
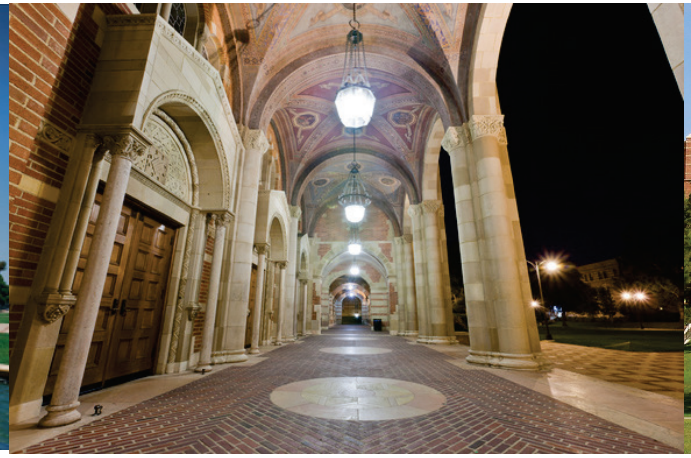


EE210A: Adaptation and Learning

Professor Ali H. Sayed

LECTURE #15

RECURSIVE LEAST-SQUARES

Sections in order: 30.1-30.6, 33.1-33.3, B.6, 33.4

LEAST-SQUARES PROBLEMS

TABLE 29.1 Normal equations associated with several least-squares problems.

Problem	Cost function	Normal equations
Standard least-squares	$\min_w \|y - Hw\|^2$	$H^* H\widehat{w} = H^* y$
Weighted least-squares	$\min_w \|y - Hw\|^2_W,\ W > 0$	$H^* W H\widehat{w} = H^* W y$
Regularized least-squares	$\min_w \|w - \bar{w}\|^2_\Pi + \|y - Hw\|^2$ $\Pi > 0$	$(\Pi + H^* H)(\widehat{w} - \bar{w}) = H^*(y - H\bar{w})$
Weighted regularized least-squares	$\min_w \|w - \bar{w}\|^2_\Pi + \|y - Hw\|^2_W$ $\Pi > 0,\ W \geq 0$	$(\Pi + H^* W H)(\widehat{w} - \bar{w}) = H^* W(y - H\bar{w})$

ORTHOGONALITY CONDITIONS

TABLE 29.2 Orthogonality conditions associated with several least-squares problems. In the statements below, $\widetilde{y} = y - \widehat{y}$ where $\widehat{y} = H\widehat{w}$.

Problem	Cost function	Orthogonality condition
Standard least-squares	$\min_w \|y - Hw\|^2$	$H^*\widetilde{y} = 0$
Weighted least-squares	$\min_w \|y - Hw\|_W^2, \; W > 0$	$H^*W\widetilde{y} = 0$
Regularized least-squares	$\min_w \|w - \bar{w}\|_\Pi^2 + \|y - Hw\|^2$ $\Pi > 0$	$H^*\widetilde{y} = \Pi(\widehat{w} - \bar{w})$
Weighted regularized least-squares	$\min_w \|w - \bar{w}\|_\Pi^2 + \|y - Hw\|_W^2$ $\Pi > 0, \; W \geq 0$	$H^*W\widetilde{y} = \Pi(\widehat{w} - \bar{w})$

MINIMUM COSTS

TABLE 29.3 Minimum costs associated with several least-squares problems. In the statements below, $\widetilde{y} = y - \widehat{y}$ where $\widehat{y} = H\widehat{w}$.

Problem	Cost function	Minimum cost
Standard least-squares	$\min_w \|y - Hw\|^2$	$y^*\widetilde{y}$
Weighted least-squares	$\min_w \|y - Hw\|_W^2,\ W > 0$	$y^*W\widetilde{y}$
Regularized least-squares	$\min_w \|w - \bar{w}\|_\Pi^2 + \|y - Hw\|^2$ $\Pi > 0$	$(y - H\bar{w})^*\widetilde{y}$
Weighted regularized least-squares	$\min_w \|w - \bar{w}\|_\Pi^2 + \|y - Hw\|_W^2$ $\Pi > 0,\ W \geq 0$	$(y - H\bar{w})^*W\widetilde{y}$

30.1 MOTIVATION

Given an $N \times 1$ measurement vector y, an $N \times M$ data matrix H and an $M \times M$ positive-definite matrix Π, we saw in Sec. 29.7 that the $M \times 1$ solution to the following regularized least-squares problem:

$$\min_{w} \; \left[\, w^{*} \Pi w \; + \; \|y - Hw\|^{2} \, \right] \tag{30.1}$$

is given by

$$\widehat{w} = (\Pi + H^{*}H)^{-1} H^{*} y \tag{30.2}$$

where, in comparison with (29.28), we are assuming $\bar{w} = 0$ for simplicity of presentation. The arguments would apply equally well to the case $\bar{w} \neq 0$ — see the remark after Lemma 30.1.

MOTIVATION

We denote the individual entries of y by $\{d(i)\}$, and the individual rows of H by $\{u_i\}$, say

$$y = \begin{bmatrix} d(0) \\ d(1) \\ d(2) \\ \vdots \\ d(N-1) \end{bmatrix}, \qquad H = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix}$$

so that the solution $\widehat{w}$ in (30.2) is determined by data $\{d(i), u_i\}$ defined up to time $N-1$. In order to indicate this fact explicitly, we shall write w_{N-1} instead of $\widehat{w}$ from now on, with a time subscript $(N-1)$. We shall also write y_{N-1} and H_{N-1} instead of y and H since these quantities are defined in terms of data up to time $N-1$ as well. With this notation, we replace problem (30.1) by

$$\min_{w} \; \left[\; w^* \Pi w \; + \; \|y_{N-1} - H_{N-1} w\|^2 \; \right] \tag{30.3}$$

and its solution (30.2) by

$$\boxed{w_{N-1} = (\Pi + H_{N-1}^* H_{N-1})^{-1} H_{N-1}^* y_{N-1}} \tag{30.4}$$

MOTIVATION

In recursive least-squares, we deal with the issue of an increasing N, and, hence of an increasing amount of data. If, for example, one more row is added to H_{N-1} and one more entry is added to y_{N-1}, leading to

$$y_N = \begin{bmatrix} y_{N-1} \\ d(N) \end{bmatrix}, \qquad H_N = \begin{bmatrix} H_{N-1} \\ u_N \end{bmatrix} \tag{30.5}$$

then the solution w_N of the time-updated least-squares problem

$$\min_w \; \left[\, w^* \Pi w \; + \; \|y_N - H_N w\|^2 \, \right] \tag{30.6}$$

would become

$$\boxed{w_N = (\Pi + H_N^* H_N)^{-1} H_N^* y_N} \tag{30.7}$$

Going from (30.4) to (30.7) is referred to as a *time-update* step since it amounts to employing new data $\{d(N), u_N\}$ in addition to all previous data $\{d(j), u_j, 0 \leq j \leq N-1\}$.

