EXPLOITING SPARSITY IN ADAPTIVE FILTERS

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Richard K. Martin

May 2001

ABSTRACT

This thesis studies a class of algorithms called Natural Gradient (NG) algorithms, and their approximations, known as ANG algorithms. The LMS algorithm is derived within the NG framework, and a family of LMS variants that exploit sparsity is derived. This procedure is repeated for other algorithms and their families, namely the Constant Modulus Algorithm (CMA), Decision-Directed LMS (DD-LMS), and algorithms using an ARMA parameterization.

Mean squared error (MSE) analysis, stability analysis, and convergence analysis of the family of sparse LMS algorithms are provided, and it is shown that if the system is sparse, then the new algorithms will often converge faster for a given total asymptotic MSE, although the choice of initialization is important. Simulations are provided to confirm the analysis. In addition, Bayesian priors matching the statistics of a database of real channels are given, and algorithms are derived that exploit these priors. Simulations using measured channels are used to show a realistic application of these algorithms.

Biographical Sketch

Rick Martin was born in Great Falls, Montana on May 20, 1976. He spent his youth in Ohio, Colorado, North Carolina, New Mexico, Alabama, and Maryland. His undergraduate years were spent at the University of Maryland, wherein he received degrees in Physics and Electrical Engineering. After graduation, he spent a summer working at Telogy Networks, where he worked on echo cancellation. However, industry life did not appeal to him, and he moved on to the graduate program at Cornell University. His current leisure interests are ballroom and swing dancing, hiking, and reading science fiction and fantasy novels. His career goal is to become a professor.

To my family...

Acknowledgements

This thesis could not have been completed without the invaluable support of my advisor, Rick Johnson, and my coauthors on the associated journal paper, Bill Sethares and Bob Williamson. My fellow students, Jaiganesh Balakrishnan and Wonzoo Chung, also provided many helpful comments. I would like to thank Rod Kennedy, Brian Hart, and the rest of the group at ANU for allowing me to visit and work on my thesis. This work was supported financially by Fox Digital and NSF Grant EEC-98 72436. A special thanks goes to my family for their support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Transmission channels are often sparse. This can mean several things. Most researchers take this to mean that if the channel is represented as a finite impulse response (FIR) filter, then there will be a few large taps, separated by many much smaller taps, as exemplified by Figure 1.1. A more inclusive definition of sparsity, which includes the former definition, is that a given tap in a sparse channel has a high probability of being small and a low probability of being high.

In this thesis, the former definition of sparsity well be referred to as "conventional sparsity" or "traditional sparsity", and the second, broader definition will be referred to as "probabilistic sparsity"; though unless otherwise noted, the word sparsity by itself will be taken to mean the latter definition.

Researchers in fields such as wireless communication, echo cancellation, underwater accoustic communication have noted that transmission channels often exhibit conventional sparsity. Typical algorithms such as the Least Mean Square (LMS) algorithm, the Constant Modulus Algorithm (CMA), and the Decision Directed (DD) algorithm do not exploit this a priori information. Many people have tried to modify existing algorithms or create new ones, with the intent to reduce the

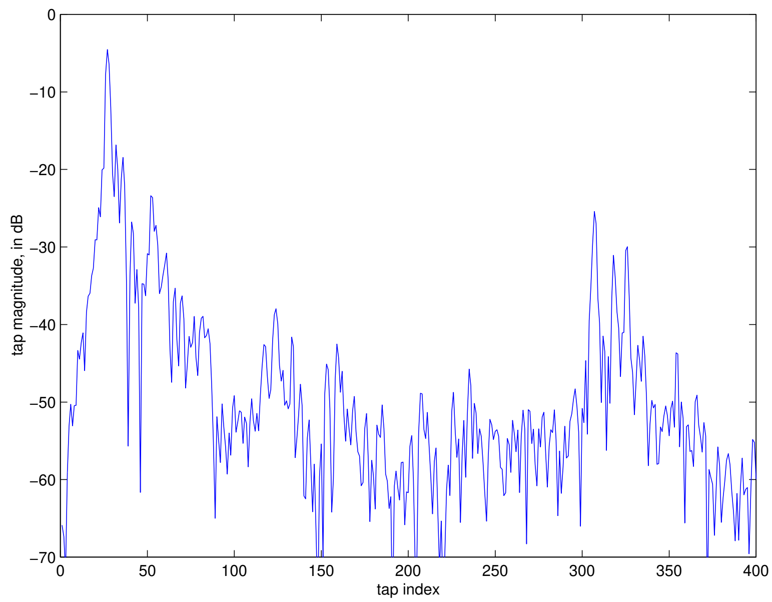

Figure 1.1: Example of a conventionally sparse channel.

complexity of their algorithms or to improve their performance. A summary of this work can be found in Chapter 2.

The most important point to note regarding this previous work is that typical approaches to exploiting sparsity are motivated by complexity reduction (at the expense of a small performance loss), which is often accomplished by only updating a subset of the channel model or equalizer taps [2], [6], [17]. In constrast, the Exponentiated Gradient (EG) algorithm [26] has recently been shown to have better performance than typical gradient methods when the target weight vector is sparse [22], [30], though it does not decrease the required complexity. This is the philosophy pursued in this thesis.

This thesis rederives the EG algorithm as a 'natural' gradient descent of a (modified) cost function, and interprets the improved performance under sparsity in terms of a reparameterization of the underlying linear process. Then similar algorithms are derived that make use of the signal restoration property of CMA and the finite alphabet structure of DD. Moreover, ARMA model versions are feasible. Like the EG algorithm, the new algorithms can be readily tailored to exploit known sparsities in the target weight vector.

Before proceeding any further, the system model and the notation should be specified. Two applications that may benefit from exploitation of sparsity are system identification and channel equalization. System identification algorithms attempt to model an unknown system as an FIR filter[1], as in Figure 1.2. Equalization deals with recovering the input to a transmission channel from its output. This is depicted in Figure 1.3.

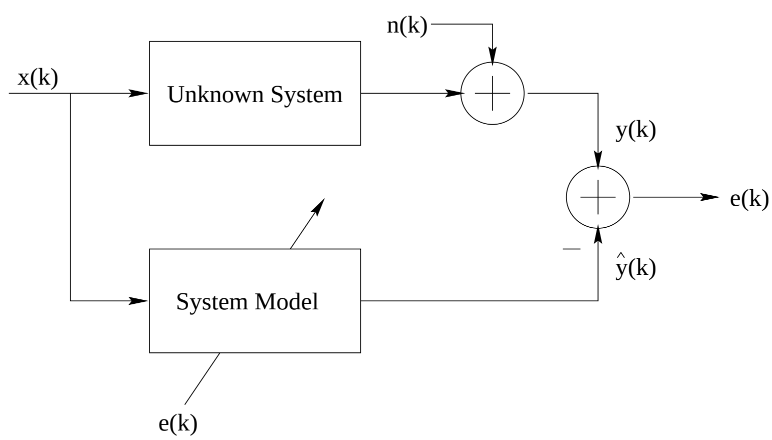

Figure 1.2: Model for system identification.

In both of these applications, we assume that the channel can be modeled by an FIR filter, with weights $c_1, ..., c_m$. The regressor $\mathbf{x}_k$ is defined to be a vector of the past m inputs at time k, such that

$$\mathbf{x}_k = [x(k), ..., x(k-m+1)], \tag{1.1}$$

[1]Other models may be used, but in this work we focus on the FIR model.

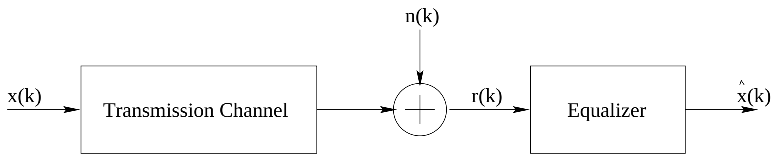

Figure 1.3: Generic equalizer structure.

where $x(k)$ is the input to the channel at time k. The output of the channel is

$$y(k) = \sum_{i=1}^{m} c_k^i \; x_k^i + n(k), \tag{1.2}$$

where x_k^i refers to the i^{th} element of the vector $\mathbf{x}_k$, and n(k) is additive noise (generally white and Gaussian). For an identifcation application, the system model will also be an FIR filter, with weights $w_1, ..., w_n$. The output of the model is given by

$$\hat{y}(k) = \sum_{i=1}^{n} w_k^i \; x_k^i. \tag{1.3}$$

We will also consider an equalization setting. Our examples will use baud spaced equalizers, which can also be modeled as FIR filters. Thus, the output of the equalizer is given by

$$\hat{x}(k) = \sum_{i=1}^{n} w_k^i \; r_k^i, \tag{1.4}$$

where $\mathbf{r}_k$ is the regressor of the recieved signal $r(k)$, formed analagourly to Equation (1.1).

The remainder of this thesis is organized as follows. Chapter 2 reviews existing methods for exploiting sparsity. Chapter 3 gives a motivating example to intuitively show the benefits of our approach to sparse algorithms. Chapter 4 provides a theoretical analysis of the excess MSE and convergence properties of the general class of algorithms introduced in the previous chapter. Chapter 5 gives a framework for rederiving these algorithms to incorporate *a priori* knowledge of sparsity, and Chapter 6 applies this framework to develop a wide variety of algorithms. Chapters 7 and 8 demonstrate the performance of these algorithms for synthetic and real data, respectively, and Chapter 9 concludes.

Chapter 2

Literature Review

Sparsity, in the context of adaptive filters, has been receiving attention since the early 1980's. Sparse channels are regularly encountered in applications such as echo cancellation, underwater acoustic communication, and wireless communication. In this section we discuss the contributions of researchers in all of these fields.

The proposed techniques include updates for all of the filter taps; some of the taps; or all of the taps, but not all at once. The techniques are sometimes applied to traditional structures, and sometimes new structures are created specifically to exploit sparsity. Some use high complexity techniques, such as computing a channel estimate to reduce the burden on the equalizer; and some of the techniques are designed with reduced complexity in mind. These maddeningly varied techniques are categorized and discussed below, and an overview is given by Figure 2.1.

2.1 Alternative Equalizer Structures

A modified DFE structure was first proposed by Aman and Tretter [3], and generalized in [20], in which some or all of the post-cursor is cancelled *before* the forward

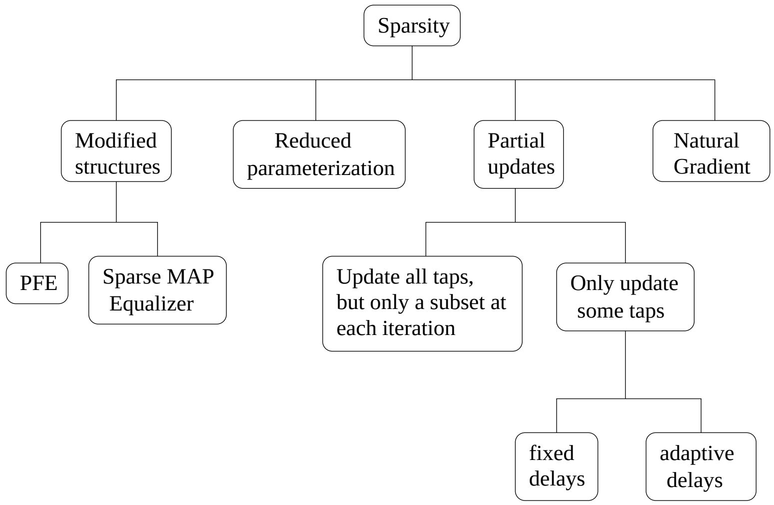

Figure 2.1: The different approaches to exploiting sparsity.

filter. These structures are equivalent to a DFE (in terms of performance) if non-sparse filters are used. However, the extra feedback filter in this structure is more likely to retain the sparseness of the channel; whereas in a traditional DFE, the forward filter smears out the sparsity, so the feedback filter is not as sparse as the channel suggests.

Techniques for reducing training and equalizer complexity through use of this structure have been developed in [5], [13], and [20]. The modified structure is generally referred to as a partial feedback equalizer (PFE), and a block diagram is shown in Figure 2.2.

Another modified equalizer structure is given by Cusani and Mattila [12]. They derive a new maximum *a posteriori* (MAP) equalizer that makes explicit use of the sparsity of the channel. The authors show comparable performance to traditional MAP equalizers, as well as significantly reduced complexity.

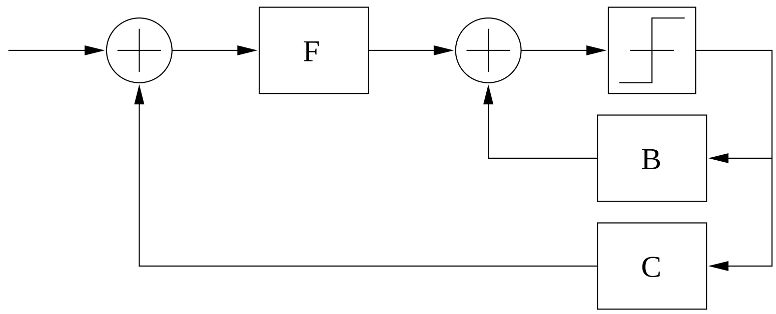

Figure 2.2: Block diagram of a partial feedback equalizer (PFE).

2.2 Partial Updates and Tap selection

The typical approach to sparsity is to simply constrain certain filter taps to zero, and only update the remaining taps. Filters using these strategies need to have some method of selecting which taps are updated. The articles discussed in this section propose tap selection procedures and evalute the effects of partial updates.

One of the earliest papers on this topic is by Treichler and Larimore [37]. The authors propose several methods for choosing the taps, and they discuss the relative merits and drawbacks of the proposed methods. They show that optimal allocation via reduction of the MSE requires a global search.

In [17], the Weiner settings are derived for a sparse, fractionally spaced linear equalizer. The minima settings of the sparse CM and MSE costs are shown to be in close proximity to the full equalizer settings. Simulations are provided to show the feasibility of using a sparse equalizer.

Douglas [15] proposes two algorithms which update all of the taps, but only a few in each time slot. Thus, this algorithm does not exploit sparsity in the sense of not updating certain taps; rather, all the taps are updated, but each tap only gets updated occasionally, so the complexity is reduced. The selection of taps to be updated is done in a periodic fashion, rather than selecting taps to minimize some cost function.

Aboulnasr and Mayyas [2] propose a partial update alorithm that updates the taps in a cyclic, round-robin fashion. As in [15], all taps get updated. However, this article chooses the taps to be updated in an intellegent fashion, with the criteria of keeping the performance for a given number of taps as close as possible to a full-update algorithm.

Douglas' algorithm and the algorithm proposed by Aboulnasr and Mayyas do not explicitly exploit sparsity, yet there are included here because there are related to the other partial update algorithms. These two algorithms could be extended to exploit sparsity by only computing the updates for the selected taps when some activity measure indicates that the taps are large.

Reference [6] proposes a tap selection procedure for the forward taps of a decision feedback equalizer, as motivated by [5]. The procedure involves estimating the channel and the noise power, and then choosing the tap positions with the largest values of an SNR metric (this is only done once, thus the delay values are not adaptive). Only the forward taps with these delays are adapted, though a full feedback equalizer is used. Performance gains are shown for several "typical" (and synthetic) channels.

The use of correlation-based channel estimates is explored in [7]. This report shows that when a training signal is present, the channel can be coarsely identified by correlating the input and output of the channel. Then the largest taps in the estimate can be picked as the taps to be updated in a sparse adaptive update model of the channel.

Etter [19] proposes a gradient-based technique for adaptively determining the delay between two highly correlated sequences. The algorithm requires that the autocorrelation be unimodal or periodic and unimodal within each period, and it must have a zero derivative only at the minimum. Thus, it does not work for white inputs. This method is generalized for use in sparse channel modelling in [18], which increases the requirement on the input autocorrelation function to be unimodal (not just periodically so). This is technically an adaptive scheme, but it is not suited to sparse channels whose delays vary over time.

Homer [23] deals with the specific case of channel identification. An activity measure and activity threshold are proposed, and only taps with an activity measure above the threshold are updated. The results are structurally consistent for both white and colored inputs; however, in the case of colored inputs, adjacent active taps must be separated by twice the maximum correlation length of the input. This means that unless the input is white, the alogrithm in this paper is only useful in long channels with few, widely separated taps. For example, the only simulation results given use channels with three active taps out of one hundred.

In [27], a sparse equalizer is implemented by first forming an accurate estimate of the channel. A sparsing matrix is calculated, which represents the locations of the largest channel taps. This sparsing matrix is then used to determine the sparse equalizer tap locations. As with many underwater acoustic equalization techniques, this is very computationally intensive. The simulations show that there is only a small performance loss compared to using a full equalizer.

A related reference is [35], which also proposes the use of channel estimation. The channel estimate is used for determining the feedback equalizer tap locations, and it is used in conjunction with the output of the decision device to subtract out the post-cursor ISI prior to the feedforward equalizer. This is conceptually similar to the PFE structure discussed in Section 2.1.

Another article that depends on a channel estimate is [28]. This is slightly different from [27] and [35] in that a heuristic method is used to jointly optimize tap locations over both the feedforward and feedback filters. Rather than thresholding the taps to determine the best locations, taps are added if they reduce the excess MSE.

Many of the references above come from the underwater accoustic communications community. A summary of standard sparsity techniques in that field can be found in Chapter 8 of [9].

2.3 Reduced Parameterization

As mentioned in Chapter 1, one interpretation of sparisty is to view the system as having a reduced parameterization, where the reduction comes about from knowing the structure of part of the system. The most common example is when a communications channel consists of the convolution of the (known) pulse shaping filter and the (unknown) transmission channel. Examples include [1], [11], [14].

2.4 Natural Gradient Algorithms

Following the work of Amari, a Bayesian distribution of the parameters to be identified can be thought of as giving rise to an alternative distance metric which must be accounted for if a gradient descent algorithm is employed. This idea is developed in [4], [16], [22], [26], [29], [30]. The algorithms involved are collectively referred to as Natural Gradient (NG) algorithms.

The first natural gradient algorithm is the exponentiated gradient (EG) algorithm, proposed by Kivinen and Warmuth [26]. The EG algorithm involves multiplicative updates to the parameters rather than additive updates (refer to equation (3.3)). [26] provides a comparison of LMS and EG in a machine learning context.

Amari [4] was the first to develop the more general concept of natural gradient algorithms. In [4], it is shown that if the parameter space has a Riemannian structure, then the steepest descent is not obtained by standard gradient descent. A new kind of descent, called the natural gradient, is defined and analyzed, and it is shown to be Fisher efficient. The EG algorithm uses this approach, though it was not developed from this perspective. Douglas and Amari [16] give a detailed overview of natural gradient adaptation.

Hill and Williamson [22] compare several variants of the EG algorithm to LMS, in terms of stady state MSE and convergence rates. It is shown that the $EG^{\pm}$ algorithm converges faster than LMS when the target weight vector is sparse and the steady state MSE's are the same.

Mahony and Williamson [29], [30] develop the concept of Riemannian structure and its impact on NG algorithms. They show that the Riemannian metric can be viewed in a Bayesian framework, and that knowledge of the distribution of the Riemannian is required to implement the algorithms. The principle contribution is the geometrical interpretation of the Riemannian structure.

The work in this thesis is an extension of the work done by Hill, Mahony, and Williamson.

Chapter 3

Motivating Examples

This chapter presents basic versions of the new algorithms and demonstrates their performance in simple settings. The goal is to motivate the futher study of these algorithms that will be given in later chapters.

3.1 The Basic EG Algorithm

As stated in Equation (1.2), the sequence y_k is assumed to be generated from input data x_k in a linear manner. The simplest of the EG algorithms (equation (3.5) in [26]) estimates y_k by

$$\hat{y}_k = \sum_i w_k^i x_k^i. \tag{3.1}$$

In the EG framework, the weights are typically assumed positive. The weight vector $\mathbf{w}_k = [w_k^1, ..., w_k^N]$ is updated at each iteration k by

$$w_{k+1}^i = w_k^i + \mu \; w_k^i \; (y_k - \hat{y}_k) \; x_k^i \tag{3.2}$$

where μ is a small positive stepsize. This can be derived from the EG perspective by an approximation (discussed in section 4.4 of [26]) of the general update strategy

$$w_{k+1}^i = w_k^i \, \exp(\mu \frac{\partial L(y_k, \hat{y}_k)}{\partial w_k^i}) \tag{3.3}$$

where the loss (or cost) function is

$$L(y_k, \hat{y}_k) = (y_k - \hat{y}_k)^2, \tag{3.4}$$

and where $\exp(\cdot)$ is understood as an element by element operation. The approximation involves taking a Taylor series expansion of the exponential and dropping terms of μ^2 or higher. The form of Equation (3.3) explains why the name "Exponentiated Gradient" was chosen.

Alternatively, (3.2) can be derived by estimating y_k by equation (3.1), but with

$$w_k^i = \gamma(z_k^i) = \frac{1}{4} \, (z_k^i)^2 \tag{3.5}$$

for some parameter vector $\mathbf{z}$. Now if we think of the algorithm as adapting over z-space, we can use the Euclidean gradient descent

$$z_{k+1}^i = z_k^i - \mu \, \frac{\partial L(y_k, \hat{y}_k)}{\partial z_k^i}. \tag{3.6}$$

The gradient term becomes

$$\frac{\partial L}{\partial(\hat{y}_k)} \frac{\partial(\hat{y}_k)}{\partial w_k^i} \frac{\partial(w_k^i)}{\partial z_k^i} = -2(y_k - \hat{y}_k) \; x_k^i \; \dot{\gamma}(z_k^i).$$

Now we absorb the 2 into the stepsize to get

$$z_{k+1}^i = z_k^i + \mu \; \dot{\gamma}(z_k^i) \; (y_k - \hat{y}_k) \; x_k^i. \tag{3.7}$$

What we are truly interested in is the effective update rule for $\mathbf{w}$, not $\mathbf{z}$, since that is what is used to generate our estimate $\hat{y}_k$ in (3.1). Since

$$w_{k+1}^i = \gamma(z_{k+1}^i) = \gamma(z_k^i + \textit{small term}),$$

a first order Taylor expansion gives the effective update to $\mathbf{w}$ as

$$w_{k+1}^i = \gamma(z_k^i) + \dot{\gamma}(z_k^i)(small\ term), \tag{3.8}$$

which results in

$$w_{k+1}^i = w_k^i + \mu\ \dot{\gamma}^2(z_k^i)\ (y_k - \hat{y}_k)\ x_k^i. \tag{3.9}$$

For the γ given by (3.5), $\dot{\gamma}^2(z_k^i) = \frac{1}{4}\ (z_k^i)^2 = w_k^i$, which gives us equation (3.2) once again. Thus the EG algorithm can also be thought of as a Euclidean gradient descent algorithm, but the gradient descent is taking place in z-space.

