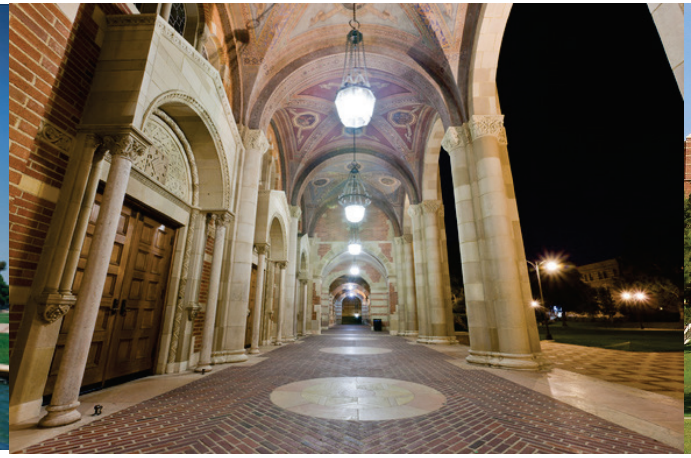




EE210A: Adaptation and Learning

Professor Ali H. Sayed

LECTURE #17

ARRAY RLS ALGORITHMS

Sections in order: 35.1, 35.2, 37.1-37.4

RLS REVIEW

Consider a collection of data $\{u_j, d(j)\}_{j=0}^{N}$, where the $\{u_j\}$ are $1 \times M$ and the $\{d(j)\}$ are scalars, in addition to an $M \times 1$ column vector $\bar{w}$, an $M \times M$ positive-definite matrix Π, and a scalar λ satisfying $0 \ll \lambda \leq 1$. Then the solution, w_N, of the regularized least-squares problem

$$\min_{w} \left[\lambda^{(N+1)} (w - \bar{w})^* \Pi (w - \bar{w}) + \sum_{j=0}^{N} \lambda^{N-j} |d(j) - u_j w|^2 \right] \tag{35.1}$$

and the corresponding minimum cost, $\xi(N)$, can be obtained recursively as follows (recall Alg. 30.2). Start with $w_{-1} = \bar{w}$, $P_{-1} = \Pi^{-1}$, $\xi(-1) = 0$, and repeat for $i \geq 0$:

$$\begin{aligned}
\gamma(i) &= 1/(1 + \lambda^{-1} u_i P_{i-1} u_i^*) \\
g_i &= \lambda^{-1} \gamma(i) P_{i-1} u_i^* \\
e(i) &= d(i) - u_i w_{i-1} \\
w_i &= w_{i-1} + g_i e(i) \\
P_i &= \lambda^{-1} P_{i-1} - g_i g_i^* / \gamma(i) \\
r(i) &= d(i) - u_i w_i \\
\xi(i) &= \lambda \xi(i-1) + r(i) e^*(i)
\end{aligned} \tag{35.2}$$

RLS REVIEW

Moreover, the following relations also hold at each iteration i:

$$\gamma(i) = 1 - u_i P_i u_i^*, \qquad g_i = P_i u_i^* \tag{35.3}$$

We further remarked following Alg. 30.2 that the $\{w_i\}$ also satisfy the following construction. Start with $w_{-1} = \bar{w}$, $s_{-1} = 0$, $\Phi_{-1} = \Pi$, and repeat for $i \geq 0$:

$$\Phi_i = \lambda \Phi_{i-1} + u_i^* u_i, \qquad s_i = \lambda s_{i-1} + u_i^* [d(i) - u_i \bar{w}] \tag{35.4}$$

Then, at each iteration i, it holds that

$$\Phi_i [w_i - \bar{w}] = s_i \tag{35.5}$$

$$\gamma(i) = 1 - u_i \Phi_i^{-1} u_i^* \tag{35.6}$$

because, by the definition (30.30), $\Phi_i = P_i^{-1}$. Equations (35.4)–(35.6) will be significant in this chapter in that they will form the basis for one of the most celebrated array variants of RLS (see Sec. 35.2).

ARRAY REVIEW

In the context of adaptive filtering, however, we shall encounter vector analogues of relations (33.10) and (33.15), such as determining a lower triangular matrix C, with positive diagonal entries, satisfying

$$CC^* = AA^* + BB^* \tag{33.19}$$

and determining a matrix F satisfying

$$FC^* = DA^* + EB^* \tag{33.20}$$

where $\{A, B, D, E\}$ are generally matrix or vector quantities. The same arguments that we used above will reveal that $\{C, F\}$ can be determined by means of an array method as follows. We form the pre-array (cf. (33.16)):

$$\mathcal{A} = \begin{bmatrix} A & B \\ D & E \end{bmatrix}$$

ARRAY REVIEW

and reduce it via a unitary transformation Θ to the lower triangular form (cf. (33.17)):

$$\begin{bmatrix} \mathsf{X} & 0 \\ \mathsf{Y} & \mathsf{Z} \end{bmatrix}$$

where X is lower triangular with positive entries along its diagonal; sometimes Z is a square matrix and Θ is also required to generate it in lower-triangular form along with X:

$$\begin{bmatrix} A & B \\ D & E \end{bmatrix} \Theta = \begin{bmatrix} \mathsf{X} & 0 \\ \mathsf{Y} & \mathsf{Z} \end{bmatrix} \tag{33.21}$$

The matrix Θ is not 2×2 any longer; but it can be implemented as a sequence of elementary (Givens) rotations or Householder reflections, as explained in App. 34, where we show how to lower triangularize a matrix via a sequence of rotations or reflections.

ARRAY REVIEW

An explicit expression for Θ in (33.21) is not needed. All we need to do is find the right sequence of rotations that yields the desired triangular post-array. Then, by "squaring" both sides of (33.21) we get

$$\begin{bmatrix} A & B \\ D & E \end{bmatrix} \underbrace{\Theta\Theta^*}_{\mathrm{I}} \begin{bmatrix} A & B \\ D & E \end{bmatrix}^* = \begin{bmatrix} \mathsf{X} & 0 \\ \mathsf{Y} & \mathsf{Z} \end{bmatrix} \begin{bmatrix} \mathsf{X} & 0 \\ \mathsf{Y} & \mathsf{Z} \end{bmatrix}^*$$

so that we must have

$$\mathsf{X}\mathsf{X}^* = AA^* + BB^* \qquad \text{and} \qquad \mathsf{Y}\mathsf{X}^* = DA^* + EB^*$$

In this way, X can be identified as the lower triangular Cholesky factor of the matrix $AA^* + BB^*$, and since Cholesky factors are unique, we conclude that X must coincide with the desired C. From the second equality above we conclude that $\mathsf{Y} = F$, so that the array algorithm (33.21) enables us to determine $\{C, F\}$. We may remark that we are not restricted to array methods with two (block) rows in the pre-array and post-arrays as in (33.21). If additional relations are available that satisfy certain norm and inner-product preservation properties, then these could be incorporated into the array algorithm as well. A demonstration to this effect is the QR algorithm of Sec. 35.2.

35.1 INVERSE QR ALGORITHM

The first array algorithm we derive is known as the inverse QR algorithm (and also as the square-root RLS algorithm). It is based on the observation that the expressions for $\{\gamma^{-1}(i), g_i\}$ in (35.2) can be put into the desirable forms (33.19) and (33.20). Indeed, note that

$$\left\{\begin{array}{lcl} \gamma^{-1}(i) & = & 1+\lambda^{-1}u_i P_{i-1}u_i^* \\ g_i\gamma^{-1}(i) & = & \lambda^{-1}P_{i-1}u_i^* \end{array}\right. \tag{35.7}$$

Comparing with (33.19) and (33.20) we see that we can make the identifications:

$$\left\{\begin{array}{lll} C \leftarrow \gamma^{-1/2}(i) & A \leftarrow 1 & B \leftarrow \lambda^{-1/2}u_i P_{i-1}^{1/2} \\ F \leftarrow g_i\gamma^{-1/2}(i) & D \leftarrow 0 & E \leftarrow \lambda^{-1/2}P_{i-1}^{1/2} \end{array}\right.$$

where $P_{i-1}^{1/2}$ denotes the Cholesky factor of P_{i-1}.

INVERSE QR

In other words, the expression for $\gamma^{-1}(i)$ in (35.7) corresponds to the norm preservation identity (remember that $\gamma(i)$ is a real variable even for complex data):

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

whereas the expression for $g_i\gamma^{-1}(i)$ in (35.7) corresponds to the inner-product preservation identity:

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

Therefore, motivated by the discussion that led to (33.21) from (33.19)–(33.20), we let Θ_i be a unitary matrix that transforms the pre-array

$$\mathcal{A} = \begin{bmatrix} 1 & \lambda^{-1/2} u_i P_{i-1}^{1/2} \\ 0 & \lambda^{-1/2} P_{i-1}^{1/2} \end{bmatrix}$$

INVERSE QR

to lower triangular form, say

$$\mathcal{B} = \begin{bmatrix} \times & 0 \\ \mathsf{y} & \mathsf{Z} \end{bmatrix}$$

for some variables $\{\times, \mathsf{y}, \mathsf{Z}\}$ to be determined, i.e.,

$$\begin{bmatrix} 1 & \lambda^{-1/2} u_i P_{i-1}^{1/2} \\ 0 & \lambda^{-1/2} P_{i-1}^{1/2} \end{bmatrix} \Theta_i = \begin{bmatrix} \times & 0 \\ \mathsf{y} & \mathsf{Z} \end{bmatrix} \tag{35.8}$$

where $\times$ is a scalar, y is a vector, and Z is a lower-triangular matrix with positive-diagonal entries. Observe that the pre- and post-arrays in (35.8) have the forms (assuming $M = 3$):

$$\left[\begin{array}{c|ccc} 1 & \times & \times & \times \\ \hline 0 & \times & 0 & 0 \\ 0 & \times & \times & 0 \\ 0 & \times & \times & \times \end{array}\right] \quad \text{and} \quad \left[\begin{array}{c|ccc} \times & 0 & 0 & 0 \\ \hline \times & \times & 0 & 0 \\ \times & \times & \times & 0 \\ \times & \times & \times & \times \end{array}\right]$$

respectively. We already know from the explanation in Sec. 33.4 that the values of $\times$ and y in (35.8) should be

$$\times = \gamma^{-1/2}(i), \qquad \mathsf{y} = g_i \gamma^{-1/2}(i)$$

INVERSE QR

Alternatively, we can identify the values of $\{\times, \mathsf{y}\}$, and also the value of Z, by "squaring" both sides of (35.8) to obtain the equality:

$$\begin{bmatrix} 1 & \lambda^{-1/2} u_i P_{i-1}^{1/2} \\ 0 & \lambda^{-1/2} P_{i-1}^{1/2} \end{bmatrix} \begin{bmatrix} 1 & \lambda^{-1/2} u_i P_{i-1}^{1/2} \\ 0 & \lambda^{-1/2} P_{i-1}^{1/2} \end{bmatrix}^* = \begin{bmatrix} \times & 0 \\ \mathsf{y} & \mathsf{Z} \end{bmatrix} \begin{bmatrix} \times & 0 \\ \mathsf{y} & \mathsf{Z} \end{bmatrix}^*$$

