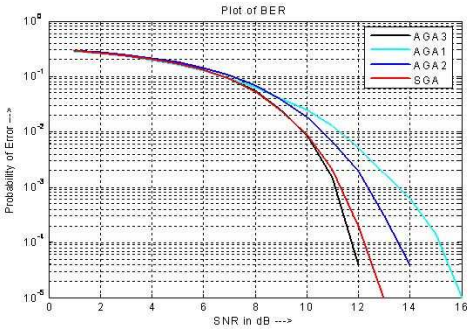


(d) CH4

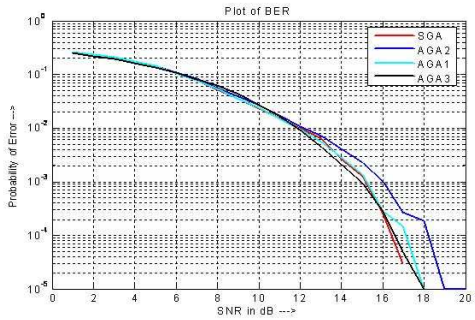
Figure 5.5: Comparison of BER of various linear channels between GA and its varieties at SNR=20dB



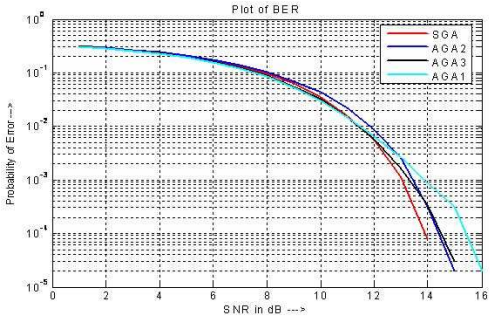
(a) NCH1



(b) NCH2



(c) NCH3



(d) NCH4

Figure 5.6: Comparison of BER of various nonlinear channels between GA and its varieties at SNR=20dB

Table 5.2: Comparison of CPU Time

Algorithms used in training the equalizer	Approximate CPU Time (in seconds)
SGA	20 - 23
AGA1	10.5 - 11.5
AGA2	8.5 - 9.5
AGA3	4 - 5

* CPU times are measured under similar conditions

5.4.2 CONCLUSION:

From BER plots it is clear that, for most of the linear and nonlinear channels, the performance of AGA3 is better than SGA where as the performance of AGA1 and AGA2 slightly deteriorates than of the SGA.

5.4.3 COMPARISON OF CONVERGENCE SPEED:

From the present study it is concluded that, the performance of AGA3 is better than that of the SGA in terms of BER and convergence rate where as the AGA1 and AGA2 based approach exhibits faster convergence at the cost of accuracy of reception .

5.5 COMPARISON BETWEEN the LMS and RCGA BASED EQUALIZERS:

The Real-Coded Genetic Algorithm (RCGA) is preferred to Binary-Coded Genetic Algorithm (SGA) because:-

1. Binary representation meets difficulties when dealing with continuous search spaces with high dimensions and when great precision is needed.

2. In RCGA, a chromosome is coded as a finite-length string of the real numbers corresponding to the design variables. Thus the coding - decoding of chromosome is eliminated.

3. The real-coded GAs are robust, accurate, and efficient because the floating point representation is conceptually closest to the real design space.

4. Range issues are eliminated.

Further the details of RCGA s explained in Chapter-4.

5.5.1 COMPUTER SIMULATIONS:

In this section we carry out the simulation study of new channel equalizer. The coefficients of the equalizer are updated using RCGA and LMS algorithm. The results of four different nonlinear channels are used. While training, the additive noises used in the channel are -30dB (low noise), -10dB (medium noise) and 0dB (high noise) to test the performance of the three different algorithms in different noise conditions. Finally the performance of the equalizers is tested by plotting the Bit-error-rate (BER).
The following nonlinear channel models are used:

1. NCH1: $H(z) = 0.2014 + 0.9586z^{-1} + 0.2014z^{-2}$
   $b(k) = a(k) + 0.2a^2(k) - 0.1a^3(k) + 0.5cos(\Pi(k))$
   $NSR = -30dB$

2. NCH2: $H(z) = 0.2600 + 0.9300z^{-1} + 0.2600z^{-2}$
   $b(k) = tanh[a(k)]$
   NSR = -10 dB,

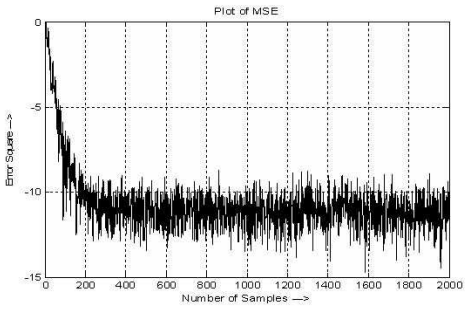
3. NCH3: $H(z) = 0.3040 + 0.9029z^{-1} + 0.3040z^{-2}$
   $b(k) = a(k) + 0.2a^{(}k) + -0.1a^{3}(k)$
   NSR = -30dB

4. NCH4: $H(z) = 0.3410 + 0.8760z^{-1} + 0.3410z^{-2}$
   $b(k) = tanh[a(k)]$
   NSR = -20 dB

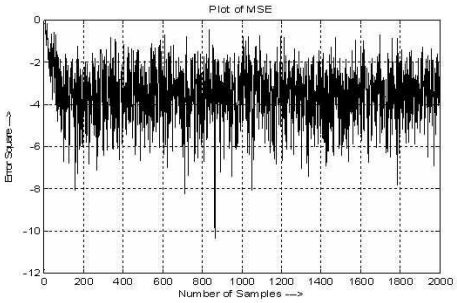
Where b (k) is the output of the nonlinear channel.

The desired signal is generated by delaying the input binary sequence by m samples where or depending upon N is even or odd where N represents the order of the channel. In the simulation study N = 8 has been taken. For LMS algorithm, mu = 0.02. For Real coded GA (RCGA), population size (M) = 60, Pc (Probability of crossover) = 0.8, Pm (Probability of mutation) = 0.1, α (range of crossover) = 0.8 and β (range of mutation) = 0.9. Again tournament selection scheme is preferred and total number of generations = 70.

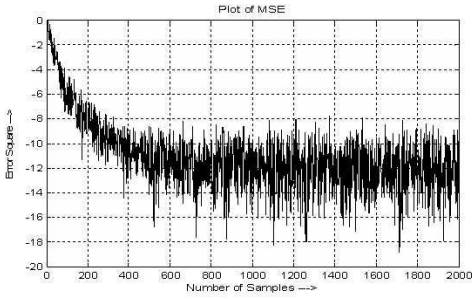
The convergence characteristics of RCGA and LMS is obtained from simulation and is shown in Fig.5.7(a, b, c and d) and Fig. 5.8(a, b, c and d). Similarly the bit error plot (BER) of LMS and RCGA equalizers for nonlinear channels are shown in Fig. 5.9 (a, b, c and d). These results are used for comparison.



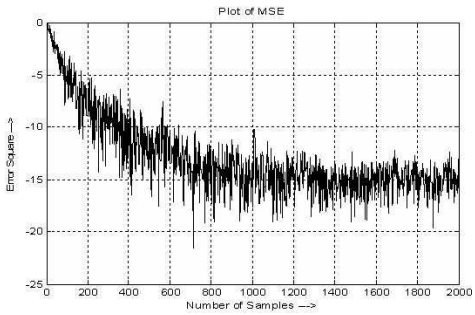
(a) NCH1, SNR=30dB



(b) NCH2, SNR=10dB

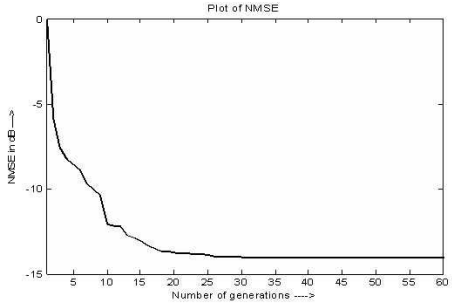


(c) NCH3, SNR=30dB

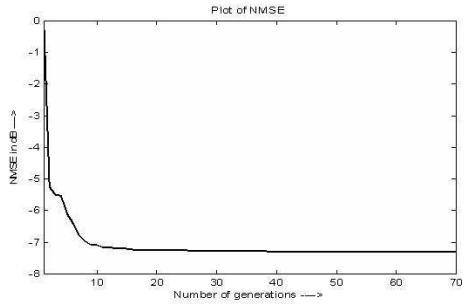


(d) NCH4, SNR=20dB

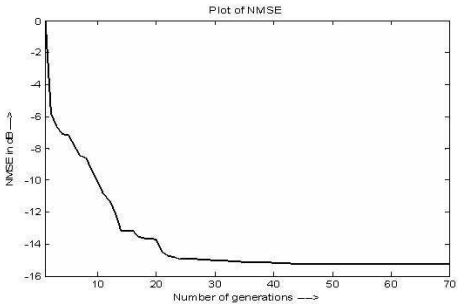
Figure 5.7: Plot of convergence characteristics of various nonlinear channels at different noise conditions using LMS algorithm



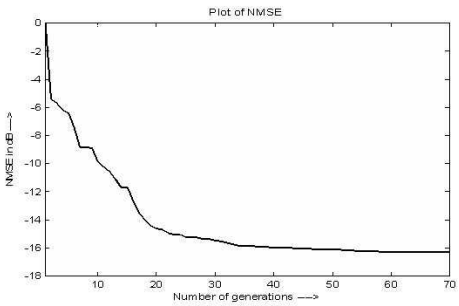
(a) NCH1, SNR=30dB



(b) NCH2, SNR=10dB

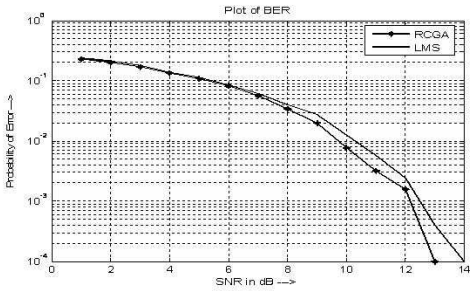


(c) NCH3, SNR=30dB

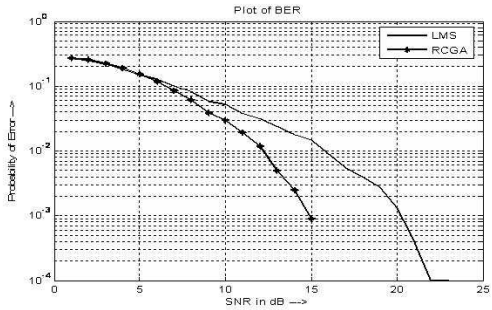


(d) NCH4, SNR=20dB

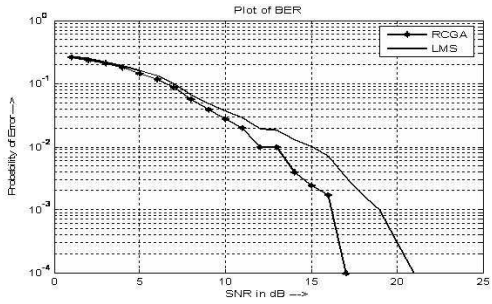
Figure 5.8: Plot of convergence characteristics of various nonlinear channels at different noise conditions using RCGA



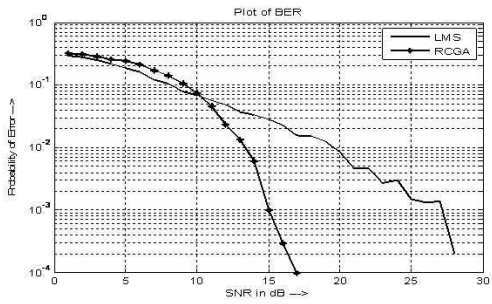
(a) NCH1, SNR=30dB



(b) NCH2, SNR=10dB



(c) NCH3, SNR=30dB



(d) NCH4, SNR=20dB

Figure 5.9: Comparison of BER of various Nonlinear channels between LMS and RCGA based equalizer at different noise conditions

Table 5.3: Comparison of CPU Time

Algorithms used in training the equalizer	Approximate CPU Time (in seconds)
LMS	1 - 1.5
SGA	20 - 23
RCGA	17

* CPU Times are measured under similar conditions.