The EG algorithm (3.2) is explicitly contrasted to the standard Gradient Descent (GD) algorithm in [22], [26] and [30]. Within the signal processing community, the GD algorithm is better known as the Least Mean Square (LMS) algorithm, and it can be written using the same notation as

$$w_{k+1}^i = w_k^i + \mu\ (y_k - \hat{y}_k)\ x_k^i. \tag{3.10}$$

The LMS algorithm can be formally derived using the Euclidean gradient descent strategy of (3.6) and the standard parameterization $\mathbf{w} = \mathbf{z}$. Observe that the only difference between the update rules (3.2) and (3.10) is the presence of the w_k^i multiplying the stepsize. In essence, when the current estimate of w_k^i is small, it's update is small, and when the current estimate of w_k^i is large, it's update is large. In terms of sparsity, this means that small taps will contribute less to the Mean Squared Error (MSE), because they will not be wiggling around as much. This allows a larger stepsize (and faster confergence) for the same saymptotic MSE. Therefore, this suggests that the effect of the EG algorithm is to speed convergence in some cases.

This can be validated quantitatively. In [22], the MSE (defined as $E(|y - \hat{y}|^2)$)

of a simple EG algorithm with a white input signal is shown to be

$$\xi = \left(1 + \mu\left(\|\mathbf{w}^*\|_1 - \frac{\|\mathbf{w}^*\|_2}{\|\mathbf{w}^*\|_1}\right)\sigma_x^2\right)\sigma_n^2, \tag{3.11}$$

where $\mathbf{w}^*$ is the target weight vector, while the MSE for LMS is

$$\xi = \left(1 + \mu N \sigma_x^2\right)\sigma_n^2, \tag{3.12}$$

The key term is $(||\mathbf{w}||_1 - \frac{||\mathbf{w}||_2}{||\mathbf{w}||_1})$, which provides a measure of the sparsity of the target weight vector. When there is considerable sparsity, this term is small, and the stepsize can be made larger for faster convergence without adversely affecting the excess MSE. When this term is large there is not much sparsity, and the stepsize must remain small in order not to increase the MSE. For example, consider the channel $[1, a, a, a, a, 0.5]$. For $|a| = 0.05$, the "measure of sparsity" equals 1.04, while for $|a| = 0.5$, it equals 3.07.

Somewhat more formally, [30] investigates the geometry of the error surface over which the EG algorithms evolve. Again, the EG algorithms are found to converge faster than LMS (for equal asymptotic total MSE) when there is sparsity in the target weight vector. Consequently, for applications in communications which have channels with a large time span, but only a (relatively) small number of significant taps, EG-like algorithms may be a good idea.

3.2 Practical Implementation

To further motivate the use of EG-like algorithms, again consider the alternative parameterization of the linear predictor given by (3.5). In essence, (3.5) estimates a function of the unknown parameters, rather than unknown parameters themselves.

However, in most signal processing and communications applications the weights are not constrained to be positive, as is required by both methods of deriving the

EG update rule (3.2). In [26], a modification called the $EG^{\pm}$ algorithm is proposed for this task, but a simpler method (first suggested in [29]) is to modify (3.5) to

$$\hat{y}_k = \sum_i \mathrm{sgn}(z_k^i)\ (z_k^i)^2\ x_k^i.$$

The Euclidean gradient algorithm in z-space then becomes

$$z_{k+1}^i = z_k^i + \mu\ |z_k^i|\ (y_k - \hat{y}_k)\ x_k^i$$

which allows both positive and negative weights, and retains the same kinds of sparsity and stepsize advantages as (3.2). Note that this also allows $\mathbf{w}$ to have positive and negative elements. This parameterization still has a pitfall, though. If $z_k^i = 0$, then it's update term is also zero. This prevents initialization by zeros, and prevents the weights in the model from changing sign even if the actual weights change sign. This equilibrium at $z_k^i = 0$ can also be removed by further modifying the parameterization to

$$\hat{y}_k = \sum_i \left(\frac{1}{2}\mathrm{sgn}(z_k^i)\ (z_k^i)^2 + \epsilon z_k^i\right)\ x_k^i. \tag{3.13}$$

which leads to the Euclidean gradient algorithm

$$z_{k+1}^i = z_k^i + \mu\ (|z_k^i| + \epsilon)\ (y_k - \hat{y}_k)\ x_k^i. \tag{3.14}$$

The effective update rule for $\mathbf{w}$ is

$$w_{k+1}^i = w_k^i + \mu\ (2|w_k^i| + \epsilon^2)\ (y_k - \hat{y}_k)\ x_k^i. \tag{3.15}$$

Compared to (3.2), the algorithm has hardly changed, though it is now more useful for signal processing applications.

The EG-inspired algorithms can exploit a priori information about the structure of the unknown weights. For example, consider the job of adapting an equalizer for a large but sparse channel. In all likelihood, there will be a cluster of large taps somewhere near the center of the equalizer, while large taps at the edges would be rare. This kind of prior information can be readily incorporated into the algorithm. For the weights near the center, an algorithm like LMS could be used, while for the weights near the edges, an EG-like algorithm might be more appropriate. As we will see in Chapter 5, it is easy to tailor the algorithms to exploit such prior knowledge.

3.3 Some Low Dimensional Examples

In order to gain some insight as to the shape of the cost surface (in z-space) that algorithms such as (3.7) evolve over, we will look at a 2-tap example. We will then use the intuition gained from this example to understand the evolution of the parameters in a somewhat more useful 10-tap example.

Consider a standard system identification problem, where the system to be identified is $\mathbf{w}_* = [1, 0.156]$. The two algorithms to be compared are LMS, as given by (3.10), and what we will call "signed sparse LMS" (or SSLMS), as given by (3.14) and (3.15). Signed sparse LMS has the parameterization $w_k^i = \gamma(z_k^i) = (\frac{1}{2}\text{sgn}(z_k^i)(z_k^i)^2 + \epsilon z_k^i)$, so the vector $\mathbf{z}_*$ that gets mapped by γ onto $\mathbf{w}_*$ is $\mathbf{z}_* = [1.353, 0.5]$.

Since both of these algorithms are gradient descent algorithms, we can plot the cost surfaces over which they are evolving. Figure 3.1 shows the standard MSE cost $L = E(y_k - \hat{y}_k)^2$ over which LMS evolves. Note that LMS is a gradient descent in w-space, so the axes are the parameters w_1 and w_2. In contrast, SSLMS is a gradient descent of the same cost function, but the gradient is with respect to $\mathbf{z}$, so Figure 3.2 shows the MSE cost plotted against the axes z_1 and z_2.

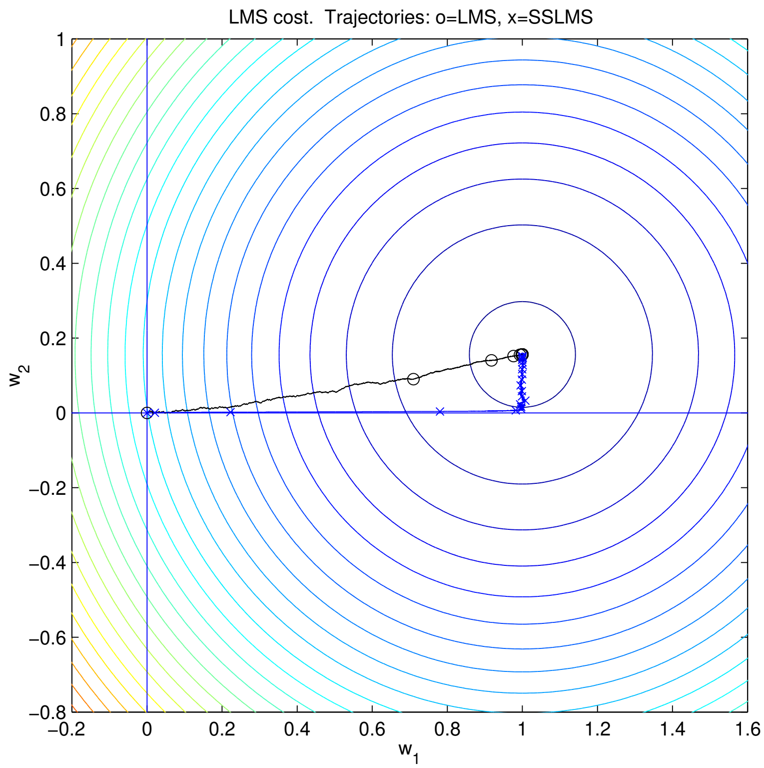

Figure 3.1: The LMS cost surface and trajectories of LMS and SSLMS.

There are several trajectories superimposed over the cost surfaces. In Figure 3.1, the trajectory with 'o' marks is that of LMS. Note that it is always perpendicular to the contour lines. The trajectory with 'x' marks is that of SSLMS. In reality, SSLMS evolves over z-space, so this trajectory was computed by letting SSLMS run in z-space, and then using γ to map the points on the trajectory into w-space. Figure 3.2 shows this same trajectory as it evolves in z-space. Note that the SSLMS trajectory is perpendicular to the SSLMS cost contours, but not the LMS cost contours.

Figure 3.2 can provide some insight as to the effect of the reparameterization. If one of the parameters is small at a given point, the cost surface at that point has

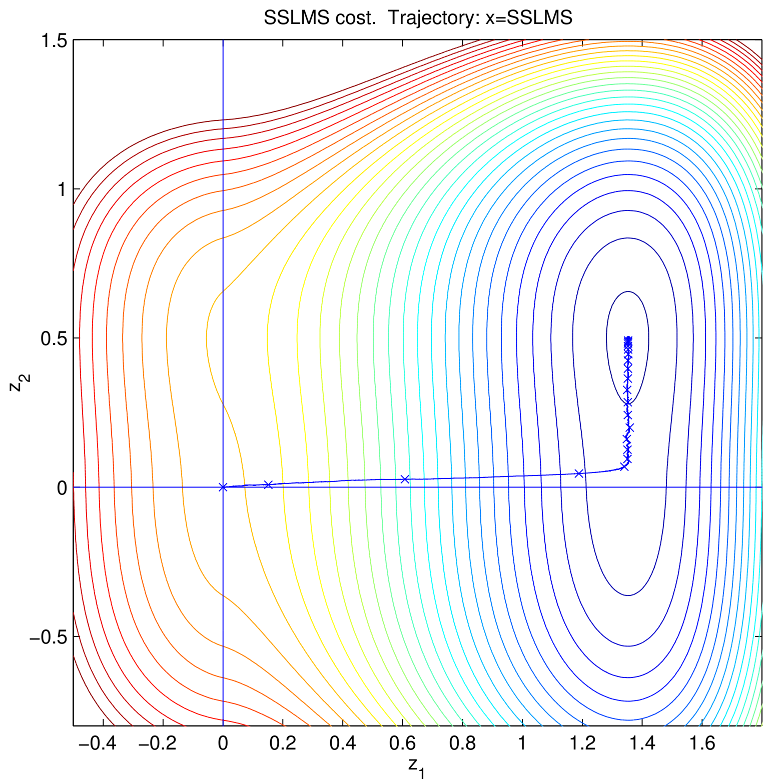

Figure 3.2: The SSLMS cost surface and a trajectory of SSLMS.

been flattened out along that coordinate. More precisely, if z^i is small, then $\frac{\partial L}{\partial z^i}(\mathbf{z})$ is smaller than it was for LMS. Thus, at a point $\mathbf{z_k}$, SSLMS tends to move fastest along the coordinate of its largest tap, $\max_i |z_k^i|$, and slowest along the coordinate of its smallest tap, $\min_i |z_k^i|$. If $\dot{\gamma}$ is a monotonically increasing function of $|z|$ (as is generally the case), then the index i of the largest $|z_k^i|$ is the same as the index of the largest $|w_k^i|$.

In terms of the trajectories, Figure 3.2 shows SSLMS evolving faster in direction z_1, which agrees with the fact that $z_1 > z_2$. As shown by the corresponding trajectoriy in Figure 3.1, the effect is to push the original LMS trajectory so that it points more in the z_1 direction than it would otherwise. This concept is readily generalized to higer dimensional examples, in which we will see that when SSLMS is used, the largest tap adapts unnaturally fast, but at the expense of the adaptation speed of the smaller taps.

The 2-tap case is not sparse enough for the sparse algorithms to gain an advantage in terms of convergence speed. In order to find a case where SSLMS converges faster, we will look at a 10-tap case where only 4 taps are active. We will implement the update rule in w-space instead of z-space. This corresponds to choosing equation (3.15) rather than equation (3.14). There are two reasons for doing this: first, it is simpler to implement. Second, in equation (3.8), we ignored terms higher than first order, but it turns out that in this particular case, the approximation is not so good near the origin. Since we want to initialize at the origin, we can avoid this issue by not reparameterizing $\mathbf{w}$.

This example will be repeated in more detail in Chapter 7, but for now it suffices to say that the stepsizes for LMS and SSLMS and the parameter ϵ are chosen to equilibrate the asymptotic total MSE.

Figure 3.3 shows the evolution of the first four taps over time. The true values of these taps are 0.1, 0, 1, and -0.5. We see that the largest tap (W_3) adapts much faster using SSLMS than using LMS. The smallest tap (W_2) should stay near zero, but LMS causes it to have a small amount of error. This error is reduced by using SSLMS. A small, but nonzero tap (W_1) is slowed slightly by SSLMS. A midsize tap (W_4) adapts at roughly the same rate for the two algorithms, although for SSLMS it is slower at first, then faster than LMS as it nears convergence.

This agrees with the results from the two-tap case. When using SSLMS, the taps tend to adapt more slowly when they are small, so taps with large true values will be the fastest to adapt. The net result is that small taps are slowed at the expense of the big taps. Since there are many small taps in this example, the large

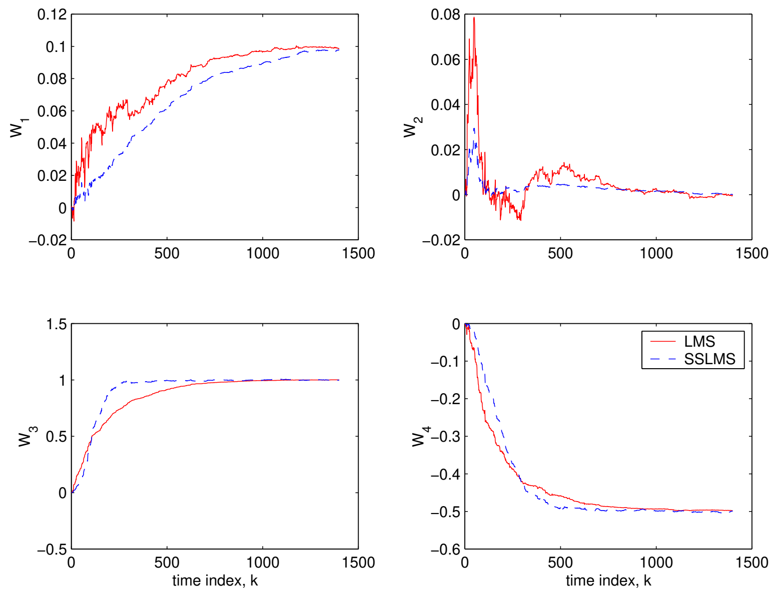

Figure 3.3: Evolution of the first four parameters (out of ten).

taps are significantly speeded up.

Figure 3.4 shows the MSE of the two algorithms versus time. We see that for a short time, SSLMS is slower than LMS. This is because we initialized at the origin, and it takes a moment to get off the plateau of the cost function at the origin (compare to Figure 3.2). However, after that point, the large taps can move very quickly, and the MSE decreases rapidly. After about 300 iterations, the largest tap has converged, and SSLMS slows down. After about 900 iterations, the second largest tap has converged, and all that is left is two small (nonzero) taps and 6 zero taps. The small taps take a long time to converge, and during that time, LMS catches up.

The results and interpretation of Figure 3.4 can be extended to the time-varying case. If the large taps move around, then SSLMS will have an easier time tracking them than LMS. However, if the small taps are the only ones that move, then

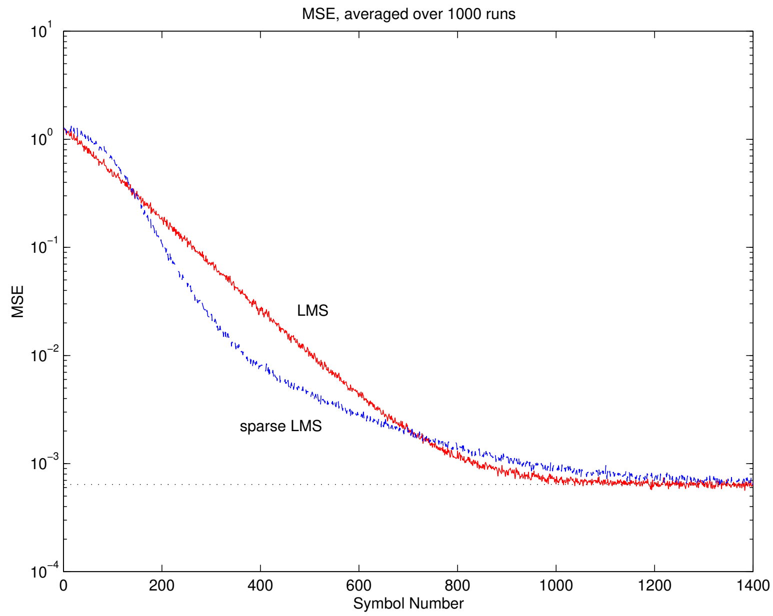

Figure 3.4: MSE versus time for LMS and SSLMS.

SSLMS will have more trouble tracking them than LMS. A formal simulation will be presented in Chapter 7.

The examples in this section have motivated more rigorous study of EG-like algorithms. A formal discussion of asymptotic MSE, stability, and directionality of convergence is explored in Chapter 4. Chapter 5 gives an alternative way to derive these algorithms in terms of prior knowledge, and Chapter 6 derives a host of algorithms, including variants of CMA and decision directed algorithms, using this framework.

Chapter 4

Analysis

This chapter derives theoretical expressions for the MSE, convergence rates, and stability criteria of the general class of algorithms introduced in Chapter 3, in a fashion similar to that in [38] and [39]. For simplicity, it is assumed throughout this chapter that the cost function is the standard MSE cost function of (3.4).

4.1 Mean Squared Error

Consider the vector update rule of

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mu \; D \; \hat{\bigtriangledown}_k(J), \tag{4.1}$$

or as in [38],

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mu \; D \; (-\bigtriangledown_k + \Gamma_k), \tag{4.2}$$

where $\bigtriangledown_k$ is the gradient of the cost surface at time k with respect to $\mathbf{w}_k$; $\hat{\bigtriangledown}_k(J)$ is the estimate of this gradient; Γ_k is the gradient noise (the difference between the true gradient and the estimate of the gradient used by the algorithm), as in [38]; and D is a diagonal matrix. This is a more general form of the LMS update rule (in which $D = I$). Examples of algorithms of this form can be found in the previous chapter, as in equations (3.2), (3.9), and (3.15); and a more general form is given in the next chapter, by equation (5.10). In general, D depends on the current value of the weight estimates, $\mathbf{w}$, but in this section we will assume that $\mathbf{w} \cong \mathbf{w}^*$, which is reasonable when we are considering asymptotic MSE.

Using the definition of the error system $\mathbf{v}_k = \mathbf{w}_k - \mathbf{w}^*$ and the fact that the gradient can be expressed as $2R\mathbf{v}_k$ where $R = E[\mathbf{X}\mathbf{X}^T]$ (see [39]),

$$\mathbf{v}_{k+1} = \mathbf{v}_k + \mu \; D \; (-2R\mathbf{v}_k + \Gamma_k)$$

or

$$\mathbf{v}_{k+1} = (I - 2\mu DR) \; \mathbf{v}_k + \mu \; D \; \Gamma_k. \tag{4.3}$$

At this point, [38] rotates the coordinate system by diagonalizing R. That is not feasible here, because if we diagonalize DR to become $Q^{-1}DRQ$ for an appropriate Q, $Q^{-1}DQ$ will not neccessarily be diagonal.

To determine the MSE, find the covariance matrix of $\mathbf{v}_k$. Define $C = E[\mathbf{v}_k \; \mathbf{v}_k^T]$, and assume the sytem is near convergence (hence $\text{cov}(\mathbf{v}_{k+1}) \cong \text{cov}(\mathbf{v}_k)$), then

$$C = (I - 2\mu DR) \; C \; (I - 2\mu DR)^T + \mu^2 D \text{cov}(\Gamma_k) D.$$

It is easily shown (see [38]) that $\text{cov}(\Gamma_k) = 4\xi_{min} R$, where ξ_{min} is the minimum MSE (the MSE for $\mathbf{w} = \mathbf{w}^*$ exactly, and $\mu = 0$). Inserting this and absorbing the 2's and the 4 into μ and μ^2 gives

$$C = (I - \mu DR) \; C \; (I - \mu DR)^T + \mu^2 \; \xi_{min} \; DRD. \tag{4.4}$$

The $(\mu DR)^2$ term can be dropped on the grounds that μ is very small. (An exact solution is given in Appendix A.) Simplifying yields

$$C(-DR)^T + (-DR) \; C \cong -(\mu \; \xi_{min} \; DRD). \tag{4.5}$$

The negative signs have been included to cast Equation (4.5) in the form of the Lyapunov equation [25]. In general, this equation cannot be solved for C in closed form. However, due to the structure of the right hand side, a solution is possible, and is given by

$$C = \frac{1}{2}\ \mu\ \xi_{min}\ D. \tag{4.6}$$

According to [25], a sufficient condition for our solution for C to be unique is if all the eigenvalues of $-DR$ have negative real parts; that is, if $-DR$ is a stability matrix.