Comparing terms on both sides we find that the following equalities must hold:

$$\begin{cases} |\times|^2 &= 1 + \lambda^{-1} u_i P_{i-1} u_i^* \\ \mathsf{y}\times^* &= \lambda^{-1} P_{i-1} u_i^* \\ \mathsf{y}\mathsf{y}^* + \mathsf{Z}\mathsf{Z}^* &= \lambda^{-1} P_{i-1} \end{cases}$$

From the first equation we get $|\times|^2 = \gamma^{-1}(i)$ so that $\times = \gamma^{-1/2}(i)$. From the second equation we get $\mathsf{y} = g_i \gamma^{-1/2}(i)$, and from the third equation we get

$$\mathsf{Z}\mathsf{Z}^* = \lambda^{-1} P_{i-1} - \mathsf{y}\mathsf{y}^* = \lambda^{-1} P_{i-1} - g_i g_i^* / \gamma(i) = P_i$$

where the last equality follows from recursion (35.2). Therefore, since Z is lower triangular with positive diagonal entries, we conclude that Z is the Cholesky factor of P_i.

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.

TERMINOLOGY

Remark 35.1 (Terminology) The above array algorithm is known as the inverse QR method in the adaptive filtering literature for the following reason. The qualification *inverse* refers to the fact that $P_i^{1/2}$ is a square-root factor of the *inverse* of Φ_i in (35.4). Just note that since, by definition, $\Phi_i = P_i^{-1}$ and $P_i = P_i^{1/2} P_i^{*/2}$, we have $\Phi_i^{-1} = P_i^{1/2} P_i^{*/2}$. The qualification QR, on the other hand, is because this array method relates to the QR decomposition of a matrix, as explained in the third remark below. The algorithm is also known as square-root RLS since it propagates the square-root factor $P_i^{1/2}$; this latter terminology is borrowed from Kalman filtering — see Prob. VIII.12.

◇

Remark 35.2 (Reliability) By propagating a square-root factor of P_i, rather than P_i itself as in a plain RLS implementation, the danger of having P_i lose its positive-definiteness due to numerical inaccuracies is essentially eliminated. If needed, P_i can be recovered by squaring, i.e., via $P_i = P_i^{1/2} P_i^{*/2}$. However, this step is not necessary since the array method already evaluates the gain vector g_i and, moreover, the entries of its post-array contain everything we need in order to proceed to the next iteration.

◇

RELATION TO QR DECOMPOSITION

Remark 35.3 (Relation to the QR decomposition) If we conjugate-transpose the array equations of Alg. 35.1 we get

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

where the rightmost array is upper triangular with positive diagonal entries. Since Θ_i is unitary, this way of expressing the array equations amounts to a QR decomposition (cf. App. B.5) of the matrix

$$\begin{bmatrix} 1 & 0 \\ \lambda^{-1/2} P_{i-1}^{*/2} u_i^* & \lambda^{-1/2} P_{i-1}^{*/2} \end{bmatrix} \tag{35.9}$$

where the matrix Θ_i plays the role of the Q factor (recall the defining expression (B.9)). This observation shows that an alternative way of implementing the array algorithm is by performing the QR decomposition of the matrix (35.9).

$\diamond$

FURTHER MOTIVATION

Remark 35.4 (Further motivation) The inverse QR algorithm could have been alternatively derived as follows. Starting from the equations for $\{g_i, \gamma(i), P_i\}$ in (35.2), we note that they can be combined together in factored form as follows:

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

To verify that this is indeed the case, simply expand both sides and compare terms. Now this equality fits precisely into the statement of Lemma 33.1 by choosing

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

so that there should exist a unitary matrix Θ_i that takes A to B.

$\diamond$

35.2 QR ALGORITHM

The second array algorithm we consider is known as the QR algorithm (and also as the square-root information RLS algorithm). It follows from the observation that the recursions for $\{\Phi_i, s_i\}$ in (35.4) can be put into the desirable forms (33.19) and (33.20).

Indeed, let $\Phi_i^{1/2}$ denote the Cholesky factor of Φ_i, and introduce the auxiliary signals

$$\boxed{\bar{d}(i) \stackrel{\Delta}{=} d(i) - u_i \bar{w}, \qquad \bar{w}_i \stackrel{\Delta}{=} w_i - \bar{w}} \tag{35.10}$$

as well as the auxiliary column vector

$$\boxed{q_i \stackrel{\Delta}{=} \Phi_i^{*/2}[w_i - \bar{w}]} \tag{35.11}$$

These auxiliary signals would reduce to $d(i)$ and $\Phi^{*/2} w_i$, respectively, when $\bar{w} = 0$, which is usually the case. We consider $\bar{w}$ for generality.

QR ALGORITHM

From the equation for w_i in (35.5), it is seen that q_i satisfies

$$\boxed{\Phi_i^{1/2} q_i = s_i} \tag{35.12}$$

With $\{\bar{d}(i), \bar{w}_i, q_i\}$ so defined, we can rewrite the recursions in (35.4) as

$$\left\{ \begin{array}{lcl} \Phi_i^{1/2}\Phi_i^{*/2} & = & \lambda\Phi_{i-1}^{1/2}\Phi_{i-1}^{*/2} + u_i^* u_i \\ \Phi_i^{1/2} q_i & = & \lambda\Phi_{i-1}^{1/2} q_{i-1} + u_i^* \bar{d}(i) \end{array} \right. \tag{35.13}$$

If we conjugate the second equation, we get

$$\left\{ \begin{array}{lcl} \Phi_i^{1/2}\Phi_i^{*/2} & = & \lambda\Phi_{i-1}^{1/2}\Phi_{i-1}^{*/2} + u_i^* u_i \\ q_i^*\Phi_i^{*/2} & = & \lambda q_{i-1}^*\Phi_{i-1}^{*/2} + \bar{d}^*(i) u_i \end{array} \right. \tag{35.14}$$

Comparing with (33.19) and (33.20) we see that we can make the identifications:

$$\left\{ \begin{array}{lll} C \leftarrow \Phi_i^{1/2} & A \leftarrow \lambda^{1/2}\Phi_{i-1}^{1/2} & B \leftarrow u_i^* \\ F \leftarrow q_i^* & D \leftarrow \lambda^{1/2} q_{i-1}^* & E \leftarrow \bar{d}^*(i) \end{array} \right.$$

QR ALGORITHM

In other words, the expression for Φ_i in (35.14) corresponds to the norm preservation identity:

$$\begin{bmatrix} \Phi_i^{1/2} & 0 \end{bmatrix} \begin{bmatrix} \Phi_i^{*/2} \\ 0 \end{bmatrix} = \begin{bmatrix} \lambda^{1/2}\Phi_{i-1}^{1/2} & u_i^* \end{bmatrix} \begin{bmatrix} \lambda^{1/2}\Phi_{i-1}^{*/2} \\ u_i \end{bmatrix} \tag{35.15}$$

whereas the expression for s_i in (35.14) corresponds to the inner-product preservation identity:

$$\begin{bmatrix} q_i^* & \times \end{bmatrix} \begin{bmatrix} \Phi_i^{*/2} \\ 0 \end{bmatrix} = \begin{bmatrix} \lambda^{1/2}q_{i-1}^* & \bar{d}^*(i) \end{bmatrix} \begin{bmatrix} \lambda^{1/2}\Phi_{i-1}^{*/2} \\ u_i \end{bmatrix} \tag{35.16}$$

Therefore, motivated again by the discussion that led to (33.21) from (33.19)–(33.20), we let Θ_i be a unitary matrix that transforms the pre-array

$$\mathcal{A} = \begin{bmatrix} \lambda^{1/2}\Phi_{i-1}^{1/2} & u_i^* \\ \lambda^{1/2}q_{i-1}^* & \bar{d}^*(i) \end{bmatrix}$$

QR ALGORITHM

to lower triangular form, say

$$\mathcal{B} = \begin{bmatrix} \mathsf{X} & 0 \\ \mathsf{y} & \mathsf{z} \end{bmatrix}$$

for some variables $\{\mathsf{X}, \mathsf{y}, \mathsf{z}\}$ to be determined, with X lower triangular with positive diagonal entries, y a row vector, and z a scalar, i.e.,

$$\begin{bmatrix} \lambda^{1/2}\Phi_{i-1}^{1/2} & u_i^* \\ \lambda^{1/2} q_{i-1}^* & \bar{d}^*(i) \end{bmatrix} \Theta_i = \begin{bmatrix} \mathsf{X} & 0 \\ \mathsf{y} & \mathsf{z} \end{bmatrix} \tag{35.17}$$

Observe that the pre- and post-arrays in (35.17) have the forms (assuming $M = 3$):

$$\left[\begin{array}{ccc|c} \times & 0 & 0 & \times \\ \times & \times & 0 & \times \\ \times & \times & \times & \times \\ \hline \times & \times & \times & \times \end{array}\right] \quad \text{and} \quad \left[\begin{array}{ccc|c} \times & 0 & 0 & 0 \\ \times & \times & 0 & 0 \\ \times & \times & \times & 0 \\ \hline \times & \times & \times & \times \end{array}\right]$$

respectively. Therefore, the purpose of Θ_i is to introduce the zeros in the last column and to generate a lower triangular post-array with positive diagonal entries. This can be achieved, for example, by using Givens rotations and pivoting with the diagonal entries of $\lambda^{1/2}\Phi_{i-1}^{1/2}$ — as explained in App. 34.