5.5.2 CONCLUSION:

From the BER plots it is evident that the RCGA equalizer outperforms the LMS equalizer under different noisy environments.

5.5.3 COMPARISON OF CONVERGENCE SPEED:

From Table 5.3 it is concluded that, RCGA exhibits faster convergence than SGA without sacrificing the accuracy of reception.

Hence RCGA can be used as a better substitute to SGA.

5.6 SUMMARY

Most of the drawbacks of derivative based algorithm which are highlighted in Chapter-4 are alleviated by the derivative free Optimization tool, such as the GA. Since GA is a population based random search mechanism it consumes more time than its counterparts i.e. it provides slower convergence rate. Hence to gear up the convergence speed, different adaptive procedures are followed which in turn decreases the CPU time without sacrificing the accuracy of reception. Finally the RCGA equalizer is developed and its performance is studied and compared .

Chapter 6

PSO Algorithm and its application to channel equalization

6.1 Introduction

Natural creatures sometimes behave as a swarm. One of the main streams of artificial life researches is to examine how natural creatures behave as a swarm and reconfigure the swarm models inside a computer. Reynolds developed boid as a swarm model with simple rules and generated complicated swarm behavior by CG animation [67].

From the beginning of 90's, new optimization technique researches using analogy of swarm behavior of natural creatures have been started. Dorigo developed ant colony optimization (ACO) mainly based on the social insect, especially ant, metaphor [68]. Each individual exchanges information through pheromone implicitly in ACO. Eberhart and Kennedy developed particle swarm optimization (PSO) based on the analogy of swarm of bird and fish school [69]. Each individual exchanges previous experiences in PSO. These researches are called "Swarm Intelligence" [70, 71]. This chapter describes mainly about PSO as one of swarm intelligence techniques.

PSO has been expanded to handle combinatorial optimization problems and both discrete and continuous variables as well. Efficient treatment of mixed-integer nonlinear optimization problems (MINLP) is one of the most difficult problems

in optimization field. Moreover, unlike other EC techniques, PSO can be realized with only small program. Namely PSO can handle MINLP with only small program. This feature of PSO is one of the advantages compared with other optimization techniques.

6.2 MOTIVATION:

Other evolutionary computation (EC) techniques such as genetic algorithm (GA) also utilize some search points in the solution space. While GA is a random search process, PSO is a more deterministic search algorithm. Again unlike GA, PSO utilizes the past history or each other's experience to solve a problem. So it is expected that being more organized PSO will consume less CPU time than its counterpart maintaining the same performance.

6.3 Basic particle swarm optimization

6.3.1 Background of particle swarm optimization

Natural creatures sometimes behave as a swarm. One of the main streams of artificial life researches is to examine how natural creatures behave as a swarm and reconfigure the swarm models inside a computer. Swarm behavior can be modelled with a few simple rules. School of fishes and swarm of birds can be modelled with such simple models. Namely, even if the behavior rules of each individual (agent) are simple, the behavior of the swarm can be complicated. Reynolds called this kind of agent as boid and generated complicated swarm behavior by CG animation [67]. He utilized the following three vectors as simple rules.

1. to step away from the nearest agent
2. to go toward the destination
3. to go to the center of the swarm

Namely, behavior of each agent inside the swarm can be modelled with simple vectors. This characteristic is one of the basic concepts of PSO.

Boyd and Richerson examine the decision process of human being and developed the concept of individual learning and cultural transmission [72]. According to their examination, people utilize two important kinds of information in decision process. The first one is their own experience; that is, they have tried the choices and know which state has been better so far, and they know how good it was. The second one is other people's experiences; that is, they have knowledge of how the other agents around them have performed. Namely, they know which choices their neighbors have found are most positive so far and how positive the best pattern of choices was. Namely each agent decides his decision using his own experiences and other peoples' experiences. This characteristic is another basic concept of PSO.

6.3.2 Basic method

According to the background of PSO and simulation of swarm of bird, Kennedy and Eberhart developed a PSO concept. Namely, PSO is basically developed through simulation of bird flocking in two-dimension space. The position of each agent is represented by XY axis position and also the velocity is expressed by v_x (the velocity of X axis) and v_y (the velocity of Y axis). Modification of the agent position is realized by the position and velocity information.

Bird flocking optimizes a certain objective function. Each agent knows its best value so far (pbest) and its XY position. This information is analogy of personal experiences of each agent. Moreover, each agent knows the best value so far in the group (gbest) among pbests. This information is analogy of knowledge of how the other agents around them have performed. Namely, each agent tries to modify its position using the following information:

1. the current positions (x, y)
2. the current velocities (v_x, v_y)
3. to go to the center of the swarm
4. the distance between the current position and pbest
5. the distance between the current position and gbest

This modification can be represented by the concept of velocity. Velocity of each agent can be modified by the following equation:

$$V_i^{(k+1)} = wV_i^k + c_1 rand_1 \times (pbest_i - s_i^k) + c_2 rand_2 \times (gbest_i - s_i^k) \tag{6.1}$$

Where V_i^k : velocity of agent i at iteration k,

w : weighting function,

c_j : weighting factor,

rand : random number between 0 and 1,

s_i^k : current position of agent i at iteration k,

$pbest_i$: pbest of agent i,

gbest : gbest of the group.

The following weighting function is usually utilized in (6.1):

$$w = w_{max} \frac{(w_{max} - w_{min})}{iter_{max}} \times iter \tag{6.2}$$

Where w_{max} : initial weight,

w_{min} : final weight,

$iter_{max}$: maximum iteration number,

iter : current iteration number.

Using the above equation, a certain velocity, which gradually gets close to pbest and gbest can be calculated. The current position (searching point in the solution space) can be modified by the following equation:

$$S_i^{(k+1)} = S_i^k + V_i^{k+1} \tag{6.3}$$

Fig.6.1shows a concept of modification of a searching point by PSO and Fig.6.2 shows a searching concept with agents in a solution space. Each agent changes its current position using the integration of vectors as shown in Fig.6.1.

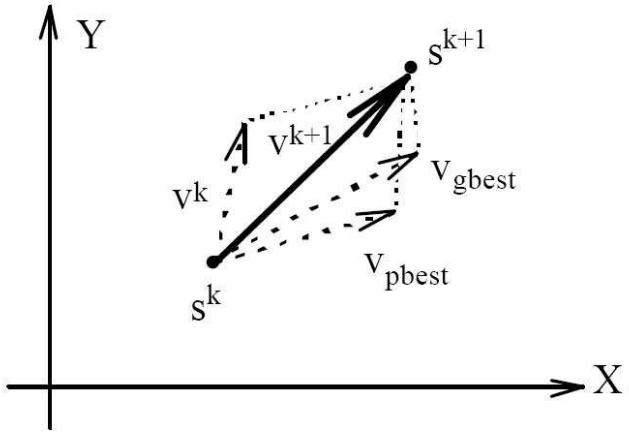


Figure 6.1: Concept of modification of a searching point by PSO

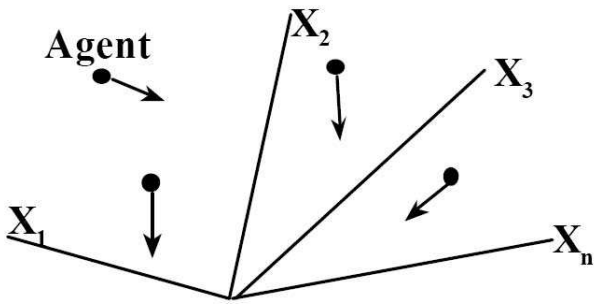


Figure 6.2: Searching concept with agents in solution space by PSO

The general flow chart of PSO can be described as follows:

1. **Step. 1** Generation of initial condition of each agent

   Initial search points (si^0) and velocities (vi^0) of each agent are usually generated randomly within the allowable range. The current searching point is set to pbest for each agent. The best-evaluated value of pbest is set to gbest and the agent number with the best value is stored.

2. **Step. 2** Evaluation of searching point of each agent

   The objective function value is calculated for each agent. If the value is better than the current pbest of the agent, the pbest value is replaced by the current value. If the best value of pbest is better than the current gbest, gbest is replaced by the best value and the agent number with the best value is stored.

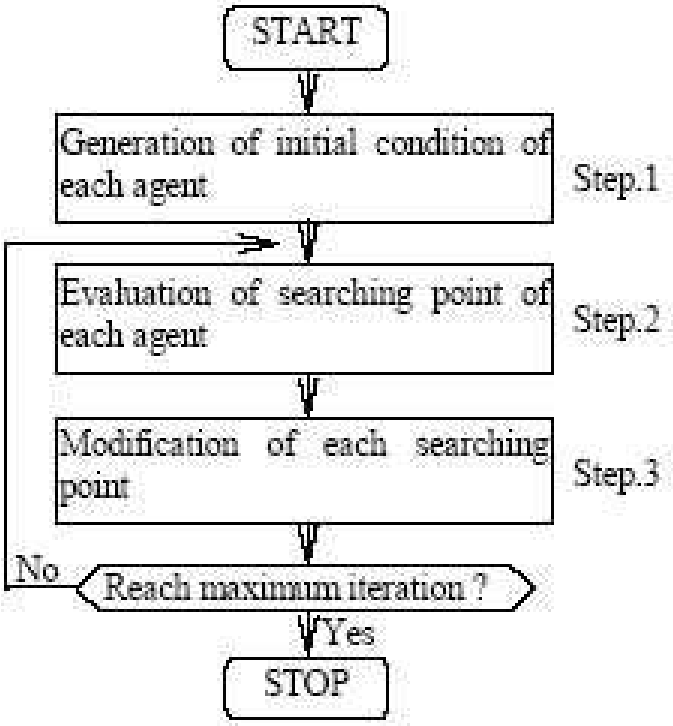


Figure 6.3: General flow chart of PSO

3. **Step. 3** Modification of each searching point

   The current searching point of each agent is changed using (6.1)(6.2)(6.3).

4. **Step. 4** Checking the exit condition

   The current iteration number reaches the predetermined maximum iteration number, then exit. Otherwise, go to step 2.

Fig.6.3 shows the general flow chart of PSO. The features of the searching procedure of PSO can be summarized as follows:

1. As shown in (6.1)(6.2)(6.3), PSO can essentially handle continuous optimization problem.

2. PSO utilizes several searching points like genetic algorithm and the searching points gradually get close to the optimal point using their pbest and the gbest.

3. The first term of the right-hand side (RHS) of (6.1) is corresponding to diversification in the search procedure. The second and third terms of that

are corresponding to intensification in the search procedure. Namely, the method has a well-balanced mechanism to utilize diversification and intensification in the search procedure efficiently.

4. The above concept is explained using only XY-axis (two-dimension space). However, the method can be easily applied to n-dimension problem. Namely, PSO can handle continuous optimization problems with continuous state variables in a n-dimension solution space.

The above feature (3) can be explained as follows [73]. The RHS of (6.1) consists of three terms. The first term is the previous velocity of the agent. The second and third terms are utilized to change the velocity of the agent. Without the second and third terms, the agent will keep on "flying" in the same direction until it hits the boundary. i.e., it tries to explore new areas and, therefore, the first term is corresponding to diversification in the search procedure. On the other hand, without the first term, the velocity of the "flying" agent is only determined by using its current position and its best positions in history, the agents will try to converge to their pbests and/or gbest and, therefore, the terms are corresponding to intensification in the search procedure. The basic PSO has been applied to a learning problem of neural networks and Schaffer f6, the famous benchmark function for GA, and efficiency of the method has been confirmed [69].

