CHAPTER

5 STOCHASTIC GRADIENT DESCENT: THE LMS ALGORITHM AND ITS FAMILY

CHAPTER OUTLINE

Machine Learning. http://dx.doi.org/10.1016/B978-0-12-801522-3.00005-7

5.1 INTRODUCTION

In Chapter 4, we introduced the notion of mean-square error (MSE) optimal linear estimation and stated the normal equations for computing the coefficients of the optimal estimator/filter. A prerequisite for the normal equations is the knowledge of the second order statistics of the involved processes/variables, so that the covariance matrix of the input and the input-output cross-correlation vector to be obtained. However, most often in practice, all the designer has at her/his disposal is a set of training points; thus, the covariance matrix and the cross-correlation vector have to be estimated somehow. More important, in a number of practical applications, the underlying statistics may be time varying. We discussed this scenario while introducing Kalman filtering. The path taken there was to adopt a state-space representation and assume that the time dynamics of the model were known. However, although Kalman filtering is an elegant tool, it does not scale well in high-dimensional spaces due to the involved matrix operations and inversions.

The focus of this chapter is to introduce *online learning* techniques for estimating the unknown parameter vector. These are time-iterative schemes, which update the available estimate every time a measurement set (input-output pair of observations) is acquired. Thus, in contrast to the so-called *batch processing* methods which process the whole block of data as a single entity, online algorithms operate on a single data point at a time; therefore, such schemes do not require the training data set to be known and stored in advance. Online algorithmic schemes learn the underlying statistics from the data in a *time iterative* fashion. Hence, one does not have to provide further statistical information. Another characteristic of the algorithmic family, to be developed and studied in this chapter, is its computational simplicity. The required complexity for updating the estimate of the unknown parameter vector is linear with respect to the number of the unknown parameters. This is one of the major reasons that have made such schemes very popular in a number of practical applications; besides complexity, we will discuss other reasons that have contributed to their popularity.

The fact that such learning algorithms work in a time-iterative mode gives them the agility to learn and *track* slow time variations of the statistics of the involved processes/variables; this is the reason these algorithms are also known as *time-adaptive* or simply *adaptive*, because they can adapt to the needs of a changing environment. Online/time-adaptive algorithms have been used extensively since the early 1960s in a wide range of applications including signal processing, control, and communications. More recently, the philosophy behind such schemes is gaining in popularity in the context of applications where data reside in large databases, with a massive number of training points; for such tasks, storing all the data points in the memory may not be possible, and they have to be considered one at a time. Moreover, the complexity of batch processing techniques can amount to prohibitive levels, for today's technology. The current trend is to refer to such applications as *big data* problems.

In this chapter, we focus on a very popular class of online/adaptive algorithms that springs from the classical gradient descent method for optimization. Although our emphasis will be on the squared error loss function, the same rationale can also be adopted for other (differentiable) loss functions. The case of nondifferentiable loss functions will be treated in Chapter 8. The online processing rationale will be a recurrent theme in this book.

5.2 THE STEEPEST DESCENT METHOD

Our starting point is the method of *gradient descent*, one of the most widely used methods for iterative minimization of a differentiable cost function, $J(\boldsymbol{\theta})$, $\boldsymbol{\theta} \in \mathbb{R}^l$. As does any other iterative technique, the method starts from an initial estimate, $\boldsymbol{\theta}^{(0)}$, and generates a sequence, $\boldsymbol{\theta}^{(i)}$, $i = 1, 2, \ldots$, such that,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}, \quad i > 0, \tag{5.1}$$

where $\mu_i > 0$. All the schemes for the iterative minimization of a cost function, which we will deal with in this book, have the general form of (5.1). Their differences are in the way that μ_i and $\Delta\boldsymbol{\theta}^{(i)}$ are chosen; the latter vector is known as the *update direction* or the *search direction*. The sequence μ_i is known as the step-size or the *step length*, at the ith iteration; note that the values of μ_i may either be constant or change at each iteration. In the gradient descent method, the choice of $\Delta\boldsymbol{\theta}^{(i)}$ is done to guarantee that

$$J(\boldsymbol{\theta}^{(i)}) < J(\boldsymbol{\theta}^{(i-1)}),$$

except at a minimizer, $\boldsymbol{\theta}_*$.

Assume that at the $i-1$ iteration step the value $\boldsymbol{\theta}^{(i-1)}$ has been obtained. Then, mobilizing a first order Taylor's expansion around $\boldsymbol{\theta}^{(i-1)}$ we can write

$$J\left(\boldsymbol{\theta}^{(i)}\right) = J\left(\boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}\right) \approx J(\boldsymbol{\theta}^{(i-1)}) + \mu_i \nabla^T J(\boldsymbol{\theta}^{(i-1)}) \Delta\boldsymbol{\theta}^{(i)}.$$

Selecting the search direction so that

$$\nabla^{\mathrm{T}} J(\boldsymbol{\theta}^{(i-1)}) \Delta\boldsymbol{\theta}^{(i)} < 0, \tag{5.2}$$

then it guarantees that $J(\boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}) < J(\boldsymbol{\theta}^{(i-1)})$. For such a choice, $\Delta\boldsymbol{\theta}^{(i)}$ and $\nabla J(\boldsymbol{\theta}^{(i-1)})$ must form an *obtuse* angle. Figure 5.1 shows the graph of a cost function in the two-dimensional case, $\boldsymbol{\theta} \in \mathbb{R}^2$, and Figure 5.2 shows the respective isovalue contours in the two-dimensional plane. Note that, in general, the contours can have any shape and are not necessarily ellipses; it all depends on the functional form of $J(\boldsymbol{\theta})$. However, because $J(\boldsymbol{\theta})$ has been assumed differentiable, the contours must be smooth and accept at any point a (unique) tangent plane, as this is defined by the respective gradient. Furthermore, recall from basic calculus that the gradient vector, $\nabla J(\boldsymbol{\theta})$, is perpendicular to the plane (line) tangent to the corresponding isovalue contour, at the point $\boldsymbol{\theta}$ (Problem 5.1). The geometry is illustrated in Figure 5.3; to facilitate the drawing and unclutter notation, we have removed the iteration index i. Note that by selecting the search direction which forms an obtuse angle with the gradient, it places $\boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}$ at a point on a contour which corresponds to a lower value of $J(\boldsymbol{\theta})$. Two issues are now raised: (a) to choose the best search direction along which to move and (b) to compute how far along this direction one can go. Even without much mathematics, it is obvious from Figure 5.3 that if

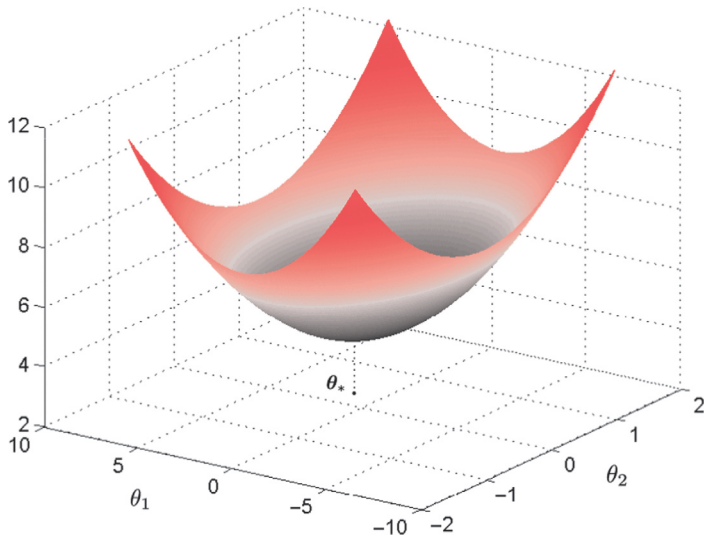


FIGURE 5.1

A cost function in the two-dimensional parameter space.

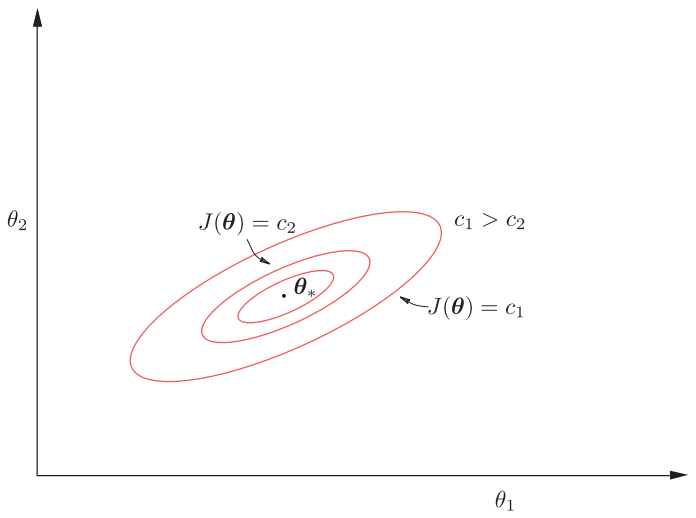


FIGURE 5.2

The corresponding isovalue curves of the cost function of Figure 5.1, in the two-dimensional plane.

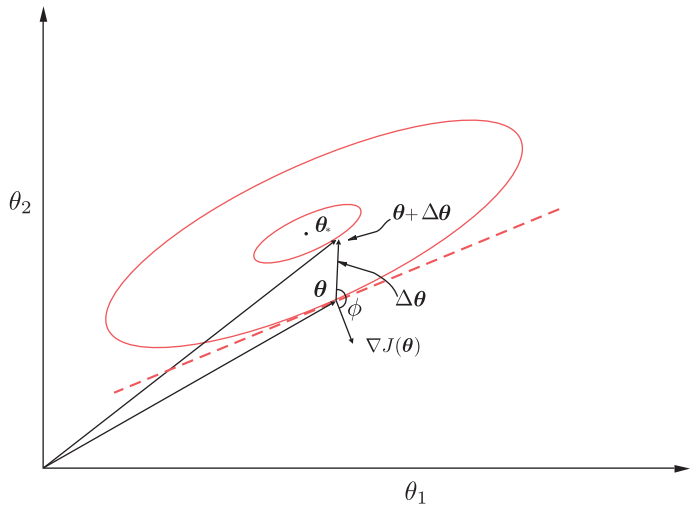


FIGURE 5.3

The gradient vector at a point $\boldsymbol{\theta}$ is perpendicular to the tangent plane at the isovalue curve crossing $\boldsymbol{\theta}$. The descent direction forms an obtuse angle, ϕ, with the gradient vector.

$\mu_i||\Delta\boldsymbol{\theta}^{(i)}||$ is too large, then the new point can be placed on a contour corresponding to a larger value than that of the current contour; after all, the first order Taylor's expansion holds approximately true for small deviations from $\boldsymbol{\theta}^{(i-1)}$.

To address the first of the two issues, let us assume $\mu_i = 1$ and search for all vectors, z, with unit Euclidean norm, $\boldsymbol{\theta}^{(i-1)}$. Then, it does not take long to see that for all possible directions, the one that gives the most negative value of the inner product, $\nabla^{\mathrm{T}} J(\boldsymbol{\theta}^{(i-1)})z$, is that of the negative gradient

$$z = -\frac{\nabla J(\boldsymbol{\theta}^{(i-1)})}{||\nabla J(\boldsymbol{\theta}^{(i-1)})||}.$$

This is illustrated in Figure 5.4. Center the unit Euclidean norm ball at $\boldsymbol{\theta}^{(i-1)}$. Then from all the unit norm vectors having their origin at $\boldsymbol{\theta}^{(i-1)}$ choose the one pointing to the negative gradient direction. Thus, for all unit Euclidean norm vectors, the steepest descent direction coincides with the (negative) *gradient descent* direction and the corresponding update recursion becomes

$$\boxed{\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i \nabla J(\boldsymbol{\theta}^{(i-1)}): \quad \text{Gradient Descent Scheme.}} \tag{5.3}$$

Note that we still have to address the second point, concerning the choice of μ_i. The choice must be done in such a way to guarantee convergence of the minimizing sequence. We will come to this issue soon.

Iteration (5.3) is illustrated in Figure 5.5 for the one-dimensional case. If at the current iteration the algorithm has "landed" at θ_1, then the derivative of $J(\boldsymbol{\theta})$ at this point is positive (the tangent of an acute angle, ϕ_1), and this will force the update to move to the left towards the minimum. The scenario is different if the current estimate was θ_2. The derivative is negative (the tangent of an obtuse angle, ϕ_2) and this will push the update to the right toward, again, the minimum. Note, however, that it is important how far to the left or to the right one has to move. A large move from, say θ_1, to the left may land the

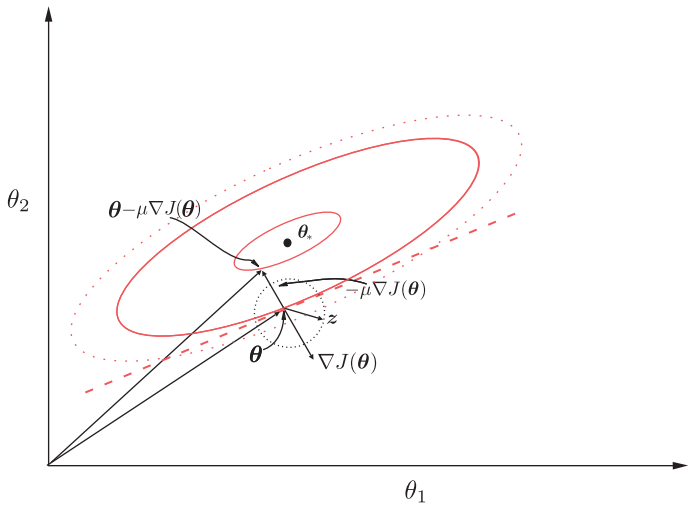


FIGURE 5.4

From all the descent directions of unit Euclidean norm (dotted circle), the negative gradient one leads to the maximum decrease of the cost function.

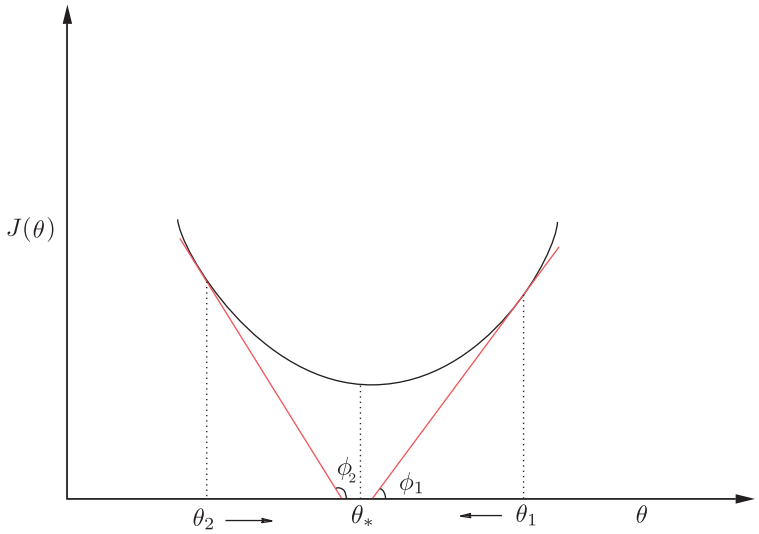


FIGURE 5.5

Once the algorithm is at θ_1, the gradient descent will move the point to the left, towards the minimum. The opposite is true for point θ_2.

update on the other side of the optimal value. In such a case, the algorithm may oscillate around the minimum and never converge. A major effort in this chapter will be devoted in providing theoretical frameworks for establishing bounds for the values of the step-size that guarantee convergence.

The gradient descent method exhibits approximately *linear convergence*; that is, the error between $\boldsymbol{\theta}^{(i)}$ and the true minimum converges to zero asymptotically in the form of a *geometric series*. However, the convergence rate depends heavily on the condition number of the Hessian matrix of $J(\boldsymbol{\theta})$. For very large values of the condition number, such as 1000, the rate of convergence can become extremely slow. The great advantage of the method lies in its low computational requirements.

Finally, it has to be pointed out that we arrived at the scheme in Eq. (5.3) by searching all directions via the unit Euclidean norm. However, there is nothing "sacred" around Euclidean norms. One can employ other norms, such as the ℓ_1 or the quadratic $\boldsymbol{v}^{\mathrm{T}}P\boldsymbol{v}$ norms, where P is a positive definite matrix. Under such choices, one will end up with alternative update iterations (see e.g., [23]). We will return to this point in Chapter 6 when dealing with Newton's iterative minimization scheme.

5.3 APPLICATION TO THE MEAN-SQUARE ERROR COST FUNCTION

Let us apply the gradient descent scheme to derive an iterative algorithm to minimize our familiar, from the previous chapter, cost function

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}\left[(\mathrm{y}-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x})^2\right] \\ &= \sigma_y^2 - 2\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{p} + \boldsymbol{\theta}^{\mathrm{T}}\Sigma_x\boldsymbol{\theta}, \end{aligned} \tag{5.4}$$

where

$$\nabla J(\boldsymbol{\theta}) = 2\Sigma_x\boldsymbol{\theta} - 2\boldsymbol{p}, \tag{5.5}$$

and the notation has been defined in Chapter 4. In this chapter, we will also adhere to zero mean jointly distributed input-output random variables, except if otherwise stated. Thus, the covariance and correlation matrices coincide. If this is not the case, the covariance in (5.5) is replaced by the correlation matrix. The treatment is focused on real data and we will point out differences with the complex-valued data case whenever needed.

Employing (5.5), the update recursion in (5.3) becomes

$$\begin{aligned} \boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}^{(i-1)} - \mu\left(\Sigma_x\boldsymbol{\theta}^{(i-1)} - \boldsymbol{p}\right) \\ &= \boldsymbol{\theta}^{(i-1)} + \mu\left(\boldsymbol{p} - \Sigma_x\boldsymbol{\theta}^{(i-1)}\right), \end{aligned} \tag{5.6}$$

where the step-size has been considered constant and it has also absorbed the factor 2. The more general case of iteration-dependent values of the step-size will be discussed soon after. Our goal now becomes that of searching for all values of μ that guarantee convergence. To this end, define

$$\boldsymbol{c}^{(i)} := \boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*, \tag{5.7}$$

where $\boldsymbol{\theta}_*$ is the (unique) optimal MSE solution that results by solving the respective normal equations,

$$\Sigma_x\boldsymbol{\theta}_* = \boldsymbol{p}.$$

Subtracting $\boldsymbol{\theta}_*$ from both sides of (5.6) and plugging in (5.7), we obtain

$$\begin{aligned}\boldsymbol{c}^{(i)} &= \boldsymbol{c}^{(i-1)} + \mu\left(\boldsymbol{p} - \Sigma_x \boldsymbol{c}^{(i-1)} - \Sigma_x \boldsymbol{\theta}_*\right) \\ &= \boldsymbol{c}^{(i-1)} - \mu\Sigma_x \boldsymbol{c}^{(i-1)} = (I - \mu\Sigma_x)\,\boldsymbol{c}^{(i-1)}.\end{aligned} \tag{5.8}$$

Recall that Σ_x is a symmetric positive definite matrix (Chapter 2), hence (Appendix A.2) it can be written as

$$\Sigma_x = Q\Lambda Q^{\mathrm{T}}, \tag{5.9}$$

where

$$\Lambda := \mathrm{diag}\{\lambda_1, \ldots, \lambda_l\} \quad \text{and} \quad Q := [\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_l],$$

with λ_j, $\boldsymbol{q}_j$, $j = 1, 2, \ldots, l$, being the (positive) eigenvalues and the respective normalized (*orthogonal*) eigenvectors of the covariance matrix,[1] represented by

$$\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{q}_j = \delta_{kj}, \quad k, j = 1, 2, \ldots, l \Longrightarrow Q^{\mathrm{T}} = Q^{-1}.$$

That is, the matrix Q is orthogonal. Plugging the factorization of Σ_x into (5.8), we obtain

$$\boldsymbol{c}^{(i)} = Q\,(I - \mu\Lambda)\,Q^{\mathrm{T}}\boldsymbol{c}^{(i-1)},$$

or

$$\boldsymbol{v}^{(i)} = (I - \mu\Lambda)\,\boldsymbol{v}^{(i-1)}, \tag{5.10}$$

where

$$\boldsymbol{v}^{(i)} := Q^{\mathrm{T}}\boldsymbol{c}^{(i)}, \quad i = 1, 2, \ldots. \tag{5.11}$$

The previously used "trick" is a standard one and its aim is to "decouple" the various components of $\boldsymbol{\theta}^{(i)}$ in (5.6). Indeed, each one of the components, $v^{(i)}(j)$, $j = 1, 2, \ldots, l$, of $\boldsymbol{v}^{(i)}$ follows an iteration path, which is independent of the rest of the components; in other words,

$$\begin{aligned}v^{(i)}(j) &= (1 - \mu\lambda_j)v^{(i-1)}(j) = (1 - \mu\lambda_j)^2 v^{(i-2)}(j) \\ &= \cdots = (1 - \mu\lambda_j)^i v^{(0)}(j),\end{aligned} \tag{5.12}$$

where $v^{(0)}(j)$ is the jth component of $\boldsymbol{v}^{(0)}$, corresponding to the initial vector. It is readily seen that if

$$|1 - \mu\lambda_j| < 1 \Longleftrightarrow -1 < 1 - \mu\lambda_j < 1, \quad j = 1, 2, \ldots, l, \tag{5.13}$$

the geometric series tends to zero and

$$\boldsymbol{v}^{(i)} \longrightarrow \boldsymbol{0} \Longrightarrow Q^{\mathrm{T}}(\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*) \longrightarrow \boldsymbol{0} \Longrightarrow \boldsymbol{\theta}^{(i)} \longrightarrow \boldsymbol{\theta}_*. \tag{5.14}$$

Note that (5.13) is equivalent to

$$\boxed{0 < \mu < 2/\lambda_{\max}: \quad \text{Condition for Convergence,}} \tag{5.15}$$

where $\lambda_{\max}$ denotes the maximum eigenvalue of Σ_x.

[1] In contrast to other chapters, we denote eigenvectors with $\boldsymbol{q}$ and not as $\boldsymbol{u}$, since at some places the latter is used to denote the input random vector.

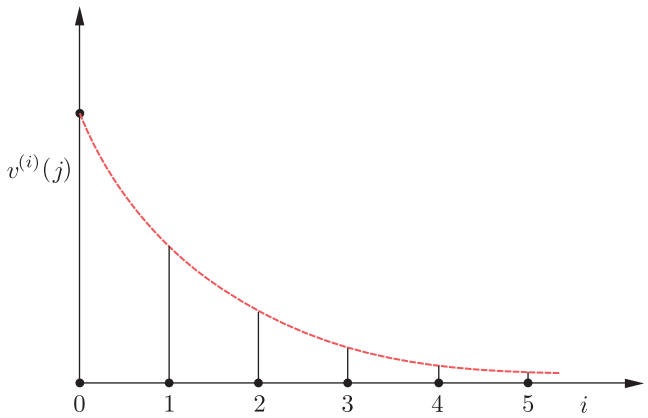


FIGURE 5.6

Convergence curve for one of the components of the transformed error vector. Note that the curve is of an approximate exponentially decreasing type.

Time constant: Figure 5.6 shows a typical sketch of the evolution of $v^{(i)}(j)$ as a function of the iteration steps for the case $0 < 1 - \mu\lambda_j < 1$. Assume that the envelope, denoted by the red line is (approximately) of an exponential form, $f(t) = \exp(-t/\tau_j)$. Plug into $f(t)$, as the values corresponding at the time instants, $t = iT$ and $t = (i-1)T$, the values of $v^{(i)}(j)$, $v^{(i-1)}(j)$ from (5.12); then, the *time constant* results as

$$\tau_j = \frac{-1}{\ln(1 - \mu\lambda_j)},$$

assuming that the sampling time between two successive iterations is $T = 1$. For small values of μ, we can write

$$\tau_j \approx \frac{1}{\mu\lambda_j}, \quad \text{for } \mu \ll 1.$$

That is, the slowest rate of convergence is associated with the component that corresponds to the smallest eigenvalue. However, this is only true for small enough values of μ. For the more general case, this may not be true. Recall that the rate of convergence depends on the value of the term $1 - \mu\lambda_j$. This is also known as the jth *mode*. Its value depends not only on λ_j but also on μ. Let us consider as an example the case of μ taking a value very close to the maximum allowable one, $\mu \simeq 2/\lambda_{\max}$. Then, the mode corresponding to the maximum eigenvalue will have an absolute value very close to one. On the other hand, the time constant of the mode corresponding to the minimum eigenvalue will be controlled by the value of $|1 - 2\lambda_{\min}/\lambda_{\max}|$, which can be much smaller than one. In such a case, the mode corresponding to the maximum eigenvalue exhibits slower convergence.

To obtain the optimum value for the step-size, one has to select its value in such a way that the resulting maximum absolute mode value is minimized. This is a min/max task,

$$\begin{aligned} \mu_o = \quad & \arg\min_\mu \max_j |1 - \mu\lambda_j|, \\ \text{s.t.} \quad & |1 - \mu\lambda_j| < 1,\ j = 1, 2, \ldots, l. \end{aligned}$$

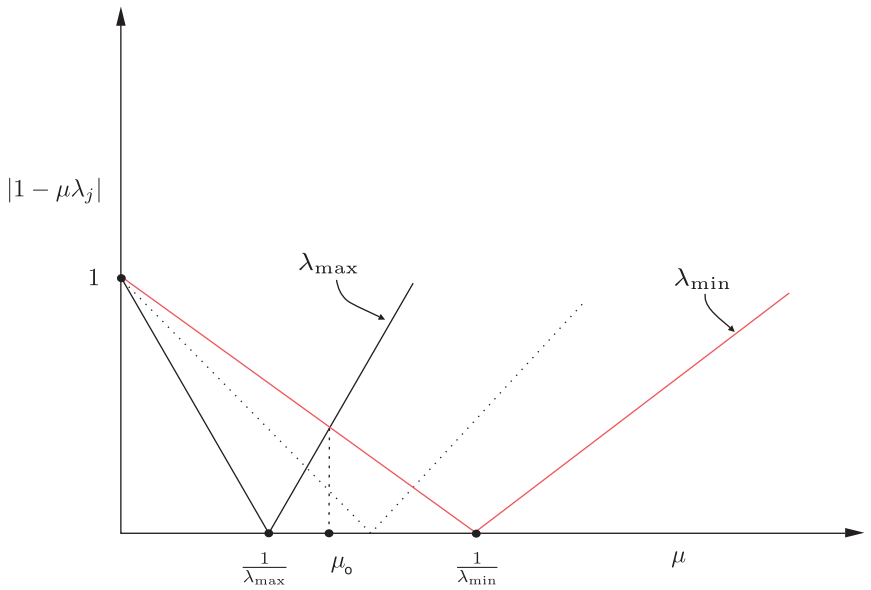


FIGURE 5.7

For each mode, increasing the value of the step-size, the time constant starts decreasing and then after a point starts increasing. The full black line corresponds to the maximum eigenvalue, the red one to the minimum, and the dotted curve to an intermediate eigenvalue. The overall optimal, μ_o, corresponds to the value where the red and the full black curves intersect.

The task can be solved easily graphically. Figure 5.7 shows the absolute values of the modes (corresponding to the maximum, minimum, and an intermediate one eigenvalues). The (absolute) values of the modes initially decrease, as μ increases and then they start increasing. Observe that the optimal value results at the point where the curves for the maximum and minimum eigenvalues intersect. Indeed, this corresponds to the minimum-maximum value. Moving μ away from μ_o, the maximum mode value increases; increasing μ_o, the mode corresponding to the maximum eigenvalue becomes larger and decreasing it, the mode corresponding to the minimum eigenvalue is increased. At the intersection, we have

$$1-\mu_o\lambda_{\min} = -(1-\mu_o\lambda_{\max}),$$

which results in

$$\mu_o = \frac{2}{\lambda_{\max}+\lambda_{\min}}. \tag{5.16}$$

At the optimal value, μ_o, there are two slowest modes; one corresponding to $\lambda_{\min}$ (i.e., $1-\mu_o\lambda_{\min}$) and another one corresponding to $\lambda_{\max}$ (i.e., $1-\mu_o\lambda_{\max}$). They have equal magnitudes but opposite signs, and they are given by,

$$\pm\frac{\rho-1}{\rho+1},$$

where

$$\rho := \frac{\lambda_{\max}}{\lambda_{\min}}.$$

In other words, the *convergence rate depends on the eigenvalues spread of the covariance matrix*.