MOTIVATION

Now, performing time-updates by relying on expressions (30.4) and (30.7) is costly both computationally and memory-wise. This is because they require that we invert the $M \times M$ coefficient matrices $(\Pi + H_N^* H_N)$ and $(\Pi + H_{N-1}^* H_{N-1})$ at the respective time instants. In addition, (30.7) requires that we store in memory the entries of $\{H_{N-1}, y_{N-1}\}$ so that $\{H_N, y_N\}$ could be formed when the new data $\{u_N, d(N)\}$ become available.

These two requirements of matrix inversion (requiring $O(M^3)$ operations) and increasing storage capacity can be alleviated by seeking an update method that would compute w_N solely from knowledge of the new data $\{d(N), u_N\}$ and from the previous solution w_{N-1}. Such a method is possible since, as we see from (30.5), the quantities $\{H_{N-1}, y_{N-1}\}$ and $\{H_N, y_N\}$ differ only by the new data $\{u_N, d(N)\}$; all other entries are identical. Exploiting this observation is the basis for deriving the recursive least-squares algorithm.

30.2 RLS ALGORITHM

Introduce the matrices

$$\boxed{P_N \triangleq (\Pi + H_N^* H_N)^{-1}, \qquad P_{N-1} \triangleq (\Pi + H_{N-1}^* H_{N-1})^{-1}} \tag{30.8}$$

with initial condition $P_{-1} = \Pi^{-1}$. Then (30.4) and (30.7) can be written more compactly as

$$\boxed{w_{N-1} = P_{N-1} H_{N-1}^* y_{N-1}, \qquad w_N = P_N H_N^* y_N} \tag{30.9}$$

The time-update relation (30.5) between $\{y_N, H_N\}$ and $\{y_{N-1}, H_{N-1}\}$ can be used to relate P_N to P_{N-1} and w_N to w_{N-1}.

DERIVATION

To begin with, note that

$$\begin{aligned} P_N^{-1} &= \Pi + H_N^* H_N \\ &= \Pi + H_{N-1}^* H_{N-1} + u_N^* u_N \end{aligned}$$

so that

$$\boxed{P_N^{-1} = P_{N-1}^{-1} + u_N^* u_N} \tag{30.10}$$

Then, by using the matrix inversion identity

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1} \tag{30.11}$$

with the identifications $A \leftarrow P_{N-1}^{-1}$, $B \leftarrow u_N^*$, $C \leftarrow 1$, and $D \leftarrow u_N$, we obtain a recursive formula for updating P_N directly rather than its inverse,

$$\boxed{P_N = P_{N-1} - \frac{P_{N-1}u_N^* u_N P_{N-1}}{1 + u_N P_{N-1} u_N^*}, \quad P_{-1} = \Pi^{-1}} \tag{30.12}$$

This recursion for P_N also gives one for updating the least-squares solution w_N itself.

DERIVATION

Using expression (30.9) for w_N, and substituting the above recursion for P_N, we find

$$\begin{aligned}
w_N &= P_N\left[H_{N-1}^* y_{N-1} + u_N^* d(N)\right] \\
&= \left(P_{N-1} - \frac{P_{N-1}u_N^* u_N P_{N-1}}{1+u_N P_{N-1}u_N^*}\right)\left[H_{N-1}^* y_{N-1} + u_N^* d(N)\right] \\
&= \underbrace{P_{N-1}H_{N-1}^* y_{N-1}}_{=w_{N-1}} - \frac{P_{N-1}u_N^*}{1+u_N P_{N-1}u_N^*}u_N \underbrace{P_{N-1}H_{N-1}^* y_{N-1}}_{=w_{N-1}} \\
&\quad + P_{N-1}u_N^*\left(1 - \frac{u_N P_{N-1}u_N^*}{1+u_N P_{N-1}u_N^*}\right)d(N)
\end{aligned}$$

That is,

$$\boxed{w_N = w_{N-1} + \frac{P_{N-1}u_N^*}{1+u_N P_{N-1}u_N^*}[d(N) - u_N w_{N-1}], \quad w_{-1} = 0} \qquad (30.13)$$

CONVERSION FACTOR

We summarize the time-updates for $\{P_N, w_N\}$ in Alg. 30.1 below, where we also introduce two important quantities. One is called the conversion factor, for reasons to be explained shortly in Sec. 30.4, and is defined by

$$\boxed{\gamma(N) \triangleq 1/(1 + u_N P_{N-1} u_N^*)} \tag{30.14}$$

whereas the other is called the gain vector and is defined by

$$\boxed{g_N \triangleq P_{N-1} u_N^* \gamma(N)} \tag{30.15}$$

Some straightforward algebra, using recursion (30.12) for P_N, shows that $\{g_N, \gamma(N)\}$ are also given by

$$\boxed{\gamma(N) = 1 - u_N P_N u_N^*} \tag{30.16}$$

and

$$\boxed{g_N = P_N u_N^*} \tag{30.17}$$

ALTERNATIVE EXPRESSIONS

To justify (30.17)–(30.16) simply note the following. Multiplying recursion (30.12) for P_N by u_N^* from the right we get

$$\begin{aligned} P_N u_N^* &= P_{N-1} u_N^* - \frac{P_{N-1} u_N^* u_N P_{N-1} u_N^*}{1 + u_N P_{N-1} u_N^*} \\ &= \frac{P_{N-1} u_N^*}{1 + u_N P_{N-1} u_N^*} \\ &= g_N \end{aligned}$$

By further multiplying the above identity by u_N from the left we get

$$u_N P_N u_N^* = \frac{u_N P_{N-1} u_N^*}{1 + u_N P_{N-1} u_N^*}$$

so that, by subtracting 1 from both sides, we obtain (30.16).

