Contents

[1] Optional reading.

Chapter 3

RLS and LMS

In chapter 2 it was mentioned that Wiener filter theory and least squares estimation theory form the basis for the two common adaptive filtering procedures, namely the LMS algorithm and the RLS algorithm. These two algorithms are derived and briefly analyzed in this chapter.
We reverse the order, and focus on the RLS procedure first, which is derived from least squares estimation theory. The LMS algorithm may then be viewed as a pruned version of the RLS algorithm. This is in contrast to the usual approach in which the LMS algorithm is derived from Wiener filter theory, based on steepest-descent optimization and instantaneous estimates of the process statistics.
For RLS, only the basic principles are described. Practical algorithms are derived in later chapters.

3.1 RLS estimation

In chapter 2, least squares estimation was explained from a batch-mode processing perspective, where a complete batch of data samples is given, and then used to compute an optimal filter. Many of the applications given in Chapter 1, however, require real-time processing, where the signal processing device has to keep pace with sampling devices that produce new data in each time step. Hence, what is needed is a recursive parameter estimation procedure, where new data samples -as they are fed in- are used to re-compute or 'update' of all the stored information, and then to produce the corresponding output.

We first focus on a recursive version of least squares estimation, referred to as RLS (for '*recursive least squares*') estimation. Only the basic idea, and corresponding formulas will be given. These formulas may be used for theoretical purposes, or so-to-speak for 'infinite precision implementation'. For finite precision implementation, alternative so-called *square-root* algorithms should be used instead, which are better behaved numerically. These will be derived in Chapter 4.

We recall the least squares criterion

$$J_{LS}(\mathbf{w}) \quad = \quad \sum_{k=1}^{L} e_k^2$$

$$= \sum_{k=1}^{L} (d_k - \mathbf{u}_k^T \mathbf{w})^2$$

where L refers to the length of the data set. To obtain a more common notation further on, we change notation as follows

$$\begin{aligned} J_{LS}(\mathbf{w}) &= \sum_{l=1}^{k} e_l^2 \\ &= \sum_{l=1}^{k} (d_l - \mathbf{u}_l^T \mathbf{w})^2. \end{aligned}$$

This is now viewed as the LS criterion *at time k*. In RLS estimation, the main idea is to compute at each time k an optimal filter from all past data. Furthermore, the filter at time $k+1$ is computed from the filter at time k, instead of 'from scratch' which saves on computation. We now outline how this can be done.

Assume that we have the optimal solution at time k

$$\begin{aligned} \mathbf{w}_{LS}(k) &= \arg\{\min_{\mathbf{w}} \sum_{l=1}^{k} (d_l - \mathbf{u}_l^T \mathbf{w})^2\} \\ &= \underbrace{[U(k)^T U(k)]^{-1}}_{[\mathbb{X}_{uu}(k)]^{-1}} \underbrace{[U(k)^T \mathbf{d}(k)]}_{\mathbb{X}_{du}(k)} \end{aligned}$$

with

$$\mathbf{d}(k) = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \end{bmatrix} \qquad U(k) = \begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ \vdots \\ \mathbf{u}_k^T \end{bmatrix}.$$

Computing $\mathbf{w}_{LS}(k+1)$ 'from scratch' would come down to computing

$$\begin{aligned} \mathbf{w}_{LS}(k+1) &= \arg\{\min_{\mathbf{w}} \sum_{l=1}^{k+1} (d_l - \mathbf{u}_l^T \mathbf{w})^2\} \\ &= \underbrace{[U(k+1)^T U(k+1)]^{-1}}_{[\mathbb{X}_{uu}(k+1)]^{-1}} \underbrace{[U(k+1)^T \mathbf{d}(k+1)]}_{\mathbb{X}_{du}(k+1)} \end{aligned}$$

with

$$\mathbf{d}(k+1) = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{k+1} \end{bmatrix} \qquad U(k+1) = \begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ \vdots \\ \mathbf{u}_{k+1}^T \end{bmatrix}.$$

Recall that the matrix inversion $[\mathbb{X}_{uu}(k+1)]^{-1}$ requires $\mathcal{O}(N^3)$ arithmetic operations[1]. This could be a lot of computation do each time some new data arrives. What is sought is an $\mathcal{O}(N^2)$ scheme that computes $\mathbf{w}_{LS}(k+1)$ directly from $\mathbf{w}_{LS}(k)$.

[1] $\mathbb{X}_{uu}(\cdot)$ is an $N \times N$ matrix.

Observe that

$$\mathbb{X}_{uu}(k+1) = \mathbb{X}_{uu}(k) + \mathbf{u}_{k+1}\mathbf{u}_{k+1}^T$$

and

$$\mathbb{X}_{du}(k+1) = \mathbb{X}_{du}(k) + \mathbf{u}_{k+1}d_{k+1}.$$

The *matrix inversion lemma*[2] then states that

$$\boxed{[\mathbb{X}_{uu}(k+1)]^{-1} = [\mathbb{X}_{uu}(k)]^{-1} - \frac{[\mathbb{X}_{uu}(k)]^{-1}\mathbf{u}_{k+1}\mathbf{u}_{k+1}^T[\mathbb{X}_{uu}(k)]^{-1}}{1+\mathbf{u}_{k+1}^T[\mathbb{X}_{uu}(k)]^{-1}\mathbf{u}_{k+1}}}$$

which allows us to compute $[\mathbb{X}_{uu}(k+1)]^{-1}$ from $[\mathbb{X}_{uu}(k)]^{-1}$ in only $\mathcal{O}(N^2)$ arithmetic steps. By making use of these equations, it is readily shown that

$$\boxed{\mathbf{w}_{LS}(k+1) = \mathbf{w}_{LS}(k) + \underbrace{[\mathbb{X}_{uu}(k+1)]^{-1}\mathbf{u}_{k+1}}_{\text{Kalman gain } \mathbf{k}_{k+1}} \cdot \underbrace{(d_{k+1} - \mathbf{u}_{k+1}^T\mathbf{w}_{LS}(k))}_{\text{a priori residual } \varepsilon_{k+1}}}.$$

The above framed formulas constitute the so-called **standard recursive least squares algorithm**. Note that $\mathbf{w}_{LS}(k+1)$ is computed from $\mathbf{w}_{LS}(k)$ with only $\mathcal{O}(N^2)$ arithmetic operations. This is indeed what we aimed at.

We should reiterate that the above formulas should only be used for theoretical purposes. It has been shown theoretically and experimentally that the standard recursive least squares algorithm – when it is run for a 'long' period of time – suffers from numerical round-off error build-up[3]. For practical computations, *square-root algorithms* should be used, which are derived in Chapter 4.

The updating formula for $\mathbf{w}_{LS}$ shows that $\mathbf{w}_{LS}(k+1)$ is equal to $\mathbf{w}_{LS}(k)$ up to a correction term with two ingredients. The so-called *Kalman gain vector* $\mathbf{k}_{k+1}$ gives the direction in which $\mathbf{w}_{LS}$ is modified and the *a priori residual* ε_{k+1} which controls the amount by which it is changed. The a priori residual ε_{k+1} is the prediction error for the data at time $k+1$, when the weight vector $\mathbf{w}_{LS}(k)$ from the previous time step is employed. Note that if it is zero, $\mathbf{w}_{LS}(k)$ fits the data at time $k+1$ and so no adaptation is necessary, *i.e.* $\mathbf{w}_{LS}(k+1) = \mathbf{w}_{LS}(k)$. A useful alternative formula for the Kalman gain vector which is readily derived and will be used later on is

$$\mathbf{k}_{k+1} = [\mathbb{X}_{uu}(k+1)]^{-1}\mathbf{u}_{k+1} = \frac{[\mathbb{X}_{uu}(k)]^{-1}}{1+\mathbf{u}_{k+1}^T[\mathbb{X}_{uu}(k)]^{-1}\mathbf{u}_{k+1}}\mathbf{u}_{k+1}.$$

Finally, the above recursive least squares algorithm obviously has to be initialized properly. We may do this by first applying batch-mode processing to the first portion of the data sequence, *i.e.* $\mathbf{u}_1, \ldots, \mathbf{u}_N$ and $d_1, \ldots, d_N$. A simpler procedure consists in starting from an initial estimate

$$\mathbb{X}_{uu}(0) = \alpha \cdot I \qquad \mathbf{w}_{LS}(0) = \mathbf{w}^o$$

[2] $(A+BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1}+DA^{-1}B)^{-1}DA^{-1}$.

