Chapter 9
QR-Decomposition-Based RLS Filters

9.1 Introduction

The application of QR decomposition [1] to triangularize the input data matrix results in an alternative method for the implementation of the recursive least-squares (RLS) method previously discussed. The main advantages brought about by the recursive least-squares algorithm based on QR decomposition are its possible implementation in systolic arrays [2–4] and its improved numerical behavior when quantization effects are taken into account [5].

The earlier proposed RLS algorithms based on the QR decomposition [2, 3] focused on the triangularization of the information matrix in order to avoid the use of matrix inversion. However, their computational requirement was of $O[N^2$ multiplications per output sample. Later, fast versions of the QR-RLS algorithms were proposed with a reduced computational complexity of $O[N]$ [4–11].

In this chapter, the QR-RLS algorithms based on Givens rotations are presented together with some stability considerations. Two families of fast algorithms are also discussed [4–11], and one fast algorithm is presented in detail. These fast algorithms are related to the tapped delay line FIR filter realization of the adaptive filter.

9.2 Triangularization Using QR-Decomposition

The RLS algorithm provides in a recursive way the coefficients of the adaptive filter which lead to the minimization of the following cost function

$$\xi^d(k) = \sum_{i=0}^{k} \lambda^{k-i} \varepsilon^2(i) = \sum_{i=0}^{k} \lambda^{k-i} [d(i) - \mathbf{x}^T(i)\mathbf{w}(k)]^2 \tag{9.1}$$

where

$$\mathbf{x}(k) = [x(k)\ x(k-1) \ldots x(k-N)]^T$$

P.S.R. Diniz, *Adaptive Filtering: Algorithms and Practical Implementation*,
DOI 10.1007/978-1-4614-4106-9_9,

is the input signal vector,

$$\mathbf{w}(k) = [w_0(k)\ w_1(k) \ldots w_N(k)]^T$$

is the coefficient vector at instant k, $\varepsilon(i)$ is the a posteriori error at instant i, and λ is the forgetting factor.

The same problem can be rewritten as a function of increasing dimension matrices and vectors which contain all the weighted signal information available so far to the adaptive filter. These matrices are redefined here for convenience:

$$\begin{aligned}\underline{\mathbf{X}}^T(k) &= \mathbf{X}(k) \\ &= \begin{bmatrix} x(k) & \lambda^{1/2}x(k-1) & \cdots & \lambda^{(k-1)/2}x(1) & \lambda^{k/2}x(0) \\ x(k-1) & \lambda^{1/2}x(k-2) & \cdots & \lambda^{(k-1)/2}x(0) & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x(k-N) & \lambda^{1/2}x(k-N-1) & \cdots & 0 & 0 \end{bmatrix} \\ &= [\mathbf{x}(k)\ \lambda^{1/2}\mathbf{x}(k-1) \ldots \lambda^{k/2}\mathbf{x}(0)] \end{aligned} \tag{9.2}$$

$$\mathbf{y}(k) = \underline{\mathbf{X}}(k)\mathbf{w}(k) = \begin{bmatrix} y(k) \\ \lambda^{1/2}y(k-1) \\ \vdots \\ \lambda^{k/2}y(0) \end{bmatrix} \tag{9.3}$$

$$\mathbf{d}(k) = \begin{bmatrix} d(k) \\ \lambda^{1/2}d(k-1) \\ \vdots \\ \lambda^{k/2}d(0) \end{bmatrix} \tag{9.4}$$

$$\boldsymbol{\varepsilon}(k) = \begin{bmatrix} \varepsilon(k) \\ \lambda^{1/2}\varepsilon(k-1) \\ \vdots \\ \lambda^{k/2}\varepsilon(0) \end{bmatrix} = \mathbf{d}(k) - \mathbf{y}(k) \tag{9.5}$$

The objective function of (9.1) can now be rewritten as

$$\xi^d(k) = \boldsymbol{\varepsilon}^T(k)\boldsymbol{\varepsilon}(k) \tag{9.6}$$

As shown in Chap. 5, (5.15), the optimal solution to the least-squares problem at a given instant of time k can be found by solving the following equation

$$\underline{\mathbf{X}}^T(k)\underline{\mathbf{X}}(k)\mathbf{w}(k) = \underline{\mathbf{X}}^T(k)\mathbf{d}(k) \tag{9.7}$$

However, solving this equation by using the conventional RLS algorithm can be a problem when the matrix $\mathbf{R}_D(k) = \underline{\mathbf{X}}^T(k)\underline{\mathbf{X}}(k)$ and its correspondent inverse estimate become ill-conditioned due to loss of persistence of excitation of the input signal or to quantization effects.

