EE210A: Adaptation and Learning
Professor Ali H. Sayed
LECTURE #09

STOCHASTIC GRADIENT ALGORITHMS

Sections in order: 10.1-10.3, 10.5-10.8, 11.1, 11.3, Chapter 12, Sections 13.1, 14.1
In Chapters 10–14 we start to develop the theory of adaptive algorithms by studying \textit{stochastic-gradient} methods. These methods are obtained from steepest-descent implementations by replacing the required gradient vectors and Hessian matrices by some suitable approximations. Different approximations lead to different algorithms with varied degrees of complexity and performance properties. The resulting methods will be generically called \textit{stochastic-gradient algorithms} since, by employing estimates for the gradient vector, the update directions become subject to random fluctuations that are often referred to as \textit{gradient noise}. 
STOCHASTIC GRADIENT ALGORITHMS

Stochastic-gradient algorithms serve at least two purposes. First, they avoid the need to know the exact signal statistics (e.g., covariances and cross-covariances), which are necessary for a successful steepest-descent implementation but are nevertheless rarely available in practice. Stochastic-gradient methods achieve this feature by means of a learning mechanism that enables them to estimate the required signal statistics.

Second, these methods possess a tracking mechanism that enables them to track variations in the signal statistics. The two combined capabilities of learning and tracking are the main reasons behind the widespread use of stochastic-gradient methods (and the corresponding adaptive filters). It is because of these abilities that we tend to describe adaptive filters as “smart systems”; smart in the sense that they can learn the statistics of the underlying signals and adjust their behavior to variations in the “environment” in order to keep the performance level at check.
10.1 MOTIVATION

We start our discussions by reviewing the linear estimation problem of Sec. 8.1 and the corresponding steepest-descent methods of Chapter 8. Thus let $d$ be a zero-mean scalar-valued random variable with variance $\sigma_d^2 = \mathbb{E} |d|^2$. Let further $u^*$ be a zero-mean $M \times 1$ random variable with a positive-definite covariance matrix, $R_u = \mathbb{E} u^* u > 0$. The $M \times 1$ cross-covariance vector of $d$ and $u$ is denoted by $R_{du} = \mathbb{E} du^*$. We know from Sec. 8.1 that the weight vector $w$ that solves

$$\min_w \mathbb{E} |d - uw|^2$$

is given by

$$w^o = R_u^{-1} R_{du}$$

and that the resulting minimum mean-square error is

$$\text{m.m.s.e.} = \sigma_d^2 - R_{ud} R_u^{-1} R_{du}$$

(10.1)

(10.2)

(10.3)
MOTIVATION

In Chapter 8 we developed several steepest-descent methods that approximate \( w^o \) iteratively, until eventually converging to it. For example, in Sec. 8.1 we studied the following recursion with a constant step-size,

\[
    w_i = w_{i-1} + \mu [R_{du} - R_u w_{i-1}], \quad w_{-1} = \text{initial guess}
\]

(10.4)

where the update direction, \( R_{du} - R_u w_{i-1} \), was seen to be equal to the negative conjugate-transpose of the gradient vector of the cost function at \( w_{i-1} \), i.e.,

\[
    R_{du} - R_u w_{i-1} = - [\nabla_w J(w_{i-1})]^* 
\]

(10.5)

where

\[
    J(w) \triangleq \mathbb{E} |d - uw|^2
\]
MOTIVATION

In Sec. 9.7 we allowed for an iteration-dependent step-size, \( \mu(i) \), and studied the recursion

\[
    w_i = w_{i-1} + \mu(i)[R_{du} - R_u w_{i-1}], \quad w_{-1} = \text{initial guess}
\]  

(10.6)

and in Sec. 9.8 we studied Newton’s recursion,

\[
    w_i = w_{i-1} + \mu R_u^{-1}[R_{du} - R_u w_{i-1}], \quad w_{-1} = \text{initial guess}
\]  

(10.7)

where \( R_u^{-1} \) resulted from using

\[
    R_u = \nabla^2_w J(w_{i-1}) = \nabla^* w \left[ \nabla_w J(w_{i-1}) \right]
\]

More generally, when regularization is employed and when the step-size is also allowed to be iteration-dependent, the recursion for Newton’s method is replaced by

\[
    w_i = w_{i-1} + \mu(i) [\epsilon(i)I + R_u]^{-1} [R_{du} - R_u w_{i-1}]
\]  

(10.8)

for some positive scalars \( \{\epsilon(i)\} \); they could be set to a constant value for all \( i \), say \( \epsilon(i) = \epsilon \).
LIMITATIONS

Now all the steepest-descent formulations described above, i.e., (10.4), (10.6), (10.7) and (10.8), rely explicitly on knowledge of \( \{R_{du}, R_u\} \). This fact constitutes a limitation in practice and serves as a motivation for the development of stochastic-gradient algorithms for two reasons:

1. **Lack of statistical information.** First, the quantities \( \{R_{du}, R_u\} \) are rarely available in practice. As a result, the true gradient vector, \( \nabla_w J(w_{i-1}) \), and the true Hessian matrix, \( \nabla^2_w J(w_{i-1}) \), cannot be evaluated exactly and a true steepest-descent implementation becomes impossible. Stochastic-gradient algorithms replace the gradient vector and the Hessian matrix by approximations for them. There are several ways for obtaining such approximations. The better the approximation, the closer we expect the performance of the resulting stochastic-gradient algorithm to be to that of the original steepest-descent method. In Parts IV (*Mean-Square Performance*) and V (*Transient Performance*) we shall study and quantify the degradation in performance that occurs as a result of such approximations.

2. **Variation in the statistical information.** Second, and even more important, the quantities \( \{R_{du}, R_u\} \) tend to vary with time.\(^4\) In this way, the optimal weight vector \( w^o \) will also vary with time. It turns out that stochastic-gradient algorithms provide a mechanism for tracking such variations in the signal statistics.
10.2 INSTANTANEOUS APPROXIMATION

Assume that we have access to several observations of the random variables \( d \) and \( u \) in (10.1), say \( \{d(0), d(1), d(2), d(3), \ldots\} \) and \( \{u_0, u_1, u_2, u_3, \ldots\} \). We refer to the \( \{u_i\} \) as regressors. Observe that, in conformity with our notation in this book, we are using the boldface letter \( d \) to refer to the random variable, and the normal letter \( d \) to refer to observations (or realizations) of it. Likewise, we write \( u \) for the random vector and \( u \) for observations of it.

One of the simplest approximations for \( \{R_{du}, R_u\} \) is to use the instantaneous values

\[
\hat{R}_{du} = d(i)u_i^*, \quad \hat{R}_u = u_i^*u_i
\]  

(10.9)

This construction simply amounts to dropping the expectation operator from the actual definitions, \( R_{du} = \mathbb{E}d u^* \) and \( R_u = \mathbb{E}u^*u \), and replacing the random variables \( \{d, u\} \) by the observations \( \{d(i), u_i\} \) at iteration \( i \).
In this way, the gradient vector (10.5) is approximated by the instantaneous value

$$- [\nabla_w J(w_{i-1})]^* \approx d(i) u_i^* - u_i^* u_i w_{i-1} = u_i^* [d(i) - u_i w_{i-1}]$$

and the corresponding steepest-descent recursion (10.4) becomes

$$w_i = w_{i-1} + \mu u_i^* [d(i) - u_i w_{i-1}], \quad w_{-1} = \text{initial guess} \quad (10.10)$$

We continue to write $w_i$ to denote the estimate that is obtained via this approximation procedure although, of course, $w_i$ in (10.10) is different from the $w_i$ that is obtained from the steepest-descent algorithm (10.4): the former is based on using instantaneous approximations whereas the latter is based on using $\{R_{du}, R_u\}$. We do so in order to avoid an explosion of notation; the distinction between both estimates is usually clear from the context.
Algorithm 10.1 (LMS Algorithm) Consider a zero-mean random variable \( d \) with realizations \( \{d(0), d(1), \ldots\} \), and a zero-mean random row vector \( u \) with realizations \( \{u_0, u_1, \ldots\} \). The optimal weight vector \( w^o \) that solves

\[
\min_w \mathbb{E} |d - uw|^2
\]

can be approximated iteratively via the recursion

\[
w_i = w_{i-1} + \mu u_i^*[d(i) - u_i w_{i-1}], \quad i \geq 0, \quad w_{-1} = \text{some initial value} \]

where \( \mu \) is a positive step-size (usually small).

The stochastic-gradient approximation (10.10) is one of the most widely used adaptive algorithms in current practice due to its striking simplicity. It is widely known as the least-mean-squares (LMS) algorithm, or sometimes as the Widrow-Hoff algorithm in honor of its originators.
\[ w_i = w_{i-1} + \mu u_i^* [d(i) - u_i w_{i-1}] \, , \quad w_{-1} = \text{initial guess} \]  

(10.10)

10.3 COMPUTATIONAL COST

A useful property of LMS is its computational simplicity. The evaluations that follow for the number of computations that are required by LMS are intended to provide an approximate idea of its computational complexity. While there can be different ways to perform specific calculations, the resulting overall filter complexity will be of the same order of magnitude (and often quite similar) to the values we derive in this section for LMS, and in subsequent sections for other adaptive filters.

Note that each step of the LMS algorithm requires a handful of straightforward computations, which are explained below:
1. Each iteration (10.10) requires the evaluation of the inner product $u_i w_{i-1}$, between two vectors of size $M$ each. This calculation requires $M$ complex multiplications and $(M - 1)$ complex additions. Using the fact that one complex multiplication requires four real multiplications and two real additions, while one complex addition requires two real additions, we find that the evaluation of this inner product requires $4M$ real multiplications and $4M - 2$ real additions.

2. The algorithm also requires the evaluation of the scalar $d(i) - u_i w_{i-1}$. This calculation amounts to one complex addition, i.e., 2 real additions.

3. Evaluation of the product $\mu [d(i) - u_i w_{i-1}]$, where $\mu$ is a real scalar, requires two real multiplications when the data is complex-valued. Usually, $\mu$ is chosen as a power of $2^{-1}$, say $2^{-m}$ for some integer $m > 0$. In this case, multiplying $\mu$ by $[d(i) - u_i w_{i-1}]$ can be implemented digitally very efficiently by means of shift registers, and we could therefore ignore the cost of this multiplication. In the text, we choose to account for the case of arbitrary step-sizes.
4. The algorithm further requires multiplying the scalar $\mu [d(i) - u_i w_{i-1}]$ by the vector $u_i^*$. This requires $M$ complex multiplications and, therefore, $4M$ real multiplications and $2M$ real additions.

5. Finally, the addition of the two vectors $w_{i-1}$ and $\mu u_i^* [d(i) - u_i w_{i-1}]$ requires $M$ complex additions, i.e., $2M$ real additions.