An exact computation of the solution to (4.4) without the approximation leading to (4.5) is given in Appendix A, namely

$$C = \frac{1}{2}\ \mu\ \xi_{min}\ D\ \left(I - \frac{\mu}{2} RD\right)^{-1}, \tag{4.7}$$

which holds provided that all eigenvalues of $\frac{\mu}{2}RD$ are less than one in magnitude. We see that for small μ, the solution given by (4.6) is indeed valid. For simplicity, (4.6) will henceforth be used instead of (4.7).

Widrow et. al. [39] show that the average excess MSE for standard LMS is given by

$$E[\acute{\mathbf{v}}_k^T\ \Lambda\ \acute{\mathbf{v}}_k] = \sum_{p=1}^{n} \lambda_p E[(\acute{v}_{p,k})^2]$$

where $Q^{-1}RQ = \Lambda$, $\acute{\mathbf{v}}_k = Q^{-1}\mathbf{v}_k$, and λ_p is the p^{th} diagonal element of Λ. To use this, note that $\mathrm{cov}(\acute{\mathbf{v}}) = Q^{-1}\mathrm{cov}(\mathbf{v})Q = Q^{-1}CQ$. From (4.6), the theoretical MSE ξ is then

$$\xi = \xi_{min}\ (1 + \frac{1}{2}\mu \sum_{p=1}^{n} \lambda_p [Q^{-1}\ D\ Q]_{p,p}) \tag{4.8}$$

where $[.]_{p,p}$ indicates the p^{th} diagonal element.

In the special case when D and R are diagonal, Q reduces to I, yielding

$$\xi = \xi_{min}\ (1 + \frac{1}{2}\mu \mathrm{tr}(DR)). \tag{4.9}$$

When D is just the Jacobian used by [22], this simplifies to the result derived in [22]. For the algorithms considered in this thesis, D is always diagonal. Though [22] shows that D is non-diagonal for the regular EG algorithm, it *is* diagonal for the approximate algorithm studied in Chapter 3. However, R is often not diagonal, such as in an equalization setting, or in any other application with a non-white input sequence.

Equation (4.8) provides a basis for a fair comparison between traditional LMS and the algorithms described in this thesis. For a given algorithm and system, it is possible to compute the asymptotic total MSE as a function of μ. The stepsizes for the different algorithms can then be adjusted such that at convergence, all have the same MSE. Then simulations can be run, and other factors (such as convergence time) can be compared fairly.

In a real environment, since D depends on $\mathbf{w}^*$, it is not known a priori. In this case, a fair comparison can be made by taking the expectation of Equation (4.8) over all D, using an appropriately normalized prior as the probability density function.

4.2 Convergence Rates and Stability Criteria

This section calculates the convergence rates and stability criteria using the methods of [34] and [39]. We first consider local convergence. Starting from the update rule obtained from (4.3) by neglecting noise,

$$\mathbf{v}_{k+1} = (I - \mu DR)\ \mathbf{v}_k, \tag{4.10}$$

diagonalize the matrix DR via

$$\hat{Q}^{-1}\ (DR)\ \hat{Q} = \hat{\Lambda}. \tag{4.11}$$

D is given as in (5.10), which assumes the algorithm is near convergence. This is the only step that has that requirement. The update rule becomes

$$\hat{\mathbf{v}}_{k+1} = (I - \mu\hat{\Lambda})\ \hat{\mathbf{v}}_k. \tag{4.12}$$

Now the elements in Equation (4.11) are completely uncoupled, and the p^{th} element decays with a time constant of $\tau_p \cong \frac{1}{\mu\hat{\lambda}_p}$, where $\hat{\lambda}_p$ is the p^{th} eigenvalue of DR. As in [39], local convergence of the average system requires that

$$0 < \mu < \frac{2}{\hat{\lambda}_{max}}. \tag{4.13}$$

Of course, this only assures convergence of the average system. In order for the stochastic system to converge, the stepsize μ must be "small," so a stability requirement for the stochastic system would be much more conservative than (4.13). Thus, local convergence of these algorithms only requires a bound on the stepsize. This is reasonable, since the same fact is well known for the traditional LMS algorithm [39].

Now consider global convergence. To relax the requirement that the algorithm is near convergence, allow the matrix D_k to be *time-varying*, instead of fixing it to be evaluated at the optimum solution. Diagonalizing using

$$\hat{Q}_k^{-1}\ (D_k R)\ \hat{Q}_k = \hat{\Lambda}_k \tag{4.14}$$

yields the time-varying system

$$\hat{\mathbf{v}}_{k+1} = (I - \mu\hat{\Lambda}_k)\ \hat{\mathbf{v}}_k. \tag{4.15}$$

The elements in Equation (4.15) are completely uncoupled, and the p^{th} element decays with an *instantaneous* time constant of $\tau_{p,k} \cong \frac{1}{\mu\hat{\lambda}_{p,k}}$, where $\hat{\lambda}_{p,k}$ is the p^{th} eigenvalue of $\hat{\Lambda}_k$. Comparing to (4.13), one requirement for global convergence of the average system is that

$$0 < \mu < \frac{2}{\hat{\lambda}_{max,k}} \tag{4.16}$$

for all k, where $\hat{\lambda}_{max,k}$ is the largest eigenvalue of D_kR. Limiting the size of D_k with a clipping function is one way to insure that $\hat{\lambda}_{max,k}$ never grows too large.

However, since the diagonalization is time-varying, any claims of stability have an additional requirement: the rate of the time variation of $(I - \mu\hat{\Lambda}_k)$ must be bounded. Assume that the previously mentioned clipping function is applied to $\mathbf{z}$ to bound the values of D_k, and assume that the stepsize μ is small. Then the "slow time variation lemma" of [34] can be applied:

Consider the time-varying system

$$X_{k+1} = A_kX_k \tag{4.17}$$

and assume that the related frozen systems

$$X_{k+1} = A_pX_k$$

are exponentially stable for every integer p, uniformly in p. If $||A_p|| < \alpha_1$ for all p and if $sup_{p>p_0}||A_{p+1} - A_p||$ is small enough for some finite p_0, then systems (4.17) are exponentially asymptotically stable.

This lemma shows that if $||(I - \mu\hat{\Lambda}_p)||$ is bounded for all p and if $\sup_{p>p_0}||\hat{\Lambda}_{p+1} - \hat{\Lambda}_p||$ is small enough for some finite p_0, then the average system (4.15) is exponentially asymptotically stable. The first requirement is met because of the clipping function on D_k, and the second by the choice of a small stepsize.

4.3 Directionality of Local Convergence

In this section we will discuss the fact that the speed of convergence (in particular, near the true weight vector) depends heavily on the direction of the parameter error vector; in fact, more so than in LMS.

Again consider the average system, with update rule

$$\mathbf{v}_{k+1} = (I - \mu D R)\ \mathbf{v}_k,$$

where $\mathbf{v}_k = \mathbf{w}_k - \mathbf{w}^*$. Define

$$\Delta\mathbf{v}_k = \mu\ D\ R\ \mathbf{v}_k.$$

The MSE at times k and $k+1$ are

$$\xi_k = \mathbf{v}_k^T\ R\ \mathbf{v}_k,$$

$$\xi_{k+1} = (\mathbf{v}_k + \Delta\mathbf{v}_k)^T\ R\ (\mathbf{v}_k + \Delta\mathbf{v}_k).$$

This means that if we are at $\mathbf{w}_k$ at time k, then the change in the MSE will be

$$(\Delta\xi)_k = \xi_{k+1} - \xi_k = (\mathbf{v}_k + \Delta\mathbf{v}_k)^T\ R\ (\mathbf{v}_k + \Delta\mathbf{v}_k) - \mathbf{v}_k^T\ R\ \mathbf{v}_k$$

$$= -\mu\mathbf{v}_k^T\ R(2D - \mu D^2)R\ \mathbf{v}_k.$$

If we assume μ is small, then we get

$$(\Delta\xi)_k = -2\mu\mathbf{v}_k^T\ R\ D\ R\ \mathbf{v}_k. \tag{4.18}$$

To proceed further, we must make some simplifying assumptions. First, assume that the input correlation matrix R is the identity, I. Second, assume $N = 2$, so that we can visualize our results. For concreteness, we will assume $D_{ii} = |w_i| + \epsilon$ (corresponding to signed sparse LMS). We would like to compare values of (4.18) for LMS and SSLMS. To do that, we must choose the stepsize for SSLMS to equilibrate the asymptotic MSE. Using (4.9), we have

$$\mu_A \; N = \mu_B \; \mathrm{Tr}(D^*),$$

where μ_A is the stepsize for LMS and μ_B is the stepsize for SSLMS. This means that the decrease in MSE for LMS is

$$(\Delta\xi)_{LMS} = -2\mu_A[(w_1 - w_1^*)^2 + (w_2 - w_2^*)^2], \tag{4.19}$$

and the decrease in MSE for SSLMS is

$$\begin{aligned}(\Delta\xi)_{SSLMS} &= \\ &- 2\mu_A \left[(w_1 - w_1^*)^2 \left(\frac{2|w_1| + 2\epsilon}{|w_1^*| + |w_2^*| + 2\epsilon}\right) + (w_2 - w_2^*)^2 \left(\frac{2|w_1| + 2\epsilon}{|w_1^*| + |w_2^*| + 2\epsilon}\right)\right].\end{aligned} \tag{4.20}$$

When $(\Delta\xi)_{SSLMS} < (\Delta\xi)_{LMS}$, SSLMS will have a larger decrease in MSE (note that in both cases $\Delta\xi < 0$). Even for this simple case, it is difficult to compute a closed form expression for the region in which this is true. However, we can compute it numerically. A plot is given in Figure 4.1, with $\mathbf{w}^* = [1, 0.1]$. The shaded region marks values of (w_1, w_2) where SSLMS is faster than LMS, in terms of decreasing MSE.

Figure 4.1 shows that in the two tap case, if we initialize at the origin, LMS will have an advantage at first; but as we approach $\mathbf{w}^*$, SSLMS will go faster. Since LMS is faster over most of the route, it should do better than SLMS in this particular case. However, this case is not truly sparse. Recall the motivating example of Section 3.3. In that case, the system was much more sparse. LMS still moved faster at first, but SSLMS gained an advantage more quickly than Figure 4.1 implies. The qualitative effects are similar, though.

Of more importance is the fact that if we initialize near the right answer, we must choose our initialization carefully. If we initialize with more of the error in

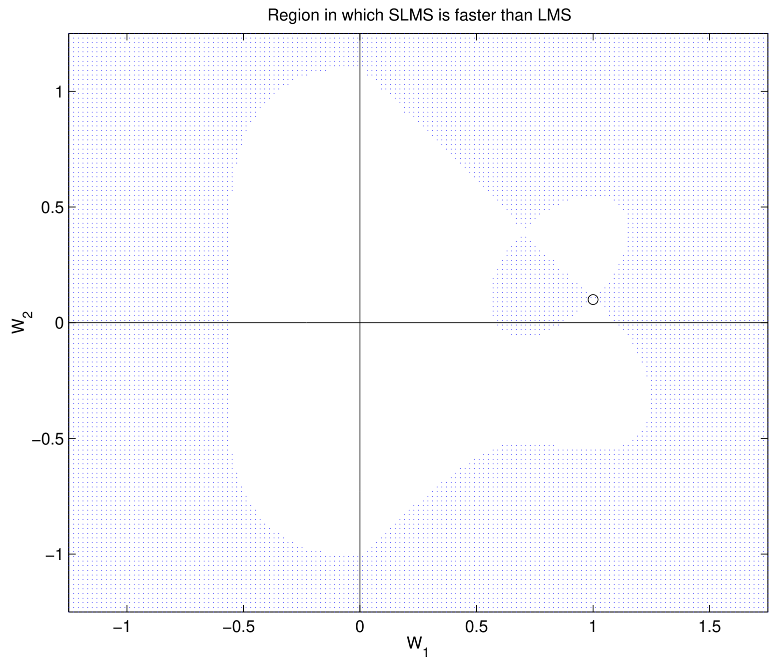

Figure 4.1: Region (shaded) in which SSLMS outperforms LMS.

parameter w_1, then SSLMS will go faster. However, if we initialize with more of the error in parameter w_2, then SSLMS will be slower than LMS. This is an interesting fact, since it shows that there is a strong directionality to this algorithm even though the input is uncorrelated. This is in contrast to LMS, which converges equally fast in all directions when the input is uncorrelated.

Now consider the following algorithm, which shall be called "normalized sparse LMS", or simply "NSLMS". Define D as before, $D_{ii} = |w_i| + \epsilon$. Now define $\hat{D} = \frac{D}{\mathrm{Tr}(D)}$ and $\hat{\mu}_B = \mu_A N$. The update equation becomes

$$\mathbf{v}_{k+1} = (I - \hat{\mu}_B \hat{D})\ \mathbf{v}_k = (I - \mu_A N \frac{D}{\mathrm{Tr}(D)})\ \mathbf{v}_k.$$

Application of equation (4.9) shows that this will have the same asymptotic MSE as the LMS update. The difference between this case and the last is that we are dealing with $\mathrm{Tr}(D)$, not $\mathrm{Tr}(D^*)$. Note that this makes the algorithm similar to $EG^{\pm}$ as in equations (3) and (4) of [22].

Working through the algebra shows that the change in MSE for LMS is still given by (4.20); but for NSLMS, the denominator is now given by $\mathrm{Tr}(D)$ instead of $\mathrm{Tr}(D^*)$, i.e.

$$\begin{aligned}(\Delta\xi)_{NSLMS} &= \\ &- 2\mu_A\left[(w_1 - w_1^*)^2\left(\frac{2|w_1| + 2\epsilon}{|w_1| + |w_2| + 2\epsilon}\right) + (w_2 - w_2^*)^2\left(\frac{2|w_2| + 2\epsilon}{|w_1| + |w_2| + 2\epsilon}\right)\right].\end{aligned} \tag{4.21}$$

NSLMS will be faster than LMS if and only if $(\Delta\xi)_{NSLMS} < (\Delta\xi)_{LMS}$ (note that in both cases $\Delta\xi < 0$). This is equivalent to

$$\left[(w_1 - w_1^*)^2\left(\frac{2|w_1| + 2\epsilon}{|w_1| + |w_2| + 2\epsilon} - 1\right) + (w_2 - w_2^*)^2\left(\frac{2|w_2| + 2\epsilon}{|w_1| + |w_2| + 2\epsilon} - 1\right)\right] > 0.$$

or

$$\left[(w_1 - w_1^*)^2\left(\frac{|w_1| - |w_2|}{|w_1| + |w_2| + 2\epsilon}\right) + (w_2 - w_2^*)^2\left(\frac{|w_2| - |w_1|}{|w_1| + |w_2| + 2\epsilon}\right)\right] > 0.$$

Since the denominator, $|w_1|+|w_2|+2\epsilon$, is always positive, the condition for NSLMS to be faster than LMS simplifies to

$$(|w_1| - |w_2|)\left((w_1 - w_1^*)^2 - (w_2 - w_2^*)^2\right) > 0. \tag{4.22}$$

Either both factors must be positive, or both must be negative. The roots of the second factor are easily shown to be $|w_1 - w_1^*| = |w_2 - w_2^*|$. Thus, the plane should be divided by four lines:

$$w_2 = w_1, \tag{4.23}$$

$$w_2 = -w_1, \tag{4.24}$$

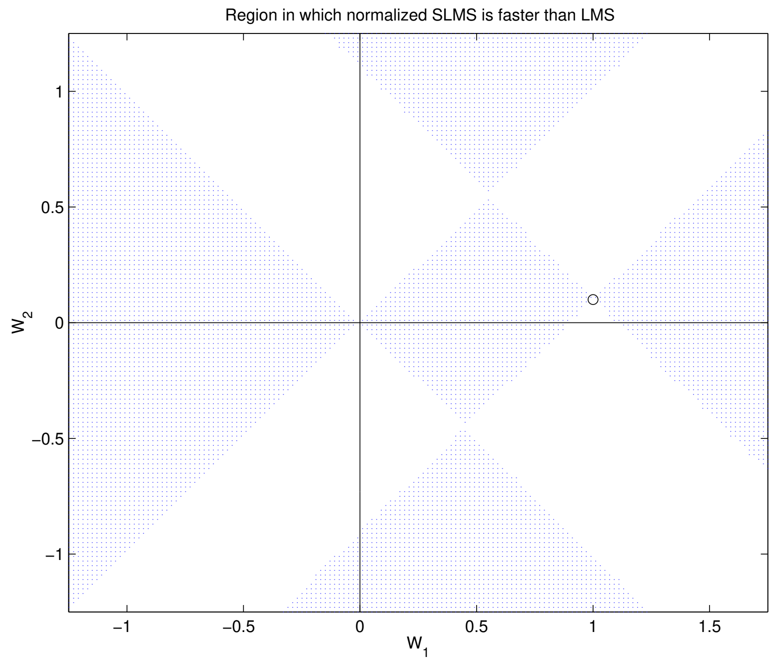

Figure 4.2: Region (shaded) in which normalized SLMS outperforms LMS.

$$w_2 = w_2^* + (w_1 - w_1^*), \tag{4.25}$$

$$w_2 = w_2^* - (w_1 - w_1^*). \tag{4.26}$$

Equations (4.23) and (4.24) come from the first factor, and equations (4.25) and (4.26) come from the second factor. A plot of the region in which NSLMS is faster than LMS is given in Figure 4.2. Again, $\mathbf{w}^* = [1, 0.1]$.

From this figure, we see that near the solution, the directionality is qualitatively the same as before. However, the path from the origin to the solution now lies entirely in space in which NSLMS moves faster than LMS. Since NSLMS is similar to $EG^{\pm}$ as in [22], this agrees with the result in [22] that $EG^{\pm}$ is faster than both LMS and unnormalized EG (which is similar to our sparse LMS). However, NSLMS is more computationally intensive than SSLMS, so it may be preferable to implement SSLMS instead (especially if you are far from the origin). Furthermore, NSLMS does not easily lend itself to the framework we will construct in the next chapter, so we will not consider it further.

The results in this section imply that algorithms of the form 4.1 are not globally faster than standard algorithms. Rather, is depends on what region of the parameter space you are in. The critical region will vary with the algorithm and with the true weight vector, $\mathbf{w}^*$.

Chapter 5

Exploiting Prior Knowledge

This chapter derives a general form of reparameterized gradient algorithms that can be understood in terms of prior knowledge or in terms of an underlying cost function. The various algorithms will differ in their choice of cost (loss) function $L(y_k, \hat{y}_k)$, in their choice of (re)parameterization of the linear filter, and in their choice of the Bayesian prior of the parameters to be identified. To be specific, let

$$\hat{y}_k = \sum_i w_k^i x_k^i \tag{5.1}$$

where

$$w_k^i = \gamma(z_k^i) \tag{5.2}$$

as in Equation (8) of [22], and $\gamma(z_k^i)$ is invertible (i.e., non-zero), at least over its domain. Normally, it will be differentiable over this domain, though isolated points of discontinuity may be allowed. Each of the different algorithms will have different $L(\cdot, \cdot)$ and/or different $\gamma(\cdot)$ functions. The algorithms we consider will be algorithms which update the entries of $\mathbf{z}$, and thus $\mathbf{w}$.

Mahony and Williamson [29] provide a general discussion of how to encode prior knowledge into learning algorithms using a geometric "preferential structure." The essence of this is to define a metric so that the algorithm evolves over an error surface that is shaped to incorporate the known prior information. For instance, if the ith component is considered to be reliable while the jth component is not, then the algorithm should take larger steps in the jth direction. This is accomplished by warping the underlying space, in effect augmenting or diminishing the stepsizes of the algorithm in the desired directions.

Mathematically, the preferential metric is a family of functions $\phi_i(z^i)$ which represent the a priori knowledge (the Bayesian prior) of the ith parameter. The idea of a Bayesian prior is that an unknown parameter that is to be estimated is viewed as a random variable with a (known) probability density function [33]. In this case, $\phi_i(z^i)$ may be viewed as the probability density function (pdf) for the unknown parameter z^i. This interpretation of $\phi_i(z^i)$ as a pdf should be used cautiously, as we will see that in some cases the integral of $\phi_i(z^i)$ cannot be normalized to 1.

Using this concept of priors, Mahony and Williamson [29] show that when the standard parameterization is used ($\gamma(z_k^i) = z_k^i$, so $\mathbf{w} = \mathbf{z}$) the "natural gradient" (NG) algorithm is

$$z_{k+1}^i = \Phi_i^{-1}\left(\Phi_i(z_k^i) - \mu \frac{\partial L}{\partial z_k^i} \frac{1}{\phi_i(z_k^i)}\right) \tag{5.3}$$

where Φ is the indefinite integral of ϕ. Note that if we interpret ϕ as a pseudo-probability function that is always positive, then its integral will be strictly increasing, and Φ^{-1} will always exist. However, if ϕ is zero over any of its domain, then Φ^{-1} will not exist.

Unfortunately, the updates of the NG algorithm (5.3) can be quite complicated due to the presence of the nonlinearities Φ and Φ^{-1}. A more tractable algorithm can be derived as a first order approximation to (5.3) by rewriting the update as

$$\Phi_i(z_{k+1}^i) = \Phi_i(z_k^i) - \mu \, \frac{\partial L}{\partial z_k^i} \, \frac{1}{\phi_i(z_k^i)}. \tag{5.4}$$

Expanding $\Phi_i(z_{k+1}^i)$ in a Taylor Series about z_k^i gives

$$\Phi_i(z_{k+1}^i) = \Phi_i(z_k^i) + \phi(z_k^i)(z_{k+1}^i - z_k^i) + o(\mu^2),$$

which uses the fact that $\frac{\partial \Phi_i(z_k^i)}{\partial z_k^i} = \phi_i(z_k^i)$. When μ is sufficiently small, the higher order terms may be neglected. Substituting into the left hand side of (5.4) gives

$$\Phi_i(z_k^i) + \phi(z_k^i)(z_{k+1}^i - z_k^i) = \Phi_i(z_k^i) - \mu \, \frac{\partial L}{\partial z_k^i} \, \frac{1}{\phi_i(z_k^i)}.$$

Finally, dividing both sides by $\phi(w_k^i)$ and rearranging gives the algorithm

$$\boxed{z_{k+1}^i = z_k^i - \mu \, \frac{\partial L}{\partial z_k^i} \, \frac{1}{\phi_i^2(z_k^i)}.} \tag{5.5}$$

Note that at this point, we are still assuming a standard parameterization ($\gamma(z) = z$), so all of the z's in equation (5.5) are equivalent to w's. This "approximate natural gradient" (ANG) algorithm may be preferred to (5.3) in applications since the updates are simpler and the resulting algorithms can be more readily analyzed using standard techniques such as averaging theory [31] and weak convergence theory [10]. Moreover, since the idea behind gradient descent algorithms is to take small steps towards the goal, the difference between the small steps of the NG algorithm and the small steps of its first order approximation (the ANG) are often minor.