QR ALGORITHM

Continuing with the array description (35.17), we already know from the explanation in Sec. 33.4 that the values of X and y in (35.17) should be $\mathsf{X} = \Phi_i^{1/2}$ and $\mathsf{y} = q_i^*$. Alternatively, we can identify the values of $\{\mathsf{X}, \mathsf{y}\}$, and also z, by "squaring" both sides of (35.17) to obtain the equality:

$$\begin{bmatrix} \lambda^{1/2}\Phi_{i-1}^{1/2} & u_i^* \\ \lambda^{1/2}q_{i-1}^* & \bar{d}^*(i) \end{bmatrix} \begin{bmatrix} \lambda^{1/2}\Phi_{i-1}^{1/2} & u_i^* \\ \lambda^{1/2}q_{i-1}^* & \bar{d}^*(i) \end{bmatrix}^* = \begin{bmatrix} \mathsf{X} & 0 \\ \mathsf{y} & \mathsf{z} \end{bmatrix} \begin{bmatrix} \mathsf{X} & 0 \\ \mathsf{y} & \mathsf{z} \end{bmatrix}^*$$

Then comparing terms on both sides we find that the following equalities must hold:

$$\begin{cases} \mathsf{X}\mathsf{X}^* &= \lambda\Phi_{i-1} + u_i^* u_i \\ \mathsf{y}\mathsf{X}^* &= \lambda s_{i-1}^* + u_i \bar{d}^*(i) \\ \mathsf{y}\mathsf{y}^* + \mathsf{z}\mathsf{z}^* &= \lambda\|q_{i-1}\|^2 + |\bar{d}(i)|^2 \end{cases}$$

From the first equation we get $\mathsf{X}\mathsf{X}^* = \Phi_i$. But since X is lower triangular with positive diagonal entries, we conclude that X is the Cholesky factor of Φ_i, namely,

$$\mathsf{X} = \Phi_i^{1/2}$$

QR ALGORITHM

From the second equation we get $\mathsf{y}\Phi_i^{*/2} = s_i^*$ so that, from (35.12),

$$\mathsf{y} = q_i^*$$

Thus, so far we have established that the array algorithm (35.17) leads to a recursion of the form

$$\begin{bmatrix} \lambda^{1/2}\Phi_{i-1}^{1/2} & u_i^* \\ \lambda^{1/2}q_{i-1}^* & \bar{d}^*(i) \end{bmatrix} \Theta_i = \begin{bmatrix} \Phi_i^{1/2} & 0 \\ q_i^* & \mathsf{z} \end{bmatrix} \tag{35.18}$$

Note in particular that the quantities $\{q_i, \Phi_i^{1/2}\}$ that are needed to form the next pre-array are available in the post-array and, hence, the procedure can be continued. However, it is useful to persist on a complete characterization of the variable z. The identification of z requires more effort.

QR ALGORITHM

From the third equation we find that the scalar z satisfies

$$\begin{aligned} \mathsf{z}\mathsf{z}^* &= \lambda\|q_{i-1}\|^2 + |\bar{d}(i)|^2 - \|\mathsf{y}\|^2 \\ &= \lambda\|q_{i-1}\|^2 + |\bar{d}(i)|^2 - \|q_i\|^2 \end{aligned}$$

Substituting q_i and q_{i-1} by their definitions (35.11), namely,

$$q_i = \Phi_i^{*/2}\bar{w}_i, \qquad q_{i-1} = \Phi_{i-1}^{*/2}\bar{w}_{i-1}$$

we obtain

$$\mathsf{z}\mathsf{z}^* = \lambda\bar{w}_{i-1}^*\Phi_{i-1}\bar{w}_{i-1} - \bar{w}_i^*\Phi_i\bar{w}_i + |\bar{d}(i)|^2$$

or, equivalently,

$$\mathsf{z}\mathsf{z}^* = \lambda\bar{w}_{i-1}^* s_{i-1} - s_i^*\bar{w}_i + |\bar{d}(i)|^2$$

since, from (35.4), $\Phi_i\bar{w}_i = s_i$ and $\Phi_{i-1}\bar{w}_{i-1} = s_{i-1}$. Using the time-update relation for s_i from (35.4) we find that

QR ALGORITHM

$$\begin{aligned} zz^* &= \lambda(\bar{w}_{i-1}^* s_{i-1} - s_{i-1}^* \bar{w}_i) + \bar{d}^*(i)[\bar{d}(i) - u_i \bar{w}_i] \\ &= \lambda(\bar{w}_{i-1}^* s_{i-1} - s_{i-1}^* \bar{w}_i) + \bar{d}^*(i)[d(i) - u_i w_i] \\ &= \lambda(\bar{w}_{i-1}^* s_{i-1} - s_{i-1}^* \bar{w}_i) + \bar{d}^*(i) r(i) \end{aligned} \tag{35.19}$$

in terms of the *a posteriori* error,

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

However, from

$$\Phi_{i-1} \bar{w}_{i-1} = s_{i-1}$$

we have

$$\lambda s_{i-1}^* \bar{w}_i = \lambda \bar{w}_{i-1}^* \Phi_{i-1} \bar{w}_i$$

so that

QR ALGORITHM

$$\begin{aligned}
\lambda(\bar{w}_{i-1}^{*}s_{i-1} - s_{i-1}^{*}\bar{w}_i) &= \bar{w}_{i-1}^{*}[\lambda s_{i-1} - \lambda\Phi_{i-1}\bar{w}_i] \\
&= \bar{w}_{i-1}^{*}[\lambda s_{i-1} - (\Phi_i - u_i^{*}u_i)\bar{w}_i] \\
&= \bar{w}_{i-1}^{*}[\lambda s_{i-1} - s_i + u_i^{*}u_i\bar{w}_i] \\
&= \bar{w}_{i-1}^{*}[-u_i^{*}\bar{d}(i) + u_i^{*}u_i\bar{w}_i] \\
&= -\bar{w}_{i-1}^{*}u_i^{*}[\bar{d}(i) - u_i\bar{w}_i] \\
&= -\bar{w}_{i-1}^{*}u_i^{*}r(i)
\end{aligned}$$

Substituting into (35.19) we find that

$$\begin{aligned}
\mathsf{z}\mathsf{z}^{*} &= [\bar{d}(i) - u_i\bar{w}_{i-1}]^{*}r(i) \\
&= [d(i) - u_i w_{i-1}]^{*}r(i)
\end{aligned}$$

or, equivalently,

$$\mathsf{z}\mathsf{z}^{*} = |\mathsf{z}|^2 = e^{*}(i)r(i) = |e(i)|^2\gamma(i) \tag{35.20}$$

in terms of the *a priori* error

$$e(i) = d(i) - u_i w_{i-1}$$

and the conversion factor $\gamma(i)$.

QR ALGORITHM

and the conversion factor $\gamma(i)$. Therefore, from the arguments presented to this point, we can only identify the magnitude of z, namely,

$$|\mathsf{z}| \;=\; |e(i)|\gamma^{1/2}(i) \tag{35.21}$$

More information is needed in order to identify the phase of z. To do so, we first note that the arrays (35.18) can be expanded in order to allow for the evaluation of the conversion factor as well. Specifically, by incorporating the row vector $[0 \quad 1]$ into the pre-array in (35.18), we obtain an array description of the form:

$$\begin{bmatrix} \lambda^{1/2}\Phi_{i-1}^{1/2} & u_i^* \\ \lambda^{1/2}q_{i-1}^* & \bar{d}^*(i) \\ 0 & 1 \end{bmatrix} \Theta_i \;=\; \begin{bmatrix} \Phi_i^{1/2} & 0 \\ q_i^* & \mathsf{z} \\ \mathsf{t} & \mathsf{s} \end{bmatrix} \tag{35.22}$$

for some row t and scalar s to be identified. Clearly, the value of s agrees with the rightmost diagonal entry of Θ_i, and this entry can be enforced to be positive — see Prob. VIII.6.

QR ALGORITHM

Equating the inner product of the top and last rows on both sides of (35.22) we get

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda^{1/2}\Phi_{i-1}^{*/2} \\ u_i \end{bmatrix} = \begin{bmatrix} \mathsf{t} & \mathsf{s} \end{bmatrix} \begin{bmatrix} \Phi_i^{*/2} \\ 0 \end{bmatrix}$$

or, equivalently, $u_i = \mathsf{t}\Phi_i^{*/2}$, so that

$$\mathsf{t} = u_i \Phi_i^{-*/2}$$

Likewise, equating the norms of the last rows on both sides of (35.22) we get

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathsf{t} & \mathsf{s} \end{bmatrix} \begin{bmatrix} \mathsf{t}^* \\ \mathsf{s}^* \end{bmatrix}$$

so that, from the just derived value for t,

$$1 = u_i \Phi_i^{-1} u_i^* + |\mathsf{s}|^2$$

QR ALGORITHM

Using expression (35.6) for $\gamma(i)$, and the fact that s is positive, we can identify s as

$$\mathsf{s} = \gamma^{1/2}(i)$$

Finally, equating the inner product of the last two rows in (35.22) we get

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda^{1/2} q_{i-1} \\ \bar{d}(i) \end{bmatrix} = \begin{bmatrix} \mathsf{t} & \mathsf{s} \end{bmatrix} \begin{bmatrix} q_i \\ \mathsf{z}^* \end{bmatrix}$$

i.e.,

$$\begin{aligned} \bar{d}(i) &= \left(u_i \Phi_i^{-*/2}\right) q_i + \mathsf{s}\mathsf{z}^* \\ &\overset{(35.12)}{=} u_i[w_i - \bar{w}] + \gamma^{1/2}(i)\mathsf{z}^* \end{aligned}$$

or, equivalently, by using $\bar{d}(i) = d(i) - u_i \bar{w}$,

$$d(i) - u_i w_i = \gamma^{1/2}(i)\mathsf{z}^* = r(i) \tag{35.23}$$

which allows us to identify z as $\mathsf{z} = e^*(i)\gamma^{1/2}(i)$

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.

FAST ARRAY RLS: MOTIVATION

All least-squares algorithms studied so far, including RLS, inverse QR, QR, and extended QR algorithms, do not assume any structure in the data. As a result, the computational complexity of each of these algorithms is $O(M^2)$ operations per iteration, where M is the order of the filter. However, when data structure is present, more efficient implementations are possible.