RLS ALGORITHM

Algorithm 30.1 (RLS algorithm) Given $\Pi > 0$, the solution w_N that minimizes the cost $w^*\Pi w + \|y_N - H_N w\|^2$ can be computed recursively as follows. Start with $w_{-1} = 0$ and $P_{-1} = \Pi^{-1}$ and iterate for $i \geq 0$:

$$\begin{aligned}
\gamma(i) &= 1/(1 + u_i P_{i-1} u_i^*) \\
g_i &= P_{i-1} u_i^* \gamma(i) \\
w_i &= w_{i-1} + g_i[d(i) - u_i w_{i-1}] \\
P_i &= P_{i-1} - g_i g_i^* / \gamma(i)
\end{aligned}$$

At each iteration, the matrix P_i has the interpretation

$$P_i = (\Pi + H_i^* H_i)^{-1}$$

and w_i minimizes $w^*\Pi w + \|y_i - H_i w\|^2$, where $y_i = \text{col}\{d(0), d(1), \ldots, d(i)\}$ and the rows of H_i are $\{u_0, u_1, \ldots, u_i\}$.

30.3 REGULARIZATION

Observe that regularization is necessary (i.e., the use of $\Pi > 0$), since it guarantees the existence of $P_N = (\Pi + H_N^* H_N)^{-1}$. In the absence of regularization, i.e., when $\Pi = 0$, the above inverse need not exist especially during the initial update stages when H_N has fewer rows than columns or even at later stages if H_N becomes rank deficient. In these situations, the RLS recursions of Alg. 30.1 would not be applicable, not only because the matrix P_N is not defined but also because of the non-practical initialization $P_{-1} = \infty \mathrm{I}$.

In Sec. 35.2 we shall describe an alternative implementation of RLS that addresses these difficulties; it can be used even in the absence of regularization and has better numerical properties than RLS itself. In preparation for the derivation of this alternative algorithm later in Sec. 35.2, we indicate here some of the RLS equations that will be needed in that section.

REGULARIZATION

Thus note that expression (30.7) for w_N is simply a rewriting of the normal equations

$$(\Pi + H_N^* H_N) w_N = H_N^* y_N \tag{30.18}$$

which characterize the solution of the regularized least-squares problem (30.6). The form (30.18) is more general than (30.7) in that it holds even when the coefficient matrix $(\Pi + H_N^* H_N)$ is singular (e.g., when $\Pi = 0$ and H_N is rank deficient) — recall (29.33) and the discussion thereafter.

Now if we define the $M \times M$ matrix $\Phi_N = \Pi + H_N^* H_N$, then, just like the recursion (30.10) for P_N^{-1}, we also obtain a recursion for Φ_N:

$$\boxed{\Phi_N = \Phi_{N-1} + u_N^* u_N, \qquad \Phi_{-1} = \Pi} \tag{30.19}$$

If we further define the column vector $s_N = H_N^* y_N$, then it is immediate to see, using (30.5), that s_N satisfies the time-update recursion:

$$\boxed{s_N = s_{N-1} + u_N^* d(N), \qquad s_{-1} = 0} \tag{30.20}$$

REGULARIZATION

With $\{\Phi_N, s_N\}$ so defined, the normal equations (30.18) can be rewritten more compactly as

$$\boxed{\Phi_N w_N = s_N} \tag{30.21}$$

The point is that this description is valid even when $\Pi = 0$ since, as we already know from Thm. 29.1, the corresponding normal equations

$$\underbrace{H_N^* H_N}_{\Phi_N} w_N = \underbrace{H_N^* y_N}_{s_N}$$

are consistent so that a solution w_N can always be found (e.g., one could choose the minimum-norm solution when multiple solutions exist). The equations (30.19)–(30.21) will form the basis for the aforementioned alternative implementation of RLS in Sec. 35.2; one that has better properties in finite-precision arithmetic and can be used even if $\Pi = 0$.

30.4 CONVERSION FACTOR

We can also derive a time-update relation for the minimum costs associated with problems (30.3) and (30.6). To do so, we first associate with RLS two error quantities: the *a priori* output error, denoted by $e(N)$,

$$\boxed{e(N) \triangleq d(N) - u_N w_{N-1}} \tag{30.22}$$

and the *a posteriori* output error, denoted by $r(N)$,

$$\boxed{r(N) \triangleq d(N) - u_N w_N} \tag{30.23}$$

It then turns out that the factor $\gamma(N)$ defined by (30.14) serves a useful purpose: it maps $e(N)$ to $r(N)$, i.e., it converts the *a priori* error into its *a posteriori* version.

CONVERSION FACTOR

To see this, we replace w_N in the definition of $r(N)$ by its RLS update from Alg. 30.1 to obtain

$$\begin{aligned} r(N) &= d(N) - u_N(\, w_{N-1} + g_N[d(N) - u_N w_{N-1}]\,) \\ &= d(N) - u_N w_{N-1} - u_N g_N e(N) \\ &= e(N) - u_N g_N e(N) \\ &= (1 - u_N g_N) e(N) \end{aligned}$$

Using (30.16) and (30.17) we get

$$\boxed{r(N) = \gamma(N) e(N)}$$

as desired. Observe further from the definition of $\gamma(N)$ in (30.14) that

$$\boxed{0 < \gamma(N) \leq 1} \tag{30.24}$$

so that it always holds that

$$|r(N)| \leq |e(N)|$$

30.5 TIME-UPDATE OF THE MINIMUM COST

Now let $\xi(N-1)$ denote the minimum cost of the regularized least-squares problem (30.3). Likewise, let $\xi(N)$ denote the minimum cost of the time-updated problem (30.6). From Thm. 29.4 we know that $\xi(N-1)$ and $\xi(N)$ are given by

$$\xi(N) = y_N^*[y_N - H_N w_N], \qquad \xi(N-1) = y_{N-1}^*[y_{N-1} - H_{N-1} w_{N-1}]$$