In order to make use of the ANG family of algorithms, we must decide upon a function $\phi_i(z^i)$ to represent our beliefs about the Bayesian prior of the target weight vector. In many cases such prior knowledge is readily described. For example, the notion of sparsity can be captured by the supposition that with high probability the tap will have have a small value, while with low probability the tap will have a large value. One prior that implies sparsity is

$$\phi(z) = \frac{1}{\sqrt{z}}, \tag{5.6}$$

since it is large for small z and small for large z. As will be shown in Section 6.2, this prior leads to algorithm (3.2). Furthermore, because all the functions are "diagonal" with respect to the weight vector (i.e., each ϕ_i is a function of z^i only, not the other z^k's), the priors can be chosen independently for different taps. If we think particular taps are likely to be large (for example, the taps near the cursor in a channel model), we could use a uniform prior, $\phi_i(z^i) = 1$, for those taps, and a prior such as (5.6) for the remaining taps.

Suppose ϕ is a probability density function representing prior beliefs about a parameter. Then (5.3) is the NG algorithm that incorporates this prior, and (5.5) is the first order ANG, both assuming that the parameterization function γ of (5.2) is the identity. The same ANG algorithm can also be derived (or interpreted) as a Euclidean gradient descent ($\phi(z) = 1$) on a modified cost function (i.e. $\gamma(z)$ not necessarily the identity). That is, given an algorithm in the form of (5.5), it is possible to find a cost function and reparameterization γ on which a Euclidean gradient descent operates to give the same updates to w. This is a bit subtle – since the parameterizations are different, the algorithms may be evolving over different z-spaces, but the effective evolution of the parameters of $\mathbf{w}$ will be the same. (Refer to Figures 3.1 and 3.2 to see the difference between w-space and z-space more clearly.) Thus, the prior can be stated directly in terms of ϕ, indirectly in terms of γ, or as some combination of the two.

To make this precise, suppose that a prior ϕ and a reparameterization function γ are given, which correspond to an NG algorithm and its ANG. Then there are other priors and parameterizations which also generate this same ANG.

Proposition 5.0.1 *Let ϕ and γ represent the priors and parameterizations of an ANG algorithm (5.5) with $\hat{y}$ parameterized as in (5.1), and let the cost function be given by $L(y, \hat{y})$. If there are functions $\bar{\gamma}$ and $\bar{\phi}$ with*

$$\frac{\dot{\gamma}^2}{\phi^2} = \frac{\dot{\bar{\gamma}}^2}{\bar{\phi}^2}, \tag{5.7}$$

then $\bar{\gamma}$ and $\bar{\phi}$ are an alternate set of priors and parameterizations that yield the same effective update rule for $\mathbf{w}$.

Proof: From (5.5), the ANG corresponding to ϕ, γ, and $L(y, \hat{y})$ is

$$z^i_{k+1} = z^i_k - \mu \, \frac{\partial L}{\partial z^i_k} \, \frac{1}{\phi^2_i(z^i_k)}.$$

Applying the chain rule yields

$$z^i_{k+1} = z^i_k - \mu \, \frac{\partial L}{\partial \hat{y}_k} \, \frac{\partial \hat{y}_k}{\partial z^i_k} \, \frac{1}{\phi^2_i(z^i_k)}.$$

Since $\frac{\partial \hat{y}_k}{\partial z^i_k} = \dot{\gamma}(z^i_k) x^i_k$, this becomes

$$z^i_{k+1} = z^i_k - \mu \, \frac{\partial L}{\partial \hat{y}_k} \, \frac{\dot{\gamma}(z^i_k)}{\phi^2_i(z^i_k)} \, x^i_k.$$

Note that γ appears implicitly in this equation via L. For example, if $L = (y_k - \hat{y}_k)^2$, then since $\hat{y}_k$ is a function of γ, so is L. Since for any choice of priors and parameterizations, y is generated according to $\hat{y}_k = \sum_i w^i_k x^i_k$, what we ought to compare is the effective change of $\mathbf{w}$ as $\mathbf{z}$ changes. Note that

$$w^i_{k+1} = \gamma(z^i_{k+1}) = \gamma(z^i_k + \textit{small term}),$$

so a first order Taylor expansion gives

$$w^i_{k+1} = \gamma(z^i_k) + \dot{\gamma}(z^i_k)(\textit{small term}), \tag{5.8}$$

which results in

$$w_{k+1}^i = w_k^i + \mu \frac{\partial L}{\partial \hat{y}_k} \; x_k^i \left(\frac{\dot{\gamma}^2(z_k^i)}{\phi_i^2(z_k^i)} \right). \tag{5.9}$$

Similarly, the ANG correponding to $\bar{\gamma}$ and $\bar{\phi}$ is

$$w_{k+1}^i = w_k^i + \mu \frac{\partial L}{\partial \hat{y}_k} \; x_k^i \left(\frac{\dot{\bar{\gamma}}^2(z_k^i)}{\bar{\phi}_i^{\,2}(z_k^i)} \right).$$

Since $\frac{\dot{\gamma}^2}{\phi^2} = \frac{\dot{\bar{\gamma}}^2}{\bar{\phi}^2}$ by assumption, these are the same algorithm. Δ

Note that the left-hand and right-hand sides of (5.7) are both functions of z, yet z is different for each side since the parameterization is different. Thus, both sides of the equation must be separately represented as functions of w, then compared. For example, consider the motivating example of Section(3.1). The algorithm as initially presented in equation (3.2) can be thought of as having the standard parameterization $w = \gamma_1(z) = z$ and prior $\phi_1(z) = \frac{1}{\sqrt{z}}$. Thus,

$$\frac{\dot{\gamma_1}^2}{\phi_1^2} = \frac{1}{\phi_1^2(z)} = \frac{1}{\phi_1^2(w)} = w.$$

On the other hand, if we reparameterize w via $\gamma_2(z) = \frac{1}{4}z^2$ and use the uniform prior of $\phi_2(z) = 1$, we have

$$\frac{\dot{\gamma_2}^2}{\phi_2^2} = \dot{\gamma_2}^2(z) = \frac{1}{4}z^2 = w.$$

Note that these two ratios used z differently, but in terms of w, the ratios are the same. Consequently, the update rules given by equations (3.2) and (3.9) were the same.

In the course of our proof, we have shown that algorithms incorporating priors and reparameterizations can be written in the general form of

$$w_{k+1}^i = w_k^i + \mu \frac{\partial L}{\partial \hat{y}_k} \; x_k^i \; [D_k]_{ii},$$

where D_k is given by

$$[D_k]_{i,j} = \left(\frac{\partial \gamma_i}{\partial (z_k^j)}\right)^2 \left(\frac{1}{\phi_i(z_k^j)}\right)^2. \tag{5.10}$$

This shows how ANG algorithms can be analyzed using the framework of Chapter 4. Also note the similarity between D_k and what Equation (15) of [22] calls the Jacobian. The difference is that the Jacobian in [22] only includes the first factor from (5.10), whereas our use here is intended to capture the effects of the prior as well.

Using this proposition, the ANG may be derivable from an alternative prior ϕ using the standard parameterization $\gamma(z) = z$. This prior will be called the 'true' prior because it represents the prior beliefs without the confounding influence of the reparameterization function. Alternatively, the ANG may be derivable from a reparameterization using the standard prior $\phi = 1$ (which corresponds to a belief that all values of the parameter are equally likely). In this case, γ can be used to give the cost function over which the algorithm is evolving (in z-space) under the standard Euclidean gradient. These are useful because sometimes it is easier to understand the behavior of an algorithm from the point of view of priors, while sometimes it is easier from the perspective of the cost function. Imposition of the requirement that $\frac{\dot{\gamma}^2}{\phi^2}$ be equal allows translation of the same algorithm to both frameworks. This is an important feature of the ANG algorithms, since this translation is not possible with the NG algorithms.

Even so, this proposition should be used with caution. When we obtained equation (5.8), we ignored terms of order μ^2 or higher. However, if $\dot{\gamma}(z_k^i)$ is small compared to μ, which often occurs near $\mathbf{z} = \mathbf{0}$, the higher order terms are of comparable magnitude to the first order term. Thus, in such a situation, the approximations used in the proposition are invalid. If we focus on algorithms that have a parameterization function of $\gamma(z) = z$, we do not need to make such approximations, and our analysis of the algorithms in w-space is exact; whereas if $\gamma(z) \neq z$, then when $\dot{\gamma}(z_k^i)$ is small our analysis of the algorithms in w-space is not as accurate.

A third possible interpretation of NG algorithms (following Amari and Douglas [4], [16]) considers the function $G^{-1}(z)$ (which is equivalent to $\dot{\gamma}(z)$) to be a Riemannian metric of distance measure imposed on the underlying space. This can be used, for instance, when signal measurements are corrupted by additive noise (and the probability distribution function of the noise is known) to create a Fisher-efficient learning algorithm. The main difference between this thesis and the work done by Amari and Douglas is that they explicitly assume they know the likelihood function of the noise, and then effectively use the uniform prior ($\phi = 1$) and standard parameterization ($\gamma = z$); whereas this thesis makes no assumptions about the noise, but uses non-standard priors and/or parameterizations.

Chapter 6

The Algorithms

This chapter derives a number of algorithms. In most cases, the natural gradient NG algorithm (5.3), the ANG algorithm (5.5), the cost function for the equivalent Euclidean gradient (correponding to $\phi = 1$), and the true prior (corresponding to $\dot{\gamma} = 1$) are given. The first two examples show how special choices of γ and ϕ result in familiar algorithms, while the remaining sections derive new algorithms specifically for signal processing and communications problems when prior information (such as sparsity) is available. The new algorithms include blind CMA and DD algorithms, and an ARMA model algorithm that corresponds to the "equation error" form [24] of standard ARMA model adaptive identifiers.

Tables 6.1 and 6.2 display the parameterizations, priors, costs, and update rules for most of the algorithms discussed in this section. However, several of the update rules were too large to fit into a table, and instead references are made to the equation numbers in the text; and some of the update rules were so unwieldy and complicated that they were not even included in the text.

Table 6.1: Algorithms using the standard cost function of $(y_k - \hat{y}_k)^2$ as in (3.4). Note that when $\phi(z) = 1$, the NG and the ANG updates are identical. Also note that these algorithms are in z-space, but when $\gamma(z) = z$, the w-space updates are the same.

Algorithm Name	$\gamma(z)$**,** $\phi(z)$ **pairs**	**NG update**	**ANG update**
LMS	z, 1	$z^i_{k+1} = z^i_k + \mu\ (y_k - \hat{y}_k)\ x^i_k$	$z^i_{k+1} = z^i_k + \mu\ (y_k - \hat{y}_k)\ x^i_k$
(see note in text)	$\ln(z)$, $\frac{1}{z}$	$z^i_{k+1} = z^i_k\ \exp(\mu(y_k - \hat{y}_k)x^i_k)$	$z^i_{k+1} = z^i_k + \mu\ z^i_k\ (y_k - \hat{y}_k)\ x^i_k$
EG natural [30]	z, $\frac{1}{z}$	$z^i_{k+1} = z^i_k \exp(\mu\ z^i_k\ (y_k - \hat{y}_k)\ x^i_k)$	$z^i_{k+1} = z^i_k + \mu\ (z^i_k)^2 (y_k - \hat{y}_k)\ x^i_k$
(EGN)	$\exp(z)$, 1	$z^i_{k+1} = z^i_k + \mu\ \exp(z^i_k)\ (y_k - \hat{y}_k)\ x^i_k$	$z^i_{k+1} = z^i_k + \mu\ \exp(z^i_k)\ (y_k - \hat{y}_k)\ x^i_k$
Sparse LMS	z, $\frac{1}{\sqrt{z}}$	$z^i_{k+1} = \left(\sqrt{z^i_k} + \frac{1}{2}\mu\sqrt{z^i_k}(y_k - \hat{y}_k)\ x^i_k\right)^2$	$z^i_{k+1} = z^i_k + \mu\ z^i_k\ (y_k - \hat{y}_k)\ x^i_k$
(SLMS)	$\frac{1}{4}z^2$, 1	$z^i_{k+1} = z^i_k + \mu\ (\frac{1}{2}z^i_k)\ (y_k - \hat{y}_k)\ x^i_k$	$z^i_{k+1} = z^i_k + \mu\ (\frac{1}{2}z^i_k)\ (y_k - \hat{y}_k)\ x^i_k$
Signed Sparse LMS	z, $\frac{1}{\sqrt{\lvert z\rvert+\epsilon}}$	too complicated to derive	$z^i_{k+1} = z^i_k + \mu(\lvert z^i_k\rvert + \epsilon)(y_k - \hat{y}_k)\ x^i_k$
(SSLMS)	$\frac{1}{4}\mathrm{sgn}(z)z^2 + \sqrt{\epsilon}z$, 1	$z^i_{k+1} = z^i_k + \mu(\frac{1}{2}\lvert z^i_k\rvert + \sqrt{\epsilon})(y_k - \hat{y}_k)x^i_k$	$z^i_{k+1} = z^i_k + \mu(\frac{1}{2}\lvert z^i_k\rvert + \sqrt{\epsilon})(y_k - \hat{y}_k)x^i_k$
Fractional LMS	z, $\frac{1}{z^\alpha}$ $(\alpha \neq 1)$	refer to Equation (6.14)	$z^i_{k+1} = z^i_k + \mu(z^i_k)^{2\alpha}(y_k - \hat{y}_k)x^i_k$
(FLMS)	$((1-\alpha)z)^{\frac{1}{1-\alpha}}$, 1	$z^i_{k+1} = z^i_k + \mu((1-\alpha)z)^{\frac{\alpha}{1-\alpha}}(y_k - \hat{y}_k)x^i_k$	$z^i_{k+1} = z^i_k + \mu((1-\alpha)z)^{\frac{\alpha}{1-\alpha}}(y_k - \hat{y}_k)x^i_k$
Signed Frac. LMS	see Eq. (6.17), 1	refer to Equation (6.18)	refer to Equation (6.18)
IPL LMS	z, $\frac{c}{\lvert z\rvert^\alpha+\epsilon}$	too complicated to derive	$z^i_{k+1} = z^i_k + \mu(\lvert z^i_k\rvert^\alpha + \epsilon)^2(y_k - \hat{y}_k)x^i_k$
Exponential LMS	z, $c\ e^{-a\lvert z\rvert}$	too complicated to derive	$z^i_{k+1} = z^i_k + \mu\ e^{2\alpha\lvert z^i_k\rvert}(y_k - \hat{y}_k)x^i_k$

Table 6.2: Algorithms using miscellaneous cost functions. When $\phi(z) = 1$, the NG and ANG updates are identical.

Algorithm Name	$\gamma(z)$, $\phi(z)$ **pairs**	**NG update**	**ANG update**
	Algorithms using the CMA cost function: $L = \frac{1}{4}(\hat{y}_k^2 - c)^2$		
CMA	w, 1	$z_{k+1}^i = z_k^i - \mu\ \hat{y}_k(\hat{y}_k^2 - c)\ x_k^i$	$z_{k+1}^i = z_k^i - \mu\ \hat{y}_k(\hat{y}_k^2 - c)\ x_k^i$
Signed Sparse CMA	z, $\frac{1}{\lvert z\rvert+\epsilon}$	$z_{k+1}^i = z_k^i \exp(-\mu(\lvert z_k^i\rvert + \epsilon)\hat{y}_k(\hat{y}_k^2 - c)x_k^i)$	$z_{k+1}^i = z_k^i - \mu(\lvert z_k^i\rvert + \epsilon)^2\hat{y}_k(\hat{y}_k^2 - c)x_k^i$
Fractional CMA	z, $\frac{1}{z^\alpha}$	refer to Equation(6.21)	$z_{k+1}^i = z_k^i - \mu(z_k^i)^{2\alpha}\hat{y}_k(\hat{y}_k^2 - c)x_k^i$
	$((1-\alpha)z)^{\frac{1}{1-\alpha}}$, 1	$z_{k+1}^i = z_k^i - \mu((1-\alpha)z)^{\frac{\alpha}{1-\alpha}}\hat{y}_k(\hat{y}_k^2 - c)x_k^i$	$z_{k+1}^i = z_k^i - \mu((1-\alpha)z)^{\frac{\alpha}{1-\alpha}}\hat{y}_k(\hat{y}_k^2 - c)x_k^i$
Signed Frac. CMA	see Eq. (6.17), 1	refer to Equation (6.22)	refer to Equation (6.22)
	Algorithms using the DD cost function: $L = \frac{1}{2}(Q(\hat{y}_k) - \hat{y}_k)^2$		
DD-LMS	z, 1	$z_{k+1}^i = z_k^i + \mu\ (Q(\hat{y}_k) - \hat{y}_k)\ x_k^i$	$z_{k+1}^i = z_k^i + \mu\ (Q(\hat{y}_k) - \hat{y}_k)\ x_k^i$
Sparse DD-LMS	z, $\frac{1}{\sqrt{\lvert z\rvert+\epsilon}}$	too complicated to derive	$z_{k+1}^i = z_k^i + \mu(\lvert z_k^i\rvert + \epsilon)(Q(\hat{y}_k) - \hat{y}_k)\ x_k^i$
	$\frac{1}{4}\mathrm{sgn}(z)z^2 + \sqrt{\epsilon}z$, 1	$z_{k+1}^i = z_k^i + \mu(\frac{1}{2}\lvert z_k^i\rvert + \sqrt{\epsilon})(Q(\hat{y}_k) - \hat{y}_k)x_k^i$	$z_{k+1}^i = z_k^i + \mu(\frac{1}{2}\lvert z_k^i\rvert + \sqrt{\epsilon})(Q(\hat{y}_k) - \hat{y}_k)x_k^i$
	Algorithms using ARMA model parameterization with $L = \frac{1}{2}(y_k - \hat{y}_k)^2$		
Equation Error	z, 1	$z_{k+1}^i = z_k^i + \mu\ (y_k - \hat{y}_k)\ y_{k-i}$	same as NG update
		$z_{k+1}^{j+n} = z_k^{j+n} + \mu\ (y_k - \hat{y}_k)\ x_{k-j}$	same as NG update
Sparse Equation Error	$\frac{1}{4}\mathrm{sgn}(z)z^2 + \sqrt{\epsilon}z$, 1	$z_{k+1}^i = z_k^i + \mu\ (y_k - \hat{y}_k)\ y_{k-i}\ (\lvert z_k^i\rvert + \epsilon)$	same as NG update
		$z_{k+1}^{j+n+1} = z_k^{j+n+1} + \mu\ (y_k - \hat{y}_k)\ x_{k-j}\ (\lvert z_k^i\rvert + \epsilon)$	same as NG update

6.1 Least Mean Squares

The prior belief that corresponds to the LMS algorithm is that all parameters are equally likely to achieve any value. Hence $\phi_i(z) = 1$, $\Phi_i(z) = z$, and $\Phi_i^{-1}(v) = v$. The appropriate cost function is the mean square cost $L(y_k, \hat{y}_k) = (y_k - \hat{y}_k)^2$, and the parameterization within $\hat{y}_k$ is the standard one ($\gamma(z) = z$). Hence $\frac{\dot{\gamma}(z)}{\phi(z)} = 1$ and $\frac{\partial L}{\partial w_k^i} = 2(y_k - \hat{y}_k)x_k^i$. Substituting into (5.3) gives exactly (3.10), the LMS algorithm. Substituting into (5.5) also gives exactly (3.10). In this case, because $\phi(z) = 1$, there is no difference between the NG and the ANG algorithms, and both reduce to the well known LMS algorithm.

Now consider the reparameterization $\gamma(z) = \ln(z)$ with the prior $\phi(z) = \frac{1}{z}$. This again gives $\frac{\dot{\gamma}(z)}{\phi(z)} = 1$, so we expect that this will also yield the LMS algorithm. Substituting into (5.3) gives

$$z_{k+1}^i = \exp(\ln(z_k^i) + \mu\ (y_k - \hat{y}_k)\ x_k^i) = z_k^i\ \exp(\mu\ (y_k - \hat{y}_k)\ x_k^i).$$

From (5.5), the corresponding ANG algorithm is

$$z_{k+1}^i = z_k^i + \mu\ z_k^i\ (y_k - \hat{y}_k)\ x_k^i. \tag{6.1}$$

Now using a Taylor expansion as in (3.8), we get the effective w update of

$$w_{k+1}^i = w_k^i + \mu\ (y_k - \hat{y}_k)\ x_k^i,$$

which is just LMS again, as expected. Of course, implementing LMS in this reparameterized, re-prioritized fashion would be impractical, but this serves to illustrate the equivalence of two algorithms which seem very different on the surface.