Thus consider a collection of $(N+1)$ data $\{d(j), u_j\}$ where the $\{u_j\}$ are $1 \times M$ and the $d(j)$ are scalars. All the aforementioned algorithms are recursive procedures for determining the solution w_N, and the minimum cost $\xi(N)$, of the regularized least-squares problem:

$$\min_{w} \left[\lambda^{(N+1)} (w - \bar{w})^* \Pi (w - \bar{w}) + \sum_{j=0}^{N} \lambda^{N-j} |d(j) - u_j w|^2 \right] \tag{37.1}$$

where $\bar{w}$ is $M \times 1$, $\Pi > 0$ is $M \times M$ and $0 \ll \lambda \leq 1$. In particular, RLS evaluates w_N recursively as follows (cf. Alg. 30.2). Start with $w_{-1} = \bar{w}$, $P_{-1} = \Pi^{-1}$, $\xi(-1) = 0$, and repeat for $i \geq 0$:

RLS ALGORITHM

$$
\begin{aligned}
\gamma(i) &= 1/(1+\lambda^{-1}u_i P_{i-1} u_i^*) \;=\; 1 - u_i P_i u_i^* \\
g_i &= \lambda^{-1}\gamma(i) P_{i-1} u_i^* \;=\; P_i u_i^* \\
e(i) &= d(i) - u_i w_{i-1} \\
w_i &= w_{i-1} + g_i e(i) \\
P_i &= \lambda^{-1} P_{i-1} - g_i g_i^* / \gamma(i) \\
r(i) &= d(i) - u_i w_i \\
\xi(i) &= \lambda \xi(i-1) \;+\; r(i) e^*(i)
\end{aligned}
\tag{37.2}
$$

These equations hold irrespective of any structure in the $\{u_j\}$.

DATA STRUCTURE

Now, it is often the case that the regressors $\{u_i\}$ exhibit some form of structure. In the chapters in this part, we study the case in which the $\{u_i\}$ arise as regressors of a tapped-delay-line implementation, as shown in Fig. 37.1. That is, we assume that the entries of u_i are formed from time-delayed samples of an input sequence $\{u(\cdot)\}$, say

$$u_i = \begin{bmatrix} u(i) & u(i-1) & \ldots & u(i-M+2) & u(i-M+1) \end{bmatrix} \tag{37.3}$$

In this case, two successive regressors will share most of their entries since, for example, u_{i-1} will be given by

$$u_{i-1} = \begin{bmatrix} u(i-1) & u(i-2) & \ldots & u(i-M+1) & u(i-M) \end{bmatrix} \tag{37.4}$$

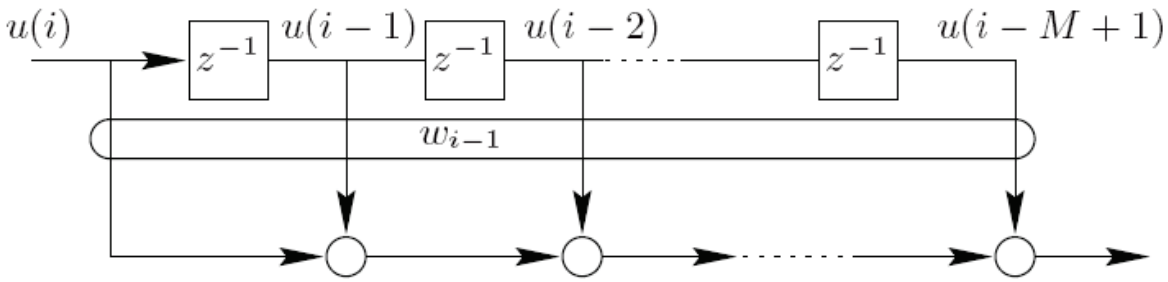


FIGURE 37.1 A tapped-delay-line structure resulting in regressors with shift-structure

DATA STRUCTURE

Comparing expressions (37.3) and (37.4) for u_i and u_{i-1} we see that u_i is obtained from u_{i-1} by shifting the entries of the latter by one position to the right and introducing a new entry, $u(i)$, at the left. One way to capture this relation is to note that the following equality holds:

$$\boxed{\begin{bmatrix} u(i) & u_{i-1} \end{bmatrix} = \begin{bmatrix} u_i & u(i-M) \end{bmatrix}} \tag{37.5}$$

That is, if we extend u_{i-1} by one entry to the left and u_i by one entry to the right, then the extended vectors will coincide.

The shift structure in the regressors can be exploited to great effect in order to devise efficient recursive least-squares solutions. By efficient we mean algorithms whose computational complexity is $O(M)$ operations per iteration, as opposed to $O(M^2)$; i.e., they are an order of magnitude more efficient than the slower implementations that we studied before (RLS, inverse QR, QR, extended QR). There are several classes of efficient RLS algorithms that can be derived by exploiting data structure. In this part we study fixed-order algorithms, while in Part X (*Lattice Filters*) we study order-recursive algorithms.

37.1 TIME-UPDATE OF THE GAIN VECTOR

Consider the RLS update (37.2) for w_i, namely,

$$w_i = w_{i-1} + g_i[d(i) - u_i w_{i-1}] \tag{37.6}$$

and note that RLS requires the gain vector g_i in order to compute w_i. In turn, the evaluation of g_i requires the matrix P_{i-1} (or P_i), and updating $\{P_{i-1}$ or $P_i\}$ requires $O(M^2)$ operations per iteration. This update step is the main computational bottleneck in the RLS algorithm. However, when the $\{u_i\}$ have shift structure, as indicated by (37.5), the gain vector g_i can be evaluated more immediately without requiring evaluation of P_{i-1} (or P_i). This will be achieved by developing a time-update for g_i itself, i.e., by showing how to compute g_i from g_{i-1} directly.

From the definition of g_i in (37.2) we have that

$$g_i\gamma^{-1}(i) = \lambda^{-1}P_{i-1}u_i^* \quad \text{and} \quad g_{i-1}\gamma^{-1}(i-1) = \lambda^{-1}P_{i-2}u_{i-1}^* \tag{37.7}$$

GAIN VECTOR

Now, because of the shift structure in the regressors $\{u_i, u_{i-1}\}$, we can express the extended vector $\text{col}\{P_{i-1}u_i^*, 0\}$ in the equivalent forms

$$\begin{bmatrix} P_{i-1}u_i^* \\ 0 \end{bmatrix} = \begin{bmatrix} P_{i-1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_i^* \\ u^*(i-M) \end{bmatrix} = \begin{bmatrix} P_{i-1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u^*(i) \\ u_{i-1}^* \end{bmatrix} \tag{37.8}$$

where in the second equality we used (37.5). Likewise, we can write

$$\begin{bmatrix} 0 \\ P_{i-2}u_{i-1}^* \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & P_{i-2} \end{bmatrix} \begin{bmatrix} u^*(i) \\ u_{i-1}^* \end{bmatrix} \tag{37.9}$$

Consequently, subtracting (37.8) and (37.9), we get

$$\lambda^{-1}\begin{bmatrix} P_{i-1}u_i^* \\ 0 \end{bmatrix} - \lambda^{-1}\begin{bmatrix} 0 \\ P_{i-2}u_{i-1}^* \end{bmatrix} = \lambda^{-1}\left(\begin{bmatrix} P_{i-1} & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & P_{i-2} \end{bmatrix}\right)\begin{bmatrix} u^*(i) \\ u_{i-1}^* \end{bmatrix}$$

GAIN VECTOR

Let δP_{i-1} denote the difference

$$\delta P_{i-1} \triangleq \begin{bmatrix} P_{i-1} & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & P_{i-2} \end{bmatrix} \qquad (M+1)\times(M+1)$$

Using the defining relations (37.7) for $\{g_i, g_{i-1}\}$, we arrive at

$$\boxed{\begin{bmatrix} g_i\gamma^{-1}(i) \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ g_{i-1}\gamma^{-1}(i-1) \end{bmatrix} + \lambda^{-1}\delta P_{i-1}\begin{bmatrix} u^*(i) \\ u_{i-1}^* \end{bmatrix}} \qquad (37.10)$$

This is a significant result. It shows that in order to time-update the gain vector from $g_{i-1}\gamma^{-1}(i-1)$ to $g_i\gamma^{-1}(i)$, it is only necessary to know what the difference δP_{i-1} is; it is not necessary to know the value of the individual matrices $\{P_i, P_{i-1}\}$ themselves. In this way, it suffices to know how to update δP_{i-1} to δP_i in order to carry out the updates:

$$g_{i-1}\gamma^{-1}(i-1) \xrightarrow{\delta P_{i-1}} g_i\gamma^{-1}(i) \xrightarrow{\delta P_i} g_{i+1}\gamma^{-1}(i+1) \longrightarrow \ldots$$

37.2 TIME-UPDATE OF THE CONVERSION FACTOR

Although knowledge of δP_{i-1} is enough to update $g_{i-1}\gamma^{-1}(i-1)$ to $g_i\gamma^{-1}(i)$, we still need to recover g_i itself. In other words, we still need to know how to remove the scaling by $\gamma^{-1}(i)$ from $g_i\gamma^{-1}(i)$. If we were to evaluate $\gamma(i)$ as suggested by the RLS implementation (37.2), in terms of P_{i-1} (or P_i), then we would be back to square one in terms of excessive computational complexity. However, it turns out that the conversion factor $\gamma(i)$ can also be time-updated in a manner that only requires knowledge of δP_{i-1}.

To see this, we use the expression for $\gamma(i)$ in (37.2) to write

$$\gamma^{-1}(i) = 1+\lambda^{-1}u_iP_{i-1}u_i^*, \quad \gamma^{-1}(i-1) = 1+\lambda^{-1}u_{i-1}P_{i-2}u_{i-1}^*$$

so that, upon subtraction, we get

$$\gamma^{-1}(i) = \gamma^{-1}(i-1) + \lambda^{-1}\left[u_iP_{i-1}u_i^* - u_{i-1}P_{i-2}u_{i-1}^*\right]$$

CONVERSION FACTOR

Again, using the partitioning (37.5), we can express the difference $\nabla = u_i P_{i-1} u_i^* - u_{i-1} P_{i-2} u_{i-1}^*$ as

$$\begin{aligned}\nabla &= \begin{bmatrix} u(i) & u_{i-1} \end{bmatrix} \left(\begin{bmatrix} P_{i-1} & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & P_{i-2} \end{bmatrix} \right) \begin{bmatrix} u^*(i) \\ u_{i-1}^* \end{bmatrix} \\ &= \begin{bmatrix} u(i) & u_{i-1} \end{bmatrix} \delta P_{i-1} \begin{bmatrix} u^*(i) \\ u_{i-1}^* \end{bmatrix}\end{aligned}$$

and, hence,

$$\boxed{\gamma^{-1}(i) = \gamma^{-1}(i-1) + \lambda^{-1} \begin{bmatrix} u(i) & u_{i-1} \end{bmatrix} \delta P_{i-1} \begin{bmatrix} u^*(i) \\ u_{i-1}^* \end{bmatrix}} \tag{37.11}$$

which shows, as desired, that knowledge of δP_{i-1} is also sufficient for the time-update of the conversion factor.