Using the partitioning (30.5) for $\{y_N, H_N\}$, as well as the RLS update

$$w_N = w_{N-1} + g_N e(N), \qquad e(N) = d(N) - u_N w_{N-1}$$

we can arrive at a relation between $\xi(N)$ and $\xi(N-1)$ by means of the following sequence of calculations:

MINIMUM COST

$$
\begin{aligned}
\xi(N) &= \begin{bmatrix} y_{N-1}^* & d^*(N) \end{bmatrix} \begin{bmatrix} y_{N-1} - H_{N-1}[w_{N-1} + g_N e(N)] \\ d(N) - u_N[w_{N-1} + g_N e(N)] \end{bmatrix} \\
&= \underbrace{y_{N-1}^*(y_{N-1} - H_{N-1} w_{N-1})}_{=\xi(N-1)} - y_{N-1}^* H_{N-1} g_N e(N) + d^*(N) e(N)[1 - u_N g_N] \\
&= \xi(N-1) + e(N) \left[d^*(N) \underbrace{(1 - u_N g_N)}_{=\gamma(N)} - y_{N-1}^* H_{N-1} g_N \right] \\
&= \xi(N-1) + e(N)\gamma(N) \left[d^*(N) - \underbrace{y_{N-1}^* H_{N-1} P_{N-1}}_{=w_{N-1}^*} u_N^* \right] \\
&= \xi(N-1) + |e(N)|^2 \gamma(N)
\end{aligned}
$$

Moreover, since $r(N) = \gamma(N) e(N)$, it holds that

$$
|e(N)|^2 \gamma(N) = e(N) r^*(N) = e^*(N) r(N) = |r(N)|^2 / \gamma(N)
$$

ERRORS AND MINIMUM COST

Lemma 30.1 (Estimation errors) Consider the same setting of Alg. 30.1. At each iteration i, the *a priori* and *a posteriori* estimation errors defined by

$$e(i) = d(i) - u_i w_{i-1} \quad \text{and} \quad r(i) = d(i) - u_i w_i$$

are related via the conversion factor $\gamma(i)$ as $r(i) = \gamma(i)e(i)$. Moreover, the minimum costs of the successive regularized least-squares problems satisfy any of the following time-update relations with initial condition $\xi(-1) = 0$:

$$\begin{aligned} \xi(i) &= \xi(i-1) + e(i)r^*(i) \\ &= \xi(i-1) + \gamma(i)|e(i)|^2 \\ &= \xi(i-1) + |r(i)|^2/\gamma(i) \end{aligned}$$

INITIAL CONDITIONS

One final remark is in place. Had we considered a regularized cost function of the form

$$\min_{w} \; \left[(w-\bar{w})^{*}\Pi(w-\bar{w}) \; + \; \|y_N - H_N w\|^2\right]$$

with a nonzero $\bar{w}$, then the only modification to the RLS algorithm would be in the value of its initial condition, w_{-1}; it becomes $w_{-1} = \bar{w}$. Moreover, the time-update for $\xi(N)$ will still hold. To see this, we only need to repeat the above derivation for $\xi(N)$ starting from the expression (cf. Table 29.3) — see Prob. VII.29:

$$\xi(N) = [y_N - H_N\bar{w}]^{*}[y_N - H_N w_N]$$

30.6 EXPONENTIALLY-WEIGHTED RLS ALGORITHM

It is more common in adaptive filtering to employ a *weighted* regularized least-squares cost function, as opposed to the unweighted cost in (30.6). More specifically, a diagonal weighting matrix is used whose purpose is to give more weight to recent data and less weight to data from the remote past.

Let λ be a positive scalar, usually very close to one (e.g., $\lambda = 0.998$ or some similar value), say $0 \ll \lambda \leq 1$, and introduce the diagonal matrix

$$\Lambda_N \triangleq \text{diag}\{\lambda^N, \lambda^{N-1}, \ldots, \lambda, 1\} \tag{30.25}$$

Then replace (30.6) by

$$\min_w \left[\lambda^{(N+1)} w^* \Pi w + (y_N - H_N w)^* \Lambda_N (y_N - H_N w) \right] \tag{30.26}$$

or, more explicitly, by

$$\boxed{\min_w \left[\lambda^{(N+1)} w^* \Pi w + \sum_{j=0}^{N} \lambda^{N-j} |d(j) - u_j w|^2 \right]} \tag{30.27}$$

EXPONENTIALLY-WEIGHTED RLS

The scalar λ is called the *forgetting factor* since past data are exponentially weighted less heavily than more recent data. The special case $\lambda = 1$ is known as the *growing memory* case and it was studied in the previous sections. Exponential weighting is one form of data windowing whereby the effective length of the window is $\approx 1/(1-\lambda)$ samples.

Observe that the regularization matrix in (30.26) is chosen as $\lambda^{(N+1)}\Pi$, with the additional scaling factor $\lambda^{(N+1)}$. Since this factor becomes smaller as time progresses, we see that the exponentially-weighted cost function (30.26) is such that it de-emphasizes regularization during the later stages of operation when the data matrix H_N is more likely to have full rank.

Comparing (30.26) with (29.37), and using the identifications $\Pi \leftarrow \lambda^{(N+1)}\Pi$ and $W \leftarrow \Lambda_N$, we find that the solution w_N is obtained by solving

$$\left[\lambda^{(N+1)}\Pi + H_N^*\Lambda_N H_N\right] w_N = H_N^*\Lambda_N y_N \tag{30.28}$$

If we now define the quantities

$$P_N \triangleq \left[\lambda^{(N+1)}\Pi + H_N^*\Lambda_N H_N\right]^{-1}, \quad g_N \triangleq \lambda^{-1}P_{N-1}u_N^*\gamma(N)$$

and $\gamma(N) = 1/(1+\lambda^{-1}u_N P_{N-1}u_N^*)$, and repeat the arguments prior to the statement of Alg. 30.1, we arrive at the following statement.

