Université d'Ottawa · University of Ottawa

REDUCED COMPLEXITY VOLTERRA-TYPE FILTERS FOR NON-LINEAR MODELLING

BY CHRIS JASZCZUR, B. A. SC.

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE DEGREE OF MASTER OF APPLIED SCIENCES IN ELECTRICAL AND COMPUTER ENGINEERING

OTTAWA CARLETON INSTITUTE FOR ELECTRICAL AND COMPUTER ENGINEERING

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

FACULTY OF ENGINEERING

UNIVERSITY OF OTTAWA,

JANUARY 2002

ABSTRACT

Volterra filters are a popular choice for modelling many nonlinear systems, in part due to their generality and simplicity in implementation and adaptation. This thesis addresses one often cited problem with this class of filters: the large increase in the number of terms in the filter for small increases in order or memory. Adaptive algorithms are presented which allow much smaller filters to perform near the performance (in terms of residual error) of the full Volterra filters. The complexity is maintained at reasonable levels by only representing the terms that are critical in reducing the error. An alternative representation with real valued exponents is introduced from the Volterra series representation, and is used to determine the optimum nonlinear filter (in the mean squared error sense) from this class of filters. Examples verifying the performance of the proposed algorithms are presented. These algorithms allow the design of models of much higher order than currently used for such problems as loudspeaker equalization, image processing, and noise removal, among others.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF MAIN SYMBOLS

a0, a1,... , a, b, c	Exponent parameters
d(n)	Desired signal (output of unknown system to be modelled)
e(n)	Error signal
H	A nonlinear system
h, h_0, h_1, ...	Volterra kernels
J	Mean squared error
k, m, n	Integer numbers, used as time indices or polynomial exponents
$r_x(k)$	Autocorrelation function of x
t	Time index
x(t)	Model input (or function input variable)
y(x)	Model output (or function output variable)
$\Theta(x)$	Probability density function
λ	RLS forgetting factor
μ	LMS step size parameter
σ_x	Variance of x

CHAPTER 1-INTRODUCTION

It is the intent of this thesis to investigate and develop methods for compact representation of nonlinear systems. In particular, Volterra filters will be used as a starting model due to their ability to model any continuous nonlinearity. Their practical uses in digital signal processing [12] and redundant modelling characteristics [30] offer the hope of success in this endeavour.

1.1 MOTIVATION

Many practical systems can be approximated fairly well using linear models. The typical system, however, is rarely entirely linear. Usually, the system is highly linear with slight nonlinearities, or in some cases quite non-linear. These nonlinearities increase the overall system error of the model, and produce inaccuracies, which may be significant.

Non-linear models address the need for accurate compensation of non-linear systems. A non-linear system is one in which the input-output relation cannot be represented as a sum of linearly scaled inputs. There are many ways of modelling non-linear systems including piece-wise linear models, neural nets, threshold models, spline function models and polynomial models among others. There is no single unified model for nonlinear systems, so the type of nonlinearity expected will dictate the type of model used. The model of interest for the remainder of this thesis is the Volterra model, which has seen

practical uses in such tasks as echo cancellation, loudspeaker equalization and noise reduction [11].

A Volterra filter is a polynomial with memory. This means that the output can be represented as the sum of terms made up of various products of inputs. It is well suited to modelling systems that are largely linear with slight nonlinearities, although given sufficient order it can model a very general class of nonlinearities. Given a small order Volterra filter, it is possible to accurately describe saturation type nonlinearities, such as those found in amplifiers. Given a much larger order filter, it is possible to model nearly discontinuous systems, such as those with on-off outputs. This thesis addresses one of the drawbacks of using Volterra filters: excessive amounts of parameters for higher order filters [12].

1.2 OBJECTIVES

It is the objective of this thesis to propose methods of obtaining many of the benefits of Volterra filters (non-linear modelling, ease of adaptation, ease of evaluation, generality), while reducing the computational requirements needed by typical truncated Volterra filters. This is done in two ways: by "creatively" truncating the Volterra filter in such a manner as to reduce redundancies between the Volterra terms, and hence maximize modelling power for a restricted size filter, and by searching for the best Volterra terms in a more general function space. These approaches are highlighted in Chapters 5 and 6 respectively.

The benefits of the new algorithms and derivations in this thesis are twofold:

1) Reduction of the complexity of a Volterra series model so a lower complexity truncated model can perform near the performance of the full complexity model.

2) Reduction of the computational requirements of a Volterra series adaptive model so a truncated model of higher order can be used in place of a full complexity lower order model to achieve better modelling than the full complexity lower order model.

These two results are often complementary, if the first is achieved the resultant reduction in complexity allows the flexibility to increase the order of the filter.

1.3 LIMITATIONS

Non-linear modelling encompasses a wide range of techniques to model a large variety of systems. Every application requires study before choosing the correct model. Volterra filters are typically used for low order, low memory applications. This thesis attempts to ease the order restriction, and to a lesser extend the memory restriction, expanding the possible uses of Volterra filters. But even with the help of the algorithms presented in this thesis, there are many applications that are better handled by other methods. For example, systems typically associated with artificial intelligence, such as pattern matching systems are efficiently implemented as neural networks. In general, Volterra systems are well suited to applications where the input/output relationship can be envisioned as a continuous function. Often Volterra filters can be combined with other techniques to yield a practical system, for example: many types of amplifiers are better

modelled using a linear system with long memory followed by a memoryless Volterra system. In this case, the strength of the linear system (effectively modelling a long impulse response), combined with the added benefit of the memoryless Volterra filter (ability to model saturation in the amplifier) results in a superior model over either individual model, and is more efficient than a general Volterra filter with a long memory.

This being said, the thesis discusses algorithms which can be used to extend the use Volterra filters from low order continuous functions to higher order continuous functions.

1.4 THESIS PLAN

This thesis begins with a general discussion of Volterra filters. Volterra filters are initially discussed as continuous time entities. Later the jump to discrete time is made, along with the appropriate conversion. As well, this thesis will examine the advantages and limitations of Volterra filters.

Zeroing in on our goal, *adaptive* Volterra filters are discussed next. The well established LMS and RLS algorithms, modified for Volterra filters, are looked at in this discussion. Examples are used to demonstrate these adaptive Volterra filters.

A discussion of the redundancies of Volterra filters follows. By being aware of these redundancies, it is possible to obtain a more compact representation of a given non-linear system using Volterra filters. Examples demonstrate these redundancies and the benefits of a tuned system that is aware of these redundancies.

A framework is laid allowing the comparison of various algorithms, defining a methodology of comparing prospective algorithms and measuring their effectiveness. Algorithms are presented for reduced complexity adaptive Volterra filters which contain a minimised set of parameters, and hence outperform simple truncated Volterra filters for a given filter order.

Finally, a generalization of the reduced complexity Volterra filters is made, allowing an adaptive algorithm to choose the best Volterra terms for a given task. This generalization includes Volterra filters as a subset.

Given this analysis of the subject, the contributions of this thesis are the proposed algorithms and derivations for reducing the parameterization of Volterra filters. Based on this reduction, it is believed that existing applications using Volterra filters can benefit from these results. Presented in Chapter 5, the term selection algorithms and method of comparison is the first contribution. The next contributions are the continuous exponent adaptation algorithms and derivations of Chapter 6, including the use of the infinite Taylor series representation of the modelling function for adaptation.

CHAPTER 2-VOLTERRA FILTERS

A Volterra filter is a particular type of mathematical construct that models many general non-linearities with memory. It provides an input-output relationship, where the input and output are either continuous functions (usually a time function), or a multi-valued vector (usually a time series) in the discrete case.

The Volterra filter is a polynomial filter. It generates its output as a weighted sum of products of its inputs. Like memoryless (single variable) polynomials, the Volterra filter contains higher order terms of its inputs. Unlike memoryless polynomials (but like polynomials of many variables), the Volterra filter also contains products between all its inputs.

Volterra filters are continuous functions with respect to their inputs, although at their limit, they can approximate discontinuities. With respect to time, Volterra filters can operate in continuous time or discrete time. These forms of the Volterra filter are introduced in the following sections.

2.1 CONTINUOUS TIME VOLTERRA FILTERS

The input-output relation of a continuous time Volterra filter is shown below in equation (2.1).

$$
\begin{aligned}
y(x(t)) = h_o & \\
& + \int_{\tau_1=-\infty}^{\infty} h_1(\tau_1)x(t-\tau_1)d\tau_1 \\
& + \int_{\tau_{2,1}=-\infty}^{\infty}\int_{\tau_{2,2}=-\infty}^{\infty} h_2(\tau_{2,1},\tau_{2,2})x(t-\tau_{2,1})x(t-\tau_{2,2})d\tau_{2,2}d\tau_{2,1} \\
& + \int_{\tau_{3,1}=-\infty}^{\infty}\int_{\tau_{3,2}=-\infty}^{\infty}\int_{\tau_{3,3}=-\infty}^{\infty} h_3(\tau_{3,1},\tau_{3,2},\tau_{3,3})x(t-\tau_{3,1})x(t-\tau_{3,2})x(t-\tau_{3,3})d\tau_{3,3}d\tau_{3,2}d\tau_{3,1} \\
& + \ldots
\end{aligned}
\qquad (2.1)
$$

The output y is a function of an input x, which itself is a function of time. The first term, h_o, is the zero order term, a constant. The second term in the Volterra filter is the familiar general linear (first order) filter, evaluated as a continuous time convolution. The function $h_1(\tau_1)$ is the convolution kernel of this first order term. The next term is a second order term. It contains all possible products of two input variables. The two-dimensional weighting function $h_2(\tau_{2,1}, \tau_{2,2})$ is known as the second order Volterra kernel. The filter contains terms for all orders. The nth order Volterra kernel is an n-dimensional weighting function.

Equation (2.1) contains redundant terms. For example, $x(t-\tau_{2,1})\,x(t-\tau_{2,2}) = x(t-\tau_{2,2})\,x(t-\tau_{2,1})$. The general Volterra filter can hence be rewritten with no loss of generality as equation (2.2) (note the limits of integration).

$$
\begin{aligned}
y(x(t)) = h_0 & \\
& + \int_{\tau_1=-\infty}^{\infty} h_1(\tau_1)x(t-\tau_1)d\tau_1 \\
& + \int_{\tau_{2,1}=-\infty}^{\infty} \int_{\tau_{2,2}=\tau_{2,1}}^{\infty} h_2(\tau_{2,1},\tau_{2,2})x(t-\tau_{2,1})x(t-\tau_{2,2})d\tau_{2,2}d\tau_{2,1} \\
& + \int_{\tau_{3,1}=-\infty}^{\infty} \int_{\tau_{3,2}=\tau_{3,1}}^{\infty} \int_{\tau_{3,3}=\tau_{3,2}}^{\infty} h_3(\tau_{3,1},\tau_{3,2},\tau_{3,3})x(t-\tau_{3,1})x(t-\tau_{3,2})x(t-\tau_{3,3})d\tau_{3,3}d\tau_{3,2}d\tau_{3,1} \\
& + \ldots
\end{aligned}
\tag{2.2}
$$

From this form, we can see that h_2 is a two dimensional function, but with coverage in only half the plane. Similarly, the other Volterra kernels have only partial coverage in their n-dimensions.

It can be seen that by letting all Volterra kernels be zero, except the first order Volterra kernels, we obtain a general linear filter. Hence the linear filter is obtained as a subset of the Volterra filter, and we can expect the Volterra filter to at least match the modelling performance of a linear filter in any application which is not otherwise constrained.

2.2 HYBRID VOLTERRA FILTERS

Although the Volterra filter is continuous with respect to its input, it does not need to have continuous Volterra kernels to retain this property. One use for discontinuous kernels is to mask certain time instances or time ranges. For example, truncation of the Volterra filter in time can be achieved by setting each Volterra kernel to zero when any index of the kernel is greater than the truncation time. In these cases, the Volterra filter does not

operate in true continuous time, although for certain time periods, it is continuous. Another specific use for discontinuous kernels is the development of discrete time Volterra filters.

2.3 DISCRETE TIME VOLTERRA FILTERS

The discrete time Volterra filter is derived in a similar manner as the discrete time linear filter. Consider the following Volterra kernels:

$$\begin{aligned} h_1(\tau_1) &= \sum_{n=-\infty}^{\infty} h_n \delta(\tau_1 - n\tau) \\ h_2(\tau_{2,1}, \tau_{2,2}) &= \sum_{m=-\infty}^{\infty} \sum_{n=m}^{\infty} h_{m,n} \delta(\tau_{2,1} - m\tau, \tau_{2,2} - n\tau) \\ &\cdots \end{aligned} \tag{2.3}$$

These are discrete time kernels sampled at τ time intervals. The Volterra filter now only depends on the vectors of discrete values h_n, $h_{m,n}$,... Substituting these values into equation (2.1) yields

$$\begin{aligned} y(k) &= h_0 \\ &+ \sum_{n=-\infty}^{\infty} h_{1,n} x(k-n) \\ &+ \sum_{m=-\infty}^{\infty} \sum_{n=m}^{\infty} h_{2,m,n} x(k-m)x(k-n) \\ &\cdots \end{aligned} \tag{2.4}$$

Equation (2.4) shows the input-output relation of the discrete time Volterra filter, where $x(k)$ and $y(k)$ are discrete time sequences, and $h_{1,n}$, $h_{2,m,n}$ ... are discrete time Volterra kernels.

There are several special cases of the discrete time Volterra filter. The first case arises when all Volterra kernels except the first order kernel are zero. In this case, the filter is linear. The second case arises when each member of all Volterra kernels are zero except when the τ indices are all zero. This corresponds to the memoryless Volterra filter, or simply a single variable polynomial. Another case occurs when each member of all Volterra kernels is zero except when m=n=..., when all indices are equal. In this case, there are no cross input terms, and the Volterra filter becomes the sum of memoryless (time shifted) polynomials.

The discrete time Volterra filter will be the focus of this thesis due to its relevancy to digital signal processing.

2.4 LIMITATIONS OF VOLTERRA FILTERS

The parameters of a Volterra filter are Volterra kernels. The primary limitation of discrete time Volterra filters is the number of parameters needed to characterize a system. The number of parameters increases greatly with small increases in order or memory. If a Volterra filter is truncated so it only contains terms with memory m or less, and of order p or less, the number of coefficients in all the Volterra kernels is given as *N[11]*:

$$N(p,m)=\binom{m+p}{m} \tag{2.5}$$

Where N(p,m) is the combinatorial function of m+p and m, defined as

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{2.6}$$

and n! is factorial n. A second order Volterra filter with memory of two can be expressed as:

$$\begin{aligned} y = {} & h_0 + h_{1,0}x[n] + h_{1,1}x[n-1] + h_{2,0,0}x[n]x[n] + \\ & h_{2,0,1}x[n]x[n-1] + h_{2,1,1}x[n-1]x[n-1] \end{aligned} \tag{2.7}$$

Equation (2.7) has six independent coefficients, as predicted by equation (2.5) i.e.

$$N(2,2) = \binom{4}{2} = 6 \tag{2.8}$$

The size of various filters is demonstrated in Table 1. It can be seen that for a relatively low filter order, the number of Volterra coefficients increases rapidly.

Table 1 - Number of coefficients for various amounts of memory and filter order

	2	3	4	5
	3	6	10	15
	4	10	20	35
	5	15	35	70
	6	21	56	126
	7	28	84	210

[redacted]	8	36	120	330
[redacted]	9	45	165	495
[redacted]	10	55	220	715
[redacted]	11	66	286	1001
[redacted]	12	78	364	1365
[redacted]	13	91	455	1820
[redacted]	14	105	560	2380
[redacted]	15	120	680	3060

This state of affairs is unacceptable for modelling higher order systems of even modest memory. Do we really need all these terms? What are they doing for us anyways? Chapter 4 answers these questions, but first we look at one of the most important tools available for non-linear modelling - adaptive filtering.

CHAPTER 3 - ADAPTIVE VOLTERRA FILTERS

The usefulness of Volterra filters becomes obvious when they are made into adaptive structures. By adapting Volterra filters, it is possible to create dynamic models of systems, provide real-time non-linear compensation and correction, and generate inverse systems on-the-fly. Since Volterra filters are very general systems, they will do well in a wide variety of applications given sufficient order and memory size [12].

Adaptation of Volterra filters is usually a simple extension of linear adaptive filtering. The typical method to extend linear adaptive filtering to Volterra filters is to represent the Volterra filter in a vector representation [11]. In this representation, we write the output as

$$y = \vec{X}^T \vec{H} \tag{3.1}$$

where **X** contains the individual terms of the Volterra filter arranged in a vector (in any order), and **H** contains the coefficients of these terms (so the order corresponds to the order of the terms in **X**). We have formulated the input output relation as a vector multiplication, and the adaptation of the coefficients as determining the optimum coefficients, **H**, of a linear equation. Under this formulation, we can use common linear adaptive algorithms to adapt the elements of **H**.

As an example of this formulation, consider the following system:

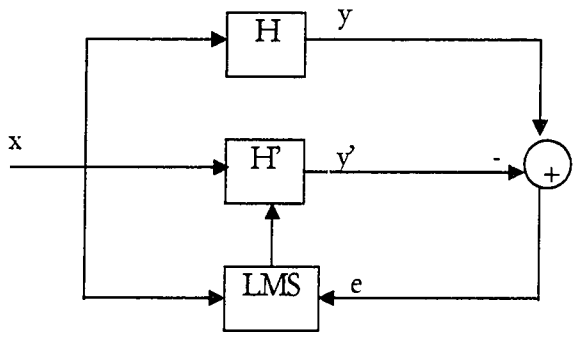


Figure 3.1 Linear system used in formulating the adaptive Volterra filter

In this figure, we have a linear system **H**, and the input of this system is a sequence of samples. **H'** is an FIR filter and is attempting to model **H** through the LMS algorithm by reducing the error *e*. So far, this is the standard set-up for adaptively modelling a linear system.

If instead of $x[n]$, we introduce $x'[n']$, which is the sequence $[x[n], x^2[n], x^3[n], x^4[n] \ldots, x[n-1]x[n], \ldots]$ i.e. a sequence containing all the Volterra terms, where x is the original sequence. The sequence $x'[n']$ contains a multiple of the number of terms of $x[n]$, with the multiple being the number of nonlinear terms in $x'[n']$. If we now pass this sequence at the higher rate through the linear system **H** and **H'**, the output is nonlinear with respect to the sequence $x[n]$ (though linear with respect to $x'[n']$). This linearity with respect to $x'[n']$ allows us to use linear adaptation techniques like LMS. However, LMS which normally takes the input vector $x[n]$, now takes the input vector $x'[n']$ (nonlinear combination of the elements of $x[n]$). So for the formulation of adaptive algorithms, we can consider the nonlinear systems as being linear with nonlinear combinations of inputs.

If we adapt the coefficients of H using this input, and generate new inputs simply by obtaining a new value for $x'[n]$ instead of shifting the input and adding a new element, we can use any pre-existing algorithms, as long as they do not care about the way the input is generated (i.e. don't care if the next input vector is a time shifted version of the previous input). All results of adaptive non-linear systems may be derived from the linear system analysis simply by assuming an input vector created in the manner described. This includes maximum step size, adaptation rate, misadjustment, existence of a single minimum, among others. For a derivation of these properties, see [11].

The remainder of this chapter describes the LMS and RLS algorithms as they apply to nonlinear systems. The derivation of these algorithms as well as a discussion of their properties can be found in [11]. This thesis will not go into these details, but will give an overview of the mechanics of the algorithms.

3.2 LMS ADAPTIVE VOLTERRA FILTERS

The LMS technique is a gradient search algorithm whereby the coefficients of a linear equation are updated by a simple update equation, where the update is proportional to the estimation error and also the input tap value. The input tap values are the elements of the input vector **X**. In a Volterra filter, the elements of the input vector **X** are not simple shifted inputs, but also include products of inputs (Volterra terms). The basic setup using LMS for modelling is shown in Figure 3.1. Hence the update of each coefficient is accomplished using the well known LMS update equation modified so the input tap value is now a Volterra term output. The update equation is shown here:

$$h_i[n] = h_i[n-1] + \mu_i e[n] f_i(\vec{x}) \tag{3.2}$$

where $f_i(x)$ is the i^{th} Volterra term, μ_i is the step size used to update this terms, and $e[n]$ is the overall error. This change in notation reflects the idea that we are updating a unified vector of coefficients rather than a set of coefficients organized into separate kernels for each term order, as shown in (2.3). When adapting the coefficient h_i of the $x[n-1]x[n-2]$ term, the update equation becomes

$$h_i[n] = h_i[n-1] + \mu_i e[n] x[n-1] x[n-2] \tag{3.3}$$

where all the values shown are scalars. The value of μ is now dependent on the term we are updating, since the input terms are no longer identically distributed.