6.4 Variations of particle swarm optimization

6.4.1 Discrete PSO

The original PSO described in section - (6.3.2) is basically developed for continuous optimization problems. However, lots of practical engineering problems are formulated as combinatorial optimization problems. Kennedy and Eberhart developed a discrete binary version of PSO for the problems [74]. They proposed a model wherein the probability of an agent's deciding yes or no, true or false, or making some other decisions, is a function of personal ans social factors as follows:

$$P(S_i^{(k+1)} = 1) = f(S_i^k, V_i^k, pbest_i, gbest) \tag{6.4}$$

The parameter v, an agent's predisposition to make one or the other choice, will determine a probability threshold. If v is higher, the agent is more likely to choose 1, and lower values favor the 0 choice. Such a threshold requires staying in the range [0,1]. One of the functions accomplishing this feature is sigmoid function, which usually utilized with neural networks.

$$sig(V_i^k) = \frac{1}{1 + \exp(-V_i^k)} \tag{6.5}$$

The agent's disposition should be adjusted for success of the agent and the group. In order to accomplish this, a formula for each V_i^k that will be some function of the difference between the agent's current position and the best positions found so far by itself and by the group. Namely, like the basic continuous version, the formula for binary version of PSO can be described as follows:

$$V_i^{k+1} = V_i^k + rand{\times}(pbest_i - s_i^k) + rand{\times}(gbest - s_i^k) \tag{6.6}$$

$$p_i^{(k+1)} \quad < \quad sig(V_i^{k+1}) \quad then \quad s_i^{k+1} = 1 : \quad else \quad s_i^{k+1} = 0 \tag{6.7}$$

where rand : a positive random number drawn from a uniform distribution with a predefined upper limit.

p_i^{k+1}: a vector of random numbers of [0.0 , 1.0]

In the binary version, the limit of rand is often set so that the two rand limits sum to 4.0. These formulas are iterated repeatedly over each dimension of each agent. The second and third term of RHS of (6.6) can be weighted like the basic continuous version of PSO. vik can be limited so that sig(v_i^k) does not approach too closely to 0.0 or 1.0. This ensures that there is always some chance of a bit flipping. A constant parameter V_{max} can be set at the start of a trial. In practice, V_{max} is often set in [-4.0, +4.0]. The entire algorithm of the binary version of PSO is almost the same as that of the basic continuous version except the above decision equations.

6.4.2 PSO for Mixed-Integer Nonlinear Optimization Problem(MINLP)

Lots of engineering problems have to handle both discrete and continuous variables using nonlinear objective functions. Kennedy and Eberhart discussed about integration of binary and continuous version of PSO [71]. Fukuyama, et al., presented a PSO for MINLP by modifying the continuous version of PSO [75]. The method can be briefly described as follows:

Discrete variables can be handled in (6.1) and (6.3) with little modification. Discrete numbers instead of continuous numbers can be used to express the current position and velocity. Namely, discrete random number is used for rand in (6.1) and the whole calculation of RHS of (6.1) is discretized to the existing discrete number. Using this modification for discrete numbers, both continuous and discrete number can be handled in the algorithm with no inconsistency. In [75], the PSO for MINLP was successfully applied to a reactive power and voltage control problem with promising results.

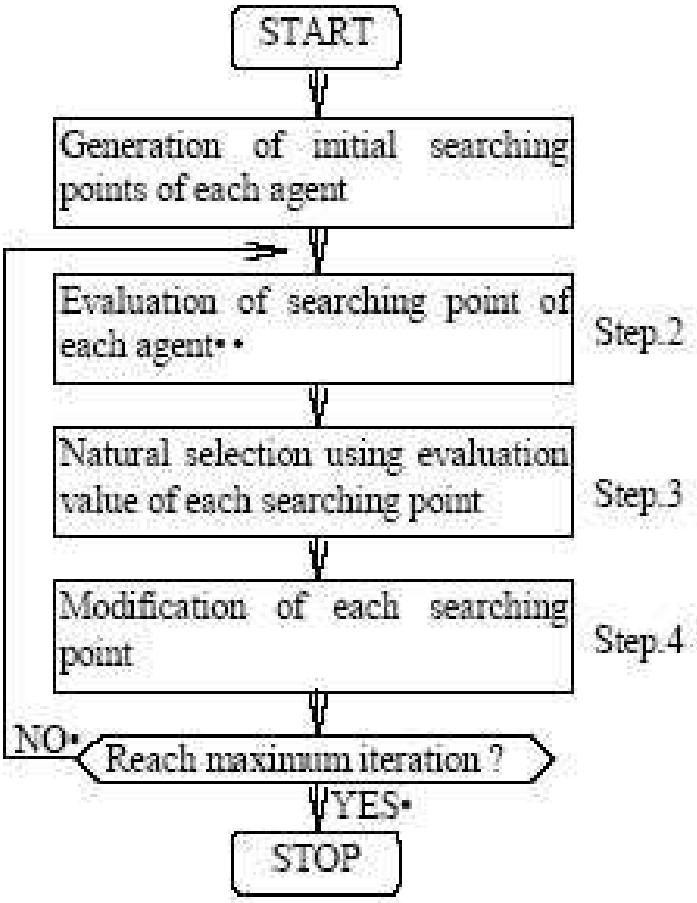


Figure 6.4: A general flow chart of HPSO

6.4.3 Hybrid PSO (HPSO)

HPSO utilizes the basic mechanism of PSO and the natural selection mechanism, which is usually utilized by EC methods such as GAs. Since search procedure by PSO deeply depends on pbest and gbest, the searching area may be limited by pbest and gbest. On the contrary, by introduction of the natural selection mechanism, effect of pbest and gbest is gradually vanished by the selection and broader area search can be realized. Agent positions with low evaluation values are replaced by those with high evaluation values using the selection. The exchange rate at the selection is added as a new optimization parameter of PSO. On the contrary, pbest information of each agent is maintained. Therefore, both intensive search in a current effective area and dependence on the past high evaluation position are realized at the same time. Fig. 6.4 shows a general flow chart of

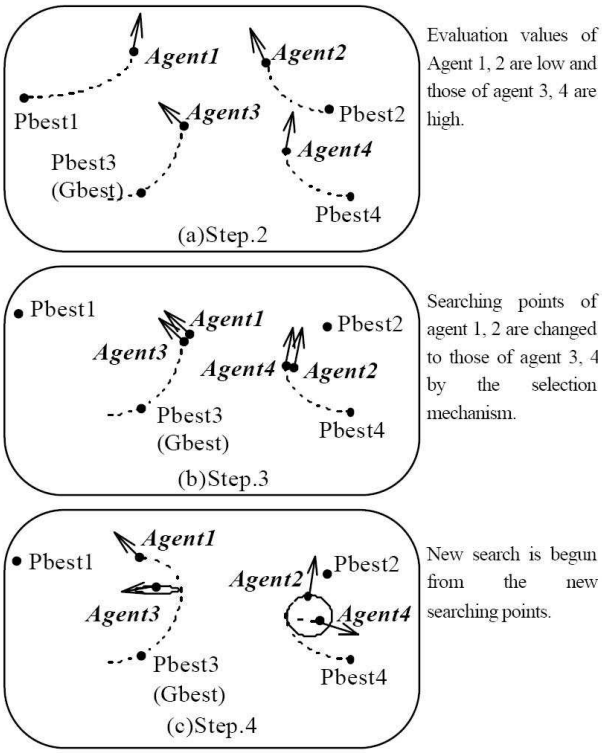


Figure 6.5: . Concept of searching process by HPSO

HPSO. Fig. 6.5 shows concept of step. 2, 3, and 4 of the general flow chart.

6.4.4 Lbest model

Eberhart and Kennedy called the above-mentioned basic method as "gbest model". They also developed "lbest model" [71]. In the model, agents have information only of their own and their nearest array neighbor' bests (lbests), rather than that of the entire group. Namely, in (6.1), gbest is replaced by lbests in the model.

6.5 Parameter selections and constriction factor approach

6.5.1 Parameter selection

PSO has several explicit parameters whose values can be adjusted to produce variations in the way the algorithm searches the solution space. The parameters in (6.1)(6.2) are as follows:

c_j : weighting factor,

W_{max} : initial weight of the weight function,

W_{min} : final weight of the weight function.

Shi and Eberhart tried to examine the parameter selection of the above parameters [76, 77]. According to their examination, the following parameters are appropriate and the values do not depend on problems:

$c_j = 2.0$, $W_{max} = 0.9$, $W_{min} = 0.4$

6.5.2 Constriction factor

The basic system equation of PSO (6.1), (6.2) and (6.3) can be considered as a kind of difference equations. Therefore, the system dynamics, namely, search procedure, can be analyzed by the eigen value analysis. The constriction factor approach utilizes the eigen value analysis and controls the system behavior so that the system behavior has the following features [78]:

1. The system does not diverge in a real value region and finally can converge,
2. The system can search different regions efficiently.

The velocity of the constriction factor approach (simplest constriction) can be expressed as follows instead of (6.1) and (6.2):

$$V_i^{k+1} = k[V_i^k + c_1 \times rand \times (pbest_i - s_i^k) + c_2 \times rand \times (gbest - s_i^k)] \tag{6.8}$$

$$K = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|} \tag{6.9}$$

6.6 SIMULATION RESULTS

6.6.1 STEPWISE REPESENTATION OF PSO BASED CHANNEL EQUALIZATION ALGORITHM:

The updating of weights of the PSO based equalizer is carried out as outlined in the following steps:

1. The structure of the equalizer is a FIR system whose coefficients are initially chosen from a population of M particles (birds). Each particle constitutes p number of parameters and each parameter represents one coefficient of the equalizer.

2. Generate $K(K \geq 500)$ number of input signal samples which are random binary in nature.

3. Each of the input samples is passed through the channel and then contaminated with additive noise of known strength. The resultant signal is passed trough the equalizer. In this way K numbers of estimated samples are produced by feeding all the K input samples.

4. Each of the input samples is passed through the channel and then contaminated with additive noise of known strength. The resultant signal is passed trough the equalizer. In this way K numbers of estimated samples are produced by feeding all the K input samples.

5. Each of the input sample is delayed which acts as desired signal.

6. Each of the desired output is compared with the corresponding channel output and K errors are produced. The mean square error (MSE) for a given group of parameters (corresponding to n^{th} particle) is determined by using the relation.
   $MSE(n) = \frac{\sum_{i=1}^{K} e_i^2}{K}$. This is repeated for M times.

7. Since the objective is to minimize MSE (n), n = 1 to M the PSO based optimization is used.

8. The velocity and position of each bird is updated using equation (6.1) and (6.3) as given in section-6.2.2.

9. In each iteration the minimum MSE, MMSE (l) (expressed in dB) is stored which shows the learning behavior of adaptive model from iteration to iteration.

10. When the MMSE (l) has reached the pre-specified level the optimization is stopped.

11. At this step all the particles attend almost identical position, which represents the desired filter coefficients of the equalizer.