EXPONENTIALLY-WEIGHTED RLS

Algorithm 30.2 (Exponentially-weighted RLS) Given $\Pi > 0$, and a forgetting factor $0 \ll \lambda \leq 1$, the solution w_N of the exponentially-weighted regularized least-squares problem

$$\min_{w} \left[\lambda^{(N+1)} w^* \Pi w \; + \; \sum_{j=0}^{N} \lambda^{N-j} |d(j) - u_j w|^2 \right]$$

and the corresponding minimum cost, $\xi(N)$, can be computed recursively as follows. Start with $w_{-1} = 0$, $P_{-1} = \Pi^{-1}$, and $\xi(-1) = 0$, and iterate for $i \geq 0$:

$$\begin{aligned}
\gamma(i) &= 1/(1 + \lambda^{-1} u_i P_{i-1} u_i^*) \\
g_i &= \lambda^{-1} P_{i-1} u_i^* \gamma(i) \\
e(i) &= d(i) - u_i w_{i-1} \\
w_i &= w_{i-1} + g_i e(i) \\
P_i &= \lambda^{-1} P_{i-1} - g_i g_i^* / \gamma(i) \\
\xi(i) &= \lambda \xi(i-1) \; + \; \gamma(i) |e(i)|^2
\end{aligned}$$

At each iteration, P_i has the interpretation $P_i = \left[\lambda^{(i+1)} \Pi + H_i^* \Lambda_i H_i\right]^{-1}$ and w_i is the solution of

$$\min_{w} \left[\lambda^{(i+1)} w^* \Pi w \; + \; \sum_{j=0}^{i} \lambda^{i-j} |d(j) - u_j w|^2 \right]$$

In addition, as was the case with (30.16)–(30.17), the following relations hold:

$$g_i = P_i u_i^*, \qquad \gamma(i) = 1 - u_i P_i u_i^* = 1 - u_i g_i, \qquad r(i) = \gamma(i) e(i)$$

where $r(i) = d(i) - u_i w_i$.

ALTERNATIVE FORM

Again, starting from the normal equations (30.28), we can define the quantities

$$\Phi_N \triangleq \lambda^{(N+1)}\Pi + H_N^*\Lambda_N H_N \tag{30.29}$$

$$s_N \triangleq H_N^*\Lambda_N y_N \tag{30.30}$$

Then it can be easily verified that they satisfy the recursions

$$\Phi_N = \lambda\Phi_{N-1} + u_N^* u_N, \qquad \Phi_{-1} = \Pi \tag{30.31}$$

$$s_N = \lambda s_{N-1} + u_N^* d(N), \qquad s_{-1} = 0 \tag{30.32}$$

and that w_N can be found by solving the normal equations

$$\Phi_N w_N = s_N \tag{30.33}$$

As mentioned in Sec. 30.3, these equations will be used in Sec. 35.2 to motivate an alternative recursive implementation of the exponentially-weighted RLS algorithm.

ALTERNATIVE FORM

Moreover, had we started with a regularized cost function of the form

$$\min_{w} \left[\lambda^{(N+1)}(w-\bar{w})^{*}\Pi(w-\bar{w}) + \sum_{j=0}^{N} \lambda^{N-j}|d(j)-u_j w|^2 \right]$$

with a nonzero $\bar{w}$, then the only modification to the RLS equations of Alg. 30.2 would be in the value of the initial condition, w_{-1}; it becomes $w_{-1} = \bar{w}$. Likewise, equations (30.31)–(30.33) would be replaced by

$$\Phi_N = \lambda\Phi_{N-1} + u_N^{*}u_N, \qquad \Phi_{-1} = \Pi \tag{30.34}$$

$$s_N = \lambda s_{N-1} + u_N^{*}[d(N) - u_N\bar{w}], \qquad s_{-1} = 0 \tag{30.35}$$

$$\Phi_N(w_N - \bar{w}) = s_N \tag{30.36}$$

33.1 SOME DIFFICULTIES

$$\boxed{P_N = P_{N-1} - \frac{P_{N-1}u_N^* u_N P_{N-1}}{1 + u_N P_{N-1} u_N^*}, \qquad P_{-1} = \Pi^{-1}} \tag{30.12}$$

Thus consider the update equation (30.12) for the variable P_N in the RLS algorithm, and assume that all variables are real and scalar-valued (and, hence, we shall write $\{u(N), P(N)\}$ instead of $\{u_N, P_N\}$). Assume further that at some iteration n_o, especially during the initial stages of adaptation where $P(N)$ is more likely to assume large values, $u(n_o) = 1$ and $P(n_o - 1)$ is sufficiently large so that, in *finite precision*, the value of $1 + P(n_o - 1)$ evaluates to $P(n_o - 1)$, i.e.,

$$1 + P(n_o - 1) = P(n_o - 1) \qquad \text{(in finite precision)} \tag{33.1}$$

DIFFICULTIES

$$\boxed{P_N = P_{N-1} - \frac{P_{N-1}u_N^* u_N P_{N-1}}{1 + u_N P_{N-1} u_N^*}, \qquad P_{-1} = \Pi^{-1}} \tag{30.12}$$

From (30.12) we have that the value of $P(n_o)$ is obtained via the update

$$P(n_o) = P(n_o - 1) - \frac{P^2(n_o - 1)}{1 + P(n_o - 1)} \tag{33.2}$$

which is also equivalent to

$$P(n_o) = \frac{P(n_o - 1)}{1 + P(n_o - 1)} \tag{33.3}$$

Now assume that the term $P^2(n_o - 1)/[1 + P(n_o - 1)]$ in (33.2) is evaluated as follows:

$$\frac{P^2(n_o - 1)}{1 + P(n_o - 1)} = P(n_o - 1) \cdot \frac{P(n_o - 1)}{1 + P(n_o - 1)}$$

That is, we first evaluate the ratio $P(n_o - 1)/[1 + P(n_o - 1)]$ and then multiply the result by $P(n_o - 1)$.

DIFFICULTIES

Then, because of (33.1), the ratio $P(n_o-1)/[1+P(n_o-1)]$ evaluates to 1 and, therefore, if we compute $P(n_o)$ using (33.2) we get

$$P(n_o) = P(n_o-1) \; - \; P(n_o-1) \cdot 1 \; = \; 0$$

On the other hand, recursion (33.3) gives $P(n_o) = 1$.