In summary, for general complex-valued signals, LMS requires $8M + 2$ real multiplications and $8M$ real additions per iteration. On the other hand, for real-valued data, LMS requires $2M + 1$ real multiplications and $2M$ real additions per iteration.
10.5 APPLICATION: ADAPTIVE CHANNEL ESTIMATION

**FIGURE 10.1** Noisy measurements of an FIR channel with an unknown impulse response vector $c$.

Figure 10.1 shows an FIR channel excited by a zero-mean random sequence $\{u(i)\}$. Its output is another zero-mean random sequence $\{d(i)\}$. At any time instant $i$, the state of the channel is captured by the regressor

$$u_i = \begin{bmatrix} u(i) & u(i - 1) & \ldots & u(i - M + 1) \end{bmatrix}$$
and its output is given by

\[ d(i) = u_i c + v(i) \]  (10.17)

where the column vector \( c \) denotes the channel impulse response sequence, and \( v(i) \) is a zero-mean noise sequence that is uncorrelated with \( u_i \). We again remind the reader of our

It is further assumed that the moments \( R_u = \mathbb{E} u_i^* u_i \), \( \sigma_d^2 = \mathbb{E} |d(i)|^2 \), and \( R_{du} = \mathbb{E} d(i) u_i^* \) are all independent of time. We also let \( \{u_i, d(i)\} \) denote observed values for the random variables \( \{u_i, d(i)\} \). It is important to distinguish between a measured value \( d(i) \) and its stochastic version \( d(i) \); similarly for \( u_i \) and \( u_i \). This distinction is relevant because while an adaptive filtering implementation operates on the measured data \( \{d(i), u_i\} \), its derivation and performance are characterized in terms of the statistical properties of the underlying stochastic variables \( \{d(i), u_i\} \).
The channel vector $c$ is modeled as an unknown constant vector. This situation is identical to the scenario discussed in Sec. 6.3, where $c$ was estimated by formulating a constrained linear estimation problem — see (6.18). In terms of the notation of the present section, the rows of the data matrix $H$ from Eq. (6.18) would be the $\{u_i\}$, while the entries of the observation vector $y$ in (6.18) would be the $\{d(i)\}$. The solution method (6.18) did not require knowledge of $\{R_{du}, R_u\}$.

There is an alternative way to estimate $c$ that does not require knowledge of $\{R_{du}, R_u\}$ either; it relies on using a stochastic-gradient (or adaptive) algorithm. The method can be motivated as follows. Assume we formulate the following linear least-mean-squares estimation problem:

$$\min_w \mathbb{E} |d(i) - u_i w|^2$$

whose solution we already know is

$$w^o = R_u^{-1} R_{du}$$

(10.18)
Then $w^o$ coincides with the desired unknown $c$. This is because if we multiply (10.17) by $u_i^*$ from the left and take expectations, we find that

$$
\mathbb{E} u_i^* d(i) = (\mathbb{E} u_i^* u_i) \cdot c + \mathbb{E} u_i^* v(i) = 0
$$

so that $c = R_u^{-1} R_{du} = w^o$. Therefore, if we can determine $w^o$ then we recover $c$. However, the moments $\{R_{du}, R_u\}$ are rarely available in practice so that determining $w^o$ via (10.18), or even via a related steepest-descent implementation such as

$$
w_i = w_{i-1} + \mu [R_{du} - R_u w_{i-1}]
$$

would not be viable. Instead, we can appeal to a stochastic-gradient approximation for estimating $w^o$. Using measurements $\{d(i), u_i\}$, we can estimate $w^o$ (and, hence, $c$) by using, e.g., the LMS recursion:

$$
w_i = w_{i-1} + \mu u_i^* [d(i) - u_i w_{i-1}] 
$$

(10.19)
FIGURE 10.2 A structure for adaptive channel estimation.
This discussion suggests the structure shown in Fig. 10.2, which employs an FIR filter with adjustable weights that is connected to the input and output signals \( \{d(i), u(i)\} \) of the channel. At each time instant \( i \), the measured output of the channel, \( d(i) \), is compared with the output of the adaptive filter, \( \hat{d}(i) = u_i w_{i-1} \), and an error signal, \( e(i) = d(i) - u_i w_{i-1} \), is generated. The error is then used to adjust the filter coefficients from \( w_{i-1} \) to \( w_i \) according to (10.19). In steady-state, if the step-size \( \mu \) is suitably chosen to guarantee filter convergence (usually, small step-sizes will do), then the error signal will assume small values, and the output \( \hat{d}(i) = u_i w_{i-1} \) of the adaptive filter will assume values close to \( d(i) \).

Consequently, from an input/output perspective, the (adaptive filter) mapping from \( u(i) \) to \( \hat{d}(i) \) will behave similarly to the channel, which maps \( u(i) \) to \( d(i) \). This construction assumes that the adaptive filter has at least as many taps as the channel itself; otherwise, performance degradation can occur due to under-modelling.
10.6 APPLICATION: ADAPTIVE CHANNEL EQUALIZATION

**FIGURE 10.3** Linear equalization of an FIR channel in the presence of additive noise.
Our second application is linear channel equalization. In Fig. 10.3, data symbols \( \{s(\cdot)\} \) are transmitted through a channel and the output sequence is measured in the presence of additive noise, \( v(i) \). The signals \( \{v(\cdot), s(\cdot)\} \) are assumed uncorrelated. The noisy output of the channel is denoted by \( u(i) \) and is fed into a linear equalizer with \( L \) taps. At any particular time instant \( i \), the state of the equalizer is given by

\[
\mathbf{u}_i = \begin{bmatrix} u(i) & u(i-1) & \ldots & u(i-L+1) \end{bmatrix}
\]

It is desired to determine the equalizer tap vector \( \mathbf{w} \) in order to estimate the signal \( d(i) = s(i - \Delta) \) optimally in the least-mean-squares sense. This application coincides with the one described in Sec. 5.4 except that now, in conformity with the notation of this chapter, we are denoting the input to the equalizer by \( u(i) \) and the symbol to be estimated by \( d(i) \).
Clearly, the equalizer \( w^o \) that solves

\[
\min_w \mathbb{E} |d(i) - u_i w|^2
\]

is given by

\[
w^o = R_u^{-1} R_{du}
\]

(10.20)

where \( R_u = \mathbb{E} u_i^* u_i \) and \( R_{du} = \mathbb{E} d(i) u_i^* \). In Sec. 5.4 we assumed knowledge of the channel tap vector \( c \) (assumed FlR) and used it to evaluate \( \{ R_{du}, R_u \} \) — refer to equations (5.15), which were defined in terms of a channel matrix \( H \). However, in practice, knowledge of \( \{ R_{du}, R_u \} \) and even \( c \) cannot be taken for granted. Actually, more often than not, these quantities are not available. For this reason, determining \( w^o \) via (10.20), or even via a related steepest-descent implementation such as

\[
w_i = w_{i-1} + \mu [R_{du} - R_u w_{i-1}]
\]

may not be viable.
In such situations, we can appeal to a stochastic-gradient approximation for estimating $w^o$. Assuming an initial training phase in which transmitted data $\{d(i) = s(i - \Delta)\}$ are known at the receiver (i.e., at the equalizer), we can then use the available measurements $\{d(i), u_i\}$ to estimate $w^o$ iteratively by using, e.g., the LMS recursion:

$$w_i = w_{i-1} + \mu u_i^* [d(i) - u_i w_{i-1}], \quad d(i) = s(i - \Delta)$$  \hspace{1cm} (10.21)

**FIGURE 10.4** Adaptive linear equalization of a channel in the presence of additive noise.
APPLICATION

This discussion suggests that we consider the structure of Fig. 10.4, with an FIR filter with adjustable weights that is connected in series with the channel. At each time instant $i$, the symbol $d(i) = s(i - \Delta)$ is compared with the output of the adaptive filter, $\hat{s}(i - \Delta)$, and an error signal, $e(i) = d(i) - u_iw_{i-1}$, is generated. The error is then used to adjust the filter coefficients from $w_{i-1}$ to $w_i$ according to (10.21). In steady-state, the error signal will assume small values and, hence, the output $\hat{s}(i - \Delta)$ of the adaptive filter will assume values close to $s(i - \Delta)$.

Consequently, from an input/output perspective, the combination channel/equalizer, which maps $s(i)$ to $\hat{s}(i - \Delta)$, behaves “like” a delay system with transfer function $z^{-\Delta}$. Observe that this scheme for adaptive channel equalization does not require knowledge of the channel.
In practice, following a training phase with a known reference sequence \( \{d(i)\} \), an equalizer could continue to operate in one of two modes. In the first mode, its coefficient vector \( w \) would be frozen and used thereafter to generate future outputs \( \{\hat{s}(i - \Delta)\} \). This mode of operation is appropriate when the training phase is successful enough to result in reliable estimates \( \hat{s}(i - \Delta) \), namely, estimates that lead to a low probability of error after they are fed into a decision device, which maps \( \hat{s}(i - \Delta) \) to the closest point in the symbol constellation, say \( \hat{s}(i - \Delta) \).

However, if the channel varies slowly with time, it may be necessary to continue to operate the equalizer in a decision-directed mode. In this mode of operation, the weight vector of the equalizer continues to be adapted even after the training phase has ended. Now, however, the output of the decision device replaces the reference sequence \( \{d(i)\} \) in the generation of the error sequence \( \{e(i)\} \). Figure 10.5 illustrates the operation of an equalizer during the training and decision-directed modes of operation.
FIGURE 10.5  An adaptive linear equalizer operating in two modes: training mode and decision-direction mode.
10.7 APPLICATION: DECISION-FEEDBACK EQUALIZATION

**FIGURE 10.6** A decision-feedback equalizer. It consists of a feedforward filter, a feedback filter, and a decision device.
Our third application is decision-feedback equalization. Again, data symbols \( \{s(\cdot)\} \) are transmitted through a channel and the output sequence is measured in the presence of additive noise, \( v(i) \). The signals \( \{v(\cdot), s(\cdot)\} \) are assumed uncorrelated. The noisy output of the channel is denoted by \( u(i) \) and is fed into a feedforward filter with \( L \) taps, as indicated in Fig. 10.6 for the case \( L = 3 \). The output of the decision device is fed into a feedback filter with \( Q \) taps; this filter works in conjunction with the feedforward filter in order to supply the decision device with an estimator \( \hat{s}(i - \Delta) \). Assuming correct decisions, at any particular time instant \( i \), the state of the equalizer is given by

\[
u_i = \begin{bmatrix} s(i - \Delta - 1) & \ldots & s(i - \Delta - Q) & u(i) & \ldots & u(i - L + 1) \end{bmatrix}
\]

That is, it contains the states of both the feedback and the feedforward filter.
APPLICATION

It is then desired to determine an equalizer tap vector

\[ w \triangleq \text{col}\{-b(1), -b(2), \ldots, -b(Q), f(0), f(1), \ldots, f(L - 1)\} \]

that estimates \( d(i) = s(i - \Delta) \) optimally in the least-mean-squares sense, where the \( \{ -b(i) \} \) denote the coefficients of the feedback filter while the \( \{ f(i) \} \) denote the coefficients of the forward filter.

Clearly, the equalizer \( w^o \) that solves \( \min_w \mathbb{E} |d(i) - u_i w|^2 \) is

\[ w^o = R_{u_i}^{-1} R_{du} \]  \hspace{1cm} (10.22)

where \( R_u = \mathbb{E} u_i^* u_i \) and \( R_{du} = \mathbb{E} d(i) u_i^* \). In Sec. 6.4, and also Prob. II.41, we assumed knowledge of the channel tap vector \( c \) (assumed FIR) and used it to evaluate \( \{ R_{du}, R_u \} \) — refer to equations (6.33) and (6.34), which were defined in terms of a channel matrix \( H \). However, in practice, knowledge of \( \{ R_{du}, R_u, c \} \) is generally unavailable.
APPLICATION

For this reason, determining $w^o$ via (10.22), or even via a related steepest-descent implementation such as

$$w_i = w_{i-1} + \mu[Rdu - R_u w_{i-1}]$$

would not be viable. Instead, we can appeal to a stochastic-gradient approximation for estimating $w^o$. Assuming an initial training phase in which transmitted data $\{d(i) = s(i - \Delta)\}$ are known at the receiver (i.e., at the equalizer), we can then use the available measurements $\{d(i), u_i\}$ to estimate $w^o$ iteratively by using, e.g., the LMS recursion:

$$w_i = w_{i-1} + \mu u_i^*[d(i) - u_i w_{i-1}], \quad d(i) = s(i - \Delta)$$  \hspace{1cm} (10.23)
FIGURE 10.7  Adaptive decision-feedback equalization. Both modes of operation are shown: training and decision-directed operation.
Thus consider the structure shown in Fig. 10.7, with FIR filters with adjustable weights used as feedforward and feedback filters. The top entries of $w_i$ would correspond to the coefficients of the feedback filter, while the bottom entries of $w_i$ would correspond to the coefficients of the feedforward filter. Likewise, the leading entries of the regressor correspond to the state of the feedback filter, while the trailing entries of the regressor correspond to the state of the feedforward filter.