6.6.2 COMPUTER SIMULATIONS:

In this section we carry out the simulation study of new channel equalizer. The coefficients of the equalizer are updated using GA, PSO and LMS algorithm. The results of two different linear and nonlinear channels are used. While training, the additive noises used in the channel are -30dB (low noise), -10dB (medium noise) and 0dB (high noise) to test the performance of the three different algorithms in different noise conditions. Finally the performance of the equalizers is tested by plotting the Bit-error-rate (BER).
The following linear channel models are used:

1. CH1: H (z) = $0.2600 + 0.9300z^{-1} + 0.2600z^{-2}$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

2. CH2: H (z) = $0.3410 + 0.8760z^{-1} + 0.3410z^{-2}$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

The following nonlinear channel models are used:

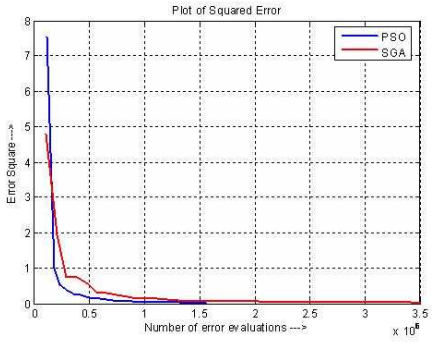
1. NCH1: H (z) = $0.2600 + 0.9300z^{-1} + 0.2600z^{-2}$
   $b(k) = tanh[a(k)]$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

2. NCH1: H (z) = $0.2600 + 0.9300z^{-1} + 0.2600z^{-2}$
   $b(k) = a(k) + 0.2a^2(k) - 0.1a^3(k) + 0.5cos(\Pi a(k))$
   NSR = -30 dB, NSR = -10 dB and NSR = 0 dB

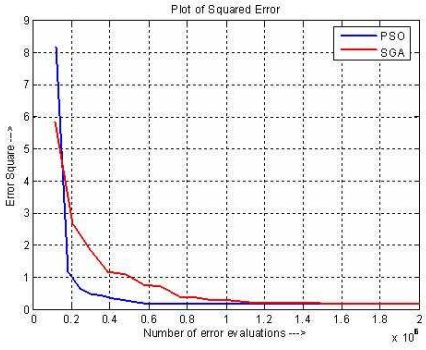
Where b(k) is the output of the nonlinear channel.
The desired signal is generated by delaying the input binary sequence by m samples

where $m = \frac{N}{2}$ or $\frac{N+1}{2}$ depending upon N is even or odd where N represents the order of the channel. In the simulation study N = 8 has been taken. For LMS algorithm, mu = 0.02. For PSO, M (swarm size) = 120, $c_1 = c_2 = 0.7$, w = 0.5 and total number of iterations = 40. Similarly for binary coded GA (BGA), population size (M) = 40, total number of bits used to represent each chromosome = 120 (i.e. 15 bits per variable), R_{min} = -2, R_{max} = 2 (where R_{min} and R_{max} represents the range or boundary values), Pc(Probability of crossover) = 0.9 and Pm(Probability of mutation) = 0.03. Again tournament selection is preferred which is followed by two point crossover.

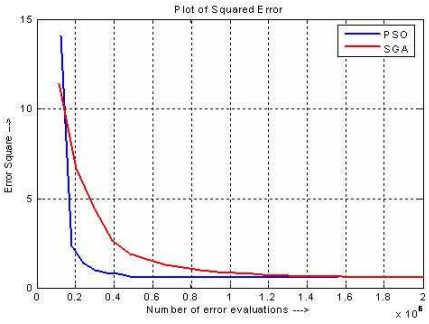
The convergence characteristics of BGA and PSO is obtained from simulation and is shown in Fig.6.6(a, b, c, d, e and f) and Fig. 6.7(a, b, c, d, e and f) for the linear channels and nonlinear channels respectively. Similarly the bit error plot (BER) of LMS, BGA and PSO equalizers for linear and nonlinear channels are shown in Fig. 6.8 (a, b, c, d, e and f) and Fig 6.9 (a, b, c, d, e and f) respectively. These results are used for comparison.



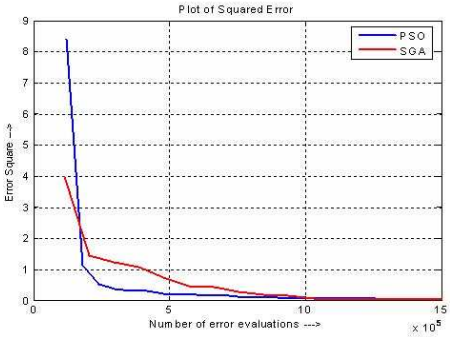
(a) CH1, SNR= 30dB



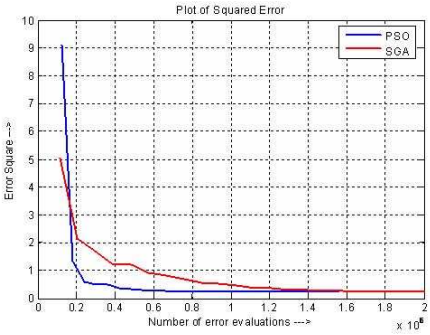
(b) CH1, SNR= 10dB



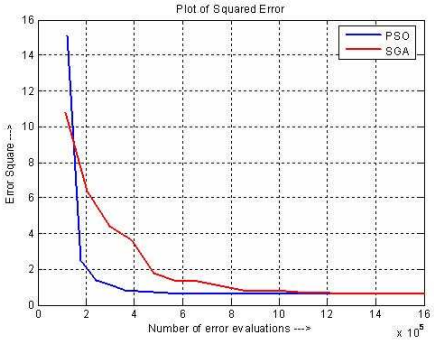
(c) CH1, SNR= 0dB



(d) CH2, SNR= 30dB

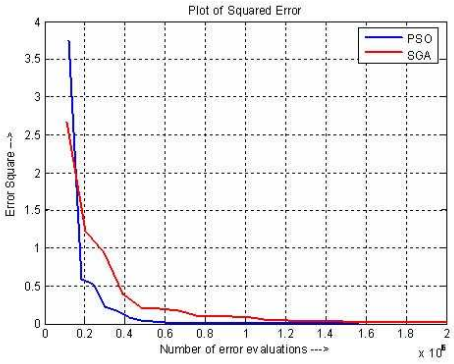


(e) CH2, SNR= 10dB

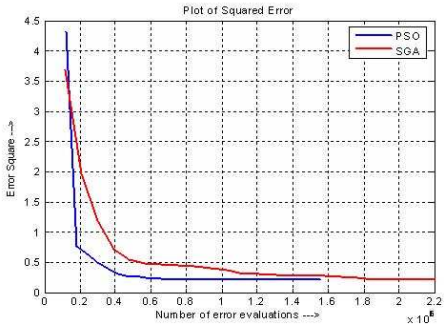


(f) CH2, SNR= 0dB

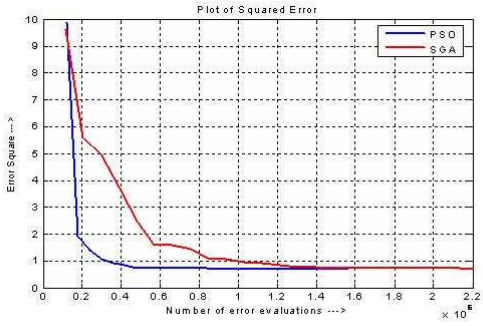
Figure 6.6: Comparison of convergence characteristics of various linear channels between PSO and SGA based equalizer at different noise conditions



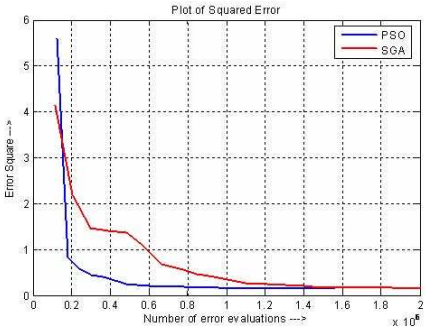
(a) NCH1, SNR= 30dB



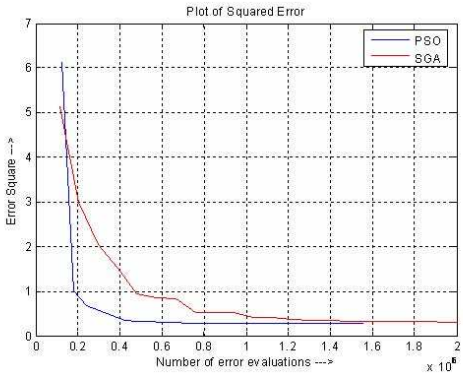
(b) NCH1, SNR= 10dB



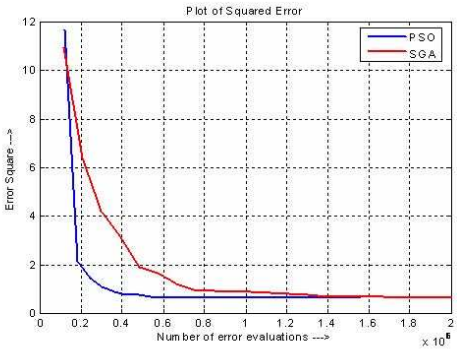
(c) NCH1, SNR= 0dB



(d) NCH2, SNR= 30dB

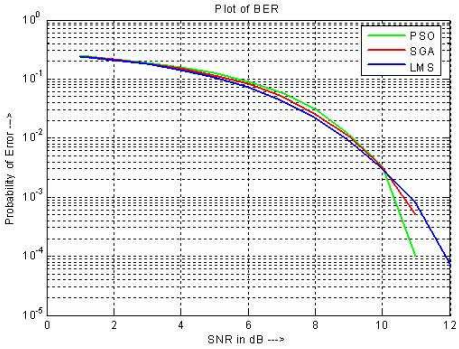


(e) NCH2, SNR= 10dB

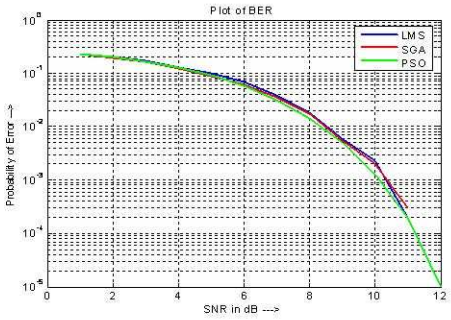


(f) NCH2, SNR= 0dB

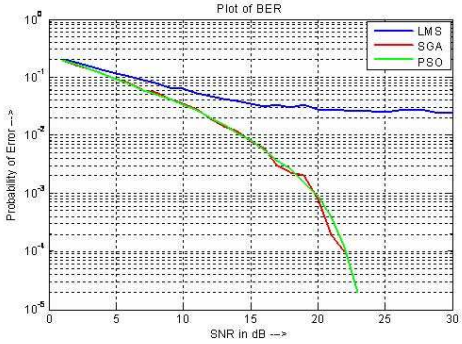
Figure 6.7: Comparison of convergence characteristics of various nonlinear channels between PSO and SGA based equalizer at different noise conditions