The values obtained for $P(n_o)$ are obviously different and the second one is the desired value since, from (33.3),

$$\lim_{P(n_o-1)\to\infty} P(n_o) = 1$$

We therefore find from the equivalent expressions (33.2) and (33.3) that these two different implementations of the *same* recursion can behave differently in the presence of round-off errors. There is a reason for the failure of the implementation based on (33.2); it evaluates the nonnegative quantity $P(n_o)$ as the difference of two nearly equal and large nonnegative numbers, and the result is an undesired cancellation; in some other more complex scenarios, the variable P_N may even lose its positive-definiteness. Such cancellations are often, but not always, bad phenomena in finite-precision implementations so much so that they are usually called *catastrophic cancellations*!

MOTIVATION

This simple example shows that there is always merit in considering alternative implementations even for the same algorithm. Another example is presented in Prob. VIII.11 where two theoretically equivalent implementations of the same algorithm are also shown to react differently in response to perturbation in the data.

For these and other reasons, we are motivated to develop *array* methods for recursive least-squares problems. The array methods will have several intrinsic advantages over a plain RLS implementation; for one thing, they will be more reliable in finite-precision implementations (as explained in the sequel and as illustrated later in the computer project at the end of the chapter). We start with a handful of definitions.

33.2 SQUARE-ROOT FACTORS

A key element in array algorithms is the concept of a square-root of a positive-definite matrix.

Definition

Although the concept of square-roots can be defined for nonnegative-definite matrices, it is enough for our purposes here to focus on positive-definite matrices. Thus let A denote an $n \times n$ positive-definite matrix and introduce its eigen-decomposition

$$A = U\Lambda U^* \tag{33.4}$$

where Λ is an $n \times n$ diagonal matrix with real positive entries, which correspond to the eigenvalues of A, and U is a unitary matrix, namely an $n \times n$ square matrix that satisfies

$$UU^* = U^*U = \mathrm{I}$$

SQUARE-ROOT FACTORS

The columns of U correspond to the orthonormal eigenvectors of A as can be seen by re-writing (33.4) as $AU = U\Lambda$. Let $\Lambda^{1/2}$ denote a diagonal matrix whose entries are the positive square-roots of the diagonal entries of Λ and, hence,

$$\Lambda = \Lambda^{1/2}\left(\Lambda^{1/2}\right)^*$$

Then we can rewrite (33.4) as

$$A = \left(U\Lambda^{1/2}\right)\cdot\left(U\Lambda^{1/2}\right)^*$$

which expresses A as the product of an $n \times n$ matrix and its conjugate transpose, namely,

$$A = XX^* \qquad \text{with} \qquad X = U\Lambda^{1/2}$$

We say that X is *a* square-root of A.

Definition 33.1 (Square-root factors) A square-root of an $n \times n$ positive-definite matrix A is any $n \times n$ matrix X satisfying $A = XX^*$.

NON-UNIQUENESS

The construction prior to the definition exhibits one possible choice for X, namely, $X = U\Lambda^{1/2}$, in terms of the eigenvectors and eigenvalues of A. However, square-root factors are highly nonunique. This is true even for scalars. For instance, the number 4 has infinitely many square-roots over the field of complex numbers, namely, $2e^{j\phi}$ for any $\phi \in [-\pi, \pi]$. For matrices, if we take the above X and multiply it by any unitary matrix Θ, say $\bar{X} = X\Theta$ where $\Theta\Theta^* = \mathrm{I}$, then $\bar{X}$ is also a square-root factor of A since

$$\bar{X}\bar{X}^* = X\underbrace{\Theta\Theta^*}_{=\mathrm{I}}X^* = XX^* = A$$

NOTATION

Notation

It is customary to use the notation $A^{1/2}$ to refer to a square-root of a matrix A and, therefore, we write

$$A = A^{1/2}\left(A^{1/2}\right)^*$$

It is also customary to employ the compact notations

$$A^{*/2} \triangleq \left(A^{1/2}\right)^*, \qquad A^{-1/2} \triangleq \left(A^{1/2}\right)^{-1}, \qquad A^{-*/2} \triangleq \left(A^{1/2}\right)^{-*}$$

so that

$$\boxed{A = A^{1/2}A^{*/2}, \qquad A^{-1} = A^{-*/2}A^{-1/2}} \tag{33.5}$$

CHOLESKY FACTOR

Cholesky Factor

One of the most widely used square-root factors of a positive-definite matrix is its Cholesky factor. Recall that we showed in App. B.3 that every positive-definite matrix A admits a *unique* triangular factorization of the form

$$A = \bar{L}\bar{L}^* \tag{33.6}$$

where $\bar{L}$ is a lower-triangular matrix with positive entries on its diagonal. We could also consider the alternative factorization $A = \bar{U}\bar{U}^*$ in terms of an upper triangular matrix $\bar{U}$. However, the lower triangular form will be the standard form for our discussions. The factor $\bar{L}$ is called the Cholesky factor of A. Comparing (33.6) with the defining relation (33.5), we see that $\bar{L}$ is also a square-root factor of A. For our purposes, whenever we refer to the square-root of a matrix A we shall mean its Cholesky factor.[17] It has two advantages in relation to other square-root factors: it is lower triangular and is uniquely defined (i.e., there is no other lower triangular square-root factor with positive diagonal entries).

ARRAY ALGORITHMS

Array Algorithms

Now, an array algorithm generally implements transformations of the form

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \Theta = \begin{bmatrix} \times & 0 & 0 & 0 \\ \times & \times & 0 & 0 \\ \times & \times & \times & 0 \\ \times & \times & \times & \times \end{bmatrix}$$

where Θ is some unitary matrix whose purpose is to transform the pre-array of numbers to some triangular form. There are many ways to implement unitary transformations of this kind. For example, it is explained in Chapter 34 that Θ could be implemented as a sequence of elementary transformations (known as Givens rotations) or reflection transformations (known as Householder reflections). Such array descriptions have many intrinsic advantages:

ADVANTAGES OF ARRAY METHODS

1. They have better numerical properties for two main reasons. First, unitary transformations are numerically well-behaved since they do not amplify numerical errors. And second, the entries in the pre- and post-arrays are usually square-root factors of certain variables and these entries tend to assume values within smaller dynamic ranges.