Parameter Error Vector Convergence: From the definitions in (5.7) and (5.11), we get

$$\begin{aligned}\boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}_* + Q\boldsymbol{v}^{(i)} \\ &= \boldsymbol{\theta}_* + [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_l][v^{(i)}(1), \ldots, v^{(i)}(l)]^{\mathrm{T}} \\ &= \boldsymbol{\theta}_* + \sum_{k=1}^{l} \boldsymbol{q}_k v^{(i)}(k),\end{aligned} \tag{5.17}$$

or

$$\theta^{(i)}(j) = \theta_*(j) + \sum_{k=1}^{l} q_k(j) v^{(0)}(k)(1 - \mu\lambda_k)^i,\ j = 1, 2, \ldots l. \tag{5.18}$$

In other words, the components of $\boldsymbol{\theta}^{(i)}$ converge to the respective components of the optimum vector $\boldsymbol{\theta}_*$ as a weighted average of exponentials, $(1 - \mu\lambda_k)^i$. Computing the respective time constant in close form is not possible; however, we can state lower and upper bounds. The lower bound corresponds to the time constant of the fastest converging mode and the upper bound to the slowest of the modes. For small values of $\mu \ll 1$. we can write

$$\frac{1}{\mu\lambda_{\max}} \leq \tau \leq \frac{1}{\mu\lambda_{\min}}. \tag{5.19}$$

The Learning Curve: We now turn our focus on the mean-square error. Recall from (4.8) that

$$J(\boldsymbol{\theta}^{(i)}) = J(\boldsymbol{\theta}_*) + (\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*)^{\mathrm{T}} \Sigma_x (\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*), \tag{5.20}$$

or, mobilizing (5.17) and (5.9) and taking into consideration the orthonormality of the eigenvectors, we obtain

$$J(\boldsymbol{\theta}^{(i)}) = J(\boldsymbol{\theta}_*) + \sum_{j=1}^{l} \lambda_j |v^{(i)}(j)|^2 \implies$$

$$J(\boldsymbol{\theta}^{(i)}) = J(\boldsymbol{\theta}_*) + \sum_{j=1}^{l} \lambda_j (1 - \mu\lambda_j)^{2i} |v^{(0)}(j)|^2, \tag{5.21}$$

which converges to the minimum value $J(\boldsymbol{\theta}_*)$ asymptotically. Moreover, observe that this convergence is monotonic, because $\lambda_j(1 - \mu\lambda_j)^2$ is positive. Following similar arguments as before, the respective time constants for each one of the modes are now,

$$\tau_j^{\mathrm{mse}} = \frac{-1}{2\ln(1 - \mu\lambda_j)} \approx \frac{1}{2\mu\lambda_j}. \tag{5.22}$$

Example 5.1. The aim of the example is to demonstrate what we have said so far, concerning the convergence issues of the gradient descent scheme in (5.6). The cross-correlation vector was chosen to be

$$\boldsymbol{p} = [0.05, 0.03]^{\mathrm{T}},$$

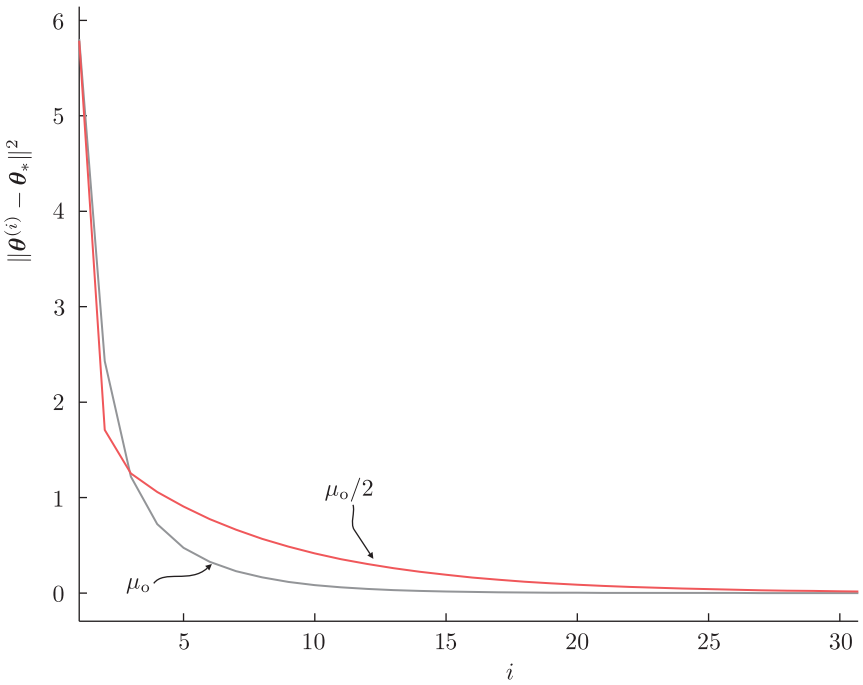


FIGURE 5.8

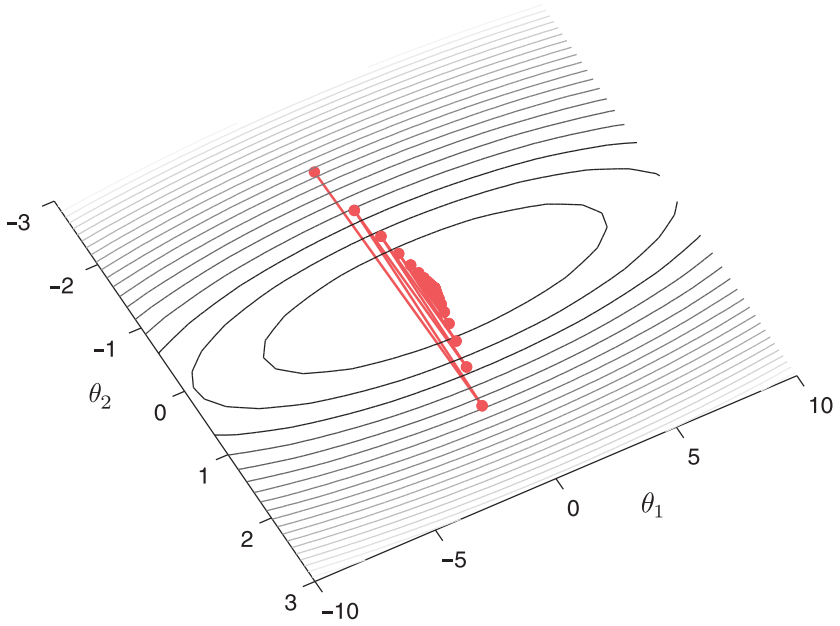
The black curve corresponds to the optimal value $\mu = \mu_o$ and the gray one to $\mu = \mu_o/2$, for the case of an input covariance matrix with unequal eigenvalues.

and we consider two different covariance matrices,

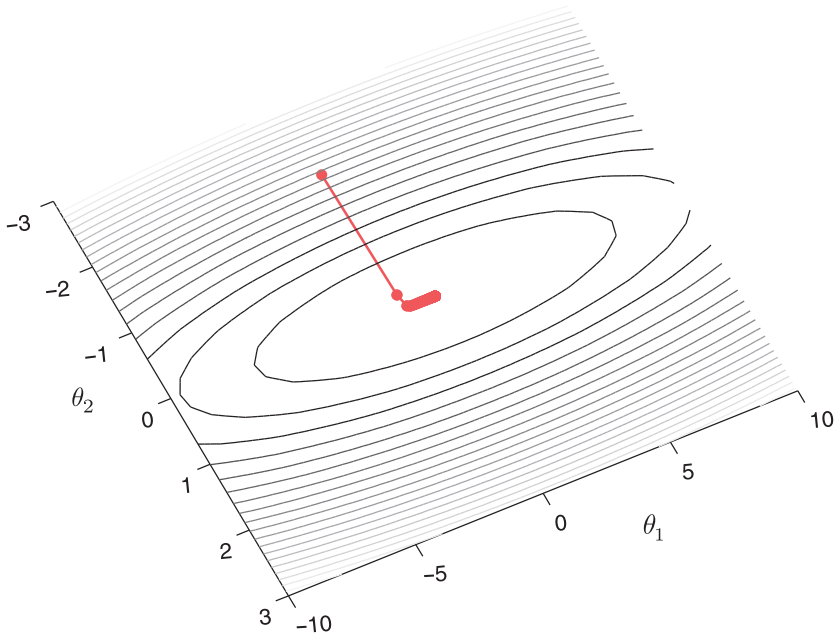
$$\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Note that, for the case of Σ_2, both eigenvalues are equal to 1, and for Σ_1 they are $\lambda_1 = 1$ and $\lambda_2 = 0.1$ (for diagonal matrices the eigenvalues are equal to the diagonal elements of the matrix).

Figure 5.8 shows the error curves for two values of μ, for the case of Σ_1; the gray one corresponds to the optimum value ($\mu_o = 1.81$) and the red one to $\mu = \mu_o/2 = 0.9$. Observe the faster convergence towards zero that is achieved by the optimal value. Note that it may happen, as is the case in Figure 5.8, that initially the convergence for some $\mu \neq \mu_o$ will be faster compared to μ_o. What the theory guarantees is that, eventually, the curve corresponding to the optimal will tend to zero faster than for any other value of μ. Figure 5.9 shows the respective trajectories of the successive estimates in the two-dimensional space, together with the isovalue curves; the latter are ellipses, as we can readily deduce if we look carefully at the form of the quadratic cost function written as in (5.20). Observe the zig-zag path, which corresponds to the larger value of $\mu = 1.81$ compared to the smoother one obtained for the smaller step-size $\mu = 0.9$.



(a)



(b)

FIGURE 5.9

The trajectories of the successive estimates (dots) obtained by the gradient descent algorithm for (a) the larger value of $\mu = 1.81$ and (b) for the smaller value of $\mu = 0.9$. In (b), the trajectory toward the minimum is smooth. In contrast, in (a), the trajectory consists of zig-zags.

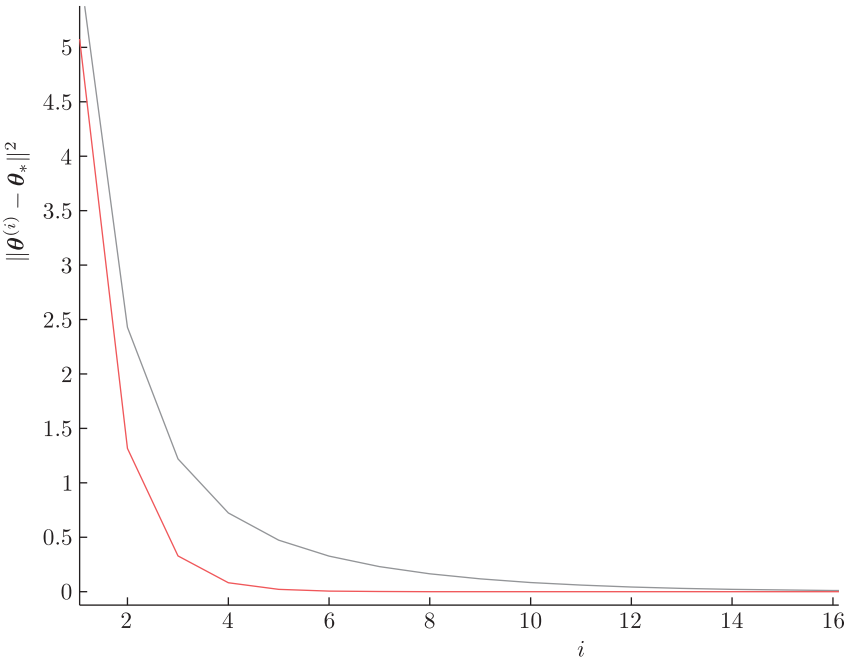


FIGURE 5.10

For the same value of $\mu = 1.81$, the error curves for the case of unequal eigenvalues ($\lambda_1 = 1$ and $\lambda_2 = 0.1$) (red) and for equal eigenvalues ($\lambda_1 = \lambda_2 = 1$). For the latter case, the isovalue curves are circles; if the optimal value $\mu_o = 1$ is used, the algorithm converges in one step. This is demonstrated in Figure 5.11.

For comparison reasons, to demonstrate the dependence of the convergence speed on the eigenvalues spread, Figure 5.10 shows the error curves using the same step size, $\mu = 1.81$, for both cases, Σ_1 and Σ_2. Observe that large eigenvalues spread of the input covariance matrix slows down the convergence rate. Note that if the eigenvalues of the covariance matrix are equal to, say, λ, the isovalue curves are circles; the optimal step size in this case is $\mu = 1/\lambda$ and convergence is achieved in only one step, Figure 5.11.

Time-varying step-sizes

The previous analysis cannot be carried out for the case of an iteration-dependent step-size. It can be shown (Problem 5.2), that in this case, the gradient descent algorithm converges if

- $\mu_i \longrightarrow 0$, as $i \longrightarrow \infty$
- $\sum_{i=1}^{\infty} \mu_i = \infty$.

A typical example of sequences, which comply with both conditions, are those that satisfy the following:

$$\sum_{i=1}^{\infty} \mu_i^2 < \infty, \quad \sum_{i=1}^{\infty} \mu_i = \infty, \tag{5.23}$$

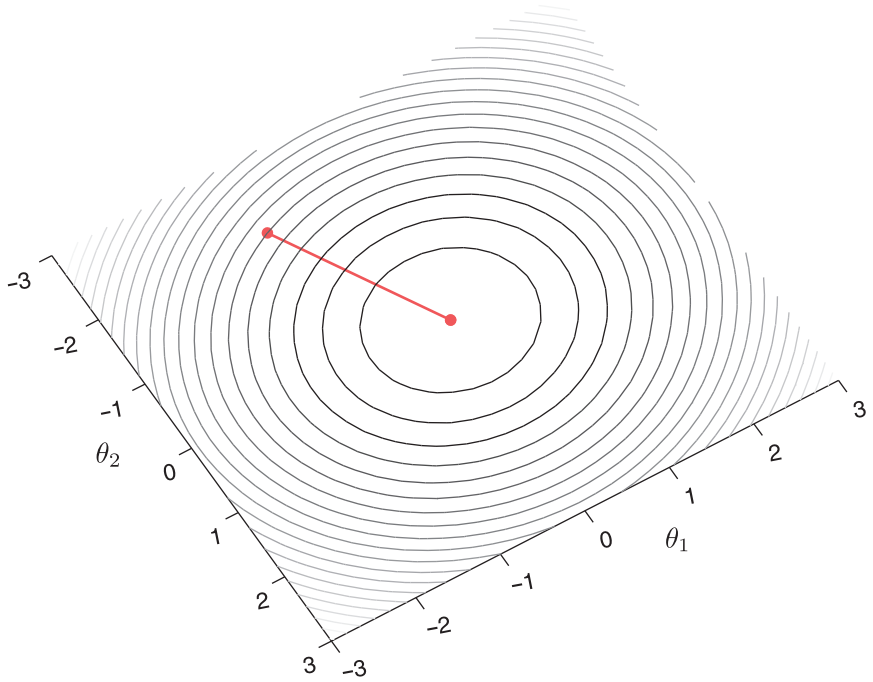


FIGURE 5.11

When the eigenvalues of the covariance matrix are all equal to a value, λ, the use of the optimal $\mu_o = 1/\lambda$ achieves convergence in one step.

as, for example, the sequence,

$$\mu_i = \frac{1}{i}.$$

Note that the two (sufficient) conditions require that the sequence tends to zero, yet its infinite sum diverges. We will meet this pair of conditions in various parts of this book. The previous conditions state that the step-size has to become smaller and smaller as iterations progress, but this should not take place in a very aggressive manner, so that the algorithm is left to be active for a sufficient number of iterations to learn the solution. If the step-size tends to zero very fast, then updates are practically frozen after a few iterations, without the algorithm having acquired enough information to get close to the solution.

5.3.1 THE COMPLEX-VALUED CASE

In Section 4.4.2, we stated that a function $f : \mathbb{C}^l \longmapsto \mathbb{R}$ is not differentiable with respect to its complex argument. To deal with such cases, the Wirtinger calculus was introduced. In this section, we use this mathematically convenient tool to derive the corresponding steepest descent direction.

To this end, we again employ a first order Taylor's series approximation [22]. Let

$$\boldsymbol{\theta} = \boldsymbol{\theta}_r + j\boldsymbol{\theta}_i.$$

Then, the cost function

$$J(\boldsymbol{\theta}) : \mathbb{C}^l \longmapsto [0, +\infty),$$

is approximated as

$$\begin{aligned} J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) &= J(\boldsymbol{\theta}_r + \Delta\boldsymbol{\theta}_r, \boldsymbol{\theta}_i + \Delta\boldsymbol{\theta}_i) \\ &= J(\boldsymbol{\theta}_r, \boldsymbol{\theta}_i) + \Delta\boldsymbol{\theta}_r^{\mathrm{T}} \nabla_r J(\boldsymbol{\theta}_r, \boldsymbol{\theta}_i) + \Delta\boldsymbol{\theta}_i^{\mathrm{T}} \nabla_i J(\boldsymbol{\theta}_r, \boldsymbol{\theta}_i), \end{aligned} \tag{5.24}$$

where ∇_r (∇_i) denotes the gradient with respect to $\boldsymbol{\theta}_r$ ($\boldsymbol{\theta}_i$). Taking into account that

$$\Delta\boldsymbol{\theta}_r = \frac{\Delta\boldsymbol{\theta} + \Delta\boldsymbol{\theta}^*}{2}, \quad \Delta\boldsymbol{\theta}_i = \frac{\Delta\boldsymbol{\theta} - \Delta\boldsymbol{\theta}^*}{2j},$$

it is easy to show (Problem 5.3) that

$$J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \mathrm{Re}\left\{\Delta\boldsymbol{\theta}^H \nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta})\right\}, \tag{5.25}$$

where $\nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta})$ is the CW-derivative, defined in Section 4.4.2 as

$$\nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta}) = \frac{1}{2}\left(\nabla_r J(\boldsymbol{\theta}) + j\nabla_i J(\boldsymbol{\theta})\right).$$

Looking carefully at (5.25), it is straightforward to observe that the direction

$$\Delta\boldsymbol{\theta} = -\mu \nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta}),$$

makes the updated cost equal to

$$J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) = J(\boldsymbol{\theta}) - \mu ||\nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta})||^2,$$

which guarantees that $J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) < J(\boldsymbol{\theta})$; it is straightforward to see, by taking into account the definition of an inner product, that the above search direction is one of the largest decrease. Thus, the counterpart of (5.3), becomes

$$\boxed{\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i \nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta}^{(i-1)}) : \quad \text{Complex Gradient Descent Scheme.}} \tag{5.26}$$

For the MSE cost function and for the linear estimation model, we get

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}\left[\left(\mathrm{y} - \boldsymbol{\theta}^H \mathbf{x}\right)\left(\mathrm{y} - \boldsymbol{\theta}^H \mathbf{x}\right)^*\right] \\ &= \sigma_y^2 + \boldsymbol{\theta}^H \Sigma_x \boldsymbol{\theta} - \boldsymbol{\theta}^H \boldsymbol{p} - \boldsymbol{p}^H \boldsymbol{\theta}, \end{aligned}$$

and taking the gradient with respect to $\boldsymbol{\theta}^*$, by treating $\boldsymbol{\theta}$ as a constant (Section 4.4.2), we obtain

$$\nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta}) = \Sigma_x \boldsymbol{\theta} - \boldsymbol{p}$$

and the respective gradient descent iteration is the same as in (5.6).

5.4 STOCHASTIC APPROXIMATION

Solving for the normal equations as well as using the gradient descent iterative scheme (for the case of the MSE), one has to have access to the second order statistics of the involved variables. However, in most of the cases, this is not known and it has to be approximated using a set of measurements. In this section, we turn our attention to algorithms that can learn the statistics iteratively via the training set. The origins of such techniques are traced back to 1951, when Robbins and Monro introduced the method of *stochastic approximation* [79] or the *Robbins-Monro algorithm*.

Let us consider the case of a function that is defined in terms of the expected value of another one, namely

$$f(\boldsymbol{\theta}) = \mathbb{E}\left[\phi(\boldsymbol{\theta}, \boldsymbol{\eta})\right], \quad \boldsymbol{\theta} \in \mathbb{R}^l,$$

where $\boldsymbol{\eta}$ is a random vector of unknown statistics. The goal is to compute a root of $f(\boldsymbol{\theta})$. If the statistics were known, the expectation could be computed, at least in principle, and one could use any root-finding algorithm to compute the roots. The problem emerges when the statistics are unknown, hence the exact form of $f(\boldsymbol{\theta})$ is not known. All one has at her/his disposal is a sequence of i.i.d. observations $\boldsymbol{\eta}_0, \boldsymbol{\eta}_1, \ldots$. Robbins and Monro proved that the following algorithm,[2]

$$\boxed{\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \phi(\boldsymbol{\theta}_{n-1}, \boldsymbol{\eta}_n): \quad \text{Robbins-Monro Scheme,}} \tag{5.27}$$

starting from an arbitrary initial condition, $\boldsymbol{\theta}_{-1}$, converges[3] to a root of $f(\boldsymbol{\theta})$, under some general conditions and provided that (Problem 5.4)

$$\boxed{\sum_n \mu_n^2 < \infty, \quad \sum_n \mu_n \longrightarrow \infty: \quad \text{Convergence Conditions.}} \tag{5.28}$$

In other words, in the iteration (5.27), we get rid of the expectation operation and use the value of $\phi(\cdot, \cdot)$, which is computed using the current observations/measurements and the currently available estimate. That is, the algorithm learns both the statistics as well as the root; two into one! The same comments made for the convergence conditions, met in the iteration-dependent step-size case in Section 5.3, are valid here, too.

In the context of optimizing a general differentiable cost function of the form,

$$J(\boldsymbol{\theta}) = \mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}, \mathrm{y}, \mathbf{x})\right], \tag{5.29}$$

Robbins-Monro scheme can be mobilized to find a root of the respected gradient, i.e.,

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}\left[\nabla \mathcal{L}(\boldsymbol{\theta}, \mathrm{y}, \mathbf{x})\right],$$

where the expectation is w.r.t. the pair $(\mathrm{y}, \mathbf{x})$. As we have seen in Chapter 3, such cost functions in the machine learning terminology are also known as the *expected risk* or the *expected loss*. Given the sequence of observations $(y_n, \boldsymbol{x}_n)$, $n = 0, 1, \ldots$, the recursion in (5.27) now becomes

$$\boxed{\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \nabla \mathcal{L}(\boldsymbol{\theta}_{n-1}, y_n, \boldsymbol{x}_n).} \tag{5.30}$$

[2] The original paper dealt with scalar variables only and the method was later extended to more general cases; see [96] for related discussion.

[3] Convergence here is meant to be in probability; see Section 2.6.

Let us now assume, for simplicity, that the expected risk has a unique minimum, $\boldsymbol{\theta}_*$. Then, according to Robbins-Monro theorem and using an appropriate sequence μ_n, $\boldsymbol{\theta}_n$ will converge to $\boldsymbol{\theta}_*$. However, although this information is important, it is not by itself enough. In practice, one has to seize iterations after a *finite* number of steps. Hence, one has to know something more concerning the rate of convergence of such a scheme. To this end, two quantities are of interest, namely the mean and the covariance matrix of the estimator at iteration n, or

$$\mathbb{E}\left[\boldsymbol{\theta}_n\right], \ \mathrm{Cov}(\boldsymbol{\theta}_n).$$

It can be shown (see [67]), that if $\mu_n = \mathcal{O}(1/n)$ and assuming that iterations have brought the estimate close to the optimal value, then

$$\boxed{\mathbb{E}\left[\boldsymbol{\theta}_n\right] = \boldsymbol{\theta}_* + \frac{1}{n}\boldsymbol{c},} \tag{5.31}$$

and

$$\boxed{\mathrm{Cov}(\boldsymbol{\theta}_n) = \frac{1}{n}V + \mathcal{O}(1/n^2),} \tag{5.32}$$

where $\boldsymbol{c}$ and V are constants that depend on the form of the expected risk. The above formulae have also been derived under some further assumptions concerning the eigenvalues of the Hessian matrix of the expected risk.[4] As we will also see, the convergence analysis of even simple algorithms is a tough task, and it is common to carry it under a number of assumptions. What is important from (5.31) and (5.32) is that both the mean as well as the standard deviations of the components follow a $\mathcal{O}(1/n)$ pattern. Furthermore, these formulae indicate that the parameter vector estimate *fluctuates* around the optimal value. This fluctuation depends on the choice of the sequence μ_n, being smaller for smaller values of the step-size sequence. However, μ_n cannot be made to decrease very fast due to the two convergence conditions, as discussed before. This is the price one pays for using the noisy version of the gradient and it is the reason that such schemes suffer from relatively slow convergence rates. However, this does not mean that such schemes are, necessarily, the poor relatives of other more "elaborate" algorithms. As we will discuss in Chapter 8, their low complexity requirements makes this algorithmic family to be the one that is selected in a number of practical applications.

Application to the MSE linear estimation

Let us apply the Robbins-Monro algorithm to solve for the optimal MSE linear estimator if the covariance matrix and the cross-correlation vector are unknown. We know that the solution corresponds to the root of the gradient of the cost function, which can be written in the form (recall the orthogonality theorem from Chapter 3),

$$\Sigma_x\boldsymbol{\theta} - \boldsymbol{p} = \mathbb{E}\left[\mathbf{x}(\mathbf{x}^{\mathrm{T}}\boldsymbol{\theta} - \mathrm{y})\right] = \mathbf{0}.$$

Given the training sequence of observations, $(y_n, \boldsymbol{x}_n)$, which are assumed to be i.i.d. drawn from the joint distribution of $(\mathrm{y}, \mathbf{x})$, the Robbins-Monro algorithm becomes,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu_n\boldsymbol{x}_n\left(y_n - \boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{\theta}_{n-1}\right), \tag{5.33}$$

[4] The proof is a bit technical and the interested reader can look at the provided reference.

which converges to the optimal MSE solution provided that the two conditions in (5.28) are satisfied. Compare (5.33) with (5.6). Taking into account the definitions, $\Sigma_x = \mathbb{E}[\mathbf{x}\mathbf{x}^{\mathrm{T}}]$, $\boldsymbol{p} = \mathbb{E}[\mathbf{x}y]$, the former equation results from the latter one by dropping out the expectation operations and using an iteration-dependent step size. Observe that the iterations in (5.33) coincide with *time updates*; time has now explicitly entered into the scene. This prompts us to start thinking about modifying such schemes appropriately to track time-varying environments. Algorithms such as the one in (5.33), which result from the generic gradient descent formulation by replacing the expectation by the respective instantaneous observations, are also known as *stochastic gradient descent schemes*.

Remarks 5.1.

- All the algorithms to be derived next can also be applied to nonlinear estimation/filtering tasks of the form,

$$\hat{y} = \sum_{k=1}^{l} \theta_k \phi_k(\mathbf{x}) = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\Phi},$$

  and the place of $\mathbf{x}$ is taken by $\boldsymbol{\Phi}$, where

$$\boldsymbol{\Phi} = [\phi_1(\mathbf{x}), \ldots, \phi_l(\mathbf{x})]^{\mathrm{T}}.$$

Example 5.2. The aim of this example is to demonstrate the pair of equations (5.31) and (5.32), which characterize the convergence properties of the stochastic gradient scheme.

Data samples were first generated according to the regression model

$$y_n = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n + \eta_n$$

where, $\boldsymbol{\theta} \in \mathbb{R}^2$ was randomly chosen and then fixed. The elements of $\boldsymbol{x}_n$ were i.i.d. generated via a normal distribution $\mathcal{N}(0, 1)$ and η_n are samples of a white noise sequence with variance equal to $\sigma^2 = 0.1$. Then, the observations $(y_n, \boldsymbol{x}_n)$ were used in the recursive scheme in (5.33) to obtain an estimate of $\boldsymbol{\theta}$. The experiment was repeated 200 times and the mean and variance of the obtained estimates were computed, for each iteration step. Figure 5.12 shows the resulting curve for one of the parameters (the trend for the other one being similar). Observe that the mean values of the estimates tend to the true value, corresponding to the red line and the standard deviation keeps decreasing as n grows. The step size was chosen equal to $\mu_n = 1/n$.