6.2 Exponentiated Gradient Algorithm

Consider the prior $\phi_i(z) = \frac{1}{z}$, which corresponds to a high likelihood that the parameter value is small and a small chance that the parameter is large. Accordingly, $\Phi_i(z) = \ln(z)$, and $\Phi_i^{-1}(v) = \exp(v)$. Using the standard cost $L(y_k, \hat{y}_k)$ of (3.4) and the standard parameterization $\gamma(z) = z$ gives the NG algorithm

$$z_{k+1}^i = \exp(\ln(z_k^i) + \mu(y_k - \hat{y}_k)x_k^i z_k^i)$$

which can be rewritten

$$z_{k+1}^i = z_k^i \exp(\mu\ z_k^i\ (y_k - \hat{y}_k)\ x_k^i). \tag{6.2}$$

This is called the exponentiated gradient algorithm in [30]. The associated ANG algorithm is

$$z_{k+1}^i = z_k^i + \mu\ (z_k^i)^2\ (y_k - \hat{y}_k)\ x_k^i. \tag{6.3}$$

An equivalent way to derive this ANG algorithm is to let $\gamma(z) = \exp(z)$ and $\phi(z) = 1$. In this case, both the NG and ANG algorithms are

$$z_{k+1}^i = z_k^i + \mu \exp(z_k^i)\ (y_k - \hat{y}_k)\ x_k^i. \tag{6.4}$$

To see the equivalence of (6.3) and (6.4), convert them into w-space. (6.3) is already in that format (since there we had $\mathbf{w} = \mathbf{z}$), so consider expanding (6.4) in a Taylor series as in (3.8). This yeilds

$$w_{k+1}^i = w_k^i + \mu\ (\exp(z_k^i))^2\ (y_k - \hat{y}_k)\ x_k^i.$$

Since $w = \gamma(z) = \exp(z)$, this is effectively

$$w_{k+1}^i = w_k^i + \mu\ (w_k^i)^2\ (y_k - \hat{y}_k)\ x_k^i,$$

which is the same as equation (6.3).

Now consider an algorithm with $\gamma(z) = z$ and a prior $\phi(z) = \frac{1}{\sqrt{z}}$. The prior is qualitatively similar to the previous example, but the algorithm will be slightly different. The NG update is

$$z^i_{k+1} = \left(\sqrt{z^i_k} + \frac{1}{2}\mu\sqrt{z^i_k}(y_k - \hat{y}_k)x^i_k\right)^2,$$

and the simpler ANG update is

$$z^i_{k+1} = z^i_k + \mu\ z^i_k\ (y_k - \hat{y}_k)\ x^i_k. \tag{6.5}$$

Although this looks just like (6.1), it is not, because z means something different here (due to a different γ function). This is exactly the example introduced in section 3.1, since $\mathbf{w} = \mathbf{z}$. An equivalent way to derive this ANG algorithm comes from setting $\gamma(z) = \frac{1}{4}z^2$ and $\phi(z) = 1$. This leads to NG and ANG algorithms given by

$$z^i_{k+1} = z^i_k + \mu\ (\frac{1}{2}z^i_k)\ (y_k - \hat{y}_k)\ x^i_k.$$

Converting the ANG algorithm into w-space as in (3.8) gives

$$w^i_{k+1} = w^i_k + \mu\ (\frac{1}{2}z^i_k)^2\ (y_k - \hat{y}_k)\ x^i_k. \tag{6.6}$$

Since $(\frac{1}{2}z^i_k)^2 = \frac{1}{4}(z^i_k)^2 = w^i_k$, equations (6.5) and (6.6) are the same in w-space. This is what was derived in section 3.1, although the framework here is more formal.

In Section 6.3, we will modify the cost function (via $\gamma(z)$) to allow both positive and negative weights. Using the duality of priors and parameterizations, we will show how such modifications affect the priors.

6.3 Practical LMS for Sparsity

Prior information about a target weight vector may arise from first principles. For instance, a channel may be known to be sparse because it consists of a number of discrete reflections well separated in space or time. Prior knowledge may also arise as a result of a statistical study showing, for example, that a particular equalizer often converges to a configuration in which most taps are near zero. However it arises, it will be necessary to model the a priori information in simple functional form in order to utilize it in algorithm design.

Suppose, for instance, that a prior form of $\phi(z) = z^{-1/2}$ appears to fit well. Then the discussion in Section 6.2 suggests that an algorithm of the general form of (3.2) may be particularly effective. As discussed in Section 3, this is equivalent to a Euclidean gradient descent in z-space with $\gamma(z) = \frac{1}{4}z^2$. As discussed in Section 3.2 (and in [29]), in order to allow positive and negative coefficients as well as to forbid the algorithm from becoming locked at $w = 0$, the parameterization can modified to be

$$\gamma(z) = \frac{1}{4}\mathrm{sgn}(z)\ (z)^2 + \sqrt{\epsilon}z \tag{6.7}$$

where $\epsilon > 0$. What is the effect of this modification?

Since $\dot{\gamma}(z) = \frac{1}{2}|z| + \sqrt{\epsilon} > 0$, this is always an increasing function, and equilibrium in the corresponding ANG algorithm can only occur when $y_k = \hat{y}_k$ (or in the degenerate case when the input $x = 0$). Specifically, ϵ keeps the update term from vanishing for small z, which would have prevented coefficients from changing sign. Now consider the question of how such ad hoc modifications influence the sparsity prior. The parameterization given by (6.7) and a Euclidean gradient over z ($\phi(z) = 1$) is equivalent to $\dot{\bar{\gamma}}(z) = 1$ and $\bar{\phi}(z) = \sqrt{\frac{1}{|z|+\epsilon}}$ by Proposition 5.0.1.

This can be shown as follows. Note that

$$(\dot{\gamma}(z))^2 = \frac{1}{4}|z|^2 + \sqrt{\epsilon}|z| + \epsilon, \tag{6.8}$$

and

$$|\gamma| = \frac{1}{4}|z|^2 + \sqrt{\epsilon}|z|. \tag{6.9}$$

Thus,

$$\left(\frac{\dot{\gamma}}{\phi}\right)^2 = (\dot{\gamma}(z))^2 = |\gamma| + \epsilon, \tag{6.10}$$

and

$$\left(\frac{\dot{\bar{\gamma}}}{\bar{\phi}}\right)^2 = |w| + \epsilon. \tag{6.11}$$

If we choose $\bar{\gamma}(z) = z$, then $\dot{\bar{\gamma}} = 1$ and $w = z$, so

$$\frac{1}{(\bar{\phi}(z))^2} = |z| + \epsilon. \tag{6.12}$$

This yields $\bar{\phi}(z) = \sqrt{\frac{1}{|z|+\epsilon}}$. Compared to the previous prior of $\phi(z) = \sqrt{\frac{1}{z}}$, there is little difference. The ϵ is a small modification that only changes the algorithm near $z = 0$. The NG and ANG algorithms are given in Table 6.1 under the headings "Sparse LMS" and "Signed Sparse LMS".

Of course, analogous designs result from other choices of priors. For instance, the parameterization $\gamma(z) = \frac{1}{3}z^3$ leads to the Euclidean descent algorithm

$$z_{k+1}^i = z_k^i + \mu\ (z_k^i)^2\ (y_k - \hat{y}_k)\ x_k^i. \tag{6.13}$$

As with all Euclidean gradient algorithms, $\phi(z) = 1$. To determine the corresponding sparsity prior, invoke (5.7) to obtain the equivalent ANG algorithm. This gives $\dot{\gamma} = 1$ and $\phi = (3|z|)^{2/3}$.

For the general case of fractional power priors, consider $\phi(z) = \frac{1}{z^\alpha}$, for $\alpha \neq 1$, for which $\Phi(z) = \frac{1}{1-\alpha} z^{1-\alpha}$ and $\Phi^{-1}(v) = ((1-\alpha)v)^{\alpha-1}$. Under the standard parameterization $\gamma(z) = z$, the NG algorithm is

$$z_{k+1}^i = (1-\alpha)\left(\frac{1}{1-\alpha}(z_k^i)^{1-\alpha} + \mu \; (y_k - \hat{y}_k) \; x_k^i \; (z_k^i)^\alpha \; \right)^{\alpha-1} \tag{6.14}$$

while the ANG algorithm is

$$z_{k+1}^i = z_k^i + \mu \; (z_k^i)^{2\alpha} \; (y_k - \hat{y}_k) \; x_k^i. \tag{6.15}$$

By Proposition 5.0.1, this corresponds to a Euclidean gradient descent with $\bar{\phi}(z) = 1$ and some $\bar{\gamma}(z)$. To find $\bar{\gamma}(z)$, note that

$$\frac{\dot{\gamma}}{\phi} = z^\alpha = w^\alpha.$$

Since $\bar{\phi}(z) = 1$,

$$\dot{\bar{\gamma}} = w^\alpha = \bar{\gamma}^\alpha,$$

$$\frac{d\bar{\gamma}}{\bar{\gamma}^\alpha} = dz,$$

$$z = \frac{\bar{\gamma}^{1-\alpha}}{1-\alpha},$$

giving the paramererization function

$$\bar{\gamma}(z) = ((1-\alpha)z)^{1/(1-\alpha)}. \tag{6.16}$$

Both (6.14) and (6.15) are only appropriate for positive targets weights w. To create a practical algorithm, the parameterization can be modified to

$$\gamma(z) = \mathrm{sgn}(z)((1-\alpha)|z|)^{1/(1-\alpha)} + \epsilon z, \tag{6.17}$$

which corresponds to the algorithm (both NG and ANG)

$$z_{k+1}^i = z_k^i + \mu \; (y_k - \hat{y}_k) \; x_k^i \; (((1-\alpha)|z_k^i|)^{\frac{\alpha}{1-\alpha}} + \epsilon). \tag{6.18}$$

Understandably, it might be undesireable to implement an algorithm this complicated.

Observe that all the above derivations occur for individual weights. Hence, if different priors are available for different weights, then different algorithm updates can be used. This might be useful, say, in an equalizer design problem, where the center taps are likely to be large while the tails of the equalizer are likely to be small and sparse.

6.4 Blind Adaptation with CMA

All the algorithms of the previous sections assumed that the cost function $L(y_k, \hat{y}_k)$ was the standard mean squared error function (3.4), albeit with various modifications to the parameterization $\gamma(z)$ within $\hat{y}_k$. However, the analysis is not limited to this cost function, which means that a number of other algorithms can be treated analagously.

In many communications problems the transmitted signal consists of elements taken from a finite alphabet, and this property can be exploited to direct the adaptation. One popular method is the Constant Modulus Algorithm (CMA) [36] with cost function

$$L(y_k, \hat{y}_k) = \frac{1}{4}(\hat{y}_k^2 - c)^2 \tag{6.19}$$

where c is a constant appropriate for the given constellation. The most useful aspect of (6.19) is that it leads to algorithms that do not require knowledge of y_k, and which can be used to adapt the weights even in the absence of a training signal. Such algorithms are called 'blind'. This section will show how the NG and ANG strategies, in combination with (6.19), can be used to derive blind algorithms in which the updates are explicitly designed to exploit prior knowledge (such as sparsity) of the weights.

To create an NG or ANG algorithm, we need a cost function L, a parameterization γ, and a prior ϕ. The cost function is given by (6.19). For the other two functions, the simplest case is the standard parameterization, $\gamma(z) = z$, and the uniform prior, $\phi(z) = 1$, which corresponds to $\Phi(z) = z$ and $\Phi^{-1}(v) = v$. This leads to NG and ANG algorithms that are both given by

$$z^i_{k+1} = z^i_k - \mu \ \hat{y}_k(\hat{y}^2_k - c) \ x^i_k. \tag{6.20}$$

Note that (6.20) is the standard CMA algorithm of [36].

Suppose, however, that prior knowledge suggests a prior of $\phi(z) = \frac{1}{z}$. Then the NG and ANG algorithms will differ from CMA and from each other. The NG algorithm is

$$z^i_{k+1} = w^i_k \exp(-\mu \ \hat{y}_k(\hat{y}^2_k - c) \ x^i_k \ z^i_k)$$

while the simpler ANG update gives

$$z^i_{k+1} = z^i_k - \mu \ \hat{y}_k(\hat{y}^2_k - c) \ x^i_k \ (z^i_k)^2.$$

Both of these are blind in the sense of not requiring knowledge of the (training) sequence y_k, and both are tailored to the prior ϕ. As in Section 6.3, to be practical, these updates must be modified so as to allow both positive and negative weights, and to remove the possibility of becoming locked at $z = 0$. Modifying the prior to $\phi(z) = \frac{1}{|z|+\epsilon}$ results in the ANG algorithm

$$z^i_{k+1} = z^i_k - \mu \ \hat{y}_k(\hat{y}^2_k - c) \ x^i_k \ (|z^i_k| + \epsilon)^2$$

Equivalently, Proposition 5.0.1 shows that this algorithm results from a Euclidean descent in z-space on the cost function (6.19) with $\phi(z) = 1$ and the reparameterization $\gamma(z)$ given by (6.7).

More generally for fractional power priors, $\phi(z) = \frac{1}{z^\alpha}$, for $\alpha \neq 1$, $\Phi(z) = \frac{1}{1-\alpha} z^{1-\alpha}$, and $\Phi^{-1}(v) = ((1-\alpha)v)^{\alpha-1}$. Under the standard parameterization $\gamma(z) = z$ and the cost function (6.19), the NG algorithm is

$$z_{k+1}^i = (1-\alpha)(\frac{1}{1-\alpha}(z_k^i)^{1-\alpha} - \mu \; \hat{y}_k(\hat{y}_k^2 - c) \; x_k^i \; (z_k^i)^\alpha \;)^{\alpha-1} \tag{6.21}$$

while the ANG algorithm is

$$z_{k+1}^i = z_k^i - \mu \; \hat{y}_k(\hat{y}_k^2 - c) \; x_k^i \; (z_k^i)^{2\alpha}.$$

By Proposition 5.0.1, this corresponds to a Euclidean gradient descent with $\bar{\phi}(z) = 1$ and $\bar{\gamma}(z)$ given by (6.16). Again, both of these algorithms are only appropriate for positive targets weights z. To create a practical algorithm, the parameterization can be modified as in (6.17), producing the NG and ANG algorithms

$$z_{k+1}^i = z_k^i - \mu \; \hat{y}_k(\hat{y}_k^2 - c) \; x_k^i \; (((1-\alpha)|z_k^i|)^{\frac{\alpha}{1-\alpha}} + \epsilon). \tag{6.22}$$

However, the true prior is too difficult to compute in this case.

Variations on $L(y_k, \hat{y}_k)$ are also common, perhaps the most obvious being to generalize (6.19) to

$$L(y_k, \hat{y}_k) = \frac{1}{pq} ||\hat{y}_k|^p - c|^q$$

where p and q take on various values. Again, this can be combined with prior information leading to a generalized CMA for sparsity. Using the parameterization of (6.17) yields the ANG algorithm

$$z_{k+1}^i = w_k^i - \mu \; |\hat{y}_k|^{p-1} ||\hat{y}_k|^p - c|^{q-1} \; x_k^i \; (((1-\alpha)|z_k^i|)^{\frac{\alpha}{1-\alpha}} + \epsilon). \tag{6.23}$$

Again, deriving the true prior and corresponding algorithms is very messy and computationally intense.

All these algorithms can also be generalized readily to the complex case, and they may be designed so that different weights have different priors, and hence different update rules matched to those priors.

6.5 Blind Adaptation with DD

Another important class of algorithms are "decision directed", blind algorithms designed for use with a finite alphabet. The DD algorithm can be viewed as a Euclidean descent over the cost function

$$L(y_k, \hat{y}_k) = \frac{1}{2}(Q(\hat{y}_k) - \hat{y}_k)^2 \tag{6.24}$$

where $Q(\cdot)$ quantizes the argument to the nearest symbol in the alphabet. (In such a derivation, it is necessary to ignore the fact that $Q(\cdot)$ is discontinuous, and to formally replace its derivative with zero.) The simplest case is the uniform prior $\phi(z) = 1$ and standard parameterization $\gamma(z) = z$, which lead to NG and ANG algorithms that are identical, and are given by

$$z_{k+1}^i = z_k^i + \mu\ (Q(\hat{y}_k) - \hat{y}_k)\ x_k^i. \tag{6.25}$$

Note that (6.25) is the standard DD algorithm of [21].

For the sparse prior $\phi(z) = \frac{1}{\sqrt{|z|+\epsilon}}$ and the standard parameterization, the ANG algorithm is

$$z_{k+1}^i = z_k^i + \mu\ (|z_k^i| + \epsilon)\ (Q(\hat{y}_k) - \hat{y}_k)\ x_k^i,$$

which corresponds to a Euclidean descent on a cost function defined by (6.24) with reparameterization

$$\gamma(z) = \frac{1}{4}\ \mathrm{sgn}(z)|z|^2 + \sqrt{\epsilon}z.$$

Other priors lead in analogous fashion to other update rules, and other modifications such as replacing the power of 2 in the cost function (6.24) with more general powers p are equally straightforward. In general, DD NG and DD ANG algorithms can be obtained by replacing $y(k)$ by $Q(\hat{y}_k)$ in any algorithm that has the same prior and parameterization, but uses the standard cost $L(y_k, \hat{y}_k)$ of (3.4) instead of (6.24).

6.6 Parameterizations for ARMA Model Adaptation

Many system modelling and signal processing applications require that estimates be made of autoregressive (as well as moving average) parameters. To be specific, suppose that the data stream arises from a model of the form

$$y_k = \sum_{i=1}^{n} (w^*)^i y_{k-i} + \sum_{j=0}^{m} (w^*)^{j+n+1} x_{k-j},$$

where the actual values of the $(w^*)^i$ are unknown. One approach is to build an estimator of the form

$$\hat{y}_k = \sum_i w_k^i y_{k-i} + \sum_j w_k^{j+n+1} x_{k-j}$$

and to try an minimize an error $L(y_k, \hat{y}_k)$ by choice of the weights. This can be accomplished by the "equation error" strategy [24], which is a Euclidean gradient on the mean square error cost function (3.4). As before, we will define γ by

$$w_k^i = \gamma(z_k^i)$$

for some parameters z_k^i. For the uniform prior $\phi(z) = 1$ and the standard parameterization $\gamma(z) = z$, both the NG and the ANG give the standard algorithm

$$z_k^i = z_{k-1}^i + \mu\ (y_k - \hat{y}_k)\ y_{k-i} \qquad i = 1, 2, ..., n$$

$$z_k^{j+n+1} = z_{k-1}^{j+n+1} + \mu\ (y_k - \hat{y}_k)\ x_{k-j} \qquad j = 0, 1, ..., m.$$

As in the previous sections, nonuniform priors can be readily incorporated. For example, for the sparse prior $\phi(w) = \frac{1}{\sqrt{|w|+\epsilon}}$ and the standard parameterization, the ANG algorithm consists of

$$z_k^i = z_{k-1}^i + \mu\ (y_k - \hat{y}_k)\ y_{k-i}(|z_{k-1}^i| + \epsilon) \qquad i = 1, 2, ..., n$$

$$z_k^{j+n+1} = z_{k-1}^{j+n+1} + \mu \; (y_k - \hat{y}_k) \; x_{k-j}(|z_{k-1}^{j+n+1}| + \epsilon) \qquad j = 0, 1, ..., m$$

which corresponds to a Euclidean descent (in z-space) on a cost function defined by (3.4) with reparameterization

$$\gamma(z) = \frac{1}{4} \; \mathrm{sgn}(z)|z|^2 + \sqrt{\epsilon} z.$$

Other priors lead in analogous fashion to other update rules, and other modifications such as replacing the power of 2 in the cost function (3.4) with more general powers p are equally straightforward.

Chapter 7

Simulations

This chapter gives experimental performance curves in several synthetic system identification scenarios that compare the performance of LMS to that of signed sparse LMS (SSLMS in Table 6.1) in terms of MSE, convergence rates, and tracking ability. For the comparisons to be fair, the stepsizes are chosen so as to equate the MSE after convergence, using the method of Section 4.1.

7.1 Performance in a Sparse Environment

This set of simulations was run in a sparse environment. The first channel had ten taps, with non-zero taps of values $[.1,\ 1,\ -.5,\ .1]$ located in positions $[1,\ 3,\ 4,\ 8]$. The second channel had 100 taps, with non-zero taps of the same values located in positions $[1,\ 30,\ 35,\ 85]$. We used zero initializations, and chose the stepsizes such that the MSE at convergence would be the same, leading to $\mu = 0.0050$, $\mu_{sparse} = 0.0215$ for channel 1, and $\mu_{sparse} = 0.0629$ for channel 2. The noise power was $(0.025)^2$, and the value of ϵ (for signed sparse LMS in Table 6.1) was 1/16. Recall that the ten-tap channel was used as the last example in Section 3.3.

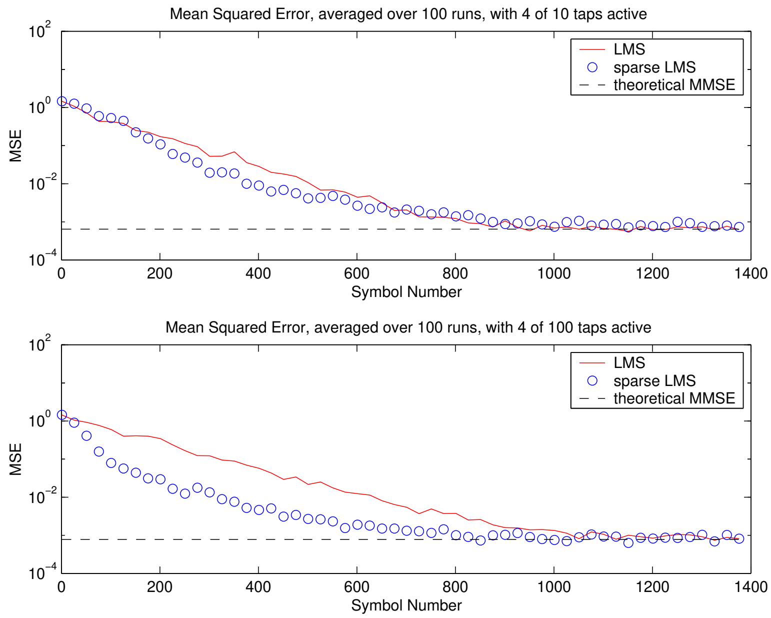

Figure 7.1: MSE curves for LMS and sparse LMS in a sparse environment.

Figure 7.1 shows the MSE versus time for both algorithms. The MSE was computed by taking the ensemble average of the squared error for each algorithm over 100 runs. The figure shows that when the environment is sufficiently sparse, the sparse version of LMS converges faster.

Figure 7.2 shows a comparison of theoretical performance curves of (4.9) to the simulated curves. In this case the system to be modeled consists of 30 non-zero taps followed by 170 zeros, with the largest taps occuring around tap 15. Unit variance BPSK was used as the input, and -20 dB measurement noise was added. The MSE curves were produced by averaging the squared error over 4000 samples, then ensemble averaging 100 such systems. The stepsizes were chosen such that the MSE values would be equal at $N = 100$, where N is the length of the estimate $\mathbf{w}$.