Figure 10.7 depicts two modes of operation: training and decision-directed. During training, delayed symbols are used as a reference sequence. At each time instant $i$, the symbol $d(i) = s(i - \Delta)$ is compared with the output of the adaptive filter, $\hat{s}(i - \Delta)$ (which is the input to the decision device), and an error signal, $e(i) = d(i) - u_i w_{i-1}$, is generated. The error is then used to adjust the coefficients of the feedback and feedforward filters according to (10.23). During training, the state (or regressor) of the equalizer is given by

$$u_i = \begin{bmatrix} s(i - \Delta - 1) & \ldots & s(i - \Delta - Q) & | & u(i) & \ldots & u(i - L + 1) \end{bmatrix}$$

while during decision-directed operation, the signal $d(i)$ is replaced by the output of the decision device, $\hat{s}(i - \Delta)$, so that the state of the equalizer is then given by

$$u_i = \begin{bmatrix} \hat{s}(i - \Delta - 1) & \ldots & \hat{s}(i - \Delta - Q) & | & u(i) & \ldots & u(i - L + 1) \end{bmatrix}$$
10.8 ENSEMBLE-AVERAGE LEARNING CURVES

It is often necessary to evaluate the performance of a stochastic-gradient algorithm. A common way to do so is to construct its ensemble-average learning curve, which is defined below.

Recall that for least-mean-squares estimation, the learning curve of a steepest-descent method was defined in Sec. 9.5 as (cf. (9.1)):

\[
J(i) \triangleq \mathbb{E} |d - uw_{i-1}|^2, \quad i \geq 0
\]

where \( w_{i-1} \) is the weight estimate at iteration \( i - 1 \) that is given by the steepest-descent algorithm. Evaluation of \( J(i) \) would require knowledge of \( \{\sigma_d^2, R_{du}, R_u\} \). However, in a stochastic-gradient implementation, we do not have access to this statistical information but only to observations of the random variables \( d \) and \( u \), namely \( \{d(i), u_i\} \).
LEARNING CURVES

If we replace $d$ by $d(i)$ and $u$ by $u_i$ in the above expression for $J(i)$, then the difference $d - uw_{i-1}$ becomes $d(i) - u_iw_{i-1}$, with the $w_{i-1}$ now denoting the weight estimate that is obtained from the stochastic-gradient implementation (e.g., LMS). We have denoted the difference $d(i) - u_iw_{i-1}$ by $e(i)$ earlier in this chapter and called it the a priori output estimation error,

$$e(i) = d(i) - u_iw_{i-1}$$

We can then estimate the learning curve of an adaptive filter as follows. We run the algorithm for a certain number of iterations, say for $0 \leq i \leq N$. The duration $N$ is usually chosen large enough so that convergence is observed. We then compute the error sequence $\{e(i)\}$ and the corresponding squared-error curve $\{|e(i)|^2, 0 \leq i \leq N\}$. We denote this squared-error curve by

$$\left\{|e^{(1)}(i)|^2, 0 \leq i \leq N\right\}$$
LEARNING CURVES

After $L$ such experiments, we obtain $L$ squared-error curves,

$$\left\{ \left| e^{(1)}(i) \right|^2, \left| e^{(2)}(i) \right|^2, \ldots, \left| e^{(L)}(i) \right|^2 \right\}, \quad 0 \leq i \leq N$$

The *ensemble-average* learning curve, over the interval $0 \leq i \leq N$, is defined as the average over the $L$ experiments:

$$\hat{J}(i) \triangleq \frac{1}{L} \left( \sum_{j=1}^{L} \left| e^{(j)}(i) \right|^2 \right), \quad i \geq 0$$

The averaged curve $\hat{J}(i)$ so defined is a sample-average approximation of the true learning curve $J(i)$. 
Example 10.1 (Learning curves)

We illustrate the construction of learning curves by considering an example in the context of channel estimation, as described in Sec. 10.5. The impulse response sequence of the channel is chosen as

\[ c = \text{col}\{1, 0.5, -1, 2\}, \text{ i.e., its transfer function is} \]

\[ C(z) = 1 + 0.5z^{-1} - z^{-2} + 2z^{-3} \]

The channel impulse response, along with its magnitude frequency response, are shown in Fig. 10.8.

**FIGURE 10.8** The impulse-response sequence and the magnitude-frequency response of the channel \( C(z) = 1 + 0.5z^{-1} - z^{-2} + 2z^{-3} \).
White input data \( \{u(i)\} \) of unit variance is fed into the channel and the output sequence is observed in the presence of white additive noise of variance 0.01. A total of \( N = 600 \) samples \( \{u(i), d(i)\} \) are generated and used to train an adaptive filter with \( M = 4 \) taps. The filter is trained by using the LMS algorithm of this chapter with step-size \( \mu = 0.01 \), as well as two other algorithms derived in Chapters 11 and 14 for comparison purposes, namely, the so-called \( \varepsilon-\)NLMS algorithm with step-size \( \mu = 0.2 \) and \( \varepsilon = 0.001 \), and the RLS algorithm with \( \lambda = 0.995 \) and the same value of \( \varepsilon \). All filters start from the same initial condition \( w_{-1} = 0 \).

Figure 10.9 shows two typical instantaneous squared-error curves for each of the algorithms over the first 200 iterations, i.e., the rows show plots of \( |e(i)|^2 \) versus time in two random simulations for each algorithm. Observe how the curves die out quicker for \( \varepsilon-\)NLMS and RLS relative to LMS. By averaging \( L = 300 \) such curves for each algorithm, we obtain the ensemble-averaging learning curves shown in Fig. 10.10. These curves illustrate the fact that the convergence speed increases as we move from LMS to \( \varepsilon-\)NLMS to RLS.
FIGURE 10.9  Typical squared-error curves for LMS, ε-NLMS, and RLS.
FIGURE 10.10    Ensemble-average learning curves for LMS, $\epsilon$-NLMS, and RLS obtained by averaging over 300 experiments.
11.1 INSTANTANEOUS APPROXIMATION

Assume again that we have access to several observations of the random variables $d$ and $u$ in (10.1), say $\{d(0), d(1), d(2), d(3), \ldots\}$ and $\{u_0, u_1, u_2, u_3, \ldots\}$. The normalized LMS algorithm can be motivated in much the same way as LMS except that now we start from the regularized Newton’s recursion (10.8) and assume that the regularization sequence $\{\epsilon(i)\}$ and the step-size sequence $\mu(i)$ are constants, say, $\epsilon(i) = \epsilon$ and $\mu(i) = \mu$. Thus using

$$w_i = w_{i-1} + \mu [\epsilon I + R_u]^{-1} [R_{du} - R_u w_{i-1}] \quad (11.1)$$

and replacing the quantities $(\epsilon I + R_u)$ and $(R_{du} - R_u w_{i-1})$ by the instantaneous approximations $(\epsilon I + u_i^* u_i)$ and $u_i^* [d(i) - u_i w_{i-1}]$, respectively, we arrive at the stochastic-gradient recursion

$$w_i = w_{i-1} + \mu [\epsilon I + u_i^* u_i]^{-1} u_i^* [d(i) - u_i w_{i-1}] \quad (11.2)$$
This recursion, in its current form, requires the inversion of the matrix \((\epsilon I + u_i^* u_i)\) at each iteration. This step can be avoided by reworking the recursion into an equivalent simpler form. Thus note that \((\epsilon I + u_i^* u_i)\) is a rank-one modification of a multiple of the identity matrix, and the inverse of every such matrix has a similar structure. To see this, we simply apply the matrix inversion formula (5.4) to get

\[
[\epsilon I + u_i^* u_i]^{-1} = \epsilon^{-1} I - \frac{\epsilon^{-2}}{1 + \epsilon^{-1} ||u_i||^2} u_i^* u_i
\]  

(11.3)

where the expression on the right-hand side is a rank-one modification of \(\epsilon^{-1} I\). If we now multiply both sides of (11.3) by \(u_i^*\) from the right we obtain

\[
[\epsilon I + u_i^* u_i]^{-1} u_i^* = \epsilon^{-1} u_i^* - \frac{\epsilon^{-2}}{1 + \epsilon^{-1} ||u_i||^2} u_i^* ||u_i||^2 = \frac{u_i^*}{\epsilon + ||u_i||^2}
\]

which is a scalar multiple of \(u_i^*\). Substituting into (11.2) we arrive at the \(\epsilon\)–NLMS recursion:

\[
\omega_i = \omega_{i-1} + \frac{\mu}{\epsilon + ||u_i||^2} u_i^* [d(i) - u_i \omega_{i-1}], \quad i \geq 0
\]  

(11.4)
Algorithm 11.1 ($\epsilon$-NLMS algorithm) Consider a zero-mean random variable $d$ with realizations $\{d(0), d(1), \ldots\}$, and a zero-mean random row vector $u$ with realizations $\{u_0, u_1, \ldots\}$. The optimal weight vector $w^o$ that solves

$$\min_w \mathbb{E} |d - uw|^2$$

can be approximated iteratively via the recursion

$$w_i = w_{i-1} + \frac{\mu}{\epsilon + \|u_i\|^2} u_i^* [d(i) - u_i w_{i-1}], \quad i \geq 0, \quad w_{-1} = \text{initial guess}$$

where $\mu$ is a positive step-size and $\epsilon$ is a small positive parameter.
Algorithm 12.1 (Sign-error LMS algorithm) The weight vector $w^*$ that solves (in terms of the mean of the $l_1$ norm — cf. Probs. III.14–III.15 and III.29):

$$\min_w \mathbb{E} |d - w^*w|$$

can be approximated iteratively via the recursion

$$w_i = w_{i-1} + \mu u_i^* \text{csgn}[d(i) - u_i w_{i-1}], \quad i \geq 0$$

\[\text{csgn}(x) \triangleq \text{sign}(x_r) + j \text{sign}(x_i) \quad \text{where} \quad \text{sign}(a) \triangleq \begin{cases} +1 & \text{if } a > 0 \\ -1 & \text{if } a < 0 \\ 0 & \text{if } a = 0 \end{cases} \quad (12.1)\]
Sign-error LMS. The motivation for introducing sign-error LMS, especially for real-valued data, is due to its computational simplicity. While it may seem from Table 12.2 that the cost per iteration of sign-error LMS is similar to that of LMS, the point is that the step-size $\mu$ is usually selected as a power of $2^{-1}$, say $\mu = 2^{-m}$ for some integer $m > 0$. When this is the case, the evaluation of $\mu u_i^T \text{sign}[e(i)]$ in the real case can be implemented digitally very efficiently by means of shift registers, and we can ignore the $M$ multiplications that are needed for a generic $\mu$. In this case, we can replace the $2M$ figure that appears in Table 12.2 by $M$ multiplications. This simplification is not possible for LMS because it uses $e(i)$ instead of its sign. Nevertheless, the simplification in computations for sign-error LMS comes at the expense of slower convergence.
Algorithm 12.2 (Leaky-LMS algorithm) The weight vector $w^o$ that solves (cf. Probs III.12 and III.26):