3.2.2 PERFORMANCE PROBLEMS OF LMS ADAPTIVE VOLTERRA FILTERS

The LMS algorithm's adaptation rate is dependent on the correlation of the inputs [9], or equivalently, the correlation between inputs to the filter taps. In the case of a Volterra filter, the terms of the filter (taps) are rarely orthogonal, since they made up of products of inputs, and the same input will be present in many terms. Because of this, the LMS adaptation rate is typically slow for Volterra filters. Adaptive step size algorithms improve the performance somewhat, but RLS is usually used when adaptation rate is important.

3.3 RLS ADAPTIVE VOLTERRA FILTERS

The RLS algorithm attempts to minimize the cost function:

$$J = \sum_{k=1}^{M} \lambda^{M-k} (d(k) - \vec{\mathbf{H}}^T \vec{\mathbf{X}})^2 \tag{3.4}$$

where λ is a forgetting factor resulting in an exponentially weighted mean square error. The updates for the RLS algorithm can found in [11] and are reproduced here:

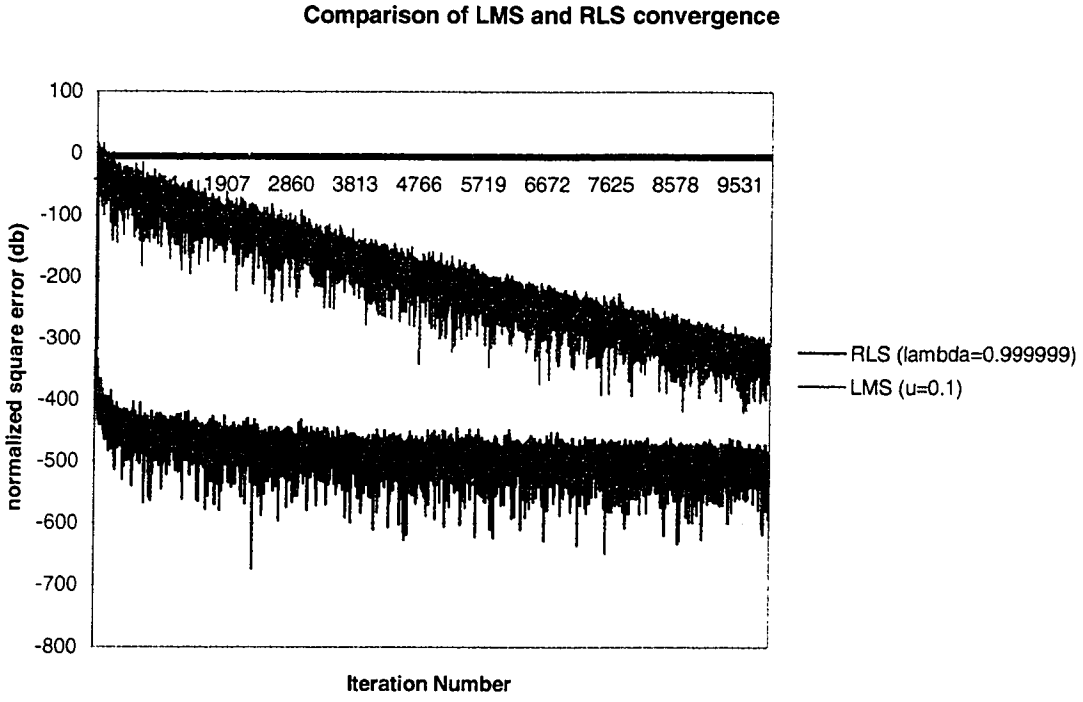
$$\begin{aligned}
&\textit{Initialization}: \\
&\mathbf{C}^{-1}(0) = \frac{1}{\partial}\mathbf{I} \\
&\mathbf{P}(0) = 0 \\
&\vec{\mathbf{H}}(0) = 0 \\
&\textit{Iterations}: \\
&e(n) = d(n) - \mathbf{X}^T(n)\vec{\mathbf{H}}(n-1) \\
&\mathbf{k}(n) = \frac{\mathbf{C}^{-1}(n-1)\mathbf{X}(n)}{\lambda + \mathbf{X}^T(n)\mathbf{C}^{-1}(n-1)\mathbf{X}(n)} \\
&\mathbf{C}^{-1}(n) = \frac{1}{\lambda}\mathbf{C}^{-1}(n-1) - \frac{1}{\lambda}\mathbf{k}(n)\mathbf{X}^T(n)\mathbf{C}^{-1}(n-1) \\
&\vec{\mathbf{H}}(n) = \vec{\mathbf{H}}(n-1) + \mathbf{k}(n)e(n)
\end{aligned} \tag{3.5}$$

where $\mathbf{C}^{-1}$ is the inverse of the exponentially weighted least-squares autocorrelation matrix of the input vector, $\mathbf{P}$ is the exponentially weighted least-squares cross-correlation vector of the input and the desired response, $\mathbf{H}$ is the vector of coefficients, $\mathbf{X}$ is the input vector (containing the output of individual Volterra terms), and $\mathbf{k}$ is the gain vector. The equations for RLS are identical to the linear RLS algorithm, with the difference being the composition of the input vector and weighting vector, as previously explained. The RLS algorithm typically outperforms the LMS algorithm and is the preferred method of updating Volterra filters. This is a direct result of the decorrelation properties of RLS algorithms and its subsequent superior performance in cases of correlated input.

3.4 COMPARATIVE EXAMPLES

Figure 3.2 shows the modelling error of LMS and RLS with respect to time when modelling a third order system with a memory of three samples (20 terms). The coefficients of the system to be modelled were chosen randomly and are uniformly distributed between 1 and -1. The modelling system is also a third order system with a memory of three samples (thus perfect modelling is possible). The modelling filter starts with all zero coefficients, and the step size and forgetting factor of LMS and RLS respectively are chosen to provide a fast but stable convergence, μ=0.1 for LMS, λ =0.999999 for RLS. The values of μ for each term were made identical to simplify the implementation. The input is uniformly distributed between -1 and 1, with no correlation between input samples. The error is displayed as $10*\log(e^2/v)$, where v is that variance of the input.

Figure 3.2 Convergence of LMS and RLS



From this plot, we can see that the convergence of RLS is indeed very fast, whereas the convergence of LMS is much slower. Again, despite the uncorrelated samples of x(n), the actual inputs to the adaptive algorithms are inherently correlated. The superior performance of RLS in such cases is well established.

CHAPTER 4-REDUNDANCIES IN VOLTERRA FILTERS

The generality of the Volterra filter comes at a high price of complexity: the filter is usually not very compact and contains redundancies. Redundancies occur because of memoryless effects, such as the correlation between terms operating upon the same data sample, and time effects, such as the correlation between input samples. In fact, the non-orthogonality of the terms that make up the Volterra filter allows the creation of other Volterra filters which approximate the original filter very closely, but which use none or only some of the original terms.

This chapter describes some useful properties of the Volterra filter and how filters using different Volterra terms can act in similar ways. The understanding of these redundancies provides a justification for research in this area. Knowledge of how to switch between approximating Volterra kernels, and hence how to choose the best filter, is a step towards adaptively choosing the most compact filter.

The problem is looked at from two aspects. The first looks at redundancies in the filter order (redundancies due to nonlinear interaction between Volterra terms). The memoryless Volterra filter is analysed, and properties are deduced. The second part looks at redundancies that occur due to the non-white autocorrelation of the input (redundancies due to a time relationship between input samples).

4.1 REDUNDANCIES IN FILTER ORDER

To demonstrate the redundancies due to the non-linear memoryless portions of the Volterra filter, an example utilizing a memoryless Volterra filter (a polynomial of one variable) is presented. This simplification is made to accentuate the types of redundancy we can expect to find in the full Volterra filter, without complicating the analysis with memory effects, such as cross terms, input correlation, etc.

In the ideal representation, all terms in a filter would be orthogonal to all other terms. This would allow the addition of more terms without recompiling the coefficients of existing terms. This is not the case with the Volterra filter. To move from an order n filter to an order $n+1$ filter requires the recompilation of the previous n coefficients (Volterra kernels). An examination of the best approximation of one Volterra filter with another Volterra filter is very helpful in understanding the redundancies between terms.

For this analysis, we use the minimum mean squared error criteria for determining the best approximating Volterra filter. We assume a process generating an input sequence x, with a probability distribution $\theta(x)$. We begin by approximating one term of the memoryless polynomial (ax^n) by another term (bx^m).

The mean squared approximation error for these two terms is given in equation (4.1). This is a measure of how well one term can be approximated by another, i.e. of the redundancies between these two terms ax^n and bx^n.

$$J = \int_{-\infty}^{\infty} (ax^n - bx^m)^2 \theta(x) dx \tag{4.1}$$

This equation can be written as equation (4.2).

$$J = a^2 \int_{-\infty}^{\infty} x^{2n} \theta(x) dx - 2ab \int_{-\infty}^{\infty} x^{m+n} \theta(x) dx + b^2 \int_{-\infty}^{\infty} x^{2m} \theta(x) dx \tag{4.2}$$

Differentiating equation (4.2) with respect to b and minimizing yields equation (4.3).

$$0 = a \int_{-\infty}^{\infty} x^{m+n} \theta(x) dx + b \int_{-\infty}^{\infty} x^{2m} \theta(x) dx \tag{4.3}$$

Finally, solving for b yields equation (4.4), which is the optimum approximation for a given m and n.

$$b = \frac{a \int_{-\infty}^{\infty} x^{m+n} \theta(x) dx}{\int_{-\infty}^{\infty} x^{2m} \theta(x) dx} \tag{4.4}$$

A quick analysis of equation (4.4) reveals the following: if the distribution of x is symmetric about 0 ($\theta(x)$ is an even function) and $m+n$ is odd (m odd-n even or m even-n odd), then $b=0$. This means that x^m cannot approximate x^n if $m+n$ is odd. *Hence for symmetric input distributions, all even terms are orthogonal to all odd terms.* We also note that the distribution of the input is important in this approximation.

Equation (4.4) can also be written as shown in (4.5)

$$b = a\frac{\mu'_{m+n}}{\mu'_{2m}} \tag{4.5}$$

where μ'_n is the n^{th} moment (taken around 0) of $\theta(x)$.

Given equation (4.4), we determine the minimum squared error of this approximation by substituting b back into equation (4.1), yielding equation (4.6).

$$J_{\min} = a^2 \int_{-\infty}^{\infty} x^{2n}\theta(x)dx - 2a^2 \frac{\int_{-\infty}^{\infty} x^{2(m+n)}\theta^2(x)dx}{\int_{-\infty}^{\infty} x^{2m}\theta(x)dx} + \frac{a\int_{-\infty}^{\infty} x^{2(m+n)}\theta^2(x)dx}{\int_{-\infty}^{\infty} x^{2m}\theta(x)dx} \tag{4.6}$$

As an example, consider approximating x^n with bx^{n+2}, x having a uniform distribution in the range [-1,1]. Figure 4.1 shows J_{min} for n ranging from 0 to 100. This ensures even terms are used to approximate even terms and odd terms are used to approximate odd terms. Allowing for general approximation of x^n with x^m is futile, since we would be approximating even terms with an odd terms (and vice-versa), and these terms would be orthogonal to each other given that the input distribution is symmetric about 0.

Figure 4.1 Squared error in approximating x^n with bx^{n+2}, for various n

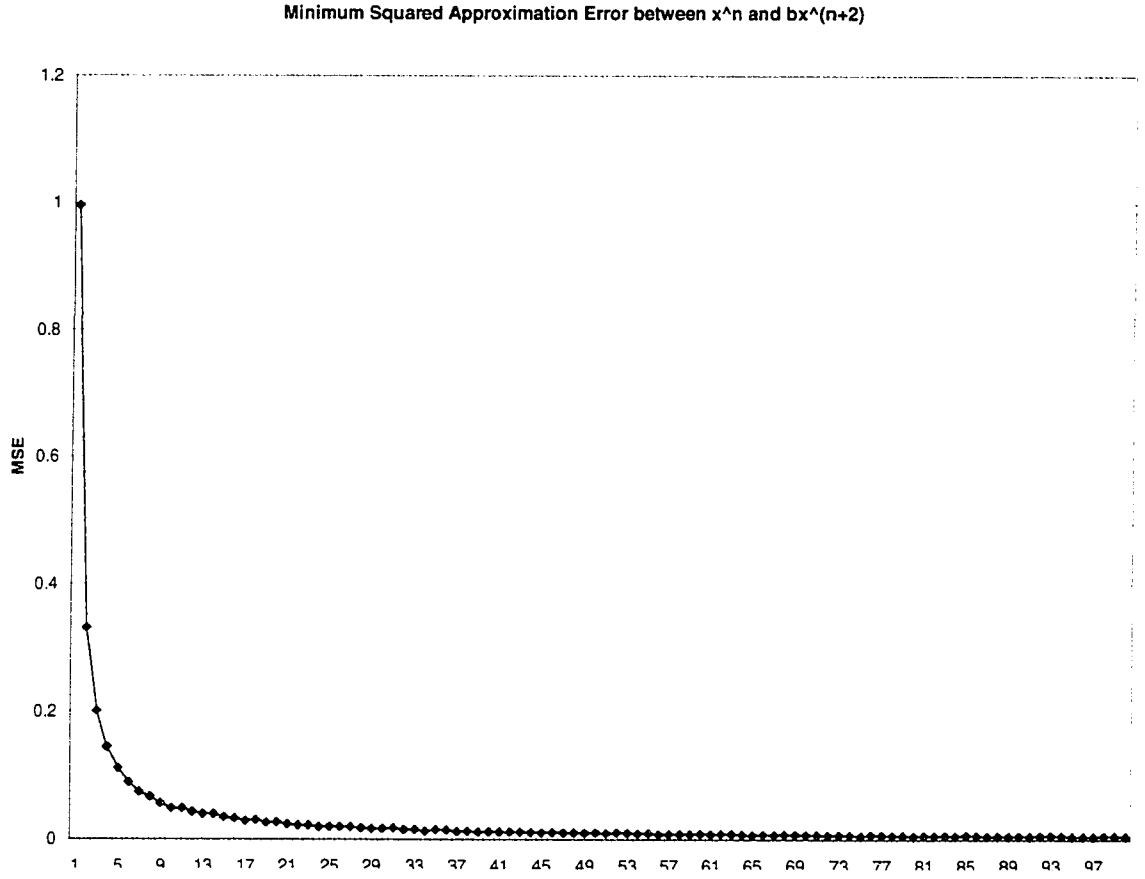


Figure 4.1 illustrates the effect of approximating one term in the polynomial with another. It can be seen that approximating low order terms with the next eligible higher order term is not very accurate, in the sense that it will produce a large error. However, higher order terms can be substituted for other higher order terms easily with very little approximation error.

If we had a filter with the input-output relation x^2+x^{50}, we could easily substitute this filter with x^2+ax^{52}, (a being optimum) without introducing much error, however the substitution bx^4+x^{50}, would result in a large error. Furthermore, to minimise the number of

terms needed to model a memoryless function, it would generally be better to choose those terms more densely at lower orders and more sparsely at higher orders.

4.2 DERIVATION OF THE ORTHOGONAL DISTRIBUTION

An interesting question arises when considering orthogonality of the terms of the memoryless Volterra filter: What is the distribution of x for which all terms in the polynomial are orthogonal to each other? We begin by defining the orthogonality condition between two terms x^n and x^m in equation (4.7).

$$0 = \int_{-\infty}^{\infty} x^{m+n}\theta(x)dx \tag{4.7}$$

Since $\theta(x)$ is non-negative, x^{m+n} in equation (4.7) must contain at least some negative values, and hence cannot be satisfied for even values of $m+n$. Therefore, there is no distribution that makes all terms in the memoryless polynomial orthogonal to each other. In fact, there is no distribution that makes any even term orthogonal to any other even term, or odd orthogonal to odd (except the deterministic distribution- $\delta(x)$).

4.3 REDUNDANCIES IN SIGNAL INPUT

Another effect that introduces redundancies into the Volterra filter is time domain redundancies. These redundancies arise due to a statistical relationship in time between samples. In the linear case, this implies an autocorrelation function.

We will begin by looking at the linear FIR filter and observing how the terms of a filter can be exchanged for other terms in the presence of a correlated input. Consider two terms of a linear FIR filter: $ax(n-k)$ and $bx(n-l)$, and an autocorrelation function $r_x(m)$. Again using the mean squared error as the criteria, we can write the degree to which one term approximates the other in equation (4.8).

$$\begin{aligned} J &= E[(ax(n-k) - bx(n-l))^2] \\ &= a^2 E[x^2(n-k)] - 2abE[x(n-k)x(n-l)] + b^2 E[x^2(n-l)] \end{aligned} \tag{4.8}$$

By minimising J with respect to b, we obtain equation (4.9).

$$0 = -2aE[x(n-k)x(n-l)] + 2bE[x^2(n-l)] \tag{4.9}$$

Solving for b, we obtain equation (4.10).

$$\begin{aligned} b &= a\frac{E[x(n-k)x(n-l)]}{E[x^2(n-l)]} \\ b &= a\frac{r_x(k-l)}{\sigma_x} \end{aligned} \tag{4.10}$$

The first order terms depend on the autocorrelation function and the variance of the input. They do not depend on the distribution of the input, other than the variance of that distribution.

Both forms of inter-term correlation (due to input autocorrelation and term multiplication correlation) combine to provide much more redundancy in Volterra filters than would be found in linear filters. Given these redundancies, it seems likely that a more compact method of representing Volterra systems can be found given an intelligent choice

of Volterra terms or a better representation. The following chapters deal with algorithms for choosing Volterra terms and with a better representation of Volterra filters.

CHAPTER 5-EXPLOITING REDUNDACIES AND REDUCING FILTER SIZE

From the previous chapter, it can be seen that it is possible for two Volterra filters to approximate a given input-output relationship within the same range of error, even while being completely disjoint in their terms. The goal of this chapter is to determine the *subset* of Volterra terms that when combined in a Volterra filter yield a near- optimum filter of a given size. It is clear that merely truncating a Volterra filter and using only low order terms is an arbitrary way of reducing the filter complexity, but it is not necessarily optimal.

One way of looking at the problem is considering terms that are almost orthogonal to each other (approximate each other poorly). For example, if we were choosing memoryless terms, we may choose a low order and a high order term, but we would not choose two high order terms, since they would have too much overlap. The advantage to such an approach is that the selected terms provide better coverage of the vector space occupied by the approximating and approximated functions. Another added bonus is that adaptive algorithms such as LMS will adapt faster using nearly orthogonal basis functions.

While this is generally a good strategy, it does not always work. For example, if we were given the function $x^{52}+x^{54}$, the optimal basis functions would obviously be x^{52} and x^{54}, two terms that approximate each other very well. Hence optimal truncation requires

information about the modelling function, and in the absence of a priori information, an adaptive algorithm will be needed to learn and collect this information in some form.

5.1 SIMPLE METHODS OF FINDING OPTIMAL TERMS

The problem is to choose n terms from a larger set of m terms that most closely model a desired system. The reason this is done is to reduce the complexity from m terms to n terms, or to reduce the modeling error by choosing n good terms as opposed to n arbitrary terms. *A successful algorithm will achieve a reduction in modeling error when compared to an equal sized filter that was arbitrarily truncated, while not incurring a significant increase in computational complexity.* Alternately, the algorithm will obtain a similar modeling error using a smaller number of terms than the arbitrarily truncated filter. The typical method of truncating Volterra filters is to truncate the memory of the system from 0 to k samples, and truncating the order of the system from 1 to i. The methods given in this chapter provide better truncation through simple methods. Some of these methods are not practical real-time methods, in that they take excessively long periods of time to run, but they are given here as starting points and reference algorithms.

The methods discussed here fall into two categories: the brute force method and the switched multiple set method that are proposed as an obvious first attempt. The brute force method gives optimal results for stationary systems and signals; however it is not practical as it uses an infeasible amount of processing time. The switched set methods are approximations to the brute force method, attempting to reduce the complexity of the brute force method; however, they no longer guarantee optimality.

5.2 BRUTE FORCE METHOD

The brute force method of choosing the best n terms from a set of m terms consists of trying all possible sets of n terms chosen from m terms. There are $C(m,n)$ unique sets of n terms to try (where $C(m,n)$ is the combinatorial function $m!/n!(m-n)!$). Hence, assuming LMS was used for modeling, the arbitrarily truncated system will have a complexity $O(n)$ per new sample. The brute force method (also using LMS) would have a complexity $O(\ C(m,n) * n)$. As m gets larger, $C(m,n)$ gets very large. For example, suppose we were choosing the best three terms chosen from all third order terms with memory three or less. There are 20 terms of third order with memory of three or less; hence the brute force method takes O(C(20,3) * 3) = O(1140 *3) as opposed to O(3). For this reason, the brute force method is useful only for comparison to other non-optimal methods. It is presented here for generating a reference.