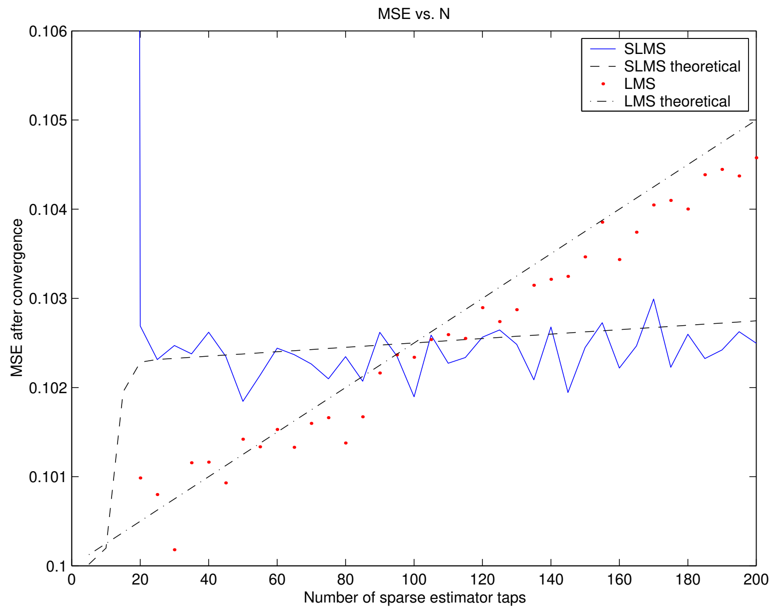

Figure 7.2: MSE versus model length in a sparse environment.

From the figure, it is clear that the simulations adequately match the theoretical curves. The uneven slope of the sparse LMS theoretical curve arises because the MSE depends on the values of the first N taps of the true system. Since the largest taps are located near tap 15, the largest slope in the MSE curve occurs at that point. Note that the theoretical curves only consider misadjustment, not the effects of undermodelling, which explains the descrepancy between the theory and the simulations for small N. The misadjustment for sparse LMS is not sensitive to the length of the model (except through ϵ, which is small), whereas the misadjustment for traditional LMS is proportional to the length of the model.

In order to see the effects of a non-diagonal R (i.e. the input is no longer white), we repeated the previous simulation with a different input sequence. A colored input sequence was obtained by passing white noise through the filter [.1,-.5,1,-.2,1,-.2,.2,0,.1]. We then passed the colored input through the channel used

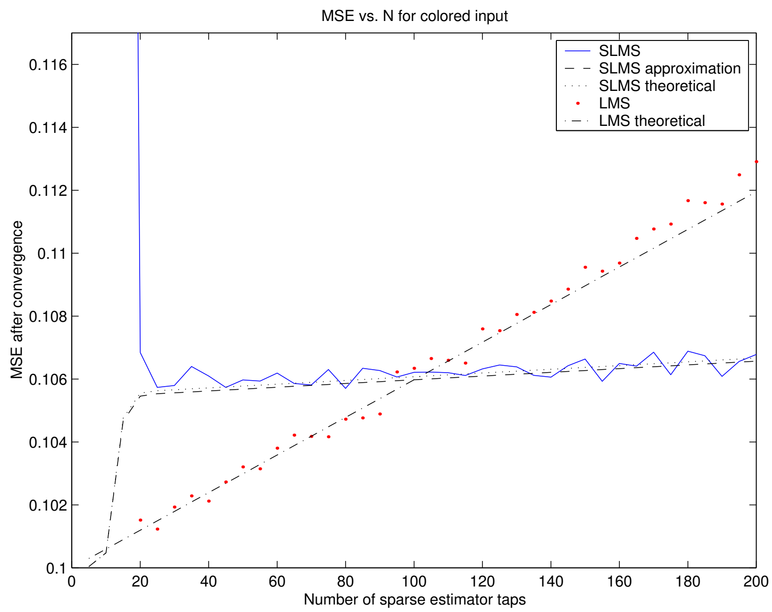

Figure 7.3: MSE versus model length in the case of colored input.

above. Figure 7.3 shows the MSE versus model length for traditional and sparse LMS. The theoretical curves are also plotted, with and without the approximation leading to Equation (4.5). We see that since μ is small (0.0005), the approximation does not significantly change the theoretical curve.

7.2 Performance in a Non-Sparse Environment

The next set of simulations was run in a non-sparse environment. The channels were changed from 4 out of 10 (and 4 out of 100) to 9 out of 10 (and 90 out of 100) nonzero taps, but the ANG algorithm was run assuming (falsely) that the sparse conditions were still present.

Figure 7.4 shows the MSE versus time for both algorithms. In these cases, the performance of sparse LMS is worse than regular LMS. However, it is worth noting

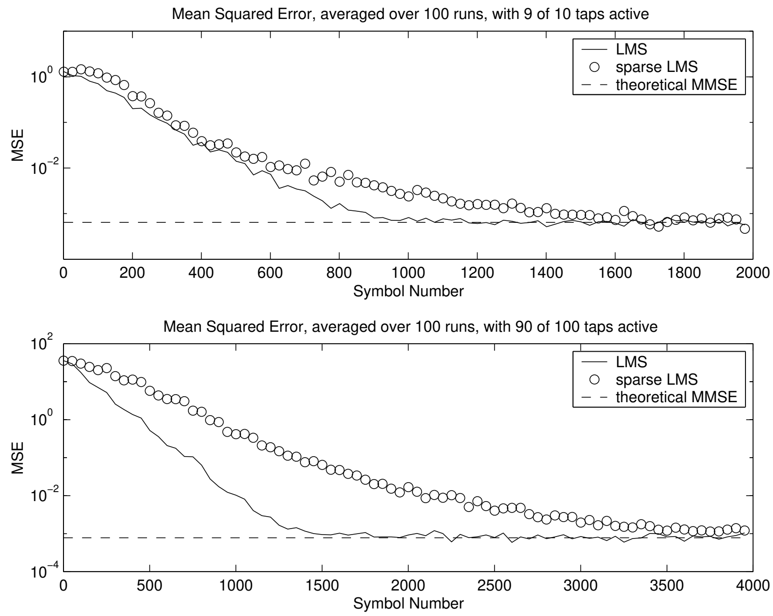

Figure 7.4: MSE curves for LMS and sparse LMS in a non-sparse environment. Note the different scales.

that the performance loss by a sparse algorithm in a non-sparse environment is comparable to it's performance gain in a sparse environment, relative to LMS.

7.3 Tracking Ability

This simulation was run on a 20 tap channel, with two taps set to vary sinusoidally in time. The actual channel consisted of the first channel from the sparse simulation in Section 7.1, with ten zeros appended. Then taps 15 and 16 were set to vary as $1 + 0.2\ \sin(k\ \pi/256)$ and $0.2\ \sin(k\ \pi/256)$, respectively, where k is the iteration number. Again, the stepsizes were chosen to equate the MSE.

Figure 7.5 shows the tracking ability of both algorithms. The upper plot shows the values of the actual taps and the estimates as both taps fluctuate. As expected,

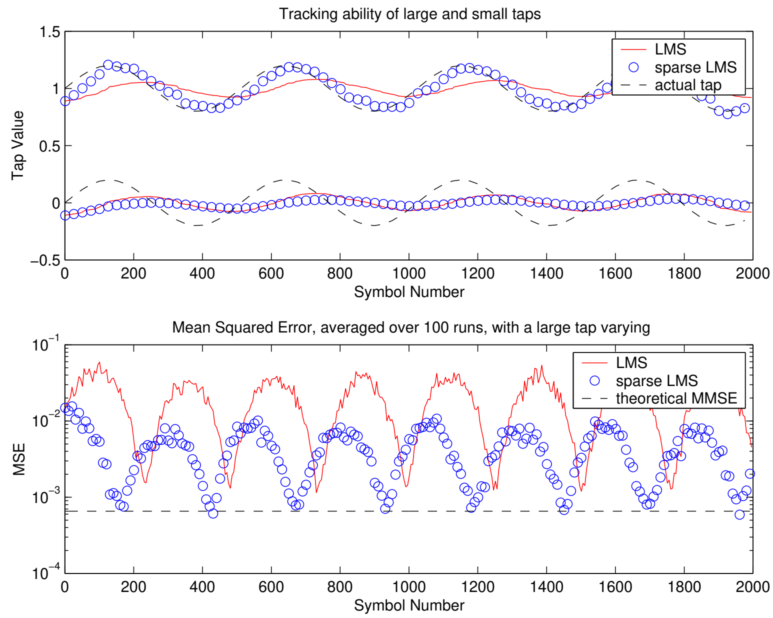

Figure 7.5: Comparison of the tracking ability of the two algorithms.

sparse LMS is better at tracking the change in the large taps, but not the small taps. The lower plot shows the MSE when only the larger tap is fluctuating. When only the large tap is fluctuating, the sparse algorithm has a lower MSE than LMS.

7.4 Bayesian Channels

Now we will investigate what happens when the tap weights are drawn from a Bayesian prior, and the algorithm is chosen to incorporate that prior. This simulation is similar to those in Section 7.1, but now the channel is different. For each of the 100 ensemble members, a 20 tap channel is picked according to the prior used in the signed sparse LMS algorithm (note that the channel is different for each ensemble member). This prior is not normailizable, so it has been truncated in order to make it a valid probability density function.

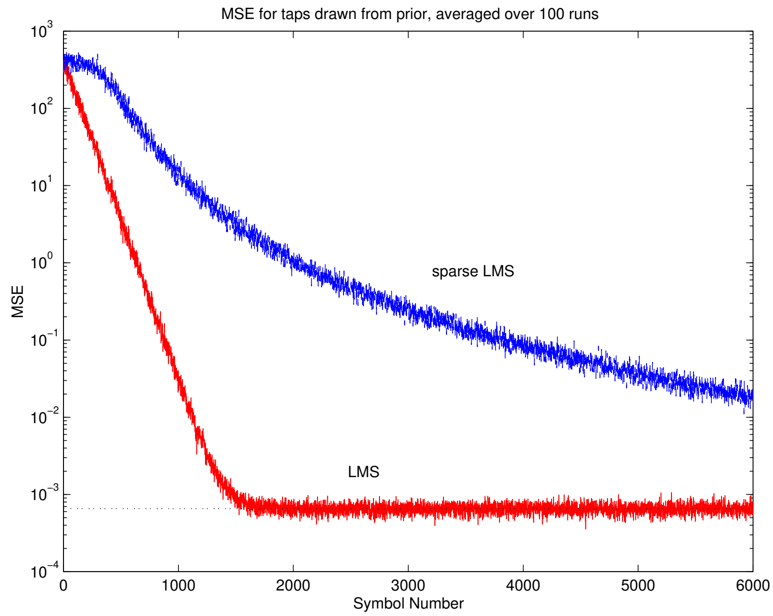

Figure 7.6: Convergence rates when taps are picked from the prior.

Figure 7.6 shows the results of running both algorithms with zero initializations. The results are very different than those in Section 7.1. In this case, sparse LMS does much worse on average than LMS. The discrepancy is easily explained. Before, most of the true wieghts were identitically zero, so their estimates did not have to evolve at all. Now, many of the true weights are small, but non-zero, so their estimates do have to evolve. In Chapter 3, we showed that the large taps evolve quickly in SSLMS at the expense of the small taps. Thus, if the small taps do not have to move at all, SSLMS wins out. However, if there are many small taps, and they all have to move a little bit, then the net effect is that SSLMS is too slow.

Also note that with a zero initialization, SSLMS moves very slowly at first. In Chapters 3 and 4, we showed that there is a plateau near the origin, which explains this effect.

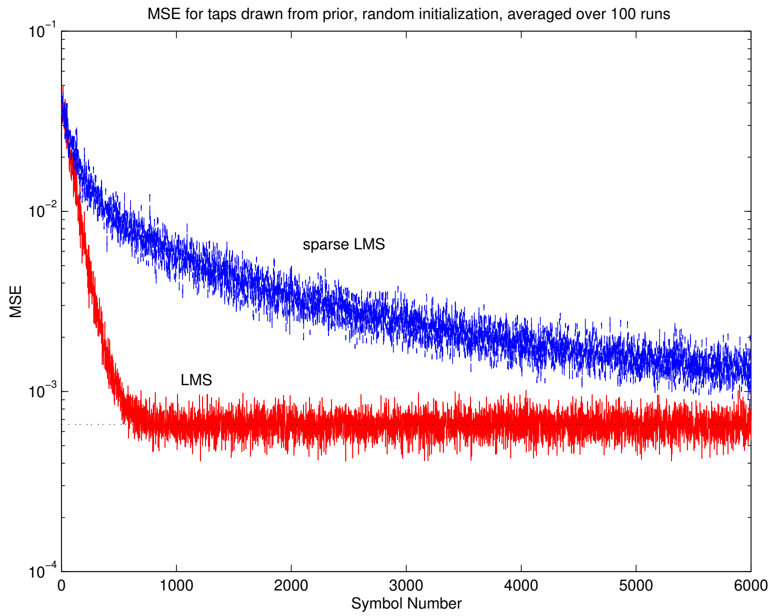

Figure 7.7: Convergence rates near the optimum, with random initializations.

The conclusion is that sparse LMS will work better if there are many extremely small taps and a few large taps, but when the taps are more evenly distributed, sparse LMS (with a zero initialization) performs poorly. Thus, it is of interest to consider the same situation, but with different initializations. In particular, it is important to consider initializations near the true weight vector.

7.5 Local Convergence

In this section we will investigate the effect on the convergence rate of varying the initialization near the right answer. As in the previous section, the tap weights will be drawn from the prior that is used in the algorithm. The only difference between this section and the previous one is that we will now change the initialization.

Figure 7.7 shows the effects of randomly initializing near the right answer, such

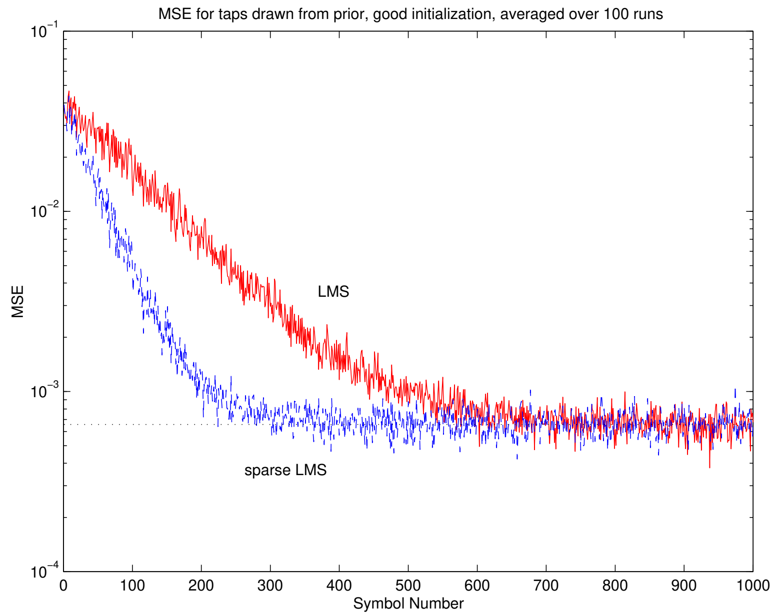

Figure 7.8: Convergence rates near the optimum, with good initializations.

that the initial error vector is in a random direction, with magnitude equal to one percent of the mangitude of the true weight vector. The channel and initialization are different with each ensemble member. This shows that even near the right answer, most initializations result in poor performance by SSLMS. However, this is as expected. Since the direction of the error vector is random, there will be a significant amount of error in all of the taps. Since most of the taps are small, it is difficult for them to converge. The large taps still converge quickly, but this is not enough to balance the effect of the slow convergence of all the small taps.

Figure 7.8 shows the effect of initializing in the "best" direction. The error vector was chosen to be in the direction of the largest tap, and the magnitude of the error vector was equal to one percent of the mangitude of the true weight vector. Again, the channel and initialization are different with each ensemble member. This shows that when all of the error is in the largest tap, sparse LMS will converge faster than LMS. This is exactly what our theoretical analysis in section 4.3 predicted (compare to Figure 4.1).

The conclusion from these simulations is that algorithms that incorporate priors are not always better than standard algorithms. The performance of these algorithms is dependent on the true weight vector and on the initialization. Fortunately, Chapter 8 will show that for a database of measured channels, it is advantageous to use algorithms that incorporate priors, even with zero initializations.

Chapter 8

Measured Channels

In April 2000, researchers (including the author) from Cornell University, University of Wisconsin-Madison, Australian National University, Applied Signal Technology, and NxtWave Communications met in Philadelphia for field measurements of digital television channels. The data collected there have been compiled into a large database of identified channels and associated MMSE decision feedback equalizers. The data from 47 of these channels were used to produce histograms of the magnitudes of typical equalizer coefficients. The histograms of the forward filter of the equalizer are well modeled by both exponential priors ($c\ e^{-\alpha z}$) and inverse power law (IPL) priors ($\frac{c}{|z|^{\alpha}+\epsilon}$). The feedback filter was well modeled by an exponential prior, and the channel itself was well modeled by an IPL prior. The methods used to fit priors to the histograms are documented in Appendix B. The exponential and IPL update algorithms for channel identification are given in Table 6.1.

The top plot in Figure 8.1 shows a histogram of the magnitudes of all the complex taps from the forward equalizers for all of the channels. The bottom plot is similar, but it shows the taps of all of the (real) feedback equalizers.

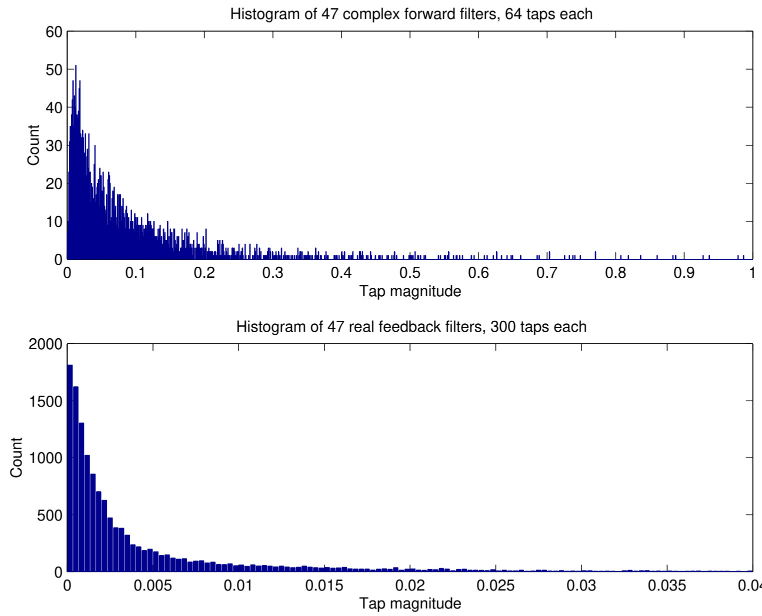

Figure 8.1: Histograms of the DFE.

Figure 8.2 shows the histogram of the magnitudes of the complex channel taps, as well as exponential and IPL curves that have been fitted to the histogram. Note that a logarithmic scale has been used, and that the exponential prior is a poor fit.

The gain constant c appears in the update rule in such a way that it can be absorbed by the stepsize. The parameter ϵ in the IPL prior and the α that appears in the exponential prior are both subject to changes in the scale of $\mathbf{w}$. Thus, a different receiver with a different automatic gain controller will have different values for these parameters.

This thesis has focused on the real case, though it is a simple matter to extend the algorithms to the complex case. Figure 8.3 shows an example of identification of one of the complex channels used for the histogram. The top plot shows the MSE versus time for traditional LMS and sparse LMS, with a variety of values of α

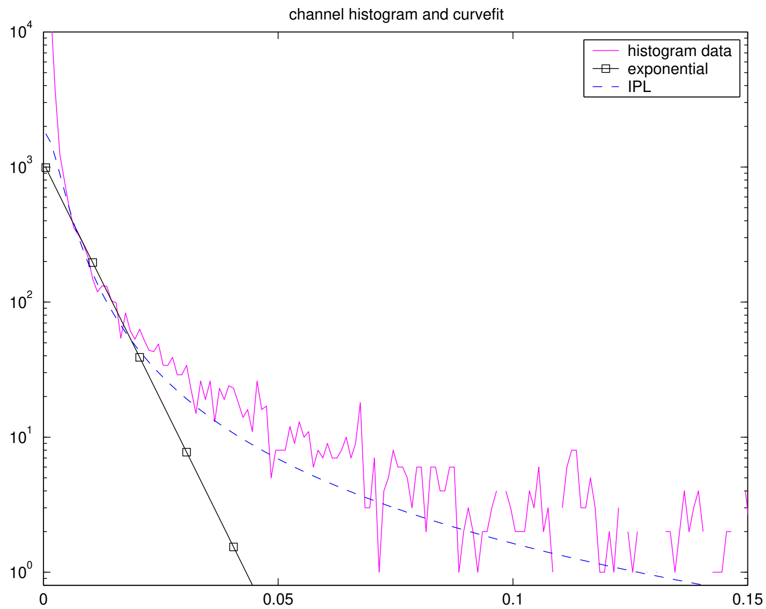

Figure 8.2: Histogram and curve fits for the channel.

(for the IPL prior). The bottom plot shows the complex channel in dB scale. The value of ϵ that we used was larger than specified by the curve fit, so as to speed initial convergence (since the model is initialized to zero). It should be noted that this channel is not a special case that just happened to show good performance for the sparse prior. The same simulation was done using ten different channels from the database, and the resulting MSE curves were almost identical to those in Figure 8.3.