[3] It has been shown that unstability results from the loss of positive definiteness of $\mathbb{X}_{uu}(k)$, due to round-off errors in the updating formula.

where α is a (small) constant. It can be verified that this changes the cost function into

$$J_{LS}(\mathbf{w}(k)) = \alpha^2 \cdot \|\mathbf{w}-\mathbf{w}^o\|_2^2 + \sum_{l=1}^{k} e_l^2.$$

The first term, in effect, penalises movement away from some nominal value $\mathbf{w}^o$. However with time (increasing k) this first term is insignificant compared with the second one and can be ignored.

3.2 Windowed RLS for time-varying systems

In this section it is shown how the RLS algorithm of the previous section can be modified so that it can deal with time-varying situations too. The RLS algorithm is supplemented with a windowing or weighting technique, which effectively places a (sliding) window over the given data. It is assumed that the system is sufficiently time-invariant within the window, so that time averaging may be applied.

We should reiterate that the RLS algorithms given here are standard RLS type algorithms which are known to be numerically unstable. In practice, square-root versions of these algorithms should be used (to be derived in Chapter 4).

In adaptive filtering, ‘time-varying’ may refer to either nonstationarity of the stochastic processes involved, or to true time-variation in the system (plant, channel, *etc.*) itself. In an acoustic echo cancellation problem, for example, the first form of time-variation would correspond to the nonstationarity of the speaker signals; the latter form of time-variation refers to changes in the acoustic channel (say, whenever an object is moved). In practice, the second form is really the only relevant form of time-variation as far as the LS/RLS procedures are concerned. By virtue of the deterministic cost function use here, the nonstationarity of the signal statistics is unimportant and indeed effectively ignored in the LS/RLS setting. This is not the case howvere in the stochastic setting, where nonstationarity strictly disallows any form of time averaging (even if, for example, in the acoustic echo example, the system/channel itself happens to be time-invariant). The stochastic setting is therefore – sometimes – less appropriate.

3.2.1 Sliding window RLS

For time-varying systems, the basic assumption is that the system is sufficienly time-invariant within a short time segment, so that time averaging in this segment is indeed a sensible thing to do. This is, admittedly, a weak piece of reasoning, but given only one sequence of data samples (which may be viewed as one realization of the stochastic processes involved), an obvious and natural way out (for the time being).
Restricting the averaging to a suitable length window, the least squares criterion at time k may be modified into

$$J_{LS}(\mathbf{w}(k)) \quad = \quad \sum_{l=k-L+1}^{k} e_l^2$$

where L is the length of the data window over which e_l^2 is averaged. The corresponding RLS algorithm is referred to as *sliding window RLS*, or *RLS with finite memory*, and is easily derived from the standard RLS algorithm.

Assume that we have the optimal solution at time k

$$\begin{aligned}\mathbf{w}_{LS}(k) &= \arg\{\min_{\mathbf{w}} \sum_{l=k-L+1}^{k} (d_l - \mathbf{u}_l^T \mathbf{w})^2\} \\ &= \underbrace{[U(k)^T U(k)]^{-1}}_{[\mathbb{X}_{uu}(k)]^{-1}} \underbrace{[U(k)^T \mathbf{d}(k)]}_{\mathbb{X}_{du}(k)}\end{aligned}$$

with

$$\mathbf{d}(k) = \begin{bmatrix} d_{k-L+1} \\ d_{k-L+2} \\ \vdots \\ d_k \end{bmatrix} \qquad U(k) = \begin{bmatrix} \mathbf{u}_{k-L+1}^T \\ \mathbf{u}_{k-L+2}^T \\ \vdots \\ \mathbf{u}_k^T \end{bmatrix}.$$

The aim is again to compute $\mathbf{w}_{LS}(k+1)$ with only $\mathcal{O}(N^2)$ operations, where

$$\begin{aligned}\mathbf{w}_{LS}(k+1) &= \arg\{\min_{\mathbf{w}} \sum_{k=k-L+2}^{k+1} (d_l - \mathbf{u}_l^T \mathbf{w})^2\} \\ &= \underbrace{[U(k+1)^T U(k+1)]^{-1}}_{[\mathbb{X}_{uu}(k+1)]^{-1}} \underbrace{[U(k+1)^T \mathbf{d}(k+1)]}_{\mathbb{X}_{du}(k+1)}\end{aligned}$$

and

$$\mathbf{d}(k+1) = \begin{bmatrix} d_{k-L+2} \\ d_{k-L+3} \\ \vdots \\ d_{k+1} \end{bmatrix} \qquad U(k+1) = \begin{bmatrix} \mathbf{u}_{k-L+2}^T \\ \mathbf{u}_{k-L+3}^T \\ \vdots \\ \mathbf{u}_{k+1}^T \end{bmatrix}.$$

It is observed that

$$\mathbb{X}_{uu}(k+1) = \underbrace{\mathbb{X}_{uu}(k) - \mathbf{u}_{k-L+1}\mathbf{u}_{k-L+1}^T}_{\mathbb{X}_{uu}^{k|k+1}} + \mathbf{u}_{k+1}\mathbf{u}_{k+1}^T$$

and

$$\mathbb{X}_{du}(k+1) = \underbrace{\mathbb{X}_{du}(k) - \mathbf{u}_{k-L+1} d_{k-L+1}}_{\mathbb{X}_{du}^{k|k+1}} + \mathbf{u}_{k+1} d_{k+1}.$$

Note that the update formulas are similar to the standard RLS procedure except that, along with the term that is to be added (*update), there is a term that has to be subtracted (downdate). Not surprisingly, the complete time update procedure turns out to consist of an update with* $(\mathbf{u}_{k+1}, d_{k+1})$ *following a downdate with* $(\mathbf{u}_{k-L+1}, d_{k-L+1})$.

The formulas for the downdate are similar to those for the update (standard RLS). One can easily prove that

$$[\mathbb{X}_{uu}^{k|k+1}]^{-1} = [\mathbb{X}_{uu}(k)]^{-1} + \frac{[\mathbb{X}_{uu}(k)]^{-1}\mathbf{u}_{k-L+1}\mathbf{u}_{k-L+1}^T[\mathbb{X}_{uu}(k)]^{-1}}{1-\mathbf{u}_{k-L+1}^T[\mathbb{X}_{uu}(k)]^{-1}\mathbf{u}_{k-L+1}}$$

$$\mathbf{w}_{LS}^{k|k+1} = \mathbf{w}_{LS}(k) - [\mathbb{X}_{uu}^{k|k+1}]^{-1}\mathbf{u}_{k-L+1} \cdot (d_{k-L+1} - \mathbf{u}_{k-L+1}^T \mathbf{w}_{LS}(k)) \ .$$

The formulas for the update are those of the standard RLS algorithm (with appropriate notational changes)

$$[\mathbb{X}_{uu}(k+1)]^{-1} = [\mathbb{X}_{uu}^{k|k+1}]^{-1} - \frac{[\mathbb{X}_{uu}^{k|k+1}]^{-1}\mathbf{u}_{k+1}\mathbf{u}_{k+1}^T[\mathbb{X}_{uu}^{k|k+1}]^{-1}}{1+\mathbf{u}_{k+1}^T[\mathbb{X}_{uu}^{k|k+1}]^{-1}\mathbf{u}_{k+1}}$$

$$\mathbf{w}_{LS}(k+1) = \mathbf{w}_{LS}^{k|k+1} + [\mathbb{X}_{uu}(k+1)]^{-1}\mathbf{u}_{k+1} \cdot (d_{k+1} - \mathbf{u}_{k+1}^T \mathbf{w}_{LS}^{k|k+1}) \ .$$

The above four formulas constitute a sliding window RLS algorithm.

Remark 2.10. *It should be noted rightaway that downdating (unlike updating) is often a 'dangerous step' from a numerical point of view. The minus sign in the denominators is indicative of this: due to round-off errors the net effect of updating with $(\mathbf{u}_{k+1}, d_{k+1})$ at time $k+1$ and downdating with $(\mathbf{u}_{k+1}, d_{k+1})$ at time $k+L+1$, is not going to be zero, so one can expect at least a linear error build-up. In conclusion, the above method, as it stands (even in square-root form), is not really useful. Remedies will be treated later.*

Remark 2.11. *In a stochastic analysis (see section 2.3.3) the formula for the MSE in the case of sliding window RLS, is obviously given as*

$$J_{MSE}(\mathbf{w}_{LS}(k)) = J_{MSE}(\mathbf{w}_{WF}(k)) + \underbrace{J_{MSE}(\mathbf{w}_{WF}(k)) \cdot \frac{N}{L}}_{J_{ex}(k)} .$$

where L is the window length. However, in a **time-varying environment***, this only gives the excess MSE due to 'weight vector noise' (as a result of noise in the estimated process statistics). In addition, there will be excess MSE due to 'weight vector lag', which results from the fact that the algorithm computes some average optimal weight vector over the time window, which will be different from the instantaneous optimal weight vector ($\mathbf{w}_{WF}(k) = \bar{\mathbb{X}}_{uu}(k)^{-1}\bar{\mathbb{X}}_{du}(k)$). This leads to*

$$J_{MSE}(\mathbf{w}_{LS}(k)) = J_{MSE}(\mathbf{w}_{WF}(k)) + \underbrace{J_{MSE}(\mathbf{w}_{WF}(k)) \cdot \frac{N}{L}}_{J_{ex}^{noise}(k)} + \underbrace{f(L)}_{J_{ex}^{lag}(k)}$$

where J_{ex}^{lag} is expected to be proportional to L, but is obviously difficult to quantify (and so we will not make an attempt). As $J_{ex}^{lag}(k)$ is proportional to L and $J_{ex}^{noise}(k)$ is inversely proportional to L, the optimal choice for L should be such that both terms are equal. However, finding such an optimal L is not an easy task.