$$\min_w \left[ \alpha \| w \|^2 + \mathbb{E} |d - uw|^2 \right]$$

for some positive constant $\alpha$, can be approximated iteratively via the recursion

$$w_i = (1 - \mu \alpha)w_{i-1} + \mu u_i^* [d(i) - u_i w_{i-1}], \quad i \geq 0$$
**Leaky-LMS.** The LMS algorithm can suffer from a potential instability problem when the covariance matrix $R_u$ is singular or close to singular (an example is given in Prob. IV.40 in the context of fractionally-spaced equalizers). When this happens, the weight estimates $w_i$ can drift and grow unbounded — see Prob. III.27 for an example and also Prob. IV.39 for a more detailed explanation. The leaky-LMS algorithm limits the growth of the weight estimates by employing a coefficient, $(1 - \mu \alpha)$, in the recursion for the weight vector. We shall study the properties of this algorithm in Part V (Transient Performance)— see Probs. V.27–V.32. In particular, we shall see that while leaky-LMS solves the drift problem, it nevertheless introduces a bias problem in that the mean value of $w_i$ will not tend to the optimal solution $w^o = R_u^{-1} R_{du}$ of the normal equations.
Algorithm 12.3 (LMF algorithm) The weight vector $w^o$ that solves (cf. Probs. III.16 and III.30):

$$\min_w \ E |d - uw|^4$$

can be approximated iteratively via the recursion

$$w_i = w_{i-1} + \mu u_i^* e(i) |e(i)|^2, \quad i \geq 0$$
**Algorithm 12.4 (LMMN algorithm)** The weight vector $w^o$ that solves (cf. Probs. III.17 and III.31):

$$
\min_w \mathbb{E} \left[ \delta |e|^2 + \frac{1}{2} (1 - \delta) |e|^4 \right], \quad e = d - uw
$$

for some constant $0 \leq \delta \leq 1$, can be approximated iteratively via the recursion

$$
w_i = w_{i-1} + \mu u_i^* e(i) \left[ \delta + (1 - \delta) |e(i)|^2 \right], \quad i \geq 0
$$
### Non-Blind Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Recursion</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS with constant step-size</td>
<td>$w_i = w_{i-1} + \mu u_i^* e(i)$</td>
</tr>
<tr>
<td>LMS with time-variant step-size</td>
<td>$w_i = w_{i-1} + \mu(i) u_i^* e(i)$</td>
</tr>
<tr>
<td>$\epsilon$-NLMS</td>
<td>$w_i = w_{i-1} + \frac{\mu}{\epsilon + |u_i|^2} u_i^* e(i)$</td>
</tr>
<tr>
<td>$\epsilon$-NLMS with power normalization</td>
<td>$p(i) = \beta p(i-1) + (1 - \beta)</td>
</tr>
<tr>
<td>Sign-error LMS</td>
<td>$w_i = w_{i-1} + \mu u_i^* \text{csgn}[e(i)]$</td>
</tr>
<tr>
<td>Leaky-LMS</td>
<td>$w_i = (1 - \mu\alpha) w_{i-1} + \mu u_i^* e(i)$, $\alpha &gt; 0$</td>
</tr>
<tr>
<td>LMF</td>
<td>$w_i = w_{i-1} + \mu u_i^*</td>
</tr>
<tr>
<td>LMMNN</td>
<td>$w_i = w_{i-1} + \mu u_i^* [e(i)\delta + (1 - \delta)</td>
</tr>
<tr>
<td></td>
<td>$0 \leq \delta \leq 1$</td>
</tr>
<tr>
<td>RLS</td>
<td>$P_i = \lambda^{-1} \left[ P_{i-1} - \frac{\lambda^{-1} P_{i-1} u_i^* u_i P_{i-1}}{1 + \lambda^{-1} u_i P_{i-1} u_i^*} \right]$</td>
</tr>
<tr>
<td></td>
<td>$P_{-1} = \epsilon^{-1} I$, $0 \ll \lambda \leq 1$</td>
</tr>
<tr>
<td>Gauss-Newton (GN)</td>
<td>$P_i = \frac{\lambda^{-1}}{1 - \alpha} \left[ P_{i-1} - \frac{\lambda^{-1} P_{i-1} u_i^* u_i P_{i-1}}{(1-\alpha)} + \lambda^{-1} u_i P_{i-1} u_i^* \right]$</td>
</tr>
<tr>
<td></td>
<td>$w_i = w_{i-1} + \mu P_i u_i^* e(i)$</td>
</tr>
<tr>
<td></td>
<td>$P_{-1} = \epsilon^{-1} I$, $\alpha &gt; 0$ (small), $0 \ll \lambda \leq 1$</td>
</tr>
</tbody>
</table>

$e(i) = d(i) - u_i w_{i-1}$. 
TABLE 12.2 A comparison of the *estimated* computational cost per iteration for several stochastic-gradient algorithms for *real-valued* data in terms of the number of *real* multiplications, *real* additions, *real* divisions, and comparisons with zero (or sign evaluations).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\times$</th>
<th>$+$</th>
<th>$/$</th>
<th>sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS with constant step-size</td>
<td>$2M + 1$</td>
<td>$2M$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2M)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMS with time-variant step-size</td>
<td>$2M + 1$</td>
<td>$2M$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2M)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\epsilon$-NLMS</td>
<td>$3M + 1$</td>
<td>$3M$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(3M)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\epsilon$-NLMS with power normalization</td>
<td>$2M + 4$</td>
<td>$2M + 2$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2M + 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sign-error LMS</td>
<td>$2M$</td>
<td>$2M$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(M)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>leaky-LMS</td>
<td>$3M + 2$</td>
<td>$2M + 1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2M + 1)</td>
<td></td>
<td>(3M + 1)</td>
<td></td>
</tr>
<tr>
<td>LMF</td>
<td>$2M + 3$</td>
<td>$2M$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2M + 2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMMN</td>
<td>$2M + 4$</td>
<td>$2M + 2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2M + 2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLS</td>
<td>$M^2 + 5M + 1$</td>
<td>$M^2 + 3M$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>GN</td>
<td>$M^2 + 7M + 1$</td>
<td>$M^2 + 3M + 1$</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
TABLE 12.3 A comparison of the estimated computational cost per iteration for several stochastic-gradient algorithms for the case of complex-valued data in terms of the number of real multiplications, real additions, real divisions, and comparisons with zero (or sign evaluations).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>×</th>
<th>+</th>
<th>/</th>
<th>sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS with constant step-size</td>
<td>$8M + 2$</td>
<td>$8M$</td>
<td>$8M$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(8M)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMS with time-variant step-size</td>
<td>$8M + 2$</td>
<td>$8M$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(8M)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\varepsilon$-NLMS</td>
<td>$10M + 2$</td>
<td>$10M$</td>
<td>$1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(10M)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\varepsilon$-NLMS with power normalization</td>
<td>$8M + 6$</td>
<td>$8M + 4$</td>
<td>$1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(8M + 2)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sign-error LMS</td>
<td>$6M$</td>
<td>$6M$</td>
<td>$2$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(4M)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>leaky-LMS</td>
<td>$10M + 3$</td>
<td>$8M + 1$</td>
<td>$2$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(8M + 2)$</td>
<td>$(9M + 1)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMF</td>
<td>$8M + 5$</td>
<td>$8M + 1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(8M + 3)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMMN</td>
<td>$8M + 6$</td>
<td>$8M + 3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(8M + 3)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLS</td>
<td>$4M^2 + 16M + 1$</td>
<td>$4M^2 + 12M - 1$</td>
<td>$1$</td>
<td></td>
</tr>
<tr>
<td>GN</td>
<td>$4M^2 + 20M + 1$</td>
<td>$4M^2 + 12M$</td>
<td>$3$</td>
<td></td>
</tr>
</tbody>
</table>
Algorithm 12.5 (CMA1-2 and NCMA) A weight vector $\mathbf{w}^o$ that solves (cf. Probs. III.21 and III.36):

$$\min_{\mathbf{w}} \mathbb{E} (\gamma - |\mathbf{u}\mathbf{w}|)^2$$

can be approximated iteratively via the recursion

$$w_i = w_{i-1} + \mu \mathbf{u}_i^* \left[ \gamma \frac{z(i)}{|z(i)|} - z(i) \right], \quad z(i) = \mathbf{u}_i w_{i-1}, \quad i \geq 0$$

or via the normalized form

$$w_i = w_{i-1} + \frac{\mu \mathbf{u}_i^*}{\|\mathbf{u}_i\|^2} \left[ \gamma \frac{z(i)}{|z(i)|} - z(i) \right], \quad z(i) = \mathbf{u}_i w_{i-1}, \quad i \geq 0$$

In both cases, we set $w_i = w_{i-1}$ when $z(i) = 0$. 
Algorithm 12.6 (CMA2-2) A weight vector \(w^o\) that solves (cf. Probs. III.18 and III.33):

\[
\min_w \mathbb{E} \left( \gamma - |uw|^2 \right)^2
\]

can be approximated iteratively via the recursion

\[
w_i = w_{i-1} + \mu u_i^* z(i) \left[ \gamma - |z(i)|^2 \right], \quad z(i) = u_i w_{i-1}, \quad i \geq 0
\]
Algorithm 12.7 (RCA) A weight vector \( w^o \) that solves (cf. Probs. III.19 and III.34):

\[
\min_w \mathbb{E} |uw - \gamma \cdot \text{sgn}(uw)|^2
\]

can be approximated iteratively via the recursion

\[
w_i = w_{i-1} + \mu u_i^* [ \gamma \text{sgn}(z(i)) - z(i) ], \quad z(i) = u_i w_{i-1}, \quad i \geq 0
\]
Algorithm 12.8 (MMA) A weight vector $w^o$ that solves (cf. Probs. III.20 and III.35):

$$\min_w \mathbb{E} \left\{ \left[ (\text{Re}(uw))^2 - \gamma \right]^2 + \left[ (\text{Im}(uw))^2 - \gamma \right]^2 \right\}$$

and where Re($\cdot$) and Im($\cdot$) denote the real and imaginary parts of their arguments, can be approximated iteratively via the recursion:

$$\begin{aligned}
z(i) &= u_i w_{i-1} \\
a(i) &= \text{Re}[z(i)] \\
b(i) &= \text{Im}[z(i)] \\
e(i) &= a(i)[\gamma - a^2(i)] + b(i)[\gamma - b^2(i)] \\
w_i &= w_{i-1} + \mu u_i^* e(i)
\end{aligned}$$
TABLE 12.4 A listing of several blind stochastic-gradient algorithms derived in the problems. In all of them, the initial weight estimate is specified at iteration $i = -1$ and, clearly, it cannot be chosen as zero since otherwise the recursions do not update the weight estimate.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Recursion</th>
</tr>
</thead>
</table>
| CMA2-2    | $w_i = w_{i-1} + \mu u_i^* z(i) \left[ \gamma - |z(i)|^2 \right]$  
$z(i) = u_i w_{i-1}$ |
| CMA1-2    | $w_i = w_{i-1} + \mu u_i^* \left[ \gamma \frac{z(i)}{|z(i)|} - z(i) \right]$  
$z(i) = u_i w_{i-1}$ |
| NCMA      | $w_i = w_{i-1} + \frac{\mu u_i^*}{\|u_i\|^2} \left[ \gamma \frac{z(i)}{|z(i)|} - z(i) \right]$  
$z(i) = u_i w_{i-1}$ |
| RCA       | $w_i = w_{i-1} + \mu u_i^* \left[ \gamma \text{csgn}(z(i)) - z(i) \right]$  
$z(i) = u_i w_{i-1}$ |
| MMA       | $w_i = w_{i-1} + \mu u_i^* e(i)$  
$z(i) = u_i w_{i-1}, \ a(i) = \text{Re}(z(i)), \ b(i) = \text{Im}(z(i))$  
$e(i) = a(i) [\gamma - a^2(i)] + j b(i) [\gamma - b^2(i)]$ |
### TABLE 12.5