The brute force method will generate all $C(m,n)$ sets of filters, and determine the final modeling error for each prospective filter. The best n-term filter is the optimum; however sub-optimum filters can *approach* the best performance. This set of errors is useful for comparison against other algorithms. One measure of accuracy of an algorithm would be how close the final error is to the error of the best set found using the brute force method. However, this comparison is of limited use since it provides no reference: an algorithm might provide an error that is only 1% larger than the best error, but all other choices of n terms may have produced an even lower error. A better method of comparison would be to determine what percentile that error lies in when compared to all the $C(m,n)$ sets tried by the brute force algorithm. For example, if an algorithm produces a filter whose final modelling

error which lies in the 50th percentile, it is considered a bad algorithm, since a guessing algorithm (choosing a arbitrary n-length set) can be expected (on average) to perform at the 50th percentile. An algorithm that achieves a final ranking better than the 50th percentile would be considered a good algorithm, with quality improving as the percentile increases.

As the algorithm chooses new "best" sets of terms, we reference the brute force results to find how well the "best" set is doing with respect to the other sets. By doing this, we can build a distribution showing how well the algorithm picks its "best" set. From this distribution, we can find two measures of how well the algorithm performs. For a good algorithm, the mean of the distribution should be better than the 50th percentile. The better algorithm will have a lower mean. Additionally, the variance of the distribution is also important, as it indicates how reproducible the result is. A low variance indicates we will meet our mean performance more often, while a high variance is an indicator of an unreliable algorithm.

5.3 SWITCHED MULTIPLE SET APPROACH: ALGORITHM 1, SINGLE TERM SWITCHING

This family of proposed algorithms attempts to find the best set of n terms chosen from a larger set of m terms of a Volterra filter by looking at a small subset of filters which differ from the current filter by *one term*. These algorithms concentrate on determining which one of the n terms is the worst performing term of the filter, and hence the term to drop. The replacement of this term is somewhat arbitrary, usually just randomly chosen.

The algorithm can be stated as follows:

1) Choose a starting filter

2) Adapt the filter to a stable approximation

3) Determine the worst performing term

4) Discard this term and replace it with another term

5) Repeat step 2

Each of these steps allows considerable flexibility in their implementation and may be replaced, often independently of the other steps. The following sections look are each of these steps.

5.3.1.1 CHOOSE A STARTING FILTER

The premise of the algorithm is that we will achieve a near optimal filter after a limited number of iterations; hence it may not be critical to choose a good starting filter.

5.3.1.2 ADAPT THE FILTER TO A STABLE APPROXIMATION

The preferred method to adapt Volterra terms is the RLS algorithm, since for small filters, it has a low complexity and adapts much quicker than LMS. The advantage of LMS is that it is simpler to implement. Overall, taking into account the complexity of the algorithm, size of test systems and the number of iterations needed to adapt, RLS provides a shorter simulation run time.

5.3.1.3 DETERMINE THE WORST PERFORMING TERM

The problem with determining which term to replace is that unless all the terms are orthogonal to each other, no term's worth can be measured independently of the other terms. Hence the problem can be restated as: what is the best set of $n-1$ terms that *work together* to model the system? Working with this problem statement, and starting with one filter of n terms, we can generate a set of n filters each with only $n-1$ terms taken from the starting filter, each modelling the desired function. The filter with the smallest mean squared error corresponds to the best set of $n-1$ terms. The term to be discarded is the term which is not part of this set of $n-1$ terms.

The cost of this algorithm is high: for each term in the filter, we need to adapt and evaluate another filter, i.e. for n terms, we need to evaluate n filters. To reduce this cost, we assume that the value of the optimum coefficients of the best $n-1$ sized filter will not change much by the addition of the extra n'th worst term, since that worst term contributes only slightly towards modeling the system. So when determining the error of each set of n filters, we can approximate each filter by the full n-length filter minus the respective term under evaluation. Hence only a set of n errors must be tallied by evaluating the filter and subtracting the contribution of each successive term, as opposed to evaluating and adapting n filters.

As an example, suppose the current filter contained the terms $x[n]$, $x[n]x[n-1]$, and $x[n]x[n-2]$. Rather than computing three respective errors with three 2 term filters (i.e. the filters with terms 1 and 2, 1 and 3, and 2 and 3), we determine the error of the full three term

filter, then compute the errors attributed to each term by subtracting out that term's contribution, i.e. $e + h_0 x[n]$, $e + h_1 x[n]x[n-1]$ and $e + h_2 x[n]x[n-2]$.

5.3.1.4 DISCARD THIS TERM AND REPLACE IT WITH ANOTHER TERM

The following methods have been investigated with respect to choosing a replacement term:

1) Random replacement: arbitrarily choose a term from the remaining $m+n$ terms

2) Replacement with the term judged to be least like the discarded term

3) Replacement with the term judged to be most like the terms left in the filter

4) Replacement with the term judged to be most different than the terms left in the filter

Random replacement offers the following advantages over the other methods:

1) Simple to implement

2) Avoids loops where term a is replaced by b, only to be replaced again by a

3) Provides a uniform coverage of the space of sets being investigated

Replacement with the term judged to be least like the discarded term works on the premise that if a term is bad, a good term will be dissimilar to it. The criteria for determining how close two terms are to each other is their correlation, defined as $E[x*y] / E[x]*E[y]$, where $E[\ ]$ is the expected value, x and y are the terms we are comparing. The problem with

implementing this method is that for each of the (m-n) prospective term, we must obtain the three statistics needed in determining the measure.

Replacement with the term judged to be most like the terms left in the filter works on the premise that more of a good thing is a good thing. The same criterion is used for judging the closeness of a term with the remaining terms.

Replacement with the term judged to be most different than the terms left in the filter works on the premise that if we are modeling a system with a set of similar terms, the most benefit we can get will be with a term that models the system in a different way.

5.3.1.5 REPEAT AT STEP 2

The algorithm will continue until we decide to stop it. A good time to stop could be when we reach a reasonably low error or the error no longer changes appreciably. Alternately, we can detect the presence of "loops", which occur when the same term is constantly swapped out of the filter. In this case, a certain amount of randomness may be added, such as dropping one of the "good" terms, or choosing to stop swapping altogether. In addition, it is possible to keep the past few filters and revert to them if they outperform the current filter.

5.3.2 JUSTIFICATION OF THE ALGORITHM

If we choose a filter with randomly selected terms, we can expect to choose a filter that on the average ranks in the top 50th percentile half the time. If we had some information about what terms do well, we can choose a filter that ranks in the top 50th percentile more often. This algorithm attempts to extract that information by looking at

each individual term and attempts to guess how well the filter would do without that term. If the term is a good term, we want to use it to model our system. If it is a bad term, we want to use a better term.

5.3.3 PROBLEMS WITH THE PROPOSED SINGLE TERM SWITCHING ALGORITHM

The main problem with this algorithm is the possibility of entering a loop where the same terms are continuously monitored, excluding other terms that may model the system better. For example, if we reach a collection of sets around the 90^{th} percentile (top 10 percentile) and never leave this collection, we will never achieve a higher percentile ranking. We are very unlikely to be caught in such a loop that is in a low (worse) percentile, but as we reach higher percentile sets, the likelihood of being trapped in a loop increases. The algorithm is more likely to break out of low percentile loops by dropping bad terms, but is likely to stay in high percentile loops by dropping reasonably good terms and replacing them with equally reasonable terms.

Another problem with this algorithm is that it only statistically reduces the probability of choosing a bad set, it does not guarantee achieving the optimum set, it only increases the chances of achieving a reasonable set.

5.3.4 METHODS OF VALIDATING THE ALGORITHM

A good algorithm will consistently choose a set of terms that perform close to the optimum filter. To judge how close a filter is to the optimum, all possible filters are evaluated for their final error (the average error they achieve after being given enough time to fully adapt to the system being modeled). These filters are then sorted from best to worst

with respect to the final error, and this sorted list is stored. The simulation is then run, and at each step of the simulation, the current filter is looked up from the list and its percentile rank is determined. A good algorithm will achieve a consistently high percentile ranking with respect to its ability to reduce the modeling error (high average percentile) and a low variability about this mark (low variance). At the very least, the algorithm should perform better than the 50th percentile. In addition to this ranking method, the error as the algorithm progresses through more steps should decrease until it reaches a steady level. Due to the nature of the algorithm, a bad set may appear momentarily, resulting in undesirable spikes in the error.

The following simulations present their results in the form of a histogram. The histogram shows how often an experiment resulted in the selection of a particular set of three terms. Each experiment is repeated with different (random) starting points and a random input. Each experiment run results in the increase of one bin of the histogram, corresponding to the final filter achieved by that experiment run. Since the horizontal axis is sorted according to the previously determined best filter, it is desirable to see a large number of experiments achieving filters in this region. Hence a histogram that is skewed to the right represents a good algorithm.

5.4 SIMULATIONS OF THE ALGORITHM

5.4.1.1 SIMULATION 1

The worst term is decided by keeping track of the error obtained by the full filter minus the contribution of that term (i.e. $e_n=d-(y-c_n)$, where d is the desired signal, y is the filter's

estimate, and c_n is the contribution of term n to the output y). The worst term is dropped from the model and is replaced by a random term.

RLS is used for adaptation, with λ=0.9999, adaptation between term switches is given 200 samples, average error is determined as $E = 0.99*E + 0.01*e*e$, where E is the average error, e is the instantaneous error. Input is uncorrelated, uniformly distributed noise ranging from −1 to 1.

The system to be modeled is the sum of the following terms:

Table 2 - Terms and coefficients of the system to be modelled

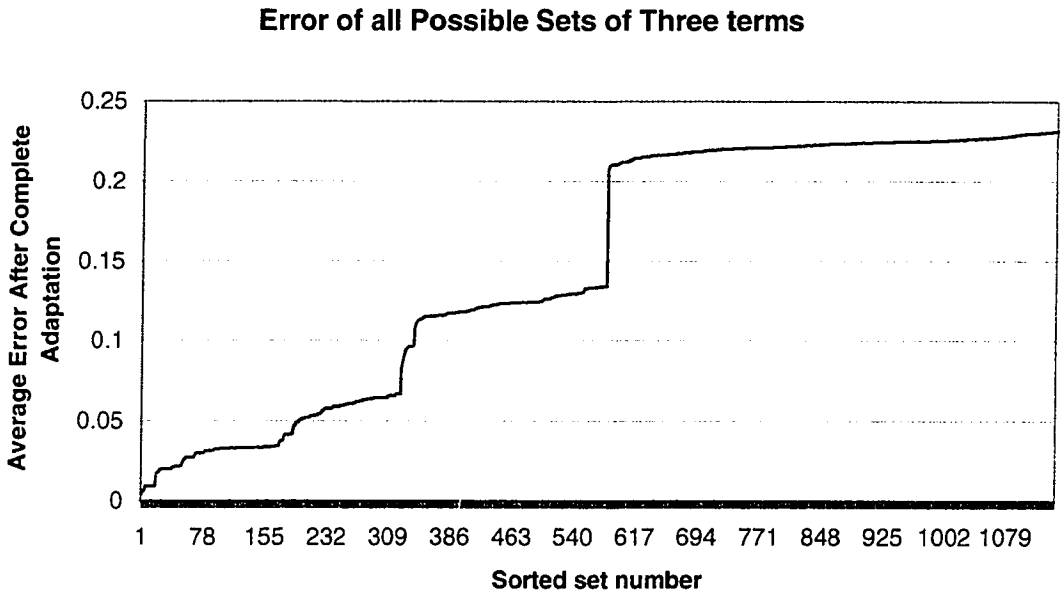
Term	*Coefficient*
x[n]x[n-1]	0.2
x[n]x[n-2]x[n-2]	0.3
x[n-1]x[n-2]x[n-2]	0.7
x[n-2]	0.8

This system contains first order, second order and third order terms, but otherwise is just an arbitrarily selected system.

This system is modelled by a three term Volterra filter. From the brute force algorithm, the performance of all possible three-term filters of order three or less with memory of three samples is shown below, sorted by error. This plot shows the modelling

error of the best performing filter on the left at position 1 and the modelling error of the worst performing filter on the right at position 1140. All other filters are shown in between.

Figure 5.1 Each individual three term filter is plotted vs. the final error for that filter. X-axis is sorted from best filter to worst filter.

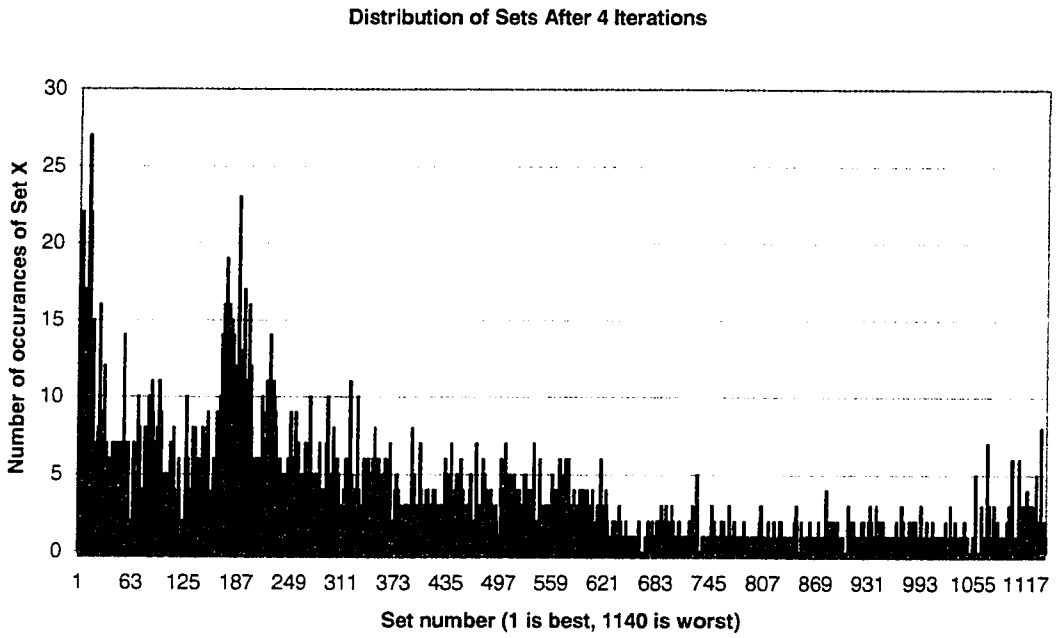


Notes: Set 1 is the best set, set 1140 is the worst set. About half of the sets perform very poorly (error >0.2), the other half perform twice as well, while the best set performs with an average error over 10 times smaller than the worst.

To obtain a clear picture of the performance of this algorithm, we run this simulation many times with different (random) starting sets. After n iterations, we graph the percentile position of the current filter as part of a histogram graph that shows the number of times each set from 1 to 1140 was chosen after iteration n. We should see a progression from a uniformly distributed histogram after 0 iterations to a distribution skewing towards the 0^{th} (best) percentile as more iterations are performed.

The histogram below shows the number of times each particular filter was chosen by the algorithm, out of 3000 runs. Filter number 1 is the filter that has previously been determined to be the best filter by the brute force algorithm. Filter 1140 is the worst filter.

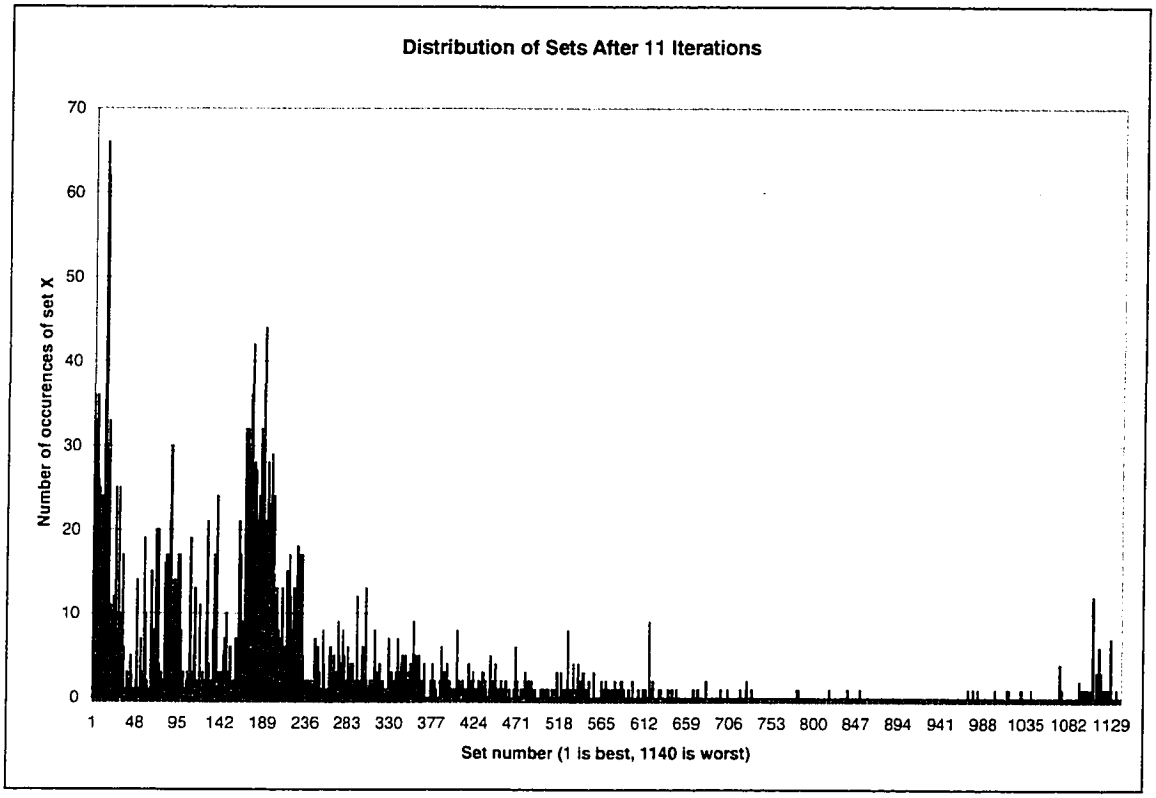
Figure 5.2 Histogram showing the distribution of chosen filters after four iterations



From Figure 5.2, we can see that the best filter was chosen in 22 runs out of 3000, while set # 290 showed up 10 times. More importantly, the top 50 sets (best 5%) showed up 300 times (almost 1/3 of the time), and the sets in the best 50 percentile were chosen over 2300 times (better than ¾ of the time).

After only four steps, the likelihood of choosing a good filter has increased significantly. However there is a cluster of sets between # 166 and # 221 that are sub optimal but still attract sets.

Figure 5.3 Histogram showing the distribution of chosen filters after eleven iterations



After 11 iterations, we can see a strong clustering around the best set (# 1) which is made up of the terms x[n-2], x[n]x[n-2]x[n-2] and x[n-1]x[n-2]x[n-2]. However, the clustering around the sub-optimal set # 191 is still present. There is also a small cluster around # 1110 which is made up of terms x[n]x[n], x[n]x[n-2] and x[n-1]x[n-2], with the probability of ending up at set # 1110 being greater after 11 steps than it was after 0 steps.

5.4.1.2 DISCUSSION

The algorithm seems promising as can be seen from the graphs. After only a few iterations, we end up in the best 15 percentile 90% of the time. The clustering around set # 191 occurs because all the filters in this region contain the term $x^3[n-2]$, which is not in the filter itself but is the term that best models x[n-2] (as shown in section 4.1). The algorithm

will most often keep this term, since x[n-2] is the dominant term in the desired filter, so we will stay within the set of filters that contain this term, always switching out the two other terms. In addition to this problem, since we subtract the contribution of each term to determine its worth, we are assuming that the one term is orthogonal to the rest of the terms. This is of course not the case, and the approximation becomes worse for lower orders, since the lower orders have better modelling ability by themselves. Hence there is a bias towards lower orders, and this algorithm will use lower order terms more often than higher order terms. Simulations 3 and 4 use an algorithm that overcomes this limitation, but at a higher computational cost.