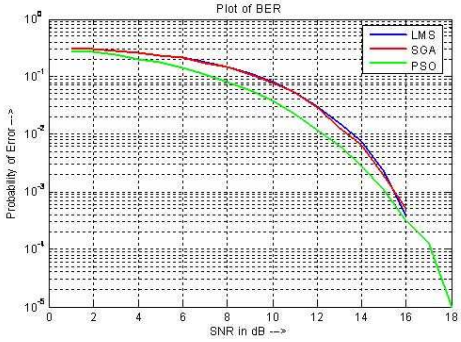
(a) CH1, SNR=30dB



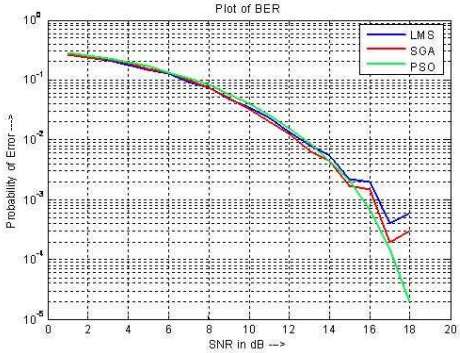
(b) CH1, SNR=10dB



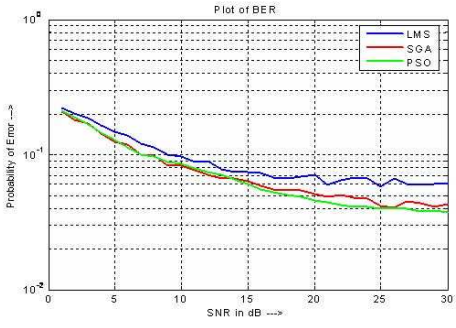
(c) CH1, SNR=0dB



(d) CH2, SNR=30dB

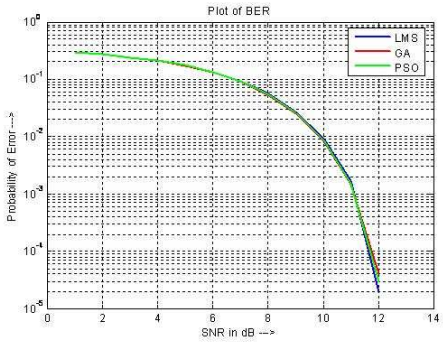


(e) CH2, SNR=10dB

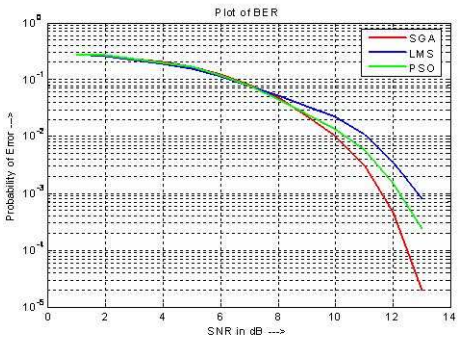


(f) CH2, SNR=0dB

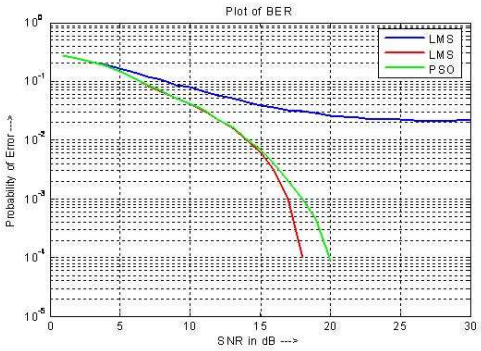
Figure 6.8: Comparison of BER of various linear channels between LMS, SGA and PSO based equalizer different noise conditions



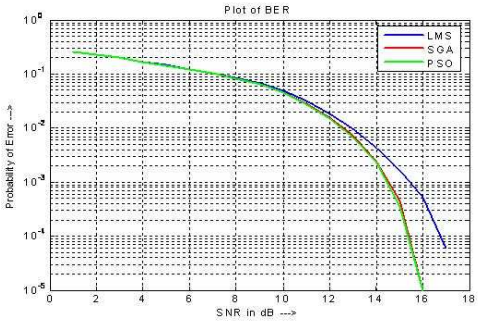
(a) NCH1, SNR=30dB



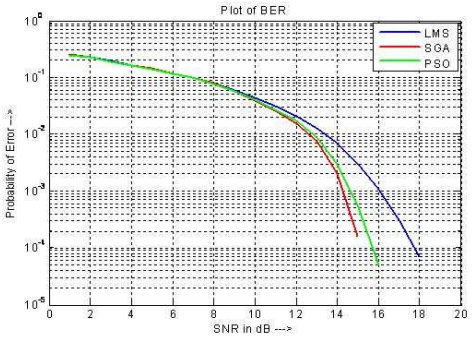
(b) NCH1, SNR=10dB



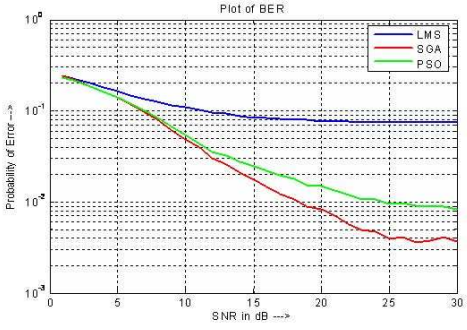
(c) NCH1, SNR=0dB



(d) NCH2, SNR=30dB



(e) NCH2, SNR=10dB



(f) NCH2, SNR=0dB

Figure 6.9: Comparison of BER of various nonlinear channels between LMS, SGA and PSO based equalizer different noise conditions

Table 6.1: Comparison of CPU Time

Algorithm used in equalization	Approximate CPU Time (in seconds)
LMS	1 - 1.5
SGA	20 - 23
PSO	5 - 6

* CPU Times are measured under similar conditions.

6.7 CONCLUSION

it can be concluded from the above section that:-

1. For linear channels, the performance of PSO equalizer is better than SGA and LMS equalizers in terms of BER plot. Again PSO exhibits faster convergence than SGA.

2. For nonlinear channels, PSO equalizer outperforms LMS equalizer in terms of BER plot but it performance slightly degrades from SGA equalizer under high noise condition. But PSO exhibits faster convergence than SGA.

6.7.1 COMPARISON OF CONVERGENCE SPEED

It is clear from the Table6.1 that PSO exhibits faster convergence than SGA although it is slower than LMS algorithm.

6.8 SUMMARY

This chapter introduced the concept that how swarm intelligence can be used to solve an optimization problem. The basic principles are discussed and the different variations of PSO are also dealt with this chapter. It is also discussed how channel equalization can be treated as a squared error optimization technique.

The performance of the PSO equalizer is compared with that of SGA and LMS based equalizer and it is concluded that PSO equalizer outperforms SGA equalizer in terms of the convergence speed although its BER performance slightly degrades under high noise condition for nonlinear channels.

Chapter 7

Efficient adaptive identification structures using GA based pruning

7.1 Introduction

System identification is a pre-requisite to analysis of a dynamic system and design of an appropriate controller for improving its performance. The more accurate the mathematical model identified for a system, the more effective will be the controller designed for it. In many identification processes, however, the obtainable model using available techniques is generally crude and approximate.

In conventional identification methods, a model structure is selected and the parameters of that model are calculated by optimizing an objective function. The methods typically used for optimization of the objective function are based on gradient descent techniques. On-line system identification used to date are based on recursive implementation of off-line methods such as least squares, maximum likeli-hood or instrumental variable. Those recursive schemes are in essence local search techniques. They go from one point in the search point to another at every sampling instant, as a new input-output pair becomes available. This process usually requires a large set of input/output data from the system which is not always available. In addition the obtained parameters may be locally optimal.

Gradient-descent training algorithms are the most common form of training algorithms in signal processing today because they have a solid mathematical

foundation and have been proven over the last five decades to work in many environments. Gradient-descent training, however leads to suboptimal performance under nonlinear conditions. Genetic Algorithm (GA) [79] has been widely used in many applications to produce a global optimal solution. This approach is a probabilistically guided optimization process which simulates the genetic evolution. The algorithm cannot be trapped in local minima as it employs a random mutation procedure. In contrast to classical optimization algorithm, genetic algorithms are not guided in their search process by local derivatives. Through coding the variables population with stronger fitness are identified and maintained while population with weaker fitness are removed. This process ensures that better offsprings are produced from their parents. This search process is stable and robust and can identify global optimal parameters of a system. The underlying principles of GA's were first published by Holland in [80].GA has been used in many diverse areas such as function optimization [81],image processing [82], the traveling salesman problem [83, 84] and system identification [84–87].

In this thesis GA is used for simultaneously pruning and weight updation. While constructing an artificial neural network [88, 89] the designer is often faced with the problem of choosing a network of the right size for the task to be carried out. The advantage of using a reduced neural network is less costly and faster in operation. However, a much reduced network cannot solve the required problem while a fully ANN may lead to accurate solution. Choosing an appropriate ANN architecture of a learning task is then an important issue in training neural networks. Giles and Omlin [90] have applied the pruning strategy for recurrent networks. Markel has employed [91] the pruning technique to FFT algorithm. He has eliminated those operations which do not contribute to estimate output. Jearanaitanakij and Pinngern [92] have analyzed on the minimum number of hidden units that is required to recognize English capital letters using ANN. Thus to achieve the cost and speed advantage, appropriate pruning of ANN structure is required. In this chapter we have considered an adequately expanded FLANN model for the identification of nonlinear plant and then used Genetic Algorithm

(GA) to train the filter weights as well to obtain the pruned input paths based on their contributions. Procedure for simultaneous pruning and training of weights have been carried out in subsequent sections to obtain a low complexity reduced structure.

7.2 PRUNING USING GA:

In this Section a new algorithm for simultaneous training and pruning of weights using binary coded genetic algorithm (BGA) is proposed. Such a choice has led to effective pruning of branch and updating of weights. The pruning strategy is based on the idea of successive elimination of less productive paths (functional expansions) and elimination of weights from the FLANN architecture. As a result, many branches (functional expansions) are pruned and the overall architecture of the FLANN based model is reduced which in turn reduces the corresponding computational cost associated with the proposed model without sacrificing the performance. Various steps involved in this algorithm are dealt in this section.

1. **Step 1-** Initialization in GA:

   A population of M chromosomes is selected in GA in which each chromosome constitutes (TE+1)L number of random binary bits where the first L number of bits are called Pruning bits (P) and the remaining bits represent the weights associated with various branches (functional expansions) of the FLANN model. Again (T - 1) represents the order the filter and E represents the number of expansions specified for each input to the filter. Thus each chromosome can be schematically represented as shown in the Fig. 4.6.

   A pruning bit (p) from the set P indicates the presence or absence of expansion branch which ultimately signifies the usefulness of a feature extracted from the time series. In other words a binary 1 will indicate that the corresponding branch contributes and thus establishes a physical connection where as a 0-bit indicates that the effect of that path is insignificant and hence can be neglected. The remaining (T.E.L) bits represent the (T.E) weight values of the model each containing L bits.

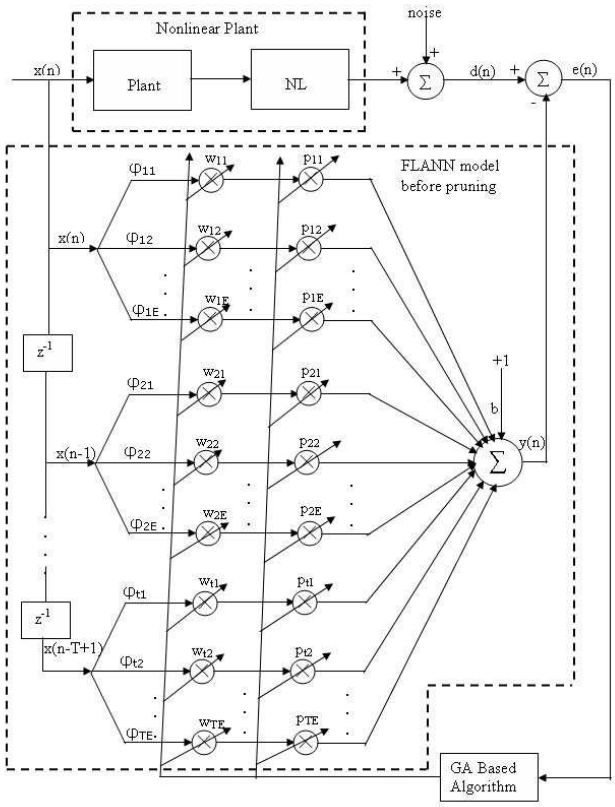


Figure 7.1: FLANN based identification model showing updating weights and pruning path

2. **Step 2-**Generation of input training data:

   $K(\geq 500)$ number of signal samples is generated. In the present case two different types of signals are generated to identify the static and feed forward dynamic plants.