2. They are easy to implement as a sequence of elementary rotations or reflections, as we explain in Chapter 34.

3. They admit modular and parallelizable implementations, since each rotation or reflection can be applied simultaneously to all rows of the pre-array.

33.3 NORM AND ANGLE PRESERVATION

Since unitary transformations are at the core of most array methods, it is important to examine some of their most distinctive properties. To begin with, a key property of unitary transformations is that the norms of vectors and the inner products between vectors are preserved by them. To see this, assume that x and y are two row vectors that are related by some unitary transformation Θ, say $x\Theta = y$, then

$$\|y\|^2 \stackrel{\Delta}{=} yy^* = x\Theta\Theta^*x^* = xx^* \stackrel{\Delta}{=} \|x\|^2$$

That is, the vectors $\{x, y\}$ will have the same Euclidean norm. We therefore say that unitary transformations preserve Euclidean norms:

$$\boxed{x\Theta = y \implies \|x\|^2 = \|y\|^2} \quad \text{(norm preservation)} \tag{33.7}$$

UNITARY TRANSFORMATIONS

In addition, if a and b are two other row vectors that are related by the same transformation, say $a\Theta = b$, then we have that

$$yb^* = x\Theta\Theta^* a^* = xa^*$$

and we find that unitary transformations also preserve inner products between vectors, i.e.,

$$\boxed{x\Theta = y \text{ and } a\Theta = b \implies xa^* = yb^*} \quad \text{(angle preservation)} \tag{33.8}$$

The reason why we are referring to the inner-product preservation property as an angle-preservation property is the following. For the case of real data we have $xa^{\mathsf{T}} = \|x\| \cdot \|a\| \cdot \cos(\theta)$, with θ denoting the angle between x and a. Likewise, $yb^{\mathsf{T}} = \|y\| \cdot \|b\| \cdot \cos(\alpha)$, with α denoting the angle between y and b. But since $\|a\| = \|b\|$ and $\|x\| = \|y\|$, we then conclude from $xa^{\mathsf{T}} = yb^{\mathsf{T}}$ that $\cos(\theta) = \cos(\alpha)$. This latter equality amounts to angle preservation. We thus say that the vectors $\{x, a\}$ are transformed into $\{y, b\}$ in such a way that their norms are preserved as well as the angles between them (see Fig. 33.1).

UNITARY TRANSFORMATIONS

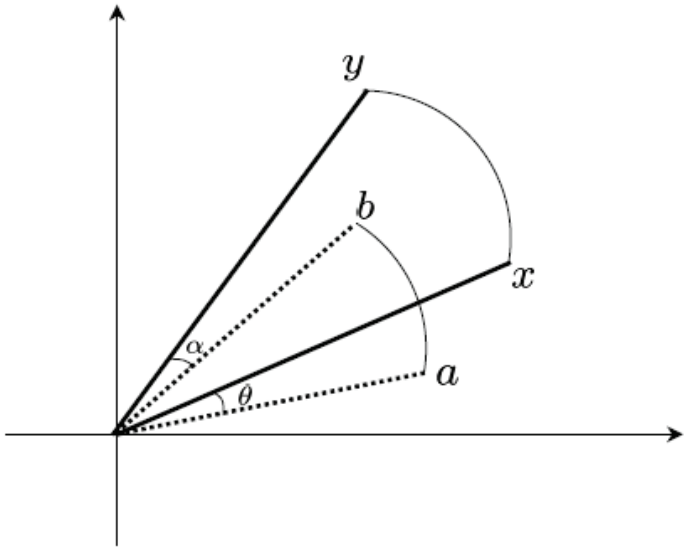

FIGURE 33.1 Rotation of vectors $\{a, x\}$ into $\{b, y\}$ with the vector norms and the angles between them preserved: $\theta = \alpha$, $\|a\| = \|b\|$, and $\|x\| = \|y\|$.

EXAMPLE: INVERSE QR ALGORITHM

Algorithm 35.1 (Inverse QR) Consider data $\{u_j, d(j)\}_{j=0}^{N}$, where the u_j are $1 \times M$ and the $d(j)$ are scalars. Consider also an $M \times 1$ vector $\bar{w}$, an $M \times M$ positive-definite matrix Π, and a scalar $0 \ll \lambda \leq 1$. The solution, w_N, of the least-squares problem (35.1) can be computed recursively as follows.

Let $\Sigma = \Pi^{-1}$ and introduce the Cholesky decomposition $\Sigma = \Sigma^{1/2}\Sigma^{*/2}$, where $\Sigma^{1/2}$ is lower triangular with positive-diagonal entries. Then start with $w_{-1} = \bar{w}$, $P_{-1}^{1/2} = \Sigma^{1/2}$, and repeat for $i \geq 0$.

1. Find a unitary matrix Θ_i that lower triangularizes the pre-array shown below and generates a post-array with positive diagonal entries. Then the entries in the post-array will correspond to

$$\begin{bmatrix} 1 & \lambda^{-1/2} u_i P_{i-1}^{1/2} \\ 0 & \lambda^{-1/2} P_{i-1}^{1/2} \end{bmatrix} \Theta_i = \begin{bmatrix} \gamma^{-1/2}(i) & 0 \\ g_i \gamma^{-1/2}(i) & P_i^{1/2} \end{bmatrix}$$

2. Update the weight vector as

$$w_i = w_{i-1} + \left[g_i \gamma^{-1/2}(i)\right] \left[\gamma^{-1/2}(i)\right]^{-1} [d(i) - u_i w_{i-1}]$$

where the quantities $\{g_i \gamma^{-1/2}(i), \gamma^{-1/2}(i)\}$ are read from the post-array.

The computational complexity of this algorithm is $O(M^2)$ operations per iteration.

EXAMPLE: QR ALGORITHM

Algorithm 35.2 (QR algorithm) Consider data $\{u_j, d(j)\}_{j=0}^{N}$, where the u_j are $1 \times M$ and the $d(j)$ are scalars. Consider also an $M \times 1$ vector $\bar{w}$, an $M \times M$ positive-definite matrix Π, and a scalar $0 \ll \lambda \leq 1$. Let $\bar{d}(i) = d(i) - u_i \bar{w}$. The solution, w_N, of the least-squares problem (35.1) can be computed recursively as follows.