3.2.2 Exponentially weighted RLS

By far the most popular alternative to finite memory RLS is exponentially weighted RLS. The least squares criterion at time k is modified to

$$J_{LS}(\mathbf{w}(k)) \quad = \quad \sum_{l=1}^{k} \lambda^{2(k-l)} e_l^2$$

where $0 < \lambda < 1$ is weighting factor or forget factor. The memory is infinite, but 'old' values of e_l^2 are given a small weight, and so do not play a significant role. Roughly speaking, only the latest $\frac{1}{1-\lambda}$ data points are given large weights and this is a 'measure of the memory of the algorithm'.

One can easily verify that

$$\begin{aligned}
\mathbf{w}_{LS}(k) &= arg\{\min_{\mathbf{w}} \sum_{l=1}^{k} \lambda^{2(k-l)}(d_l - \mathbf{u}_l^T\mathbf{w})^2\} \\
&= \underbrace{[U(k)^T U(k)]^{-1}}_{[\mathbb{X}_{uu}(k)]^{-1}} \underbrace{[U(k)^T \mathbf{d}(k)]}_{\mathbb{X}_{du}(k)}
\end{aligned}$$

with

$$\mathbf{d}(k) = \begin{bmatrix} \lambda^{k-1}d_1 \\ \lambda^{k-2}d_2 \\ \vdots \\ \lambda^0 d_k \end{bmatrix} \qquad U(k) = \begin{bmatrix} \lambda^{k-1}\mathbf{u}_1^T \\ \lambda^{k-2}\mathbf{u}_2^T \\ \vdots \\ \lambda^0 \mathbf{u}_k^T \end{bmatrix}.$$

which leads to

$$\mathbb{X}_{uu}(k+1) = \lambda^2 \mathbb{X}_{uu}(k) + \mathbf{u}_{k+1}\mathbf{u}_{k+1}^T$$

and

$$\mathbb{X}_{du}(k+1) = \lambda^2 \mathbb{X}_{du}(k) + \mathbf{u}_{k+1}d_{k+1}.$$

and hence

$$\boxed{[\mathbb{X}_{uu}^{k|k+1}]^{-1} = \frac{1}{\lambda^2}[\mathbb{X}_{uu}(k)]^{-1}}$$

$$\boxed{[\mathbb{X}_{uu}(k+1)]^{-1} = [\mathbb{X}_{uu}^{k|k+1}]^{-1} - \frac{[\mathbb{X}_{uu}^{k|k+1}]^{-1}\mathbf{u}_{k+1}\mathbf{u}_{k+1}^T[\mathbb{X}_{uu}^{k|k+1}]^{-1}}{1+\mathbf{u}_{k+1}^T[\mathbb{X}_{uu}^{k|k+1}]^{-1}\mathbf{u}_{k+1}}}$$

$$\boxed{\mathbf{w}_{LS}(k+1) = \mathbf{w}_{LS}(k) + [\mathbb{X}_{uu}(k+1)]^{-1}\mathbf{u}_{k+1} \cdot (d_{k+1} - \mathbf{u}_{k+1}^T \mathbf{w}_{LS}(k))}.$$

The three framed formulas constitute the exponentially weighted RLS algorithm. It is seen that the exponential weighting introduces only one simple additional step, i.e. the weighting of the covariance matrix.

Remark 2.12. *Exponential weighting not only leads to an extremely simple RLS algorithm, but it is also crucial in view of numerical stability. Even though the standard RLS forms are known to be numerically unstable, the corresponding square-root forms (to be derived Chapter 4), are indeed numerically stable and owe their stability to the weighting scheme. Roughly speaking, exponential weighting indeed not only wipes out 'old data', but also 'old errors'.*
For this reason, some form of exponential weighting is usually also included in

the sliding window RLS algorithms[4]*, as well as in the directionally weighted RLS scheme (see next section).*

Remark 2.13. *In a stochastic analysis (see section 2.3.1), one proves that (for small values of $1-\lambda$) the formula for the MSE is given as*

$$J_{MSE}(\mathbf{w}_{LS}(k)) = J_{MSE}(\mathbf{w}_{WF}(k)) + \underbrace{J_{MSE}(\mathbf{w}_{WF}(k)) \cdot \frac{N(1-\lambda)}{2}}_{J_{ex}^{noise}(k)} + \underbrace{f(\frac{2}{1-\lambda})}_{J_{ex}^{lag}(k)}$$

where L has been replaced by the 'effective' window length $\frac{2}{1-\lambda}$. Again J_{ex}^{lag} is expected to be proportional to $\frac{2}{1-\lambda}$, but difficult to quantify (and so we will not make an attempt), and hence choosing an optimal λ is not an easy task.

3.2.3 Directionally weighted RLS[5]

The exponentially weighted RLS may give rise to numerical trouble when the incoming data are not 'persistently exciting'. This refers to the situation where the input signal does not contain the information we are trying to find. In the extreme case where the data are all-zero, e.g. from time k on, it is seen that all information is gradually wiped out, while no new information is added, so that $\mathbb{X}_{uu}^{-1}$ blows up. An appealing alternative strategy consists in exponentionally discounting old data only when there is new incoming information (with vector data this can correspond to certain subspaces or 'directions'). This prevents the matrix $\mathbb{X}_{uu}^{-1}$ from blowing up when the excitation is not uniform over the parameter space (persistent).

We will not try and give all the theoretical background of the resulting so-called directionally weighted RLS algorithm, only indicate how the formulas come about. The algorithm is as follows and differs from the exponentially weighted RLS algorithm only in the first formula :

$$[\mathbb{X}_{uu}^{k|k+1}]^{-1} = [\mathbb{X}_{uu}(k)]^{-1} - \frac{1-\lambda^2}{\lambda^2} \cdot \frac{[\mathbb{X}_{uu}(k)]^{-1}\mathbf{u}_k\mathbf{u}_k^T[\mathbb{X}_{uu}(k)]^{-1}}{\mathbf{u}_k^T[\mathbb{X}_{uu}(k)]^{-1}\mathbf{u}_k} .$$

$$[\mathbb{X}_{uu}(k+1)]^{-1} = [\mathbb{X}_{uu}^{k|k+1}]^{-1} - \frac{[\mathbb{X}_{uu}^{k|k+1}]^{-1}\mathbf{u}_{k+1}\mathbf{u}_{k+1}^T[\mathbb{X}_{uu}^{k|k+1}]^{-1}}{1+\mathbf{u}_{k+1}^T[\mathbb{X}_{uu}^{k|k+1}]^{-1}\mathbf{u}_{k+1}}$$

$$\mathbf{w}_{LS}(k+1) = \mathbf{w}_{LS}(k) + [\mathbb{X}_{uu}(k+1)]^{-1}\mathbf{u}_{k+1} \cdot (d_{k+1} - \mathbf{u}_{k+1}^T\mathbf{w}_{LS}(k)) .$$

The first formula may be explained as follows. First, it is seen (see second formula) that $[\mathbb{X}_{uu}(k+1)]^{-1}$ is derived from the intermediate $[\mathbb{X}_{uu}^{k|k+1}]^{-1}$ by a rank-one update, which means that the corresponding (multidimensional) quadratic cost function is affected

[4]The U's are are then exponentially weighted, but still truncated matrices (*i.e.* with always L rows). The basic updating formula is derived from

$$\mathbb{X}_{uu}(k+1) = \underbrace{\lambda^2\mathbb{X}_{uu}(k) - \lambda^{2\cdot L}\mathbf{u}_{k-L+1}\mathbf{u}_{k-L+1}^T}_{\mathbb{X}_{uu}^{k|k+1}} + \mathbf{u}_{k+1}\mathbf{u}_{k+1}^T$$

[5]Optional reading.