   (a) To identify a feed forward dynamic plant, a zero mean signal which is uniformly distributed between ± 0.5 is generated.

   (b) To identify a static system, a uniformly distributed signal is generated within ± 1. Each of the input samples are passed through the unknown plant (static and feed forward dynamic plant) and K such outputs are obtained. The plant output is then added with the measurement noise (white uniform noise) of known strength, there by producing k number of desired signals. Thus the training data are produced to train the

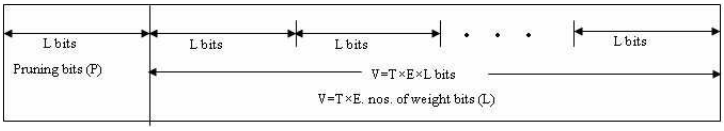


Figure 7.2: Bit allocation scheme for pruning and weight updating network.

3. **Step 3-**Decoding:

   Each chromosome in GA constitutes random binary bits. So these chromosomes need to be converted to decimal values lying between some ranges to compute the fitness function. The equation that converts the binary coded chromosome in to real numbers is given by

$$RV = R_{min} + \frac{R_{max} - R_{min}}{2^L - 1} \times DV \tag{7.1}$$

   Where $R_{min}, R_{max}, RV and DV$ represent the minimum range, maximum range, decimal and decoded value of an L bit coding scheme representation. The first L number of bits is not decoded since they represent pruning bits.

4. **Step 4-** To compute the estimated output:

   At n^{th} instant the estimated output of the neuron can be computed as

$$y(n) = \sum_{i=1}^{T} \sum_{j=1}^{E} \phi_{ij}(n) \times w_{ij}^{m}(n) \times P_{ij}^{m}(n) + b^{m}(n) \tag{7.2}$$

   Where $\varphi_{ij}(n)$ represents jth expansion of the ith signal sample at the n^{th} instant. $W_{ij}^{m}(n)$ and $P_{ij}^{m}(n)$ represent the j^{th} expansion weight and j^{th} pruning weight of the i^{th} signal sample for m^{th} chromosome at k^{th} instant. Again $b^{m}(n)$ corresponds to the bias value fed to the neuron for m^{th} chromosome at n^{th} instant.

5. **Step 5-** Calculation of cost function:

   Each of the desired output is compared with corresponding estimated output and K errors are produced. The Mean-square-error (MSE) corresponding to m^{th} chromosome is determined by using the relation:

$$MSE(m) = \sum_{k=1}^{K} \frac{e_k^2}{K} \tag{7.3}$$

This is repeated for M times (that is for all the possible solutions).

6. **Step 6-** Operations of GA:

   Here the GA is used to minimize the MSE. The crossover, mutation and selection operators are carried out sequentially to select the best M individuals which will be treated as parents in the next generation.

7. **Step 7-** Stopping Criteria:

   The training procedure will be ceased when the MSE settles to a desirable level. At this moment all the chromosomes attain the same genes. Then each gene in the chromosome represents an estimated weight.

7.3 SIMULATION RESULTS:

Extensive simulation studies are carried out with several examples from static as well as feed forward dynamic systems. The performance of the proposed Pruned FLANN model is compared with that of basic FLANN structure.

1. **Static Systems**

   Here different nonlinear static systems are chosen to examine the approximation capabilities of the basic FLANN and proposed Pruned FLANN models. In all the simulation studies reported in this Section a single layer FLANN structure having one input node and one neuron is considered. Each input pattern is expanded using trigonometric polynomials i.e. by using $cos(n\Pi u)$ and $sin(n\Pi u)$, for n = 0,1,2,6. In addition a bias is also fed to the output. In the simulation work the data used are K = 500, M = 40, N = 15, L = 30, probability of crossover = 0.7 and probability of mutation = 0.1. Besides that the R_{max} and R_{min} values are judiciously chosen to attain satisfactory results. Three nonlinear static plants considered for this study are as follows:

   (a) **Example-1**: $f_1(u) = u^3 + 0.3u^2 - 0.4u$

(b) **Example-2**: $f_2(u) = 0.6sin(\Pi u) + 0.3sin(3\Pi u) + 0.1sin(5\Pi u)$

(c) **Example-3**: $f_3(u) = \frac{4u^3-1.2u^2+1.2}{0.4u^5+0.8u^4-1.2u^3+0.2u^2-3}$

At any n^{th} instant, the output of the ANN model y (n) and the output of the system d (n) is compared to produce error e(n) which is then utilized to update the weights of the model. The LMS algorithm is used to adapt the weights of basic FLANN model where as a proposed GA based algorithm is employed for simultaneous adaptation of weights and pruning of the branches. The basic FLANN model is trained for 30000 iterations where as the pruned FLANN model is trained for only 60 generations. Finally the weights of the ANN are stored for testing purpose. The responses of both the networks are compared during testing operation and shown in Figs.7.3 (a), (b), (c). The comparison of computational complexity between FLANN and pruned FLANN is given in Table7.1 .

Table 7.1: Comaprison of Computational Complexities between a basic FLANN and Pruned FLANN model

	Number of operations				Number of weights	
		Additions	Multiplication			
Treatment	FLANN	Pruned FLANN	FLANN	Pruned FLANN	FLANN	Pruned FLANN
Ex-1	14	3	14	3	15	4
Ex-2	14	2	14	3	15	3
Ex-3	14	5	14	5	15	6

2. **Dynamic Systems**

In the following the simulation studies of nonlinear dynamic feed forward systems has been carried out with the help of several examples. In each example, one particular model of the unknown system is considered. In this simulation a single layer FLANN structure having one input node and one neuron is considered. Each input pattern is expanded using the direct input as well as the trigonometric polynomials that is by using $u, sin(n\Pi u) and cos(n\Pi u)$, for n = 1. In this case the bias is removed. In the simulation work we have considered K = 500, M = 40, N = 9, L = 20, probability of crossover = 0.7 and probability of mutation = 0.03. Besides that the R_{max} and R_{min} values

are judiciously chosen to attain satisfactory results. The three nonlinear dynamic feed forward plants considered for this study are as follows:

(a) **Example-4**: Parameter of the linear system of the plant [0.2600 , 0.9300 , 0.2600]
Nonlinearity associated with the plant $y_n(k) = y_k + 0.2y_k^2 - 0.1y_k^3$

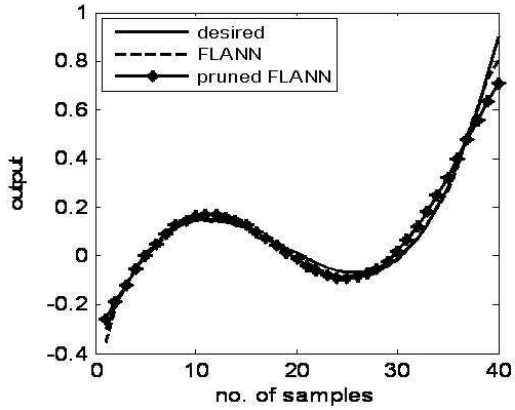
(b) **Example-5**: Parameter of the linear system of the plant [0.3040 , 0.9029 , 0.3040]
Nonlinearity associated with the plant $y_n(k) = tanh(y_k)$

(c) **Example-5**: Parameter of the linear system of the plant [0.3410 , 0.8760 , 0.3410]
Nonlinearity associated with the plant $y_n(k) = y_k - 0.9y_k^3$

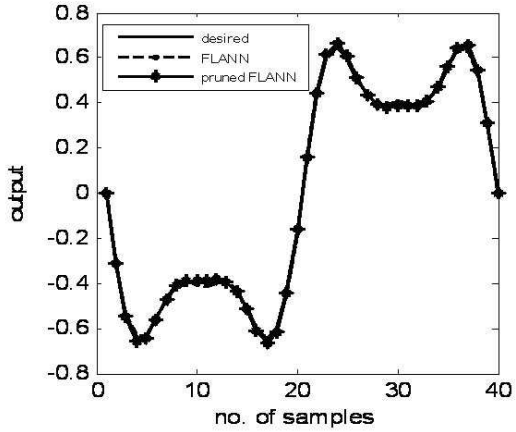
The basic FLANN model is trained for 2000 iterations where as the proposed FLANN is trained for only 60 generations. While training, a white uniform noise of strength -30dB is added to actual system response to assess the performance of two different models under noisy condition. Then the weights of the ANN are stored for testing. Finally the testing of the networks model is undertaken by presenting a zero mean white random signal to the identified model. Performance comparison between the FLANN and pruned FLANN structure in terms of estimated output of the unknown plant has been carried out. The responses of both the networks are compared during testing operation and shown in Fig.7.4 (a), (b), (c). The comparison of computational complexity between FLANN and pruned FLANN is given in Table.7.2.

Table 7.2: Comaprison of Computational Complexities between a basic FLANN and Pruned FLANN model

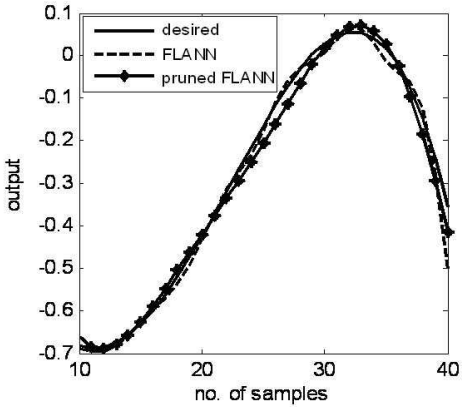
	Number of operations				Number of weights	
		Additions	Multiplication			
Treatment	FLANN	Pruned FLANN	FLANN	Pruned FLANN	FLANN	Pruned FLANN
Ex-1	8	3	9	4	9	4
Ex-2	8	2	9	3	9	3
Ex-3	8	2	9	3	9	3



(a) Ex.1

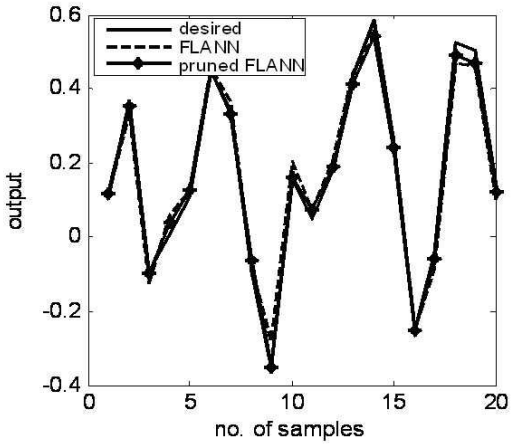


(b) Ex.2

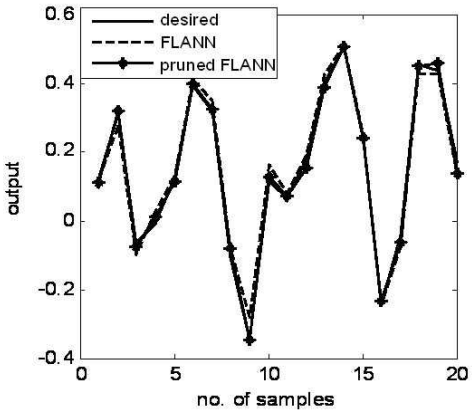


(c) Ex.3

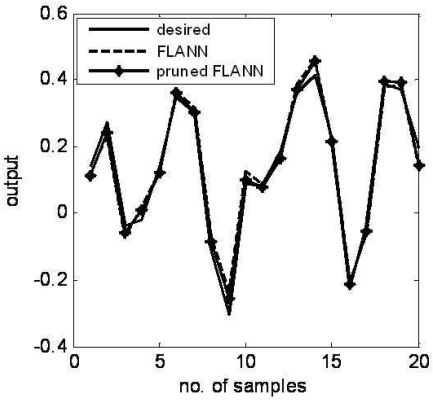
Figure 7.3: Output plots for various static systems



(a) Ex.4



(b) Ex.5



(c) Ex.6

Figure 7.4: Output plots for various dynamic systems

7.3.1 SUMMARY

Simultaneous weight updating and pruning of FLANN identification models using GA is presented. The pruning strategy is based on idea of successive elimination of less productive path. For each weight a separate pruning bit reserved in this process. Computer simulation studies on static and dynamic nonlinear plants demonstrate that there is more than 50 % active paths are pruned keeping response matching almost identical with those obtained from conventional FLANN identification models.