The QR decomposition approach avoids inaccurate solutions to the RLS problem and allows easy monitoring of the positive definiteness of a transformed information matrix in ill-conditioned situations.

9.2.1 Initialization Process

During the initialization period, i.e., from $k = 0$ to $k = N$, the solution of (9.7) can be found exactly without using any matrix inversion. From (9.7), it can be found that for $k = 0$ and $x(0) \neq 0$

$$w_0(0) = \frac{d(0)}{x(0)} \tag{9.8}$$

for $k = 1$

$$\begin{aligned} w_0(1) &= \frac{d(0)}{x(0)} \\ w_1(1) &= \frac{-x(1)w_0(1) + d(1)}{x(0)} \end{aligned} \tag{9.9}$$

for $k = 2$

$$\begin{aligned} w_0(2) &= \frac{d(0)}{x(0)} \\ w_1(2) &= \frac{-x(1)w_0(2) + d(1)}{x(0)} \\ w_2(2) &= \frac{-x(2)w_0(2) - x(1)w_1(2) + d(2)}{x(0)} \end{aligned} \tag{9.10}$$

at the instant k, we can show by induction that

$$w_i(k) = \frac{-\sum_{j=1}^{i} x(j)w_{i-j}(k) + d(i)}{x(0)} \tag{9.11}$$

The above equation represents the so-called back-substitution algorithm.

9.2.2 Input Data Matrix Triangularization

After the instant $k = N$, the above (9.11) is no longer valid and the inversion of $\mathbf{R}_D(k)$ or the calculation of $\mathbf{S}_D(k)$ is required to find the optimal solution for the coefficients $\mathbf{w}(k)$. This is exactly what makes the conventional RLS algorithm more sensitive to quantization effects and input signal conditioning. The matrix $\underline{\mathbf{X}}(k)$ at instant $k = N + 1$ is given by

$$\begin{aligned}\underline{\mathbf{X}}(N+1) &= \begin{bmatrix} x(N+1) & x(N) & \cdots & x(1) \\ \lambda^{1/2}x(N) & \lambda^{1/2}x(N-1) & \cdots & \lambda^{1/2}x(0) \\ \lambda x(N-1) & \lambda x(N-2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \lambda^{\frac{N+1}{2}}x(0) & 0 & \cdots & 0 \end{bmatrix} \\ &= \begin{bmatrix} x(N+1)\, x(N) \cdots x(1) \\ \lambda^{1/2}\underline{\mathbf{X}}(N) \end{bmatrix} = \begin{bmatrix} \mathbf{x}^T(N+1) \\ \lambda^{1/2}\underline{\mathbf{X}}(N) \end{bmatrix} \end{aligned} \tag{9.12}$$

As it is noted, the matrix $\underline{\mathbf{X}}(k)$ is no longer upper triangular, and, therefore, the back-substitution algorithm cannot be employed to find the tap-weight coefficients.

The matrix $\underline{\mathbf{X}}(N+1)$ can be triangularized through an orthogonal triangularization approach such as Givens rotations, Householder transformation, or Gram–Schmidt orthogonalization [1]. Since here the interest is to iteratively apply the triangularization procedure to each new data vector added to $\underline{\mathbf{X}}(k)$, the Givens rotation seems to be the most appropriate approach.

In the Givens rotation approach, each element of the first line of (9.12) can be eliminated by premultiplying the matrix $\underline{\mathbf{X}}(N+1)$ by a series of Givens rotation matrices given by

$$\begin{aligned}\tilde{\mathbf{Q}}(N+1) &= \mathbf{Q}'_N(N+1) \cdot \mathbf{Q}'_{N-1}(N+1) \cdots \mathbf{Q}'_0(N+1) \\ &= \begin{bmatrix} \cos\theta_N(N+1) & \cdots & 0 & \cdots & -\sin\theta_N(N+1) \\ \vdots & & & & \vdots \\ 0 & & \mathbf{I}_N & & 0 \\ \vdots & & & & \vdots \\ \sin\theta_N(N+1) & \cdots & 0 & \cdots & \cos\theta_N(N+1) \end{bmatrix} \\ &\quad \cdot \begin{bmatrix} \cos\theta_{N-1}(N+1) & \cdots & 0 & \cdots & -\sin\theta_{N-1}(N+1) & 0 \\ \vdots & & & & \vdots & \vdots \\ 0 & & \mathbf{I}_{N-1} & & 0 & 0 \\ \vdots & & & & \vdots & \vdots \\ \sin\theta_{N-1}(N+1) & \cdots & 0 & \cdots & \cos\theta_{N-1}(N+1) & 0 \\ 0 & & \cdots 0 & \cdots & 0 & 1 \end{bmatrix} \end{aligned}$$