*Estimated* computational cost per iteration for *real-valued* data in terms of the number of *real* multiplications, *real* additions, *real* divisions, and comparisons with zero (or sign evaluations).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\times$</th>
<th>$+$</th>
<th>$/$</th>
<th>sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMA2-2</td>
<td>$2M + 3$ ((2M + 2))</td>
<td>$2M$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMA1-2</td>
<td>$2M + 2$ ((2M + 1))</td>
<td>$2M$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>NCMA</td>
<td>$3M + 2$ ((3M + 1))</td>
<td>$3M - 1$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>RCA</td>
<td>$2M + 1$ ((2M))</td>
<td>$2M$</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>MMA</td>
<td>$2M + 3$ ((2M + 1))</td>
<td>$2M$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**TABLE 12.6** *Estimated* computational cost per iteration for the case of *complex-valued* data in terms of the number of *real* multiplications, *real* additions, *real* divisions, and comparisons with zero (or sign evaluations).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\times$</th>
<th>$+$</th>
<th>/</th>
<th>sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMA2-2</td>
<td>$8M + 8$\newline$(8M + 6)$</td>
<td>$8M + 2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMA1-2</td>
<td>$8M + 4$\newline$(8M + 2)$</td>
<td>$8M$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>NCMA</td>
<td>$10M + 4$\newline$(10M + 2)$</td>
<td>$10M - 1$</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>RCA</td>
<td>$8M + 2$\newline$(8M)$</td>
<td>$8M$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>MMA</td>
<td>$8M + 6$\newline$(8M + 4)$</td>
<td>$8M$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
14.1 INSTANTANEOUS APPROXIMATION

Just like $\epsilon$-NLMS and $\epsilon$–APA, we again start from the regularized Newton’s recursion (10.8), namely,

$$w_i = w_{i-1} + \mu(i) [\epsilon(i)I + R_u]^{-1} [R_{du} - R_u w_{i-1}]$$  \hspace{1cm} (14.1)$$

and replace $(R_{du} - R_u w_{i-1})$ by the instantaneous approximation $u_i^* [d(i) - u_i w_{i-1}]$. Now, however, we replace $R_u$ by a better estimate for it, which we choose as the exponentially weighted sample average

$$\hat{R}_u = \frac{1}{i+1} \sum_{j=0}^{i} \lambda^{i-j} u_j^* u_j$$

for some scalar $0 \ll \lambda \leq 1$. Assume first that $\lambda = 1$. Then the above expression for $\hat{R}_u$ amounts to averaging all past regressors up to time $i$, namely,

$$\hat{R}_u = \frac{1}{i+1} \sum_{j=0}^{i} u_j^* u_j \quad \text{when} \quad \lambda = 1$$
We further assume that the step-size in (14.1) is chosen as \( \mu(i) = 1/(i+1) \), whereas the regularization factor is chosen as \( \epsilon(i) = \lambda^{i+1} \epsilon/(i+1) \) for \( i \geq 0 \) and for some small positive scalar \( \epsilon \). This choice for \( \epsilon(i) \) is such that regularization disappears as time progresses. With these approximations and choices, the regularized Newton’s recursion (14.1) becomes

\[
w_i = w_{i-1} + \left[ \lambda^{i+1} \epsilon I + \sum_{j=0}^{i} \lambda^{i-j} u_j^* u_j \right]^{-1} u_i^* [d(i) - u_i w_{i-1}] \tag{14.2}
\]

This recursion is inconvenient in its present form since it requires, at each time instant \( i \), that all previous and present data be combined to form the matrix

\[
\Phi_i \triangleq \left( \lambda^{i+1} \epsilon I + \sum_{j=0}^{i} \lambda^{i-j} u_j^* u_j \right) \tag{14.3}
\]

which then needs to be inverted. These two complications (of data storing and matrix inversion) can be alleviated as follows.
Observe from the definition of $\Phi_i$ that it satisfies the recursion

$$
\Phi_i = \lambda \Phi_{i-1} + u_i^* u_i, \quad \Phi_{-1} = \epsilon I
$$

(14.4)

Let $P_i = \Phi_i^{-1}$. Then applying the matrix inversion formula (5.4) to (14.4) gives

$$
P_i = \lambda^{-1} \left[ P_{i-1} - \frac{\lambda^{-1} P_{i-1} u_i^* u_i P_{i-1}}{1 + \lambda^{-1} u_i P_{i-1} u_i^*} \right], \quad P_{-1} = \epsilon^{-1} I
$$

(14.5)

This recursion shows that the update from $P_{i-1}$ to $P_i$ requires only knowledge of the most recent regressor $u_i$. In this way, at each time instant $i$, the algorithm only needs to have access to the data $\{w_{i-1}, d(i), u_i, P_{i-1}\}$ in order to determine $\{w_i, P_i\}$. The matrix $P_{i-1}$ essentially summarizes the information from all previous regressors.
Algorithm 14.1 (RLS algorithm) Consider a zero-mean random variable $d$ with realizations $\{d(0), d(1), \ldots\}$, and a zero-mean random row vector $u$ with realizations $\{u_0, u_1, \ldots\}$. The weight vector $w^*$ that solves

$$\min_w \mathbb{E} |d - uw|^2$$

can be approximated iteratively via the recursion

$$P_i = \lambda^{-1} \left[ P_{i-1} - \frac{\lambda^{-1} P_{i-1} u_i^* u_i P_{i-1}}{1 + \lambda^{-1} u_i P_{i-1} u_i^*} \right]$$

$$w_i = w_{i-1} + P_i u_i^* [d(i) - u_i w_{i-1}], \ i \geq 0$$

with initial condition $P_{-1} = \epsilon^{-1} I$ and where $0 \ll \lambda \leq 1$. 
### TABLE 14.1

Estimated computational cost of the RLS algorithm per iteration for real-valued data in terms of the number of real multiplications, real additions, and real divisions.

<table>
<thead>
<tr>
<th>Term</th>
<th>$\times$</th>
<th>$+$</th>
<th>$/$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_i w_{i-1}$</td>
<td></td>
<td>$M$</td>
<td>$M - 1$</td>
</tr>
<tr>
<td>$d(i) - u_i w_{i-1}$</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$\lambda^{-1} u_i^*$</td>
<td></td>
<td>$M$</td>
<td></td>
</tr>
<tr>
<td>$P_{i-1}(\lambda^{-1} u_i^*)$</td>
<td></td>
<td>$M^2$</td>
<td>$M(M-1)$</td>
</tr>
<tr>
<td>$u_i P_{i-1}(\lambda^{-1} u_i^*)$</td>
<td></td>
<td>$M$</td>
<td>$M - 1$</td>
</tr>
<tr>
<td>$1 + u_i P_{i-1}(\lambda^{-1} u_i^*)$</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$1/[1 + u_i P_{i-1}(\lambda^{-1} u_i^*)]$</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$(\lambda^{-1} u_i P_{i-1} u_i^<em>) \cdot \frac{1}{1 + u_i P_{i-1}(\lambda^{-1} u_i^</em>)}$</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$(\lambda^{-1} P_{i-1} u_i^<em>) \times \frac{\lambda^{-1} u_i P_{i-1} u_i^</em>}{1 + \lambda^{-1} u_i P_{i-1} u_i^*}$</td>
<td></td>
<td>$M$</td>
<td></td>
</tr>
<tr>
<td>$P_i u_i^*$</td>
<td></td>
<td></td>
<td>$M$</td>
</tr>
<tr>
<td>$P_i u_i^*[d(i) - u_i w_{i-1}]$</td>
<td></td>
<td>$M$</td>
<td></td>
</tr>
<tr>
<td>$w_i$</td>
<td></td>
<td>$M$</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL per iteration</strong></td>
<td>$M^2 + 5M + 1$</td>
<td>$M^2 + 3M$</td>
<td>1</td>
</tr>
</tbody>
</table>
## COMPLEXITY (COMPLEX DATA)

**TABLE 14.2** *Estimated* computational cost of the RLS algorithm per iteration for *complex-valued* data in terms of the number of *real* multiplications, *real* additions, and *real* divisions.

<table>
<thead>
<tr>
<th>Term</th>
<th>×</th>
<th>+</th>
<th>/</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_i w_{i-1}$</td>
<td>$4M$</td>
<td>$4M - 2$</td>
<td></td>
</tr>
<tr>
<td>$d(i) - u_i w_{i-1}$</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$\lambda^{-1} u_i^*$</td>
<td>$2M$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_{i-1}(\lambda^{-1} u_i^*)$</td>
<td>$4M^2$</td>
<td>$M(4M - 2)$</td>
<td></td>
</tr>
<tr>
<td>$u_i P_{i-1}(\lambda^{-1} u_i^*)$</td>
<td>$4M$</td>
<td>$4M - 2$</td>
<td></td>
</tr>
<tr>
<td>$1 + u_i P_{i-1}(\lambda^{-1} u_i^*)$</td>
<td>$4M$</td>
<td>$4M - 2$</td>
<td></td>
</tr>
<tr>
<td>$1/[1 + u_i P_{i-1}(\lambda^{-1} u_i^*)]$</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$(\lambda^{-1} u_i P_{i-1} u_i^<em>) \cdot \frac{1}{1 + u_i P_{i-1}(\lambda^{-1} u_i^</em>)}$</td>
<td>$1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(\lambda^{-1} P_{i-1} u_i^<em>) \times \frac{\lambda^{-1} u_i P_{i-1} u_i^</em>}{1 + \lambda^{-1} u_i P_{i-1} u_i^*}$</td>
<td>$2M$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_i u_i^*$</td>
<td></td>
<td>$2M$</td>
<td></td>
</tr>
<tr>
<td>$P_i u_i^*[d(i) - u_i w_{i-1}]$</td>
<td>$4M$</td>
<td>$2M$</td>
<td></td>
</tr>
<tr>
<td>$w_i$</td>
<td>$2M$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL per iteration</strong></td>
<td>$4M^2 + 16M + 1$</td>
<td>$4M^2 + 12M - 1$</td>
<td>1</td>
</tr>
</tbody>
</table>
The LMS and $\epsilon$-NLMS algorithms were obtained by using simple instantaneous approximations for the covariance and cross-covariance quantities $\{R_{du}, R_u\}$. More involved algorithms, with better performance but at increased computational costs, can be obtained by resorting to more sophisticated approximations for $\{R_{du}, R_u\}$. We illustrate this situation by motivating the so-called affine projection algorithm.