only in one 'direction', namely $[\mathbb{X}_{uu}^{k|k+1}]^{-1}\mathbf{u}_{k+1}$. *In order to apply the weighting only to that particular direction, one can calculate the dyad corresponding to that direction, remove it from* $\mathbb{X}_{uu}(k+1)$*, weigh the dyad and then include it again, i.e.*

$$\mathbb{X}_{uu}^{k+1|k+2} = \underbrace{\overbrace{\mathbb{X}_{uu}(k+1)}^{rank=N} - \frac{\mathbf{u}_{k+1}\mathbf{u}_{k+1}^T}{\mathbf{u}_{k+1}^T[\mathbb{X}_{uu}(k+1)]^{-1}\mathbf{u}_{k+1}}}_{rank=N-1} + \lambda^2 \frac{\mathbf{u}_{k+1}\mathbf{u}_{k+1}^T}{\mathbf{u}_{k+1}^T[\mathbb{X}_{uu}(k+1)]^{-1}\mathbf{u}_{k+1}}$$

The underbraced matrix represents a positive semidefinite (rank $N-1$*) quadratic form, which is zero in the* $[\mathbb{X}_{uu}^{k|k+1}]^{-1}\mathbf{u}_{k+1}$ *direction, i.e.*

$$\mathbf{u}_{k+1}^T[\mathbb{X}_{uu}^{k|k+1}]^{-T} \cdot \left[\mathbb{X}_{uu}(k+1) - \frac{\mathbf{u}_{k+1}\mathbf{u}_{k+1}^T}{\mathbf{u}_{k+1}^T[\mathbb{X}_{uu}(k+1)]^{-1}\mathbf{u}_{k+1}} \right] \cdot [\mathbb{X}_{uu}^{k|k+1}]^{-1}\mathbf{u}_{k+1} = 0$$

which means that the relevant dyad is indeed extracted. The dyad is then multiplied by λ^2 *and added again. By changing the time indices, one obtains*

$$[\mathbb{X}_{uu}^{k|k+1}] = [\mathbb{X}_{uu}(k)] - (1-\lambda^2) \cdot \frac{\mathbf{u}_k\mathbf{u}_k^T}{\mathbf{u}_k^T[\mathbb{X}_{uu}(k)]^{-1}\mathbf{u}_k}.$$

By applying the matrix inversion lemma one obtains the $\mathbb{X}_{uu}^{k|k+1}$ *-formula of the given algorithm.*

Remark 2.14. *Some additional exponential weighting is sometimes included, leading to*

$$[\mathbb{X}_{uu}^{k|k+1}]^{-1} = \frac{1}{\tilde{\lambda}^2}[\mathbb{X}_{uu}(k)]^{-1} - \frac{1-\lambda^2}{\lambda^2} \cdot \frac{[\mathbb{X}_{uu}(k)]^{-1}\mathbf{u}_k\mathbf{u}_k^T[\mathbb{X}_{uu}(k)]^{-1}}{\mathbf{u}_k^T[\mathbb{X}_{uu}(k)]^{-1}\mathbf{u}_k}$$

where $\tilde{\lambda}$ *is a second forget factor (mostly* $0 < \lambda < \tilde{\lambda} < 1$*).*

3.3 LMS and NLMS

We have seen that for recursive processing the RLS algorithm achieves an $\mathcal{O}(N)$ *reduction in the number of arithmetic operations per time update, which is done by invoking the matrix inversion lemma. The computational complexity is reduced to* $\mathcal{O}(N^2)$ *operations per time update. In this section, we focus on the family of* **least mean squares (LMS)** *algorithms, which have an* $\mathcal{O}(N)$ *computational complexity, i.e. achieve another* $\mathcal{O}(N)$ *gain over RLS. The price to be paid is that these cheaper algorithms give only approximate solutions at each time instant.*

The original LMS algorithm is a simple one line updating algorithm, which may be derived from the RLS algorithm by ignoring the covariance matrix update and setting $[\mathbb{X}_{uu}(k+1)]^{-1} \Leftarrow I$ *in the updating formula for* $\mathbf{w}$*, leading to*

$$\mathbf{w}_{LMS}(k+1) = \mathbf{w}_{LMS}(k) + \mu \cdot \mathbf{u}_{k+1} \cdot (d_{k+1} - \mathbf{u}_{k+1}^T\mathbf{w}_{LMS}(k)).$$

Here μ *is the step-size parameter, the significance of which will be explained later when we will also derive some bounds on its value.*

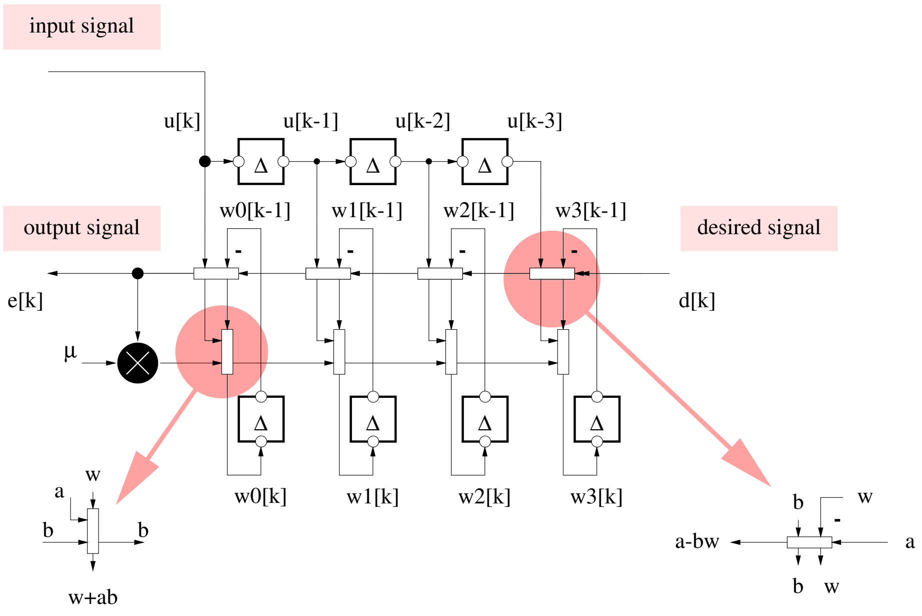

Figure 3.1: LMS signal flow graph

Note that this one line algorithm is remarkably simple, requiring not more than roughly 2N multiply-add operations per time update. A signal flow graph is straightforwardly derived and shown in **Figure 3.1** *(see also Figure 1.21). In contrast, the convergence/stability analysis of the LMS algorithm is, however, remarkably complex (see below).*

A popular alternative to the standard LMS algorithm that we will mention rightaway is **normalized LMS (NLMS)**, *which may be derived in a similar fashion by setting* $[\mathbb{X}_{uu}(k)]^{-1} \Leftarrow I$, *leading to*[6]

$$\mathbf{w}_{NLMS}(k+1) = \mathbf{w}_{NLMS}(k) + \frac{\bar{\mu}}{1+\mathbf{u}_{k+1}^T\mathbf{u}_{k+1}} \cdot \mathbf{u}_{k+1} \cdot (d_{k+1} - \mathbf{u}_{k+1}^T \mathbf{w}_{NLMS}(k)).$$

A more general form of NLMS is

$$\mathbf{w}_{NLMS}(k+1) = \mathbf{w}_{NLMS}(k) + \frac{\bar{\mu}}{\alpha^2+\mathbf{u}_{k+1}^T\mathbf{u}_{k+1}} \cdot \mathbf{u}_{k+1} \cdot (d_{k+1} - \mathbf{u}_{k+1}^T \mathbf{w}_{NLMS}(k)) .$$

where α *is a constant. Clearly, NLMS may be viewed as LMS with a time-varying step-size* $\mu = \frac{\bar{\mu}}{\alpha+\mathbf{u}_{k+1}^T\mathbf{u}_{k+1}}$.

As already mentioned the convergence/stability analysis of the LMS algorithm turns out to be a remarkably difficult mathematical task. A completely general analysis seems to be lacking even. The LMS algorithm is usually analyzed in the (Wiener filter type) stochastic framework, where simplifying independence assumptions can be made. Again, such assumptions are virtually always false in practice, however the analyses do provide results which are found to be in agreement with experiments and computer simulations, and hence seem to provide reliable design guidelines.

We will not make an attempt to reproduce such LMS analyses, only indicate some major results. As already mentioned, the analysis is mostly done in a purely stochastic

[6] By employing the alternative formula for the Kalman gain vector.

framework. It is therefore instructive to first indicate how LMS may be derived from an iterative steepest-descent algorithm, applied to the Wiener filtering problem. This is explained first.