5.5 THE LEAST-MEAN-SQUARES ADAPTIVE ALGORITHM

The stochastic gradient algorithm in (5.33) converges to the optimal mean-square error solution provided that μ_n satisfies the two convergence conditions. Once the algorithm has converged, it "locks" at the obtained solution. In a case where the statistics of the involved variables/processes and/or the unknown parameters starts changing, the algorithm cannot track the changes. Note that if such changes occur, the error term

$$e_n = y_n - \boldsymbol{\theta}_{n-1}^{\mathrm{T}} \boldsymbol{x}_n$$

will get larger values; however, because μ_n is very small, the increased value of the error will not lead to corresponding changes of the estimate at time n. This can be overcome if one sets the value of μ_n

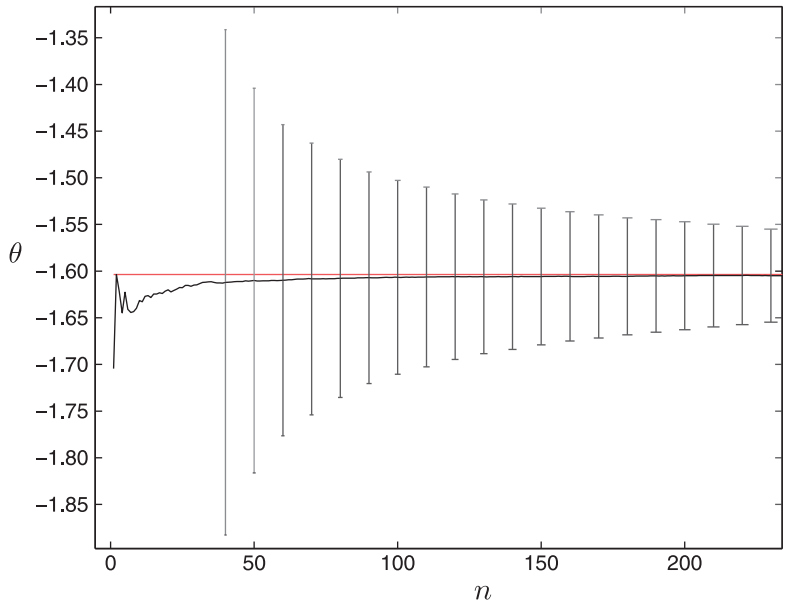


FIGURE 5.12

The red line corresponds to the true value of the unknown parameter. The black curve corresponds to the average over 200 realizations of the experiment. Observe that the mean value converges to the true value. The bars correspond to the respective standard deviation, which keeps decreasing as n grows.

to a preselected *fixed* value, μ. The resulting algorithm is the celebrated least-mean-squares (LMS) algorithm [102].

Algorithm 5.1 (The LMS algorithm).

- Initialize
  - $\boldsymbol{\theta}_{-1} = \mathbf{0} \in \mathbb{R}^l$; other values can also be used.
  - Select the value of μ.
- **For** $n = 0, 1, \ldots,$ **Do**
  - $e_n = y_n - \boldsymbol{\theta}_{n-1}^{\mathrm{T}} \boldsymbol{x}_n$
  - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu e_n \boldsymbol{x}_n$
- **End For**

In case the input is a time series,[5] u_n, the initialization also involves the samples, $u_{-1}, \ldots, u_{-l+1} = 0$, to form the input vectors, $\boldsymbol{u}_n, n = 0, 1, \ldots, l-2$. The complexity of the algorithm amounts to $2l$

[5] Recall our adopted notation from Chapter 2, that in this case we use $\boldsymbol{u}_n$ in place of $\boldsymbol{x}_n$.

multiplications/additions (MADs) per time update. We have assumed that observations start arriving at time instant $n = 0$, to be in line with most references treating the LMS.

Let us now comment on this simple structure. Assume that the algorithm has converged close to the solution; then the error term is expected to take small values and thus the updates will remain close to the solution. If the statistics and/or the system parameters now start changing, the error values are expected to increase. Given that μ has a constant value, the algorithm has now the "agility" to update the estimates in an attempt to "push" the error to lower values. This small variation of the iterative scheme has important implications. The resulting algorithm is no more a member of the Robbins-Monro stochastic approximation family. Thus, one has to study its convergence conditions as well as its performance properties. Moreover, since the algorithm now has the potential to track changes in the values of the underlying parameters, as well as the statistics of the involved processes/variables, one has to study its performance in nonstationary environments; this is associated to what is known as the *tracking* performance of the algorithm, and it will be treated at the end of the chapter.

5.5.1 CONVERGENCE AND STEADY-STATE PERFORMANCE OF THE LMS IN STATIONARY ENVIRONMENTS

The goal of this subsection is to study the performance of the LMS in stationary environments. That is, to answer the questions: (a) does the scheme converge and under which conditions? and (b) if it converges, where does it converge? Although we introduced the scheme having in mind nonstationary environments, still we have to know how it behaves under stationarity; after all, the environment can change very slowly, and it can be considered "locally" stationary.

The convergence properties of the LMS, as well as of any other online/adaptive algorithm, are related to its *transient* characteristics; that is, the period from the initial estimate until the algorithm reaches a "steady-state" mode of operation. In general, analyzing the transient performance of an online algorithm is a formidable task indeed. This is also true even for the very simple structure of the LMS summarized in Algorithm 5.1. The LMS update recursions are equivalent to a time-varying, nonlinear (Problem 5.5) and stochastic in nature estimator. Many papers, some of them of high scientific insight and mathematical skill, have been produced. However, with the exception of a few rare and special cases, the analysis involves approximations. Our goal in this book is not to treat this topic in detail. Our focus will be restricted on the most "primitive" of the techniques, which is easier for the reader to follow compared to more advanced and mathematically elegant theories; after all, even this primitive approach provides results that turn out to be in agreement to what one experiences in practice.

Convergence of the parameter error vector

Define

$$\boldsymbol{c}_n := \boldsymbol{\theta}_n - \boldsymbol{\theta}_*,$$

where $\boldsymbol{\theta}_*$ is the optimal solution resulting from the normal equations. The LMS update recursion can now be written as

$$\boldsymbol{c}_n = \boldsymbol{c}_{n-1} + \mu \boldsymbol{x}_n (y_n - \boldsymbol{\theta}_{n-1}^{\mathrm{T}} \boldsymbol{x}_n + \boldsymbol{\theta}_*^{\mathrm{T}} \boldsymbol{x}_n - \boldsymbol{\theta}_*^{\mathrm{T}} \boldsymbol{x}_n).$$

Because we are going to study the statistical properties of the obtained estimates, we have to switch our notation from that referring to observations to the one involving the respective random variables. Then we can write that,

$$\begin{aligned}\mathbf{c}_n &= \mathbf{c}_{n-1} + \mu\mathbf{x}(\mathrm{y} - \boldsymbol{\theta}_{n-1}^{\mathrm{T}}\mathbf{x} + \boldsymbol{\theta}_*^{\mathrm{T}}\mathbf{x} - \boldsymbol{\theta}_*^{\mathrm{T}}\mathbf{x}) \\ &= \mathbf{c}_{n-1} - \mu\mathbf{x}\mathbf{x}^{\mathrm{T}}\mathbf{c}_{n-1} + \mu\mathbf{x}\mathrm{e}_* \\ &= (I - \mu\mathbf{x}\mathbf{x}^{\mathrm{T}})\mathbf{c}_{n-1} + \mu\mathbf{x}\mathrm{e}_*, \end{aligned} \tag{5.34}$$

where

$$\mathrm{e}_* = \mathrm{y} - \boldsymbol{\theta}_*^{\mathrm{T}}\mathbf{x} \tag{5.35}$$

is the error random variable associated with the optimal $\boldsymbol{\theta}_*$. Compare (5.34) with (5.8). They look similar, yet they are very different. First, the latter of the two, involves the expected value, Σ_x, in place of the respective variables. Moreover in (5.34), there is a second term that acts as an external input to the difference stochastic equation. From (5.34), we obtain

$$\mathbb{E}[\mathbf{c}_n] = \mathbb{E}\left[(I - \mu\mathbf{x}\mathbf{x}^{\mathrm{T}})\mathbf{c}_{n-1}\right] + \mu\,\mathbb{E}[\mathbf{x}\mathrm{e}_*]. \tag{5.36}$$

To proceed, it is time to introduce assumptions.

Assumption 1. The involved random variables are jointly linked via the regression model,

$$\mathrm{y} = \boldsymbol{\theta}_o^{\mathrm{T}}\mathbf{x} + \eta, \tag{5.37}$$

where η is the noise variable with variance σ_η^2 and it is assumed to be independent of $\mathbf{x}$. Moreover, successive samples η_n, that generate the data, are assumed to be i.i.d. We have seen in Remarks 4.2 and Problem 4.4 that in this case, $\boldsymbol{\theta}_* = \boldsymbol{\theta}_o$, and $\sigma_{e_*}^2 = \sigma_\eta^2$. Also, due to the orthogonality condition, $\mathbb{E}[\mathbf{x}\mathrm{e}_*] = \mathbf{0}$. In addition, a stronger condition will be adopted, and e_* and $\mathbf{x}$ will be assumed to be *statistically independent*. This is justified by the fact that under the above model, $e_{*,n} = \eta_n$, and the noise sequence has been assumed to be independent of the input.

Assumption 2. (Independence Assumption) Assume that $\mathbf{c}_{n-1}$ is statistically independent of both $\mathbf{x}$ and e_*. No doubt this is a strong assumption, but one we will adopt to simplify computations. Sometimes there is a tendency to "justify" this assumption by resorting to some special cases, which we will not do. If one is not happy with the assumption, he/she has to look for more recent methods, based on more rigorous mathematical analysis; of course, this does not mean that such methods are free of assumptions.

I. *Convergence in the mean*: Having adopted the previous assumptions, (5.36) becomes

$$\begin{aligned}\mathbb{E}[\mathbf{c}_n] &= \mathbb{E}\left[\left(I - \mu\mathbf{x}\mathbf{x}^{\mathrm{T}}\right)\mathbf{c}_{n-1}\right] \\ &= (I - \mu\Sigma_x)\,\mathbb{E}[\mathbf{c}_{n-1}]. \end{aligned} \tag{5.38}$$

Following similar arguments as in Section 5.3, we obtain

$$\mathbb{E}[\mathbf{v}_n] = (I - \mu\Lambda)\,\mathbb{E}[\mathbf{v}_{n-1}],$$

where, $\Sigma_x = Q\Lambda Q^{\mathrm{T}}$ and $\mathbf{v}_n = Q^{\mathrm{T}}\mathbf{c}_n$. The last equation leads to

$$\mathbb{E}[\boldsymbol{\theta}_n] \longrightarrow \boldsymbol{\theta}_*, \quad \text{as } n \longrightarrow \infty,$$

provided that

$$0 < \mu < \frac{2}{\lambda_{\max}}.$$

In other words, in a stationary environment, the LMS converges to the optimal MSE solution *in the mean*. Thus, by fixing the value of the step-size to be a constant, we lose something; the obtained estimates, even after convergence, hover around the optimal solution. The obvious task to be considered next is to study the respective covariance matrix.

II. *Error vector covariance matrix*: From (5.38), applying it recursively and assuming for the initial condition that $\mathbb{E}[\mathbf{c}_{-1}] = \mathbf{0}$, we have that $\mathbb{E}[\mathbf{c}_n] = \mathbf{0}$. In any case, borrowing the same arguments used in establishing the convergence in the mean, it turns out that the latter is approximately true for large enough values of n, irrespective of the initialization. Thus, from (5.34) we get,

$$\begin{aligned}\Sigma_{c,n} := \mathbb{E}[\mathbf{c}_n\mathbf{c}_n^{\mathrm{T}}] = \Sigma_{c,n-1} - \mu\,\mathbb{E}[\mathbf{x}\mathbf{x}^{\mathrm{T}}\mathbf{c}_{n-1}\mathbf{c}_{n-1}^{\mathrm{T}}]\\ - \mu\,\mathbb{E}[\mathbf{c}_{n-1}\mathbf{c}_{n-1}^{\mathrm{T}}\mathbf{x}\mathbf{x}^{\mathrm{T}}] + \mu^2\,\mathbb{E}[\mathrm{e}_*^2\mathbf{x}\mathbf{x}^{\mathrm{T}}]\\ + \mu^2\,\mathbb{E}[\mathbf{x}\mathbf{x}^{\mathrm{T}}\mathbf{c}_{n-1}\mathbf{c}_{n-1}^{\mathrm{T}}\mathbf{x}\mathbf{x}^{\mathrm{T}}],\end{aligned} \tag{5.39}$$

where we have used the independence of e_* with $\mathbf{c}_{n-1}$ and the fact that e_* is orthogonal to $\mathbf{x}$ in order to set to zero some of the terms. Taking into consideration the adopted independence assumptions and assuming that the input vector follows a *Gaussian distribution*, (5.39) becomes

$$\begin{aligned}\Sigma_{c,n} = \Sigma_{c,n-1} - \mu\Sigma_x\Sigma_{c,n-1} - \mu\Sigma_{c,n-1}\Sigma_x\\ + 2\mu^2\Sigma_x\Sigma_{c,n-1}\Sigma_x + \mu^2\Sigma_x\mathrm{trace}\{\Sigma_x\Sigma_{c,n-1}\}\\ + \mu^2\sigma_\eta^2\Sigma_x,\end{aligned} \tag{5.40}$$

where the Gaussian assumption has been exploited to express the term involving fourth order moments (e.g., [74]) as

$$\mathbb{E}[\mathbf{x}\mathbf{x}^T\Sigma_{c,n-1}\mathbf{x}\mathbf{x}^T] = 2\Sigma_x\Sigma_{c,n-1}\Sigma_x + \Sigma_x\mathrm{trace}\{\Sigma_x\Sigma_{c,n-1}\}.$$

Mobilizing the definition of $\mathbf{v}_n = Q^{\mathrm{T}}\mathbf{c}_n$, (5.40) results in (Problem 5.6)

$$\begin{aligned}\Sigma_{v,n} = Q^T\Sigma_{c,n}Q = \Sigma_{v,n-1} - \mu\Lambda\Sigma_{v,n-1} - \mu\Sigma_{v,n-1}\Lambda\\ + 2\mu^2\Lambda\Sigma_{v,n-1}\Lambda + \mu^2\Lambda\mathrm{trace}\{\Lambda\Sigma_{v,n-1}\} + \mu^2\sigma_\eta^2\Lambda.\end{aligned} \tag{5.41}$$

Note that our interest lies at the diagonal elements of $\Sigma_{v,n}$, since these correspond to the variances of the respective elements of $\boldsymbol{\theta}_n - \boldsymbol{\theta}_*$, and correspondingly to $\mathbf{v}_n - \boldsymbol{v}_*$. Collecting all the *diagonal* elements in a vector, $\boldsymbol{s}_n$, a close inspection of the diagonal elements of $\Sigma_{v,n}$ in (5.41) can persuade the reader that the following difference equation is true,

$$\boldsymbol{s}_n = (I - 2\mu\Lambda + 2\mu^2\Lambda^2 + \mu^2\boldsymbol{\lambda}\boldsymbol{\lambda}^{\mathrm{T}})\boldsymbol{s}_{n-1} + \mu^2\sigma_\eta^2\boldsymbol{\lambda}, \tag{5.42}$$

where

$$\boldsymbol{\lambda} := [\lambda_1, \lambda_2, \ldots, \lambda_l]^{\mathrm{T}}.$$

It is well known from the linear system theory that, the difference equation in (5.42) is stable if the eigenvalues of the matrix,

$$A := I - 2\mu\Lambda + 2\mu^2\Lambda^2 + \mu^2\boldsymbol{\lambda}\boldsymbol{\lambda}^{\mathrm{T}}$$
$$= (I - \mu\Lambda)^2 + \mu^2\Lambda^2 + \mu^2\boldsymbol{\lambda}\boldsymbol{\lambda}^{\mathrm{T}}, \tag{5.43}$$

have magnitude less than one. This can be guaranteed if the step size μ is chosen such as (Problem 5.7)

$$0 < \mu < \frac{2}{\sum_{i=1}^{l}\lambda_i},$$

or

$$\boxed{\mu < \frac{2}{\mathrm{trace}\{\Sigma_x\}}.} \tag{5.44}$$

The last condition guarantees that the variances remain *bounded*. Recall the number of assumptions made. Thus, to be on the safe side, μ must be selected so that it is not close to this upper bound.

III. *Excess Mean-Square Error*: We know that the minimum MSE is achieved at $\boldsymbol{\theta}_*$. Any other weight vector results in higher values of the mean-square error. We have already said that in the steady-state, the estimates obtained via the LMS fluctuate randomly around $\boldsymbol{\theta}_*$; thus, the mean-square error will be larger than the minimum $J_{\min}$. This "extra" error power, denoted as J_{exc}, is known as the *excess* mean-square error. Also, the ratio

$$\mathcal{M} := \frac{J_{\mathrm{exc}}}{J_{\min}},$$

is known as the *misadjustment*. No doubt, we should seek for the relationship of $\mathcal{M}$ with μ and in practice we would like to adjust μ accordingly, in order to get a value of $\mathcal{M}$ as small as possible. Unfortunately, we will soon see that there is a trade-off in achieving that. Making $\mathcal{M}$ small, the convergence speed becomes slower and vice versa; there is no free lunch!

By the respective definitions, we have[6]

$$\mathrm{e}_n = \mathrm{y}_n - \boldsymbol{\theta}_{n-1}^{\mathrm{T}}\mathbf{x}$$
$$= \mathrm{e}_{*,n} - \mathbf{c}_{n-1}^{\mathrm{T}}\mathbf{x},$$

or

$$\mathrm{e}_n^2 = \mathrm{e}_{*,n}^2 + \mathbf{c}_{n-1}^{\mathrm{T}}\mathbf{x}\mathbf{x}^{\mathrm{T}}\mathbf{c}_{n-1} - 2\mathrm{e}_{*,n}\mathbf{c}_{n-1}^{\mathrm{T}}\mathbf{x}. \tag{5.45}$$

Taking the expectation on both sides and exploiting the assumed independence between $\mathbf{c}_{n-1}$ and $\mathbf{x}$ and $\mathrm{e}_{*,n}$, as well as the orthogonality between $\mathrm{e}_{*,n}$ and $\mathbf{x}$, we get

$$J_n := \mathbb{E}[\mathrm{e}_n^2] = J_{\min} + \mathbb{E}\left[\mathbf{c}_{n-1}^{T}\mathbf{x}\mathbf{x}^{T}\mathbf{c}_{n-1}\right]$$
$$= J_{\min} + \mathbb{E}\left[\mathrm{trace}\{\mathbf{c}_{n-1}^{T}\mathbf{x}\mathbf{x}^{T}\mathbf{c}_{n-1}\}\right]$$
$$= J_{\min} + \mathrm{trace}\{\Sigma_x\Sigma_{c,n-1}\}, \tag{5.46}$$

[6] The time index n is explicitly used for $\mathrm{e}, \mathrm{e}_*, \mathrm{y}$, since the formula to be derived is also valid for time-varying environments, and it is going to be used later on for the time-varying statistics case.

where the property trace$\{AB\}$ = trace$\{BA\}$ has been used. Thus, we can finally write that,

$$\boxed{J_{\text{exc},n} = \text{trace}\{\Sigma_x \Sigma_{c,n-1}\}: \quad \text{Excess MSE at Time Instant } n.} \tag{5.47}$$

Let us now elaborate on it a bit more. Taking into account that $QQ^{\mathrm{T}} = I$, we get

$$\begin{aligned} J_{\text{exc},n} &= \text{trace}\{QQ^T \Sigma_x QQ^T \Sigma_{c,n-1} QQ^T\} \\ &= \text{trace}\{Q\Lambda \Sigma_{v,n-1} Q^T\} = \text{trace}\{\Lambda \Sigma_{v,n-1}\} \\ &= \sum_{i=1}^{l} \lambda_i \left[\Sigma_{v,n-1}\right]_{ii} = \boldsymbol{\lambda}^T \boldsymbol{s}_{n-1}, \end{aligned} \tag{5.48}$$

where $\boldsymbol{s}_n$ is the vector of the diagonal elements of $\Sigma_{v,n}$ and obeys the difference equation in (5.42). Assuming that μ has been chosen so that convergence is guaranteed, then for large values of n the *steady-state* has been reached. In a more formal way, an online algorithm has reached the steady state if

$$\mathbb{E}[\theta_n] = \mathbb{E}[\theta_{n-1}] = \text{Constant}, \tag{5.49}$$

$$\Sigma_{\theta,n} = \Sigma_{\theta,n-1} = \text{Constant}. \tag{5.50}$$

Thus, in steady-state, we assume in Eq. (5.42) that $\boldsymbol{s}_n = \boldsymbol{s}_{n-1}$; if this is exploited in (5.48) leads to (Problem 5.10),

$$J_{\text{exc},\infty} := \lim_{n\to\infty} J_{\text{exc},n} \simeq \frac{\mu\sigma_\eta^2 \text{trace}\{\Sigma_x\}}{2 - \mu\text{trace}\{\Sigma_x\}}, \tag{5.51}$$

and for the misadjustment (since under our assumptions $J_{\min} = \sigma_\eta^2$),

$$\mathcal{M} \simeq \frac{\mu\text{trace}\{\Sigma_x\}}{2 - \mu\text{trace}\{\Sigma_x\}},$$

which, for small values of μ, leads to

$$\boxed{J_{\text{exc},\infty} \simeq \frac{1}{2}\mu\sigma_\eta^2 \text{trace}\{\Sigma_x\}: \quad \text{Excess MSE},} \tag{5.52}$$

and

$$\boxed{\mathcal{M} \simeq \frac{1}{2}\mu\text{trace}\{\Sigma_x\}: \quad \text{Misadjustment}.} \tag{5.53}$$

That is, the smaller the value of μ the smaller the excess mean-square error.

IV. *Time constant*. Note that the transient behavior of the LMS is described by the difference equation in (5.42) and its convergence rate (the speed with which it forgets the initial conditions till it settles to its steady-state operation) depends on the eigenvalues of A in (5.43). To simplify the formulas, assume μ to be small enough so that A is approximated as $(I - \mu\Lambda)^2$. Following similar arguments as those used for the gradient descent in Section 5.3, we can write that

$$\tau_j^{LMS} \simeq \frac{1}{2\mu\lambda_j}.$$

That is, the time constant for each one of the modes is inversely proportional to μ. Hence, the slower the rate of convergence (small values of μ) the lower the misadjustment and vice versa. Viewing it differently, the more time the algorithm spends on learning, prior to reaching the steady-state, the smaller is its deviation from the optimal.

5.5.2 CUMULATIVE LOSS BOUNDS

In the previously reported analysis method for the LMS performance, there is an underlying assumption that the training samples are generated by a linear model. The focus of the analysis was to investigate how well our algorithm estimates the unknown model, once steady-state has been reached. This path of analysis is very popular and well suited for a number of tasks such as system identification.

An alternative route for studying the performance of an algorithm, shedding light from a different angle, is via the so-called *cumulative loss*. Recall that the main goal in machine learning is *prediction*; hence, measuring the prediction accuracy of an algorithm, given a set of observations, becomes the main goal. However, this performance index should be measured against the generalization ability of the algorithm, as pointed out in Chapter 3. In practice, this can be done in different ways, such as via the Leave-One-Out method, Section 3.13.

For the case of the squared error loss function, the cumulative loss over N observation samples is defined as

$$\boxed{\mathcal{L}_{\text{cum}} = \sum_{n=0}^{N-1}(y_n - \hat{y}_n)^2 = \sum_{n=0}^{N-1}(y_n - \boldsymbol{\theta}_{n-1}^{\mathrm{T}}\boldsymbol{x}_n)^2 : \quad \text{Cumulative Loss.}} \tag{5.54}$$

Note that $\boldsymbol{\theta}_{n-1}$ has been estimated based on observations up to and including the time instant $n-1$. So, the training pair $(y_n, \boldsymbol{x}_n)$, can be considered as a test sample, not involved in the training, for measuring the error. It must be pointed out, however, that the cumulative loss is *not* a direct measure of the generalization performance associated with the finally obtained parameter vector. Such a measure should involve, $\boldsymbol{\theta}_{N-1}$, tested against a number of test samples.

The goal of the family of methods, which build around the cumulative loss, is to derive corresponding upper bounds. Our aim here is to outline the essence behind such approaches, without resorting to proofs; these comprise a series of bounds and can become a bit technical. The interested reader can consult the related references. We will return to the cumulative loss and related bounds in the context of the so-called *regret analysis* in Chapter 8.

In the context of the LMS, the following theorem has been proved in [21].

Theorem 5.1. *Let* $C = \max_n ||\boldsymbol{x}_n||$, $\mu = \beta/C^2$, $0 < \beta < 2$. *Then, the set of predictions,* $\hat{y}_0, \ldots, \hat{y}_{N-1}$, *generated by the Algorithm 5.1, satisfies the following bound*

$$\sum_{n=0}^{N-1}(y_n - \hat{y}_n)^2 \leq \inf_{\boldsymbol{\theta}}\left\{\frac{C^2||\boldsymbol{\theta}||^2}{2\beta(1-\beta)c} + \frac{\mathcal{L}(\boldsymbol{\theta}, S)}{(2-\beta)^2 c(1-c)}\right\}, \tag{5.55}$$

where, $0 < c < 1$ *and*

$$\mathcal{L}(\boldsymbol{\theta}, S) = \sum_{n=0}^{N-1}(y_n - \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n)^2, \tag{5.56}$$

where $S = \{(y_n, \boldsymbol{x}_n), \quad n = 0, 1, \ldots, N-1\}$.

One can then tune β optimally to minimize the upper bound. Note that this is a *worst-case* scenario. The tuning is achieved by restricting the set of linear functions, so that $||\boldsymbol{\theta}|| \leq \Theta$. Let

$$L_{\Theta}(S) = \min_{||\boldsymbol{\theta}|| \leq \Theta} \mathcal{L}(\boldsymbol{\theta}, S), \tag{5.57}$$

and also assume that there is an upper bound L, such as

$$|L_{\Theta}(S)| \leq L. \tag{5.58}$$

Then, it can be shown [21] that

$$\sum_{n=0}^{N-1} (y_n - \hat{y}_n)^2 \leq L_{\Theta}(S) + 2\Theta C\sqrt{L} + (\Theta C)^2. \tag{5.59}$$

Note that the previous analysis has been carried out without invoking any probabilistic arguments. Alternative bounds are derived by mobilizing different assumptions [21]. Bounds of this kind, involving similar assumptions, are frequently encountered for the analysis of various algorithms. We will meet such examples later in this book.

Remarks 5.2.

- The analysis method presented in this section, concerning the transient and steady-state performance of the LMS, can be considered as the most primitive and goes back to the early work of Widrow and Hopf, [102]. Another popular path for analyzing stochastic algorithms in an averaging sense is the so-called *averaging method* that operates under the assumption of small values of the step-size, μ [56]. Another approach that builds around the assumption of small step sizes, $\mu \simeq 0$, is the so-called ordinary-differential-equation approach (ODE) [57]. The difference update equation is "transformed" to a differential equation, which paves the way for using arguments from the Lyapunov stability theory. An alternative elegant theoretical tool, which can be used as a vehicle for analyzing the transient, the steady-state as well as the tracking performance of adaptive schemes is the so-called *energy conservation* method, developed by Sayed and Rupp [82] and later on extended in Refs. [3, 105]. More on the performance analysis of the LMS as well as of other online schemes can be found in more specialized books and papers [4, 13, 47, 62, 83, 91, 92, 96, 103].
- *H^{∞} optimality of the LMS*. It may come as a surprise that the LMS algorithm, a very simple structure that was obtained via an approximation, has survived the time and is one of the most popular and widely used schemes in practical applications. The reason is that, besides its low complexity, it enjoys the luxury of robustness. An alternative optimization flavor of the LMS algorithm has been given via the theory of H^{∞} optimization for estimation.

  Assume that our data obey the regression model of (5.37), where now no assumption is made on the nature of η. Given the sequence of output observation samples, $y_0, y_1, \ldots, y_{N-1}$, the goal is to obtain estimates, $\hat{s}_{n|n-1}$, of s_n, generated as $s_n = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n$, based on the training set up to and including time $n-1$ (causality), such that

$$\frac{\sum_{n=0}^{N-1} |\hat{s}_{n|n-1} - s_n|^2}{\mu^{-1}||\boldsymbol{\theta}||^2 + \sum_{n=0}^{N-1} |\eta_n|^2} < \gamma^2. \tag{5.60}$$

  The numerator is the total squared estimation error. The denominator involves two terms. One is the noise/disturbance energy, and the other is the norm of the unknown parameter vector.

Assuming that one starts iterations from $\boldsymbol{\theta}_{-1} = \mathbf{0}$, this term measures the energy of the disturbance from the initial guess. It turns out that the LMS scheme is the one that minimizes the following cost

$$\gamma_{\text{opt}}^2 = \inf_{\{\hat{s}_{n|n-1}\}} \sup_{\{\boldsymbol{\theta},\eta_n\}} \left(\frac{\sum_{n=0}^{\infty} |\hat{s}_{n|n-1} - s_n|^2}{\mu^{-1}||\boldsymbol{\theta}||^2 + \sum_{n=0}^{\infty} |\eta_n|^2} \right).$$

Moreover, it turns out that the optimum corresponds to $\gamma_{\text{opt}}^2 = 1$ [46, 83]. Note that, basically, the LMS optimizes a worst-case scenario. It makes the estimation error minimum under the worst (maximum) disturbance circumstances. This type of optimality explains the robust performance of the LMS under "nonideal" environments, which are often met in practice, where a number of modeling assumptions are not valid. Such deviations from the model can be accommodated in η_n, which then "loses" its i.i.d., white, Gaussian, or any other mathematically attractive property. Finally, it is interesting to point out the similarity of the bound in (5.60) with that in (5.55). Indeed, in the former make the following substitutions,

$$\hat{s}_{n|n-1} = \hat{y}_n, \quad s_n = y_n, \quad \eta_n = y_n - \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n.$$

Ignoring the values of the constants, the involved quantities are the same. After all, H^{∞} is about maximizing the worst-case scenario [55].