## 13.1 Instantaneous Approximation

Just like $\epsilon$-NLMS, we again start from the regularized Newton’s recursion (10.8), namely,

$$w_i = w_{i-1} + \mu [\epsilon' I + R_u]^{-1} [R_{du} - R_u w_{i-1}]$$

(13.1)

albeit with a fixed step-size $\mu$ and a fixed regularization parameter $\epsilon'$. Now, however, we shall employ a better approximation for both the covariance matrix, $R_u$, and the cross-covariance vector, $R_{du}$. 
Specifically, we choose a positive integer $K$ (usually $K \leq M$, where $M \times 1$ is the size of the weight vector) and replace $\{R_{du}, R_u\}$ by the following instantaneous approximations:

$$
\hat{R}_u = \frac{1}{K} \left( \sum_{j=i-K+1}^{i} u_j^* u_j \right), \quad \hat{R}_{du} = \frac{1}{K} \left( \sum_{j=i-K+1}^{i} d(j) u_j^* \right)
$$

In other words, at each iteration $i$, we use the $K$ most recent regressors and the $K$ most recent observations,

$$
\{u_i, u_{i-1}, \ldots, u_{i-K+1}\}, \quad \{d(i), d(i-1), \ldots, d(i-K+1)\}
$$

to compute the approximate values for $\{R_u, R_{du}\}$. 
Let $\epsilon = \epsilon' / K$. If we introduce the $K \times M$ block data matrix

$$
U_i \triangleq \begin{bmatrix}
  u_i \\
  u_{i-1} \\
  \vdots \\
  u_{i-K+1}
\end{bmatrix} \quad (K \times M)
$$

(13.2)

and the $K \times 1$ data vector

$$
d_i = \begin{bmatrix}
  d(i) \\
  d(i - 1) \\
  \vdots \\
  d(i - K + 1)
\end{bmatrix} \quad (K \times 1)
$$

(13.3)

then we can express $\{\hat{R}_u, \hat{R}_{du}\}$ more compactly as

$$
\hat{R}_u = \frac{1}{K} U_i^* U_i, \quad \hat{R}_{du} = \frac{1}{K} U_i^* d_i
$$
so that Newton’s recursion (13.1) becomes

$$w_i = w_{i-1} + \mu (\epsilon I + U_i^* U_i)^{-1} U_i^* [d_i - U_i w_{i-1}]$$  \hspace{1cm} (13.4)$$

Although $U_i^* U_i$ is singular when $K \leq M$, the term $\epsilon I$ guarantees the invertibility of $\epsilon I + U_i^* U_i$.

Recursion (13.4) requires the inversion of the $M \times M$ matrix $(\epsilon I + U_i^* U_i)$ at each iteration. Alternatively, we can invoke the matrix inversion formula (5.4) to verify that

$$(\epsilon I + U_i^* U_i)^{-1} U_i^* = U_i^* (\epsilon I + U_i U_i^*)^{-1}$$

in which case (13.4) becomes

$$w_i = w_{i-1} + \mu U_i^* (\epsilon I + U_i U_i^*)^{-1} [d_i - U_i w_{i-1}]$$  \hspace{1cm} (13.5)$$

This form of the algorithm requires inverting the (usually smaller) $K \times K$ matrix $(\epsilon I + U_i U_i^*)$ at each iteration; this form of the recursion is also useful even when $\epsilon = 0$ since $U_i U_i^*$ is generally invertible when $K \leq M$. Recursion (13.5) is what is known as the affine projection algorithm.
Algorithm 13.1 (ε-APA algorithm) Consider a zero-mean random variable $d$ with realizations $\{d(0), d(1), \ldots\}$, and a zero-mean random row vector $u$ with realizations $\{u_0, u_1, \ldots\}$. The optimal weight vector $w^*$ that solves

$$\min_w \mathbb{E} |d - uw|^2$$

can be approximated iteratively via the recursion

$$w_i = w_{i-1} + \mu U_i^* (\epsilon I + U_i U_i^*)^{-1} [d_i - U_i w_{i-1}]$$

where $\{U_i, d_i\}$ are defined by (13.2)–(13.3) and $K$ is a positive integer that denotes the filter order (usually $K \leq M$).
### Table 13.1

Estimated computational cost of $\epsilon$-APA per iteration for *real-valued* data in terms of the number of *real* multiplications and *real* additions.

<table>
<thead>
<tr>
<th>Term</th>
<th>(\times)</th>
<th>(+)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(U_iw_{i-1})</td>
<td>(KM)</td>
<td>(K(M - 1))</td>
</tr>
<tr>
<td>(d_i - U_iw_{i-1})</td>
<td></td>
<td>(K)</td>
</tr>
<tr>
<td>(U_iU_i^*)</td>
<td>(K^2M)</td>
<td>(K^2(M - 1))</td>
</tr>
<tr>
<td>(\epsilon I + U_iU_i^*)</td>
<td></td>
<td>(K)</td>
</tr>
<tr>
<td>((\epsilon I + U_iU_i^*)^{-1})</td>
<td>(K^3)</td>
<td>(K^3)</td>
</tr>
<tr>
<td>((\epsilon I + U_iU_i^*)^{-1}[d_i - U_iw_{i-1}])</td>
<td></td>
<td>(K(K - 1))</td>
</tr>
<tr>
<td>(U_i^<em>(\epsilon I + U_iU_i^</em>)^{-1}[d_i - U_iw_{i-1}])</td>
<td>(KM)</td>
<td>((K - 1)M)</td>
</tr>
<tr>
<td>(w_i)</td>
<td></td>
<td>(M)</td>
</tr>
<tr>
<td><strong>TOTAL per iteration</strong></td>
<td>((K^2 + 2K)M)</td>
<td>((K^2 + 2K)M)</td>
</tr>
<tr>
<td></td>
<td>(K^3 + K)</td>
<td>(K^3 + K^2)</td>
</tr>
</tbody>
</table>
TABLE 13.2  Estimated computational cost of $\epsilon$-APA algorithm per iteration for complex-valued data in terms of the number of real multiplications and real additions.

<table>
<thead>
<tr>
<th>Term</th>
<th>$\times$</th>
<th>$+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U_i w_{i-1}$</td>
<td>$4KM$</td>
<td>$2K(2M - 1)$</td>
</tr>
<tr>
<td>$d_i - U_i w_{i-1}$</td>
<td></td>
<td>$2K$</td>
</tr>
<tr>
<td>$U_i U_i^*$</td>
<td>$4K^2M$</td>
<td>$2K^2(2M - 1)$</td>
</tr>
<tr>
<td>$\epsilon I + U_i U_i^*$</td>
<td></td>
<td>$2K$</td>
</tr>
<tr>
<td>$(\epsilon I + U_i U_i^*)^{-1}$</td>
<td>$4K^3$</td>
<td>$4K^3$</td>
</tr>
<tr>
<td>$(\epsilon I + U_i U_i^*)^{-1}[d_i - U_i w_{i-1}]$</td>
<td>$4K^2$</td>
<td>$2K(2K - 1)$</td>
</tr>
<tr>
<td>$U_i^* (\epsilon I + U_i U_i^*)^{-1}[d_i - U_i w_{i-1}]$</td>
<td>$4KM$</td>
<td>$2(2K - 1)M$</td>
</tr>
<tr>
<td>$w_i$</td>
<td></td>
<td>$2M$</td>
</tr>
<tr>
<td><strong>TOTAL per iteration</strong></td>
<td>$4(K^2 + 2K)M + 4K^3 + 4K$</td>
<td>$4(K^2 + 2K)M + 4K^3 + 2K^2$</td>
</tr>
</tbody>
</table>
Finally, it is worth mentioning that the APA form (13.5), with $\epsilon = 0$, can be rewritten in an equivalent form in terms of orthogonalized regressors $\{\tilde{u}_i\}$ that are obtained from the original regressors $\{u_i\}$ via a Gram-Schmidt orthogonalization procedure (see Prob. III.43). The details are carried out in Prob. III.44 and they lead to the following statement.

**Algorithm 13.2 (APA with orthogonal update factors)** Consider the APA update

$$w_i = w_{i-1} + \mu U_i^* (U_i U_i^*)^{-1} [d_i - U_i w_{i-1}]$$

with a full rank regression matrix $U_i$ and $K \leq M$. This update can be equivalently implemented as follows. For each iteration $i$, perform the following steps. Start with $w_{i-1}^{(0)} = w_{i-1}$ and repeat for $k = 0, 1, \ldots, K - 1$:

$$\tilde{u}_{i-k} = u_{i-k} \left( I - \sum_{j=1}^{k} \frac{\tilde{u}_{i-k+j}^* \tilde{u}_{i-k+j}}{\| \tilde{u}_{i-j+k} \|^2} \right)$$

$$e^{(k)}(i - k) = d(i - k) - u_{i-k} w_{i-1}^{(k)}$$

$$w_{i-1}^{(k+1)} = w_{i-1}^{(k)} + \frac{\tilde{u}_{i-k}}{\| \tilde{u}_{i-k} \|^2} e^{(k)}(i - k)$$

Then set

$$w_i = (1 - \mu) w_{i-1} + \mu w_{i-1}^{(K)}$$
Project III.1 (Constant-modulus criterion) Refer to Prob. III.18, where we introduced the cost function $J(w) = E \left( \gamma - |uw|^2 \right)^2$, for a given positive constant $\gamma$. This cost arises in the context of blind equalization where it is used to derive blind adaptive filters — see, e.g., Prob. III.33. In this project, we use $J(w)$ to highlight some of the issues that arise in the design of steepest-descent methods.

(a) Assume first that $w$ is one-dimensional and $u$ is scalar-valued with variance $\sigma_u^2 = E|u|^2$. Assume further that $u$ and $w$ are real-valued and that $u$ is Gaussian so that its fourth moment is given by $E u^4 = 3\sigma_u^4$. Verify that, under these conditions, the cost function $J(w)$ evaluates to $J(w) = \gamma^2 - 2\gamma\sigma_u^2 w^2 + 3\sigma_u^4 w^4$. Conclude that $J(w)$ has a local maximum at $w = 0$ and two global minima at $w^\circ = \pm \sqrt{\gamma / 3\sigma_u^2}$. Plot $J(w)$ and determine the values of $w^\circ$ using $\gamma = 1$ and $\sigma_u^2 = 0.5$. Find also the corresponding minimum cost.

(b) Argue that a steepest-descent method for minimizing $J(w)$ can be taken as

$$w(i) = w(i - 1) + 4\mu\sigma_u^2 w(i - 1)[\gamma - 3\sigma_u^2 w^2(i - 1)], \quad w(-1) = \text{initial guess}$$

with scalar estimates $\{w(i)\}$. Does this method converge to a global minimum if the initial weight guess is chosen as $w(-1) = 0$? Why?
(c) Simulate the steepest-descent recursion of part (b) in the following cases and comment on its behavior in each case:

1. \( \mu = 0.2 \) and \( w(-1) = 0.3 \) or \( w(-1) = -0.3 \).

2. \( \mu = 0.6 \) and \( w(-1) = 0.3 \) or \( w(-1) = -0.3 \).

3. \( \mu = 1 \) and \( w(-1) = -0.2 \).

In each case, plot the evolution of \( w(i) \) as a function of time. Plot also the graph \( J[w(i)] \times w(i) \).

(d) Now consider the setting of part (b) in Prob. III.18 where \( w \) is two-dimensional.

(d.1) Generate a plot of the contour curves of the cost function for \( \gamma = 1 \), i.e.,

\[
J(w) = \gamma^2 + 2(\alpha^2 + 2\beta^2)^2 - 2\gamma(\alpha^2 + 2\beta^2)
\]

(d.2) Choose \( \mu = 0.02 \) and apply the resulting algorithm,

\[
\begin{bmatrix}
\alpha(i) \\
\beta(i)
\end{bmatrix} =
\begin{bmatrix}
\alpha(i-1) \\
\beta(i-1)
\end{bmatrix} + \mu(\gamma - 2|\alpha(i-1)|^2 - 4|\beta(i-1)|^2)
\begin{bmatrix}
\alpha(i-1) \\
2\beta(i-1)
\end{bmatrix}
\]

starting from the initial conditions \( w_{-1} = \text{col}\{1, -0.25\} \) and \( w_{-1} = \text{col}\{-1, 0.25\} \).

Iterate over a period of length \( N = 1000 \). Plot the trajectory of the weight estimates superimposed on the contour curves of \( J(w) \).
Project III.1 (Constant-modulus criterion) The programs that solve this project are the following.

1. `partA.m` This program prints the locations of the global minima as well as the minimum value of the cost function at these minima. It also plots $J(w)$.