3.3.1 Steepest-descent algorithm

Recall that the MSE criterion is given as

$$J_{MSE}(\mathbf{w}) = \mathcal{E}\{e_k^2\}$$

and that the solution may be obtained by solving the Wiener-Hopf equations

$$\underbrace{\mathcal{E}\{\mathbf{u}_k\mathbf{u}_k^T\}}_{\bar{\mathbb{X}}_{uu}}\mathbf{w}_{WF} = \underbrace{\mathcal{E}\{\mathbf{u}_k d_k\}}_{\bar{\mathbb{X}}_{du}}.$$

(so we are back to the 'given statistics' problem for a while). Instead of solving the set of equations by means of a direct method (such as, e.g., Gauss elimination), one can apply an iterative procedure. The method of steepest descent begins with picking an initial value $\mathbf{w}(0)$, and then iterates as follows [7].

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\mu}{2}\cdot\left[\ \frac{-\partial J_{MSE}(\mathbf{w})}{\partial\mathbf{w}}\ \right]_{\mathbf{w}=\mathbf{w}(n)}$$

i.e.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\cdot(\bar{\mathbb{X}}_{du} - \bar{\mathbb{X}}_{uu}\mathbf{w}(n))$$

So the 'next guess' $\mathbf{w}(n+1)$ is equal to the 'present guess' $\mathbf{w}(n)$ plus a correction in the steepest-descent direction, which is known to be opposite to that of the gradient vector in $\mathbf{w}(n)$. Again, μ is the step-size parameter, that tells how far one moves in the steepest-descent direction. Moving too far in that direction might actually overshoot the minimum and result in instability. Bounds on μ will be derived further on.
Before moving on to LMS, we give a few properties of this steepest-descent algorithm in the following remarks. We should reiterate that we are back to the 'given statistics' case for while, where we assume that the process statistics (i.e. the $\bar{\mathbb{X}}_{uu}$ and $\bar{\mathbb{X}}_{du}$) are given, and that n is the iteration index (rather than a time index) in the iterative search for $\mathbf{w}_{WF} = \bar{\mathbb{X}}_{uu}^{-1}\bar{\mathbb{X}}_{du}$.

Remark 2.15. *With $\bar{\mathbb{X}}_{uu}\mathbf{w}_{WF} = \bar{\mathbb{X}}_{du}$, the above iteration is readily transformed into*

$$\begin{aligned}[\mathbf{w}(n+1) - \mathbf{w}_{WF}] &= (I-\mu\bar{\mathbb{X}}_{uu})\cdot[\mathbf{w}(n) - \mathbf{w}_{WF}] \\ &= (I-\mu\bar{\mathbb{X}}_{uu})^{n+1}\cdot[\mathbf{w}(0) - \mathbf{w}_{WF}]\end{aligned}$$

which means that the weight-error vector gets multiplied by $(I-\mu\bar{\mathbb{X}}_{uu})$ in each iteration. The aim obviously is to have the weight error reduced in each step, which is only the case when the eigenvalues of $(I-\mu\bar{\mathbb{X}}_{uu})$ are strictly smaller than 1 in absolute value. For analysis purposes it is convenient to make use of

[7] The factor $\frac{1}{2}$ is used merely for convenience, *i.e.* to avoid a factor 2 in the second line.

the eigenvalue decomposition of $\bar{\mathbb{X}}_{uu}$. As $\bar{\mathbb{X}}_{uu}$ is symmetric and assumed to be positive definite, one has

$$\bar{\mathbb{X}}_{uu} = Q_{uu}\Lambda_{uu}Q_{uu}^T$$

where Q_{uu} is an orthogonal matrix ($Q_{uu}^T Q_{uu} = I$) with the eigenvectors in its columns, and Λ_{uu} is a diagonal matrix with the corresponding eigenvalues $\Lambda_{uu} = diag\{\lambda_i\}$. This leads to

$$\begin{aligned}[\mathbf{w}(n+1) - \mathbf{w}_{WF}] &= Q_{uu}(I - \mu\Lambda_{uu})^{n+1} Q_{uu}^T \cdot [\mathbf{w}(0) - \mathbf{w}_{WF}] \\ &= Q_{uu} diag\{1 - \mu\lambda_i\}^{n+1} Q_{uu}^T \cdot [\mathbf{w}(0) - \mathbf{w}_{WF}]\end{aligned}$$

It follows that for stability, or convergence of the steepest-descent algorithm, one should have

$$-1 < 1 - \mu\lambda_i < 1 \qquad \forall i.$$

With this, one will have $\mathbf{w}(\infty) = \mathbf{w}_{WF}$. Since the eigenvalues λ_i are all real and positive[8]*, the necessary and sufficient condition for convergence is*

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

where λ_{max} is the largest eigenvalue of $\bar{\mathbb{X}}_{uu}$. This is illustrated in **Figure 3.2**, *where $1 - \mu\lambda_i$ is plotted for various choices of μ. As will become clear in the sequel, the optimal choice for convergence is*

$$\boxed{\mu_{opt} = \frac{2}{\lambda_{max} + \lambda_{min}}}$$

such that

$$(1 - \mu_{opt}\lambda_{min}) = -(1 - \mu_{opt}\lambda_{max}) = \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}}$$

This is also illustrated in Figure 3.2. From the last formula it already follows then whenever $\lambda_{max} \gg \lambda_{min}$, convergence is going to be very slow (μ will be small and hence many iterations will be required).

In the absence of knowledge of the eigenvalues of $\bar{\mathbb{X}}_{uu}$, one often can use an (admittedly sometimes conservative) upper bound for λ_{max}, namely

$$\lambda_{max} < \sum_{i=1}^{N} \lambda_i = trace\{\bar{\mathbb{X}}_{uu}\}$$

In the FIR case one has $trace\{\bar{\mathbb{X}}_{uu}\} = N\bar{x}_{uu}(0) = N \cdot \mathcal{E}\{u_k^2\}$, leading to

$$\boxed{0 < \mu < \frac{2}{N \cdot \mathcal{E}\{u_k^2\}}}\,.$$

[8] This follows from the fact that $\bar{\mathbb{X}}_{uu}$ is symmetric and assumed positive definite.

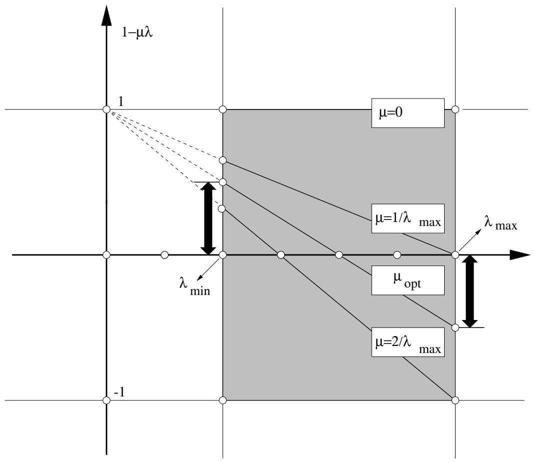

Figure 3.2: Step-size parameters for the steepest-descent algorithm

Remark 2.16. *It is instructive to investigate the transient behavior of both the weight vectors* $\mathbf{w}(n)$ *and the corresponding MSE* $J_{MSE}(\mathbf{w}(n))$*. First*

$$\underbrace{Q_{uu}^T[\mathbf{w}(n)-\mathbf{w}_{WF}]}_{\delta\tilde{\mathbf{w}}(n)} \quad = \quad diag\{1-\mu\lambda_i\}^n \cdot \underbrace{Q_{uu}^T \cdot [\mathbf{w}(0)-\mathbf{w}_{WF}]}_{\delta\tilde{\mathbf{w}}(0)}$$

where $\delta\tilde{\mathbf{w}}(n)$ *represents the weight-error vector expanded in terms of the basis of eigenvectors (i.e. columns of* Q_{uu}*). It is seen that the weight error in the direction of the i-th eigenvector gets multiplied by* $1-\mu\lambda_i$ *in each iteration. The error reduction in this i-th direction is slow if* $1-\mu\lambda_i \approx 1$*. This typically happens when* $\bar{\mathbb{X}}_{uu}$ *is ill-conditioned, i.e. when* $\lambda_{max} \gg \lambda_{min}$*. The large* λ_{max} *then results in a small* μ *so that* $1-\mu\lambda_{min} \approx 1$.