5.4.1.3 SIMULATION 2

This simulation runs the same algorithm as in simulation 1, but modelling a different system. The modelling system is still a 3 term filter chosen from the set of terms with memory of three samples or less, and order three or less. The modelled function contains the three terms shown in the table below. Simulation parameters are given by:

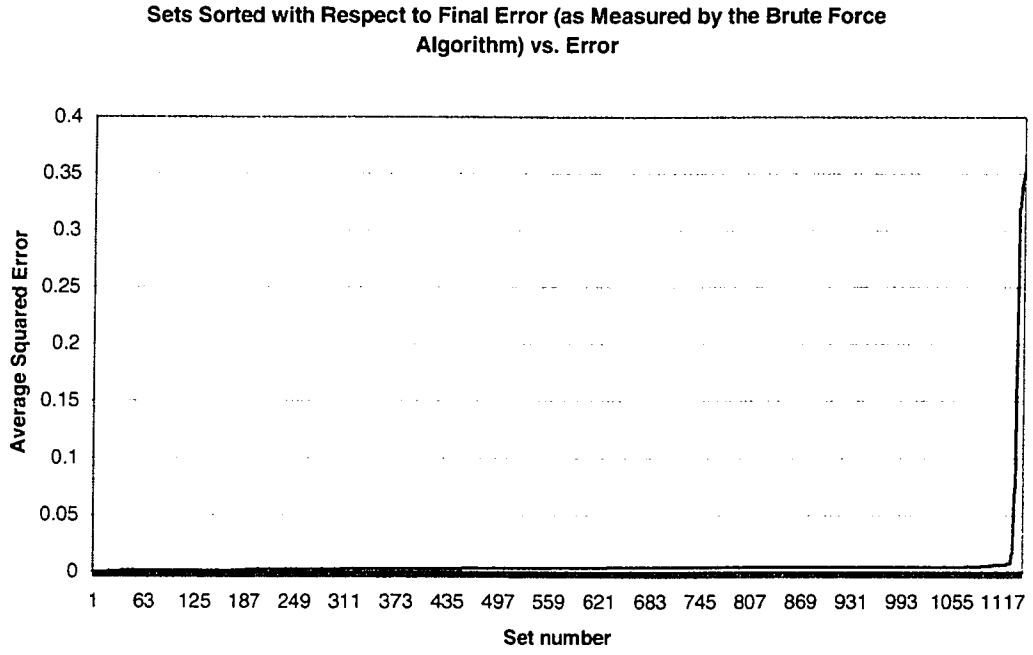
Table 3 - Settings of simulation 2

Uncorrelated uniform input	
λ=0.9999	E=E*0.99 + e*e*0.01
x[n]x[n]x[n-1]	0.7
x[n]x[n-1]x[n-2]	0.2

x[n-1]x[n-2]	0.3

For this simulation, the sets of filters with three terms and their corresponding errors (obtained from the brute force algorithm) are shown in the graph below:

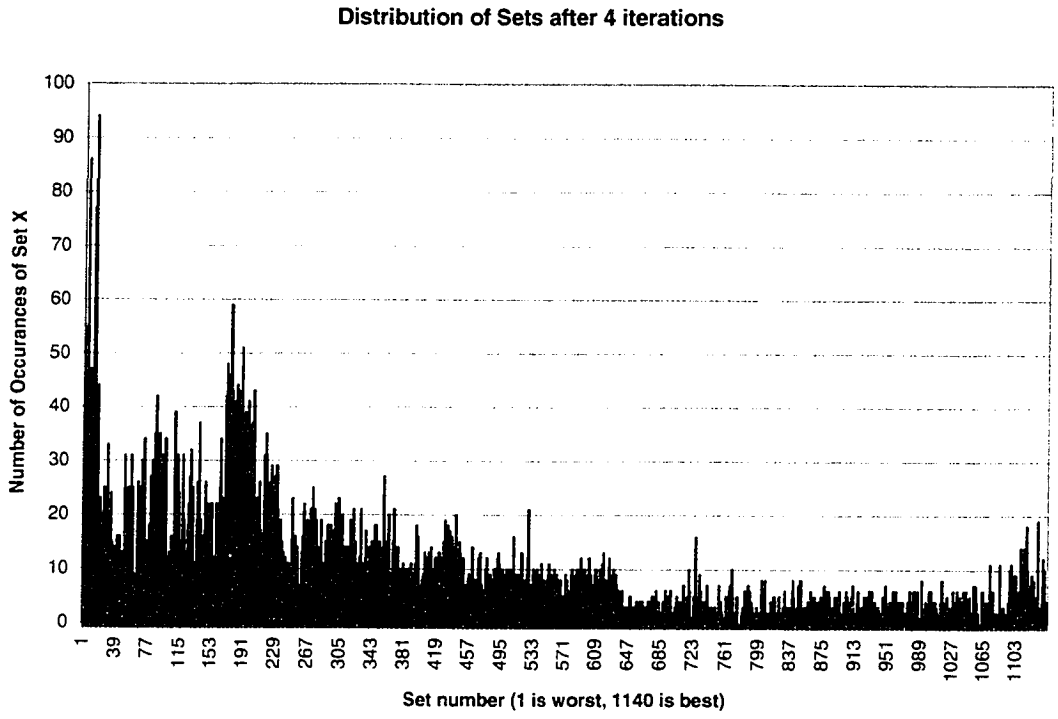
Figure 5.4 Errors for the given system. X-axis is sorted from best filter to worst filter



It should be noted that the ranking of the sets is not the same as in simulation 1 (e.g. set 1 for this simulation does not correspond to set 1 in simulation 1). For this simulation, most sets of terms perform well with the exception of a few that perform much worse than the others. The set on the left has an error of 0, while the set just before the knee of the curve has an error of 0.006.

The next graph shows the histogram of the sets chosen after the fourth iteration, out of 10000 runs.

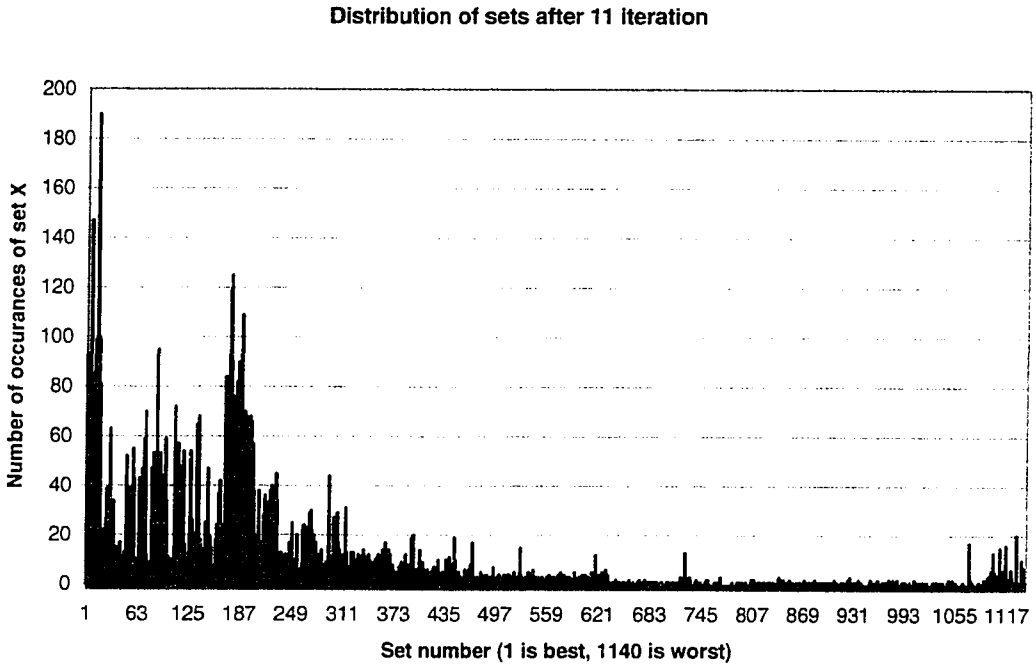
Figure 5.5 Distribution of sets after four iterations for simulation 2



Again there is clustering around # 190 and # 0 as well as around # 1128. The reason for this is the same as in the first simulation, namely the bias towards lower order terms.

The next graph is the histogram of filters after 11 iterations. At this point, the algorithm is performing well, except for the few times it lands in the poor performing filters.

Figure 5.6 Eleven iterations of simulation 2



The clustering continues, but overall the distribution is improved, we are able to find the best set within 10 sets (best 1 percentile) about 10% of the time, and the algorithm chooses a set better then the 50th percentile 95% of the time.

5.4.2 DISCUSSION OF RESULTS AND ALGORITHM

Simulations 1 and 2 will perform well if all three terms in the filter are orthogonal to each other. With non-linear terms of different orders, however, there is no input distribution which will yield orthogonal terms, which results in inaccurate estimates of which term to drop.

Simulations 3 and 4 use a different criterion for selecting which term to discard. When running the simulation, the main modeling filter contains three terms. Three

supplementary filters with two terms each are used to determine which of the three main terms to drop. Each of these three filters contain two of the three terms present in the main filter, with a different term missing from each filter. These supplementary filters are adapted simultaneously with the main filter (the main filter's output is not used by the algorithm, but will typically be used by the application), and the modeling error of each of these three filters is measured. After one complete algorithm iteration, the supplementary filter that produces the smallest error is determined, and the term missing from that filter is deemed the least important term in the main filter (i.e. the two remaining terms did the best job at reducing the modeling error). This least important term is discarded and replaced with a random term. For comparison, simulation 3 attempts to model the same system as simulation 1 and simulation 4 models the system of simulation 2.

5.4.2.1 SIMULATION

This system is modelling the four-term system shown below with a three term filter.

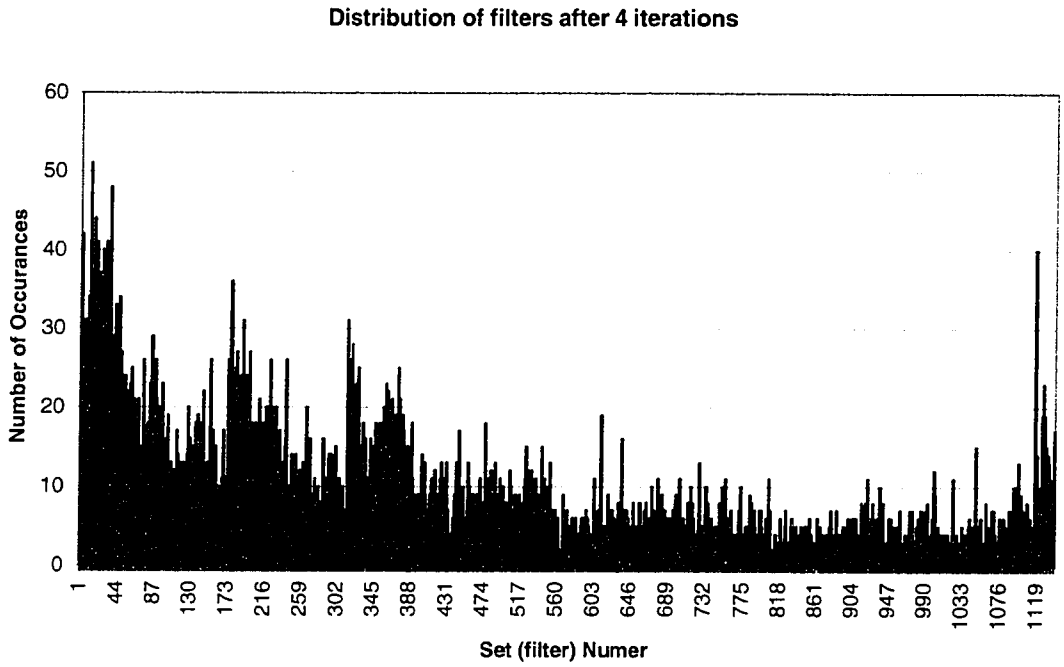
Table 4 - Settings of simulation 3

Uncorrelated uniform input	
λ =0.9999	E=E*0.99+e*e*0.01
x[n]x[n-1]	0.2
x[n]x[n-2]x[n-2]	0.3
x[n-1]x[n-2]x[n-2]	0.7

x[n-2]	0.8

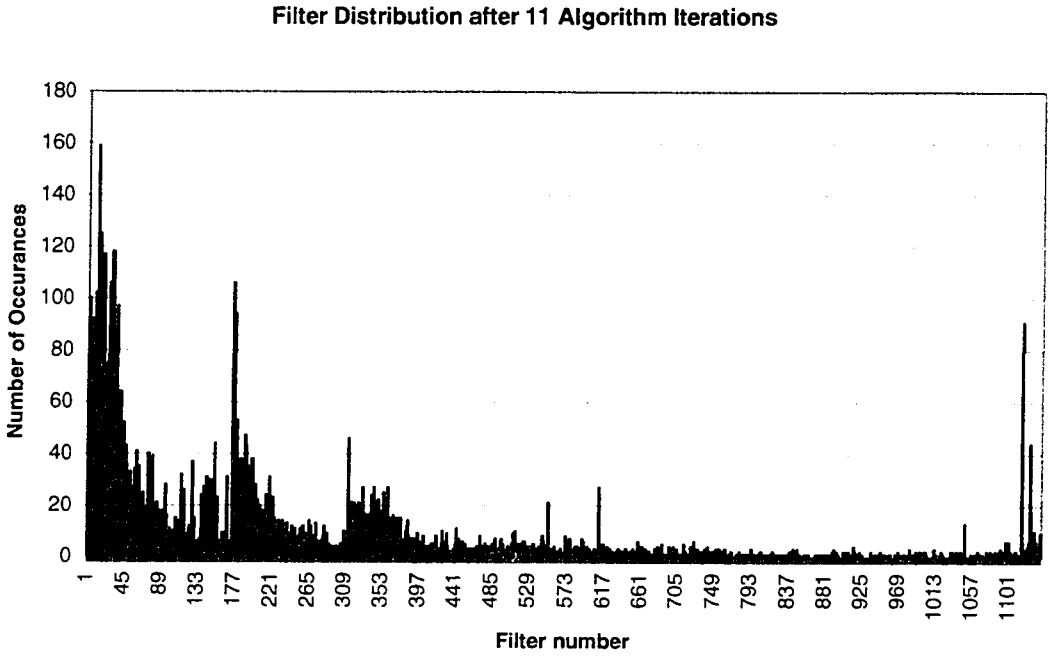
The distribution of filters after four algorithm iterations is shown below:

Figure 5.7 Distribution of filters after four iterations of simulation 3



After 11 iterations, this algorithm gives the following distribution:

Figure 5.8 Eleven iterations of simulation 3



After 11 iterations, the best filters appear much more frequently. There is still a problem with a few filters appearing that should not appear. This includes a cluster around # 190, one around # 350, and one near # 1130. These clusters are a smaller problem than in simulation 1, but are still an annoyance.

5.4.2.2 SIMULATION 4

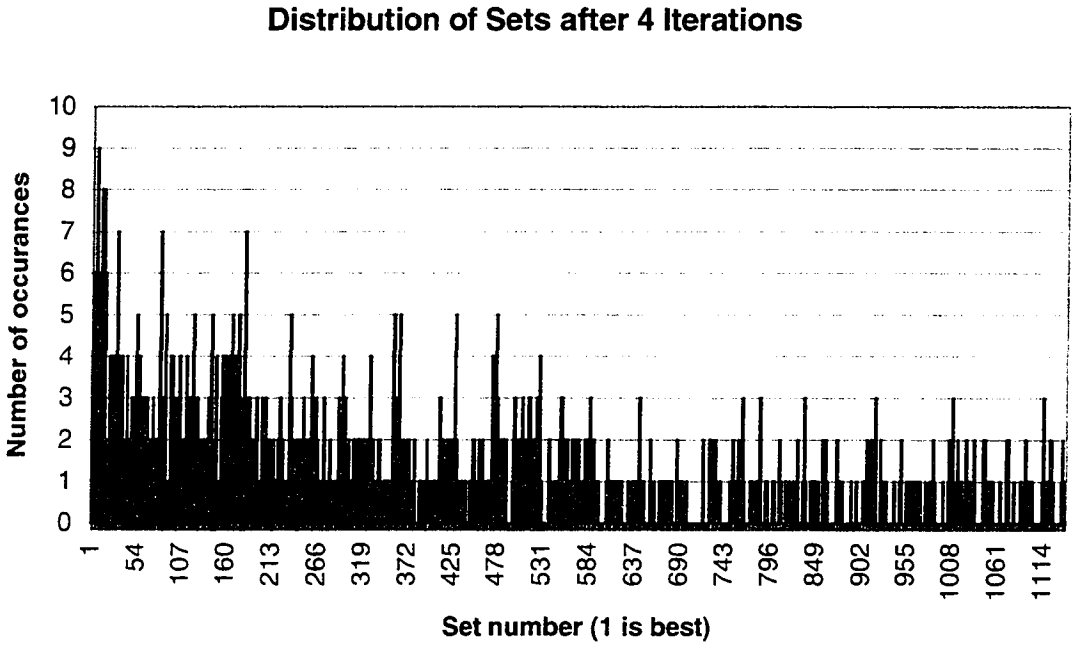
This simulation models the same system as simulation 2.

Table 5 - Settings of simulation 4

Uncorrelated uniform input	
λ=0.9999	E=E*0.99 + e*e*0.01
x[n]x[n]x[n-1]	0.7
x[n]x[n-1]x[n-2]	0.2
x[n-1]x[n-2]	0.3

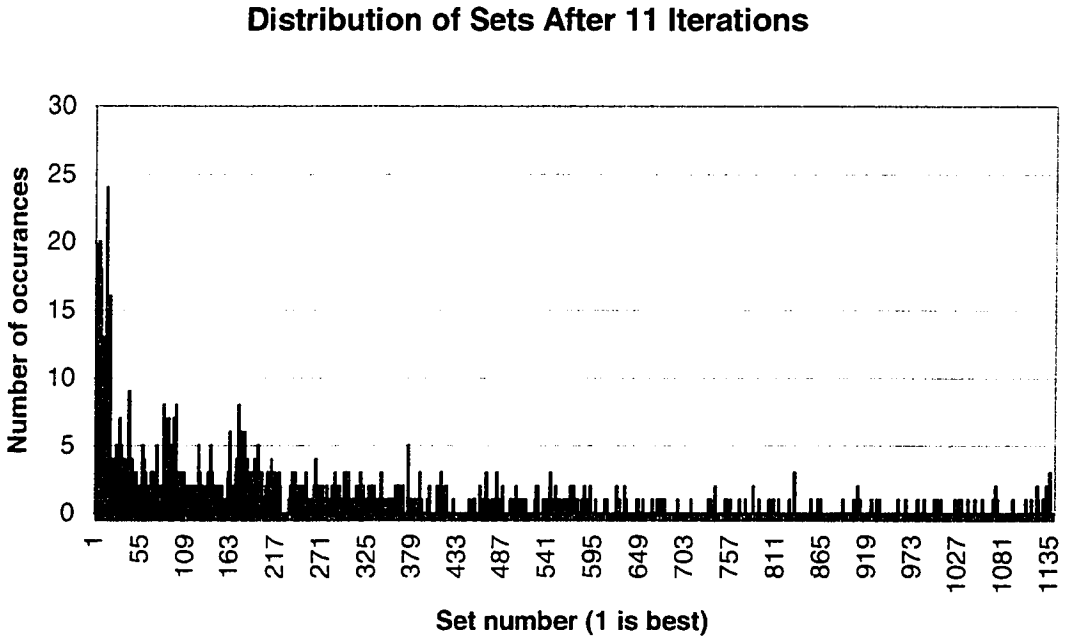
The following graph shows the distribution of filters after four algorithm iterations.

Figure 5.9 Simulation four after four iterations



The next histogram shows the distribution after 11 iterations.

Figure 5.10 Simulation four after 11 iterations



In this simulation, the sets cluster towards the best set, with no other clustering found.

Overall, this method of choosing a replacement term outperforms the first method, but it takes more time to execute.

5.4.3 CONCLUSIONS

For the systems tried, the term switching algorithms improve the likelihood of obtaining a good filter. The main questions about these algorithms are:

1) What happens if we keep trying the same sub-optimum sets over and over again?

2) Under what conditions can we expect to achieve the optimum filter/under what conditions will we end up at a sub-optimum filter?

The second method of choosing replacement terms where multiple test (speculative) filters are run in conjunction with the main filter is superior to the first method, but cost execution time and complexity.

Further investigation needs to be done on choosing a replacement term. One successful attempt to choose a better replacement term is to choose a term based on whether the error is more of an even function of the input or an odd function of the input. With this information, we can choose a replacement term that models the system better. The drawback is that by restricting our choices to even/odd terms, we increase our chances of being trapped in a loop where we choose only even terms because the error is predominately even (or alternately odd), and never try the sets with odd replacement terms.

Overall, these algorithms provide a simple means of obtaining a better truncation of Volterra filters than the arbitrary method of truncating them. The next section presents an algorithm that lies between the algorithms presented here and the brute force algorithm.

5.5 SWITCHED MULTIPLE SETS

The switched set methods consists of trying p sets, $p < C(m,n)$, ranking these p sets, then forming another p sets (dissolving the previous p sets), based on the information obtained from the previous p sets. The complexity of this algorithm starts out at $O(p*n)$. The simplest mode of operation for this class of algorithms is to choose p sets randomly. If the random selection is unbiased towards any sets of filters, the terms will be uniformly distributed with respect to percentile ranking when compared to all $C(m,n)$ sets. Hence, we can reasonably expect one of these sets to be in the top $100/p$ percentile. We can say with certainty that the best set will be in a better percentile than $100*(C(m,n)-p) / C(m,n)$. For example, if we choose p to be 4, on average, one of the filters will be in the top 25% percentile. If we choose p to be $C(m,n)$, we can say with certainty that the best term will be the optimum, since this algorithm becomes the brute force algorithm in this case. We can get as close to the optimum term as we like, being bounded by the second condition, but statistically following the first condition. However, the price in increasing p is the increase in complexity.