2. `partC.m` This program generates three plots — see Figs. 1–3. Each plot shows the evolution of $w(i)$ as a function of time, as well as the evolution of $J[w(i)]$.

3. `partD.m` This program generates a plot showing the contour curves of $J(w)$ superimposed on the trajectories of the weight estimates — see Fig. 4.
**Figure III.1.** The top plot shows the evolution of the scalar weight \( w(i) \) from two different initial conditions and for \( \mu = 0.2 \); the filter is seen to converge to the global minimum that is closest to the initial condition. The bottom plot shows the evolution of \( J[w(i)] \) superimposed on the cost function \( J(w) \) for both initial conditions.
Figure III.2. The top plot shows the evolution of $w(i)$ starting from two different initial conditions and for $\mu = 0.6$. Due to the larger value of the step-size, the filter is seen to oscillate around the global minimum that is closest to the initial condition. The bottom plot shows the evolution of $J[w(i)]$ superimposed on the cost function $J(w)$ for both initial conditions.
Figure III.3. The top plot shows the evolution of $w(i)$ starting from two different initial conditions and for a large step-size, $\mu = 1$. In this case, the filter does not converge.
Figure III.4. A plot of the contour curves of $J(w)$, along with the trajectories of the resulting weight estimates starting from two different initial conditions (one is on the left and the other is on the right). Here $w = \text{col}\{\alpha, \beta\}$. 
Project III.2 (Constant-modulus algorithm) In Prob. III.18 we introduced the constant-modulus criterion

$$\min_w E \left( \gamma - |uw|^2 \right)^2$$

and developed a steepest-descent method for it, namely

$$w_i = w_{i-1} + \mu \left( \gamma R_u w_{i-1} - E \left[ u^* u w_{i-1} |uw_{i-1}|^2 \right] \right)$$

We reconsidered this method in Prob. III.33 and derived the corresponding stochastic gradient approximation, known as CMA2-2, namely,

$$w_i = w_{i-1} + \mu u_i^* z(i) \left[ \gamma - |z(i)|^2 \right], \quad z(i) = u_i w_{i-1}$$

In this project we compare the performance of the steepest-descent method and the CMA2-2 recursion. For this purpose, we set $\gamma = 1$ and let $u$ be a 2-dimensional circular Gaussian random vector with covariance matrix $R_u = \text{diag}\{1,0\}$. We showed in Prob. III.18 that, under these conditions, the steepest-descent method collapses to the form

$$\begin{bmatrix} \alpha(i) \\ \beta(i) \end{bmatrix} = \begin{bmatrix} \alpha(i-1) \\ \beta(i-1) \end{bmatrix} + \mu \left( \gamma - 2|\alpha(i-1)|^2 - 4|\beta(i-1)|^2 \right) \begin{bmatrix} \alpha(i-1) \\ 2\beta(i-1) \end{bmatrix}$$

where $w_i = \text{col}\{\alpha(i), \beta(i)\}$. In addition, we showed in Prob. III.18 that the corresponding cost function evaluates to

$$J(w_i) = \gamma^2 + 2(|\alpha(i)|^2 + 2|\beta(i)|^2)^2 - 2\gamma(|\alpha(i)|^2 + 2|\beta(i)|^2)$$
(a) Plot the learning curve $J(i) = J(w_{i-1})$ for the steepest-descent method; here $w_{i-1}$ is the weight estimate that results from the steepest-descent recursion. Plot also an ensemble-average learning curve $\widehat{J}(i)$ for the CMA2-2 algorithm that is generated as follows:

$$\widehat{J}(i) = \frac{1}{L} \sum_{j=1}^{L} \left( \gamma - |z^{(j)}(i)|^2 \right)^2, \quad i \geq 0$$

with the data generated from $L$ experiments, say of duration $N$ each. Choose $L = 200$ and $N = 500$. Use $\mu = 0.001$ and $w_{-1} = \text{col}\{-0.8, 0.8\}$. Remark. We are assuming complex-valued regressors $u$, with the two entries of $u$ having variances $\{1, 2\}$. In order to generate such regressors, create four separate real-valued zero-mean and independent Gaussian numbers $\{a, b, p, q\}$ with variances $\{1/2, 1/2, 1, 1\}$, respectively, and then set $u = [a + jb \quad p + jq]$.

(b) Plot the contour curves of the cost function $J(w) = \mathbb{E} (\gamma - |uw|^2)^2$ superimposed on the four weight trajectories that are generated by CMA2-2 for the following four choices of initial conditions:

$$w_{-1} \in \left\{ \begin{bmatrix} 0.8 & 0.8 \\ -0.8 & 0.8 \end{bmatrix}, \begin{bmatrix} 0.8 & -0.8 \\ -0.8 & 0.8 \end{bmatrix} \right\}$$

Use $\mu = 0.001$ and $N = 1000$ iterations in each case. Print the final value of the weight vector estimate in each case. Show also the trajectories of the weight estimates that are generated by the steepest-descent method. Remark. Although the weight estimates in the steepest-descent method are always real-valued for a real-valued initial condition $w_{-1}$, the same is not true for CMA2-2. However, the imaginary parts of the successive weight estimates will be small compared to the real parts. For this reason, when plotting the weight trajectories, we shall ignore the imaginary parts.
Project III.2 (Constant-modulus algorithm) The programs that solve this project are the following.

1. partA.m This program generates a plot showing the learning curve of the steepest-descent method and an ensemble-average learning curve for CMA2-2. The program allows the user to modify the values of $\gamma$, $w_{-1}$, $L$, and $N$. A typical output is shown in Fig. 5.

![Theoretical and ensemble-average learning curves](image)

Figure III.5. Learning curve of the steepest-descent method corresponding to the cost function $J(w) = \mathbb{E} \left( \gamma - |uw|^2 \right)^2$, along with an ensemble-average curve generated by running CMA2-2 and averaging over 200 experiments.
2. `partB.m` This program generates two plots. One plot shows the contour curves of the cost function along with the four weight-vector trajectories generated by CMA2-2 (see Fig. 6). The second plot shows the same contour curves with the weight-vector trajectories generated by steepest-descent (see Fig. 7). The program allows the user to modify the values of $\gamma$, $w_{-1}$, and $N$.

Figure III.6. Contour curves of the cost function $J(w) = E (\gamma - |w|^2)^2$, along with the trajectories generated by the steepest-descent method from four distinct initial conditions.
Figure III.7. Contour curves of the cost function $J(w) = \text{E}\left(\gamma - |uw|^2\right)^2$, along with the trajectories generated by CMA2-2 from four distinct initial conditions.
**Project III.3 (Adaptive channel equalization)**  In Computer Projects II.1 and II.3 we dealt with the design of minimum mean-square error equalizers. In this project we examine the design of *adaptive* equalizers. We consider the same channel of Computer Project II.3, namely,

\[ C(z) = 0.5 + 1.2z^{-1} + 1.5z^{-2} - z^{-3} \]

and proceed to design an adaptive linear equalizer for it. The equalizer structure is shown in Fig. III.2. Symbols \( \{ s(i) \} \) are transmitted through the channel and corrupted by additive complex-valued white noise \( \{ v(i) \} \). The received signal \( \{ u(i) \} \) is processed by the FIR equalizer to generate estimates \( \{ \hat{s}(i - \Delta) \} \), which are fed into a decision device. The equalizer possesses two modes of operation: a training mode during which a delayed replica of the input sequence is used as a reference sequence, and a decision-directed mode during which the output of the decision-device replaces the reference sequence. The input sequence \( \{ s(i) \} \) is chosen from a quadrature-amplitude modulation (QAM) constellation (e.g., 4-QAM, 16-QAM, 64-QAM, or 256-QAM).
FIGURE III.2 An adaptive linear equalizer operating in two modes: training mode and decision-direction mode.
(a) Write a program that trains the adaptive filter with 500 symbols from a QPSK constellation, followed by decision-directed operation during 5000 symbols from a 16-QAM constellation. Choose the noise variance $\sigma_v^2$ in order to enforce an SNR level of 30 dB at the input of the equalizer. Note that symbols chosen from QAM constellations do not have unit variance. For this reason, the noise variance needs to be adjusted properly for different QAM orders in order to enforce the desired SNR level — see Prob. II.16. Choose $\Delta = 15$ and equalizer length $L = 35$. Use $\epsilon$-NLMS to train the equalizer with step-size $\mu = 0.4$ and $\epsilon = 10^{-6}$. Plot the scatter diagrams of $\{s(i), u(i), \hat{s}(i - \Delta)\}$.

(b) For the same setting as part (a), plot and compare the scatter diagrams that would result at the output of the equalizer if training is performed only for 150, 300, and 500 iterations. Repeat the simulations using LMS with $\mu = 0.001$.

(c) Now assume the transmitted data are generated from a 256-QAM constellation rather than a 16-QAM constellation. Plot the scatter diagrams of the output of the equalizer, when trained with $\epsilon$-NLMS using 500 training symbols.

(d) Generate symbol-error-rate (SER) curves versus signal-to-noise ratio (SNR) at the input of the equalizer for $(4, 16, 64, 256)$-QAM data. Let the SNR vary between 5 dB and 30 dB in increments of 1 dB.

(e) Continue with SNR at 30 dB. Design a decision-feedback equalizer with $L = 10$ feedforward taps and $Q = 2$ feedback taps. Use $\Delta = 7$ and plot the resulting scatter diagram of the output
of the equalizer. Repeat for $L = 20$, $Q = 2$ and $\Delta = 10$. In both cases, choose the transmitted data from a 64-QAM constellation.

(f) Generate SER curves versus SNR at the input of the DFE for $(4, 16, 64, 256)$-QAM data. Let the SNR vary between 5 dB and 30 dB. Compare the performance of the DFE with that of the linear equalizer of part (d).

(g) Load the file channel, which contains the impulse response sequence of a more challenging channel with spectral nulls. Set the SNR level at the input of the equalizer to 40 dB and select a linear equalizer structure with 55 taps. Set also the delay at $\Delta = 30$. Train the equalizer using $\epsilon$—NLMS for 2000 iterations before switching to decision-directed operation. Plot the resulting scatter diagram of the output of the equalizer. Now train it again using RLS for 100 iterations before switching to decision-directed operation, and plot the resulting scatter diagram. Compare both diagrams.
Project III.3 (Adaptive channel equalization) The programs that solve this project are the following.

1. partA.m This program solves part (a) and generates two plots showing the scatter diagrams of the transmitted, received, and estimated sequences \( \{s(i), u(i), \hat{s}(i - \Delta)\} \) during training and decision-directed modes of operation. A typical output is shown in Figs. 8–9. No erroneous decisions were observed in this simulation.

![Training sequence](image1)

![Transmitted sequence](image2)

**Figure III.8.** Scatter diagrams of the QPSK training sequence (left) and the 16-QAM transmitted data (right).
Figure III.9. Scatter diagrams of the received sequence (*left*) and the output of the equalizer (*right*). Observe how the equalizer outputs are centered around the symbols of the 16-QAM constellation.
2. **partB.m** This program solves part (b) and generates a plot showing the scatter diagrams of the output of the equalizer for three lengths of training period (150, 300, and 500), and for two adaptation schemes (LMS and ε-NLMS), as shown in Fig. 10. No erroneous decisions were observed in this simulation in the last two cases of ε-NLMS, while 1 error was observed in the last case for LMS. Observe how the performance of LMS is inferior. If the training period is increased further, the performance of LMS can be improved.

3. **partC.m** This program solves part (c) and generates a plot showing the scatter diagram of the output of the equalizer when the transmitted data is selected from a 256-QAM constellation and the equalizer is trained using ε-NLMS with 500 QPSK symbols. A typical output is shown in Fig. 11. No erroneous decisions were observed in this simulation.