5.6 THE AFFINE PROJECTION ALGORITHM

As it will soon be verified in the simulations section, a major drawback of the basic LMS scheme is its fairly slow convergence speed. In an attempt to improve upon it, a number of variants have been proposed over the years. The *affine projection algorithm* (APA) belongs to the so-called *data-reusing* family, where, at each time instant, past data are reused. Such a rationale helps the algorithm to "learn faster" and improve the convergence speed. However, besides the increased complexity, the faster convergence speed is achieved at the *expense of an increased misadjustment level*.

The APA was proposed originally in Ref. [48] and later on in Ref. [72]. Let the currently available estimate be $\boldsymbol{\theta}_{n-1}$. According to APA, the updated estimate, $\boldsymbol{\theta}$, must satisfy the following constraints:

$$\boldsymbol{x}_{n-i}^{\mathrm{T}}\boldsymbol{\theta} = y_{n-i}, \quad i = 0, 1, \ldots, q-1.$$

In other words, we *force* the parameter vector, $\boldsymbol{\theta}$, to provide at its output the desired response samples, for the q most recent time instants, where q is a user-defined parameter. At the same time, APA requires $\boldsymbol{\theta}$ to be as close as possible, in the Euclidean norm sense, to the current estimate, $\boldsymbol{\theta}_{n-1}$. Thus, APA, at each time instant, solves the following constrained optimization task,

$$\begin{aligned} &\boldsymbol{\theta}_n = \arg\min_{\boldsymbol{\theta}} ||\boldsymbol{\theta} - \boldsymbol{\theta}_{n-1}||^2 \\ &\text{s.t.} \qquad \boldsymbol{x}_{n-i}^{\mathrm{T}}\boldsymbol{\theta} = y_{n-i}, \quad i = 0, 1, \ldots, q-1. \end{aligned} \tag{5.61}$$

If one defines the $q \times l$ matrix

$$X_n = \begin{bmatrix} \boldsymbol{x}_n^{\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_{n-q+1}^{\mathrm{T}} \end{bmatrix},$$

then the set of constraints can be compactly written as,

$$X_n\boldsymbol{\theta} = \boldsymbol{y}_n,$$

where

$$\boldsymbol{y}_n = [y_n \ldots y_{n-q+1}]^{\mathrm{T}}.$$

Using Lagrange multipliers in (5.61) results in (Problem 5.11),

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + X_n^{\mathrm{T}}\left(X_n X_n^{\mathrm{T}}\right)^{-1}\boldsymbol{e}_n, \tag{5.62}$$

$$\boldsymbol{e}_n = \boldsymbol{y}_n - X_n\boldsymbol{\theta}_{n-1}, \tag{5.63}$$

provided that $X_n X_n^{\mathrm{T}}$ is invertible. The resulting scheme is summarized in Algorithm 5.2.

Algorithm 5.2 (The affine projection algorithm).

- Initialize
  - $\boldsymbol{x}_{-1} = \ldots = \boldsymbol{x}_{-q+1} = \mathbf{0}$, $y_{-1} \ldots y_{-q+1} = 0$
  - $\boldsymbol{\theta}_{-1} = \mathbf{0} \in \mathbb{R}^l$ (or any other value).
  - Choose $0 < \mu < 2$ and δ to be small.
- **For** $n = 0, 1, \ldots,$ **Do**
  - $\boldsymbol{e}_n = \boldsymbol{y}_n - X_n\boldsymbol{\theta}_{n-1}$
  - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu X_n^{\mathrm{T}}\left(\delta I + X_n X_n^{\mathrm{T}}\right)^{-1}\boldsymbol{e}_n$
- **End For**

When the input is a time series, the corresponding input vector, denoted as $\boldsymbol{u}_n$, is initialized by setting to zero all required samples with negative time index, $u_{-1}, u_{-2}, \ldots$. Note that in the algorithm, a parameter, δ, of a small value has also been used to prevent numerical problems in the associated matrix inversion. Also, a step-size μ has been introduced, that controls the size of the update and whose presence will be justified soon. The complexity of APA is increased, compared to that of the LMS, due to the involved matrix inversion and matrix operations, requiring $O(q^3)$ MADs. Fast versions of the APA, which exploit the special structure of $X_n X_n^{\mathrm{T}}$, for the case where the involved input-output variables are realizations of stochastic processes, have been developed, see [42, 43].

The convergence analysis of the APA is more involved than that of the LMS. It turns out that provided that $0 < \mu < 2$, stability of the algorithm is guaranteed. The misadjustment is approximately given by [2, 34, 83]

$$\boxed{\mathcal{M} \simeq \frac{\mu q \sigma_\eta^2}{2-\mu}\,\mathbb{E}\left[\frac{1}{||\mathbf{x}_n||^2}\right]\mathrm{trace}\{\Sigma_x\}: \quad \text{Misadjustment for the APA.}}$$

In words, the misadjustment increases as the parameter q increases; that is, as the number of the reused past data samples increases.

Geometric interpretation of APA

Let us look at the optimization task in (5.61), associated with APA. Each one of the q constraints defines a hyperplane in the l-dimensional space. Hence, since $\boldsymbol{\theta}_n$ is constrained to lie on all these hyperplanes, it will lie in their *intersection*. Provided that $\boldsymbol{x}_{n-i}$, $i = 0, \ldots, q-1$, are linearly independent, these

hyperplanes share a nonempty intersection, which is an *affine set* of dimension $l - q$. An affine set is the translation of a linear subspace (i.e., a plane crossing the origin) by a constant vector; that is, it defines a plane in a general position. Thus, $\boldsymbol{\theta}_n$ can lie anywhere in this affine set. From the infinite number of points lying in this set, APA selects the one that lies closest, in the Euclidean distance sense, to $\boldsymbol{\theta}_{n-1}$. In other words, $\boldsymbol{\theta}_n$ is the *projection* of $\boldsymbol{\theta}_{n-1}$ on the affine set defined by the intersection of the q hyperplanes. Recall from geometry that, the projection $P_H(\boldsymbol{a})$ of a point $\boldsymbol{a}$ on a linear subspace/affine set, H, is the point in H whose distance from $\boldsymbol{a}$ is minimum. Figure 5.13 illustrates the geometry for the case of $q = 2$; this special case of APA is also known as the binormalized data reusing LMS [7].

In the ideal noiseless case, the unknown parameter vector would lie in the intersection of all the hyperplanes defined by $(y_n, \boldsymbol{x}_n)$, $n = 0, 1, \ldots, q - 1$, and this is the information that APA tries to exploit to speed up convergence. However, this is also its drawback. In any practical system, noise is present;

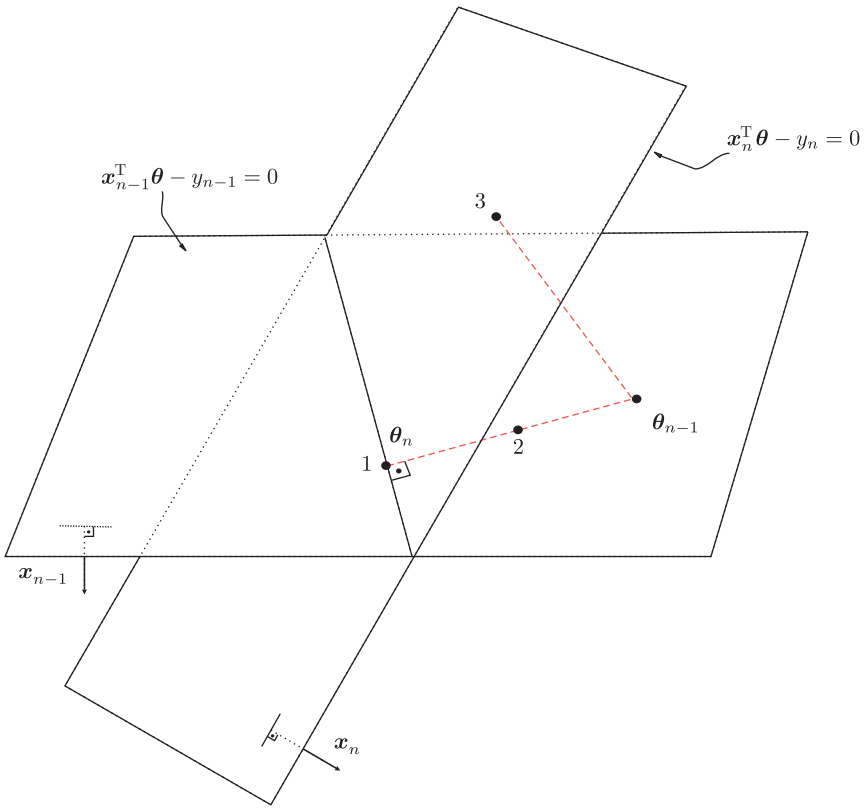


FIGURE 5.13

The geometry associated with the APA algorithm, for $q = 2$ and $l = 3$. The intersection of the two hyperplanes is a straight line (affine set of dimension $3 - 2 = 1$). $\boldsymbol{\theta}_n$ is the projection of $\boldsymbol{\theta}_{n-1}$ on this line (point 1) for $\mu = 1$ and $\delta = 0$. Point 2 corresponds to the case $\mu < 1$. Point 3 is the projection of $\boldsymbol{\theta}_n$ on the hyperplane defined by $(y_n, \boldsymbol{x}_n)$. This is the case for $q = 1$. The latter case corresponds to the normalized LMS of Section 5.6.1.

thus forcing the updates to lie in the intersection of these hyperplanes is not necessarily good, since their position in space is also determined by the noise. As a matter of fact, the reason that μ is introduced is to account for such cases. An alternative technique, which exploits projections, and at the same time replaces hyperplanes by hyperslabs (whose width depends on the noise variance) to account for the noise, will be treated in Chapter 8. In addition there, no matrix inversion will be required.

Orthogonal projections

Projections and projection matrices/operators play a crucial part in machine learning, signal processing and optimization in general; after all, a projection corresponds to a minimization task, when the loss is interpreted as a "distance." Given an $l \times k$, $k < l$ matrix, A, with column vectors, $\boldsymbol{a}_i$, $i = 1, \ldots, k$, and an l-dimensional vector $\boldsymbol{x}$, then the orthogonal projection of $\boldsymbol{x}$ on the subspace spanned by the columns of A (assumed to be linearly independent) is given by

$$\boxed{P_{\{\boldsymbol{a}_i\}}(\boldsymbol{x}) = A(A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}\boldsymbol{x},} \tag{5.64}$$

where in complex spaces the transpose operation is replaced by the Hermitian one. One can easily check that $P^{\perp}_{\{\boldsymbol{a}_i\}}(\boldsymbol{x}) := (I - P_{\{\boldsymbol{a}_i\}})\boldsymbol{x}$ is orthogonal to $P_{\{\boldsymbol{a}_i\}}(\boldsymbol{x})$ and

$$\boxed{\boldsymbol{x} = P_{\{\boldsymbol{a}_i\}}(\boldsymbol{x}) + P^{\perp}_{\{\boldsymbol{a}_i\}}(\boldsymbol{x}).}$$

When A has orthonormal columns, we obtain our familiar from geometry expansion

$$P_{\{\boldsymbol{a}_i\}}(\boldsymbol{x}) = AA^{\mathrm{T}}\boldsymbol{x} = \sum_{i=1}^{k}(\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x})\boldsymbol{a}_i.$$

Thus, the factor $(A^{\mathrm{T}}A)^{-1}$, for the general case, accounts for the lack of orthonormality of the columns. The matrix $A(A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}$ is known as the respective *projection matrix* and $I - A(A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}$ as the projection matrix on the respective *orthogonal complement* space.

The simplest case occurs when $k = 1$; then the projection of $\boldsymbol{x}$ onto $\boldsymbol{a}_1$ is equal to

$$P_{\{\boldsymbol{a}_1\}}(x) = \frac{\boldsymbol{a}_1\boldsymbol{a}_1^{\mathrm{T}}}{||\boldsymbol{a}_1||^2}\boldsymbol{x},$$

and the corresponding projection matrices are given by,

$$P_{\{\boldsymbol{a}_1\}} = \frac{\boldsymbol{a}_1\boldsymbol{a}_1^{\mathrm{T}}}{||\boldsymbol{a}_1||^2}, \quad P^{\perp}_{\{\boldsymbol{a}_1\}} = I - \frac{\boldsymbol{a}_1\boldsymbol{a}_1^{\mathrm{T}}}{||\boldsymbol{a}_1||^2}.$$

Figure 5.14 illustrates the geometry.

Let us apply the previously reported linear algebra results in the case of the APA algorithm of (5.62) and (5.63), and rewrite them as

$$\begin{aligned}\boldsymbol{\theta}_n = &\left(I - X_n^{\mathrm{T}}(X_nX_n^{\mathrm{T}})^{-1}X_n\right)\boldsymbol{\theta}_{n-1}\\ &+ X_n^{\mathrm{T}}(X_nX_n^{\mathrm{T}})^{-1}\boldsymbol{y}_n.\end{aligned}$$

The first term on the right-hand side is the projection $P^{\perp}_{\{\boldsymbol{x}_n,\ldots,\boldsymbol{x}_{n-q+1}\}}(\boldsymbol{\theta}_{n-1})$. This is the most natural. By the definition of the respective affine set, as the intersection of the hyperplanes

$$\boldsymbol{x}_{n-i}^{\mathrm{T}}\boldsymbol{\theta} - y_{n-i} = 0, \quad i = 0, \ldots, q-1,$$

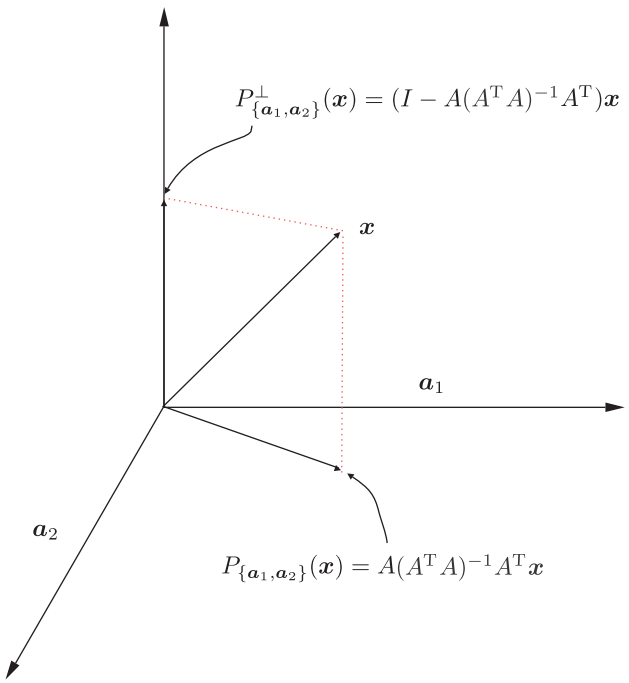


FIGURE 5.14

Geometry indicating the orthogonal projection operation on the subspace spanned by $\boldsymbol{a}_1$, $\boldsymbol{a}_2$, using the projection matrix. Note that $A = [\boldsymbol{a}_1, \boldsymbol{a}_2]$.

each vector $\boldsymbol{x}_{n-i}$ is *orthogonal* to the respective hyperplane (Figure 5.13) (Problem 5.12). Hence, projecting $\boldsymbol{\theta}_{n-1}$ on the intersection of all these hyperplanes is equivalent to projecting on an affine set, which is *orthogonal* to all $\boldsymbol{x}_n, \ldots, \boldsymbol{x}_{n-q+1}$. Note that the matrix $X_n^T(X_nX_n^T)^{-1}X_n$ is the projection matrix that projects on the subspace spanned by $\boldsymbol{x}_n, \ldots, \boldsymbol{x}_{n-q+1}$. The second term accounts for the fact that the affine set, on which we project, does not include the origin, but it is translated to another point in the space, whose direction is determined by $\boldsymbol{y}_n$. Figure 5.15 illustrates the case for $l = 2$ and $q = 1$. Because $\boldsymbol{\theta}_{n-1}$ does not lie on the line (plane) $\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{\theta} - y_n = 0$, whose direction is defined by $\boldsymbol{x}_n$, we know from geometry (and it is easily checked out) that its distance from this line is $s = \frac{|\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{\theta}_{n-1} - y_n|}{||\boldsymbol{x}_n||}$. Also, $\boldsymbol{\theta}_{n-1}$, lies on the negative side of the straight line, so that $\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{\theta}_{n-1} - y_n < 0$. Hence, taking into account the directions of the involved vectors, it turns out that

$$\Delta\boldsymbol{\theta} = \frac{y_n - \boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{\theta}_{n-1}}{||\boldsymbol{x}_n||}\frac{\boldsymbol{x}_n}{||\boldsymbol{x}_n||}.$$

Thus, for this specific case, the correction

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \Delta\boldsymbol{\theta}$$

coincides with the recursion of the APA.

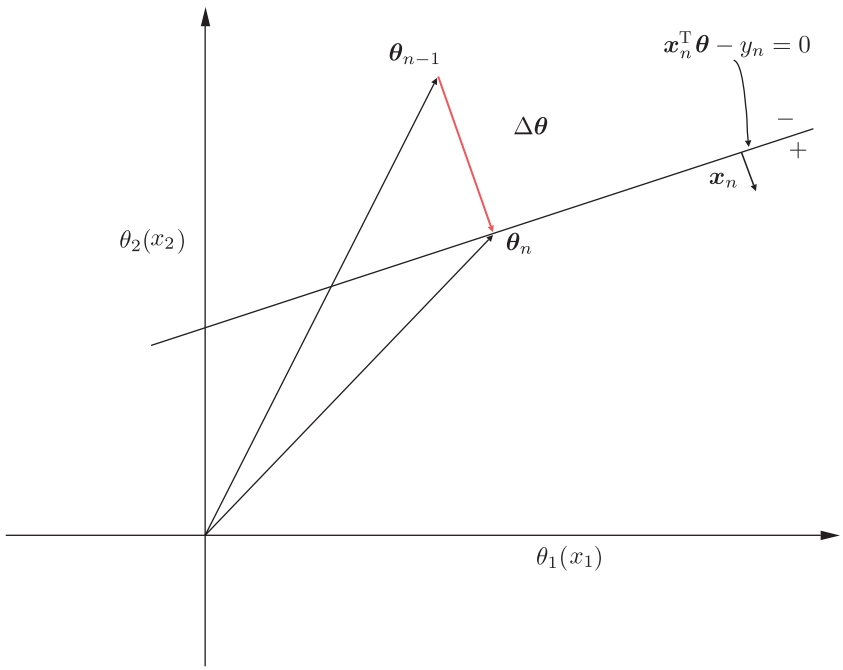


FIGURE 5.15

$\Delta\boldsymbol{\theta}$ is equal to the (signed) distance of $\boldsymbol{\theta}_{n-1}$ from the plane times the unit vector $\frac{\boldsymbol{x}_n}{||\boldsymbol{x}_n||}$.

5.6.1 THE NORMALIZED LMS

The normalized LMS is a special case of the APA and it corresponds to $q = 1$, see Figure 5.13. We treat it separately due to its popularity and it is summarized in Algorithm 5.3.

Algorithm 5.3 (The normalized LMS).

- Initialization
  - $\boldsymbol{\theta}_{-1} = \mathbf{0} \in \mathbb{R}^l$, or any other value.
  - Choose $0 < \mu < 2$, and δ a small value.
- **For** $n = 0, 1, 2, \ldots,$ **Do**
  - $e_n = y_n - \boldsymbol{\theta}_{n-1}^{\mathrm{T}}\boldsymbol{x}_n$
  - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \frac{\mu}{\delta + \boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{x}_n}\boldsymbol{x}_n e_n$
- **End For**

The complexity of the normalized LMS, is $3l$ MADs. Stability of the normalized LMS is guaranteed if

$$0 < \mu < 2.$$

One can look at the normalized LMS as an LMS whose step size is left to vary with the iterations, as represented by

$$\mu_n = \frac{\mu}{\delta + \boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{x}_n},$$

which turns out to have a beneficial effect on the convergence speed, compared to the LMS. More on the performance analysis of the normalized LMS, the interested reader can obtain from Refs. [15, 83, 90].

Remarks 5.3.

- To deal with sparse models, the so-called *proportionate* NLMS and related versions of the other algorithms have been derived [11, 35]. The idea is to use a separate step size for each one of the parameters. This gives the freedom to the coefficients, which correspond to small (or zero) values of the model, to adapt at a different rate than the rest, and this has a significant effect on the convergence performance of the algorithm. Such schemes can be considered as the ancestors of the more theoretically elegant sparsity-promoting online algorithms to be treated in Chapter 10.
- Another trend that has received attention more recently is to appropriately (convexly) combine the outputs of two (or more) learning structures. This has the effect of decreasing the sensitivity of the learning algorithms to choices of parameters such as the step-size, or to the dimensionality of the problem (size of the filter). The two (or more) algorithms run independently and the mixing parameters of the outputs are learned during the training. In general, this approach turns out to be more robust in the choice of the involved user-defined parameters [8, 81].
- Sometimes in bibliography, the user-defined parameter, δ, in the NLMS and the APA algorithm is suggested to be given a very small positive value. Often, the explanation for this option is that the use of δ is to avoid division by zero. However, in practice, there are cases, such as the echo cancellation task, where δ need to be set to quite large values (even larger than 1) to attain good performance. It is fairly recently that the importance of δ was emphasized and a formula for its proper tuning was proposed both for the case of NLMS and APA and their proportionate counterparts [12, 73]. There, it is indicated that without the proper setup of this parameter, the performance of these algorithms may be significantly affected, and they may not even converge.

5.7 THE COMPLEX-VALUED CASE

In Section 4.4, when

$$\mathrm{y}_n \in \mathbb{C} \quad \text{and} \quad \mathbf{x}_n \in \mathbb{C}^l,$$

the *widely linear* formulation of the estimation task was introduced to deal with complex-valued data that do not obey the circularity conditions. The output of a widely linear estimator is given by,

$$\hat{\mathrm{y}}_n = \boldsymbol{\varphi}^H \tilde{\mathbf{x}}_n,$$

where

$$\boldsymbol{\varphi} := \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{v} \end{bmatrix} \quad \text{and} \quad \tilde{\mathbf{x}}_n = \begin{bmatrix} \mathbf{x}_n \\ \mathbf{x}_n^* \end{bmatrix},$$

with $\boldsymbol{\theta}, \boldsymbol{v} \in \mathbb{C}^l$. The MSE cost function is

$$J(\boldsymbol{\varphi}) = \mathbb{E}\left[|\mathrm{y}_n - \boldsymbol{\varphi}^H \tilde{\mathbf{x}}_n|^2\right],$$

and following standard arguments analogous to Section 5.3.1, the minimum w.r.t. $\boldsymbol{\varphi}$ is given by the root of

$$\Sigma_{\tilde{x}}\boldsymbol{\varphi} - \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{q}^* \end{bmatrix} = \mathbf{0} \text{ or } \mathbb{E}\left[\tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^H\right]\boldsymbol{\varphi} = \mathbb{E}\left[\tilde{\mathbf{x}}_n \mathrm{y}_n^*\right].$$

The widely linear LMS

Employing the Robbins-Monro scheme and fixing the value of μ_n to be a constant, we obtain

$$\begin{aligned} \boldsymbol{\varphi}_n &= \boldsymbol{\varphi}_{n-1} + \mu\tilde{\boldsymbol{x}}_n e_n^*, \\ e_n &= y_n - \boldsymbol{\varphi}_{n-1}^H \tilde{\boldsymbol{x}}_n. \end{aligned}$$

Breaking $\boldsymbol{\varphi}_n$ and $\tilde{\boldsymbol{x}}_n$ into their components, the widely linear LMS results.

Algorithm 5.4 (The widely linear LMS).

- Initialize
  - $\boldsymbol{\theta}_{-1} = \mathbf{0},\ \boldsymbol{v}_{-1} = \mathbf{0}$
  - Choose μ.
- **For** $n = 0, 1, \ldots,$ **Do**

$$e_n = y_n - \boldsymbol{\theta}_{n-1}^H \boldsymbol{x}_n - \boldsymbol{v}_{n-1}^H \boldsymbol{x}_n^* \tag{5.65}$$

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu\boldsymbol{x}_n e_n^* \tag{5.66}$$

$$\boldsymbol{v}_n = \boldsymbol{v}_{n-1} + \mu\boldsymbol{x}_n^* e_n^* \tag{5.67}$$

- **End For**

Note that whatever has been said for the stability conditions concerning the LMS is applied here as well, if Σ_x, is replaced by $\Sigma_{\tilde{x}}$. For circularly symmetric variables, we set $\boldsymbol{v}_n = \mathbf{0}$ and the *complex linear LMS* results.

The widely linear APA

Let $\boldsymbol{\varphi}_n$ and $\tilde{\boldsymbol{x}}_n$ be defined as before. The widely linear APA results and it is given in Algorithm 5.5 (Problem 5.13).

Algorithm 5.5 (The widely linear APA).

- Initialize
  - $\boldsymbol{\varphi}_{-1} = \mathbf{0}$
  - Choose μ.
- **For** $n = 0, 1, 2, \ldots,$ **Do**

$$\begin{aligned} \boldsymbol{e}_n^* &= \boldsymbol{y}_n^* - \tilde{X}_n\boldsymbol{\varphi}_{n-1} \\ \boldsymbol{\varphi}_n &= \boldsymbol{\varphi}_{n-1} + \mu\tilde{X}_n^H(\delta I + \tilde{X}_n\tilde{X}_n^H)^{-1}\boldsymbol{e}_n^* \end{aligned}$$

- **End For**

Note that,

$$\tilde{X}_n = \begin{bmatrix} \tilde{\boldsymbol{x}}_n^H \\ \vdots \\ \tilde{\boldsymbol{x}}_{n-q+1}^H \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_n^H,\ \boldsymbol{x}_n^{\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_{n-q+1}^H,\ \boldsymbol{x}_{n-q+1}^{\mathrm{T}} \end{bmatrix},$$

and $\boldsymbol{\varphi}_n \in \mathbb{C}^{2l}$. For circular variables/processes, the complex linear APA results by setting

$$\tilde{X}_n = X_n = \begin{bmatrix} \boldsymbol{x}_n^H \\ \vdots \\ \boldsymbol{x}_{n-q+1}^H \end{bmatrix}$$

and

$$\boldsymbol{\varphi}_n = \boldsymbol{\theta}_n \in \mathbb{C}^l.$$

5.8 RELATIVES OF THE LMS

In addition to the three basic stochastic gradient descent schemes that were previously reviewed, a number of variants have been proposed over the years, in an effort either to improve performance or to reduce complexity. Some notable examples are described below.

The sign-error LMS

The update recursion for this algorithm becomes (e.g., [17, 30, 63])

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu \text{csgn}[e_n^*]\boldsymbol{x}_n,$$

where the complex sign of a complex number, $z = x + jy$, is defined as

$$\text{csgn}(z) = \text{sgn}(x) + j\,\text{sgn}(y).$$

If in addition, μ is chosen to be a power of two, then the recursion becomes *multiplication free*, and l multiplications are only needed for the computation of the error. It turns out that the algorithm minimizes, in the stochastic approximation sense, the following cost function,

$$J(\boldsymbol{\theta}) = \mathbb{E}\left[|\text{y} - \boldsymbol{\theta}^H \mathbf{x}|\right],$$

and stability is guaranteed for sufficiently small values of μ [83].

The least-mean-fourth (LMF) algorithm

The scheme minimizes the following cost function,

$$J(\boldsymbol{\theta}) = \mathbb{E}\left[|\text{y} - \boldsymbol{\theta}^H \boldsymbol{x}|^4\right],$$

and the corresponding update recursion is given by,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu |e_n|^2 \boldsymbol{x}_n e_n^*.$$

It has been shown [101] that minimization of the fourth power of the error may lead to an adaptive scheme with better compromise between convergence rate and excess mean-square error than the LMS, if the noise source is sub-Gaussian. In a sub-Gaussian distribution, the tails of the pdf graph are decaying at a faster rate compared to the Gaussian one. The LMF could be seen as a version of the LMS with time-varying step-size, equal to $\mu|e_n|^2$. Hence, when the error is large, the step size increases, which helps the LMF to converge faster. On the other hand, when the error is small, the equivalent step size is reduced, leading the excess mean-square error to lower values. This idea turns out to work well when the noise is sub-Gaussian. However, the LMF tends to become unstable in the presence of outliers. This is also

understood, because for very large error values the equivalent step size becomes very large leading to instability. This is, for example, the situation when the noise follows a distribution with high-tails; that is, when the corresponding pdf curve decays at slower rates compared to the Gaussian, such as in the case of super-Gaussian distributions. Results concerning the analysis of the LMF can be found in Refs. [69, 70, 83]. Robustness and stability of the algorithm are guaranteed for sufficiently small values of μ [83].

Transform-domain LMS

We have already commented that the convergence speed of the LMS heavily depends on the condition number $\left(\frac{\lambda_{\max}}{\lambda_{\min}}\right)$ of the covariance matrix; this will soon be demonstrated in the examples below.