To improve on this simple mode of operation, we can attempt to use information gathered from previous filters we've already tried. We will use this information to make a somewhat intelligent choice of the next filters to try. This intelligent selection can be seen as

a random selection with a distribution skewed towards the sets in the higher percentile. We can influence this skewing by choosing terms that are similar to the previous best filter. For example, we could try filters that differ from the previous best filter by only one term. Alternately, we can try permutations of the previous two best filters. These two methods will only work if the best filter is already fairly good, and certainly ranks better than the top 50 percentile. Another method is to choose the best terms (the terms that contribute most to reducing the error in that filter) from all p filters (or the best of the p filters) and amalgamate them into a new filter.

5.5.1 THE ALGORITHM

This algorithm attempts to find the optimum or near-optimum filter by examining a reference filter and all filters which differ from the reference filter by just one term. The analysis involves determining which filter from the above set best matches our desired system response. When this filter is identified, it becomes the new reference filter, and all filters differing from it by just one term are analyzed. By iterating through this algorithm, it will be shown that we will often achieve the optimum filter after a very small number of steps.

The algorithm proceeds as follows:

1) Choose a starting filter

2) Find all filters differing from the starting filter by 1 term

3) Adapt the filters to an optimal value

4) Choose the best filter

5) Continue at step 2

Simulations show this algorithm is capable of finding the optimum filter in very few steps with very good accuracy. The major drawback is the complexity of executing this algorithm. For example, if we decide on a three term filter with the terms chosen from 20 terms, we must adapt 17+17+17=51 filters.

5.5.2 JUSTIFICATION OF THE ALGORITHM

This algorithm is a hybrid between the brute force algorithm, where we try all combinations of terms to find the best filter, and the term switching algorithms, where we try one filter at a time, constraining ourselves to filters that differ from the current filter by just one term. In effect, this algorithm is the brute force method of choosing replacement terms in the term switching algorithm. We decide on which term to drop and which to add by trying all combinations of terms which are similar to the current filter. The similarity of the candidate filters and the current filter provides the control needed to slowly step to the optimum filter.

5.5.3 METHODS OF VALIDATING THE ALGORITHM

To evaluate this algorithm, we choose a starting filter, iterate through the algorithm, and record the path of filters the algorithm traverses. To be thorough, we can choose every possible filter as the starting point and iterate through the algorithm for each. We can then examine the recorded paths and determine whether the optimum filter was reached, and whether the algorithm loops, a condition where filter A leads to filter B which leads to C ...

and eventually back to A. If the optimum filter is within the loop, and the worst filter within the loop is fairly close to the optimum, then the algorithm can be considered successful.

5.5.4 PROBLEMS WITH THE MULTIPLE SET ALGORITHM

This algorithm is based on a heuristic approach, no optimality is guaranteed. Depending on the system to be modeled, we can be placed in a situation where we loop around multiple bad filters because we fall into a local minimum. The larger the number of filters being searched in a set, the smaller the chance of being placed in a local minima, but it can still happen. The other problem is the complexity. If we decide on using an n-term filter to do the modeling, and choose these n terms from a larger set of m terms, at each iteration we will need to adapt $n*(m-n)$ filters.

5.5.5 SEARCH SCOPE OF THE ALGORITHM

The current algorithm searches a subset of filters which differ from the current filter by one term. Assuming that the algorithm continues for a number of steps, how many filters are within range of a given starting set after q steps?

Given an initial n-term filter, with terms chosen from a set of m-terms, we need to determine the number of filters which differ from this initial filter by exactly q terms.

For example, if q=1, the number of filters differing from the initial filter is $n*(m-n)$, since we have n positions in the filter, and at each position we have $(m-n)$ replacements (since duplicate terms are not allowed, we must choose a replacement from the m terms that doesn't exist in the n term filter in another position).

We begin the analysis by noting that the operation consists of two steps, choosing the q terms to replace (or equivalently n-q terms to leave), and choosing the q replacement terms which are not present in the original filter. The result is the product of these two operations.

The number of combinations of q terms chosen from n terms is $C(n,q)$ (note $C(n, n-q) = C(n,q)$). The number of combinations of q terms chosen from m-n terms is $C(m-n,q)$ (m-n terms are not present in the original filter). We are left with the result:

The number of filters which differ from the starting filter by q terms is $C(n,q)*C(m-n,q)$.

The current experiment will search all filters differing by 0 terms, 1 term, 2 terms...q terms. In the current set of experiments, we have $m=20$, $n=3$. A table showing the number of filters within the search space after q steps is given below:

Table 6 - Number of filters that can be searched after 'q' iterations

q	*Number of Filters in Search Space*
1	52 (51+1)
2	460 (408+52)
3	1140 (680+460)

1140 is the total number of combinations of three term filters, so after 3 steps we have the capability to search the entire space.

5.5.6 SIMULATIONS OF THE ALGORITHM

The first system we try to model is the four term filter used in previous simulations:

5.5.6.1 CONDITIONS:

The worst term is selected by keeping track of the error obtained by the full filter minus the contribution of that term (i.e. $e_n=d-(y-c_n)$, where d is the desired signal, y is the filter's estimate, and c_n is the contribution of term n. Terms are randomly replaced.

RLS is used for adaptation, with $\lambda=0.9999$, adaptation between term switches takes 200 samples, average error is determined as E = 0.99*E + 0.01*e*e, where E is the average error, e is the instantaneous error. Input is uncorrelated uniformly distributed noise ranging from -1 to 1.

The system to be modeled is:

Table 7 - Settings for simulation

Term	*Coefficient*
x[n]x[n-1]	0.2
x[n]x[n-2]x[n-2]	0.3
x[n-1]x[n-2]x[n-2]	0.7
x[n-2]	0.8

We attempt to model this filter with a three term filter, chosen from the set of Volterra terms with memory of 3 or less, and order 3 or less. There are 20 terms in this set, so at each iteration, we analyze 51 filters. To verify this algorithm, we try all possible 1140

starting filters and iterate through the algorithm. At each step, we record what the current "best" filter is before proceeding to the next step. The raw results are shown here for starting points near the best filter and starting points near the worst filter. Filter number 1 is the best performing filter, as found through the brute force method, and filter 1140 is the worst filter, with all other filters numbered according to their rank based on the brute force method. Each row corresponds to a run starting the indicated set number. For example, when the algorithm started at set number 7, it then determined set number 1 was the next best set differing by one term from set 7 in the first iteration.

Table 8 - Filters chosen by algorithm given a starting filter

Starting filter	1 iteration	2 iterations	3 iterations	4 iterations	5 iterations	6 iterations
0	1	0	1	2	0	1
1	0	1	2	0	1	19
2	0	18	0	1	0	1
3	0	1	0	1	2	0
4	0	1	0	1	0	1
5	0	1	0	1	0	1
6	0	1	0	1	8	0
7	1	4	0	1	0	1
8	0	1	0	1	4	0
9	0	1	0	1	0	2
10	1	0	1	0	1	0
11	0	1	0	1	0	1
12	1	4	0	1	0	1
13	0	1	14	0	1	0
14	0	1	2	0	1	0
15	0	1	0	1	0	1
16	0	1	0	1	0	1
17	1	0	1	0	1	0
18	0	1	0	1	2	0
19	1	0	1	0	1	0
...						
1100	128	12	0	1	0	1
1101	145	15	0	1	3	0
1102	130	15	0	1	0	1
1103	130	15	1	0	4	1
1104	129	15	1	2	0	1

1105	135	12	1	0	1	2
1106	135	12	0	1	0	1
1107	140	7	0	1	0	1
1108	137	15	0	1	0	1
1109	136	6	0	1	0	1
1110	137	15	0	1	2	0
1111	129	31	11	0	1	0
1112	150	10	0	1	0	1
1113	161	9	0	1	16	1
1114	128	13	0	1	0	9
1115	150	12	0	1	0	1
1116	159	10	1	0	1	0
1117	169	9	0	1	0	1
1118	145	15	0	1	2	1
1119	159	15	0	1	0	1
1120	156	10	0	1	0	1
1121	23	4	0	1	0	1
1122	166	9	1	0	1	0
1123	150	12	0	1	4	1
1124	162	10	0	1	0	1
1125	137	15	0	1	0	11
1126	169	9	1	2	0	1
1127	146	6	0	1	0	1
1128	22	3	0	1	0	1
1129	23	4	0	4	0	1
1130	23	4	0	1	0	1
1131	94	2	0	1	2	0
1132	106	3	1	0	1	0
1133	118	4	0	1	0	1
1134	119	4	1	0	1	2
1135	24	1	0	1	0	1
1136	81	1	0	1	0	1

The middle sets are not included due to the length of the table, however the table obtained shows that no matter what the starting point is, we will reach the best set within five iterations of the algorithm. Due to the randomness of the input, occasionally the algorithm does not choose the best available filter. This is because the error generated by the best filter and other good filters is often very close, and in any particular run, the good filters may slightly outperform the best filter, due to the finite run length. Nevertheless, the sets 11, 18, and 19 are all very good performers, operating above the 98th percentile. The minor

variations from the best set are due to the adaptation and randomness of the input. This can be shown by the following experiment:

1) Run the brute force algorithm on all 1140 filters, and rank the filters from best to worst.

2) For each of the 1140 filters, follow the algorithm and record the filters at each set in a table. Instead of adapting the filters to find the best performing filter, simply look up the filters from the table generated by the brute force algorithm. This has the effect of eliminating the randomness of the input since the lookup will always yield the same filter consistently, while adaptation will have minor fluctuations which may affect the final error of a filter, and hence its ranking.

The results of this experiment indicated that no matter what the starting filter is, in less then five iterations, the current filter will fluctuate from 0 to 1 and back to 0. Since this process is completely deterministic, once we get into a repeating pattern, the pattern will repeat forever. For brevity, the results of this experiment are not shown here.

This simulation has been run on several other systems up to third order, memory of three, with similar results, regardless of the nature of the input (correlated, uncorrelated, many distributions).

5.6 ORTHOGONAL POLYNOMIALS

The problems of the previous sections can be eliminated if only the terms of the modelling function were orthogonal to each other. Discussed in this section is the method

used to generate orthogonal basis functions using Volterra terms. Given a particular distribution, an orthogonal *set* of polynomials may be found using Gram-Schmidt orthogonalization. Using an orthogonal set of polynomials rather than non-orthogonal polynomial terms as the filter basis greatly simplifies searches for the best subset of basis polynomials. Given this orthogonal set, the best set of n basis functions that most reduce the mean squared error of the system are the n basis functions that individually most reduce the squared error of the system. Having an orthogonal basis makes all bases independent.

An example of this set of polynomials is of the form:

$$\begin{aligned} P_o(x) &= 1 \\ P_1(x) &= x - a_{1,1}\frac{P_0(x)}{|P_0(x)|} \\ &\cdots \\ P_i(x) &= x^i - a_{i,i-1}\frac{P_{i-1}(x)}{|P_{i-1}(x)|} - a_{i,i-2}\frac{P_{i-2}(x)}{|P_{i-2}(x)|}\cdots \end{aligned} \tag{5.1}$$

where,

$$a_{i,j} = -\frac{1}{|P_{j-1}(x)|}\int_{-\infty}^{\infty} x^i P_{j-1}(x)\theta(x)dx \tag{5.2}$$

In this case, the n-th orthogonal polynomial in the set is of order n (may contain terms of power 0-n). Of course, with a symmetric distribution, two sets of polynomials may be independently derived, since even-powered terms are automatically orthogonal to odd-powered terms. These sets of polynomials will be orthogonal to each other.

To calculate the set of orthogonal polynomials, we need to determine $n(n-1)/2$ coefficients, where n is the number of terms in the polynomial. However, using a set of orthogonal polynomials, it is possible to extend the order of a system by adding the next higher order polynomial. In this case, the previous coefficients do not need to be recomputed. In addition, when searching for the best terms to keep in a filter representation, less useful polynomials can be dropped without affecting the other polynomials. The usefulness of this is mostly outweighed by the large number of coefficients which need to be calculated to determine the orthogonal polynomials.

5.7 CONCLUSIONS

This chapter has presented three classes of algorithms for determining optimal or near-optimal Volterra filters chosen from a finite set of terms. These algorithms include:

- The brute force algorithm which tries all combinations of terms and decides on the best combination
- The single term switching algorithm which starts at some collection of terms and switches out one of those terms and replaces it with another
- The multiple set switching algorithm which maintains a number of prospective filters and chooses the best performing of these filters.

Each of these algorithms has different ways of obtaining information about the redundancy of the terms of the filter. This information is not stored directly in any of these algorithms, but the adaptation is effectively searching out the least redundant set of filters.

The brute force algorithm stores this information in the form of a list of errors of each filter. The single term switching algorithm maintains this information in its current state, which includes the current filter being analyzed and the transition to the next filter, and the multiple set switching algorithm maintains this information in the form of its current state which includes the list of prospective filters and the current best filter.

CHAPTER 6-DETERMINING THE OPTIMUM VOLTERRA TERM

The algorithms presented in Chapter 5 have a number of drawbacks, including computational complexity, non-optimality, and discontinuous error performance (due to the switching of terms). This chapter presents an algorithm for determining the best terms in a more continuous adaptive fashion. The method in this chapter extends the class of non-linear filters from the Volterra filter to a more general structure. This extended class of filters can then be used to determine the best set of terms to use and can be used directly for filtering.

This chapter first generalizes the Volterra filter to real valued exponents, to gain continuity for adaptation. The following subsection describes a method for finding the single best Volterra term based on a Taylor series representation of the modelling function. The next subsection provides results for extending the single term algorithm to multiple terms. The chapter ends with a discussion of truncating real valued exponents back to integer values.

6.1 AN EXTENDED CLASS OF FILTERS

A truncated discrete time Volterra filter can be represented as:

$$y[n] = \sum_i h_i \prod_{k=0}^{j} x[n-k]^{g_{k,i}} \tag{6.1}$$

where i is the number of terms in the truncated filter, j is the filter time span (filter truncation in time), h_i is the coefficient of the i^{th} Volterra term, and $g_{k,i}$ are **integer** exponents. Of course $g_{k,i}$ can be zero, or any positive integer. An extended class of filters can be obtained by allowing $g_{k,i}$ to be real numbers. In this case, the output will be real if the input is real and positive. This extended class of filters has greater flexibility in modeling systems, and is particularly useful for truncated filters. As will be shown, the continuous nature of the exponents lends itself more readily to adaptation than do the discrete (integer) exponents. Until now, we have restricted the continuous adaptation to the coefficients. The search through the exponents was achieved by switching terms rather than adapting the exponents or sets of terms.

Truncating Volterra filters is hampered by the jumps in performance between neighbouring terms. For example, x^2 and x^3 have very different modelling characteristics. Finding the discrete set of terms that best model a system is a tricky task if we are trying to jump from one term to another with integer exponents since there is no continuity between one set of terms and another. However, by allowing the exponents to take on real values, the terms become continuous. Between x^2 and x^3, there are an infinite number of steps, each step modeling the previous step in a minutely different way. The continuity of these exponents provides a method of adapting them.

Computation of the exponential function is generally more complex when real numbered exponents are used rather than integer values. Often a truncated polynomial will

be used for this evaluation and can take several times longer than an integer exponential to compute. Depending on the desired accuracy, the truncated polynomial will often use 7 terms or more, meaning a 7x increase in execution time. The C math library implementation of the *pow()* function will be dependent on the compiler's implementation. It should be noted that some processors provide instructions to perform this operation. For example, recent Intel processors include the F2XM1 (compute 2^x-1) instruction, which can be used in conjunction with FYL2X (compute $y*\log_2 x$) to implement x^y in two machine instructions [1], making it practical to implement these algorithms. In other circumstances, using the C math library *pow()* function will provide sufficient performance.

6.1.2 SHIFTING INPUTS TO NON-LINEAR SYSYEMS

Using real valued exponents implies positive input value. In many cases, it may be necessary to shift the input values so they are always positive. Unlike linear systems, non-linear systems are not invariant to linear transforms of the input. Hence if an input has a distribution with negative portions, we cannot simply add a constant to it to make it all positive without affecting the model. For example, suppose we were modelling the function $y=x^2$, with a uniformly distributed input from -1 to 1. If we wanted to shift the input by 1 to make it all positive before we pass it into our modelling function ($x'=x+1$), the optimum modelling function would become $(x'-1)^2$. This makes the modelling function different (in terms of representation) than the modelled function. However, the modelling function with the shift is identical to the modelled function without the shift. The shifted version contains more terms (when expanded), but it could have just as easily contained less, and most likely would contain the same number of terms as the unshifted version. A linear transform of an input will not increase the order of a polynomial. Hence given a polynomial of a certain

order, the ability of that polynomial to model an unknown system is unchanged by a shift of the polynomial's input. This is a good thing, since typically sampled values are scaled and shifted from their actual physical values. For example, a temperature sensor outputs measurements in volts, even though it is measuring temperature (typical units of Celsius, Fahrenheit, Kelvin...). Furthermore, sampling the output of this temperature sensor changes the units, for example by shifting and scaling them into the range of 0 to 255. Hence the final digital samples are in some other units than the original intended units. These units could be shifted back, but it is unnecessary to do so, since the modelling function will take all the shifts into account.

6.2 ANALYSIS OF THE MEAN SQUARED ERROR FUNCTION

Before embarking on a description of the algorithms for adapting exponents, we analyse the mean squared error function. From this analysis we will show the existence of multiple minima and obtain an understanding of the mean squared error surface under examination.

In the typical modelling situation, we have and input sequence x(n), an output sequence y(x) and a modelled output sequence y'(x). To simplify the analysis, we work with a memoryless polynomial with two terms as our modelling function.

$$y'(x) = gx^a + hx^b \tag{6.2}$$

Our instantaneous error is shown below:

$$e(x) = y(x) - y'(x) \tag{6.3}$$

The mean squared error is:

$$E[e^2(x)] = E[(y(x) - y'(x))^2] \tag{6.4}$$

Expanding equation (6.4), we obtain:

$$E[e^2(x)] = \int_0^\infty [y^2(x) - 2y(x)y'(x) + y'^2(x)]\theta(x)dx \tag{6.5}$$

The function Θ(x) is the probability density function of x (as allowed by the change of variables theorem [30]). The lower limit of the integral reflects the fact that x must be positive. Substituting (6.2) into (6.5), we get:

$$E[e^2(x)] = \int_0^\infty [y^2(x) - 2y(x)(gx^a + hx^b) + g^2x^{2a} + h^2x^{2b} + 2ghx^{a+b}]\theta(x)dx \tag{6.6}$$

To find the optimum parameter values, we differentiate (6.6) with respect to g, h, a and b respectively, and find the points at which these derivatives are zero, corresponding to the minimum mean squared error. The coefficients g and h yield the following well known minima equations:

$$g = \frac{\int_0^\infty [y(x)x^a + hx^{a+b}]\theta(x)dx}{\int_0^\infty x^{2a}\theta(x)dx} = \frac{E[y(x)x^a] + hE[x^{a+b}]}{E[x^{2a}]} \tag{6.7}$$

and

$$h = \frac{\int_0^\infty [y(x)x^b + gx^{a+b}\theta(x)]dx}{\int_0^\infty x^{2b}\theta(x)dx} = \frac{E[y(x)x^b] + gE[x^{a+b}]}{E[x^{2b}]} \tag{6.8}$$

Since these two equations are linear in g and h, there will be one minimum value of (g, h) for each pair (a,b).

The optimum values of a and b are more difficult to determine. Differentiating (6.6) with respect to a and finding the minimum, we obtain:

$$0 = \int_0^\infty [-y(x)x^a + gx^{2a} + hx^{a+b}]\ln(x)\theta(x)dx \tag{6.9}$$

Similarly for b:

$$0 = \int_0^\infty [-y(x)x^b + hx^{2b} + gx^{a+b}]\ln(x)\theta(x)dx \tag{6.10}$$

Solving for a and b is difficult in the general case. However, these equations are second order in the random variables x^a and x^b (multiplied by the monotonically increasing function ln(x)). Since x^a is a monotonically increasing function of a, equations (6.9) and (6.10) will intersect on at most 2 points given fixed values of g and h. Hence the mean squared error will have up to two minima, one of which may be a local minima. The same analysis can be extended to a system with n terms (n equations each of power n) to show that there may be

up to n! minima of the mean squared error function. If g=h, then the two minima will both be global minima (i.e. (a,b) and (b,a)), due to the symmetry of equation (6.2).

6.3 DERIVATION OF EXPONENT ADAPTATION ALGORITHMS

The algorithm derived in this chapter provides a means of adapting the exponents of the Volterra modelling function. To simplify the explanation and provide insight into the derivation of the algorithm, this chapter starts by looking at simplified systems, and progressively adds features until the final algorithm can be used with a general Volterra filter.