Start with $\Phi_{-1}^{1/2} = \Pi^{1/2}$, $q_{-1} = 0$, and repeat for $i \geq 0$.

1. Find a unitary matrix Θ_i that lower triangularizes the pre-array shown below and generates a post-array with positive diagonal entries in $\Phi_i^{1/2}$, as well as a positive rightmost corner entry in the last row of the post-array. The entries in the post-array will then correspond to

$$\begin{bmatrix} \lambda^{1/2}\Phi_{i-1}^{1/2} & u_i^* \\ \lambda^{1/2} q_{i-1}^* & \bar{d}^*(i) \\ 0 & 1 \end{bmatrix} \Theta_i = \begin{bmatrix} \Phi_i^{1/2} & 0 \\ q_i^* & e^*(i)\gamma^{1/2}(i) \\ u_i \Phi_i^{-*/2} & \gamma^{1/2}(i) \end{bmatrix}$$

2. Obtain w_i by solving the triangular system of equations $\Phi_i^{*/2}[w_i - \bar{w}] = q_i$, where the quantities $\{\Phi_i^{*/2}, q_i\}$ are read from the post-array.

The computational complexity of this algorithm is $O(M^2)$ operations per iteration.

REVIEW: SVD

B.6 SINGULAR VALUE DECOMPOSITION

the SVD of a matrix A states that if A is $n \times m$ then there exist an $n \times n$ unitary matrix U $(UU^* = \mathrm{I})$, an $m \times m$ unitary matrix V $(VV^* = \mathrm{I})$, and a diagonal matrix Σ with nonnegative entries such that:

(i) If $n \leq m$, then Σ is $n \times n$ and

$$A = U \begin{bmatrix} \Sigma & 0 \end{bmatrix} V^*, \quad A : n \times m, \quad n \leq m$$

(i) If $n \geq m$, then Σ is $m \times m$ and

$$A = U \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^*, \quad A : n \times m, \quad n \geq m$$

Observe that U and V are square matrices, while the central matrix has the dimensions of A. The diagonal entries of Σ are called the singular values of A and are usually ordered in decreasing order, say $\Sigma = \mathrm{diag}\{\sigma_1, \sigma_2, \ldots, \sigma_p, 0, \ldots, 0\}$ with $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p > 0$. If Σ has p nonzero diagonal entries then A has rank p. The columns of U and V are called the left- and right-singular vectors of A, respectively.

CONSTRUCTIVE PROOF

One proof of the SVD decomposition follows from the eigen-decomposition of a Hermitian nonnegative definite matrix. The argument given here assumes $n \leq m$, but it can be adjusted to handle the case $n \geq m$.

Note that AA^* is a Hermitian nonnegative-definite matrix and, consequently, there exists an $n \times n$ unitary matrix U and an $n \times n$ diagonal matrix Σ^2, with nonnegative entries, such that $AA^* = U\Sigma^2 U^*$. This representation simply corresponds to the eigen-decomposition of AA^*. The diagonal entries of Σ^2 are the eigenvalues of AA^*, which are nonnegative (and, hence, the notation Σ^2); they are also equal to the nonzero eigenvalues of A^*A. The columns of U are the corresponding orthonormal eigenvectors. By proper reordering, we can always arrange the diagonal entries of Σ^2 in decreasing order so that Σ^2 can be put into the form $\Sigma^2 = \text{diagonal}\,\{\sigma_1^2, \sigma_2^2, \ldots, \sigma_p^2, 0, \ldots, 0\}$, where $p = \text{rank}(AA^*)$ and $\sigma_1^2 \geq \sigma_2^2 \geq \ldots \geq \sigma_p^2 > 0$. If we define the $m \times m$ matrix

$$V_1 = A^* U \,\text{diag}\{\sigma_1^{-1}, \ldots, \sigma_p^{-1}, 0, \ldots, 0\}$$

then it is immediate to conclude that $A = U\Sigma V_1^*$, and that V_1 satisfies

$$V_1^* V_1 = \begin{bmatrix} \mathrm{I} & 0 \\ 0 & 0 \end{bmatrix}$$

But the columns of V_1 can be completed to a full unitary basis V of an $m-$ dimensional space, say $V = \begin{bmatrix} V_1 & V_2 \end{bmatrix}$, such that $VV^* = V^*V = \mathrm{I}$. This allows us to conclude that we can write $A = U \begin{bmatrix} \Sigma & 0 \end{bmatrix} V^*$, which is the desired SVD of A in the $n \leq m$ case. A similar argument establishes the SVD decomposition of A in the $n > m$ case.

BASIS ROTATION

Lemma 33.1 (Basis rotation) Given two $n \times M$ $(n \leq M)$ matrices A and B. Then $AA^* = BB^*$ if, and only if, there exists an $M \times M$ unitary matrix Θ such that $A = B\Theta$.

Proof: One direction is obvious. If $A = B\Theta$, for some unitary matrix Θ, then

$$AA^* = (B\Theta)(B\Theta)^* = B(\Theta\Theta^*)B^* = BB^*$$

One proof for the converse implication follows by using the singular value decompositions of A and B (cf. App. B.6) — see Prob. VIII.2 for another proof:

$$A = U_A \begin{bmatrix} \Sigma_A & 0 \end{bmatrix} V_A^*, \quad B = U_B \begin{bmatrix} \Sigma_B & 0 \end{bmatrix} V_B{}^*$$

where U_A and U_B are $n \times n$ unitary matrices, V_A and V_B are $M \times M$ unitary matrices, and Σ_A and Σ_B are $n \times n$ diagonal matrices with nonnegative entries. The squares of the diagonal entries of Σ_A (Σ_B) are the eigenvalues of AA^* (BB^*). Moreover, U_A (U_B) are constructed from an orthonormal basis for the right eigenvectors of AA^* (BB^*). Hence, it follows from the identity $AA^* = BB^*$ that $\Sigma_A = \Sigma_B$ and $U_A = U_B$. Let $\Theta = V_B V_A^*$. Then $\Theta\Theta^* = \mathrm{I}$ and $B\Theta = A$.

$\diamond$