Transform-domain techniques exploit the decorrelation properties of certain transforms, such as the DFT and DCT, in order to decorrelate the input variables. When the input comprises a stochastic process, we say that such transforms "*prewhiten*" the input process. Moreover, in the case where the involved variables are part of a random process, one can exploit the time-shifting property and employ *block processing techniques*; by processing a block of data samples per time instant, one can exploit efficient implementations of certain transforms, such as the fast Fourier transform (FFT), to reduce the overall complexity. Such schemes are appropriate for applications where long filters are involved. For example, in some applications, such as echo cancellation, filter orders of a few hundred taps are commonly encountered [10, 39, 53, 66, 68].

Let T be a unitary transform in the complex domain, represented by $TT^H = T^H T = I$. Define

$$\hat{\boldsymbol{x}}_n = T^H \boldsymbol{x}_n, \tag{5.68}$$

and apply the transform matrix T^H on both sides of the complex LMS recursion in (5.66) to obtain

$$\hat{\boldsymbol{\theta}}_n = \hat{\boldsymbol{\theta}}_{n-1} + \mu \hat{\boldsymbol{x}}_n e_n^*, \tag{5.69}$$

where

$$\hat{\boldsymbol{\theta}}_n = T^H \boldsymbol{\theta}_n. \tag{5.70}$$

Note that

$$e_n = y_n - \boldsymbol{\theta}_{n-1}^H \boldsymbol{x}_n = y_n - \boldsymbol{\theta}_{n-1}^H T T^H \boldsymbol{x}_n$$

$$= y_n - \hat{\boldsymbol{\theta}}_{n-1}^H \hat{\boldsymbol{x}}_n.$$

Hence, the error term is not affected. Note that until now, we have not achieved much. Indeed,

$$\Sigma_{\hat{x}} = T^H \Sigma_x T,$$

is a similarity transformation that does not affect the condition number of the matrix, Σ_x; it is known from linear algebra that both matrices share the same eigenvalues (Problem 5.14). Let us now choose $T = Q$, where Q is the unitary matrix comprising the orthonormal eigenvectors of Σ_x. For this case, $\Sigma_{\hat{x}} = \Lambda$, which is the diagonal matrix with entries the eigenvalues of Σ_x. In the sequel, we modify the transform-domain LMS in (5.69), so that to use a different step size per component, according to the following scenario,

$$\hat{\boldsymbol{\theta}}_n = \hat{\boldsymbol{\theta}}_{n-1} + \mu \Lambda^{-1} \hat{\boldsymbol{x}}_n e_n^*, \tag{5.71}$$

or

$$\bar{\boldsymbol{\theta}}_n = \bar{\boldsymbol{\theta}}_{n-1} + \mu \bar{\boldsymbol{x}}_n e_n^*, \tag{5.72}$$

where

$$\bar{\boldsymbol{\theta}}_n := \Lambda^{1/2} \hat{\boldsymbol{\theta}}_n,$$

and

$$\bar{\boldsymbol{x}}_n := \Lambda^{-1/2}\hat{\boldsymbol{x}}_n.$$

Note that the error is not affected, because $\bar{\boldsymbol{\theta}}_n^H \bar{\boldsymbol{x}}_n = \hat{\boldsymbol{\theta}}_n^H \hat{\boldsymbol{x}}_n$. We have now achieved our original goal, since

$$\Sigma_{\bar{x}} = \Lambda^{-1/2}\Sigma_{\hat{x}}\Lambda^{-1/2} = \Lambda^{-1/2}\Lambda\Lambda^{-1/2} = I.$$

That is, the condition number of $\Sigma_{\bar{x}}$ is equal to 1. In practice, this technique is difficult to apply, due to the complexity associated with the eigendecomposition task; more importantly, Σ_x must be known, which in adaptive implementations is not the case. In practice, we resort to unitary transforms, T, that approximately whiten the input, such as the DFT and DCT. Then, (5.71) is replaced by

$$\hat{\boldsymbol{\theta}}_n = \hat{\boldsymbol{\theta}}_{n-1} + \mu D^{-1}\hat{\boldsymbol{x}}_n e_n^*,$$

where D is the diagonal matrix with entries the variances of the respective components of $\hat{\boldsymbol{x}}_n$, or

$$[D]_{ii} = \mathbb{E}\left[(\hat{\mathrm{x}}_n(i))^2\right] = \sigma_i^2, \;\; i = 1, 2, \ldots, l,$$

where $\hat{\mathrm{x}}_n(i)$ is the ith entry of $\hat{\mathbf{x}}_n$, which has been assumed to be a zero mean vector; this is justified from the fact that if $\Sigma_{\hat{x}}$ is truly diagonal, and equal to Λ, the eigenvalues correspond to the variances of the respective elements. The entries σ_i^2 are estimated in a time-adaptive fashion, and the scheme given in Algorithm 5.6 results.

Algorithm 5.6 (The transform-domain LMS).

- Initialization
  - $\hat{\boldsymbol{\theta}}_{-1} = \mathbf{0}$; or any other value.
  - $\sigma_{-1}^2(i) = \delta, \;\; i = 1, 2, \ldots, l$; δ a small value.
  - Choose μ, and $0 \ll \beta < 1$.
- **For** $n = 0, 1, 2, \ldots,$ **Do**
  - $\hat{\boldsymbol{x}}_n = T^H \boldsymbol{x}_n$
  - $e_n = y_n - \hat{\boldsymbol{\theta}}_{n-1}^H \hat{\boldsymbol{x}}_n$
  - $\hat{\boldsymbol{\theta}}_n = \hat{\boldsymbol{\theta}}_{n-1} + \mu D^{-1}\hat{\boldsymbol{x}}_n e_n^*$
  - **For** $i = 1, 2, \ldots, l,$ **Do**
    - $\sigma_i^2(n) = \beta\sigma_i^2(n-1) + (1-\beta)|\hat{x}_n(i)|^2$
  - **End For**
  - $D = \mathrm{diag}\{\sigma_i^2(n)\}$
- **End For**

Subband adaptive filters is a related family of algorithms where whitening is achieved via block data processing and the use of a multirate filter band. Such schemes were first proposed in Refs. [41, 53, 54] and have successfully been used in applications such as in echo cancellation [45, 53].

Another approach that has been adopted to improve the convergence rate, for the case of system identification applications, is to select the input excitation signal to be of a specific type. For example, in Ref. [5], it is pointed out that the excitation signal that optimizes the convergence speed of the NLMS algorithm is a deterministic perfect periodic sequence (PPSEQ) with period equal to the impulse response of the system. Such sequences have been used in Refs. [24, 25] to derive versions of the LMS,

which not only converge fast but they are also of very low computational complexity, requiring only one multiplication, one addition, and one subtraction per update.

5.9 SIMULATION EXAMPLES

Example 5.3. The goal of this example is to demonstrate the sensitivity of the convergence rate of the LMS on the eigenvalues spread of the input covariance matrix. To this end, two experiments were conducted, in the context of regression/system identification task. Data were generated according to our familiar model

$$y_n = \boldsymbol{\theta}_o^T \boldsymbol{x}_n + \eta_n.$$

The (unknown) parameters, $\boldsymbol{\theta}_o \in \mathbb{R}^{10}$, were randomly chosen from a $\mathcal{N}(0, 1)$ and then frozen. In the first experiment, the input vectors were formed by a white noise sequence with samples i.i.d. drawn from a $\mathcal{N}(0, 1)$. Thus, the input covariance matrix was diagonal with all the elements being equal to the corresponding noise variance (Section 2.4.3). The noise samples, η_n, were also i.i.d. drawn from a Gaussian with zero mean and variance $\sigma^2 = 0.01$. In the second experiment, the input vectors were formed by an AR(1) process with coefficient equal to $a_1 = 0.85$ and the corresponding white noise excitation was of variance equal to 1 (Section 2.4.4). Thus, the input covariance matrix is no more diagonal and the eigenvalues are not equal. The LMS was run on both cases with the same step-size $\mu = 0.01$. Figure 5.16 summarizes the results. The vertical axis (denoted as MSE) shows the squared

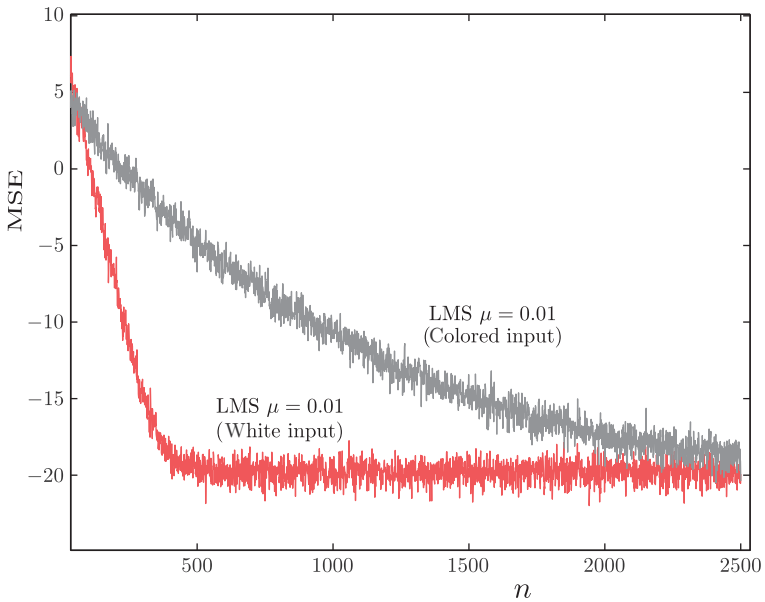


FIGURE 5.16

Observe that for the same step-size, the convergence of the LMS is faster when the input is white. The two curves are the result of averaging 100 independent experimental realizations.

error, e_n^2, and the horizontal axis the time instants (iterations) n. Note that both curves level out at the same error floor. However, the convergence rate for the case of the white noise input is significantly higher. The curves shown in the figure are the result of averaging 100 independent experimental realizations.

It must be emphasized that, when comparing convergence performance of different algorithms, either all algorithms should converge to the same error floor and compare the respective convergence rates, or all algorithms should have the same convergence rate and compare respective error floors.

Example 5.4. In this example, the dependence of the LMS on the choice of the step-size is demonstrated. The unknown parameters, $\boldsymbol{\theta}_o \in \mathbb{R}^{10}$, as well as the data were exactly the same with the white noise case of Example 5.3.

The LMS was run using the generated samples, with two different step-sizes, namely, $\mu = 0.01$ and $\mu = 0.075$. The obtained averaged (over 100 realizations) curves are shown in Figure 5.17. Observe that the larger the step-size, for the same set of observation samples, the faster the convergence becomes albeit at the expense of a higher error floor (misadjustment), in accordance to what has been discussed in Section 5.5.1.

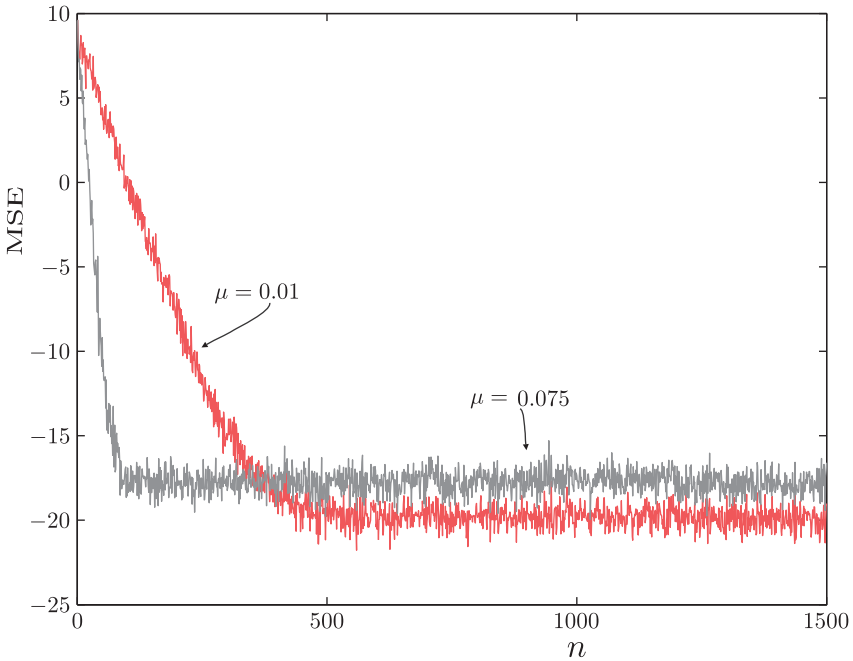


FIGURE 5.17

For the same input, the larger the step-size for the LMS the faster the convergence becomes, albeit at the expense of higher error floor.

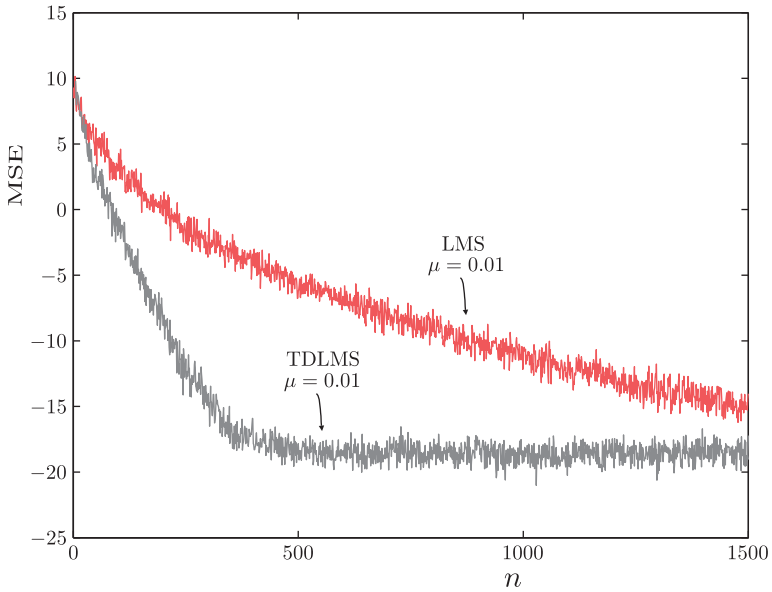


FIGURE 5.18

Observe that for the same step-size, the convergence of the transform-domain LMS is significantly faster compared to the LMS, for similar error floors. The higher the eigenvalues spread of the input covariance matrix is, the more the obtained performance improvement becomes.

Example 5.5 (LMS versus transform-domain LMS)**.** In this example, the stage of the experimental setup is exactly the same as that considered in Example 5.3 for the AR(1) case. The goal is to compare the LMS and the transform-domain LMS. Figure 5.18 shows the obtained averaged error curves. The step-size was the same as the one used in Example 5.3: $\mu = 0.01$; hence the curve for the LMS is the same as the corresponding one appearing in Figure 5.16. Observe the significantly faster convergence achieved by the transform-domain LMS, due to its (approximate) whitening effect on the input.

Example 5.6. The experimental setup is similar to that of Example 5.4, with the only exception that the unknown parameter vector was of higher dimension, $\boldsymbol{\theta}_o \in \mathbb{R}^{60}$, so that the differences in the performance of the algorithms is more clear. The goal is to compare the LMS, the normalized LMS (NLMS) and the affine projection algorithm (APA). The step size of the LMS was chosen equal to $\mu = 0.025$ and for the NLMS $\mu = 0.35$ and $\delta = 0.001$, so that both algorithms have similar convergence rates. The step size for the APA was chosen equal to $\mu = 0.1$, so that $q = 10$ will settle at the same error floor as that of the NLMS. For the APA, we also chose $\delta = 0.001$. The results are shown in Figure 5.19.

Observe the lower error floor, for the same convergence rate, obtained by the NLMS compared to the LMS and the improved performance obtained by the APA for $q = 10$. Increasing the number of past data samples (re)used in APA to $q = 30$, we can see the improved convergence rate that is obtained, although at the expense of higher error floor, as predicted by the theoretical results reported in Section 5.6.

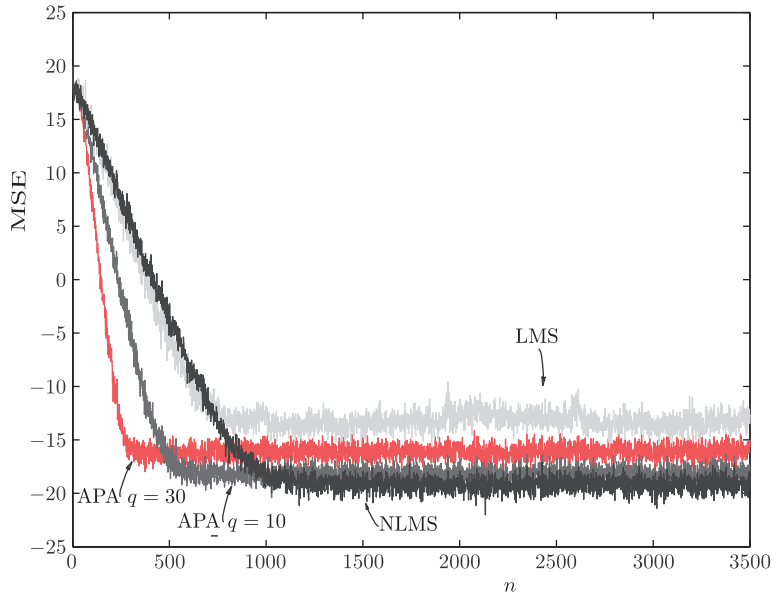


FIGURE 5.19

For the same step-size, the NLMS converges at the same rate to a lower error floor compared to the LMS. For the APA, increasing q, improves the convergence, at the expense of higher error floors.

5.10 ADAPTIVE DECISION FEEDBACK EQUALIZATION

The task of channel equalization was introduced in Figure 4.12. The input to the equalizer is a stochastic process (random signal), which, according to the notational convention introduced in Section 2.4, will be denoted as u_n. Note that upon receiving the noisy and distorted by the (communications) channel sample, u_n, one has to obtain an estimate of the originally transmitted information sequence, s_n, delayed by L time lags, which accounts for the various delays imposed by the overall transmission system involved. Thus, at time n, the equalizer decides for $\hat{s}_{n-L+1}$. Ideally, if one knew the true values of the initially transmitted information sequence up to and including time instant $n-L$, represented by $s_{n-L}, s_{n-L-1}, s_{n-L-2}\ldots$, it could only be beneficial to use this information, together with the received sequence, u_n, to recover an estimate of $\hat{s}_{n-L+1}$. This idea is explored in the *decision feedback equalizer* (DEE). The equalizer's output, for the complex-valued data case, is now written as

$$\begin{aligned}\hat{d}_n &= \sum_{i=0}^{L-1} w_i^{f*} u_{n-i} + \sum_{i=0}^{l-1} w_i^{b*} s_{n-L-i} \\ &= \boldsymbol{w}^{\mathrm{H}} \boldsymbol{u}_{e,n}\end{aligned} \tag{5.73}$$

where

$$\boldsymbol{w} := \begin{bmatrix} \boldsymbol{w}^f \\ \boldsymbol{w}^b \end{bmatrix} \in \mathbb{C}^{L+l}, \quad \boldsymbol{u}_{e,n} := \begin{bmatrix} \boldsymbol{u}_n \\ \boldsymbol{s}_n \end{bmatrix} \in \mathbb{C}^{L+l},$$

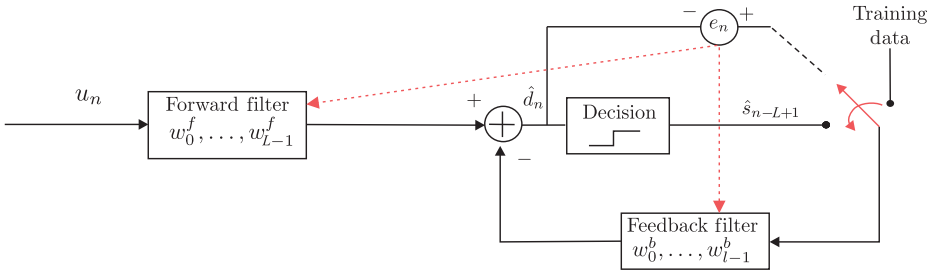


FIGURE 5.20

The forward part of the DFE acts on the received samples, while the backward acts on the training data/decisions, depending on the mode of operation.

and $\boldsymbol{s}_n := [s_{n-L}, \ldots, s_{n-L-l+1}]^T$. The desired response at time n is

$$d_n = s_{n-L+1}.$$

In practice, after the initial training period, the information samples, s_{n-L-i} are replaced by their computed estimates, $\hat{s}_{n-L-i}$, $i = 0, 1, \ldots, l-1$, which are available from decisions taken in previous time instants. It is said that the equalizer operates in the *decision directed* mode. The basic DFE structure is shown in Figure 5.20. Note that during the training period, the parameter vector, $\boldsymbol{w}$, is trained so that to minimize the power of the error,

$$e_n = d_n - \hat{d}_n = s_{n-L+1} - \hat{d}_n.$$

Once all the available training samples have been used, training carries on using the estimates $\hat{s}_{n-L+1}$. For example, for a binary information sequence $s_n \in \{1, -1\}$, the decision concerning the estimate at time n is obtained by passing $\hat{d}_n$ through a threshold device and $\hat{s}_{n-L+1}$ is obtained. Note that DFE is one of the early examples of *semisupervised* learning, where training data is not enough, and then the estimates are used for training [94]. In this way, assuming that at the end of the training phase, $\hat{s}_{n-L+1} = s_{n-L+1}$, and also that time variations are slow, so that to guarantee that $\hat{d}_n \simeq d_n$, then we expect, with good enough probability, that $\hat{s}_{n-L+1}$ will remain equal to s_{n-L+1}, so that the equalizer can track the changes. More on DFEs and their error performance can be obtained from Ref. [77].

Any of the adaptive schemes treated so far can be used in a DFE scenario by replacing in the input vector, $\boldsymbol{u}_e$, the term $\boldsymbol{s}_n$ by $\hat{\boldsymbol{s}}_n$, when operating in the decision directed mode. Note that adaptive algorithms in the context of the equalization task were employed first in Refs. [44, 78]. A version of the DFE, operating in the frequency domain, has been proposed for the first time in [14].

Thus the LMS recursion for the linear DFE, in its complex-valued formulation, becomes,

$$\begin{aligned}
\hat{d}_n &= \boldsymbol{w}_{n-1}^H \boldsymbol{u}_{e,n} \\
d_n &= s_{n-L+1}; \quad \text{in the training mode, or} \\
d_n &= T\left[\hat{d}_n\right]; \quad \text{in the decision directed mode,} \\
e_n &= d_n - \hat{d}_n \\
\boldsymbol{w}_n &= \boldsymbol{w}_{n-1} + \mu \boldsymbol{u}_{e,n} e_n^*,
\end{aligned}$$

where $T[\cdot]$ denotes the thresholding operation.

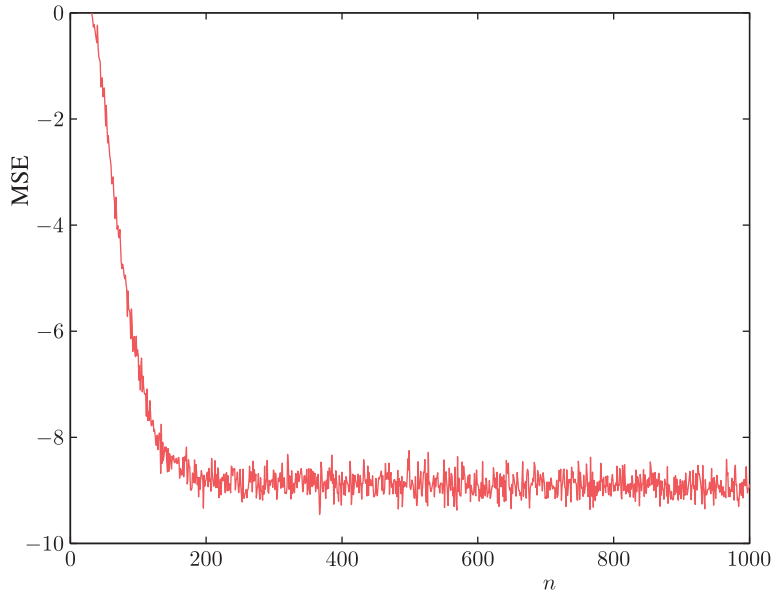


FIGURE 5.21

The MSE in dBs for the DFE of Example 5.7. After the time instant $n = 250$, the LMS is trained with the decisions $\hat{s}_{n-L+1}$.

Example 5.7. Let us consider a communication system where the input information sequence comprises a stream of randomly generated symbols $s_n = \pm 1$, with equal probability. This sequence is sent to a channel with impulse response,

$$\boldsymbol{h} = [0.04, -0.05, 0.07, -0.21, 0.72, 0.36, 0.21, 0.03, 0.07]^T.$$

The output of the channel is contaminated by white Gaussian noise at the SNR $= 11$dB level. A DFE is used with the length of the feedforward section being equal to $L = 21$ and the length of the feedback part equal to $l = 10$. The DFE was trained with 250 symbols; then, it was switched on to the decision directed mode and it was run for 10,000 iterations. At each iteration, the decision $(\text{sgn}(\hat{d}_n))$ was compared with the true transmitted symbol s_{n-L+1}. The error rate (total number of errors over the corresponding number of transmitted symbols), was approximately 1%. Figure 5.21 shows the averaged MSE curve as a function of the number of iterations. For the LMS, we used $\mu = 0.025$.

5.11 THE LINEARLY CONSTRAINED LMS

The task of linearly constrained MSE estimation was introduced in Section 4.9.2. Here, we turn our attention into its online stochastic gradient counterpart. The discussion will be carried out within the notational convention used for linear filtering involving processes, since a typical application is that of beamforming; this is also the reason that we will involve the complex-valued formulation. However, everything to be said carries on to the more general linear estimation task.

In Section 4.9.2, the goal was to minimize, subject to a set of constraints, either the noise variance or the output of the filter (beamformer), leading to the costs $\boldsymbol{w}^{\mathrm{H}}\Sigma_{\eta}\boldsymbol{w}$ or $\boldsymbol{w}^{\mathrm{H}}\Sigma_{u}\boldsymbol{w}$, respectively. In a more general setting, we will require the output to be "close" to a desired response random signal d_n. For the beamforming setting, this corresponds to a desired (training) signal; that is, besides the desired direction, which is provided via the constraints, we are also given a desired signal sequence. Moreover, we will assume that our solution has to satisfy more than one constraints. The task now becomes,

$$\begin{aligned} \boldsymbol{w}_* &= \arg\min_{\boldsymbol{w}} \mathbb{E}\left[\left|\mathrm{d}_n - \boldsymbol{w}^{\mathrm{H}}\mathbf{u}_n\right|^2\right] \\ \text{s.t.} \qquad & \boldsymbol{w}^{\mathrm{H}}\boldsymbol{c}_i = g_i, \quad i = 1, 2, \ldots, m. \end{aligned} \tag{5.74}$$

for some $g_i \in \mathbb{R}$. The corresponding Lagrangian is,

$$\begin{aligned} L(\boldsymbol{w}) = \sigma_d^2 &+ \boldsymbol{w}^{\mathrm{H}}\,\mathbb{E}[\mathbf{u}_n\mathbf{u}_n^{\mathrm{H}}]\boldsymbol{w} - \boldsymbol{w}^{\mathrm{H}}\,\mathbb{E}[\mathbf{u}_n\mathrm{d}_n^*] - \mathbb{E}[\mathbf{u}_n^H\mathrm{d}_n]\boldsymbol{w} \\ &+ \sum_{i=1}^{m}\lambda_i\left(\boldsymbol{w}^{\mathrm{H}}\boldsymbol{c}_i - g_i\right), \end{aligned}$$

where $\lambda_i,\ i = 1, 2, \ldots, m$, are the corresponding Lagrange multipliers. Taking the gradient w.r.t. $\boldsymbol{w}^*$ and considering $\boldsymbol{w}$ to be a constant, we get

$$\nabla_{w^*}L(\boldsymbol{w}) = \mathbb{E}[\mathbf{u}_n\mathbf{u}_n^{\mathrm{H}}]\boldsymbol{w} - \mathbb{E}[\mathbf{u}_n\mathrm{d}_n^*] + \sum_{i=1}^{m}\lambda_i\boldsymbol{c}_i.$$

Applying the Robbins-Monro scheme to find the root, we get,

$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} + \mu\boldsymbol{u}_n e_n^* - \mu C\boldsymbol{\lambda}_n, \tag{5.75}$$

where we have used a constant step-size, μ, and

$$\hat{d}_n = \boldsymbol{w}_{n-1}^H\boldsymbol{u}_n,$$

and

$$e_n = d_n - \hat{d}_n,$$

and have allowed $\boldsymbol{\lambda}$ to be time-dependent. C is defined as the matrix having as columns the vectors $\boldsymbol{c}_i,\ i = 1, 2, \ldots, m$.