To start, we take a look at the simple one term memoryless exponential modelling function taking the form:

$$f(x) = x^a \tag{6.11}$$

where a is a real number exponent and the value we wish to adapt. We note that the function $f(x)$ is continuous in the region $(0,\infty]$ for all real a. With this condition, we can state that there exists a Taylor series representation of this function over this region. We can write the function $f(x)$ as a Taylor series as shown:

$$f(x) = \sum_{0}^{\infty} \frac{f^n(c)}{n!}(x-c)^n \tag{6.12}$$

$f^n(c)$ is the n^{th} derivative of $f(x)$ evaluated at c. We now have two equivalent representations of the modelling function. The Taylor series representation is a power series with fixed exponents and variable coefficients, whereas the exponential representation has a fixed coefficient (1 in this case), and a variable exponent. The fact that we have represented this

function as a Taylor series is important since, as we have already shown in Chapter 3, we know how to adapt *coefficients* of a power series. In this case, there is an infinite number of coefficients since the Taylor series of the exponential function will have an infinite number of derivatives for all exponents that are not integers. This does not worry us, because for the function in question, there is only one degree of freedom, hence if one coefficient is known, all others can be derived from it. In fact, if one coefficient is known, the value of the exponent can be determined, so we can change from the Taylor series representation to the exponential representation at will. We know that n^{th} derivative of the function is:

$$f^n(x) = x^{a-n} \prod_{k=0}^{n-1} (a-k) \tag{6.13}$$

For example, the first derivative is ax^{a-1}, and the second derivative is $a(a-1)x^{a-2}$. In this case since we only have one degree of freedom, we only need one derivative to find the value of the exponent, so we choose the simplest one (the first), though any will do. If we choose c in the Taylor series representation to be 1 (any value of $c > 0$ will also do, 1 is the simplest), we eliminate the x^{a-n} product, and we are left with

$$f^1(1) = a \tag{6.14}$$

So the Taylor series representation of x^a will have a coefficient of a (equivalent to the exponent) for the first order term when the Taylor series is centred on 1.

We can now state the adaptation algorithm for the one term exponential function. We will use the variables d to represent the desired signal (output of the desired function), y as the output of the modelling exponential function, and e as the modelling error, defined as

$$e = d - y \tag{6.15}$$

and a being the modelling exponent. We have our modelling function:

$$y = x^a \tag{6.16}$$

This corresponds to the LMS modelling diagram in Figure 6.1:

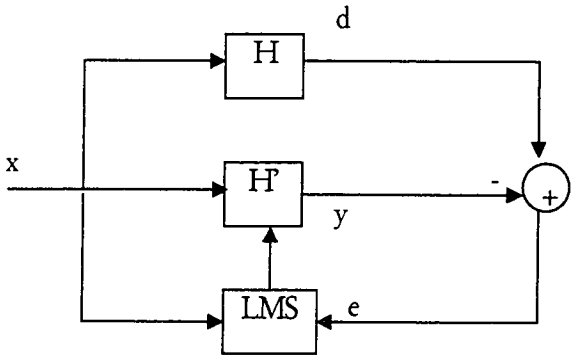


Figure 6.1 System used for modeling

RLS is not directly suitable for this type of modelling, since it does not utilize an error feedback. RLS determines the correlation between the input and the desired output to adapt the coefficients, hence (6.16) is never used. Thus the Taylor representation used is not constrained by (6.16), and is instead constrained by making all terms not present in the coefficient vector zero. Because of this difference, RLS will find the optimum coefficient of the term *(x-1)* rather than the exponent of x.

The algorithm for adapting a is given in the following six steps.

1) Choose a starting value for a. Choose a value that won't overflow the arithmetic, 0 is often a good choice.

2) Obtain a sample of x and d.

3) Evaluate the modelling function's output, y, using the exponential form (x^a).

4) Determine the modelling error, e.

5) Update the exponent using the Taylor representation, specifically using the first order term of the Taylor representation centred on 1. The update may use or LMS or some other adaptation algorithm which includes an error feedback. The LMS update is: $a = a + \mu * e * (x-1)$. The $(x-1)$ is the first order term of the Taylor series (centred on 1), of which a is the coefficient, hence we are correctly adapting the coefficient of the first order term which is equivalent to the exponent.

6) Continue back at step 2.

Note that in step 3, when we evaluate the function's output, we are using the exponential form of the output, not the Taylor form. Using the exponential form is mathematically equivalent to using the full infinite series Taylor form, so by adapting the first order term, we are in effect adapting the coefficients of all the terms of the Taylor form, since they are constrained by each other.

This algorithm uses the first order term of the Taylor series, but any term could have been used, as long as the correct transformation between the coefficient and the exponent was done. For the first order term, there is no necessary transformation, since the coefficient is equivalent to the exponent. If we had instead used the second order term, which has a coefficient of $a(a-1)$, we would now have adapted the Taylor coefficient h_2 (for

the second order term). Now $h_2=a(a-1)$, so we would adapt h_2 and transform it by solving this equality for a. Using the quadratic formula, we would write

$$a = \frac{1 \pm \sqrt{1 + 4h_2}}{2} \tag{6.17}$$

which has two solutions, only one of which is correct. To do adaptation, we would need to adapt using complex numbers, since there is a chance that during adaptation the value of the exponent becomes complex. The factor of 2! dividing h_2 in the Taylor series representation can be effectively ignored, since it would only affect the starting point, and adaptation will have the affect of absorbing this constant factor. For most purposes taking this route is not effective, and it is shown here only for completeness.

6.3.2 ADAPTING MULTIPLE TERM MODELLING FUNCTIONS

Since we have an infinite number of coefficients to choose from in the Taylor series, we effectively had an infinite number of equations with one unknown in the previous section. This leads us to the discussion of adapting multiple termed modelling functions.

The multiple term memoryless modelling function takes the form:

$$f(x) = x^{a0} + x^{a1} + \ldots + x^{am} \tag{6.18}$$

where $f(x)$ is the modelling function, x is the input and $a0 \ldots am$ are the exponents. We can say that this function is continuous on the same region as the single term function (it is a sum of continuous functions on this region), hence we can say that there exists a Taylor series for this function. Again we will be interested in the Taylor series centred around 1

(because it eliminates the exponent in the coefficient). We can determine the derivatives of this function:

$$f^n(x) = x^{a0-n}\prod_{k=0}^{n-1}(a0-k) + x^{a1-n}\prod_{k=0}^{n-1}(a1-k) + \ldots + x^{am-n}\prod_{k=0}^{n-1}(am-k) \tag{6.19}$$

evaluated at x = 1, the derivatives become

$$f^n(1) = \prod_{k=0}^{n-1}(a0-k) + \prod_{k=0}^{n-1}(a1-k) + \ldots + \prod_{k=0}^{n-1}(am-k) \tag{6.20}$$

We now have multiple exponents that we need to find, and there is no single derivative we can use that isolates each exponent into a Taylor coefficient. However, we do have an infinite number of Taylor coefficients, each with a different corresponding derivative. From this we can choose m equations and solve for the m unknown exponents after each step of adaptation. We end up with the following sets of equations by choosing the m lowest order derivatives:

$$\begin{aligned} h_1 &= a0 + a1 + a2 + \ldots \\ h_2 &= a0(a0-1) + a1(a1-1) + a2(a2-1) + \ldots \\ h_3 &= a0(a0-1)(a0-2) + a1(a1-1)(a1-2) + a2(a2-1)(a2-2) + \ldots \\ &\ldots \end{aligned} \tag{6.21}$$

By adding and subtracting equations, we arrive at the following set of equations:

$$
\begin{aligned}
h_1 &= a0 + a1 + a2 + \ldots \\
h_2 + h_1 &= a0^2 + a1^2 + a2^2 + \ldots \\
h_3 + h_2 - h_1 &= a0^3 + a1^3 + a2^3 + \ldots \\
&\ldots
\end{aligned}
\tag{6.22}
$$

With h_1, h_2, h_3, ... all known through adaptation, obtaining the exponents is a matter of solving the simultaneous equations. They can be solved analytically, if the order is not too high, or algorithmically, using a root finding method.

Alternately, we can obtain equations by evaluating Taylor expansions centring on values other than 1. There are an uncountable number of centres we can use, yielding as many equations as we need. For example, using the first derivative but differing centres, we obtain the following equations:

$$
\begin{aligned}
h_1 &= a0{c_0}^{a0} + a1{c_0}^{a1} + \ldots \\
h_2 &= a0{c_1}^{a0} + a1{c_1}^{a1} + \ldots \\
&\ldots
\end{aligned}
\tag{6.23}
$$

These equations can be solved analytically (the solution will involve the *lambertw* function [30]), or algorithmically. In addition, the two methods can be mixed, using different centres and different derivatives to form the system of equations.

Because of the need to solve a system of equations at every algorithm iteration, these forms are not well suited to adaptation. This method of finding the exponents is tedious and difficult, compounded by the fact that real values of h_n may yield imaginary exponents. However, we can rewrite the single term as shown in equation (6.24), where n is any real number.

$$x^a = x^n x^{a-n} \tag{6.24}$$

We can easily write (6.24) in the form of a Taylor series centred at 1, as shown in equation (6.25).

$$\begin{aligned} x^a &= x^n x^{a-n} \\ &= x^n [\sum_{i=0}^{\infty} \frac{(x-1)^i}{i!} \prod_{j=0}^{i-1} (a-n-j)] \\ &= x^n + \frac{(a-n)x^n(x-1)}{1!} + \frac{(a-n)(a-n-1)x^n(x-1)^2}{2!} + \ldots \end{aligned} \tag{6.25}$$

This form has the following desirable characteristics

1) The lowest order term is x^n

2) The coefficient is linear in *a* for the first term it appears in

The first property is useful because we can now *shift* our Taylor series representations by any order, meaning that if we have a sum of terms as in equation (6.18), we can shift the Taylor series representation for the first term by 0, shift the second term by 1, the third term by 2, and so on. Thus the term *(x-1)* in the summation of these Taylor series' will have a coefficient of *a0*, the same as the first exponent. The coefficient for the term *x(x-1)* is *(a1-1)*, the coefficient for x^2*(x-1)* is *(a2-2)*, and so on as shown in (6.26).

$$
\begin{aligned}
x^{a0} + x^{a1} + x^{a2} + \ldots = \\
&(1 + a0(x-1) + \frac{(a0-1)(x-1)^2}{2} + \ldots) + \\
&(x + (a1-1)(x-1)x + \frac{(a1-2)(x-1)^2 x}{2} + \ldots) + \\
&(x^2 + (a2-2)(x-1)x^2 + \frac{(a1-3)(x-1)^2 x^2}{2} + \ldots) + \\
&\ldots
\end{aligned}
\tag{6.26}
$$

This ability to isolate the exponents into their own coefficients eliminates the need to solve a system of equations. The fact that the first coefficient is linear in a is useful since no nonlinear equations need to be solved for the exponent (which is now independent of the other exponents). Also, all terms in the Taylor series representation that arise from the first term of (6.18) all have the form $h(x-1)^i$, where h is the coefficient. All terms that arise from the second term have the form $h*x(x-1)^i$, from the third term $h*x^2(x-1)^i$, and so on. Thus the Taylor terms do not mix with other Taylor terms. This means that given the value for $a0$, the value of all the coefficients belonging to the first term in (6.18) can be determined, regardless of what the other exponents are. This independence is what frees us from a system of nonlinear equations, and gives us independent linear equations.

The representation in (6.26) is not a unique representation. We are adding many Taylor series together. Each one could individually represent equation (6.18), and together they no longer form a unique representation of the function. However when constrained to modelling a system, this representation will be unique.

Using the shifted Taylor representation, we can write the updates for LMS:

$$
\begin{aligned}
&a0' = a0' + \mu e(x-1) \\
&a1' = a1' + \mu e x(x-1) \\
&a2' = a2' + \mu e x^2(x-1) \\
&\ldots \\
&\textit{and} \\
&a0 = a0' \\
&a1 = a1' + 1 \\
&a2 = a2' + 2
\end{aligned}
\qquad (6.27)
$$

Here we are finding the coefficients (6.26) in the values *a0', a1' a2'*, etc. a0, a1, a2... are related to these coefficients by the addition of 0, 1, 2,... respectively, so it is merely a matter of adding these constants to the coefficients to find the exponents. The addition of +1, +2, ... to the final exponents in every iteration is optional, since it only effectively changes the starting point of adaptation. It is shown here to illustrate the connection between the updates and the terms in (6.26).

6.3.2.2 AN EXAMPLE ADAPTATION

The following example illustrates the adaptation of a two term model modelling a two term reference function. The reference function is $d=x^2+x^3$ and the model function is $y=x^a+x^b$. The optimum values are at either $a=2$, $b=3$ or $a=3$ $b=2$. The value of μ used was 0.02 (which provides a smooth adaptation in a reasonable number of steps). The contour surface shown the value of the mean squared error for various values of a and b, and the line shows the path taken by the adaptation algorithm, starting from (0,0).

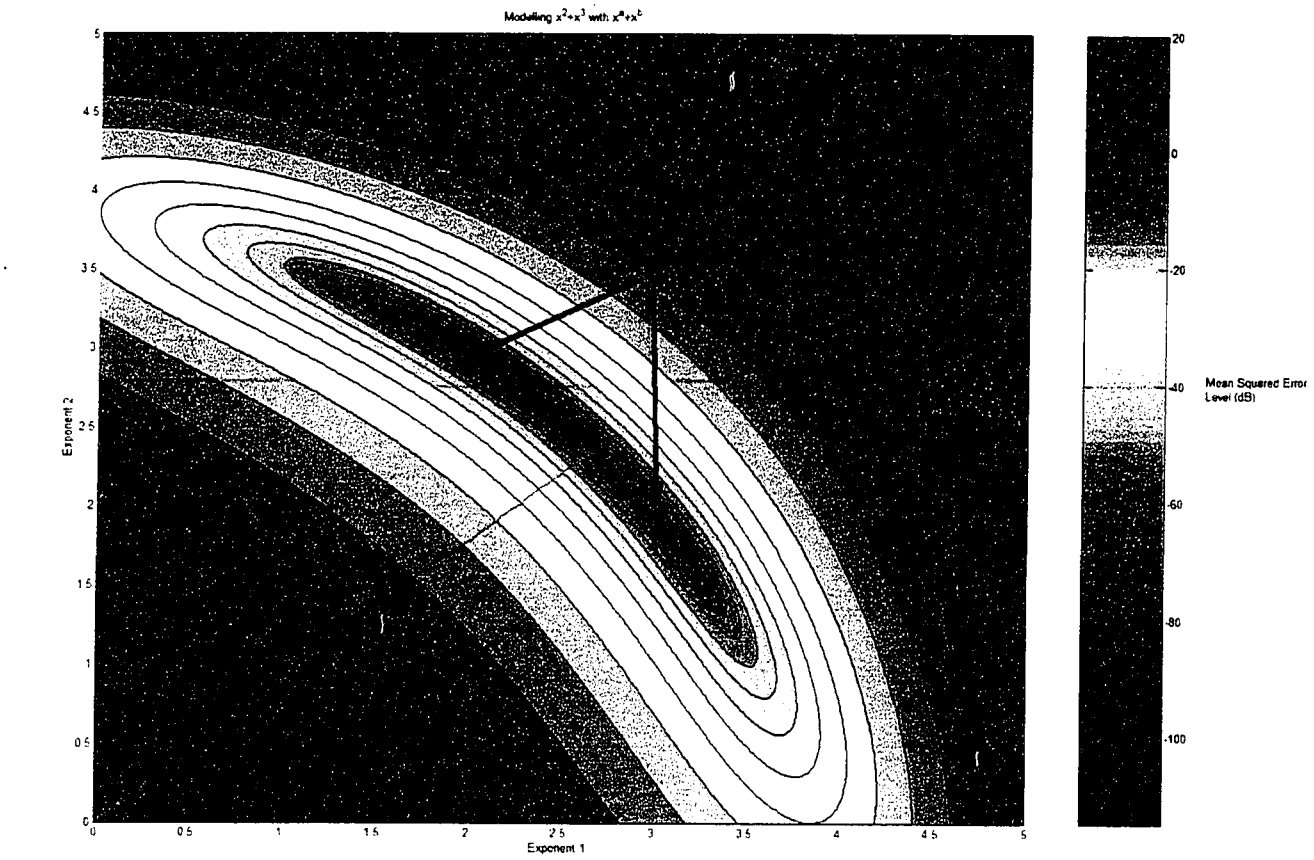


Figure 6.2 Error surface of a typical 2 term system

The location of the two minima is shown, and the adaptation algorithm finds one easily. Depending on which update is used, the algorithm will find one or the other minima. The extra factor of x in the algorithm will either increase or decrease the average step size (depending on whether the average value of x is smaller or larger than 1), causing the bias that results in one optimum or the other being chosen.

6.3.3 EXPONENT ADAPTATION FOR SYSTEMS WITH MEMORY

The previous analysis can easily be extended to system with memory by using the multi-variable Taylor series expansion of a function. Equation (6.28) shows the multivariable Taylor series representation of a function centred at $(c_0, c_1, \ldots, c_{mV})$.

$$f(x_0, x_1, ..., x_{m-1}) =$$

$$\sum_{i=0}^{\infty}\sum_{j=0}^{\infty}\cdots\sum_{k=0}^{\infty}\left[\frac{d^i y(c_0, c_1, ..., c_{m-1})}{dx_o}\times\frac{d^j y(c_0, c_1, ..., c_{m-1})}{dx_1}\times\ldots\times\frac{d^k y(c_0, c_1, ..., c_{m-1})}{dx_{m-1}}\times\frac{(x-c_0)^i(x-c_1)^i\ldots(x-c_{m-1})^k}{i!\,j!\ldots k!}\right] \tag{6.28}$$

The system we are using is shown in equation (6.29). It contains p terms with a memory of m.

$$y = [x_0^{a(0,0)} x_1^{a(1,0)} \ldots x_{m-1}^{a(m-1,0)}] + [x_0^{a(0,1)} x_1^{a(1,1)} \ldots x_{m-1}^{a(m-1,1)}] + \ldots + [x_0^{a(0,p-1)} x_1^{a(1,p-1)} \ldots x_{m-1}^{a(m-1,p-1)}] \tag{6.29}$$

From (6.28) we can see that the Taylor series representation of this function has coefficients of the term $(x_0\text{-}c_0)^i (x_1\text{-}c_1)^j \ldots (x_{m1}\text{-}c_{m1})^k$ of the form:

$$\frac{\frac{d^i y(c_0, c_1, ..., c_{m-1})}{dx_o}\times\frac{d^j y(c_0, c_1, ..., c_{m-1})}{dx_1}\times\ldots\times\frac{d^k y(c_0, c_1, ..., c_{m-1})}{dx_{m-1}}}{i!\,j!\ldots k!} \tag{6.30}$$

Like before, we choose the centre of the Taylor expansion to be (1,1,... 1), to simplify the derivatives. Using this centre, and considering only the first term of equation (6.29) we can write the first order Taylor terms in (6.28) as:

$$\frac{a(0,0)\times(x_o-1)}{2}$$
$$\frac{a(1,0)\times(x_1-1)}{2}$$
$$\ldots$$
$$\frac{a(m-1,0)\times(x_{m-1}-1)}{2} \tag{6.31}$$

We have an infinite number of terms in the Taylor series expansion, but the linear terms isolate the exponents of this system in such a way that makes them easy to determine. However, we have p terms, and finding the exponents of each variable in each term will require us to isolate each exponent into its own Taylor term. This is accomplished the same way as before, by factoring out an integer power of $x_0, x_1, \ldots, x_{m-1}$, from each term resulting in the modified Taylor series similar to equation (6.25). This shifted Taylor series representation of equation (6.29) is derived from equation (6.32).

$$y = [x_0^{a(0,0)} x_1^{a(1,0)} \ldots x_{m-1}^{a(m-1,0)}] + x_0 x_1 \ldots x_{m-1} [x_0^{a(0,1)-1} x_1^{a(1,1)-1} \ldots x_{m-1}^{a(m-1,1)-1}] + \ldots$$
$$+ x_0^{p-1} x_1^{p-1} \ldots x_{m-1}^{p-1} [x_0^{a(0,p-1)-(p-1)} x_1^{a(1,p-1)-(p-1)} \ldots x_{m-1}^{a(m-1,p-1)-(p-1)}] \tag{6.32}$$

The resulting Taylor series contains the following terms that isolate each of the exponents:

$$(a(i,j)-j)x^j(x-1) \tag{6.33}$$

Where j is the number of the term (j=0..p-1) numbered arbitrarily, i is the index of the variable, the memory (i=0..m-1). With this term, it is easy to see the LMS update equations of $a_{i,j}$, which is shown in equation (6.34).

$$
\begin{aligned}
&a'(i,j) = a'(i,j + \mu e x_i^{\,j}(x_i - 1) \\
&and \\
&a(i,j) = a'(i,j) + j
\end{aligned}
\tag{6.34}
$$

6.3.4 HANDLING COEFFICIENTS

Thus far, the systems studied were incomplete, lacking coefficients other than 1 for the terms used. This section completes the derivation of the adaptation algorithm by including a system with memory, multiple terms and coefficients, as shown in equation (6.35).

$$
\begin{aligned}
y = h_0[x_0^{\,a(0,0)} x_1^{\,a(1,0)} \ldots x_{m-1}^{\,a(m-1,0)}] + h_1[x_0^{\,a(0,1)} x_1^{\,a(1,1)} \ldots x_{m-1}^{\,a(m-1,1)}] + \ldots \\
+ h_{p-1}[x_0^{\,a(0,p-1)} x_1^{\,a(1,p-1)} \ldots x_{m-1}^{\,a(m-1,p-1)}]
\end{aligned}
\tag{6.35}
$$

We can obtain infinitely many separate Taylor series terms for each exponent, but now they will be weighted by the coefficient. Even if we can determine h_j, simply dividing it out of the exponent is not an ideal method of adapting the exponents, since problems arise when the coefficient comes close to zero.