Note that the results here are different from those in Section 7.4. This is unexpected, since we can think of a particular channel as having taps drawn from the prior corresponding to the histogram. There are two probable reasons for the discrepancy. The first is that the prior does not model the histogram well for the smallest bin, so there are actually many more taps in this bin than the prior

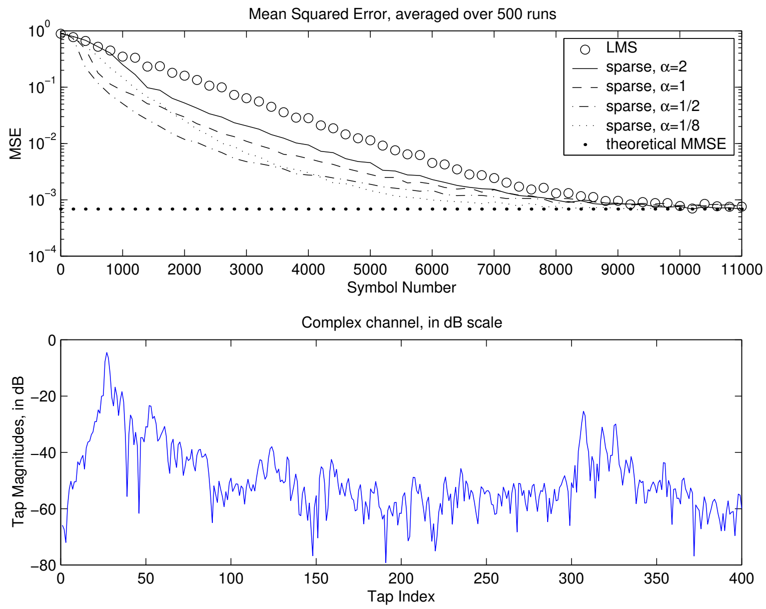

Figure 8.3: A measured complex channel and plot of convergence rates for channel identification.

suggests. Thus, the channel is sparser than it would be if it had been drawn from the prior randomly. The second reason is that the histogram bins have a non-zero width. Assume the smallest histogram bin contains taps whose magnitudes lie in $[0, \alpha]$. If the taps are generated randomly, their magnitudes will lie throughout this range. However, for the measured channels, many of the taps are essentially zero, so they lie at the bottom edge of the bottom bin. This means that those taps do not have to adapt at all, whereas taps that are as large as α will have to adapt a little bit. The cumulative effect of having many small but non-zero taps can be significant.

Chapter 9

Conclusions and Future Work

We have derived the LMS algorithm and a wide range of variants through the framework of the Natural Gradient algorithms. In a similar fashion, other algorithms (such as CMA and DD-LMS) and their variants were derived. Using the concepts of reparameterization and priors, it is possible to exploit prior knowledge of the probability density function of the unknown parameters, with particular attention to the case of a sparse distribution of taps. The modifications to the algorithms were shown to be component-wise modifications to the step size, and the important idea is how the ANG algorithms show how to choose these modifications.

An analysis of the mean-square-error, convergence rates, and stability has been provided. It was shown that the convergence criteria for sparse algorithms are similar to that of LMS, although the rate of convergence of sparse algorithms is much more sensitive to the initialization than LMS.

Simulations verify that if one has accurate knowledge of the prior, then substantial performance gains can be achieved, depending on the initialization. Conversely, if a false prior is assumed, performance degradation often occurs.

Future work may involve:

- a more detailed approach to choosing the stepsize when the system is unknown,
- analyzing the stability and convergence behavior of the algorithms in greater detail,
- considering equalization applications (a DFE in particular) in more depth
- comparing bit error rate curves to the MSE curves,
- examining sensitivities of the algorithms to incorrect priors,
- and theoretically determining the priors of a channel from common statistical propagation models.

Appendix A

Derivation of Equation (4.7)

This appendix derives Equation (4.7). Starting from Equation (4.4), postulate that the solution for C is in the form of the approximate solution, (4.6), plus an order μ^2 error term:

$$C = \frac{\mu}{2}\, \xi_{min}\; D + E_1. \tag{A.1}$$

Substituting into (4.4) and simplifying yields

$$DRE_1 + E_1RD = \mu DRE_1RD + \frac{\mu^2}{2}\xi_{min}DRDRD. \tag{A.2}$$

Now postulate that the error term E_1 is of the form

$$E_1 = \frac{\mu^2}{4}\, \xi_{min}\; D\; R\; D + E_2, \tag{A.3}$$

where E_2 is an order μ^3 error term. Substitution of (A.3) into (A.2) yields

$$DRE_2 + E_2RD = \mu DRE_2RD + \frac{\mu^3}{4}\xi_{min}D(RD)^3.$$

Thus at each step the remaining error term is of the form of the solution obtained by ignoring the term of highest order of μ, plus a yet higher order error term.

Continuing this process indefinitely produces the infinite series

$$C = \frac{1}{2}\ \mu\ \xi_{min}\ D\ \sum_{i=0}^{\infty} \left(\frac{\mu}{2} RD\right)^{i},$$

which converges if and only if every eigenvalue of $\frac{\mu}{2}RD$ has a magnitude less than one. The infinite series can then be expressed as

$$C = \frac{1}{2}\ \mu\ \xi_{min}\ D\ \left(I - \frac{\mu}{2} RD\right)^{-1}, \tag{A.4}$$

which is the desired result. □

Appendix B

Curve Fits

In Chapter 8 we used a measured database of channels and equalizers to determine realistic priors. The priors were determined by fitting curves to histograms of the channels and equalizers. This appendix discusses how those curve fits were implemented.

Let $f(\mathbf{x})$ represent the values (counts) of a histogram, where $\mathbf{x}$ is a vector containing the centers of the bins used in the histogram. The goal is to find some function g, such that when g is evaluated at the various points of $\mathbf{x}$, it is as close as possible to $f(\mathbf{x})$. One approach is to use the standard least squares fitting technique of [8]. To be precise, we wish to minimize the following cost function:

$$J = \frac{1}{2} \sum_k (f(x_k) - g(x_k))^2. \tag{B.1}$$

The shape of the histograms suggests either an exponential prior,

$$g(x) = c\ e^{-\alpha x}, \tag{B.2}$$

or an inverse power law (IPL) prior,

$$h(x) = \frac{c}{|x|^\alpha + \epsilon}, \tag{B.3}$$

where c, α, and ϵ are constants to be determined (separately for each prior). Note that the function $h(x)$ serves the same purpose as $g(x)$, but a different letter was used for clarity.

For an exponential prior, one approach is to use an exponential regression to determine the parameters c and α. An approach that works on a wider range of functional forms is gradient descent. We first consider the exponential prior, $g(x)$. The gradient terms are

$$\frac{\partial}{\partial c}\left(\frac{1}{2}\sum_k (f(x_k) - ce^{-\alpha x})^2\right) = -\sum_k (f(x_k) - ce^{-\alpha x})^2\ e^{-\alpha x}$$

$$\frac{\partial}{\partial \alpha}\left(\frac{1}{2}\sum_k (f(x_k) - ce^{-\alpha x})^2\right) = \sum_k (f(x_k) - ce^{-\alpha x})^2\ cx_k e^{-\alpha x},$$

which yield the update rules

$$c_{n+1} = c_n + \mu \sum_k (f(x_k) - ce^{-\alpha x_k})^2\ e^{-\alpha x_k} \tag{B.4}$$

$$\alpha_{n+1} = \alpha_n - \mu \sum_k (f(x_k) - ce^{-\alpha x_k})^2\ cx_k e^{-\alpha x_k}. \tag{B.5}$$

With a suitably small stepsize, and with a good initialization, this algorithm should yield the values of c and α which minimize (B.1) using an exponential prior.

Now consider the IPL prior. There are three constants to be determined, and the gradient terms are

$$\frac{\partial}{\partial c}\left(\frac{1}{2}\sum_k \left(f(x_k) - \frac{c}{|x_k|^\alpha + \epsilon}\right)^2\right) = -\sum_k \left(f(x_k) - \frac{c}{|x_k|^\alpha + \epsilon}\right)\frac{1}{|x_k|^\alpha + \epsilon}$$

$$\frac{\partial}{\partial \alpha}\left(\frac{1}{2}\sum_k \left(f(x_k) - \frac{c}{|x_k|^\alpha + \epsilon}\right)^2\right) = \sum_k \left(f(x_k) - \frac{c}{|x_k|^\alpha + \epsilon}\right)\frac{c}{(|x_k|^\alpha + \epsilon)^2}|x_k|^\alpha \ln(x_k)$$

$$\frac{\partial}{\partial \epsilon}\left(\frac{1}{2}\sum_k \left(f(x_k) - \frac{c}{|x_k|^\alpha + \epsilon}\right)^2\right) = \sum_k \left(f(x_k) - \frac{c}{|x_k|^\alpha + \epsilon}\right)\frac{c}{(|x_k|^\alpha + \epsilon)^2}$$

which yield the update rules

$$c_{n+1} = c_n + \mu \sum_k \left(f(x_k) - \frac{c}{|x_k|^\alpha + \epsilon} \right) \frac{1}{|x_k|^\alpha + \epsilon} \tag{B.6}$$

$$\alpha_{n+1} = \alpha_n - \mu \sum_k \left(f(x_k) - \frac{c}{|x_k|^\alpha + \epsilon} \right) \frac{c}{(|x_k|^\alpha + \epsilon)^2} |x_k|^\alpha \ln(x_k) \tag{B.7}$$

$$\epsilon_{n+1} = \epsilon_n - \mu \sum_k \left(f(x_k) - \frac{c}{|x_k|^\alpha + \epsilon} \right) \frac{c}{(|x_k|^\alpha + \epsilon)^2}. \tag{B.8}$$

Again, with a suitably small stepsize, and with a good initialization, this algorithm should yield the values of c, α, and ϵ which minimize (B.1) using an IPL prior.

Similar algorithms can be derived to fit other priors to the data. However, for the data discussed in Chapter 8, at least one of these two priors matched the histogram for each parameter of interest. The accuracy of the different priors can be determined either visually (by comparing plots of the histogram and the prior) or by the final cost (check which prior gives a lower value in (B.1)).

One might think of building a reciever that keeps track of its channel models or equalizers over time, constructs its own histogram, and determines priors specific to its envronment. The problem with the gradient descent approach is that the curve fitting algorithms given here are rather complex, and their convergence can be slow. Thus, if the priors are to be determined on the fly, a simpler method of curve fitting is desired. Two such approaches, from [32], are considered below. For the sake of clarity, we will refer to these simply as method 2 and method 3.

Method 2: consider the IPL prior of (B.3). This can be rewritten as

$$z = A|x|^\alpha + B, \tag{B.9}$$

where $z = \frac{1}{h(x)}$, $A = \frac{1}{c}$, and $B = \frac{\epsilon}{c}$. Since ϵ is expected to be small, temporarily assume it is zero. Then the first step is to perform a linear regression on the log

Table B.1: Results of the curve fits.

Algorithm	c	α	ϵ
gradient descent	0.0132	2.065	0.430e-5
method 2	0.0397	1.852	0.000
method 3	0.0069	2.438	0.604e-5

of (B.9),

$$\ln(z) = \ln(A) + \alpha \ln|x|,$$

to estimate α and $\ln(A)$. Substituting into (B.9) yields

$$B = z - \hat{A}|x|^{\hat{\alpha}}. \tag{B.10}$$

Now B can be estimated by taking the mean of the right-hand side, using the previous estimates $\hat{A}$ and $\hat{\alpha}$. However, this ad hoc estimation of B will worsen the fit; so the plot below assumes $B = 0$.

Method 3: this approach also starts from (B.9). Note that for a given value of α, a linear regression yeilds estimates of A and B, hence the estimates $\hat{c}$ and $\hat{\epsilon}$. The regrssion also has an associated least squares cost. If many regressions are done, each using a different value of α, then the best estimate of α is the one which yeilds the lowest cost when it is used to perform the regression. The best estimates $\hat{c}$ and $\hat{\epsilon}$ are those obtained from that particular regression.

The results from the fits are given in Table B.1 and Figures B.1 and B.2. It should be noted that the fit using gradient descent took several hours to run, and required a good initial guess in order to converge, whereas methods 2 and 3 combined ran in a few seconds, producing comparable results. Thus, the latter two methods would be more appropriate for use on a chip.

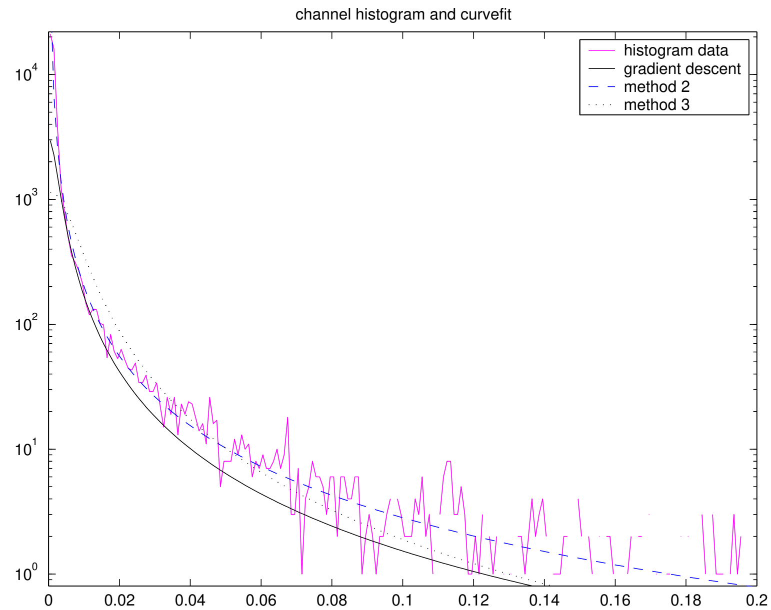

Figure B.1: Results of three curve fitting methods.

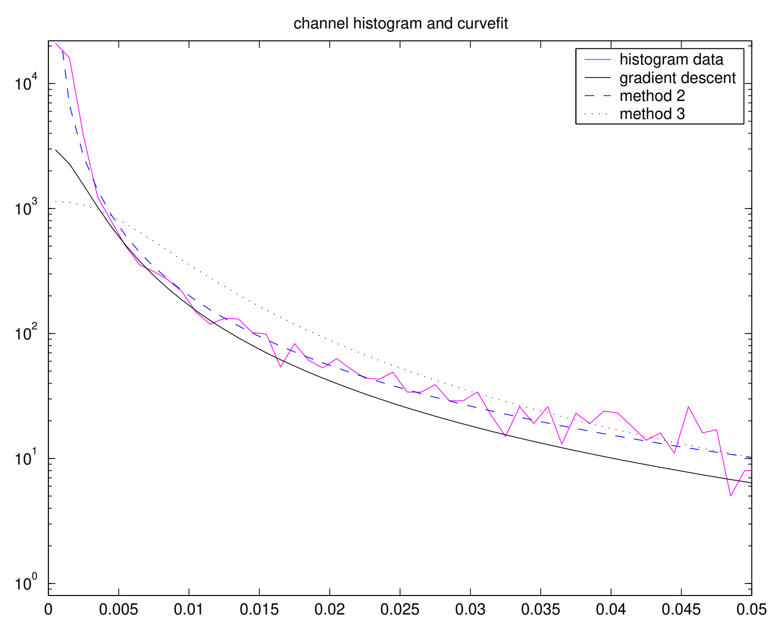

Figure B.2: Results of three curve fitting methods (zoomed in).

Appendix C

Matlab Code

This appendix contains the Matlab code used to generate the more important figures in this thesis. For some of the plots, different lines should be commented or uncommented.

C.0.1 Cost Functions

The file `slmscost3.m` was used to generate Figures 3.1 and 3.2.

```
% slmscost3.m


ep = 1/16;
%----------------------
% Set up true parameters
z1  = 1.3531;
z2  = 0.5;
g1 = 1/2*sign(z1).*z1.*z1 + ep*z1;
g2 = 1/2*sign(z2).*z2.*z2 + ep*z2;
```

```
z1h = -0.50:0.01:2.00;
z2h = -1.00:0.01:1.50;
z1t = z1 - z1h;
z2t = z2 - z2h;
g1t = g1 - z1h;
g2t = g2 - z2h;

%----------------------
% Simulate LMS and SSLMS
kmax = 25000;
sig_n = 0;
Z0z = [0; 0];
G0z = 1/2*sign(Z0z).*Z0z.*Z0z + ep*Z0z;
Z_star = [z1; z2];
G_star = [g1; g2];
N = length(Z_star);
mu = 0.001;
mu_s = mu*N/sum( 2*abs(Z_star)+ep^2 );

a = 0;
Rx = [1 a; a 1];
ah = (1+sqrt(1-4*a*a))/(2*a);
if a==0, ah=0; end;
flt = [1, ah]/sqrt(1+ah*ah);  % coloring filter
%x = randn(kmax+N-1,1);          % Gaussian(0,1)
```

```
x = 2*round(rand(kmax+N-1,1))-1; % BPSK
x2 = filter(flt,1,x);
x2 = x2(length(x2)-length(x)+1:end);
d = zeros(kmax,1);
for k=1:kmax
   X = x2(k:k+N-1);
   d(k) = G_star.' * X;
end
%----LMS-----
W = G0z;
y = zeros(kmax,1);
e = zeros(kmax,1);
n = zeros(kmax,1);
W1n = zeros(kmax,1);
W2n = zeros(kmax,1);
h3 = waitbar(0,'Running LMS...');
for k=1:kmax
   X = x2(k:k+N-1);
   y(k) = X.' * W;
   e(k) = d(k)-y(k)+n(k);
   W = W + mu*e(k)*conj(X);
   W1n(k) = W(1);
   W2n(k) = W(2);
   waitbar(k/kmax,h3);
end
close(h3);
```

```
W1n = [G0z(1); W1n];
W2n = [G0z(2); W2n];
%---SLMS-----
Z = Z0z;
Z1s = zeros(kmax,1);
Z2s = zeros(kmax,1);
h3 = waitbar(0,'Running sparse LMS...');
for k=1:kmax
   X = x2(k:k+N-1);
   gamma = (1/2)*sign(Z).*Z.*Z + ep*Z;
   y(k) = X.' * gamma;
   e(k) = d(k)-y(k)+n(k);
   Z = Z + mu*e(k)*(abs(Z)+ep).*conj(X);
   Z1s(k) = Z(1);
   Z2s(k) = Z(2);
   waitbar(k/kmax,h3);
end
close(h3);
Z1s = [Z0z(1); Z1s];
Z2s = [Z0z(2); Z2s];
W1s = (1/2)*sign(Z1s).*Z1s.*Z1s + ep*Z1s;
W2s = (1/2)*sign(Z2s).*Z2s.*Z2s + ep*Z2s;

%-------------------------
% Compute the cost surfaces
J = zeros(length(z1t),length(z2t));
```

```
J2 = zeros(length(z1t),length(z2t));

h3 = waitbar(0,'Computing cost surfaces...');
L = length(z1t);
for i=1:length(z1t)
   for j=1:length(z2t)
      wvec = [g1t(i); g2t(j)];
      J(i,j) = wvec'*Rx*wvec;
      temp1 = 0.5*(sign(z1)*z1*z1-sign(z1h(i))*z1h(i)*z1h(i)) + ...
        ep*z1t(i);
      temp2 = 0.5*(sign(z2)*z2*z2-sign(z2h(j))*z2h(j)*z2h(j)) + ...
        ep*z2t(j);
      wvec = [temp1; temp2];
      J2(i,j) = wvec'*Rx*wvec;
   end
   waitbar(i/L,h3);
end
close(h3);

%----------------
% Plot everything
levels = [0.02:0.10:3];
figure(1);
contour(z1h,z2h,J.',levels);
hold on;
plot(g1,g2,'.');
```

```
plot(z1h,zeros(size(z1h)),'b-');
plot(zeros(size(z2h)),z2h,'b-');
plot(W1n,W2n,'k-');
plot(W1s,W2s,'b-');
plot(W1n(1:(end-1)/20:end),W2n(1:(end-1)/20:end),'ko');
plot(W1s(1:(end-1)/20:end),W2s(1:(end-1)/20:end),'bx');
hold off;
axis([-0.2 1.6 -0.8 1]);
axis square;
%title(['LMS cost, Rx=[1 ',num2str(a),'; ',num2str(a),' 1]']);
title('LMS cost.  Trajectories: o=LMS, x=SSLMS');
xlabel('w_1');
ylabel('w_2');

figure(2);
contour(z1h,z2h,J2.',levels/2);
hold on;
plot(z1,z2,'.');
plot(z1h,zeros(size(z1h)),'b-');
plot(zeros(size(z2h)),z2h,'b-');
plot(Z1s,Z2s,'b-');
plot(Z1s(1:(end-1)/20:end),Z2s(1:(end-1)/20:end),'bx');
hold off;
axis([-0.5 1.8 -0.8 1.5]);
axis square;
%title(['SLMS cost, Rx=[1 ',num2str(a),'; ',num2str(a),' 1]']);
```

```
title('SSLMS cost.  Trajectory: x=SSLMS');
xlabel('z_1');
ylabel('z_2');
```

C.0.2 Plots in Chapter 4

The file `twotaprace.m` was used to generate Figures 4.1 and 4.2.

```
% twotaprace.m


epsilon = 1/16; % (doesn't matter, as long as it is > 0)
W_star = [1; 0.1];
Ds = abs(W_star) + epsilon;
N = length(W_star);
w1 = W_star(1);
w2 = W_star(2);


w1h = -1.25:0.02:1.75;
w2h = -1.25:0.02:1.25;
w1t = w1 - w1h;
w2t = w2 - w2h;


muA = 0.001;
muB = muA * N/sum(Ds);


dMa = zeros(length(w1h),length(w2h));
```

```
dMb = zeros(length(w1h),length(w2h));

h3 = waitbar(0,'Computing delta MSE...');
L = length(w1h);
for i=1:length(w1h)
   for j=1:length(w2h)
      v1sq = (w1t(i))^2;
      v2sq = (w2t(j))^2;
      H1 = (2*abs(w1h(i))+2*epsilon)/ ...
         (abs(w1h(i))+abs(w2h(j))+2*epsilon);
      H2 = (2*abs(w2h(j))+2*epsilon)/ ...
         (abs(w1h(i))+abs(w2h(j))+2*epsilon);
      K1 = (2*abs(w1h(i))+2*epsilon)/(abs(w1)+abs(w2)+2*epsilon);
      K2 = (2*abs(w2h(j))+2*epsilon)/(abs(w1)+abs(w2)+2*epsilon);
      dMa(i,j) = v1sq + v2sq;
      dMb(i,j) = v1sq * H1 + v2sq * H2;
      dMb2(i,j) = v1sq * K1 + v2sq * K2;
   end
   waitbar(i/L, h3);
end
close(h3);

swin = dMb > dMa;
[I,J] = find(swin);
xax = w1h(I);
yax = w2h(J);
```

```
figure(1);
plot(xax,yax,'b.','MarkerSize',1);
hold on;
plot(w1,w2,'ko');
plot(w1h,zeros(size(w1h)),'k-');
plot(zeros(size(w2h)),w2h,'k-');
hold off;
axis image;
title('Region in which normalized SLMS is faster than LMS');
xlabel('W_1');
ylabel('W_2');

swin2 = dMb2 > dMa;
[I,J] = find(swin2);
xax = w1h(I);
yax = w2h(J);

figure(2);
plot(xax,yax,'b.','MarkerSize',1);
hold on;
plot(w1,w2,'ko');
plot(w1h,zeros(size(w1h)),'k-');
plot(zeros(size(w2h)),w2h,'k-');
hold off;
axis image;
```

```
title('Region in which SSLMS is faster than LMS');
xlabel('W_1');
ylabel('W_2');
```