This effect is illustrated in **Figure 3.3***, where the contours of* $J_{MSE}(\mathbf{w})$ *are plotted for a 2-dimensional problem, i.e. with* $\mathbf{w}^T = \begin{bmatrix} w_1 & w_2 \end{bmatrix}$ *(see also Figure 2.6). From*

$$\begin{aligned} J_{MSE}(\mathbf{w}) &= J_{MSE}(\mathbf{w}_{WF}) + (\mathbf{w}-\mathbf{w}_{WF})^T\bar{\mathbb{X}}_{uu}(\mathbf{w}-\mathbf{w}_{WF}) \\ &= J_{MSE}(\mathbf{w}_{WF}) + \delta\tilde{\mathbf{w}}^T\Lambda_{uu}\delta\tilde{\mathbf{w}} \end{aligned}$$

it is seen that the contours (loci of constant $J_{MSE}(\mathbf{w})$*) are ellipsoids (ellipses in 2-D) with principal axes defined by the eigenvectors of* $\bar{\mathbb{X}}_{uu}$*. Note that, as defined above, the major axis corresponds to the* $\mathbf{q}_2$ *direction. The gradient directions are orthogonal to the contours. For a sufficiently small* μ*, (see Figure 3.3.a) convergence is observed in both directions with faster convergence in the* $\mathbf{q}_1$*-direction (eigenvector corresponding to the largest eigenvalue* λ_1*). As a result it is seen that the error in the* $\mathbf{q}_1$*-direction is reduced much faster than the error in the* $\mathbf{q}_2$*-direction, for small* μ*. The effect gets worse, when the contours are more 'stretched', i.e. when* $\lambda_1 \nearrow$ *or* $\lambda_2 \searrow$*. If* μ *is now increased, fewer iterations are required to reach the minimum of the error surface (see Figure 3.3.b). However the weight vector estimates no longer appraoch the optimim smoothly but show a tendancy to oscillate about it. If* μ *is too big, the iteration directly overshoots the minimum in the* $\mathbf{q}_1$*-direction. This is illustrated in Figure 3.3.c and 3.3.d, where* $|1-\mu\lambda_1| = 1$ *and* $|1-\mu\lambda_1| > 1$ *respectively. Note that because* $\lambda_2 < \lambda_1$*, the weight vector in the* $\mathbf{q}_2$ *direction continues to converge smoothly to the optimum.*

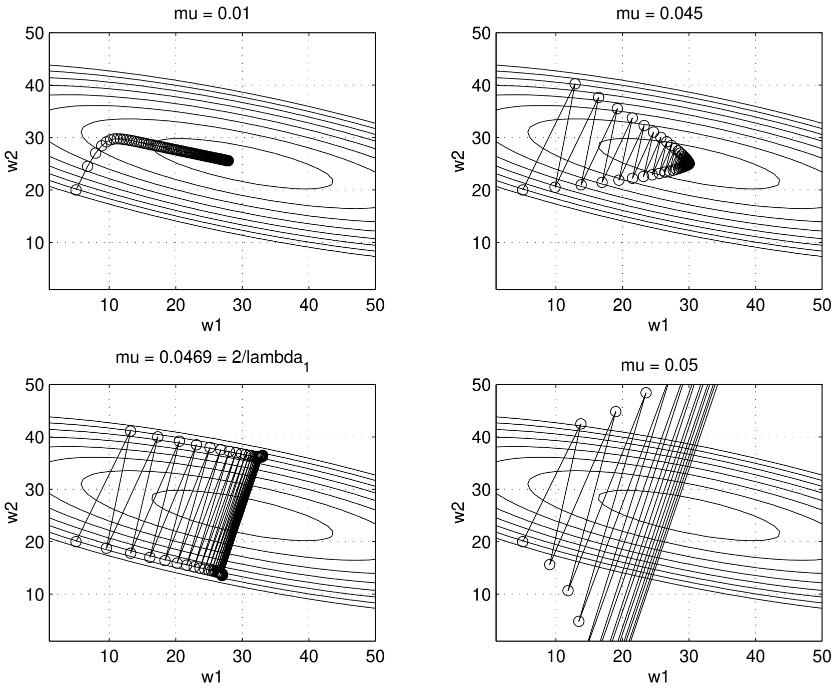

Figure 3.3: Steepest-descent algorithm

Finally, by substituting

$$\delta\tilde{\mathbf{w}}(n) = diag\{1-\mu\lambda_i\}^n \cdot \delta\tilde{\mathbf{w}}(0)$$

in the formula for $J_{MSE}(\mathbf{w})$*, one obtains*

$$J_{MSE}(\mathbf{w}(n)) = J_{MSE}(\mathbf{w}_{WF}) + \sum_{i=1}^{N} \lambda_i (1-\mu\lambda_i)^{2n} \cdot \delta\tilde{w}_i(0)^2$$

where $\delta\tilde{w}_i(0)$ *is the initial error in the direction of the i-th eigenvector. The curve obtained by plotting* $J_{MSE}(\mathbf{w}(n))$ *versus the iteration number n, is called the learning curve. It is seen that it consists of a sum of N exponentials. All these exponentials decay with different 'time constants'. The worst (slowest decay) components are again those that have* $|1-\mu\lambda_i| \approx 1$.

3.3.2 LMS algorithm

We now return to the 'given data' case, where $\bar{\mathbb{X}}_{uu}$ *and* $\bar{\mathbb{X}}_{du}$ *are unknown, and derive a 'given data' version of the steepest-descent algorithm, i.e. the LMS algorithm. We recall the steepest descent iteration*

$$\mathbf{w}(n+1) \quad = \quad \mathbf{w}(n) + \mu \cdot (\mathcal{E}\{\mathbf{u}_k d_k\} - \mathcal{E}\{\mathbf{u}_k \mathbf{u}_k^T\}\mathbf{w}(n)).$$

The first (subtile) modification is that the iteration index n is changed into the time index k, which means that the steepest-descent algorithm is turned into a recursive al-

gorithm with one weight-update (one iteration) per time step. So we have

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu \cdot (\mathcal{E}\{\mathbf{u}_k d_k\} - \mathcal{E}\{\mathbf{u}_k \mathbf{u}_k^T\}\mathbf{w}(k)).$$

The second and last modification is that of leaving out the expectation operators. In effect this means that we substitute an instantaneous estimate for the gradient, sometimes referred to as the stochastic gradient or the noisy gradient. This leads to

$$\boxed{\mathbf{w}_{LMS}(k+1) = \mathbf{w}_{LMS}(k) + \mu \cdot \mathbf{u}_{k+1}(d_{k+1} - \mathbf{u}_{k+1}^T \mathbf{w}_{LMS}(k))}\,.$$

which is indeed the LMS formula we gave at the beginning of the section.

Substituting the stochastic gradient for the true gradient has a tangible impact on the convergence properties. Also, it makes the theoretical convergence analysis dramatically more complex. As already mentioned, we will not make an attempt here to present an LMS analysis, rather only indicate some of the major effects of the substitution. The two issues we will be interested in are the effect in the allowable step-size μ, and the resulting so-called excess MSE or mismatch.

1. *First, it is instructive to rewrite the LMS formula as*

$$\begin{aligned}[\mathbf{w}_{LMS}(k+1) - \mathbf{w}_{WF}] &= (I - \mu \mathbf{u}_{k+1}\mathbf{u}_{k+1}^T) \cdot [\mathbf{w}_{LMS}(k) - \mathbf{w}_{WF}] \\ &\quad + \mu \cdot \mathbf{u}_{k+1}(d_{k+1} - \mathbf{u}_{k+1}^T \mathbf{w}_{WF}).\end{aligned}$$

   One can now take the expected value of both sides of the equation. The last term then vanishes because of the orthogonality principle. This leads to

$$\mathcal{E}\{\mathbf{w}_{LMS}(k+1) - \mathbf{w}_{WF}\} = \mathcal{E}\{(I - \mu \mathbf{u}_{k+1}\mathbf{u}_{k+1}^T) \cdot [\mathbf{w}_{LMS}(k) - \mathbf{w}_{WF}]\}$$

   The right-hand side term is problematic. However it turns out one can make[9] *some independence assumptions that leads to*

$$\begin{aligned}\mathcal{E}\{\mathbf{w}_{LMS}(k+1) - \mathbf{w}_{WF}\} &= (I - \mu \mathcal{E}\{\mathbf{u}_{k+1}\mathbf{u}_{k+1}^T\}) \cdot \mathcal{E}\{\mathbf{w}_{LMS}(k) - \mathbf{w}_{WF}\} \\ &= (I - \mu \bar{\mathbb{X}}_{uu}) \cdot \mathcal{E}\{\mathbf{w}_{LMS}(k) - \mathbf{w}_{WF}\}.\end{aligned}$$

   ¿From this, one concludes that the 'expected behavior' for LMS is the steepest descent behavior. One therefore deduces that the LMS algorithm is **convergent in the mean**, *i.e. that the mean of the $\mathbf{w}_{LMS}(k)$ converges to $\mathbf{w}_{WF}$ as $L \to \infty$, provided that the step-size parameter statisfies the steepest-descent bound*

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

2. *The true difference between LMS and the steepest-descent algorithm is that LMS works with instantaneous estimates of the gradient directions, while the steepest-descent algorithm computes true gradient directions. As a result, even when the LMS algorithm reaches $\mathbf{w}_{WF}$ at some point, it will continue to bounce around, because the estimated gradient directions are never going to be exactly equal to zero. This in turn results in*

$$J_{MSE}(\mathbf{w}_{LMS}(\infty)) > J_{MSE}(\mathbf{w}_{WF}).$$

[9]The details are omitted here.