It so happens that we can easily find a Taylor term that contains only h_j as the coefficient, namely x^j, where as before j is the term number. This simplifies finding the exponent. If we assume the coefficient is constant, we can ignore the coefficient's magnitude and only the sign will affect the exponent's update. This simplification ensures the correct direction of each update is preserved, while solving conditioning problems that occur when the coefficient is close to zero. This is similar to the signed LMS algorithms.

We can now easily write the LMS update equations for the series of exponents and coefficients as shown in equation (6.36).

$$h_j = h_j + \mu e x^j$$
$$a'(i,j) = a'(i,j) + \mu e x_i^{\,j}(x_i - 1)sign(h_j) \tag{6.36}$$

6.3.5 EXPONENT ADAPTATION PROBLEMS

Exponent adaptation suffers from three main problems:

1) Infinite exponents/zero valued coefficients

2) Convergence speed

3) Multiple minima

Of these problems, only the last is of major concern as we will see.

6.3.5.1 INFINITE EXPONENTS/ZERO VALUED COEFFICIENTS

The modelling functions used in this algorithm can generate infinite valued exponents as valid values. For example, if the input is distributed between 0 and 1, and it is raised to an infinite exponent, the output is 0, which is a stable system. Alternately if the input is always greater than 1, negative infinity is a valid exponent. In some implementations it may be necessary to clamp exponent values at a sufficiently high value.

Related to this problem, if the exponent is infinite or the coefficient is zero, the system behaves identically (zero output). Hence aside from the multiple equivalent solutions mentioned earlier, the model can have equivalent solutions if the coefficient is zero (in which case any value of the exponent will do), or if the exponent is infinite (in which case any value of the coefficient will do). This problem only occurs in over modelling situations, when the effect of extraneous terms is zeroed (i.e. the algorithm can choose to zero a useless term by multiplying it by zero or by making the exponent negative/positive infinity). In this case, it may be necessary to clamp the coefficient if the exponent is infinite, to prevent it from overflowing.

6.3.5.2 CONVERGENCE SPEED

The next problem has to do with the speed of convergence. The exponent adaptation algorithm is based on the LMS algorithm. It is well known that LMS suffers from performance problems when the input taps are correlated. The exponent adaptation algorithm generates LMS taps in such a way that each tap is highly correlated with the other taps. In fact, as more taps are added to the exponent adaptation algorithm, the correlation between taps increases. This is due to the fact that each tap is generated by multiplying the previous tap by x, which is roughly equivalent to generating taps of orders 1, 2, 3 ... n. As demonstrated in section 4.1, higher order taps are more highly correlated with each other. Various modified LMS algorithms exist to increase convergence speed, some of which could be used in this application [9].

6.3.5.3 MULTIPLE MINIMA

Another problem occurs because of the existence of multiple minima. If the modelling function shown in (6.35) is used to model a function with the same form and the same number of terms, n, then there will be n! minima for the exponents given fixed coefficient values. If all coefficients have the same value, these minima will achieve the same minimum value, but if the coefficients are all different, only one minimum will be the global minimum. This can be illustrated by the following function:

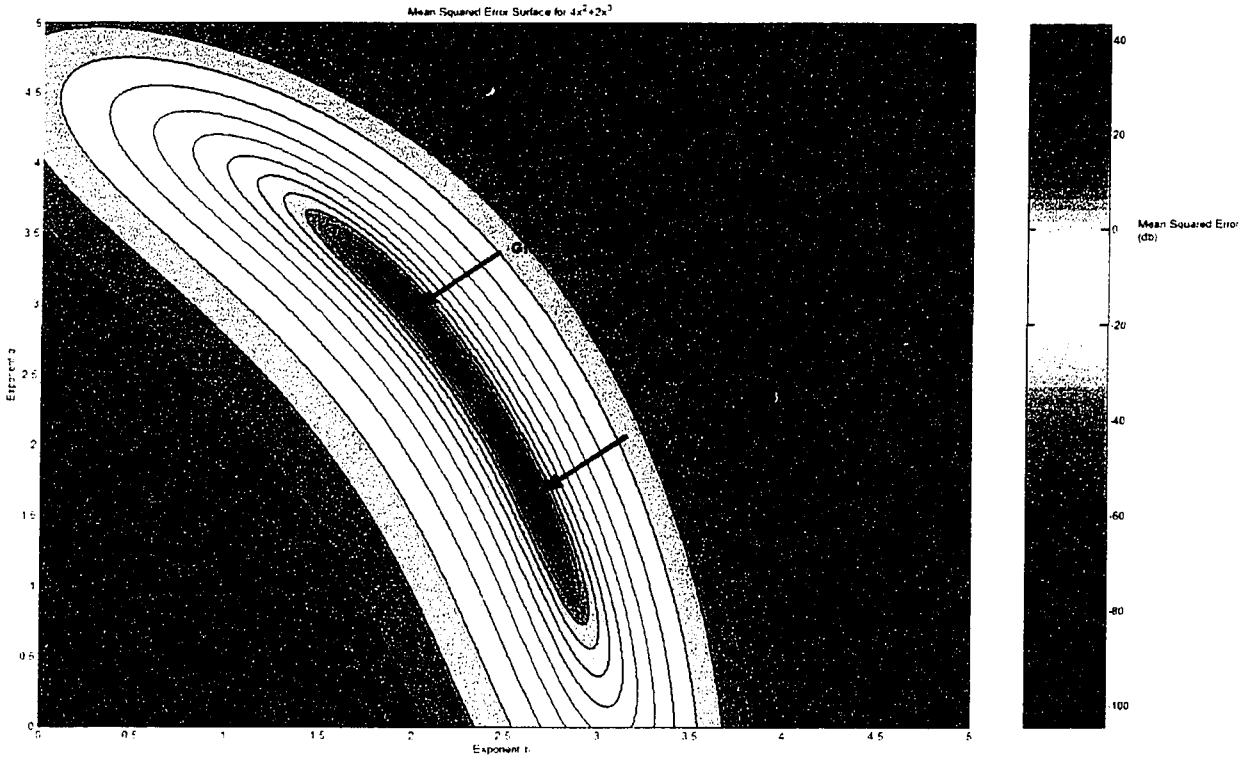
$$2x^3 + 4x^2 \tag{6.37}$$

When we model the system with (6.38), there is obviously only one global minimum, a=3 and b=2

$$2x^a + 4x^b \tag{6.38}$$

However, a local minimum exists for this function, as shown by the following error surface (error surface computed with a uniform input between 0.5 and 1.5):

Figure 6.3 Error Surface



The additional minimum at (2.66, 1.66) offers a trap into which a gradient search algorithm may fall. The above example assumes that the coefficients are constant and only the exponents are changing, which may be close to the case if the adaptation of coefficients occurs slower than the adaptation of exponents.

The problem of multiple minima is clearly a fact for all nontrivial nonlinear optimization problems. In this case, there will be n! minima, where n is the number of terms in the modelling function.

6.3.6 RESILIENCE TO MULTIPLICATIVE NOISE

Since this transform changes multiplication into addition, this algorithm shares the standard noise resilience properties of the adaptation method which apply to additive noise.

In the presence of multiplicative noise, the output of the term is

$$h * x[n]^a * x[n-1]^b ... * r \tag{6.39}$$

where r is the multiplicative noise. After transforming with the logarithm, the output becomes:

$$\begin{aligned} &a * \log(x[n]) + b * \log(x[n-1]) + ... + \log(r) + \log(h) = \\ &a * \log(x[n]) + b * \log(x[n-1]) + ... + r' + \log(h) \end{aligned} \tag{6.40}$$

Under the conditions of the adaptation algorithm, the filter will converge in the presence of multiplicative noise; however the coefficient will be weighted by the mean of the multiplicative noise. If the mean of the multiplicative noise is zero, the filter will not converge properly. For any multiplicative noise, the exponents are not affected, and for multiplicative noise of mean 1, the coefficient is not affected. An example is presented later on in this chapter.

6.3.7 PROBLEMS WITH ADDITIVE NOISE

In the presence of additive noise, the additive noise will create an offset which affects the final values of the exponents. For example, if we were modeling the system $x[n] + r$, where r was additive noise, we would expect to obtain an exponent of 1 for $x[n]$. However the exponent we would obtain would in fact be the optimum exponent which models $x[n] + E[r]$, where $E[r]$ is the expected value of r. If $E[r] = 0$, or zero mean noise, additive noise will not affect our final values, but for any other noise, the exponent will change.

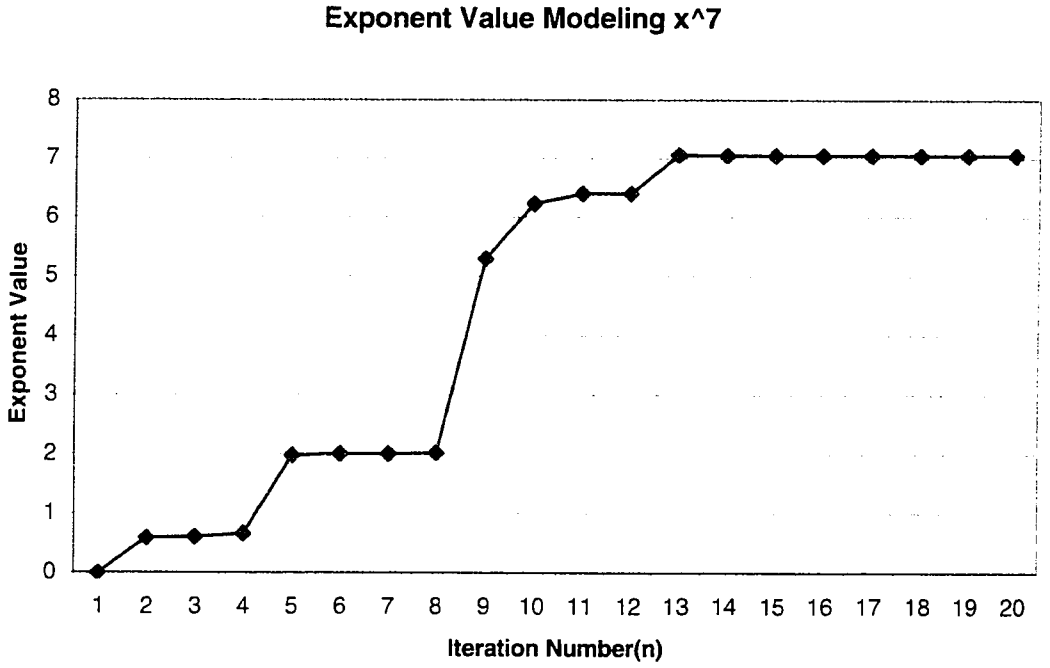
6.4 SIMULATION RESULTS

As a demonstration of this technique, a number of simulations have been run. These simulations demonstrate the convergence properties and limitations of this technique.

6.4.1 SIMULATION 1 - SINGLE MEMORYLESS TERM, NO COEFFICIENT

In this simulation, we try the simplest system and obtain the correct result. We model this memoryless system $x[n]^7$ using the general memoryless system $x[n]^a$. After adaptation, the value of a should be 7. The value of the coefficient plotted against the iteration number is shown below, given a starting point of $a=0$, using the LMS algorithm.

Figure 6.4 Convergence of a simple single product memoryless system

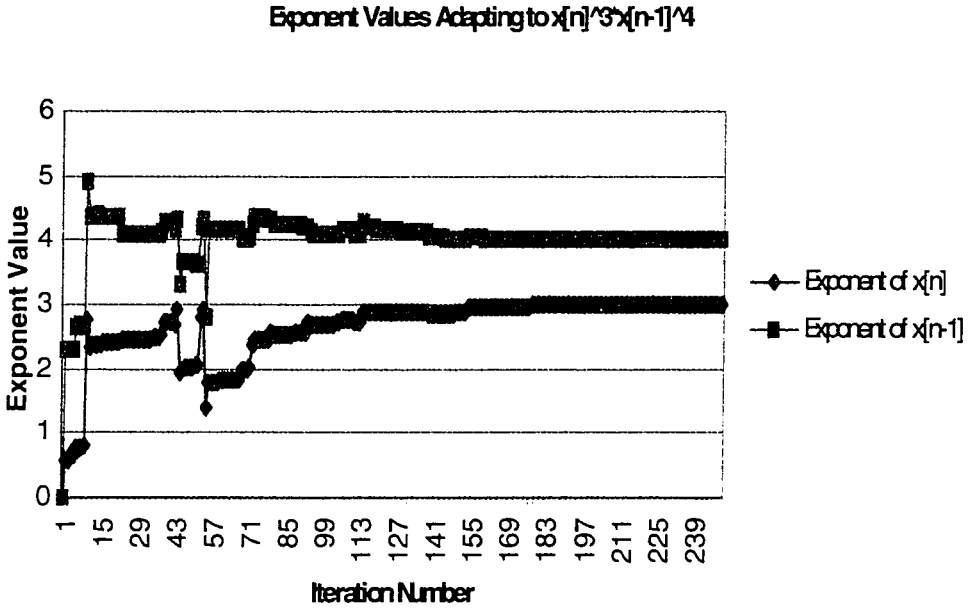


As can be seen, the correct result is obtained in a reasonable number of steps.

6.4.2 SIMULATION 2 - SINGLE TERM WITH MEMORY, NO COEFFICIENT

We now attempt to determine the exponents of a system with a memory of two samples. We use the general term $x[n]^a x[n-1]^b$ as our modelling function, and $x[n]^3 x[n-1]^4$ as the function to be modelled. After LMS converges, the values of a and b are *3* and *4* respectively. The plot below shows the values of a and b as they change with iteration number, for an uncorrelated input.

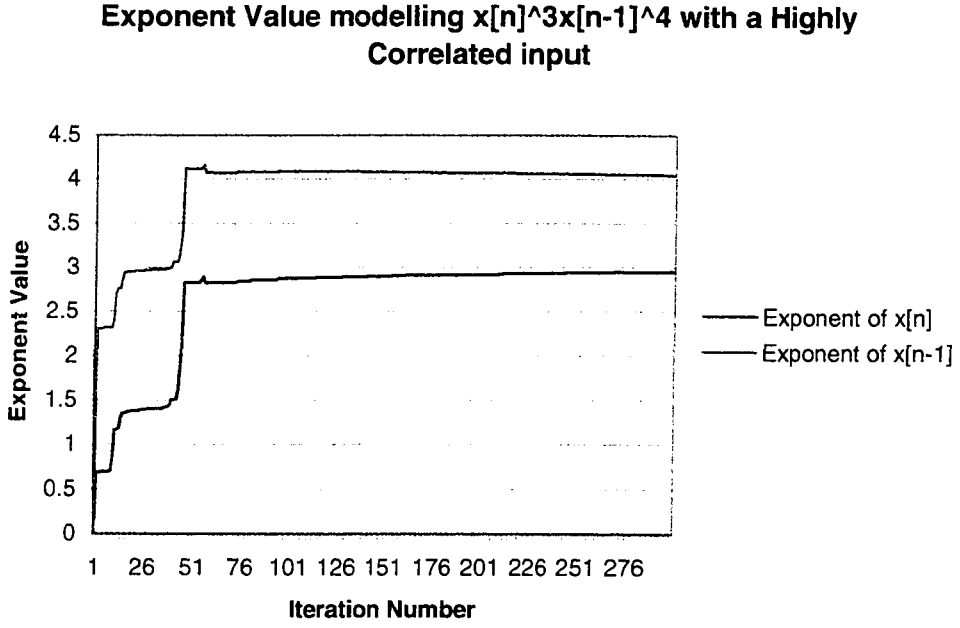
Figure 6.5 Simulation with uncorrelated input



The jumps in the graph are due to the large step size, a smaller step size may be appropriate if these jumps are not desirable.

The next plot shows the exponents a and b when the input is highly correlated.

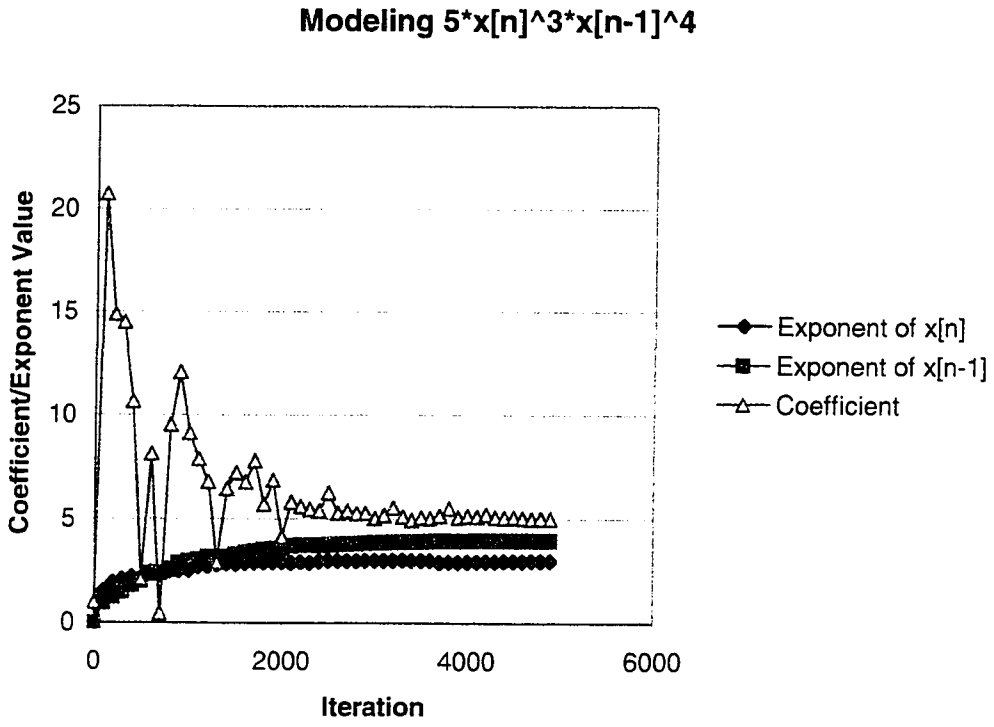
Figure 6.6 Simulation with correlated input



6.4.3 SIMULATION 3 - SINGLE TERM WITH MEMORY AND A COEFFICIENT

We now include a coefficient in addition to exponents. We use the general system $h*x[n]^a*x[n-1]^b$ to model the system $5*x[n]^3*x[n-1]^4$, and expect $h=5$, $a=3$, $b=4$ as the results. The values of h, a, and b are shown in the plot below. The input to this system is uncorrelated uniformly distributed noise. The parameters converge to their values as expected.

Figure 6.7 Simulation of a two input system with a coefficient

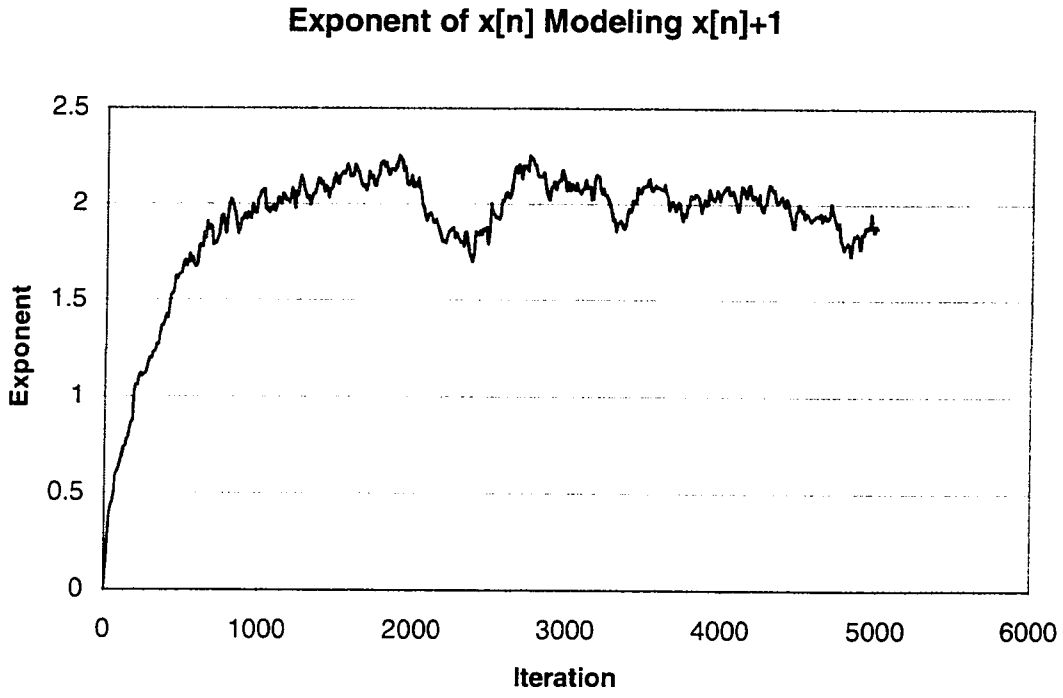


Again, the jumps could be decreased by making the step size smaller.