Plugging (5.75) into the constraints in (5.74), which can be compactly written as $C^{\mathrm{H}}\boldsymbol{w} = \boldsymbol{g}$ (recall, $g_i \in \mathbb{R}$), we readily obtain

$$-\mu\boldsymbol{\lambda}_n = (C^{\mathrm{H}}C)^{-1}\boldsymbol{g} - (C^{\mathrm{H}}C)^{-1}C^{\mathrm{H}}\left(\boldsymbol{w}_{n-1} + \mu\boldsymbol{u}_n e_n^*\right),$$

and the update recursion becomes

$$\boldsymbol{w}_n = \left(I - C(C^{\mathrm{H}}C)^{-1}C^{\mathrm{H}}\right)\left(\boldsymbol{w}_{n-1} + \mu\boldsymbol{u}_n e_n^*\right) + C\left(C^{\mathrm{H}}C\right)^{-1}\boldsymbol{g}.$$

Note that $\left(I - C(C^{\mathrm{H}}C)^{-1}C^{\mathrm{H}}\right)$ is the orthogonal projection matrix on the intersection (affine set) of the hyperplanes defined by the constraints (recall that, $C(C^{\mathrm{H}}C)^{-1}C^{\mathrm{H}}$ is the respective projection on the subspace spanned by $\boldsymbol{c}_i,\ i = 1, 2, \ldots, m$). In case the goal is to minimize the output, then one has to set in e_n the desired response $d_n = 0$.

The constrained LMS was first treated in [40]. Besides the previously used constraints, other constraints can also be used, for example, for the constrained normalized LMS, one additionally demands

$$\boldsymbol{w}_n^{\mathrm{H}}\boldsymbol{u}_n = d_n$$

see (e.g., [6]).

5.12 TRACKING PERFORMANCE OF THE LMS IN NONSTATIONARY ENVIRONMENTS

We have already considered the convergence behavior of the LMS and made related comments for the other algorithms that have been discussed. As it has been stated before, convergence is a *transient* phenomenon; that is, it concerns the period from the initial kick-off point, to the steady-state. The steady-state was also discussed for stationary environments; that is, environments in which the unknown parameter vector as well as the underlying statistics of the involved random variables/processes remain unchanged.

We now turn our focus to cases where the true (yet unknown) parameter vector/system undergoes changes. Thus, this affects the output observations and consequently their statistics. Note that the statistics of the input can also change. However, we are not going to consider such cases, as the analysis can become quite involved. Our goal is to study the *tracking* performance of the LMS; that is, the ability of the algorithm to track changes of the unknown parameter vector. Note that tracking is a *steady-state* phenomenon. In other words, we assume that enough time has elapsed so that the influence of the initial conditions has been forgotten and has no effect, any more, on the algorithm. Tracking agility and convergence speed are two *different* properties of an algorithm. An algorithm may converge fast, but it *may not* necessarily have a good tracking performance and vice versa. We will see such cases later on.

The setting of our discussion will be similar to that of Section 5.5.1. In conformity with the discussion there, we consider the real-data case; similar results hold true for the complex-valued linear estimation scenario. However, in contrast to the adopted model in (5.37), a *time-varying* model is adopted here using the following assumptions.

Assumption 1. The output observations are generated according to the model,

$$\mathrm{y}_n = \boldsymbol{\theta}_{o,n-1}^T\mathbf{x} + \eta, \tag{5.76}$$

which is in line with the prediction model used in LMS, at time n. That is, the unknown set of parameters is a time-varying one. The statistical properties of the input vector, $\mathbf{x}$, as well as the noise variable η are assumed to be time-independent, and this is the reason we have not used the time index; equivalently, in the case where the input is a random process, u_n, it is assumed to be *stationary*. Moreover, the input variables are assumed to be independent of the zero mean noise variable, η. Furthermore, successive samples of η are i.i.d. (white noise sequence) of variance σ_η^2. So far, we have not gone much beyond the Assumption 1, stated in Section 5.5.1.

Assumption 2. The time varying model follows a random walk variation, represented by

$$\boldsymbol{\theta}_{o,n} = \boldsymbol{\theta}_{o,n-1} + \boldsymbol{\omega}. \tag{5.77}$$

The random vector $\boldsymbol{\omega}$ is assumed to be zero mean with covariance matrix

$$\mathbb{E}\left[\boldsymbol{\omega}\boldsymbol{\omega}^{\mathrm{T}}\right] = \Sigma_\omega.$$

Note that the variance of a random walk grows unbounded with time; this is readily shown by applying (5.77) recursively.

A variant of this model would be more sensible to use,

$$\theta_{o,n} = a\theta_{o,n-1} + \omega,$$

with $|a| < 1$ [106]. However, the analysis gets more involved, so we will stick with the model in (5.77). After all, our goal here is to highlight and have a first touch on the notion of tracking and get an idea of its effects on the misadjustment in steady-state.

Assumption 3. As in Section 5.5.1, we assume that $\mathbf{c}_{n-1} := \theta_{n-1} - \theta_{o,n-1}$ is independent of $\mathbf{x}$ and η. This time, we will also assume independence of $\mathbf{c}_n$ and ω.

Recall from (5.47) that, the excess mean-square error, at time n, is given by

$$J_{\text{exc},n} = \text{trace}\{\Sigma_x \Sigma_{c,n-1}\}.$$

Thus, our goal now is to compute $\Sigma_{c,n-1}$ for the time-varying model case. It is straightforward to see that the counterpart of (5.34) now becomes,

$$\mathbf{c}_n = \left(I - \mu\mathbf{x}\mathbf{x}^{\mathrm{T}}\right)\mathbf{c}_{n-1} + \mu\mathbf{x}\eta - \omega. \tag{5.78}$$

Adopting the previously stated three assumptions, as well as the *Gaussian* assumption for $\mathbf{x}$, and following exactly the same steps used for (5.40), we end up with,

$$\begin{aligned}\Sigma_{c,n} = \Sigma_{c,n-1} - \mu\Sigma_x\Sigma_{c,n-1} - \mu\Sigma_{c,n-1}\Sigma_x + 2\mu^2\Sigma_x\Sigma_{c,n-1}\Sigma_x \\ +\mu^2\Sigma_x\text{trace}\{\Sigma_x\Sigma_{c,n-1}\} + \mu^2\sigma_\eta^2\Sigma_x + \Sigma_\omega.\end{aligned} \tag{5.79}$$

Note that if complex data are involved, the only difference is that the fourth term on the right-hand side is not multiplied by 2 (Problem 5.15). This equation governs the propagation of $\Sigma_{c,n}$, which in turn can provide the excess MSE error.

A more convenient form results for small values of μ, where the fourth and fifth terms on the right-hand side can be neglected, being small w.r.t. $\mu\Sigma_x\Sigma_{c,n-1}$. Moreover, at the steady-state, $\Sigma_{c,n} = \Sigma_{c,n-1} := \Sigma_c$, and taking the trace on both sides, we end up with (recall $\text{trace}\{A+B\} = \text{trace}\{A\} + \text{trace}\{B\}$ and $\text{trace}\{AB\} = \text{trace}\{BA\}$),

$$\boxed{J_{\text{exc}} = \text{trace}\{\Sigma_x\Sigma_c\} = \frac{1}{2}\left(\mu\sigma_\eta^2\text{trace}\{\Sigma_x\} + \frac{1}{\mu}\text{trace}\{\Sigma_\omega\}\right).} \tag{5.80}$$

Note that this is exactly the same approximation as the one resulting from the more sound theory of energy conservation [83].

Compare (5.80) with (5.52); in the current setting, there is another term associated with the noise, which drifts the model around its mean. Thus, the excess MSE is contributed (a) by the inability of the LMS to obtain the optimum value exactly, and (b) there is an extra term measuring its "inertia" to track the changes of the model fast enough. This is the most important outcome of the current discussion. In time-varying environments, the misadjustment increases. Moreover, looking at (5.80) and at the effect of μ, it is observed that small values of μ have a beneficial effect on the first term, but they increase the contribution of the second one. The opposite is true for relatively big values of μ. This is natural. Small step-sizes give the algorithm the chance to learn better under stationary environments, but the

algorithm cannot track the changes fast enough. Thus, the choice of μ should be the outcome of a trade-off. Minimizing the excess error in (5.80), it is easily shown to result in

$$\mu_{\text{opt}} = \sqrt{\frac{\text{trace}\{\Sigma_\omega\}}{\sigma_\eta^2 \text{trace}\{\Sigma_x\}}}.$$

Note, however, that such choices for μ are of theoretical importance only. In practice, the time variation of the system can hardly correspond to that of the adopted model; the latter, due to the complexity of the analysis, was chosen to be a simple one in an effort to simplify the mathematical manipulations. Moreover, for the sake of simplifying the analysis, a number of assumptions were adopted. In practice, μ is chosen more according to the user's practical experience, after experimentation, than based on the theory. The theory, however, has pointed out the trade-off between the speed of convergence and that of tracking.

More mathematically rigorous analysis of the performance of online/adaptive schemes in non-stationary environments can be obtained from Refs. [16, 38, 47, 61, 70, 83]. Simulation results, demonstrating the tracking performance of the LMS compared to other algorithms are given in Example 6.3.

5.13 DISTRIBUTED LEARNING: THE DISTRIBUTED LMS

The focus of our attention is now turned toward a problem that has gained increasing importance over the last decade or so. There is a growing number of applications where data are received/reside in different sensors/databases, which are spatially distributed. However, all this spatially distributed information has to be exploited towards achieving a common goal; that is, to perform a *common estimation/inference* task. We refer to such tasks as *distributed* or *decentralized* learning. At the heart of this problem lies the concept of *cooperation*, which is another name for the process of exchanging learning experience/information in order to reach a common goal/decision. Human societies have survived because of cooperation (and have disappeared due to lack of cooperation).

Distributed learning is common in many biological systems, where no individual/agent is in charge, yet the group exhibits a high degree of intelligence (we humans refer to it as instinct). Look at the way birds fly in formation and bees swarm in a new hive.

Besides sociology and biology, science and engineering have used the concept of distributed learning; *wireless sensor networks* (WSNs) are a typical example. WSNs were originally suggested as spatially distributed autonomous sensors to monitor physical and environmental conditions, such as pressure, temperature, sound and to cooperatively pass their data to a central unit. Although WSNs were originally motivated for military applications, today are targeted at a diverse number of applications, such as traffic control, homeland security and surveillance, health care and environmental modeling. Each sensor node is equipped with an onboard processor, in order to perform locally some simple processing and transmit the required and *partially* processed data. Sensors/nodes are characterized by low processing, memory, and communication capabilities due to low energy and bandwidth constraints [1, 104].

Other typical examples of distributed learning applications are the modeling and study of the way individuals are linked in social networks, modeling pathways defined over complex power grids, cognitive radio systems, and pattern recognition; the common characteristic of all these applications is

that data are partially processed in each individual node/agent, and the processed information is passed over to the network under a *certain protocol*.

The obvious question that the unfamiliar reader may ask is why not use only one node/agent and the locally residing information to perform the inference task. The answer, of course, is that we can come with better estimates/results by exploiting the available data/information across the whole network. This brings us to the notion of *consensus*.

According to the American Heritage dictionary, "consensus" is defined as "an opinion or position reached by a group as a whole"; that is, consensus is the process that guarantees an "accepted agreement" within the group. The term "accepted agreement" is not uniquely defined. In some cases, this may refer to a *unanimous* decision, in other cases it refers to a *majority* rule. In some cases, all the opinions of all agents are equally weighted whereas in others, different weights are imposed, based on some relative significance measures. However, in all cases, the essence of any consensus-based process is the trust that a "better" decision is reached when compared to the process of each agent/person acting individually.

In this section, we focus on the task of *parameter estimation*. Each individual agent has access to partial information via a "local" acquisition process of data. Although each agent has access to a different set of data, they all share a common goal; that is, to estimate the *same* unknown set of parameters. This task will be achieved in a collaborative manner. However, different cooperation scenarios can be adopted.

5.13.1 COOPERATION STRATEGIES

In distributed learning, each individual agent is represented as a node in a graph. Edges between nodes indicate that the respective agents can exchange information. Undirected edges indicate that information can be exchanged in both directions, while directed edges indicate the allowed direction of information flow.[7]

Centralized networks

Under this scenario of cooperation, nodes communicate their measurements to a central *fusion* unit for processing. The obtained estimate can be communicated back to each one of the nodes. Figure 5.22 illustrates the topology. In Figure 5.22a all nodes are connected directly to the fusion center, indicated by a square. In Figure 5.22b, some of the nodes can be linked directly to the fusion center, while others communicate their measurements to a linked neighbor, which then passes the received as well as the locally available observations/measurements either to a neighboring node or to the fusion center. The major advantage in this cooperation strategy is that the fusion center can compute optimal estimates, since it has access to all the available information. However, the optimality is obtained under a number of drawbacks, such as: demand for increased communication costs and delays, especially for large networks. In addition, when the fusion center breaks down, the whole network collapses. Moreover, in certain applications, privacy issues are involved. For example, when data concern medical records, the nodes do not wish to send the available (training) data, but it is preferably to communicate certain locally obtained processed information. To overcome the drawbacks of the centralized processing scenario, different distributed processing schemes have been proposed.

[7] More rigorous definitions on graphs are given in Chapter 15.

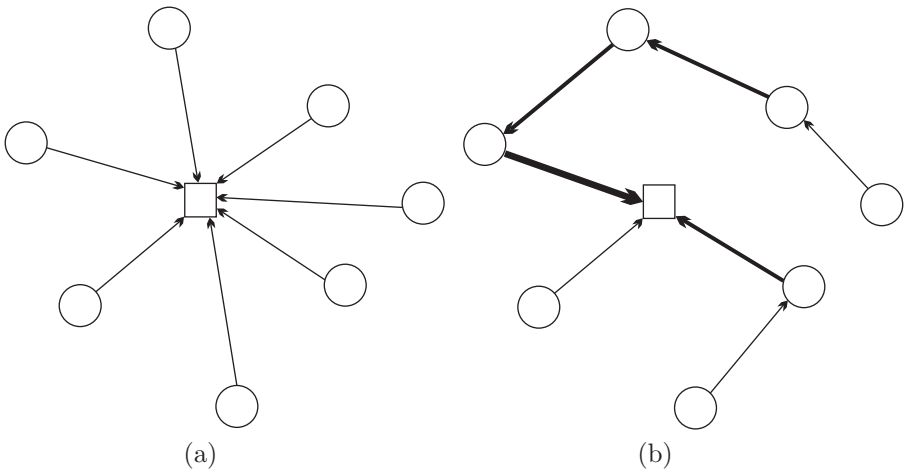


FIGURE 5.22

The square indicates the fusion center. (a) All nodes communicate directly to the fusion center. (b) Some nodes are connected directly to the fusion center. Others, communicate their own data to a neighboring node and so on, until the information reaches the fusion center. The bolder a connection is drawn, the higher the amount of data transmitted via the corresponding link.

Decentralized networks

Under this scenario, there is no central fusion center. Processing is performed *locally* at each node, employing the locally received measurements, and in the sequel, each node communicates the locally obtained estimates to its neighbors; that is, to the nodes it is linked with. These links are denoted as edges in the respective graph. There are different decentralized schemes.

- *Incremental/ring networks*: These require the existence of a *cyclic path*[8] following the edges through the network. Starting from a node, such a cycle has to visit every node, at least once, and then return to the first one. Such a topology implements an *iterative* computational scheme. At each iteration, every node performs its data acquisition and processing locally and communicates the required information to its neighbor in the cyclic path. It has been shown that incremental schemes achieve global performance (e.g., [58]). The main disadvantage of this mode of cooperation is that, cycling information around at each iteration is a problem in large networks. It is also important to stress at this point that the construction and maintenance of a cyclic graph, visiting each node, is an NP-hard task [52]. Moreover, the whole network collapses if one node malfunctions. The corresponding graph topology is shown in Figure 5.23a.
- *Ad hoc networks*: According to this philosophy of cooperation, nodes perform locally data acquisition as well as processing, at each iteration. However, the constraint of a cyclic path is removed. Each node communicates information to its neighboring nodes with which it shares an

[8] Consider a set of nodes $x_1, \ldots, x_k$ in a graph such that there is an edge connecting (x_{i-1}, x_i), $i = 2, \ldots k$. The set of edges connecting the k nodes is a path.

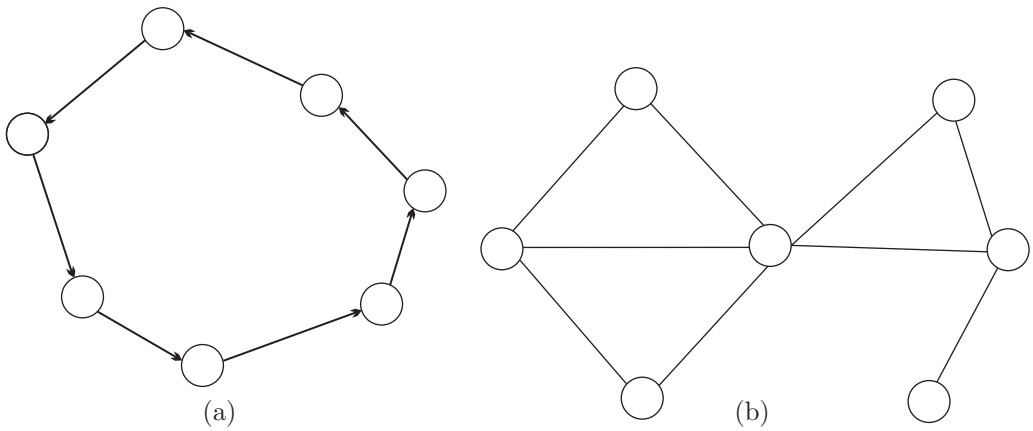


FIGURE 5.23

(a) The incremental or ring topology. The information flow follows a cyclic path. (b) Topology corresponding to a diffusion strategy. Each node communicates information to the nodes with which it shares an edge.

edge; in this way, information is *diffused* across the whole network. An advantage of such schemes is that operation is not seized if some nodes are malfunctioning. Also, the topology of the network may not be fixed. The price one pays for such "extras" is that the final obtained performance, after convergence, is inferior to those obtained by its incremental counterpart and by the centralized processing. This is natural, since at each iteration every node has access to only a limited amount of information. Figure 5.23b illustrates an example of the topology for ad hoc networks.

Besides the previous schemes, a number of variants exists. For example, the neighbors of each node may change probabilistically, which introduces randomness in the way information is diffused at each iteration [33, 60].

In this section, our focus will be on diffusion schemes. Our aim is to provide the reader with a sample of basic techniques around the LMS scheme and not to cover distributed learning in general, which is a field on its own with a long history; see [9, 19, 29, 51, 76, 95, 107] for sample references from classical to more recent contributions to the field. Besides distributed inference, a number of related aspects concerning the topology of the network and learning over graphs are attracting a lot of attention in the context of the emerging field of *complex networks*; see, [18, 37, 87, 89, 100] and the references therein.

5.13.2 THE DIFFUSION LMS

Let us consider a network of K agents/nodes. Each node exchanges information with the nodes in its neighborhood. Given a node, k, in a graph, let $\mathcal{N}_k$ be the set of nodes with which this node shares an edge; moreover, node k is also included in $\mathcal{N}_k$. This comprises the neighborhood set of k. We will denote the cardinality of this set as n_k. Figure 5.24 shows a graph with six nodes. For example, the neighborhood of node $k = 6$ is $\mathcal{N}_6 = \{2, 3, 6\}$, with cardinality $n_6 = 3$. The cardinality of $\mathcal{N}_k$ is also

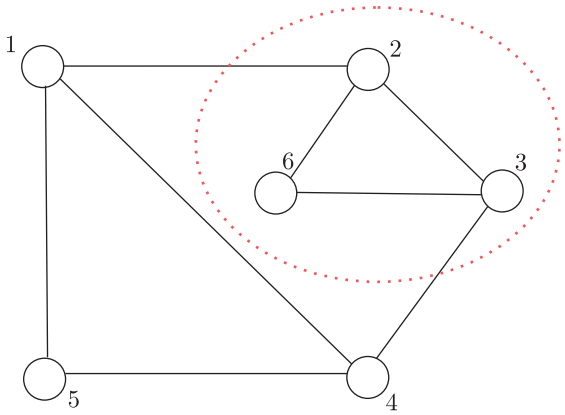


FIGURE 5.24

A graph corresponding to a network operating under a diffusion strategy. The red dotted line encircles the nodes comprising the neighborhood of node $k = 6$.

known as the *degree* of node k. On the contrary, nodes $k = 6$ and $k = 5$ are not neighbors, because they are not directly linked via an edge. For the needs of the section, we assume that the graph is a *strongly connected* one; that is, there is at least one path of edges that connects any pair of nodes.

Each node in the network has access to a local data acquisition process, that provides the pair of training data[9] $(y_k(n),\ \boldsymbol{x}_k(n))$, $k = 1, 2, \ldots, K,\ n = 0, 1, \ldots$, which are i.i.d. observations drawn from the respected stochastic zero-mean jointly distributed variables, $\mathrm{y}_k,\ \mathbf{x}_k,\ k = 1, \ldots, K$. We further assume that, in all cases, the pairs of the input-output variables are associated with a common to all (unknown) parameter vector $\boldsymbol{\theta}_o$. For example, in every node, the data are assumed to be generated by a corresponding regression model

$$\mathrm{y}_k = \boldsymbol{\theta}_o^{\mathrm{T}}\mathbf{x}_k + \eta_k, \quad k = 1, 2, \ldots, K, \tag{5.81}$$

where $\mathbf{x}_k$, as well as the zero mean noise variable, η_k, obey, in general, *different* statistical properties in each node. We will discuss such applications soon.

Treating each node individually, the MSE optimal solution, which minimizes the *local* cost function,

$$J_k(\boldsymbol{\theta}) = \mathbb{E}\left[|\mathrm{y}_k - \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}_k|^2\right],$$

will be given by the respective normal equations, involving the respective covariance matrix and cross-correlation vector; in other words,

$$\Sigma_{x_k}\boldsymbol{\theta}_* = \boldsymbol{p}_k. \tag{5.82}$$

Recall that for the case of a regression model, $\boldsymbol{\theta}_* = \boldsymbol{\theta}_o$ and the same solution results from all nodes. No doubt, if the statistics $\Sigma_{x_k},\ \boldsymbol{p}_k,\ k = 1, 2, \ldots, K$, were known we could stop here. However, we already know that this is not the case and in practice they have to be estimated. Alternatively, one has to resort

[9] The time index is used in parentheses, to unclutter notation due to the presence of the node index k.

to iterative techniques to learn the statistics as well as the unknown parameters. This is where one has to consider all nodes, to benefit from all the measurements/observations, which are distributed across the network. Thus, a more natural criterion to adopt is

$$J(\boldsymbol{\theta}) = \sum_{k=1}^{K} J_k(\boldsymbol{\theta}) = \sum_{k=1}^{K} \mathbb{E}\left[\left|\mathrm{y}_k - \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}_k\right|^2\right]. \tag{5.83}$$

Using the standard arguments, which we have employed a number of times so far, it is readily seen that the (common) estimate of the unknown $\boldsymbol{\theta}_o$ will be provided as a solution of

$$\left(\sum_{k=1}^{K} \Sigma_{x_k}\right)\boldsymbol{\theta}_* = \sum_{k=1}^{K} \boldsymbol{p}_k.$$

Let us use the global cost in (5.83) as our kick-off point to apply a gradient descent optimization scheme

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu \sum_{k=1}^{K} \left(\boldsymbol{p}_k - \Sigma_{x_k}\boldsymbol{\theta}^{(i-1)}\right), \tag{5.84}$$

from which a corresponding stochastic gradient scheme results, by replacing expectations with instantaneous observations and associating iteration steps with time updates, so that

$$\begin{aligned}\boldsymbol{\theta}_n &= \boldsymbol{\theta}_{n-1} + \mu \sum_{k=1}^{K} \boldsymbol{x}_k(n)e_k(n),\\ e_k(n) &= y_k(n) - \boldsymbol{\theta}_{n-1}^{\mathrm{T}}\boldsymbol{x}_k(n),\end{aligned}$$

and a constant step size has been used, adopting the rationale behind the classical LMS formulation. Such an LMS-type recursion is perfect for a centralized scenario, where all data are transmitted to a fusion center. This is one of the extremes, having at its opposite end the scenario with nodes acting individually without cooperation. However, there is an intermediate path, which will lead us to the distributed diffusion mode of operation.

Instead of trying to minimize (5.83), let us select a specific node k, and construct a local cost as the weighted aggregate in $\mathcal{N}_k$, represented by

$$J_k^{loc}(\boldsymbol{\theta}) = \sum_{m\in\mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}), \quad k = 1, 2, \ldots, K, \tag{5.85}$$

so that

$$\sum_{k=1}^{K} c_{mk} = 1, \quad c_{mk} \geq 0, \quad \text{and } c_{mk} = 0 \quad \text{if } m \notin \mathcal{N}_k, \quad m = 1, 2, \ldots, K. \tag{5.86}$$

Let C be the $K \times K$ matrix with entries $[C]_{mk} = c_{mk}$, then the summation condition in (5.86) can be written as

$$C\mathbf{1} = \mathbf{1}, \tag{5.87}$$

where $\mathbf{1}$ is the vector with all its entries being equal to 1. That is, all the entries across a row are summing to 1. Such matrices are known as *right stochastic* matrices. In contrast, a matrix is said to be left stochastic if

$$C^{\mathrm{T}}\mathbf{1} = \mathbf{1}.$$

Also, a matrix that is both left and right stochastic is known as *doubly stochastic* (Problem 5.16). Note that due to this matrix constraint, we still have that

$$\sum_{k=1}^{K} J_k^{loc}(\boldsymbol{\theta}) = \sum_{k=1}^{K} \sum_{m\in\mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}) = \sum_{k=1}^{K}\sum_{m=1}^{K} c_{mk} J_m(\boldsymbol{\theta})$$

$$= \sum_{m=1}^{K} J_m(\boldsymbol{\theta}) = J(\boldsymbol{\theta}).$$

That is, summing all local costs, the global one results.

Let us focus on minimizing (5.85). The gradient descent scheme results in

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m\in\mathcal{N}_k} c_{mk}\left(\boldsymbol{p}_m - \Sigma_{x_m}\boldsymbol{\theta}_k^{(i-1)}\right).$$

However, since nodes in the neighborhood exchange information, they could also share their current estimates. This is justified by the fact that the ultimate goal is to reach a common estimate; thus, exchanging current estimates could be used for the benefit of the algorithmic process to achieve this goal. To this end, we will modify the cost in (5.85) by regularizing it, leading to

$$\tilde{J}_k^{loc}(\boldsymbol{\theta}) = \sum_{m\in\mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}) + \lambda||\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}||^2, \tag{5.88}$$

where $\tilde{\boldsymbol{\theta}}$ encodes information with respect to the unknown vector, which is obtained by the neighboring nodes and $\lambda > 0$. Applying the gradient descent on (5.88) (and absorbing the factor "2", which comes from the exponents, into the step-size), we obtain

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m\in\mathcal{N}_k} c_{mk}\left(\boldsymbol{p}_m - \Sigma_{x_m}\boldsymbol{\theta}_k^{(i-1)}\right) + \mu_k\lambda(\tilde{\boldsymbol{\theta}} - \boldsymbol{\theta}_k^{(i-1)}), \tag{5.89}$$

which can be broken into the following two steps:

$$\text{Step 1:}\quad \boldsymbol{\psi}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m\in\mathcal{N}_k} c_{mk}\left(\boldsymbol{p}_m - \Sigma_{x_m}\boldsymbol{\theta}_k^{(i-1)}\right),$$

$$\text{Step 2:}\quad \boldsymbol{\theta}_k^{(i)} = \boldsymbol{\psi}_k^{(i)} + \mu_k\lambda(\tilde{\boldsymbol{\theta}} - \boldsymbol{\theta}_k^{(i-1)}).$$

Step 2 can slightly be modified and replace $\boldsymbol{\theta}_k^{(i-1)}$ by $\boldsymbol{\psi}_k^{(i)}$, since this encodes more recent information, and we obtain

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\psi}_k^{(i)} + \mu_k\lambda(\tilde{\boldsymbol{\theta}} - \boldsymbol{\psi}_k^{(i)}).$$

Furthermore, a reasonable choice of $\tilde{\boldsymbol{\theta}}$, at each iteration step, would be

$$\tilde{\boldsymbol{\theta}} = \tilde{\boldsymbol{\theta}}^{(i)} := \sum_{m\in\mathcal{N}_{k\backslash k}} b_{mk}\boldsymbol{\psi}_m^{(i)},$$

where

$$\sum_{m\in\mathcal{N}_{k\backslash k}} b_{mk} = 1, \quad b_{mk} \geq 0,$$

and $\mathcal{N}_{k\backslash k}$ denotes the elements in $\mathcal{N}_k$ excluding k. In other words, at each iteration, we update $\boldsymbol{\theta}_k$ so that to move it toward the descent direction of the local cost and at the same time we constrain it to stay close to the convex combination of the rest of the updates, which are obtained during the computations in step 1 from *all* the nodes in its neighborhood. Thus, we end up with the following recursions:

Diffusion gradient descent

$$\text{Step 1:} \quad \boldsymbol{\psi}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m\in\mathcal{N}_k} c_{mk}\left(\boldsymbol{p}_m - \Sigma_{x_m}\boldsymbol{\theta}_k^{(i-1)}\right), \tag{5.90}$$

$$\text{Step 2:} \quad \boldsymbol{\theta}_k^{(i)} = \sum_{m\in\mathcal{N}_k} a_{mk}\boldsymbol{\psi}_m^{(i)}, \tag{5.91}$$

where we set

$$a_{kk} = 1 - \mu_k\lambda \quad \text{and} \quad a_{mk} = \mu_k\lambda b_{mk}, \tag{5.92}$$

which leads to

$$\sum_{m\in\mathcal{N}_k} a_{mk} = 1, \quad a_{mk} \geq 0, \tag{5.93}$$

for small enough values of $\mu_k\lambda$. Note that by setting $a_{mk} = 0$, $m \notin \mathcal{N}_k$ and defining A to be the matrix with entries $[A]_{mk} = a_{mk}$, we can write

$$\sum_{m=1}^{K} a_{mk} = 1 \;\Rightarrow\; A^{\mathrm{T}}\mathbf{1} = 1, \tag{5.94}$$

that is, A is a left stochastic matrix. It is important to stress here that, irrespective of our derivation before, *any* left stochastic matrix A in (5.91) can be used.