In Figure 2.6, for example, the average point in the $\{w_1, w_2\}$-plane will be $\mathbf{w}_{WF}$, but the average point on the error-performance surface will be above the $J_{MSE}(\mathbf{w}_{WF})$-level. The **excess MSE** $J_{ex}(\infty)$ *and* **mismatch** $\mathcal{M}$ *are then defined as follows*

$$\begin{aligned} J_{MSE}(\mathbf{w}_{LMS}(\infty)) &= J_{MSE}(\mathbf{w}_{WF}) + J_{ex}(\infty) \\ &= J_{MSE}(\mathbf{w}_{WF}) \cdot (1 + \mathcal{M}). \end{aligned}$$

One proves that (under various technical assumptions) the LMS algorithm is **convergent in the mean square**, *i.e.* $J_{MSE}(\mathbf{w}_{LMS}(k))$ *converges to a finite steady-state value if, and only if, the step-size parameter also satisfies*

$$\sum_{i=1}^{N} \frac{\mu\lambda_i}{2 - \mu\lambda_i} < 1.$$

For small values of μ *(compared to* $\frac{2}{\lambda_{max}}$*) this reduces to*

$$\boxed{0 < \mu < \frac{2}{\sum_{i=1}^{N} \lambda_i}}$$

This bound is clearly much tighter than the bound for convergence in the mean. In the FIR case one has $\sum_{i=1}^{N} \lambda_i = trace\{\bar{\mathbb{X}}_{uu}\} = N\bar{x}_{uu}(0) = N \cdot \mathcal{E}\{u_k^2\}$*, leading to*

$$\boxed{0 < \mu < \frac{2}{N \cdot \mathcal{E}\{u_k^2\}}}.$$

It is seen that the upper bound for μ *is inversely proportional to the filter length N.*
Finally, it is found that if the above conditions are satisfied

$$J_{ex}(\infty) = J_{MSE}(\mathbf{w}_{WF}) \cdot \underbrace{\frac{\sum_{i=1}^{N} \frac{\mu\lambda_i}{2-\mu\lambda_i}}{1 - \sum_{i=1}^{N} \frac{\mu\lambda_i}{2-\mu\lambda_i}}}_{\mathcal{M}}.$$

For small values of μ *this reduces to*

$$\boxed{J_{ex}(\infty) \approx J_{MSE}(\mathbf{w}_{WF}) \cdot \underbrace{\frac{\mu}{2} \sum_{i=1}^{N} \lambda_i}_{\mathcal{M}}}.$$

For the FIR case that is

$$\boxed{J_{ex}(\infty) \approx J_{MSE}(\mathbf{w}_{WF}) \cdot \underbrace{\frac{\mu}{2} N \cdot \mathcal{E}\{u_k^2\}}_{\mathcal{M}}}.$$

The mismatch $\mathcal{M}$ *is often expressed as a percentage. A mismatch of 10% is ordinarily considered to be satisfactory, meaning that the LMS algorithm produces an MSE that is (upon convergence) 10% greater than the minimum MSE, i.e* $J_{MSE}(\mathbf{w}_{WF})$*. Hence, a choice for* μ *could be*

$$\boxed{\mu < \frac{0.2}{\sum_{i=1}^{N} \lambda_i}}$$

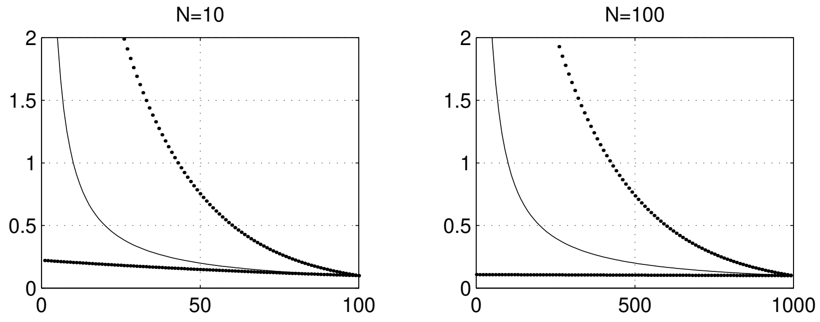

Figure 3.4: LMS and RLS convergence

Obviously, the choice of μ must be based on a compromise between fast convergence and small excess mean-square error. A (much) smaller μ jeopardizes fast convergence even further.

3.3.3 LMS versus RLS convergence

It has been noted that the upper bound for μ is inversely proportional to the filter length N. For large N, where – compared to RLS – the LMS algorithm indeed achieves a large reduction in the computational complexity (cf. $\mathcal{O}(N)$ instead of $\mathcal{O}(N^2)$ operations per update), one may expect poor convergence behavior. Furthermore, it is usual that the larger the value of N, the larger the eigenvalue spread of the covariance matrix will be which again means slower convergence.

This is illustrated in **Figure 3.4**. *We consider two cases, namely $N = 10$ and $N = 100$. Recall that the excess MSE for RLS (in the unwindowed/time-invariant case) is proportional to $\frac{N}{k}$, and so if 10% excess MSE is considered to be satisfactory, an RLS algorithm needs $k = 10 \cdot N$ data samples to achieve this. This is indicated in Figure 3.4, where the excess MSE is plotted as a function of the number of samples k. The solid line curves are the RLS curves, with 0.1 excess MSE at $k = 10 \cdot N$.*
Let us now assume that LMS converges to the same excess MSE level, for the same number of data points k. This allows us to check what the initial weight-vector error should be in order to achieve this. Recall that the expected convergence behavior for LMS is the steepest-descent behavior, so that the MSE converges (to the 10% excess MSE level) exponentially. The smallest eigenvalue of the correlation matrix defines the slowest decay, corresponding to (see the $J_{MSE}(\mathbf{w})$ -formula for the steepest-descent algorithm)

$$(1-\mu\lambda_{min})^{2k}.$$

First, if (in the ideal case) all eigenvalues are equal, we have

$$\mu = \frac{0.2}{\sum_{i=1}^{N} \lambda_i} = \frac{0.2}{N \cdot \lambda_{min}}$$

and so

$$(1-\mu\lambda_{min})^{2k} = (1-\frac{0.2}{N})^{2k}.$$

Such curves, passing through MSE=0.1 at $k = 10 \cdot N$ *are indicated in Figure 3.4, by the upper dotted lines. It is seen that (both for* $N = 10$ *and* $N = 100$*) the initial MSE (for* $L = 0$*) can be quite large, so that there are apparently no convergence problems. However, if (in a more realistic setting) we assume that the eigenvalue spread is 'linear', i.e* $\lambda_{N-i} = i \cdot \lambda_{min}$, *we have*

$$\mu = \frac{0.2}{\sum_{i=1}^{N} \lambda_i} \approx \frac{0.4}{N^2 \cdot \lambda_{min}}$$

and so

$$(1 - \mu\lambda_{min})^{2k} = (1 - \frac{0.4}{N^2})^{2k}.$$

Such curves, passing through MSE=0.1 at $k = 10 \cdot N$ *are indicated in Figure 3.4, by the lower dotted lines. It is seen that (for* $N = 10$, *and even more for* $N = 100$*) the initial MSE (for* $k = 0$*) has to be close to the the final excess MSE, so that here we definitely have convergence problems. That is to say the LMS algorithm will take a long time to converge with respect to the RLS algorithm (unless we are lucky and guess a good value for the initial weight vector* $\mathbf{w}_{LMS}(0)$*).*

3.3.4 LMS for time-varying set-ups

The LMS algorithm works with an instantaneous estimate of the gradient, which at time k is computed from data at time k only. So in a sense, LMS works with a rectangular window of length $L = 1$. *It can do so because only gradient information is needed, and no correlation/cross-correlation information (unlike in RLS). Hence the LMS algorithm may be applied to time-varying set-ups immediately.*

To allow fast time-varying systems to be tracked properly, the step-size parameter μ *should be chosen sufficiently large. With too small a* μ, *the LMS solution will 'lag behind'. As a consequence, the lag error has a mean-square value that decreases with an increase in* μ. *The total MSE is*

$$J_{MSE}(\mathbf{w}_{LMS}(k)) = J_{MSE}(\mathbf{w}_{WF}(k)) + \underbrace{J_{MSE}(\mathbf{w}_{WF}(k)) \cdot \frac{\mu}{2} \sum_{i=1}^{N} \lambda_i}_{J_{ex}^{noise}(k)} + \underbrace{f(\frac{1}{\mu})}_{J_{ex}^{lag}(k)}$$

The choice of μ *must be based on a compromise between fast tracking (large* μ*) and small* J_{ex}^{noise} *(small* μ*). For rapidly time-varying set-ups, even with the largest possible value for* μ, *the lag error may dominate. LMS is then inappropriate and so one must rely on more complex algorithms (such as RLS type algorithms).*

3.3.5 LMS variants

A number of closely related variants to LMS exist. Normalized LMS has been mentioned already, and is also treated in the next section.