6.4.4 SIMULATION 4 - UNDERMODELING

This simulation demonstrates the algorithm's behaviour when under-modeling a system. This unknown system, $x[n]+1$, cannot be represented by the general Volterra term, so the adaptation should find the best general Volterra term which most closely matches it. We use the system $x[n]^a$ as the general modeling term. The plot below shows the exponent a against the iteration number.

Figure 6.8 Simulating a under-modeled system



The final misadjustment can be reduced by using a smaller step size, or alternately using a variable step size LMS algorithm.

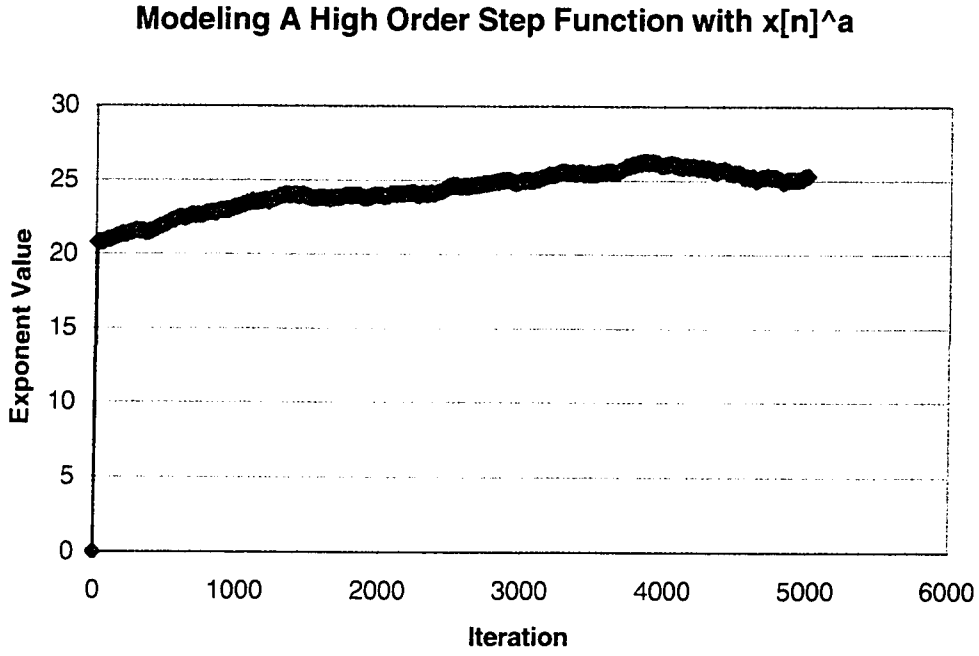
6.4.5 SIMULATION 5 - MODELING THE STEP FUNCTION

In this simulation, we model the memoryless function

$$f(x) = \begin{cases} 0 : x < 0.95 \\ 1 : x \geq 0.95 \end{cases} \tag{6.41}$$

with a random input uniformly distributed between *0* and *1*. This system is a high order system that cannot be completely modeled using the general Volterra term. We use the general term $x[n]^a$ to model this function. The value of the exponent is plotted against the iteration number in the plot.

Figure 6.9 Simulating a high order step function

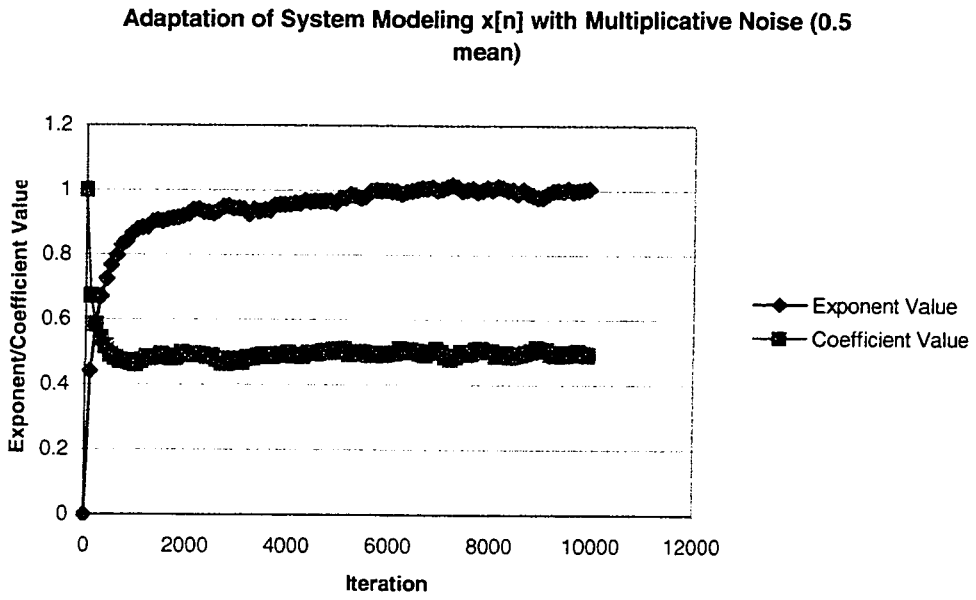


The algorithm identifies the order of the system, with minor misadjustment due to a large step size.

6.4.6 SIMULATION 6 - MODELING IN THE PRESENCE OF MULTIPLICATIVE NOISE

The system modeled is $x[n]*r$, where r is uniformly distributed noise in the range of 0 to 1 (0.5 mean value). We adapt the exponent and coefficient of $h * x[n]^a$, and expect the result to be $a=1$ and $h = 0.5$ after adaptation.

Figure 6.10 Modeling in the presence of multiplicative noise

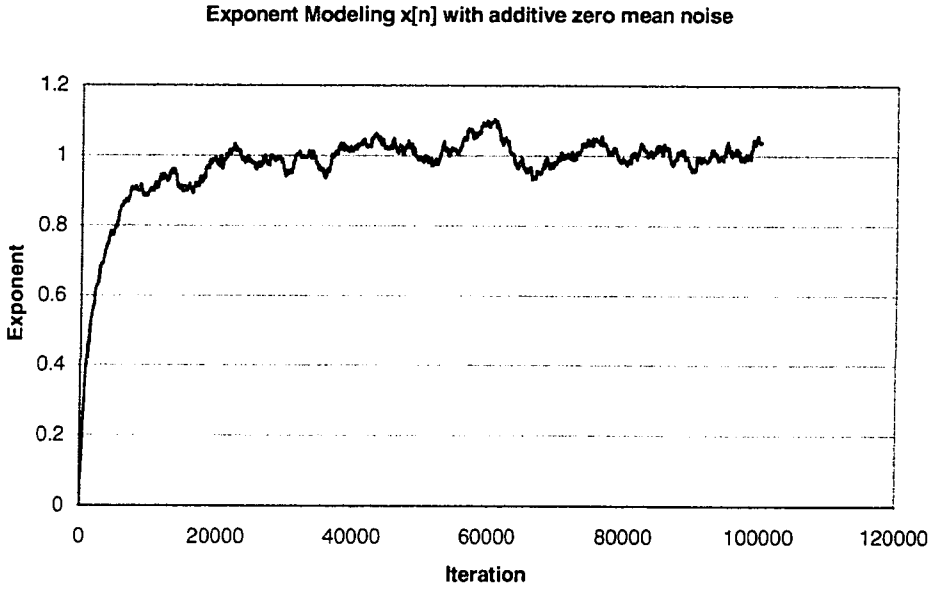


As expected, the algorithm determined the exponent correctly and determined the coefficient weighted by the mean of the multiplicative noise.

6.4.7 SIMULATION 7 - MODELING IN THE PRESENCE OF ADDITIVE NOISE

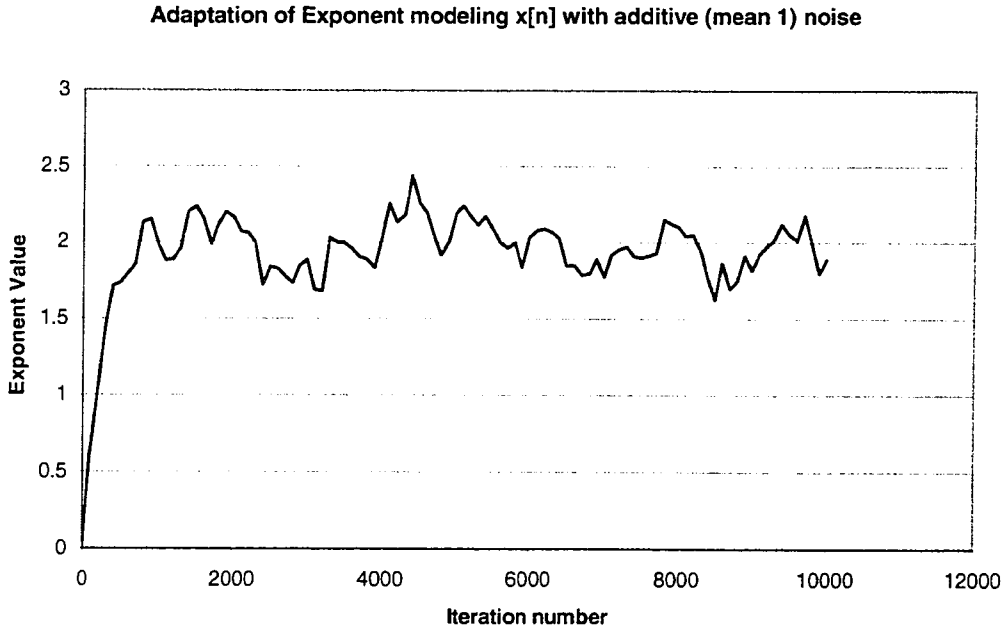
We now model the system $x[n] + r$, where r is additive noise. In the first plot, we show the exponent a when r is uniformly distributed between -1 and 1 (0 mean). In the second case, r is distributed between 0 and 2 (mean value of 1). Note that in the second case, the exponent is similar to the exponent obtained in simulation 4, where we were modeling $x[n]+1$, since E[r] = 1.

Figure 6.11 Modeling in the presence of additive noise (zero mean)



The exponent adapts towards 1, as expected.

Figure 6.12 Modeling in the presence of additive noise (mean 1)



This result is also expected. The exponent adapts towards 2, which is the same result obtained in simulation 4. In fact, the generation of the random noise of mean *1* is accomplished by the following formula: $r' = 1.0 + r$, where r' is the noise with mean of *1*, and r is zero mean noise. As shown in the first part of this experiment, the addition of zero mean noise does not affect the adaptation, so we are left with the addition of the constant *1* (mean of noise) to the output of the system, resulting in an identical results as simulation 4.

6.5 CONCLUSION

By allowing exponents to take on real values, the problem of finding the optimum set of terms is simplified into adaptively finding the optimum set of exponents. The proposed approach involves transforming the modelling function into a Taylor series, using both the Taylor series representation and the exponential representation at convenient points within the algorithm to find optimum exponent values. We demonstrate the performance of this algorithm under many different cases and show that it is capable of identifying the correct exponent values, and hence the optimum terms. This algorithm is well suited for high order non-linearities, so comparison with existing applications is hampered by the fact that most truncated Volterra filters are used on low order systems.

It should be noted that implementation of real exponent is possible on most modern processors and is even supported in hardware in others, such as the Intel x86 family. It should also be emphasized that rounding or truncating exponents back to integer values will not necessarily yield optimum results. Take for example an error surface that looks like Figure 6.2. The oblong 'trough' could easily pass between two integer values but intersect a

third integer further away than the two closer integers. In this case the third integer would achieve a lower error than the other two integers. In this case, truncating the exponents back to integer values undermines the optimization, and yields a non-optimized system.

CHAPTER 7-CONCLUSION

This thesis has developed numerous methods for computing and adapting compact representations of nonlinear Volterra systems. The need to represent high order nonlinearities without the drawback of excessive parameterization has been addressed.

In particular, many of the benefits of Volterra filters, including non-linear modelling, ease of adaptation, ease of evaluation, and generality have been preserved, while reducing the computational requirements needed by typical truncated Volterra filters. This has been accomplished by two broad classes of algorithms introduced in this thesis: the term/set switching algorithms of Chapter 5 and the exponent adaptation methods of Chapter 6.

The term/set switching algorithms attempt to find the best set of Volterra terms taken from a larger, less restrictive set of terms. These algorithms include the single term switching algorithms and the multiple set searching algorithm.

The exponent adaptation algorithm first extends the class of Volterra filters to a superset class of functions of the same form, but containing real-valued exponents rather than strictly whole number exponents. This class can be represented by a Taylor series which is itself a polynomial with whole number exponents; however the Taylor series representation has an infinite number of terms. Using these two representations, the infinite Taylor series and the finite real-numbered exponential series, the algorithm can update the real-numbered exponent by updating a coefficient of the Taylor series. The Taylor series update uses the well-known LMS algorithm. The use of real exponents does not place a

significant restriction on real time use of this algorithm, since some processors already provide low level support for this operation.

The benefits of the new algorithms and derivations may manifest in the complementary objectives:

1) Reduction of the complexity of a Volterra series model so a lower complexity truncated model can perform near the performance of the full complexity model.

2) Reduction of the computational requirements of a Volterra series adaptive model so a truncated model of higher order can be used in place of a full complexity lower order model to achieve better modelling than the full complexity lower order model.

The primary drawbacks of these methods are often excessive computational complexity, non-optimality, and discontinuous error performance in the case of the term switching algorithms (Chapter 5) and the need to compute real exponents (hence necessitating positive inputs) in the case of the exponent adaptation algorithms (Chapter 6). In many respects, the algorithms in Chapter 5 and those in Chapter 6 fulfil complementary needs.

7.1 SUGGESTIONS FOR FURTHER RESEARCH

The problem of optimally truncating Volterra filters has not been completely addressed by this thesis, since the term switching algorithms of Chapter 5 do not guarantee

optimality and the exponent adaptation algorithms of Chapter 6 offer optimality only for real exponents, not those constrained to be integers (as is the case for pure Volterra filters). Hence one unresolved issue is how best to convert the real valued exponents back to integer values. It is known that merely truncating or rounding the exponents is not sufficient.

Other issues not addressed by this thesis involve a more in-depth analysis of the exponent adaptation algorithms, including bounds for the adaptation specific parameters such as LMS step size, adaptation rate, residual error, effect of residual exponent error on over system error, and other adaptation related problems.

The final issue is that of local versus global convergence. Analysis of the mean squared error surface shows that n! minima will exist for a system with n terms. If the coefficients of these terms are not all equal, the local minima will exist.

REFERENCES

[1] "IA-32 Intel Architecture Software Developer's Manual Volume 2: Instruction Set Reference", Intel Corporation, Mt. Prospect, IL, 2001.

[2] Henry Abarbanel, "Obtaining Order in a World of Chaos— Time-Domain Analysis of Nonlinear and Chaotic Signals", *IEEE Signal Processing Magazine*, vol. 15, no. 3, May 1998.

[3] A. Carini, V. J. Mathews and G. L. Sicuranza, "Equalization of Recursive Polynomial Systems," *IEEE Signal Processing Letters*, Vol. 6, No. 12, December 1999.

[4] Carravetta F., A. Germani and M. Raimondi, "Polynomial Filtering for Linear Discrete Time Non-Gaussian Systems", *SIAM Journal on Control and Optimization*, Vol. 34, no. 5, pp. 1666--1690, September, 1996.

[5] De Santis A., A. Germani and M. Raimondi, "Optimal Recursive Second Order Polynomial Estimation for Linear Discrete Time Non-Gaussian Systems", *IEEE Trans. on Automatic Control*, Vol. 40, No. 7, July 1995.

[6] Daniel Graupe, "An Output-Whitening Approach to Adaptive Active Noise Cancellation", *IEEE Transactions on Circuits and Systems*, vol. 32, no. 11, November 1991.

[7] Ajit Kumar Chaturvedi and Govind Sharma, "A New Family of Concurrent Algorithms for Adaptive Volterra and Linear Filters", *IEEE Transactions on Signal Processing*, vol. 47, no. 9, September 1999.

[8] E. E. Kuruoglu et al., "Least lp-Norm Impulsive Noise Cancellation with Polynomial Filters", *Signal Processing*, vol. 69, no. 1, 1998.

[9] Jae S. Lim and Alan V. Oppenheim, "Advanced Topics in Signal Processing", Prentice Hall, Englewood Cliff, New Jersey, 1988.

[10] L. Li and V. J. Mathews, "Frequency-Domain Realizations of Adaptive Parallel-Cascade Quadratic Filters", *IEEE Trans. Circuits and Systems II -- Analog and Digital Signal Processing*, Vol. 46, No. 4, April 1999.

[11] V. J. Mathews and Giovanni L. Sicuranza, "Polynomial Signal Processing", John Wiley & Sons, New York, NY., 2001.

[12] V. J. Mathews, "Adaptive polynomial filters," *IEEE Signal Processing Magazine*, vol. 8, no. 3, July 1991.

[13] V. John Mathews, "Adaptive Volterra Filters Using Orthogonal Structures", IEEE *Signal Processing Letters*, vol. 3, no. 12, December 1996.

[14] Mayer, "Identification of multivariate Volterra series using GA", *Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applications*, 1996.

[15] Enzo Mumolo and Albero Carini, "On the Stability of Discrete Time Recursive Volterra Filters", *IEEE Signal Processing Letters*, vol. 6, no. 9, September 1999.

[16] Robert D. Nowak and Barry D. Van Veen, "Volterra Filter Equalization: A Fixed Point Approach", *IEEE Transactions on Signal Processing*, vol. 45, no. 2, February 1997.

[17] Robert D. Nowak and Richard G. Baraniuk, "Wavelet-Based Transformations for Nonlinear Signal Processing", *IEEE Transactions on Signal Processing*, vol. 47, no. 7, July 1999.

[18] Robert D. Nowak, "Penalized Least Squares Estimation of Volterra Filters and Higher Order Statistics", *IEEE Transactions on Signal Processing*, vol. 46, no.2, February 1998.

[19] Thomas M. Panicker and V. J. Mathews, "Parallel-Cascade Realizations and Approximations of Truncated Volterra Systems", *IEEE Transactions on Signal Processing*, vol. 46, no. 10, October 1998.

[20] Thomas M. Panicker, V. John Mathews, and Giovanni L. Sicuranza, "Adaptive Parallel-Cascade Truncated Volterra Filters", *IEEE Transactions on Signal Processing*, vol. 46, no. 10, October 1998.

[21] Edwin Purcell and Dale Varberg, "Calculus with Analytic Geometry", Prentice-Hall Inc., Englewood Cliffs, N. J., 1987.

[22] Ramesh A. Gopinath and C. Sidney Burrus, "On Upsampling, Downsampling, and Rational Sampling Rate Filter Banks", *IEEE Transactions on Signal Processing*, vol. 42, no. 4, April 1994.

[23] Gil M. Raz and Barry D. Van Veen, "Baseband Volterra Filters for Implementing Carrier Based Nonlinearities", *IEEE Transactions on Signal Processing*, vol. 46, no. 1, January 1998.

[24] Arthur J. Redfern and G. Tong Zhou, "A Root Method for Volterra System Equalization", *IEEE Signal Processing Letters*, vol. 5, no. 11, November 1998.

[25] Riitta Niemistö, Ioan Tabus and Jaakko Astola, "A Fast Algorithm for Adaptive Polynomial Filtering", *Signal Processing*, vol. 70, 1998.

[26] Raghuveer M. Rao and Ajit S. Bopardikar, "Wavelet Transforms - Introduction to Theory and Applications", Addison Wesley Longman, Inc. Reading, MA, 1998.

[27] Kyle Siegrist, "Virtual Laboratories in Probability and Statistics", http://www.math.uah.edu/stat, September 21, 2001.

[28] A. Stenger et al., "Nonlinear Acoustic Echo Cancellation with 2nd Order Adaptive Volterra Filters", *Proc. Int. Conf. on Acoustics, Speech & Signal Processing (ICASSP)*, IEEE, Phoenix, USA, 1999.

[29] Li Tan and Jean Jiang, "An Adaptive Technique for Modeling Second-Order Volterra Systems with Sparse Kernels", *IEEE Transactions on Circuits and System II— Analog and Digital Signal Processing*, vol. 45, no. 12, December 1998.

[30] Eric Weistein, "Eric Weistein's World of Mathematics", http://mathworld.wolfram.com/.

[31] Bernard Widrow and Samuel Stearns, "Adaptive Signal Processing", Prentice-Hall Inc., Englewood Cliffs, N. J., 1985.