37.3 INITIAL CONDITIONS

In order to complete the argument, we need to show how to compute the factor δP_{i-1} that appears in (37.10) and (37.11) in an efficient manner. This step requires that the regularization matrix Π be chosen in an appropriate manner, as we now explain.

Assume $u(i) = 0$ for $i < 0$ so that $u_{-1} = 0$ (i.e., the initial state of the filter is zero). Then $g_{-1} = 0, \gamma(-1) = 1, P_{-1} = \Pi^{-1}$, and $P_{-2} = \lambda P_{-1} = \lambda\Pi^{-1}$, so that the difference δP_i at time -1 is given by

$$\delta P_{-1} = \begin{bmatrix} P_{-1} & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & P_{-2} \end{bmatrix} = \begin{bmatrix} \Pi^{-1} & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & \lambda\Pi^{-1} \end{bmatrix} \tag{37.12}$$

The value of this difference depends on our choice of Π. So assume that we choose Π^{-1} as the diagonal matrix

$$\boxed{\Pi^{-1} = \eta \cdot \text{diag}\left\{\lambda^2, \lambda^3, \ldots, \lambda^{M+1}\right\}} \tag{37.13}$$

where η is a positive scalar (usually large) and λ is the forgetting factor

INITIAL CONDITIONS

Then δP_{-1} becomes $\delta P_{-1} = \eta\lambda^2 \cdot \text{diag}\{1, 0, \ldots, 0, -\lambda^M\}$. That is, δP_{-1} reduces to a rank-two matrix with one positive eigenvalue and one negative eigenvalue. Actually, it is not difficult to verify that δP_{-1} can be factored as

$$\boxed{\delta P_{-1} = \lambda \cdot \bar{L}_{-1} S_{-1} \bar{L}_{-1}^*}$$

where $\bar{L}_{-1}$ is $(M+1) \times 2$ and S_{-1} is 2×2 and given by

$$\bar{L}_{-1} = \sqrt{\eta\lambda} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & \lambda^{M/2} \end{bmatrix}, \qquad S_{-1} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{37.14}$$

The signature matrix, S_{-1}, indicates that the difference δP_{-1} has one positive eigenvalue and one negative eigenvalue; all other eigenvalues are zero since the rank is 2.

INITIAL CONDITIONS

This argument shows that the choice (37.13) for Π leads to a rank 2 difference matrix δP_{-1} with signature $(1 \oplus -1)$. A striking property that is established in the next subsection is that the rank and inertia properties of δP_i remain *invariant* over time; the rank never goes up; it is always 2, with the same signature matrix S_{-1}, except in rare degenerate situations where the rank can only drop below 2. More specifically, we argue ahead that once the low-rank property holds at a certain time instant $i-1$, say

$$\begin{bmatrix} P_{i-1} & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & P_{i-2} \end{bmatrix} = \lambda \cdot \bar{L}_{i-1} S_{i-1} \bar{L}_{i-1}^* \tag{37.15}$$

for some $\{\bar{L}_{i-1}, S_{i-1}\}$, then three important facts hold:

1. The low-rank property will be valid at time i as well, say

$$\begin{bmatrix} P_i & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & P_{i-1} \end{bmatrix} = \lambda \cdot \bar{L}_i S_i \bar{L}_i^* \qquad \text{for some } \{\bar{L}_i, S_i\}.$$

2. There will exist an array algorithm that updates $\bar{L}_{i-1}$ to $\bar{L}_i$, and this same algorithm will also provide the gain vector g_i that is needed in (37.6).

3. The signature matrices $\{S_{i-1}, S_i\}$ will coincide.

37.4 ARRAY ALGORITHM

The desired array algorithm for updating $\{\bar{L}_{i-1}, g_{i-1}, \gamma(i-1)\}$ to $\{\bar{L}_i, g_i, \gamma(i)\}$ can be motivated and derived in much the same manner as we did in Sec. 33.4.

Thus starting from (37.10) and (37.11), namely,

$$\gamma^{-1}(i) = \gamma^{-1}(i-1) + \begin{bmatrix} u(i) & u_{i-1} \end{bmatrix} \bar{L}_{i-1} S_{i-1} \bar{L}_{i-1}^* \begin{bmatrix} u^*(i) \\ u_{i-1}^* \end{bmatrix}$$

$$\begin{bmatrix} g_i \gamma^{-1}(i) \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ g_{i-1}\gamma^{-1}(i-1) \end{bmatrix} + \bar{L}_{i-1} S_{i-1} \bar{L}_{i-1}^* \begin{bmatrix} u^*(i) \\ u_{i-1}^* \end{bmatrix}$$

we note that these expressions are of the form

$$CC^* = AA^* + BSB^*, \qquad FC^* = DA^* + ESB^* \tag{37.16}$$

with the following identifications

FAST ARRAY ALGORITHM

$$\begin{cases} C \leftarrow \gamma^{-1/2}(i) \\ F \leftarrow \begin{bmatrix} g_i \gamma^{-1/2}(i) \\ 0 \end{bmatrix} \\ S \leftarrow S_{i-1} \end{cases} \qquad \begin{matrix} A \leftarrow \gamma^{-1/2}(i-1) \\ D \leftarrow \begin{bmatrix} 0 \\ g_{i-1}\gamma^{-1/2}(i-1) \end{bmatrix} \end{matrix} \qquad \begin{matrix} B \leftarrow \begin{bmatrix} u(i) & u_{i-1} \end{bmatrix} \bar{L}_{i-1} \\ E \leftarrow \bar{L}_{i-1} \end{matrix}$$

Generic Array Algorithm

The forms (37.16) play a role similar to the "norm-preserving" and "inner-product preserving" equalities (33.19) and (33.20), which we used in Sec. 33.4 to motivate and derive array algorithms. The main distinction is the presence of the signature matrix S in (37.16). Still, the same arguments that we employed in Sec. 33.4 could be repeated here with the only issue being that we now need to deal with hyperbolic transformations as opposed to unitary transformations.

More specifically, as in (33.21), given general equalities of the form (37.16) where it is desired to evaluate $\{C, F\}$ from knowledge of $\{A, B, D, E, S\}$, we would form the pre-array

FAST ARRAY ALGORITHM

More specifically, as in (33.21), given general equalities of the form (37.16) where it is desired to evaluate $\{C, F\}$ from knowledge of $\{A, B, D, E, S\}$, we would form the pre-array

$$\mathcal{A} = \begin{bmatrix} A & B \\ D & E \end{bmatrix} \quad \text{and reduce it to the form} \quad \begin{bmatrix} \mathrm{X} & 0 \\ \mathrm{Y} & \mathrm{Z} \end{bmatrix} \tag{37.17}$$

by annihilating B by means of a transformation Θ that is now required to be $(\mathrm{I} \oplus S)$-unitary, i.e., it should satisfy

$$\Theta \begin{bmatrix} \mathrm{I} & \\ & S \end{bmatrix} \Theta^* = \begin{bmatrix} \mathrm{I} & \\ & S \end{bmatrix}$$

The question of course is whether such a Θ exists. While in Sec. 33.4 we started from equality (33.19) and appealed to (33.9) to justify the existence of a Θ that achieves (33.21), this same argument cannot be applied to the equality

$$CC^* = AA^* + BSB^* \tag{37.18}$$

due to the presence of the signature matrix S.

FAST ARRAY ALGORITHM

due to the presence of the signature matrix S. However, Lemma 36.5 already extends the result (33.9) to handle such more general cases with signature matrices. Specifically, by writing (37.18) as

$$\begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} C^* \\ 0 \end{bmatrix} = \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} \mathrm{I} & \\ & S \end{bmatrix} \begin{bmatrix} A^* \\ B^* \end{bmatrix}$$

we are able to appeal to Lemma 36.5 to conclude that an $(\mathrm{I} \oplus S)-$unitary Θ exists that maps $[A \quad B]$ to $[C \quad 0]$. Therefore, in a manner similar to the explanations that led to (33.21), in order to determine $\{C, F\}$ we would first use one such $(\mathrm{I} \oplus S)-$unitary matrix Θ to perform the transformation

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \Theta = \begin{bmatrix} \mathsf{X} & 0 \\ \mathsf{Y} & \mathsf{Z} \end{bmatrix}$$

and then "square" and compare entries on both sides of the equality:

FAST ARRAY ALGORITHM

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \underbrace{\Theta \begin{bmatrix} \mathrm{I} & \\ & S \end{bmatrix} \Theta^*}_{(\mathrm{I} \oplus S)} \begin{bmatrix} A & B \\ C & D \end{bmatrix}^* = \begin{bmatrix} \mathsf{X} & 0 \\ \mathsf{Y} & \mathsf{Z} \end{bmatrix} \begin{bmatrix} \mathrm{I} & \\ & S \end{bmatrix} \begin{bmatrix} \mathsf{X} & 0 \\ \mathsf{Y} & \mathsf{Z} \end{bmatrix}^*$$

in order to identify X as C and Y as F.

Fast RLS Array Algorithm

Applying this construction to the RLS case (37.16), we would first form the pre-array

$$\mathcal{A} = \begin{bmatrix} \gamma^{-1/2}(i-1) & \begin{bmatrix} u(i) & u_{i-1} \end{bmatrix} \bar{L}_{i-1} \\ \begin{bmatrix} 0 \\ g_{i-1}\gamma^{-1/2}(i-1) \end{bmatrix} & \bar{L}_{i-1} \end{bmatrix}$$

FAST ARRAY ALGORITHM

For example, when $M = 3$, this array has the form (recall that $\bar{L}_i$ is $(M+1) \times 2$):

$$\mathcal{A} = \left[\begin{array}{c|cc} \times & \times & \times \\ \hline 0 & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{array}\right]$$

Then we would define the signature matrix $J = (1 \oplus S_{i-1})$, and choose a $J-$unitary matrix Θ_i (i.e., $\Theta_i J \Theta_i^* = J$) such that it transforms $\mathcal{A}$ to the form

$$\mathcal{B} = \begin{bmatrix} \times & 0 \\ \mathsf{y} & \mathsf{Z} \end{bmatrix} \tag{37.19}$$

for some quantities $\{\times, \mathsf{y}, \mathsf{Z}\}$ to be determined, with $\times$ a positive scalar, y a column vector, and Z a two column matrix. Again, for $M = 3$, the post-array will be of the form:

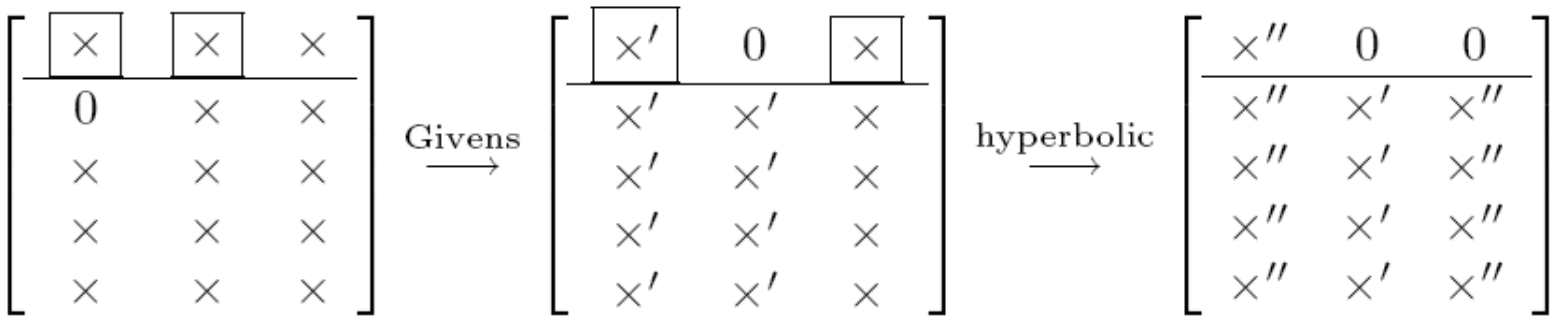
FAST ARRAY ALGORITHM

and Z a two column matrix. Again, for $M = 3$, the post-array will be of the form:

$$\mathcal{B} = \left[\begin{array}{c|cc} \times & 0 & 0 \\ \hline \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{array}\right]$$

The matrix Θ_i can be implemented in many ways (as described in Chapter 36). For example, we could employ a circular (Givens) rotation that pivots with the left-most entry of the first row and annihilates its second entry. We could then employ a hyperbolic rotation that pivots again with the left-most entry and annihilates the last entry of the same row:

FAST ARRAY ALGORITHM



Now, in order to identify the entries $\{\times, \mathsf{y}, \mathsf{Z}\}$ in the post-array (37.19), i.e., in the equality below

$$\underbrace{\begin{bmatrix} \gamma^{-1/2}(i-1) & \begin{bmatrix} u(i) & u_{i-1} \end{bmatrix} \bar{L}_{i-1} \\ \begin{bmatrix} 0 \\ g_{i-1}\gamma^{-1/2}(i-1) \end{bmatrix} & \bar{L}_{i-1} \end{bmatrix}}_{\mathcal{A}} \Theta_i = \underbrace{\begin{bmatrix} \times & 0 \\ \mathsf{y} & \mathsf{Z} \end{bmatrix}}_{\mathcal{B}}$$

we simply compare entries on both sides of the equality $\mathcal{A}\Theta_i J \Theta_i^* \mathcal{A}^* = \mathcal{B} J \mathcal{B}^*$ to find that

FAST ARRAY ALGORITHM

$$\begin{cases} |\times|^2 & = \gamma^{-1}(i-1) + \begin{bmatrix} u(i) & u_{i-1} \end{bmatrix} \bar{L}_{i-1} S_{i-1} \bar{L}_{i-1}^* \begin{bmatrix} u^*(i) \\ u_{i-1}^* \end{bmatrix} \\ \mathsf{y}\times^* & = \begin{bmatrix} 0 \\ g_{i-1}\gamma^{-1}(i-1) \end{bmatrix} + \bar{L}_{i-1} S_{i-1} \bar{L}_{i-1}^* \begin{bmatrix} u^*(i) \\ u_{i-1}^* \end{bmatrix} \\ \mathsf{y}\mathsf{y}^* + \mathsf{Z} S_{i-1} \mathsf{Z}^* & = \begin{bmatrix} 0 \\ g_{i-1}\gamma^{-1/2}(i) \end{bmatrix} \begin{bmatrix} 0 \\ g_{i-1}\gamma^{-1/2}(i-1) \end{bmatrix}^* + \bar{L}_{i-1} S_{i-1} \bar{L}_{i-1}^* \end{cases} \tag{37.20}$$

The right-hand side of the first equality coincides with that of (37.11) so that we can identify $\times$ as $\times = \gamma^{-1/2}(i)$. Similarly, the right-hand side of the second equality in (37.20) coincides with that of (37.10), so that we can identify y as

$$\mathsf{y} = \begin{bmatrix} g_i \gamma^{-1/2}(i) \\ 0 \end{bmatrix}$$

FAST ARRAY ALGORITHM

Finally, the last equality in (37.20) leads to

$$
\begin{aligned}
\mathsf{Z}S_{i-1}\mathsf{Z}^* &= \begin{bmatrix} 0 \\ g_{i-1}\gamma^{-1/2}(i-1) \end{bmatrix}\begin{bmatrix} 0 \\ g_{i-1}\gamma^{-1/2}(i-1) \end{bmatrix}^* \\
&\quad + \begin{bmatrix} \lambda^{-1}P_{i-1} & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & \lambda^{-1}P_{i-2} \end{bmatrix} - \begin{bmatrix} g_i\gamma^{-1/2}(i) \\ 0 \end{bmatrix}\begin{bmatrix} g_i\gamma^{-1/2}(i) \\ 0 \end{bmatrix}^* \\
&= \begin{bmatrix} 0 & 0 \\ 0 & \lambda^{-1}P_{i-2} - P_{i-1} \end{bmatrix} + \begin{bmatrix} \lambda^{-1}P_{i-1} & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & \lambda^{-1}P_{i-2} \end{bmatrix} \\
&\quad - \begin{bmatrix} \lambda^{-1}P_{i-1} - P_i & 0 \\ 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} P_i & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & P_{i-1} \end{bmatrix} = \delta P_i
\end{aligned}
$$

The difference δP_i is, by definition, $\lambda \bar{L}_i S_i \bar{L}_i^*$, so that $\mathsf{Z}S_{i-1}\mathsf{Z}^* = \lambda \bar{L}_i S_i \bar{L}_i^*$. This result shows that the difference δP_i has the same signature matrix, S_{i-1}, as δP_{i-1} and, consequently, S_i remains invariant for all i. For this reason, we shall drop the time subscript i from S_i and write S instead. Moreover, we can identify Z as $\mathsf{Z} = \sqrt{\lambda} \cdot \bar{L}_i$.

FAST ARRAY ALGORITHM

Algorithm 37.1 (Fast RLS array 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 $\bar{w}$, a scalar $0 \ll \lambda \leq 1$, and an $M \times M$ positive-definite matrix Π^{-1} of the form

$$\Pi^{-1} = \eta \cdot \text{diagonal}\,\{\lambda^2, \lambda^3, \ldots, \lambda^{M+1}\}, \quad \eta > 0$$

When the $\{u_j\}$ correspond to regressors of a tapped-delay-line implementation, the solution w_N of the least-squares problem (37.1) can be recursively computed as follows. Start with $w_{-1} = \bar{w}$, $\gamma^{-1/2}(-1) = 1$, $g_{-1} = 0$, $\bar{L}_{-1}$ and S as in (37.14), $J = (1 \oplus S)$, and repeat for $i \geq 0$:

1. Find a $J-$unitary matrix Θ_i that annihilates the last two entries in the top row of the post-array below and generates a positive leading entry. Then the entries of the post-array will correspond to

$$\begin{bmatrix} \gamma^{-1/2}(i-1) & \begin{bmatrix} u(i) & u_{i-1} \end{bmatrix} \bar{L}_{i-1} \\ \begin{bmatrix} 0 \\ g_{i-1}\gamma^{-1/2}(i-1) \end{bmatrix} & \bar{L}_{i-1} \end{bmatrix} \Theta_i$$

$$= \begin{bmatrix} \gamma^{-1/2}(i) & \begin{bmatrix} 0 & 0 \end{bmatrix} \\ \begin{bmatrix} g_i\gamma^{-1/2}(i) \\ 0 \end{bmatrix} & \sqrt{\lambda}\,\bar{L}_i \end{bmatrix}$$

2. Update $w_i = w_{i-1} + \left[g_i\gamma^{-1/2}(i)\right]\left[\gamma^{-1/2}(i)\right]^{-1}\left[d(i) - u_i w_{i-1}\right]$, where the quantities $\{g_i\gamma^{-1/2}(i), \gamma^{-1/2}(i)\}$ are read from the post-array.

FAST TRANSVERSAL FILTER

Algorithm 39.1 (Fast transversal filter) Consider the same setting as Alg. 37.1 with $\bar{w} = 0$. The solution w_N can be recursively computed as follows. Start with $u_{M,-1} = 0$, $\gamma_M(-1) = 1$, $c_{M,-1} = 0$, $w^f_{M,-1} = 0$, $w^b_{M,-1} = 0$, $w_{M,-1} = 0$, $\zeta^f_M(-1) = \eta^{-1}\lambda^{-2}$, $\zeta^b_M(-1) = \eta^{-1}\lambda^{-(M+2)}$ and repeat for $i \geq 0$:

$$
\begin{aligned}
\alpha_M(i) &= u(i) - u_{M,i-1} w^f_{M,i-1} \\
f_M(i) &= \gamma_M(i-1)\alpha_M(i) \\
\zeta^f_M(i) &= \lambda\zeta^f_M(i-1) + \alpha^*_M(i) f_M(i) \\
\gamma_{M+1}(i) &= \gamma_M(i-1)\ \lambda\zeta^f_M(i-1)/\zeta^f_M(i) \\
w^f_{M,i} &= w^f_{M,i-1} + f_M(i) c_{M,i-1} \\
c_{M+1,i} &= \begin{bmatrix} 0 \\ c_{M,i-1} \end{bmatrix} + \frac{\alpha^*_M(i)}{\lambda\zeta^f_M(i-1)} \begin{bmatrix} 1 \\ -w^f_{M,i-1} \end{bmatrix} \\
\nu_{M+1}(i) &= \text{last entry of } c_{M+1,i} \\
\beta_M(i) &= \lambda\zeta^b_M(i-1)\nu^*_{M+1}(i) \\
\gamma_M(i) &= \gamma_{M+1}(i)/[1 - \beta_M(i)\gamma_{M+1}(i)\nu_{M+1}(i)] \\
\begin{bmatrix} c_{M,i} \\ 0 \end{bmatrix} &= c_{M+1,i} - \nu_{M+1}(i) \begin{bmatrix} -w^b_{M,i-1} \\ 1 \end{bmatrix} \\
b_M(i) &= \gamma_M(i)\beta_M(i) \\
\zeta^b_M(i) &= \lambda\zeta^b_M(i-1) + \beta^*_M(i) b_M(i) \\
w^b_{M,i} &= w^b_{M,i-1} + b_M(i) c_{M,i} \\
e(i) &= d(i) - u_{M,i} w_{i-1} \\
r(i) &= \gamma_M(i) e(i) \\
w_i &= w_{i-1} + r(i) c_{M,i}
\end{aligned}
$$