In Block LMS, the gradient vectors are averaged over several (K) iterations prior to making adjustments of the filter coefficients. The filter coefficients are updated only once every K iterations

$$\mathbf{w}_{LMS}((k+1)\cdot K) = \mathbf{w}_{LMS}(k\cdot K) + \frac{\mu}{K}\cdot\sum_{i=0}^{K-1}\mathbf{u}_{k\cdot K+i}\cdot(d_{k\cdot K+i} - \mathbf{u}_{k\cdot K+i}^T\mathbf{w}_{LMS}(k\cdot K)).$$

The averaging operation reduces the noise in the estimate of the gradient vector, and hence the excess mean-square error. Block processing also allows the computation of the gradients (as well as the prediction errors) in the frequency domain, which (for large K) leads to additional computational saving (through the use of FFT techniques). This is referred to as frequency domain LMS. Frequency domain LMS and the related class of transform domain LMS techniques will be treated later.

To further reduce the number of multiplications in the LMS algorithm, the signed-error algorithm has been suggested. The updating formula is

$$\mathbf{w}_{LMS}(k+1) = \mathbf{w}_{LMS}(k) + \mu\cdot\mathbf{u}_{k+1}\cdot sign(d_{k+1} - \mathbf{u}_{k+1}^T\mathbf{w}_{LMS}(k)).$$

This may be viewed as a stochastic gradient algorithm that attempts to minimize the least absolute value of the error, i.e. $J(\mathbf{w}) = \mathcal{E}\{|e_k|\}$.
An alternative way to reduce the numerical complexity is to apply the signum function to the vector $\mathbf{u}_{k+1}$ *in the updating formula (signed-regressor LMS)*

$$\mathbf{w}_{LMS}(k+1) = \mathbf{w}_{LMS}(k) + \mu\cdot sign(\mathbf{u}_{k+1})\cdot(d_{k+1} - \mathbf{u}_{k+1}^T\mathbf{w}_{LMS}(k))$$

or to the complete gradient estimate (sign-sign LMS)

$$\mathbf{w}_{LMS}(k+1) = \mathbf{w}_{LMS}(k) + \mu\cdot sign(\mathbf{u}_{k+1})\cdot sign(d_{k+1} - \mathbf{u}_{k+1}^T\mathbf{w}_{LMS}(k)).$$

Such algorithms are simple to implement since extracting the sign of a number is easy to do in hardware (cf multiplication). Howevere they are not very well understood, and sometimes even show divergence for 'suitably pathological' input signals.

3.3.6 Normalized LMS (NLMS)

¿From the analysis of the LMS algorithm, we know that the stable range for the step-size μ *is inversely proportional to the sum of the eigenvalues of* $\bar{\mathbb{X}}_{uu}$ *and hence the power of the input signals* $\mathbf{u}_k$. *To end up with an algorithm that is robust with respect to power variations, one may introduce a normalization of the step-size by means of a quantity that behaves roughly as the power. One such quantity is* $\mathbf{u}_k^T\mathbf{u}_k$. *This results in the already cited normalized LMS (NLMS) algorithm, the updating formula of which is repeated here for convenience*

$$\boxed{\mathbf{w}_{NLMS}(k+1) = \mathbf{w}_{NLMS}(k) + \frac{\bar{\mu}}{\alpha^2 + \mathbf{u}_{k+1}^T\mathbf{u}_{k+1}}\cdot\mathbf{u}_{k+1}\cdot(d_{k+1} - \mathbf{u}_{k+1}^T\mathbf{w}_{NLMS}(k))}\ .$$

The α *is a (small) constant that prevents overflow in the event of* $\mathbf{u}_k^T\mathbf{u}_k = 0$.

Suffice it to say that in an NLMS convergence/stability analysis it often assumed that

$$\frac{\bar{\mu}}{\alpha^2 + \mathbf{u}_{k+1}^T\mathbf{u}_{k+1}} \approx \frac{\bar{\mu}}{\alpha^2 + trace\{\bar{\mathbb{X}}_{uu}\}} \approx \ constant$$

so that the LMS fomulas can be used. ¿From this it is seen the NLMS is **convergent in the mean** *as well as* **convergent in the mean square** *if*

$$\boxed{0 < \bar{\mu} < 2}\,.$$

This may be viewed as a 'normalized' version of the μ*-bounds for LMS. A 10% excess MSE is obtained with (roughly)*

$$\boxed{\bar{\mu} < 0.2}\,.$$

Remark 2.17. *We have derived the NLMS algorithm as a natural modification to the LMS algorithm. It is interesting to not, however, that NLMS may be derived as the solution to a specific optimization problem, which provides further insight in the method. Given* $\mathbf{w}(k)$*, we may want to compute a* $\mathbf{w}(k+1)$ *that minimizes*

$$\tilde{J}(\mathbf{w}(k+1)) = \alpha^2 \|\mathbf{w}(k+1) - \mathbf{w}(k)\|_2^2 + (d_{k+1} - \mathbf{u}_{k+1}^T \mathbf{w}(k+1))^2.$$

i.e. compute $\mathbf{w}(k+1)$ *in order to best fit the current data without changing its value too much from the previous time (i.e.* $\mathbf{w}(k)$*). One can prove*[10] *that the solution to this problem is indeed given by the NLMS formula, albeit with* $\bar{\mu} = 1$*. Note that for* $\alpha \to 0$*, the optimization problem is effectively turned into a constrained optimization problem*

$$\tilde{J}(\mathbf{w}(k+1)) = \|\mathbf{w}(k+1) - \mathbf{w}(k)\|_2^2$$

subject to

$$d_{k+1} = \mathbf{u}_{k+1}^T \mathbf{w}(k+1).$$

So in the NLMS algorithm with $\alpha = 0$ *and* $\bar{\mu} = 1$*, the* $\mathbf{w}(k+1)$ *is the weight vector that fits the last observations exactly, while it is as close to* $\mathbf{w}(k)$ *as possible.*

Remark 2.19. [11] *Interestingly, it turns out that the NMLS may also be derived from the directionally weighted RLS algorithm, when a time varying forget factor* λ *is used. This is a third way of deriving the NLMS algorithm, and is explained only briefly here.*

Let us first assume that in the directionally weighted RLS algorithm (section 2.3.3)

$$\mathbb{X}_{uu}^{k|k+1} = I.$$

With this, one obtains

$$[\mathbb{X}_{uu}(k+1)]^{-1} = I - \frac{\mathbf{u}_{k+1}\mathbf{u}_{k+1}^T}{1 + \mathbf{u}_{k+1}^T \mathbf{u}_{k+1}}$$

$$\boxed{\mathbf{w}(k+1) = \mathbf{w}(k) + \frac{1}{1+\mathbf{u}_{k+1}^T \mathbf{u}_{k+1}} \mathbf{u}_{k+1} \cdot (d_{k+1} - \mathbf{u}_{k+1}^T \mathbf{w}(k))}\,.$$

[10] Compare the above cost function with the one given at the end of section 3.1 (RLS initialization). The NLMS formula then follows directly from a single, properly initialized, RLS iteration.

[11] Optional reading.

The last formula is indeed the NLMS formula with $\mu = 1$ *and* $\alpha = 1$. *One can verify that by setting*

$$\lambda_{k+1}^2 = \frac{1}{1+\mathbf{u}_{k+1}^T \mathbf{u}_{k+1}}$$

the next 'intermediate' covariance matrix is also going to be equal to the identity matrx, i.e.

$$\mathbb{X}_{uu}^{k+1|k+2} = I$$

which 'closes the loop'. In conclusion, NLMS (with $\mu = \alpha = 1$*) may be viewed as a directionally weighted RLS algorithm, where in each time update the weighting factor* λ *is chosen such that the covariance matrix basically remains unchanged always. The end result is then that the covariance matrix updating formulas may indeed be left out, so that the complexity of the algorithm is reduced by a factor* $\mathcal{O}(N)$.