A slightly different path to arrive at (5.89) is via the interpretation of the gradient descent scheme as a minimizer of a regularized linearization of the cost function around the currently available estimate. The regularizer used is $||\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)}||^2$ and it tries to keep the new update as close as possible to the currently available estimate. In the context of the distributed learning, instead of $\boldsymbol{\theta}^{(i-1)}$ we can use a convex combination of the available estimates obtained in the neighborhood [84], [26, 27].

We are now ready to state the first version of the diffusion LMS, by replacing in (5.90) and (5.91) expectations with instantaneous observations and interpreting iterations as time updates.

Algorithm 5.7 (The adapt-then-combine diffusion LMS).

- Initialize
  - **For** $k = 1, 2, \ldots, K$, **Do**
    - $\boldsymbol{\theta}_k(-1) = \mathbf{0} \in \mathbb{R}^l$; or any other value.
  - **End For**
  - Select $\mu_k,\ k = 1, 2, \ldots, K$; a small positive number.
  - Select C : $C\mathbf{1} = \mathbf{1}$
  - Select A : $A^{\mathrm{T}}\mathbf{1} = \mathbf{1}$

- **For** $n = 0, 1, \ldots,$ **Do**
  - **For** $k = 1, 2, \ldots, K,$ **Do**
    - **For** $m \in \mathcal{N}_k,$ **Do**
      - $e_{k,m}(n) = y_m(n) - \boldsymbol{\theta}_k^{\mathrm{T}}(n-1)\boldsymbol{x}_m(n)$; For complex-valued data, change $T \rightarrow H$.
    - **End Do**
    - $\boldsymbol{\psi}_k(n) = \boldsymbol{\theta}_k(n-1) + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk}\boldsymbol{x}_m(n)e_{k,m}(n)$; For complex-valued data, $e_{k,m}(n) \rightarrow e_{k,m}^*(n)$.
  - **End For**
  - **For** $k = 1, 2, \ldots, K$
    - $\boldsymbol{\theta}_k(n) = \sum_{m \in \mathcal{N}_k} a_{mk}\boldsymbol{\psi}_m(n)$
  - **End For**
- **End For**

The following comments are in order:

- This form of diffusion LMS (DiLMS) is known as *Adapt-then-Combine DiLMS* (ATC) since the first step refers to the update and the combination step follows.
- In the special case of $C = I$, then the adaptation step becomes

$$\boldsymbol{\psi}_k(n) = \boldsymbol{\theta}_k(n-1) + \mu\boldsymbol{x}_k(n)e_k(n),$$

  and nodes *need not exchange* their observations/measurements.
- The adaptation rationale is illustrated in Figure 5.25. At time n, all three neighbors exchange the received data. In case the input vector corresponds to a realization of a random signal, $\mathrm{u}_k(n)$, the exchange of information comprises two values $(y_k(n), u_k(n))$ in each direction for each one of the links. In the more general case, where the input is a random vector of jointly distributed variables, then all l variables have to be exchanged. After this message passing, adaptation takes place as shown in Figure 5.25a. Then, the nodes exchange their obtained estimates, $\boldsymbol{\psi}_k(n),\ k = 1, 2, 3,$ across the links, 5.25b.

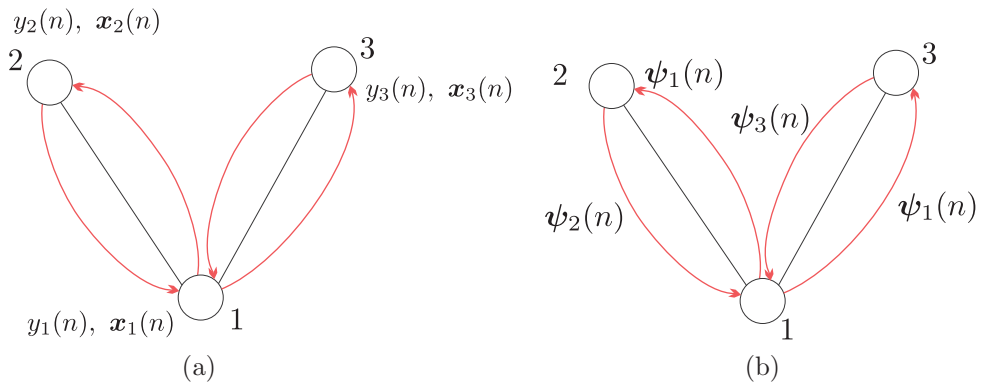


FIGURE 5.25

Adapt-then-Combine: (a) In step 1, adaptation is carried out after the exchange of the received observations. (b) In step 2, the nodes exchange their locally computed estimates to obtain the updated one.

A different scheme results if one reverses the order of the two steps and performs first the combination and then the adaptation.

Algorithm 5.8 (The combine-then-adapt diffusion LMS).

- Initialization
  - **For** $k = 1, 2, \ldots, K$, **Do**
    - $\boldsymbol{\theta}_k(-1) = \mathbf{0} \in \mathbb{R}^l$; or any other value.
  - **End For**
  - Select C : $C\mathbf{1} = \mathbf{1}$
  - Select A : $A^{\mathrm{T}}\mathbf{1} = \mathbf{1}$
  - Select $\mu_k,\ k = 1, 2, \ldots, K$; a small value.
- **For** $n = 0, 1, 2, \ldots,$ **Do**
  - **For** $k = 1, 2, \ldots, K$, **Do**
    - $\boldsymbol{\psi}_k(n-1) = \sum_{m \in \mathcal{N}_k} a_{mk}\boldsymbol{\theta}_m(n-1)$
  - **End For**
  - **For** $k = 1, 2, \ldots, K$, **Do**
    - **For** $m \in \mathcal{N}_k$, **Do**
      - $e_{k,m}(n) = y_m(n) - \boldsymbol{\psi}_k^{\mathrm{T}}(n-1)\boldsymbol{x}_m(n)$; For complex-valued data, change $T \rightarrow H$.
    - **End For**
    - $\boldsymbol{\theta}_k(n) = \boldsymbol{\psi}_k(n-1) + \mu_k \sum_{m \in \mathcal{N}_k} \boldsymbol{x}_m(n)e_{k,m}(n)$; For complex-valued data, $e_{k,m}(n) \rightarrow e^*_{k,m}(n)$.
  - **End For**
- **End For**

The rationale of this adaptation scheme is the reverse of that illustrated in Figure 5.25, where the phase in 5.25b precedes that of 5.25a. In case $C = I$, then there is no input-output data information exchange and the parameter update for the k node becomes

$$\boldsymbol{\theta}_k(n) = \boldsymbol{\psi}_k(n-1) + \mu_k \boldsymbol{x}_k(n)e_k(n).$$

Remarks 5.4.

- One of the early reports on the diffusion LMS can be found in [59]. In [80, 93], versions of the algorithm for diminishing step sizes are presented and its convergence properties are analyzed. Besides the DiLMS, a version for incremental distributed cooperation has been proposed in [58]. For a related review, see [84, 86, 87].
- So far, nothing has been said about the choice of the matrices C (A). There are a number of possibilities. Two popular choices are:

  Averaging Rule:

$$c_{mk} = \begin{cases} \frac{1}{n_k}, & \text{if } k = m, \text{ or if nodes } k \text{ and } m \text{ are neighbors,} \\ 0, & \text{otherwise,} \end{cases}$$

  and the respective matrix is left stochastic.

Metropolis Rule:

$$c_{mk} = \begin{cases} \frac{1}{\max\{n_k, n_m\}}, & \text{if } k \neq m \text{ and } k, m \text{ are neighbors,} \\ 1 - \sum_{i \in \mathcal{N}_k \backslash k} c_{ik}, & m = k, \\ 0, & \text{otherwise,} \end{cases}$$

which makes the respective matrix to be doubly stochastic.

- Distributed LMS-based algorithms for the case where different nodes estimate different, yet overlapping, parameter vectors, have also been derived, [20, 75].

5.13.3 CONVERGENCE AND STEADY-STATE PERFORMANCE: SOME HIGHLIGHTS

In this subsection, we will summarize some findings concerning the performance analysis of the DiLMS. We will not give proofs. The proofs follow similar lines as for the standard LMS, with a slightly more involved algebra. The interested reader can obtain proofs by looking at the original papers as well as in [84].

- The gradient descent scheme in (5.90), (5.91) is guaranteed to converge, meaning

$$\boldsymbol{\theta}_k^{(i)} \xrightarrow[i \to \infty]{} \boldsymbol{\theta}_*,$$

provided that

$$\mu_k \leq \frac{2}{\lambda_{\max}\{\Sigma_k^{loc}\}},$$

where

$$\Sigma_k^{loc} = \sum_{m \in \mathcal{N}_k} c_{mk} \Sigma_{x_m}. \tag{5.95}$$

This corresponds to the condition in (5.15).

- If one assumes that C is doubly stochastic, it can be shown that the convergence rate to the solution for the distributed case is higher than that corresponding to the noncooperative scenario, when each node operates individually, using the same step size, $\mu_k = \mu$, for all cases and provided this common value guarantees convergence. In other words, cooperation *improves the convergence speed*. This is in line with the general comments made in the beginning of the section.
- Assume that in the model in (5.81), the involved noise sequences are both spatially and temporally white, as represented by

$$\mathbb{E}[\eta_k(n)\eta_k(n-r)] = \sigma_k^2 \delta_r, \qquad \delta_r = \begin{cases} 1, & r = 0 \\ 0, & r \neq 0 \end{cases}$$

$$\mathbb{E}[\eta_k(n)\eta_m(r)] = \sigma_k^2 \delta_{km}\delta_{nr}, \qquad \delta_{km}, \delta_{nr} = \begin{cases} 1, & k = m,\ n = r \\ 0, & \text{otherwise.} \end{cases}$$

Also, the noise sequences are independent of the input vectors,

$$\mathbb{E}\left[\mathbf{x}_m(n)\eta_k(n-r)\right] = \mathbf{0}, \quad k, m = 1, 2, \ldots, K, \forall r,$$

and finally, the independence assumption is mobilized among the input vectors, spatially as well as temporally, namely

$$\mathbb{E}\left[\mathbf{x}_k(n)\mathbf{x}_m^T(n-r)\right] = O, \text{ if } k \neq m, \text{ and } \forall r.$$

Under the previous assumptions, which correspond to the assumptions adopted when studying the performance of the LMS, the following hold true for the diffusion LMS.
Convergence in the mean: Provided that

$$\mu_k < \frac{2}{\lambda_{\max}\{\Sigma_k^{\text{loc}}\}}, \tag{5.96}$$

then

$$\mathbb{E}\left[\theta_k(n)\right] \xrightarrow[n\to\infty]{} \boldsymbol{\theta}_*, \quad k = 1, 2, \ldots, K.$$

It is important to state here that the stability condition in (5.96) depends on C and not on A.

- If in addition to the previous assumption, C is chosen to be doubly stochastic, then the convergence in the mean, in any node under the distributed scenario, is *faster* than that obtained if the node is operating individually without cooperation, provided $\mu_k = \mu$ is the same and it is chosen so as to guarantee convergence.
- *Misadjustment*: under the assumptions of C and A being doubly stochastic, the following are true:
  - The average misadjustment over all nodes in the steady-state for the adapt-then-combine strategy is always smaller than that of the combine-then-adapt one.
  - The average misadjustment over all the nodes of the network in the distributed operation is always lower than that obtained if nodes are adapted individually, without cooperation, by using the same $\mu_k = \mu$ in all cases. That is, cooperation does not only improve convergence speed but it also *improves the steady-state performance*.

Example 5.8. In this example, a network of $L = 10$ nodes is considered. The nodes were randomly connected with a total number of 32 connections; the resulting network was checked out that it was strongly connected. In each node, data are generated according to a regression model, using the same vector $\boldsymbol{\theta}_o \in \mathbb{R}^{30}$. The latter was randomly generated by a $\mathcal{N}(0, 1)$. The input vectors, $\boldsymbol{x}_k$ in (5.81), were i.i.d. generated according to a $\mathcal{N}(0, 1)$ and the noise level was different for each node, varying from 20 to 25 dBs.

Three experiments were carried out. The first involved the distributed LMS in its adapt-then-combine (ATC) form and the second one the combine-then-adapt (CTA) version. In the third experiment, the LMS algorithm was run independently for each node, without cooperation. In all cases, the step size was chosen equal to $\mu = 0.01$. Figure 5.26 shows the average (over all nodes) MSD(n) : $\frac{1}{K}\sum_{k=1}^{K} ||\boldsymbol{\theta}_k(n) - \boldsymbol{\theta}_o||^2$ obtained for each one of the experiments. It is readily seen that cooperation improves the performance significantly, both in terms of convergence as well as in steady-state error floor. Moreover, as stated in Section 5.13.3, the ATC performs slightly better than the CTA version.

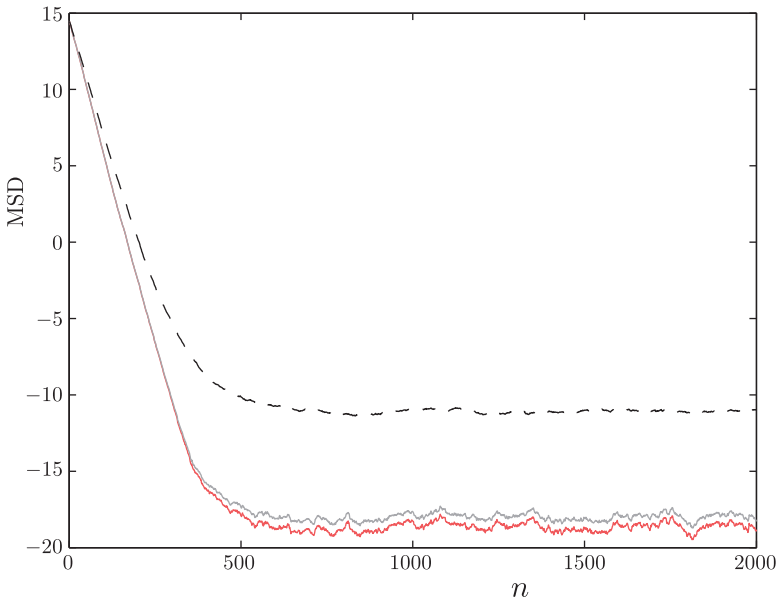


FIGURE 5.26

Average (over all the nodes) error convergence curves (MSD) for the LMS in noncooperative mode of operation (dotted line) and for the case of the diffusion LMS, in the ATC mode (red line) and the CTA mode (gray line). The step-size μ was the same in all three cases. Cooperation among nodes significantly improves performance. For the case of the diffusion LMS, the ATC version results in slightly better performance compared to that of the CTA.

5.13.4 CONSENSUS-BASED DISTRIBUTED SCHEMES

An alternative path for deriving an LMS version for distributed networks was followed in [64, 88]. Recall that, so far, in our discussion in deriving the DiLMS, we required the update at each node to be close to a convex combination of the available estimates in the respective neighborhood. Now we will demand such a requirement to become very strict. Although we are not going to get involved with details, since this would require to divert quite a lot from the material and the algorithmic tools which have been presented so far, let us state the task in the context of the linear MSE estimation.

To bring (5.83) in a distributed learning context, let us modify it by allowing different parameter vectors for each node, k, so that

$$J(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K) = \sum_{k=1}^{K} \mathbb{E}\left[\left|\mathrm{y}_k - \boldsymbol{\theta}_k^{\mathrm{T}} \mathbf{x}_k\right|^2\right].$$

Then, the task is cast according to the following constrained optimization problem,

$$\{\hat{\boldsymbol{\theta}}_k, k = 1, \ldots, K\} = \arg \min_{\{\boldsymbol{\theta}_k,\, k=1,\ldots,K\}} J(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K)$$
$$\text{s.t.} \qquad \boldsymbol{\theta}_k = \boldsymbol{\theta}_m, \quad k = 1, 2, \ldots, K,\ m \in \mathcal{N}_k.$$

In other words, one demands *equality* of the estimates within a neighborhood. As a consequence, these constraints lead to network-wise equality, since the graph that represents the network has been assumed to be connected. The optimization is carried out iteratively by employing stochastic approximation arguments and building on the alternating direction method of multipliers (ADMM) (Chapter 8), [19]. The algorithm, besides updating the vector estimates, has to update the associated Lagrange multipliers as well.

In addition to the previously reported ADMM-based scheme, a number of variants known as *consensus-based algorithms* have been employed in several studies [19, 33, 49, 50, 71]. A formulation around which a number of stochastic gradient consensus-based algorithms evolve is the following [33, 49, 50]:

$$\boldsymbol{\theta}_k(n) = \boldsymbol{\theta}_k(n-1) + \mu_k(n)\left[\boldsymbol{x}_k(n)e_k(n) + \lambda \sum_{m\in\mathcal{N}_k\backslash k} \big(\boldsymbol{\theta}_k(n-1) - \boldsymbol{\theta}_m(n-1)\big)\right], \tag{5.97}$$

where

$$e_k(n) := y_k(n) - \boldsymbol{\theta}_k^{\mathrm{T}}(n-1)\boldsymbol{x}_k(n)$$

and for some $\lambda > 0$. Observe the form in (5.97); the term in the bracket on the right-hand side is a regularizer whose goal is to enforce equality among the estimates within the neighborhood of node k. Several alternatives to the formula (5.97) have been proposed. For example, in [49] a different step size is employed for the consensus summation on the right-hand side of (5.97). In [99], the following formulation is provided,

$$\boldsymbol{\theta}_k(n) = \boldsymbol{\theta}_k(n-1) + \mu_k(n)\left[\boldsymbol{x}_k(n)e_k(n) + \sum_{m\in\mathcal{N}_k\backslash k} b_{m,k}\big(\boldsymbol{\theta}_k(n-1) - \boldsymbol{\theta}_m(n-1)\big)\right], \tag{5.98}$$

where $b_{m,k}$ stands for some nonnegative coefficients. If one defines the weights,

$$a_{m,k} := \begin{cases} 1 - \sum_{m\in\mathcal{N}_k\backslash k} \mu_k(n) b_{m,k}, & m = k \\ \mu_k(n) b_{m,k}, & m \in \mathcal{N}_k \setminus k, \\ 0, & \text{otherwise}, \end{cases} \tag{5.99}$$

recursion (5.98) can be equivalently written as,

$$\boxed{\boldsymbol{\theta}_k(n) = \sum_{m\in\mathcal{N}_k} a_{m,k}\boldsymbol{\theta}_m(n-1) + \mu_k(n)\boldsymbol{x}_k(n)e_k(n).} \tag{5.100}$$

The update rule in (5.100) is also referred to as *consensus strategy* (see, e.g., [99]). Note that the step-size is considered to be time-varying. In particular, in Refs. [19, 71], a diminishing step-size is employed, within the stochastic gradient rationale, which has to satisfy the familiar pair of conditions in order to guarantee convergence to a consensus value over all the nodes,

$$\sum_{n=0}^{\infty} \mu_k(n) = \infty, \quad \sum_{n=0}^{\infty} \mu_k^2(n) < \infty. \tag{5.101}$$

Observe the update recursion in (5.100). It is readily seen that the update $\boldsymbol{\theta}_k(n)$ involves only the error $e_k(n)$ of the corresponding node. In contrast, looking carefully at the corresponding update recursions

in both Algorithms 5.7, 5.8, $\boldsymbol{\theta}_k(n)$ is updated according to the average error within the neighborhood. This is an important difference.

The theoretical properties of the consensus recursion (5.100), which employs a *constant step-size*, as well as a comparative analysis against the diffusion schemes has been presented in [86, 99]. There, it has been shown that the diffusion schemes outperform the consensus-based ones, in the sense that (a) they converge faster, (b) they reach lower steady-state mean-square deviation error floor, and (c) their mean-square stability is insensitive to the choice of the combination weights.

5.14 A CASE STUDY: TARGET LOCALIZATION

Consider a network consisting of K nodes, whose goal is to estimate and track the location of a specific target. The location of the unknown target, say $\boldsymbol{\theta}_o$, is assumed to belong to the two-dimensional space. The position of each node is denoted by $\boldsymbol{\theta}_k = [\theta_{k1}, \theta_{k2}]^{\mathrm{T}}$, and the true distance between node k and the unknown target equals to

$$r_k = \|\boldsymbol{\theta}_o - \boldsymbol{\theta}_k\|. \tag{5.102}$$

The vector, whose direction points from node k to the unknown source, is given by,

$$\boldsymbol{g}_k = \frac{\boldsymbol{\theta}_o - \boldsymbol{\theta}_k}{\|\boldsymbol{\theta}_o - \boldsymbol{\theta}_k\|}. \tag{5.103}$$

Obviously, the distance can be rewritten in terms of the direction vector as,

$$r_k = \boldsymbol{g}_k^{\mathrm{T}}(\boldsymbol{\theta}_o - \boldsymbol{\theta}_k). \tag{5.104}$$

It is reasonable to assume that each node, k, "senses" the distance and the direction vectors via noisy observations. For example, such a noisy information can be inferred from the strength of the received signal or other related information. Following a similar rationale as in [84, 98], the noisy distance can be modeled as,

$$\hat{r}_k(n) = r_k + v_k(n), \tag{5.105}$$

where n stands for the discrete time instance and $v_k(n)$ for the additive noise term. The noise in the direction vector is a consequence of two effects: (a) a deviation occurring along the perpendicular direction to $\boldsymbol{g}_k$ and (b) a deviation that takes place along the parallel direction of $\boldsymbol{g}_k$. All in one, the noisy direction vector (see Figure 5.27), occurring at time instance n, can be written as,

$$\hat{\boldsymbol{g}}_k(n) = \boldsymbol{g}_k + v_k^{\perp}(n)\boldsymbol{g}_k^{\perp} + v_k^{\parallel}(n)\boldsymbol{g}_k, \tag{5.106}$$

where $v_k^{\perp}(n)$ is the noise corrupting the unit norm perpendicular direction vector $\boldsymbol{g}_k^{\perp}$ and $v_k^{\parallel}(n)$ is the noise occurring at the parallel direction vector. Taking into consideration the noisy terms, (5.105) is written as

$$\hat{r}_k(n) = \hat{\boldsymbol{g}}_k^{\mathrm{T}}(n)(\boldsymbol{\theta}_o - \boldsymbol{\theta}_k) + \eta_k(n), \tag{5.107}$$

where

$$\eta_k(n) = v_k(n) - v_k^{\perp}(n)\boldsymbol{g}_k^{\perp T}(\boldsymbol{\theta}_o - \boldsymbol{\theta}_k) - v_k^{\parallel}(n)\boldsymbol{g}_k^{T}(\boldsymbol{\theta}_o - \boldsymbol{\theta}_k). \tag{5.108}$$

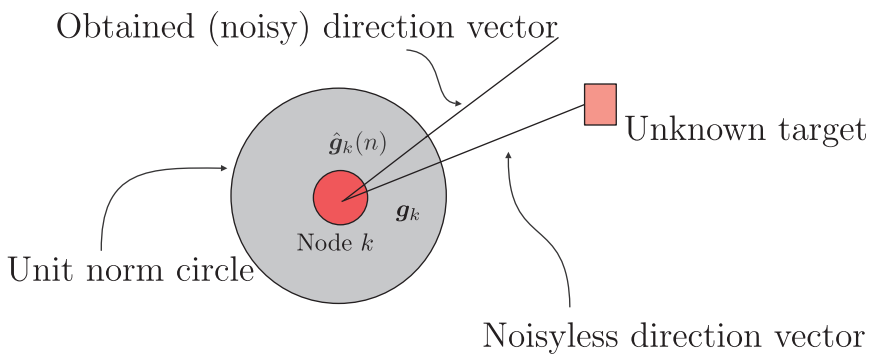


FIGURE 5.27

Illustration of a node, the target source, and the direction vectors.

Equation (5.108) can be further simplified if one recalls that by construction $\boldsymbol{g}_k^{\perp T}(\boldsymbol{\theta}_o - \boldsymbol{\theta}_k) = 0$. Moreover, typically, the contribution of $v_k^{\perp}(n)$ is assumed to be significantly larger than the contribution of $v_k^{\parallel}(n)$. Henceforth, taking into consideration these two arguments, (5.108) can be simplified to

$$\eta_k(n) \approx v_k(n). \tag{5.109}$$

If one defines $y_k(n) := \hat{r}_k(n) + \hat{\boldsymbol{g}}_k^T(n)\boldsymbol{\theta}_k$ and combines (5.107) with (5.109) the following model results:

$$y_k(n) \approx \boldsymbol{\theta}_o^T \hat{\boldsymbol{g}}_k(n) + v_k(n). \tag{5.110}$$

Equation (5.110) is a linear regression model. Using the available estimates, for each time instant, one has access to $y_k(n)$, $\hat{\boldsymbol{g}}_k(n)$ and any form of distributed algorithm can be adopted in order to obtain a better estimate of $\boldsymbol{\theta}_o$.

Indeed, it has been verified that the information exchange and fusion enhances significantly the ability of the nodes to estimate and track the target source. The nodes can possibly represent fish schools, which seek a nutrition source, bee swarms, which search for their hive, or bacteria seeking nutritive sources [28, 84, 85, 97].

Some other typical applications of distributed learning are social networks [36], radio resource allocation [32], and network cartography [65].

5.15 SOME CONCLUDING REMARKS: CONSENSUS MATRIX

In our treatment of the diffusion LMS, we used the combination matrices A (C), which we assumed to be left (right) stochastic. Also, in the performance-related section, we pointed out that some of the reported results hold true if these matrices are, in addition, doubly stochastic. Although it was not needed in this chapter, in the general distributed processing theory, a matrix of significant importance is the so-called *consensus matrix*. A matrix $A \in \mathbb{R}^{K\times K}$ is said to be a consensus matrix, if in addition to being doubly stochastic, as represented by

$$A\mathbf{1} = \mathbf{1}, \quad A^{\mathrm{T}}\mathbf{1} = \mathbf{1},$$

it also satisfies the property,

$$\left|\lambda_i\{A^{\mathrm{T}} - \frac{1}{K}\mathbf{1}\mathbf{1}^{\mathrm{T}}\}\right| < 1, \quad i = 1, 2, \ldots, K.$$

In other words, all eigenvalues of the matrix

$$A^{\mathrm{T}} - \frac{1}{K}\mathbf{1}\mathbf{1}^{\mathrm{T}}$$

have magnitude strictly less than one. To demonstrate its usefulness, we will state a fundamental theorem in distributed learning.

Theorem 5.2. *Consider a network consisting of K nodes, and each one of them having access to a state vector* $\boldsymbol{x}_k$. *Consider the recursion,*

$$\boldsymbol{\theta}_k^{(i)} = \sum_{m \in \mathcal{N}_k} a_{mk}\boldsymbol{\theta}_m^{(i-1)}, \quad k = 1, 2, \ldots, K, \; i > 0 : \text{Consensus Iteration},$$

with

$$\boldsymbol{\theta}_k^{(0)} = \boldsymbol{x}_k, \quad k = 1, 2, \ldots, K.$$

Define $A \in \mathbb{R}^{K \times K}$, *to be the matrix with entries*

$$[A]_{mk} = a_{mk}, \quad m, k = 1, 2, \ldots, K,$$

where $a_{mk} \geq 0$, $a_{mk} = 0$ *if* $m \notin \mathcal{N}_k$. *If A is a consensus matrix, then [31],*

$$\boldsymbol{\theta}_k^{(i)} \longrightarrow \frac{1}{K}\sum_{k=1}^{K}\boldsymbol{x}_k.$$

The opposite is also true. If convergence is always guaranteed, then A is a consensus matrix.