FAEST FILTER

Algorithm 39.2 (FAEST filter) Consider the same setting as Alg. 37.1 with $\bar{w} = 0$. The solution w_N can be recursively computed as follows. Start with $u_{M,-1} = 0$, $\gamma_M^{-1}(-1) = 1$, $c_{M,-1} = 0$, $w_{M,-1}^f = 0$, $w_{M,-1}^b = 0$, $w_{M,-1} = 0$, $\zeta_M^f(-1) = \eta^{-1}\lambda^{-2}$, $\zeta_M^b(-1) = \eta^{-1}\lambda^{-(M+2)}$ and repeat for $i \geq 0$:

$$
\begin{aligned}
\alpha_M(i) &= u(i) - u_{M,i-1} w_{M,i-1}^f \\
f_M(i) &= \alpha_M(i)/\gamma_M^{-1}(i-1) \\
\zeta_M^f(i) &= \lambda \zeta_M^f(i-1) + \alpha_M^*(i) f_M(i) \\
\gamma_{M+1}^{-1}(i) &= \gamma_M^{-1}(i-1) + |\alpha_M(i)|^2/\lambda\zeta_M^f(i-1) \\
w_{M,i}^f &= w_{M,i-1}^f + f_M(i) c_{M,i-1} \\
c_{M+1,i} &= \begin{bmatrix} 0 \\ c_{M,i-1} \end{bmatrix} + \frac{\alpha_M^*(i)}{\lambda\zeta_M^f(i-1)} \begin{bmatrix} 1 \\ -w_{M,i-1}^f \end{bmatrix} \\
\nu_{M+1}(i) &= \text{last entry of } c_{M+1,i} \\
\beta_M(i) &= \lambda \zeta_M^b(i-1) \nu_{M+1}^*(i) \\
\gamma_M^{-1}(i) &= \gamma_{M+1}^{-1}(i) - \beta_M(i)\nu_{M+1}(i) \\
\begin{bmatrix} c_{M,i} \\ 0 \end{bmatrix} &= c_{M+1,i} - \nu_{M+1}(i) \begin{bmatrix} -w_{M,i-1}^b \\ 1 \end{bmatrix} \\
b_M(i) &= \beta_M(i)/\gamma_M^{-1}(i) \\
\zeta_M^b(i) &= \lambda\zeta_M^b(i-1) + \beta_M^*(i) b_M(i) \\
w_{M,i}^b &= w_{M,i-1}^b + b_M(i) c_{M,i} \\
e(i) &= d(i) - u_{M,i} w_{i-1} \\
r(i) &= e(i)/\gamma_M^{-1}(i) \\
w_i &= w_{i-1} + r(i) c_{M,i}
\end{aligned}
$$

FAST KALMAN FILTER

Algorithm 39.3 (Fast Kalman filter) Consider the same setting as Alg. 37.1 with $\bar{w} = 0$. The solution w_N can be recursively computed as follows. Start with $u_{M,-1} = 0$, $g_{M,-1} = 0$, $w^f_{M,-1} = 0$, $w^b_{M,-1} = 0$, $w_{M,-1} = 0$, $\zeta^f_M(-1) = \eta^{-1}\lambda^{-2}$, and repeat for $i \geq 0$:

$$
\begin{aligned}
\alpha_M(i) &= u(i) - u_{M,i-1} w^f_{M,i-1} \\
w^f_{M,i} &= w^f_{M,i-1} + \alpha_M(i) g_{M,i-1} \\
f_M(i) &= u(i) - u_{M,i-1} w^f_{M,i} \\
\zeta^f_M(i) &= \lambda \zeta^f_M(i-1) + \alpha^*_M(i) f_M(i) \\
g_{M+1,i} &= \begin{bmatrix} 0 \\ g_{M,i-1} \end{bmatrix} + \frac{f^*_M(i)}{\zeta^f_M(i)} \begin{bmatrix} 1 \\ -w^f_{M,i} \end{bmatrix} \\
\sigma_{M+1}(i) &= \text{last entry of } g_{M+1,i} \\
\beta_M(i) &= u(i-M) - u_{M,i} w^b_{M,i-1} \\
g_{M,i} &= \frac{g_{M+1,i}[0 : M-1]}{1 - \sigma_{M+1}(i)\beta_M(i)} \\
w^b_{M,i} &= w^b_{M,i-1} + \beta_M(i) g_{M,i} \\
e(i) &= d(i) - u_{M,i} w_{i-1} \\
w_i &= w_{i-1} + g_{M,i} e(i)
\end{aligned}
$$

COMPUTER PROJECT: QR METHODS

Project VIII.1 (Performance of array implementations in finite precision) The purpose of this project is to compare the performance of RLS and one of its array variants in finite-precision. Refer to the channel estimation application shown in Fig. VIII.1.

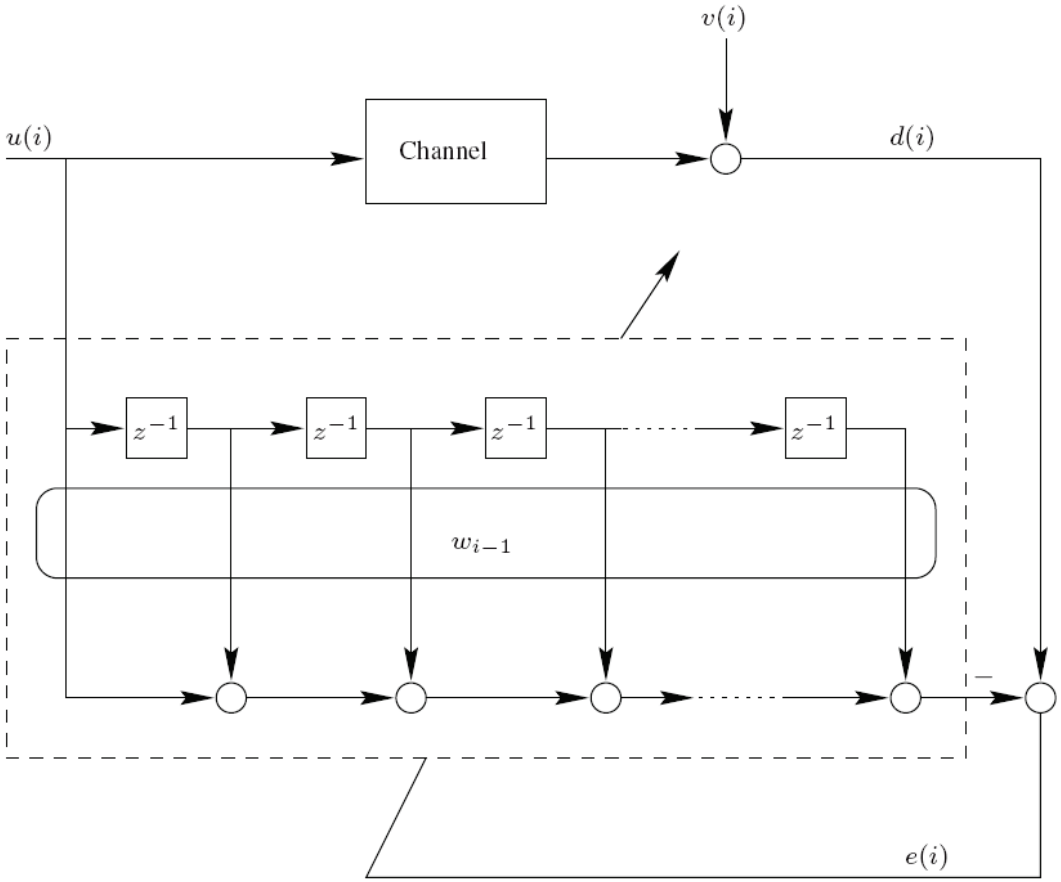


FIGURE VIII.1 Adaptive configuration for channel estimation.

COMPUTER PROJECT

Generate five random samples of a channel impulse response sequence and normalize the norm of this impulse response to unity. Feed unit-variance Gaussian input data through the channel and add Gaussian noise to its output. Set the noise power at 30 dB below the input signal power. Train an adaptive filter for $N = 200$ iterations using RLS and the QR array method of Alg. 35.2. Use $\lambda = 0.995$, $\Pi = 1 \times 10^{-6}\mathrm{I}$ and $\bar{w} = 0$. Assume also a finite-precision implementation is used with B_r bits for signals and B_c bits for coefficients (including the sign bit in both cases). In order to simulate the quantized behavior of the filters, you may use a routine quantize.m. The routine receives as input two parameters: a real number x and the total number of bits, B (including the sign bit), for its quantized representation. The routine then returns a real number that corresponds to the quantized value of x. This value is determined as follows. With B total bits, the largest integers that can be represented are

$$\pm 2^{(B-1)} - 1$$

If x exceeds these extreme values then its quantized representation is taken as either one of them, depending on the sign of x. If, on the other hand, x falls within the interval

$$x \in \left(-2^{(B-1)} - 1,\ 2^{(B-1)} - 1 \right)$$

then the routine determines how many bits are needed to represent the integer part of x, and the remaining bits are used to represent the fractional part of x.

COMPUTER PROJECT

For each algorithm (RLS and QR array method), generate an ensemble-average learning curve by averaging over 100 experiments and for the following choices:

1. $B_r = B_c = 30$ bits.
2. $B_r = B_c = 25$ bits.
3. $B_r = B_c = 16$ bits.

COMPUTER PROJECT

Project VIII.1 (Performance of array implementations in finite precision) The program that solves this probject is the following.

1. partA.m The program generates ensemble-average learning curves for RLS and its QR array variant using three choices for the number of bits, $\{30, 25, 16\}$ and averaging over 100 experiments. A typical output is shown in Fig. 1. It is seen that the QR array method is superior in finite-precision implementations. For instance, at 16 bits, RLS fails while the QR method still functions properly.