C.0.3 MSE Comparison

The files `testbed2.m`, `LMS.m`, and `LMSw.m` were used to generate most of the figures in Chapter 7 (those that compared MSE over time for different algorithms) and the plots for the ten-tap example in Chapter 3.

```
% testbed2.m
% RKM June 28, 2000

kmax = 1000; % number of iterations per run
ensnum = 100; % ensemble size

%a = 2;  % a parameter of the IPL prior
a = -1; % (1/sqrt(|w|+epsilon)) pior

% Channel choices:
W_star = zeros(10,1);                          % ~~ #1, #2, #3, #4, #7
W_star([1,3,4,8]) = [.1 1 -.5 .1];                 % #1
%W_star = 0.04*randn(100,1);
%W_star([1,30,35,80]) = [.1 1 -.5 .1];              % #2
%W_star([10,130,140,800]) = [.1 1 -.5 .1];          % #3
%W_star = [.1 -.05 1 -.5 .5 -.1 0 .1 -.1 0.05].';   % #4
```

```
%W_star = sign(rand(100,1)).*((rand(100,1)).^2);    % #5 (a)
%W_star = 2*rand(100,1)-1;                          % #5 (b)
%indices = randperm(100);                           % #5 also
%W_star(indices(1:10)) = zeros(10,1);               % #5 still
%W_star = sign(rand(1000,1)).*((rand(1000,1)).^2);  % #6 (a)
%W_star = 2*rand(1000,1)-1;                         % #6 (b)
%indices = randperm(1000);                          % #6 also
%W_star(indices(1:10)) = zeros(10,1);               % #6 still
%W_star([1,3,4,8,15]) = [.1 1 -.5 .1 1];            % #7
%W_star = [ 0.0  0.1 -0.1  1.0 -0.1  0.0  0.0  1.0  0.1  0.0 ...
%       0.1  0.0 -0.1  0.0  0.1  0.0  0.0 -0.1  0.0  0.1 ].';
N = length(W_star);

sig_n = 0.025;  % variance = (0.025)^2
epsilon = 1/16;
%epsilon = 0.430e-5;
mu = 0.0050;
if a==-1
   mu_s = mu*N/sum( abs(W_star)+epsilon );
else
   mu_s = mu*N/sum( ((abs(W_star)).^(a)+epsilon).^2 );
end
freq = 1/256;  % for time variations

MSE1 = zeros(kmax,1);
MSE2 = zeros(kmax,1);
```

```
W0z = zeros(size(W_star));
%W0z = 1.01*delta*ones(size(W_star));
%W0z = 0.98*W_star;

% form a perturbation in a random direction
% with norm equal to 1% of the norm of W_star
pert = 2*rand(size(W_star))-1;
pert = pert*(0.01*norm(W_star)/norm(pert));
%W0z = W_star+pert;

% form a perturbation in the best direction
% with norm equal to 1% of the norm of W_star
[dummy,ind] = max(abs(W_star));
pert = zeros(size(W_star));
pert(ind) = 0.01*norm(W_star);
%W0z = W_star+pert;

%(d(k))

Winit = W0z;
%errflg = zeros(ensnum,1);
%h = waitbar(0,'Running the ensemble...');
for index = 1:ensnum;
   index/ensnum

   %% comment out this section unless picking taps from
```

```
%%  SSLMS prior
%for ind=1:20, W_star(ind) = sign(2*rand(1)-1) * pricdf(ep,R);
%end;
%mu_s = mu*N/sum( abs(W_star)+epsilon );
%% form a perturbation in the best direction
%% with norm equal to 1% of the norm of W_star
%[dummy,ind] = max(abs(W_star));
%pert = zeros(size(W_star));
%pert(ind) = 0.01*norm(W_star);
%Winit = W_star+pert;

x = randn(kmax+N-1,1);  % fix the input sequence
d = zeros(kmax,1);
for k=1:kmax
   %W_star(15) = 1 + 0.2*sin(pi*freq*k);
   %W_star(16) = 0.2*sin(pi*freq*k);
   X = x(k:k+N-1);
   d(k) = W_star.' * X;
end

[W1,e1,sspe1,W1n,W2n,W3n,W4n] = ...
   LMS(Winit,mu,d,x,kmax,sig_n,W_star);
MSE1 = MSE1+abs(e1.*e1);

[W2,e2,sspe2,W1s,W2s,W3s,W4s] = ...
   LMSw(Winit,mu_s,d,x,kmax,sig_n,a,epsilon,W_s$
```

```
   MSE2 = MSE2+abs(e2.*e2);

   %waitbar(index/ensnum,h);
end
%close(h);

MSE1 = MSE1/ensnum;
MSE2 = MSE2/ensnum;
% theoretical asymptotic MSE
MSEth = ((1+(1/2)*mu*N)*sig_n*sig_n)*ones(kmax,1); % theoretical
%MSEth1 = ((1+(1/2)*mu*N)*sig_n*sig_n);
%MSEth2 = ((1+(1/2)*mu_s*sum( abs(W_star)+epsilon ))*sig_n*sig_n);

% approximate MSE: use summed squared parameter error...
% (use this with ensnum=1 for faster results)
sspe1 = sspe1*1+sig_n^2;
sspe2 = sspe2*1+sig_n^2;

% theoretical MSE over time (for LMS)
Vo = Winit - W_star;
MSEj = zeros(kmax,1);
temp = 1;
for j=1:kmax
   temp=temp*(1-mu)*(1-mu);
   MSEj(j) = temp * sum(real( Vo.*conj(Vo) )) + sig_n*sig_n;
end;
```

```
% theoretical MSE over time (for sparse LMS)
MSEk = zeros(kmax,1);
Jo = abs(W0z) + epsilon;
Js = abs(W_star) + epsilon;
temp = ones(length(Js),1);
for j=1:kmax
   %Jt=Jo+(min(i,5000)/5000)*(Js-Jo);  % this is cheating....
   Jt=Js;
   temp=temp.*real( (1-mu_s*Jt).*conj(1-mu_s*Jt) );
   MSEk(j) = sum(temp .* Vo.^2) + sig_n*sig_n;
end;

%-------------------------------------------------
% Summary (figure)
%-------------------------------------------------

%   'Position', winpos, ...
fig_sum1 = figure('IntegerHandle', 'off', ...
   'Name', 'Summary',...
   'NumberTitle', 'off', ...
   'Position', [100,100,600,800], ...
   'MenuBar', 'none', ...
   'Resize', 'off');
set(gcf,'PaperType','usletter');
PaperSize = get(gcf,'PaperSize');
```

```
set(gcf,'PaperPosition',...
   [0.5 0.5 PaperSize(1)-0.5 PaperSize(2)-0.5]);

subplot(3,1,1);
%semilogy([1:kmax],MSE1,'r-',[1:kmax],MSE2,'b-',...
%    [1:kmax],MSE3,'m--',[1:kmax],MSE4,'k--',[1:kmax],MSEth,'k:');
semilogy([1:kmax],MSE1,'r-', ...
   [1:kmax],MSE2,'bo',[1:kmax],MSEth,'k:');
v = axis;
axis = ([v(1) v(2) v(3) min(v(4),1e5)]);
title(...
   ['Mean Squared Error, averaged over ',num2str(ensnum),' runs']);
legend('LMS','sparse LMS','theoretical',1);
xlabel('Symbol Number');
ylabel('MSE');
zoom on;

subplot(3,1,2);
%semilogy([1:kmax],sspe1,'r-',[1:kmax],sspe2,'b-',...
%    [1:kmax],sspe3,'m--',[1:kmax],sspe4,'k--');
semilogy([1:kmax],sspe1,'r-', [1:kmax],sspe2,'b-',...
    [1:kmax],MSEj,'k--', [1:kmax],MSEk,'k:');
v = axis;
%axis = ([v(1) v(2) v(3) min(v(4),1e5)]);  %?
axis tight
title('Sample Run:  SSPE*E[s^2] + E[noise^2]');
```

```
legend(...
   'LMS','sparse LMS','LMS (theoretical)','sLMS (theoretical)',1);
xlabel('Symbol Number');
ylabel('SSPE');
zoom on;

subplot(3,1,3);
semilogy(...
   [1:N],abs(W1),'r-',[1:N],abs(W2),'b:',[1:N],abs(W_star),'ko');
v = axis;
axis = ([v(1) v(2) v(3) min(v(4),2)]);
title('Magnitudes of System and Estimates');
legend('LMS','sparse LMS','Actual',4);
xlabel('Tap Number');
ylabel('Tap Value');
zoom on;

%-------
figure(2);
%semilogy([1:kmax],sspe1,'r-', [1:kmax],sspe2,'b-', ...
%  [1:kmax],MSE1,'g-', [1:kmax],MSE2,'m-',[1:kmax],MSEth,'k:');
%semilogy([1:kmax],sspe1,'r-', [1:kmax],sspe2,'b--');
semilogy(...
   [1:kmax],MSE1,'r-', [1:kmax],MSE2,'b--',[1:kmax],MSEth,'k:');
v = axis;
axis = ([v(1) v(2) v(3) min(v(4),1e5)]);
```

```
title(['MSE, averaged over ',num2str(ensnum),' runs']);
%legend('LMS','sparse LMS','LMS MSE','sparse LMS MSE',1);
%legend('LMS','sparse LMS',1);
xlabel('Symbol Number');
ylabel('MSE');
zoom on;

kk1 = 1:kmax;
figure(3);
title('test');
subplot(2,2,1);
plot(kk1,W1n,'r-', kk1,W1s(kk1),'b--');
ylabel('W_1');
subplot(2,2,2);
plot(kk1,W2n,'r-', kk1,W2s(kk1),'b--');
ylabel('W_2');
subplot(2,2,3);
plot(kk1,W3n,'r-', kk1,W3s(kk1),'b--');
xlabel('time index, k');
ylabel('W_3');
subplot(2,2,4);
plot(kk1,W4n,'r-', kk1,W4s(kk1),'b--');
xlabel('time index, k');
ylabel('W_4');
legend('LMS','SSLMS');
```

```
function [W,e,sspe,W1,W2,W3,W4] = LMS(W0,mu,d,x,kmax,sig_n,W_star)
% [W,e,sspe,W1,W2,W3,W4] = LMS(W0,mu,d,x,kmax,sig_n,W_star)
%       LMS update routine
%       RKM June 28, 2000
%
%       W(1) multiplies the newest x
%       x should have a length of at least kmax+N-1

sspe_flg = 1;
N = length(W0);

if nargin < 7, error('not enough inputs'); end;
%if nargin == 7, sspe_flg = 1; end;
if length(x)<kmax+N-1, error('x is too short.'); end;

x = x(:);
y = zeros(kmax,1);
e = zeros(kmax,1);
n = sig_n*randn(kmax,1);
sspe = zeros(kmax,1);
W1 = zeros(kmax,1);
W2 = zeros(kmax,1);
W3 = zeros(kmax,1);
W4 = zeros(kmax,1);
W = W0;
```

```
h3 = waitbar(0,'Running LMS...');
for k=1:kmax
   X = x(k:k+N-1);
   y(k) = X.' * W;
   e(k) = d(k)-y(k)+n(k);
   W = W + mu*e(k)*conj(X);
   W1(k) = W(1);
   W2(k) = W(2);
   W3(k) = W(3);
   W4(k) = W(4);
   if sspe_flg, sspe(k) = real( sum((W-W_star)'*(W-W_star)) ); end;
   waitbar(k/kmax,h3);
end
close(h3);


function [W,e,sspe,W1,W2,W3,W4] =
LMSw(W0,mu,d,x,kmax,sig_n,a,epsilon,W_star)
% [W,e,sspe,W1,W2,W3,W4] =
% LMSw(W0,mu,d,x,kmax,sig_n,a,epsilon,W_star)
%       update routine for LMS with a prior
%       RKM June 28, 2000
%
%       W(1) multiplies the newest x
%       x should have a length of at least kmax+N-1
```

```
sspe_flg = 1;
N = length(W0);

if nargin < 9, error('not enough inputs'); end;
%if nargin == 9, sspe_flg = 1; end;
if length(x)<kmax+N-1, error('x is too short.'); end;

x = x(:);
y = zeros(kmax,1);
e = zeros(kmax,1);
n = sig_n*randn(kmax,1);
sspe = zeros(kmax,1);
W1 = zeros(kmax,1);
W2 = zeros(kmax,1);
W3 = zeros(kmax,1);
W4 = zeros(kmax,1);
W = W0;

if a==-1,  % use prior 1/sqrt(|w|+epsilon)
 h = waitbar(0,'Running sparse LMS...');
 for k=1:kmax
   X = x(k:k+N-1);
   y(k) = X.' * W;
   e(k) = d(k)-y(k)+n(k);
   W = W + mu*e(k)*(abs(W)+epsilon).*conj(X);
```

```
    W1(k) = W(1);
    W2(k) = W(2);
    W3(k) = W(3);
    W4(k) = W(4);
    if sspe_flg, sspe(k) = real( sum((W-W_star)'*(W-W_star)) ); end;
    waitbar(k/kmax,h);
  end
  close(h);

else % a>0 means a is a parameter of the IPL prior
  h2 = waitbar(0,'Running IPL LMS...');
  for k=1:kmax
    X = x(k:k+N-1);
    y(k) = X.' * W;
    e(k) = d(k)-y(k)+n(k);
    W = W + mu*e(k)*((abs(W)).^a+epsilon).^2.*conj(X);
    W1(k) = W(1);
    W2(k) = W(2);
    W3(k) = W(3);
    W4(k) = W(4);
    if sspe_flg, sspe(k) = real( sum((W-W_star)'*(W-W_star)) ); end;
    waitbar(k/kmax,h2);
  end
  close(h2);
end
```

Bibliography

[1] K. Abed-Meraim and Y. Hua. Knowledge Based Identification of Sparse Multipath Channels. In *Second Australian Workshop on Signal Processing Applications*, pages 127–130, Brisbane, Qld., Australia, 1997.

[2] T. Aboulnasr and K. Mayyas. Complexity Reduction of the NLMS Algorithm via Selective Coefficient Update. *IEEE Transactions on Signal Processing*, 47(5):1421–1424, May 1999.

[3] A. K. Aman and S. A. Tretter. A New Type of Decision Feedback Equalizer. In *Asilomar Conference on Circuits, Systems, and Computers*, volume 1, pages 265–269, where?, 1991.

[4] S. Amari. Natural Gradient Works Efficently in Learning. *Neural Computation*, 10(2):251–276, February 1998.

[5] S. Ariyavisitakul and L. J. Greenstein. Reduced-Complexity Equalization Techniques for Broadband Wireless Channels. *IEEE Journal on Selected Areas in Communications*, 15(1):5–15, January 1997.

[6] S. Ariyavisitakul, N. R. Sollenberger, and L. J. Greenstein. Tap-Selectable Decision-Feedback Equalization. *IEEE Transactions on Communications*, 45(12):1497–1500, December 1997.

[7] J. Balakrishnan, W. A. Sethares, and C. R. Johnson, Jr. Approximate Channel Identification via delta-Signed Correlation. In *The 35th Annual Conference on Information Sciences and Systems*, Baltimore, MD, 2001.

[8] A. Bjorck. *Numerical Methods for Least Squares Problems.* SIAM, Philadelphia, PN, 1996.

[9] D. Brady and Preisig. *Underwater Acoustic Communications*, in chapter 8 of *Wireless Communications: Signal Processing Perspectives*, edited by Poor and Wornell. Upper Saddle River, NJ, 1998.

[10] J. A. Bucklew, T. Kurtz, and W. A. Sethares. Weak Convergence and Local Stability Properties of Fixed Stepsize Recursive Algorithms. *IEEE Transactions on Information Theory*, 39(3):966–978, May 1993.

[11] M. Cederval, B. C. Ng, and A. Paulraj. Structured Methods for Blind Multi-Channel Idnetification. In *13th International Conference on Digital Signal Processing*, volume 1, pages 387–390, Santorini, Greece, 1997.

[12] R. Cusani and J. Mattila. Equalization of Digital Radio Land Channels With Large Multipath Delay for Cellular Land Mobile Applications. *IEEE Transactions on Communications*, 47(3):348–351, March 1999.

[13] P. De, J. Bao, and T. Poon. A Calculation-Efficient Algorithm for Decision Feedback Equalizers. *IEEE Transactions on Consumer Electronics*, 45(3):526–532, August 1999.

[14] Z. Ding. Multipath Channel Identification Based on Partial System Information. *IEEE Transactions on Signal Processing*, 45(1):235–240, January 1997.

[15] S. C. Douglas. Adaptive Filters Employing Partial Updates. *IEEE Transanctions on Circuits and Systems II*, 44:209–216, March 1997.

[16] S. C. Douglas and S. Amari. *Natural-Gradient Adaptation*, in *Unsupervised Adaptive Filtering*, volume 1, edited by S. Haykin. John Wiley & Sons, Inc., New York, NY, 2000.

[17] T. J. Endres, R. A. Casas, S. N. Hulyalkar, and C. H. Strolle. On Sparse Equalization Using Mean-Square-Error and Constant Modulus Criteria. In *The 34th Annual Conference on Information Sciences and Systems*, volume 1, pages TA7b–7–12, Princeton, NJ, 2000.

[18] D. M. Etter. An Adaptive Technique for Identification of Sparse Impulse Response Systems. In *International Conference on Digital Processing of Signals in Communications*, pages 29–32, Loughborough, UK, 1985.

[19] D. M. Etter and S. D. Stearns. Adaptive Estimation of Time Delays in Sampled Data Systems. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-29(3):582–587, June 1981.

[20] I. J. Fevrier, S. B. Gelfand, and M. P. Fitz. Reduced Complexity Decision Feedback Equalization for Multipath Channels with Large Delay Spreads. *IEEE Transactions on Communications*, 47(6):927–937, June 1999.

[21] S. Haykin. *Adaptive Filter Theory*. Prentice-Hall, Englewood Cliffs, NJ, 1991.

[22] S. I. Hill and R. C. Williamson. Convergence of Exponentiated Gradient Algorithms. Accepted for publication in IEEE Transactions on Signal Processing.

[23] J. Homer. Detection Guided LMS Estimation of Sparse Channels. In *IEEE GLOBECOM 1998*, volume 6, pages 3704–3709, Brisbane, Qld., Australia, 1998.

[24] C. R. Johnson, Jr. *Lectures on Adaptive Parameter Estimation*. Prentice-Hall, Englewood Cliffs, NJ, 1988.

[25] T. Kailath. *Linear Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1980.

[26] J. Kivinen and M. K. Warmuth. Exponentiated Gradient Versus Gradient Descent for Linear Predictors. *Information and Computation*, 132(1):1–64, January 1997.

[27] M. Kocic, D. Brady, and M. Stojanovic. Sparse Equalization for Real-Time Digital Underwater Acoustic Communications. In *Oceans '95*, volume 3, pages 1417–1422, New York, NY, USA, 1995.

[28] M. J. Lopez, A. C. Singer, S. L. Whitney, and G. S. Edelson. A DFE Coefficient Placement Algorithm for Underwater Digital Acoustic Communications. In *Oceans '99*, volume 2, pages 996–1001, Piscataway, NJ, USA and Washington, DC, USA, 1999.

[29] R. E. Mahony and R. C. Williamson. Prior Knowledge and Preferential Structures in Gradient Descent Learning Algorithms. Submitted to Journal of Machine Learning Research, on August 4, 2000.

[30] R. E. Mahony and R. C. Williamson. Riemannian Structure of Some New Gradient Descent Learning Algorithms. In *Adaptive Systems for Signal Processing, Communication and Control Symposium (AS-SPCC)*, pages 197–202, Lake Louise, Alberta, Canada, 2000.

[31] I. Mareels and J. W. Polderman. *Adaptive Systems: An Introduction*. Birkhauser, Boston, 1996.

[32] K. A. Mathews, November 2000. Personal communication.

[33] C. P. Robert. *The Bayesian Choice*. Springer, New York, NY, 1994.

[34] W. A. Sethares, B. D. O. Anderson, and C. R. Johnson, Jr. Adaptive Algorithms With Filtered Regressor and Filtered Error. *Mathematics of Control, Signals and Sytems*, 2(4):381–403, 1989.

[35] M. Stojanovic, L. Freitag, and M. Johnson. Channel-Estimation-Based Adaptive Equalization of Underwater Acoustic Signals. In *Oceans '99*, volume 2, pages 985–990, Piscataway, NJ, USA and Washington, DC, USA, 1999.

[36] J. R. Treichler, C. R. Johnson, Jr., and M. G. Larimore. *Theory and Design of Adaptive Filters*. Wiley, New York, 1987.

[37] J. R. Treichler and M. G. Larimore. Thinned Impulse Responses for Adaptive FIR Filters. In *International Conference on Acoustics, Speech and Signal Processing*, volume 2, pages 631–634, Paris, France, 1982.

[38] B. Widrow, J. R. Glover, Jr., J. M. McCool, J. Kaunitz, C. S. Williams, R. H. Hearn, J. R. Zeidler, E. Dong, Jr., and R. C Goodlin. Adaptive Noise Cancelling: Principles and Applications. *Proceedings of the IEEE*, 63(12):1692–1716, December 1975.

[39] B. Widrow, J. McCool, M. G. Larimore, and C. R. Johnson, Jr. Stationary and Nonstationary Learning Characteristics of the LMS Adaptive Filter. *Proceedings of the IEEE*, 64(8):1151–1162, August 1976.