Chapter 8

Conclusion and Future Work

The derivative based algorithms such as Least-Mean-Square (LMS) algorithm, Recursive-Least-Square (RLS) algorithm and Back-propagation (BP) algorithms are associated with local minima problem when these are used to train the weights of the equalizers. Use of these algorithms in the design of adaptive equalizers at times fails to provide satisfactory performance. To alleviate these limitations, the thesis purposes varieties of derivative free optimization techniques such as Genetic Algorithm (Binary coded Genetic Algorithm (BGA), Adaptive Genetic Algorithm (AGA) and Real coded Genetic Algorithm (RCGA)), Particle Swarm Optimization (PSO). These are suitably applied to train the weights of channel equalizers. The performance of these equalizers is evaluated in terms of speed of convergence, computational time and bit-error-rate (BER) and is compared with its LMS based counter part. It is observed that the new set of adaptive equalizers offer improved performance so far as the accuracy of reception is concerned. The results of simulation also reveal that in terms of training time, these equalizers may be arranged as the AGA, PSO, RCGA and BGA based equalizers.

Being a population based algorithm, the standard Genetic Algorithm (SGA) suffers from slower convergence rate. To minimize the training time three different adaptive GAs (AGAs) are proposed in the thesis and their convergence times have been compared. It is observed that keeping the bit-error-rate (BER) performance same, the AGA equalizer requires less training time (4-5s) as compared to the training time of SGA (20-23s). The thesis also investigates on the new equalizers using RCGA to resolve coding-decoding and boundary limit issues. The perfor-

mance of RCGA based equalizer is evaluated and compared with the LMS and BGA equalizers. It is observed that the RCGA based approach requires less training time (17s) as compared to the training time required by of the BGA (20-23s).

The Particle Swarm Optimization (PSO) is studied and used in training the equalizer weights. Unlike GA, PSO utilizes its past memory and share each other's experience to reach at the global optimum. The performance of the PSO equalizer is obtained and compared with that of the SGA and LMS based equalizers. It is found that retaining the same BER performance, the PSO based method takes much lesser training time (5-6s) as compared to the training time offered by the SGA equalizer (20-23s).

Finally the BGA is employed as a pruning algorithm to find the optimal architecture of a Functional Link Artificial Neural Network (FLANN) to solve system identification problem. It is observed that about 50% of the total signal paths can be pruned keeping the performance of the pruned structure identical to that of the original FLANN structure.

In summery, the present thesis has proposed a novel approach of employing the GA and PSO optimization tools for training the weights of the adaptive channel equalizer. The results obtained through simulation study are observed and compared with other standard methods. It is demonstrated that the new adaptive GA equalizers outperform the conventional GA based equalizers in terms of performance and training time. Further through simulation study It is shown that the GA approach is a good candidate for pruning the structure of FLANN. Keeping the performance intact, it is observed that about 50% pruning of the structure is possible.

The scope of future work is outlined below:

1. The PSO algorithm used in the thesis is not adaptive in nature. The equalization problem can be solved using various adaptive PSO algorithms. The performance obtained can be compared with those obtained from other standard methods.

2. The convergence analysis of various algorithms is not included in the present work. This problem can also be worked out in future.

3. The real coded GA (RCGA) reduces number of operation (binary to decimal conversion and vice-versa). The RCGA can also be made adaptive and then may be applied in channel equalizers. This will not only enhance the speed of operation but also it will improve the performance. This problem can also be tried.

Bibliography

[1] J. Patra, R. Pal, B.N.Chatterji, and G. Panda, "Identification of nonlinear dynamic systems using functional link artificial neural networks," *IEEE Transactions on Systems,Man and cybernetics*, vol. 29, no. 2, pp. 254–262, April 1999.

[2] S. Qureshi, "Adaptive equalization," *IEEE Communications Magazine*, vol. 73, no. 9, pp. 1349–1387, Sept 1985.

[3] C. Siller, "Multipath propagation," *IEEE Communications Magazine*, vol. 22, no. 2, pp. 6–15, Feb 1984.

[4] S. S.Chen, B.Mulgrew, "Adaptive bayesian equaliser with decision feedback," *IEEE Trans Signal Processing*, vol. 41, no. 9, pp. 2918–2927, Sept 1993.

[5] C. G.Gibson, S.Siu, Ed., *Multi-layer perceptron structures applied to adaptive equalisers for data communication*, ser. 1183-1186, University of Glasgow. UK: Proc. IEEE Int. Conf. ASSP, May 1989.

[6] C. G.Gibson, S.Siu, "Decision feedback equalisation using neural network structures and performance comparison with standard architecture," in *IEE Proc. Part I, Communication, Speech, and Vision*, vol. 137, no. 4, Aug 1990, pp. 221–225.

[7] C. B. Mulgrew, "Equalization techniques using nonlinear adaptive filters," in *Adaptive Algorithms Applications and Non Classical Schemes*, D.Docampo and A.R.Figueras, Eds., Universidad de Vigo, 1991, pp. 1–19.

[8] S.Chen and A.F.Murray, "Adaptive equalization using neural netwoks," *Applications of Neural Networks,Kluwer*, pp. 241–265, 1995.

[9] W. C.Eng-Siong, H.Yang, "Reduced complexity implementation of bayesian equaliser using local rbf network for channel equalisation problem," *IEE Electronics Letters*, vol. 32, no. 1, pp. 17–19, Jan 1996.

[10] I. J.Park, "Universal approximation using radial basis function networks," *IEEE Transactions on Neural Networks*, vol. 3, pp. 246–257, 1991.

[11] S.Haykin, *'Neural Networks - A comprehensive foundation*, first edition ed. New York: Macmillan College Publishing, 1994.

[12] B. E.Chng, S.Chen, "Gradient radial basis function networks for nonlinear and nonstationary time series prediction," *IEEE Transactions on Neural Networks*, vol. 7, no. 1, pp. 190–4, Jan 1996.

[13] B. S.Chen, Y.Wu, "Combined genetic algorithm optimization and regularized orthogonal least squares learning for radial basis function networks," *IEEE Transactions on Neural Networks,*, vol. 10, no. 5, pp. 1239–43, Sept 1999.

[14] K.Funahashi, "On the approximate realisation of continuous mappings by neural networks," *IEEE Transactions on Neural Networks,*, vol. 2, pp. 183–192, 1989.

[15] F.Sweeney, "Optimisation for non-linear channel equalization," Ph.D. dissertation, The Queen's University of Belfast, 1999.

[16] J.C.Patra, R.N.Pal, R.Baliarsingh, and G.Panda, "Nonlinear channel equalization for qam signal constellation using artificial neural networks," *IEEE Trans. Syst. Man Cybern. B, Cybern*, vol. 2, no. 29, pp. 262–271, 1999.

[17] T. W.S.Gan, J.J.Sorahgan, Ed., *Functional-link models for adaptive channel equalizer*, vol. 3. IEEE Proc. Acoustics, Speech, and Signal Processing, 1994.

[18] D.Goldberg, *Genetic algorithms in search optimization.* Addison-Wesley, 1989.

[19] W.D.Weng and C.T.Yen, "Reduced-decision feedback flann nonlinear channel equaliser for digital communication systems," in *IEE Proceedings Communications*, vol. 151, no. 4, Aug 2004, pp. 305–311.

[20] C. A. W. S.Ng, S.Leung, "The genetic search approach," *IEEE signal processing magazine*, vol. 13, no. 6, pp. 38–46, Nov 1996.

[21] S.Siu, "Non-linear adaptive equalisation based on multi-layer perceptron architecture," Ph.D. dissertation, University of Edinburgh, 1990.

[22] S. S.Chen, B.Mulgrew, "Adaptive bayesian equaliser with decision feedback," *IEEE Trans Signal Processing*, vol. 41, no. 9, pp. 2918–2927, Sept 1993.

[23] S. B.Widrow, *Adaptive signal processing.* New Jersey: Prentice-Hall Signal processing series, 1985.

[24] O.Macchi, *Adaptive processing, the least mean squares approach with applications in transmission.* West Sussex. England: John Wiley and Sons, 1995.

[25] X. I.Reed, *Error control coding for data networks.* Boston: Kluwer academic publishers, 1999.

[26] G.Forney, "Maximum-likelihood sequence estimation of digital sequences in the presence of inter-symbol interference," *IEEE Trans, inform, theory*, vol. 18, pp. 363–378, 1972.

[27] B. Widrow and S. Sterns, *Adaptive Signal Processing.* Pearson Education, Inc., 1985.

[28] E. Walach and B. Widrow, "The least mean forth (lmf) adaptive algorithm and its family," *IEEE Transaction on Information Theory*, vol. 30, no. 2, pp. 275–283, Mar 1984.

[29] B.Friedlander, Ed., *Lattice filters for adaptive processing*, ser. 829-867, vol. 70(8). Proc. IEEE, Aug 1982.

[30] B. Widrow and S. D. Stearns, *Adaptive Signal Processing.* Upper Saddle River, NJ: Prentice-Hall,Inc., 1985.

[31] C. Imen, A. Sabeur, and F. Farhat, "Genetic approach for linear quadratic channel identification with usual communication inputs," in *International joint conference on Neural Networks*, 12-17 Aug 2007, pp. 1703–1707.

[32] T.Kumon, M.Iwasaki, T.Suzuki, T.Hasiyama, N.Matsui, and S.Okuma, "Nonlinear system identification using genetic algorithm," in *International Elecronics society (IECON)*, vol. 4. IEEE, 22-28 Oct 2000, pp. 2485–2491.

[33] L. Al-Baba, S.L.Horner, W.Holls, and P.B.Crilly, "Adaptive echo cancellation using genetic algorithm," in *Proceeding of 1992 International conference on Industrial Electronics,control,Insrumentationand automation*, vol. 2. IEEE, 1992, pp. 1041–1044.

[34] S. Ohta, Y. Kajikawa, and Y. Nomura, "Acoustic echo cancellation using sub adaptive filter," in *International symposium on Intelligent signal processing and communications(ISPACS).* IEEE, 12-15 Dec 2006, pp. 841–844.

[35] F. Zhigang, C. Xizhi, and H.Huang, "An improved adaptive noise cancelling method," in *Canadian conference on Electrical computer engeering*, vol. 1. IEEE, 14-17 Sept 1993, pp. 47–50.

[36] R.W.Lucky, "Techniques for adaptive equalization of digital communication systems," *Bell Sys. Tech. J.*, pp. 255–286, Feb 1966.

[37] E.A.Robinson and T.Durrani, *Geophysical Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1986.

[38] N.Wiener, *Extrapolation, Interpolation, and Smoothing of Stationary Time Series with Engineering Applications.* Cambridge, MA: MIT Press, 1949.

[39] N.Levinson, "The wiener rms (root-mean-square) error criterion in filter design and prediction," *Math Phys.*, vol. 25, pp. 261–278, 1947.

[40] W. H.Fan, "A new adaptive iir filter," *IEEE Trans. Circuits and Systems*, vol. 33, no. 10, pp. 939–947, Oct 1986.

[41] K.C.Ho and Y.T.Chan, "Bias removal in equation-error adaptive iir filters," *IEEE Trans. Signal Processing*, vol. 43, no. 1, Jan 1995.