In other words, this theorem states that updating each node by convexly combining, with appropriate weights, the current estimates in its neighborhood, then the network converges to the average value in a consensus rationale (Problem 5.17).

PROBLEMS

5.1 Show that the gradient vector is perpendicular to the tangent at a point of an isovalue curve.

5.2 Prove that if

$$\sum_{i=1}^{\infty}\mu_i^2 < \infty, \quad \sum_{i=1}^{\infty}\mu_i = \infty,$$

the steepest descent scheme, for the MSE loss function and for the iteration-dependent step size case, converges to the optimal solution.

5.3 Derive the steepest gradient descent direction for the complex-valued case.

5.4 Let θ, x be two jointly distributed random variables. Let also the function (regressor)

$$f(\theta) = \mathbb{E}[\mathrm{x}|\theta].$$

Show that under the conditions in (5.28), the recursion

$$\theta_n = \theta_{n-1} - \mu_n x_n$$

converges in probability to a root of $f(\theta)$.

5.5 Show that the LMS algorithm is a nonlinear estimator.

5.6 Show Eq. (5.41).

5.7 Derive the bound in (5.44).

Hint. Use the well-known property from linear algebra, that the eigenvalues of a matrix, $A \in \mathbb{R}^{l\times l}$, satisfy the following bound,

$$\max_{1\le i\le l} |\lambda_i| \le \max_{1\le i\le l} \sum_{j=1}^{l} |a_{ij}| := \|A\|_1.$$

5.8 *Gershgorin circle theorem*. Let A be an $l \times l$ matrix, with entries a_{ij}, $i,j = 1, 2, \ldots, l$. Let $R_i := \sum_{\substack{j=1 \\ j\neq i}}^{l} |a_{ij}|$, be the sum of absolute values of the nondiagonal entries in row i. Then show that if λ is an eigenvalue of A, then there exists at least one row i, such that the following is true,

$$|\lambda - a_{ii}| \le R_i.$$

The last bound defines a circle, which contains the eigenvalue λ.

5.9 Apply the Gershgorin circle theorem to prove the bound in (5.44).

5.10 Derive the misadjustment formula given in (5.51).

5.11 Derive the APA iteration scheme.

5.12 Given a value $\boldsymbol{x}$, define the hyperplane comprising all values of $\boldsymbol{\theta}$ such as

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\theta} - y = 0.$$

Then $\boldsymbol{x}$ is perpendicular to the hyperplane.

5.13 Derive the recursions for the widely linear APA.

5.14 Show that a similarity transformation of a square matrix, via a unitary matrix does not affect the eigenvalues.

5.15 Show that if $\mathbf{x} \in \mathbb{R}^l$ is a Gaussian random vector, then

$$F := \mathbb{E}[\mathbf{x}\mathbf{x}^{\mathrm{T}} S \mathbf{x}\mathbf{x}^{\mathrm{T}}] = \Sigma_x \mathrm{trace}\{S\Sigma_x\} + 2\Sigma_x S \Sigma_x,$$

and if $\mathbf{x} \in \mathbb{C}^l$,

$$F := \mathbb{E}[\mathbf{x}\mathbf{x}^{\mathrm{H}} S \mathbf{x}\mathbf{x}^{\mathrm{H}}] = \Sigma_x \mathrm{trace}\{S\Sigma_x\} + \Sigma_x S \Sigma_x.$$

5.16 Show that if a $l \times l$ matrix C is right stochastic, then all its eigenvalues satisfy

$$|\lambda_i| \le 1, \quad i = 1, 2, \ldots, l.$$

The same holds true for left and doubly stochastic matrices.

5.17 Prove Theorem 5.2.

MATLAB Exercises

5.18 Consider the MSE cost function in (5.4). Set the cross-correlation equal to $\boldsymbol{p} = [0.05, 0.03]^{\mathrm{T}}$. Also, consider two covariance matrices,

$$\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Compute the corresponding optimal solutions, $\boldsymbol{\theta}_{(*,1)} = \Sigma_1^{-1}\boldsymbol{p}$, $\boldsymbol{\theta}_{(*,2)} = \Sigma_2^{-1}\boldsymbol{p}$. Apply the gradient descent scheme of (5.6) to estimate $\boldsymbol{\theta}_{(*,2)}$; set the step-size equal to (a) its optimal value μ_o according to (5.16) and (b) equal to $\mu_o/2$. For these two choices for the step-size, plot the error $\|\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_{(*,2)}\|^2$, at each iteration step i. Compare the convergence speeds of these two curves towards zero. Moreover, in the two-dimensional space, plot the coefficients of the successive estimates, $\boldsymbol{\theta}^{(i)}$, for both step-sizes, together with the isovalue contours of the cost function. What do you observe regarding the trajectory towards the minimum?
Apply (5.6) for the estimation of $\boldsymbol{\theta}_{(*,1)}$ employing Σ_1^{-1} and $\boldsymbol{p}$. Use as step size μ_o of the previous experiment. Plot, in the same figure, the previously computed error curve $\|\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_{(*,2)}\|^2$ together with the error curve $\|\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_{(*,1)}\|^2$. Compare the convergence speeds. Set now the step size equal to the optimum value associated with Σ_1. Again, in the two-dimensional space, plot the values of the successive estimates and the isovalue contours of the cost function. Compare the number of steps needed for convergence, with the ones needed in the previous experiment. Play with different covariance matrices and step sizes.

5.19 Consider the linear regression model

$$y_n = \boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{\theta}_o + \eta_n,$$

where $\boldsymbol{\theta}_o \in \mathbb{R}^2$. Generate the coefficients of the unknown vector, $\boldsymbol{\theta}_o$, randomly according to the normalized Gaussian distribution, $\mathcal{N}(0, 1)$. The noise is assumed to be white Gaussian with variance 0.1. The samples of the input vector are i.i.d. generated via the normalized Gaussian. Apply the Robbins-Monro algorithm in (5.33), for the optimal MSE linear estimation, with a step size equal to $\mu_n = 1/n$. Run 1000 independent experiments and plot the mean value of the first coefficient of the 1000 produced estimates, at each iteration step. Also, plot the horizontal line crossing the true value of the first coefficient of the unknown vector. Furthermore, plot the standard deviation of the obtained estimate, every 30 iteration steps. Comment on the results. Play with different rules of diminishing step-sizes and comment on the results.

5.20 Generate data according to the regression model

$$y_n = \boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{\theta}_o + \eta_n,$$

where $\boldsymbol{\theta}_o \in \mathbb{R}^{10}$, and whose elements are randomly obtained using the Gaussian distribution $\mathcal{N}(0, 1)$. The noise samples are also i.i.d. generated from $\mathcal{N}(0, 0.01)$.

Generate the inputs samples as part of two processes: (a) a white noise sequence, i.i.d. generated via $\mathcal{N}(0, 1)$ and (b) an autoregressive AR(1) process with $a_1 = 0.85$ and the corresponding white noise excitation is of variance equal to 1. For these two choices of the input, run the LMS algorithm, on the generated training set $(y_n, \boldsymbol{x}_n)$, $n = 0, 1, \ldots$, to estimate $\boldsymbol{\theta}_o$. Use a step size equal to $\mu = 0.01$. Run 100 independent experiments and plot the average error per iteration in dBs, using $10\log_{10}(e_n^2)$, with $e_n^2 = (y_n - \hat{\boldsymbol{\theta}}_{n-1}^{\mathrm{T}}\boldsymbol{x}_n)^2$. What do you observe regarding the convergence speed of the algorithm for the two cases? Repeat the experiment with different values of the AR coefficient a_1 and different values of the step-size. Observe how

the learning curve changes with the different values of the step-size and/or the value of the AR coefficient. Choose, also, a relatively large value for the step-size and make the LMS algorithm to diverge. Comment and justify theoretically the obtained results concerning convergence speed and the error floor at the steady-state after convergence.

5.21 Use the data set generated form the the AR(1) process of the previous exercise. Employ the transform-domain LMS (Algorithm 5.6) with step-size equal to 0.01. Also, set $\delta = 0.01$ and $\beta = 0.5$. Moreover, employ the DCT transform. As in the previous exercise, run 100 independent experiments and plot the average error per iteration. Compare the results with that of the LMS with the same step-size.

Hint. Compute the DCT transformation matrix using the *dctmtx* MATLAB function.

5.22 Generate the same experimental setup, as in Exercise 5.20, with the difference that $\boldsymbol{\theta}_o \in \mathbb{R}^{60}$. For the LMS algorithm set $\mu = 0.025$ and for the NLMS (Algorithm 5.3) $\mu = 0.35$ and $\delta = 0.001$. Employ also the APA (Algorithm 5.2) algorithm with parameters $\mu = 0.1$, $\delta = 0.001$ and $q = 10,\ 30$. Plot in the same figure the error learning curves of all these algorithms, as in the previous exercises. How does the choice of q affect the behavior of the APA algorithm, both in terms of convergence speed as well as the error floor at which it settles after convergence? Play with different values of q and of the step-size μ.

5.23 Consider the decision feedback equalizer described in Section 5.10.

- **(a)** Generate a set of 1000 random ± 1 values (BPSK) (i.e., s_n). Direct this sequence into a linear channel with impulse response $\boldsymbol{h} = [0.04, -0.05, 0.07, -0.21, 0.72, 0.36, 0.21, 0.03, 0.07]^T$ and add to the output 11dB white Gaussian noise. Denote the output as u_n.
- **(b)** Design the adaptive decision feedback equalizer (DFE) using $L = 21$, $l = 10$, and $\mu = 0.025$ following the training mode only. Perform a set of 500 experiments feeding the DFE with different random sequences from the ones described in step 5.23a. Plot the mean-square error (averaged over the 500 experiments). Observe that around $n = 250$ the algorithm achieves convergence.
- **(c)** Design the adaptive decision feedback equalizer using the parameters of step 5.23b. Feed the equalizer with a series of 10,000 random values generated as in step 5.23a. After the 250th data sample, change the DFE to decision-directed mode. Count the percentage of the errors performed by the equalizer from the 251th to the 10,000th sample.
- **(d)** Repeat steps 5.23a to 5.23c changing the level of the white Gaussian noise added to the BPSK values to 15, 12, 10 dBs. Then, for each case, change the delay to $L = 5$. Comment on the results.

5.24 Develop the MATLAB code for the two forms of the diffusion LMS, adapt-then-combine (ATC) and combine-then-adapt (CTA) and reproduce the results of Example 5.8. Play with the choice of the various parameters. Make sure that the resulting network is strongly connected.

REFERENCES

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, A survey on sensor networks, IEEE Commun. Mag. 40(8) (2002) 102-114.

[2] S.J.M. Almeida, J.C.M. Bermudez, N.J. Bershad, M.H. Costa, A statistical analysis of the affine projection algorithm for unity step size and autoregressive inputs, IEEE Trans. Circuits Syst. I 52(7) (2005) 1394-1405.

[3] T.Y. Al-Naffouri, A.H. Sayed, Transient analysis of data-normalized adaptive filters, IEEE Trans. Signal Process. 51(3) (2003) 639-652.

[4] S. Amari, Theory of adaptive pattern classifiers, IEEE Trans. Electron. Comput. 16(3) (1967) 299-307.
[5] C. Antweiler, M. Dörbecker, Perfect sequence excitation of the NLMS algorithm and its application to acoustic echo control, Ann. Telecommun. 49(7-8) (1994) 386-397.
[6] J.A. Appolinario, S. Werner, P.S.R. Diniz, T.I. Laakso, Constrained normalized adaptive filtering for CDMA mobile communications, in: Proceedings, EUSIPCO, Rhodes, Greece, 1998.
[7] J.A. Appolinario, M.L.R. de Campos, P.S.R. Diniz, The binormalized data-reusing LMS algorithm, IEEE Trans. Signal Process. 48 (2000) 3235-3242.
[8] J. Arenas-Garcia, A.R. Figueiras-Vidal, A.H. Sayed, Mean-square performance of a convex combination of two adaptive filters, IEEE Trans. Signal Process. 54(3) (2006) 1078-1090.
[9] S. Barbarossa, G. Scutari, Bio-inspired sensor network design: Distributed decisions through self-synchronization, IEEE Signal Process. Mag. 24(3) (2007) 26-35.
[10] J. Benesty, T. Gänsler, D.R. Morgan, M.M. Sondhi, S.L. Gay, Advances in Network and Acoustic Echo Cancellation, Springer Verlag, Berlin, 2001.
[11] J. Benesty, S.L. Gay, An improved PNLMS algorithm, in: IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 2, 2002.
[12] J. Benesty, C. Paleologu, S. Ciochina, On regularization in adaptive filtering, IEEE Trans. Audio Speech Lang. Process. 19(6) (2011) 1734-1742.
[13] A. Benvenniste, M. Metivier, P. Piouret, Adaptive Algorithms and Stochastic Approximations, Springer-Verlag, NY, 1987.
[14] K. Berberidis, P. Karaivazoglou, An efficient block adaptive DFE implemented in the frequency domain, IEEE Trans. Signal Proc. 50 (9) (2002) 2273–2286.
[15] N.J. Bershad, Analysis of the normalized LMS with Gaussian inputs, IEEE Trans. Acoust. Speech Signal Process. 34(4) (1986) 793-806.
[16] N.J. Bershad, O.M. Macchi, Adaptive recovery of a chirped sinusoid in noise. Part 2: Performance of the LMS algorithm, IEEE Trans. Signal Process. 39 (1991) 595-602.
[17] J.C.M. Bermudez, N.J. Bershad, A nonlinear analytical model for the quantized LMS algorithm: The arbitrary step size case, IEEE Trans. Signal Process. 44 (1996) 1175-1183.
[18] A. Bertrand, M. Moonen, Seeing the bigger picture, IEEE Signal Process. Mag. 30(3) (2013) 71-82.
[19] D.P. Bertsekas, J.N. Tsitsiklis, Parallel and Distributed Computations: Numerical Methods, Athena Scientific, Belmont, MA, 1997.
[20] N. Bogdanovic, J. Plata-Chaves, K. Berberidis, Distributed incremental-based LMS for node-specific adaptive parameter estimation, IEEE Trans. Signal Proc. 62 (20) (2014) 5382–5397.
[21] N. Cesa-Bianchi, P.M. Long, M.K. Warmuth, Worst case quadratic loss bounds for prediction using linear functions and gradient descent, IEEE Trans. Neural Networks 7(3) (1996) 604-619.
[22] P. Bouboulis, S. Theodoridis, Extension of Wirtinger's Calculus to Reproducing Kernel Hilbert Spaces and the Complex Kernel LMS, IEEE Trans. Signal Process. 53(3) (2011) 964-978.
[23] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.
[24] A. Carini, Efficient NLMS and RLS Algorithms for perfect and imperfect periodic sequences, IEEE Trans. Signal Process. 58(4) (2010) 2048-2059.
[25] A. Carini, G.L. Sicuranza, V.J. Mathews, Efficient adaptive identification of linear-in-the-parameters nonlinear filters using periodic input sequences, Signal Process. 93(5) (2013) 1210-1220.
[26] F.S. Cattivelli, A.H. Sayed, Diffusion LMS strategies for distributed estimation, IEEE Trans. Signal Process. 58(3) (2010) 1035-1048.
[27] F.S. Cattivelli, Distributed Collaborative Processing over Adaptive Networks, Ph.D. Thesis, University of California, LA, 2010.
[28] J. Chen, A.H. Sayed, Bio-inspired cooperative optimization with application to bacteria mobility, in: IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, 2011, pp. 5788-5791.
[29] S. Chouvardas, Y. Kopsinis, S. Theodoridis, Sparsity-aware distributed learning, in: S. Cui, A. Hero, J. Moura, Z.Q. Luo (Eds.), Big Data over Networks, Cambridge University Press, 2014.

[30] T.A.C.M. Claasen, W.F.G. Mecklenbrauker, Comparison of the convergence of two algorithms for adaptive FIR digital filters, IEEE Trans. Acoust. Speech Signal Process. 29 (1981) 670-678.
[31] M.H. DeGroot, Reaching a consensus, J. Amer. Stat. Assoc. 69(345) (1974) 118-121.
[32] P. Di Lorenzo, S. Barbarossa, Swarming algorithms for distributed radio resource allocation, IEEE Signal Process. Mag. 30(3) (2013) 144-154.
[33] A.G. Dimakis, S. Kar, J.M.F. Moura, M.G. Rabbat, A. Scaglione, Gossip algorithms for distributed signal processing, Proc. IEEE 98(11) (2010) 1847-1864.
[34] P.S.R. Diniz, Adaptive Filtering: Algorithms and Practical Implementation, fourth ed., Springer, 2013.
[35] D.L. Duttweiler, Proportionate NLMS adaptation in echo cancelers, IEEE Trans. Audio Speech Lang. Process. 8 (2000) 508-518.
[36] C. Chamley, A. Scaglione, L. Li, Models for the diffusion of belief in social networks, IEEE Signal Process. Mag. 30(3) (2013) 16-28.
[37] C. Eksin, P. Molavi, A. Ribeiro, A. Jadbabaie, Learning in network games with incomplete information, IEEE Signal Process. Mag. 30(3) (2013) 30-42.
[38] D.C. Farden, Tracking properties of adaptive signal processing algorithms, IEEE Trans. Acoust. Speech Signal Process. 29 (1981) 439-446.
[39] E.R. Ferrara, Fast implementations of LMS adaptive filters, IEEE Trans. Acoust. Speech Signal Process. 28 (1980) 474-475.
[40] O.L. Frost III, An algorithm for linearly constrained adaptive array processing, Proc. IEEE 60 (1972) 962-935.
[41] I. Furukawa, A design of canceller of broadband acoustic echo, in: Proceedings, International Teleconference Symposium, 1984, pp. 1-8.
[42] S.L. Gay, S. Tavathia, The fast affine projection algorithm, in: Proceedings International Conference on Acoustics, Speech and Signal Processing, ICASSP, 1995, pp. 3023-3026.
[43] S.L. Gay, J. Benesty, Acoustical Signal Processing for Telecommunications, Kluwer, 2000.
[44] A. Gersho, Adaptive equalization of highly dispersive channels for data transmission, Bell Syst. Tech. J. 48 (1969) 55-70.
[45] A. Gilloire, M. Vetterli, Adaptive filtering in subbands with critical sampling: analysis, experiments and applications to acoustic echo cancellation, IEEE Trans. Signal Process. 40 (1992) 1862-1875.
[46] B. Hassibi, A.H. Sayed, T. Kailath, H^{∞} optimality of the LMS algorithm, IEEE Trans. Signal Process. 44(2) (1996) 267-280.
[47] S. Haykin, Adaptive Filter Theory, fourth ed., Pentice Hall, 2002.
[48] T. Hinamoto, S. Maekawa, Extended theory of learning identification, IEEE Trans. 95(10) (1975) 227-234 (in Japanese).
[49] S. Kar, J. Moura, Convergence rate analysis of distributed gossip (linear parameter) estimation: fundamental limits and tradeoffs, IEEE J Select. Topics Signal Process. 5(4) (2011) 674-690.
[50] S. Kar, J. Moura, K. Ramanan, Distributed parameter estimation in sensor networks: nonlinear observation models and imperfect communication, IEEE Trans. Informat. Theory 58(6) (2012) 3575-3605.
[51] S. Kar, J.M.F. Moura, Consensus + innovations distributed inference over networks, IEEE Signal Process. Mag. 30(3) (2013) 99-109.
[52] R.M. Karp, Reducibility among combinational problems, in: R.E. Miller, J.W. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, NY, 1972, pp. 85-104.
[53] W. Kellermann, Kompensation akustischer echos in frequenzteilbandern, Aachener Kolloquim, Aachen, Germany, 1984, pp. 322-325.
[54] W. Kellermann, Analysis and design of multirate systems for cancellation of acoustical echos, in: Proceedings, IEEE International Conference on Acoustics, Speech and Signal Processing, New York, 1988, pp. 2570-2573.
[55] J. Kivinen, M.K. Warmuth, B. Hassibi, The p-norm generalization of the LMS algorithms for filtering, IEEE Trans. Signal Process. 54(3) (2006) 1782-1793.

[56] H.J. Kushner, G.G. Yin, Stochastic Approximation Algorithms and Applications, Springer, New York, 1997.
[57] L. Ljung, System Identification: Theory for the User, Prentice Hall, Englewood Cliffs, NJ, 1987.
[58] C.G. Lopes, A.H. Sayed, Incremental adaptive strategies over distributed networks, IEEE Trans. Signal Process. 55(8) (2007) 4064-4077.
[59] C.G. Lopes, A.H. Sayed, Diffusion least-mean-squares over adaptive networks: Formulation and performance analysis, IEEE Transactions on Signal Processing 56(7) (2008) 3122-3136.
[60] C. Lopes, A.H. Sayed, Diffusion adaptive networks with changing topologies, in: Proceedings International Conference on Acoustics, Speech and Signal processing, CASSP, Las Vegas, April 2008, pp. 3285-3288.
[61] O.M. Macci, N.J. Bershad, Adaptive recovery of chirped sinusoid in noise. Part 1: Performance of the RLS algorithm, IEEE Trans. Signal Process. 39 (1991) 583-594.
[62] O. Macchi, Adaptive Processing: The Least-Mean-Squares Approach with Applications in Transmission, Wiley, New York, 1995.
[63] V.J. Mathews, S.H. Cho, Improved convergence analysis of stochastic gradient adaptive filters using the sign algorithm, IEEE Trans. Acoust. Speech Signal Process. 35 (1987) 450-454.
[64] G. Mateos, I.D. Schizas, G.B. Giannakis, Performance analysis of the consensus-based distributed LMS algorithm, EURASIP J. Adv. Signal Process. Article: ID981030, doi: 10.1155/2009/981030, 2009.
[65] G. Mateos, K. Rajawat, Dynamic network cartography, IEEE Signal Process. Mag. 30(3) (2013) 129-143.
[66] R. Merched, A. Sayed, An embedding approach to frequency-domain and subband filtering, IEEE Trans. Signal Process. 48(9) (2000) 2607-2619.
[67] N. Murata, A statistical study on online learning, in: D. Saad (Ed.), Online Learning and Neural Networks, Cambridge University Press, UK, 1998, pp. 63-92.
[68] S.S. Narayan, A.M. Peterson, Frequency domain LMS algorithm, Proc. IEEE 69(1) (1981) 124-126.
[69] V.H. Nascimento, J.C.M. Bermudez, Probability of divergence for the least mean fourth algorithm, IEEE Trans. Signal Process. 54 (2006) 1376-1385.
[70] V.H. Nascimento, M.T.M. Silva, Adaptive filters, in: R. Chellappa, S. Theodoridis (Eds.), Signal Process, E-Ref. 1 (2014) 619-747.
[71] A. Nedic, A. Ozdaglar, Distributed subgradient methods for multi-agent optimization, IEEE Trans. Automat. Control 54(1) (2009) 48-61.
[72] K. Ozeki, T. Umeda, An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties, IEICE Trans. 67-A(5) (1984) 126-132 (in Japanese).
[73] C. Paleologu, J. Benesty, S. Ciochina, Regularization of the affine projection algorithm, IEEE Trans. Circuits Syst. II: Express Briefs 58(6) (2011) 366-370.
[74] A. Papoulis, S.U. Pillai, Probability, Random Variables and Stochastic Processes, fourth ed., McGraw Hill, 2002.
[75] J. Plata-Chaves, N. Bogdanovic, K. Berberidis, Distributed diffusion-based LMS for node-specific adaptive parameter estimation, *arXiv:1408.3354* (2014). http://arxiv.org/abs/1408.3354.
[76] J.B. Predd, S.R. Kulkarni, H.V. Poor, Distributed learning in wireless sensor networks, IEEE Signal Process. Mag. 23(4) (2006) 56-69.
[77] J. Proakis, Digital Communications, fourth ed., McGraw Hill, New York, 2000.
[78] J. Proakis, J.H. Miller, Adaptive receiver for digital signalling through channels with intersymbol interference, IEEE Trans. Informat. Theory 15 (1969) 484-497.
[79] H. Robbins, S. Monro, A stochastic approximation method, Ann. Math. Stat. 22 (1951) 400-407.
[80] S.S. Ram, A. Nedich, V.V. Veeravalli, Distributed stochastic subgradient projection algorithms for convex optimization, J. Optim. Theory Appl. 147(3) (2010) 516-545.
[81] M. Martinez-Ramon, J. Arenas-Garcia, A. Navia-Vazquez, A.R. Figueiras-Vidal, An adaptive combination of adaptive filters for plant identification, in: Proceedings the 14th International Conference on Digital Signal Processing (DSP), 2002, pp. 1195-1198.

[82] A.H. Sayed, M. Rupp, Error energy bounds for adaptive gradient algorithms, IEEE Trans. Signal Process 44(8) (1996) 1982-1989.
[83] A.H. Sayed, Fundamentals of Adaptive Filtering, John Wiley, 2003.
[84] A.H. Sayed, Diffusion adaptation over networks, in: R. Chellappa, S. Theodoridis (Eds.), Academic Press Library in Signal Processing, Vol. 3, Academic Press, 2014, pp. 323-454
[85] A.H. Sayed, S.-Y. Tu, X. Zhao, Z.J. Towfic, Diffusion strategies for adaptation and learning over networks, IEEE Signal Process. Mag. 30(3) (2013) 155-171.
[86] A.H. Sayed, Adaptive networks, Proc. IEEE 102(4) (2014) 460-497.
[87] A.H. Sayed, Adaptation, learning, and optimization over networks, Foundat. Trends Machine Learn. 7(4-5) (2014) 311-801.
[88] I.D. Schizas, G. Mateos, G.B. Giannakis, Distributed LMS for consensus-based in-network adaptive processing, IEEE Trans. Signal Process. 57(6) (2009) 2365-2382.
[89] D.I. Shuman, S.K. Narang, A. Ortega, P. Vandergheyrst, The emerging field of signal processing on graphs, IEEE Signal Process. Mag. 30(3) (2013) 83-98.
[90] D.T. Slock, On the convergence behavior of the LMS and normalized LMS algorithms, IEEE Trans. signal Process. 40(9) (1993) 2811-2825.
[91] V. Solo, X. Kong, Adaptive Signal Processing Algorithms: Stability and Performance, Prentice Hall, Upper Saddle River, NJ, 1995.
[92] V. Solo, The stability of LMS, IEEE Trans. Signal Process. 45(12) (1997) 3017-3026.
[93] S.S. Stankovic, M.S. Stankovic, D.M. Stipanovic, Decentralized parameter estimation by consensus based stochastic approximation, IEEE Trans. Automat. Control 56(3) (2011) 531-543.
[94] S. Theodoridis, K. Koutroumbas, Pattern Recognition, fourth ed., Academic Press, 2009.
[95] J.N. Tsitsiklis, Problems in Decentralized Decision Making and Computation, Ph.D. Thesis, MIT, 1984.
[96] Y.Z. Tsypkin, Adaptation and Learning in Automatic Systems, Academic Press, New York, 1971.
[97] S.-Y. Tu, A.H. Sayed, Foraging behavior of fish schools via diffusion adaptation, in: Proceedings Cognitive Information Processing, CIP, 2010, pp. 63-68.
[98] S.-Y. Tu, A.H. Sayed, Mobile adaptive networks, IEEE J. Selected Topics Signal Process. 5(4) (2011) 649-664.
[99] S.-Y. Tu, A.H. Sayed, Diffusion strategies outperform consensus strategies for distributed estimation over adaptive networks, IEEE Trans. Signal Process. 60(12) (2012) 6217-6234.
[100] K. Vikram, V.H. Poor, Social learning and Bayesian games in multiagent signal processing, IEEE Signal Process. Mag. 30(3) (2013) 43-57.
[101] E. Walach, B. Widrow, The least mean fourth (LMF) adaptive algorithm and its family, IEEE Trans. Informat. Theory 30(2) (1984) 275-283.
[102] B. Widrow, M.E. Hoff, Adaptive switching circuits, IRE Part 4, IRE WESCON Convention Record, 1960, pp. 96-104.
[103] B. Widrow, S.D. Stearns, Adaptive Signal Processing, Prentice Hall, Englewood Cliffs, 1985.
[104] J.-J. Xiao, A. Ribeiro, Z.-Q. Luo, G.B. Giannakis, Distributed compression-estimation using wireless networks, IEEE Signal Process. Mag. 23 (4) (2006) 27–41.
[105] N.R. Yousef, A.H. Sayed, A unified approach to the steady-state and tracking analysis of adaptive filters, IEEE Trans. Signal Process. 49(2) (2001) 314-324.
[106] N.R. Yousef, A.H. Sayed, Ability of adaptive filters to track carrier offsets and random channel nonstationarities, IEEE Trans. Signal Process. 50(7) (2002) 1533-1544.
[107] F. Zhao, J. Lin, L. Guibas, J. Reich, Collaborative signal and information processing, Proc. IEEE 91(8) (2003) 1199-1209.