4. **partD.m** This program solves part (d) and generates a plot of the SER curve versus SNR for several QAM constellations. A typical output is shown in Fig. 12. Observe how, for a fixed SNR, the probability of error increases as the order of the constellation increases.

5. **partE.m** This program solves part (e) and generates two plots of the scatter diagrams of the input and output of the equalizer for two different configurations. A typical output is shown in Fig. 13. No erroneous decisions were observed in both simulations.

6. **partF.m** This program solves part (f) and generates a plot of the SER curve versus SNR for several QAM constellations. A typical output is shown in Fig. 14.

7. **partG.m** This program solves part (g). Figure 15 shows the impulse and frequency responses of the channel, and it is seen that the channel has a pronounced spectral null; a fact that makes the equalization task more challenging. Figure 16 shows the frequency and impulse responses of the channel, the RLS
equalizer at the end of the adaptation process, and of the convolution of the channel and equalizer impulse response sequences (we are only plotting the real parts of the impulse response sequences). Observe how the combined frequency response still exhibits a spectral null; the equalizer has not been successful in fully removing the null from the channel spectrum (i.e., it has not been successful in flattening the channel). The program also generates a plot of the scatter diagrams of the transmitted 4-QAM data and the outputs of the linear equalizer for two cases. In one case, the equalizer is trained with RLS for 100 iterations, and in the second case the equalizer is trained with $\epsilon$-NLMS for 2000 iterations. A typical output is shown in Fig. 17. Observe how $\epsilon$-NLMS fails for this channel while no erroneous decisions were observed for RLS in this simulation. The performance of RLS can be improved by increasing the length of the equalizer.
Figure III.10. Scatter diagrams of the signal at the output of the equalizer for three different number of training symbols, 150, 300, and 500, and for both algorithms, LMS (top row) and $\epsilon$-NLMS (bottom row).
Figure III.11. Scatter diagram of the signal at the output of the equalizer. The equalizer is trained with 500 QPSK symbols using $\epsilon$-NLMS and the transmitted data during the decision-directed mode is from a 256-QAM constellation.
Figure III.12. Plots of the SER as a function of the SNR at the input of the equalizer and the order of the QAM constellation. The adaptive filter is trained with $\epsilon$-NLMS using 500 QPSK training symbols.
Figure III.13. Scatter diagrams of the input and output of the decision-feedback equalizer for 64-QAM transmissions. The top row corresponds to a DFE with 10 feedforward taps, 2 feedback taps, and delay $\Delta = 7$. The lower plot corresponds to a DFE with 20 feedforward taps, 2 feedback taps, and delay $\Delta = 10$. 
Figure III.14. A plot of the SER as a function of the SNR at the input of the equalizer and the order of the QAM constellation. The DFE is trained with ε-NLMS using 500 QPSK training symbols.
Figure III.15. Impulse (left) and frequency (right) responses of channel.
Figure III.16. Frequency and impulse responses of the channel (left), the RLS equalizer at the end of the adaptation process (center), and of the convolution of the channel and equalizer (right).
Figure III.17. Scatter diagrams of the transmitted sequence and the outputs of the equalizer for two training algorithms: RLS (center) and $\epsilon$-NLMS (far right). The former is trained with 100 QPSK symbols while the latter is trained with 2000 QPSK symbols.
Project III.4 (Blind adaptive equalization) We consider the same channel used in Computer Project III.3,

\[ C(z) = 0.5 + 1.2z^{-1} + 1.5z^{-2} - z^{-3} \]

and proceed to design blind adaptive equalizers for it. The equalizer structure is shown in Fig. III.3. Symbols \( \{s(i)\} \) are transmitted through the channel and corrupted by additive complex-valued white noise \( \{v(i)\} \). The received signal \( \{u(i)\} \) is processed by a linear equalizer, whose outputs \( \{z(i)\} \) are fed into a decision device to generate \( \{\hat{s}(i - \Delta)\} \). These signals are delayed decisions and the value of \( \Delta \) is determined by the delay that the signals undergo when travelling through the channel and the equalizer. In this project, the equalizer is supposed to operate blindly, i.e., without a reference sequence and therefore without a training mode. Most blind algorithms use the output of the equalizer, \( z(i) \), to generate an error signal \( e(i) \), which is used to adapt the equalizer coefficients according to the rule

\[ w_i = w_{i-1} + \mu u_i^* e(i) \]

where \( u_i \) is the regressor at time \( i \). Some blind algorithms use the output of the decision device, \( \hat{s}(i - \Delta) \), to evaluate \( e(i) \) (e.g., the “stop-and-go” variant described in part (d) below).
FIGURE III.3  A representation of a general structure for blind adaptive equalization.
(a) Write a program that transmits 10000 QPSK symbols through the channel and trains a 35-tap equalizer using CMA2-2,

\[ w_i = w_{i-1} + \mu u_i^* z(i) \left[ \gamma - |z(i)|^2 \right], \quad z(i) = u_i w_{i-1} \]

Choose the value of \( \gamma \) as \( E|s|^4/E|s|^2 \), which is in terms of the second and fourth-order moments of the symbol constellation. For QPSK data, \( \gamma = 1 \). Set the SNR level at the input of the equalizer to 30 dB and use \( \mu = 0.001 \). Plot the impulse responses of the channel, the equalizer, and the combination channel-equalizer at the end of the adaptation process. How much delay does the signal undergo in travelling through the channel and the equalizer? Plot the scatter diagrams of the transmitted sequence, the received sequence, and the sequence at the output of the equalizer. Ignore the first 2000 transmissions and count the number of erroneous decisions in the remaining decisions (you should take into account the delay introduced by the channel-equalizer system).

(b) Repeat the simulations of part (a) using 16-QAM data, for which \( \gamma = 13.2 \) (verify this value). Use \( \mu = 0.000001 \). Run the simulation for 30000 symbols and ignore the first 15000 for error calculation. These numbers are larger than in the QPSK case of part (a), and the step-size is also significantly smaller, since the equalizer converges at a slower pace now.

(c) Repeat the simulations of part (b) using the multi-modulus algorithm (MMA) of Prob. III.35.
(d) Repeat the simulations of part (b) using the following three additional blind adaptive algorithms:

(d.1) CMA1-2 from Prob. III.36, where $\gamma$ is now chosen as $\gamma = E|s|^2/E|s|$ in terms of the second moment of the symbol constellation divided by the mean of the magnitude of the symbols. For 16-QAM data we find $\gamma = 3.3385$ (verify this value). Use $\mu = 0.0001$ and increase the SNR level at the input of the equalizer to 60 dB. Simulate for 30000 iterations and plot the scatter diagram of the output of the equalizer after ignoring the first 15000 samples.

(d.2) The reduced constellation algorithm (RCA) of Prob. III.34, where $\gamma$ is now chosen as $\gamma = E|s|^2/E|s|_1$, in terms of the second moment of the symbol constellation divided by the mean of the $l_1$-norm of the symbols (remember that the $l_1$ norm of a complex number amounts to adding the absolute values of its real and imaginary parts, as in Prob. III.14). For 16-QAM data we find $\gamma = 2.5$ (verify this value). Use the same step-size and same simulation duration as part (d.1).
(d.3) The “stop-and-go” algorithm is a blind adaptation scheme that employs the decision-directed error,

\[ e_d(i) = \hat{s}(i - \Delta) - z(i) \]

It also employs a flag to indicate how reliable \( e_d(i) \) is. The flag is set by comparing \( e_d(i) \) to another error signal, say the one used by the RCA recursion,

\[ e_s(i) = \gamma \text{csgn}(z(i)) - z(i), \quad \gamma = \frac{E|s|^2}{E|s|_1} \]

If the complex signs of \( \{e_d(i), e_s(i)\} \) differ, then \( e_d(i) \) is assumed unreliable and the flag is set to zero (see Picchi and Prati (1987)). More explicitly, the stop-and-go recursion takes the form:

\[
\begin{align*}
  z(i) &= u_i w_{i-1} \\
  e_d(i) &= \hat{s}(i - \Delta) - z(i) \quad \text{(decision-directed error)} \\
  e_s(i) &= \gamma \text{csgn}(z(i)) - z(i) \quad \text{(RCA error)} \\
  f(i) &= \begin{cases} 
    1 & \text{if } \text{csgn}(e_d(i)) = \text{csgn}(e_s(i)) \\
    0 & \text{if } \text{csgn}(e_d(i)) \neq \text{csgn}(e_s(i)) 
  \end{cases} \quad \text{(a flag)} \\
  e(i) &= f(i) e_d(i) \\
  w_i &= w_{i-1} + \mu u_i^* e(i)
\end{align*}
\]

Use the same step-size and simulation duration as in part (d.1).
Project III.4 (Blind adaptive equalization) The programs that solve this project are the following.

1. partA.m This program solves part (a) and generates two plots. Figure 18 shows the impulse responses of the channel, the equalizer, and the combination channel-equalizer. Observe that the latter has a peak at sample 19 so that the delay introduced by the channel and equalizer is 19. Figure 19 shows a typical plot of the scatter diagrams at transmission, reception, and output of the equalizer.

![Graphs of Channel, Equalizer, and Combination Impulse Responses](image)

Figure III.18. Impulse responses of the channel, the blind equalizer, and the combination channel-equalizer for QPSK transmissions. Observe that the latter has a peak at sample 19.
Figure III.19. Scatter diagrams of the transmitted sequence, received sequence, and the output of the equalizer for QPSK transmissions and using CMA2-2. The first 2000 samples are ignored in order to give the equalizer sufficient time for convergence.
2. `partB.m` This program solves part (b) and generates three plots. Figure 20 shows the impulse responses of the channel, the equalizer, and the combination channel-equalizer. Figure 21 shows a typical plot of the scatter diagrams at transmission, reception, and output of the equalizer. Observe that although symbols from 16-QAM do not have constant modulus, CMA2-2 still functions properly; albeit more slowly. Figure 22 shows the real and imaginary parts of 50 transmitted symbols and the corresponding 50 decisions by the slicer (with the delay of 19 samples accounted for in order to synchronize the signals). These 50 symbols are chosen from the later part of the data after the equalizer has converged.

![Graphs](image)

**Figure III.20.** Impulse responses of the channel, the blind equalizer, and the combination channel-equalizer for 16-QAM transmissions.
3. *partC.m* This program solves part (c) and generates three plots as in part (b). Here we only show in Fig. 23 the resulting scatter diagrams for MMA. Observe how the scatter diagram at the output of the equalizer is more focused when compared with CMA2-2.

4. *partD.m* This program solves part (d) and generates scatter diagrams for three additional blind adaptive algorithms. Typical outputs are shown in Fig. 24.

![Graphs showing transmitted, received, and equalizer output sequences.](image)

**Figure III.21.** Scatter diagrams of the transmitted sequence, received sequence, and the output of the equalizer for 16-QAM transmissions and using CMA2-2. The first 15000 samples are ignored in order to give the equalizer sufficient time for convergence.
Figure III.22. Time diagrams of 50 transmitted symbols and the corresponding decisions after equalizer convergence. The top two plots correspond to the real parts, while the bottom two plots correspond to the imaginary parts.
Figure III.23. Scatter diagrams of the transmitted sequence, received sequence, and the output of the equalizer for 16-QAM transmissions and using MMA. The first 15000 samples are ignored in order to give the equalizer sufficient time for convergence.
Figure III.24. Scatter diagrams of the transmitted (left) and received sequences (center), and of the output of the equalizer (right) for three blind algorithms: CMA1-2 (top row), RCA (middle row), and stop-and-go algorithm (bottom row).