[42] J.E.Cousseau and P.S.R.Diniz, "New adaptive iir filtering algorithms based on the steiglitz-mcbride method," *IEEE Trans. Signal Processing*, vol. 45, no. 5, May 1997.

[43] I. M. W. K.L.Blackmore, R.C.Williamson, "Online learning via congregational gradient descent," *Mathematics of Control, Signals, and Systems*, vol. 10, pp. 331–363, 1997.

[44] S. Haykin, *Adaptive Filter Theory.* Upper Saddle River, NJ: Prentice-Hall Inc., 1996.

[45] W. Kirkland and D. Taylor, "On the application of feed forward neural networks to channel equalization," in *Proc. IEEE*, vol. 2, no. 2, Jun 1992, pp. 919–924.

[46] E. Z. G. Kechriotis and E. Manolakos, "Using recurrent neural networks for adaptive communication channel equalization," *IEEE Transaction on Neural Networks*, vol. 5, no. 2, pp. 267–278, Mar 1994.

[47] J. M. F.Albu, A. Mateescu and B. Dorizzi, "Adaptive channel equalization using neural network," in *Proc. IEEE on Telecommunications symposium*, vol. 2, Aug 1998, pp. 438–441.

[48] Y. Kim and D. Park, "Nonlinear channel equalization using new neural network model," in *IEEE letter*, vol. 34, no. 23, Nov 1998, pp. 827–830.

[49] R. Pichevar and V. Vakili, "Channel equalization using neural networks," in *IEEE conference on personal wireless communication*, Feb 1999, pp. 240–243.

[50] J. Patra, R. Pal, R. Baliarsingh, and G. Panda, "Nonlinear channel equalization for qam signal constellation using artificial neural network," *IEEE Transaction on Systems, Man and Cybernetics*, vol. 29, no. 2, pp. 262–272, Apr 1999.

[51] D.E.Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[52] T.Chung and H.Leung, "A genetic algorithm approach in optimal capacitor selection with harmonic distortion considerations," *International Journal of Electrical Power and Energy Systems*, vol. 21, no. 8, pp. 561–9, Nov 1999.

[53] G.Jiabao and M.Aral, "Progressive genetic algorithm for solution of optimization problems with non-linear equality and inequality constraints," *Applied Mathematical Modeling*, vol. 23, no. 4, pp. 329–43, Apr 1999.

[54] D. X. Lei, E.Lerch, "Genetic algorithm solution to economic dispatch problems," *European Transactions on Electrical Power*, vol. 9, no. 6, pp. 347–53, Dec 1999.

[55] J. S. A. S.Peigin, B.Mantel, "Application of a genetic algorithm to a heat flux optimization proble," *Surveys on Mathematics for Industry*, vol. 9, no. 3, pp. 235–45, 2000.

[56] Y.Rong, "Solving large travelling salesman problems with small populations," in *IEE Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, 1997, pp. 157–62.

[57] B. H.Sayoud, K.Takahashi, "Minimum cost topology optimisation of atm networks using genetic algorithms," Electronics Letters, pp. 2051–3, Nov 2000.

[58] R. Nambiar and P. Mars, "Genetic and annealing approaches to adaptive digital filtering," in *Proc. IEEE*, vol. 2, Oct 1992, pp. 871–875.

[59] D.Fogel, "What is evolutionary computing," IEEE spectrum magazine, Feb 2000.

[60] Q.Ma, "The application of genetic algorithms to the adaptation of iir filters," Ph.D. dissertation, Loughborough University of Technology, Loughborough, Nov 1995.

[61] C. P.Power, F.Sweeney, "Ea crossover schemes for a mlp channel equalizer," in *6th IEEE International Conference on Electronics, Circuits and Systems.*, vol. 1. Piscataway, NJ, USA: Proceedings of ICECS '99, Sept 1999, pp. 407–411.

[62] P. F.Sweeney and C.Cowan, "The use of evolutionary optimisation in channel equalization," in *Eusipco '98 proceedings*, vol. 4, Rhodes, Greece, Sept 1998, pp. 2221–2224.

[63] P. P. J. Kim, S.W. Kim and T. Park, "On the similarities between binary coded ga and real coded ga in wide search space," in *Proc. IEEE on Evolutionary computation*, vol. 1, May 2002, pp. 681–686.

[64] Y. Z. W.Q. Xiong and P. Wei, "An improved real coded genetic algorithm," in *Proc. IEEE on Machine learning and cybernetics*, vol. 29, no. 2, Aug 2004, pp. 714–719.

[65] M. Srinivas and L. Patanaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Transaction on Systems, Man and Cybernetics*, vol. 24, no. 4, pp. 656–667, Apr 1994.

[66] H. C. J. Zhang and B. Hung, "Adaptive probabilities of crossover and mutation in genetic algorithms based on clustering technique," in *Proc. IEEE on Control systems*, vol. 2, Jun 2004, pp. 2280–2287.

[67] C. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," *Compter Graphics*, vol. 21, no. 4, pp. 25–34, 1987.

[68] M. D. A. Colorni and V. Maniezzo, Eds., *Distributed Optimization by Ant Colonies*, MA: MIT Press. Cambridge: Proc. of First European Conference on Artificial Life, 1991.

[69] J. Kennedy and R. Eberhart, Eds., *Particle Swarm Optimization*, vol. 4, no. pp.1942-1948. Perth, Australia: Proceedings of IEEE International Conference on Neural Networks (ICNN'95), 1995.

[70] M. D. E. Bonabeau and G. Theraulaz, *Swarm Intelligence : From Natural to Artificial Systems*. Oxford Press, 1999.

[71] J. Kennedy and R. Eberhart, *Swarm Intelligence*. Morgan Kaufmann Publishers, 2001.

[72] R. Boyd and P. Recharson, *Culture and the Evolutionary Process*. University of Chicago Press, 1985.

[73] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proceedings of IEEE International Conference on Evolutionary Computation (ICEC'98)*, Anchorage, May 1998, pp. 69–73.

[74] J. Kennedy and R. Eberhart, "A discrete binary version of the particle swarm optimization algorithm," in *Proc. of the 1997 conference on Systems, Man, and Cybernetics (SMC'97)*, 1997, pp. 4104–4109.

[75] Y. Fukuyama, "A particle swarm optimization for reactive power and voltage control considering voltage security assessment," *IEEE Transaction on Power Systems*, vol. 15, no. 4, Nov 2000.

[76] Y. Shi and R. Eberhart, "Parameter selection in particle swarm optimization," in *Proc. of the 1998 Annual Conference on Evolutionary Programming*, San Diego, 1998.

[77] Y. Shi and Eberhart, "A modified particle swarm optimizer," in *Proc. of IEEE International Conference on Evolutionary Computation (ICEC'98)*, Anchorage, Alaska, May 1998, pp. 4–9.

[78] M. Clerc, Ed., *The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization.* Proc. of IEEE International Conference on Evolutionary Computation (ICEC'99), 1999.

[79] J.H.Holland, *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press, 1975.

[80] J. H.Holland, "Outline for a logical theory of adaptive systems," J. ACM, July 1962.

[81] K. A. DeJong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation (CCS), Univ. Mich. ,MI, Ann Arbor, 1975.

[82] J. J. J. M.Fitzpatrick and G. D. Van, Eds., *Image registration by genetic search*, ser. pp.460-464. in Proc. IEEE Southeastcon '84, 1984.

[83] D. E. Goldherg and R.Lingle, Eds., *Alleles, loci and the traveling salesman problem*, ser. pp. 154-159. in Proc. Int. Conf. Genetic Algorithms and Their Appl., 1985.

[84] B. D. J. J. Grefenstette, R.Gopal, Ed., *Genetic algorithms for the traveling salesman problem*, ser. pp. 160-165. in Proc. Int. Conf Genetic Algorithms and Their Applications, 1985.

[85] M. J. D. M. Etter and K. H. Cho, "Recursive adaptive filter design using an adaptive genetic algorithm," in *Proc. IEEE Int. Conf Acoustics, Speech, Signal Processing*, vol. 2, 1982, pp. 635–638.

[86] K.Kristinsson and G. A. Dumont, "Genetic algorithms in system identification," Third IEEE Int. Symp. Intelligent Contr., 1988.

[87] T.Smith and K.A.DeJong, "Genetic algorithms applied to the calibration of information driven models of us migration patterns," in *in Proc. 12th Annu. Pittsburgh Conf Modeling and Simulation*, 1981, pp. 955–959.

[88] Y. Goh and E. Tan, "Pruning neural networks during training by backpropagation," in *IEEE conference on Frontiers of Computer Technology*, vol. 2, Aug 1994, pp. 805–808.

[89] K. H. P.V.S. Ponnapalli and M. Thomson, "A formal selection and pruning algorithm for feed forward neural network optimzation," *IEEE Transaction on Neural Networks*, vol. 10, no. 4, pp. 964–968, Jul 1999.

[90] G. C. Lee and C. W. Omlin, "Pruning recurrent neural networks for improved generalization performance," *IEEE Trans. on Neural Networks*, vol. 5, no. 5, pp. 848–851, Sept 1994.

[91] J.D.Markel, "Fft pruning," *IEEE Trans. on Audio and Electro Acoustics*, vol. AU-19, no. 4, pp. 305–311, Dec 1971.

[92] K.Jearanaitanakij and O.Pinngern, "Hidden unit reduction of artificial neural network on english capital letter reduction," in *IEEE conference on Artificial Intelligence*, June 2006, pp. 1–5.

Dissemination of Work

Published Paper

1. G. Panda, D. Mohanty, B.Majhi and A. Choubey, "*Development of GA Base Adaptive Techniques for Nonlinear System Identification*", Proc. of International Conference on Information Technology (CIT-2005), Bhubaneswar, India, 20-23, December, 2005, pp. 198-204.

2. G. Panda, D. Mohanty, B. Majhi A. Choubey and S. Mishra, "*Development of Novel Digital Channel Equalisers using Genetic Algorithms*", Proc. of National Conference on Communication (NCC-2006), IIT Delhi, India, 27-29,January, 2006, pp.117-121.

3. G. Panda, D. Mohanty, B. Majhi and B. N. Biswal, "*Application of Adaptive Genetic Algorithm in Nonlinear System Identification*", Souvenir of National Conference of Cyber Security, Data Mining and ICT for Society, GG University Bilaspur, India, 18-19, January, 2006.

4. G. Panda, D. Mohanty and Babita Majhi, "*An Adaptive Genetic Algorithm for System Identification : a faster approach*", Proc. of National Conference on Soft Computing Techniques for Engineering Applications (SCT-2006), NIT Rourkela, India, 24-26, March, 2006, pp. 329-336.

5. G. Panda, D. Mohanty and Babita Majhi, "*Development of novel digital nonlinear channel equalizers using Particle Swarm Optimization technique*", National Conference on Data Mining and e-Goverence, GG University, Bilaspur, India, 17th Feb., 2007

6. Debidutta Mohanty, Babita Majhi and G.Panda, "*Recovery of digital data using Real coded Genetic algorithms*" National Seminar on Evolutionary computation, NMIT, Bhubaneswar, Mar. 2007

7. G. Panda, D. Mohanty, B. Majhi and G. Sahoo, "*Identification of Nonlinear Systems using Particle Swarm Optimization Technique*", accepted at IEEE International Congress on Evolutionary Computation(CEC-2007), Singapore to be held during 25-28, September, 2007.

8. G. Panda, D. Mohanty, Babita Majhi and A. Sahoo, "*A GA-based Pruning Strategy and Weight Update Algorithm for Efficient Nonlinear System Identification*", submitted to International Conference on Pattern Recognition and Machine Intelligence (PReMI' 07), ISI, Kolkata, India to be held during 18-22, December,2 007.