COMPUTER PROJECT

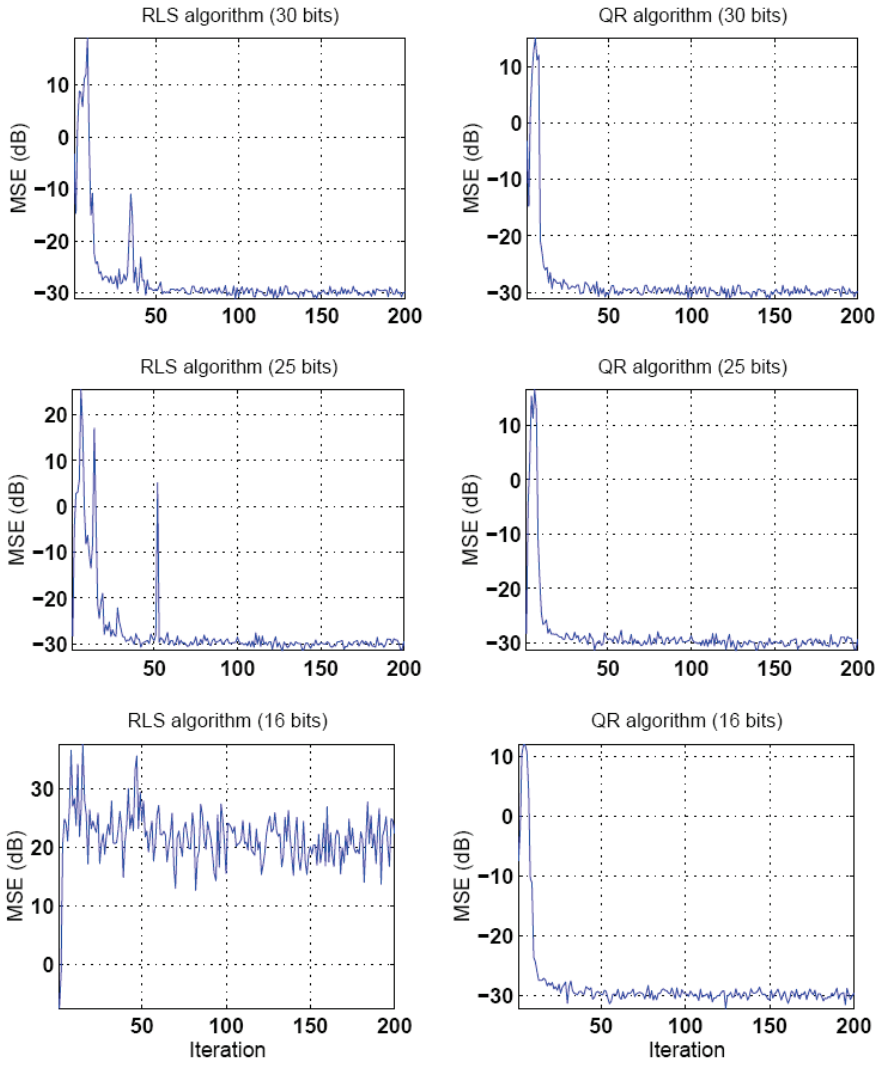


Figure VIII.1. Ensemble-average learning curves for RLS and its QR array variant for three choices of the number of quantization bits.

COMPUTER PROJECT: FAST RLS

Project IX.1 (Stability issues in fast least-squares) The purpose of this project is to illustrate some of the stability problems that arise when dealing with fast fixed-order least-squares algorithms. We do so by considering the same adaptive channel estimation application shown in Fig. VIII.1.

(a) Load the file channel.mat. It contains the 64 samples of a randomly generated channel impulse response sequence. The norm of this impulse response has been normalized to unity. Feed unit-variance Gaussian input data through the channel and add Gaussian noise to its output. Set the noise power at 30 dB below the input signal power. Train an adaptive filter using:

1. The fast transversal filter of Alg. 39.1;
2. The same FTF algorithm but incorporating the rescue mechanism (39.10);
3. The stabilized FTF implementation of Alg. 39.4;
4. The fast array filter of Alg. 37.1.

For each algorithm, generate an ensemble-average learning curve by averaging over 100 experiments. Compare the performance of the algorithms.

COMPUTER PROJECT

(b) All computations in MATLAB are performed in double precision. In order to illustrate some of the instability problems that arise in finite precision, we repeat the simulation of part (a) in a quantized environment. To do so, you may rely on the routine quantize.m from Computer Project VIII.1. Usually, in practice, the internal variables of a filter implementation are suitably scaled in order to reduce the occurrences of overflow and underflow when operating in finite precision. In order to simplify this project, and thereby avoid the need for incorporating scaling, we shall choose the number of bits to be relatively high. Thus fix the word-length for both data and coefficients at $B = 36$ bits (including the sign bit). Although high, this number of bits will still be useful to illustrate some quantization effects.

Repeat the simulation of part (a) and generate ensemble-average learning curves for each of the algorithms. Compare their performance.

COMPUTER PROJECT

Project IX.1 (Stability issues in fast least-squares) The programs that solve this project are the following.

1. partA.m This program generates ensemble-average learning curves for four fast fixed-order implementations. A typical output is shown in Fig. 1. It is seen that in floating-point precision, the algorithms behave rather similarly.

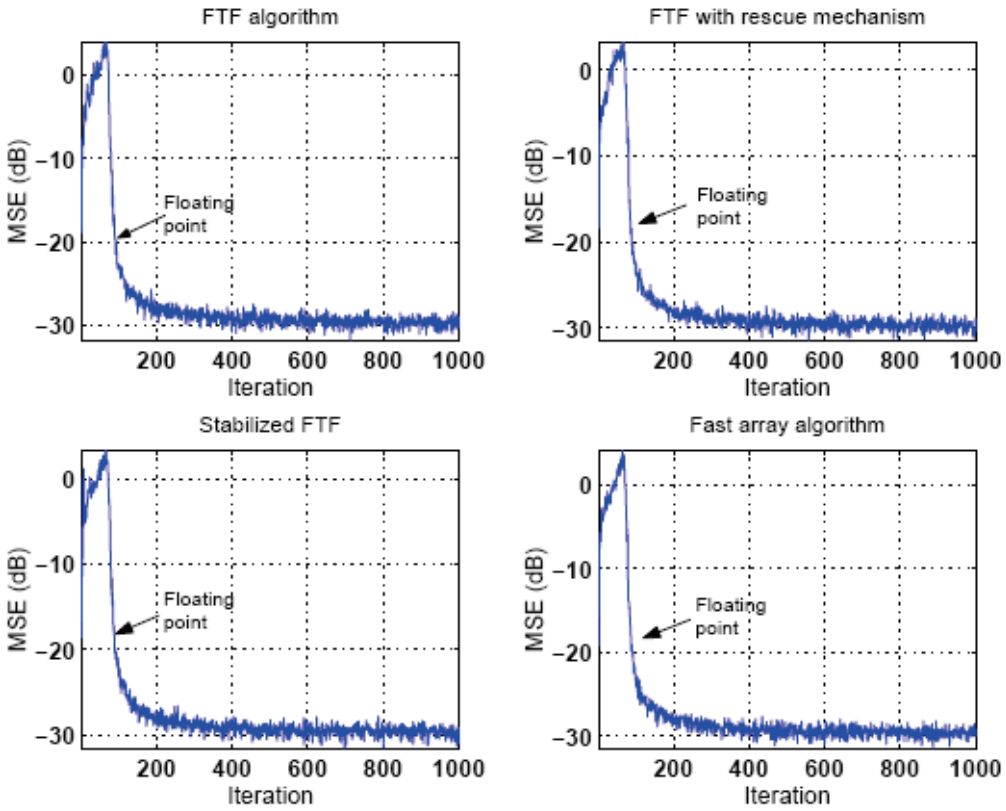


Figure IX.1. Ensemble-average learning curves for four fast fixed-order filter implementations in floating-point precision.

COMPUTER PROJECT

2. partB.m This program generates ensemble-average learning curves for four fast fixed-order filters using quantized implementations with 36 bits for both signals and coefficients. A typical output is shown in Fig. 2. It is seen that, even at this number of bits, the standard FTF implementation becomes unstable. When implemented with the rescue mechanism (39.10), the rescue procedure is called upon on several occasions but it still fails to recover filter performance. The stabilized FTF implementation also fails in this experiment. Only the fast array implementation is able to maintain the original filter performance in this case.

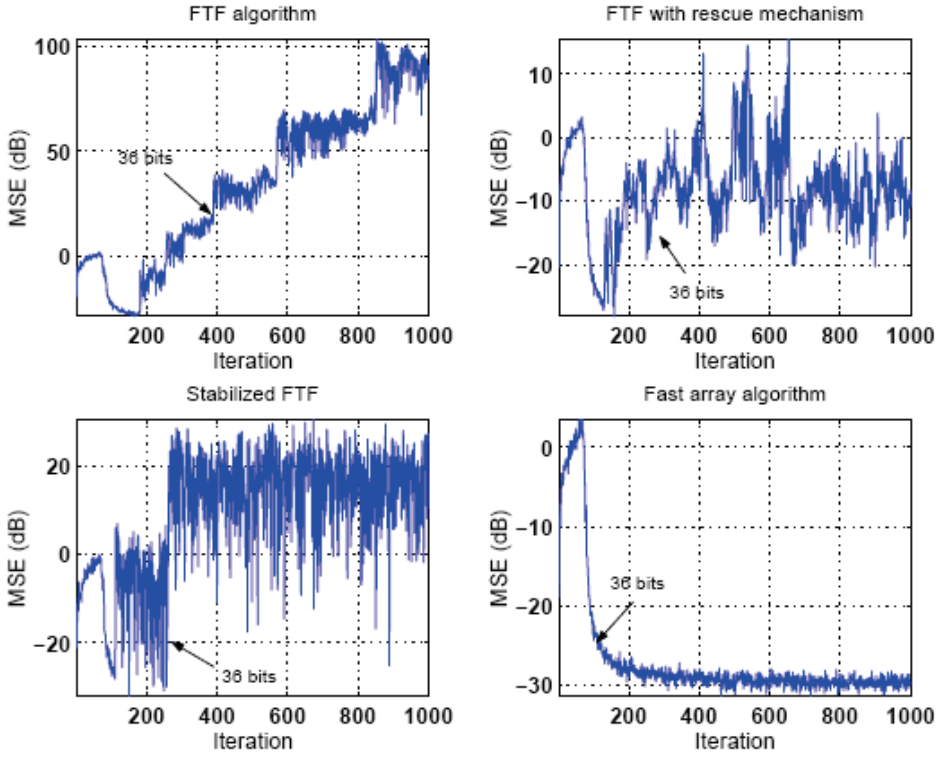


Figure IX.2. Ensemble-average learning curves for four fast fixed-order filter implementations averaged over 100 experiments in a quantized environment with 36 bits.