$$\cdots \begin{bmatrix} \cos\theta_0(N+1) & -\sin\theta_0(N+1) & \cdots 0 \cdots & 0 \\ \sin\theta_0(N+1) & \cos\theta_0(N+1) & \cdots 0 \cdots & 0 \\ \vdots & \vdots & & \\ 0 & 0 & \mathbf{I}_N & \\ \vdots & \vdots & & \\ 0 & 0 & & \end{bmatrix} \tag{9.13}$$

where $\mathbf{I}_i$ is an i by i identity matrix. The rotation angles θ_i are chosen such that each entry of the first row of the resulting matrix is zero. Consider first the matrix product $\mathbf{Q}_0'(N+1)\underline{\mathbf{X}}(N+1)$. If:

$$\cos\theta_0(N+1)x(1) - \sin\theta_0(N+1)\lambda^{1/2}x(0) = 0 \tag{9.14}$$

the element in the position $(1, N+1)$ of the resulting matrix product will be zero. If it is further considered that $\cos^2\theta_0(N+1) + \sin^2\theta_0(N+1) = 1$, it can be easily deduced that

$$\cos\theta_0(N+1) = \frac{\lambda^{1/2}x(0)}{\sqrt{\lambda x^2(0) + x^2(1)}} \tag{9.15}$$

$$\sin\theta_0(N+1) = \frac{x(1)}{\sqrt{\lambda x^2(0) + x^2(1)}} \tag{9.16}$$

Next, $\mathbf{Q}_1'(N+1)$ premultiplies $\mathbf{Q}_0'(N+1)\underline{\mathbf{X}}(N+1)$ with the objective of generating a zero element at the position $(1, N)$ in the resulting product matrix. Note that the present matrix product does not remove the zero of the element $(1, N+1)$. The required rotation angle can be calculated by first noting that the elements $(1, N)$ and $(3, N)$ of $\mathbf{Q}_0'(N+1)\underline{\mathbf{X}}(N+1)$ are, respectively

$$a = \cos\theta_0(N+1)x(2) - \lambda^{1/2}x(1)\sin\theta_0(N+1) \tag{9.17}$$

$$b = \lambda x(0) \tag{9.18}$$

From these expressions we can compute the elements required in the following rotation, which are given by

$$\cos\theta_1(N+1) = \frac{b}{\sqrt{a^2 + b^2}} \tag{9.19}$$

$$\sin\theta_1(N+1) = \frac{a}{\sqrt{a^2 + b^2}} \tag{9.20}$$

In this manner, after the last Givens rotation the input signal information matrix will be transformed in a matrix with null first row

$$\tilde{\mathbf{Q}}(N+1)\underline{\mathbf{X}}(N+1) = \begin{bmatrix} 0\ 0 \cdots & 0 \\ \mathbf{U}(N+1) & \end{bmatrix} \tag{9.21}$$

where $\mathbf{U}(N+1)$ is an upper triangular matrix.

In the next iteration, the input signal matrix $\underline{\mathbf{X}}(N+2)$ receives a new row that should be replaced by a zero vector through a QR decomposition. In this step, the matrices involved are the following

$$\underline{\mathbf{X}}(N+2) = \begin{bmatrix} x(N+2)\, x(N+1) \cdots x(2) \\ \lambda^{1/2}\underline{\mathbf{X}}(N+1) \end{bmatrix} \tag{9.22}$$

and

$$\begin{bmatrix} 1\, 0 \cdots \cdots \\ 0 \\ \vdots \quad \tilde{\mathbf{Q}}(N+1) \\ \vdots \end{bmatrix} \underline{\mathbf{X}}(N+2) = \begin{bmatrix} x(N+2) & x(N+1) & \cdots x(2) \\ 0 & 0 & \cdots \ 0 \\ \multicolumn{3}{c}{\lambda^{1/2}\mathbf{U}(N+1)} \end{bmatrix} \tag{9.23}$$

In order to eliminate the new input vector through rotations with the corresponding rows of the triangular matrix $\lambda^{1/2}\mathbf{U}(N+1)$, we apply the QR decomposition to (9.23) as follows:

$$\tilde{\mathbf{Q}}(N+2) \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{Q}}(N+1) \end{bmatrix} \underline{\mathbf{X}}(N+2) = \begin{bmatrix} 0\, 0 \cdots & 0 \\ 0\, 0 \cdots & 0 \\ \mathbf{U}(N+2) & \end{bmatrix} \tag{9.24}$$

where again $\mathbf{U}(N+2)$ is an upper triangular matrix and $\tilde{\mathbf{Q}}(N+2)$ is given by

$$\begin{aligned} \tilde{\mathbf{Q}}(N+2) &= \mathbf{Q}'_N(N+2)\mathbf{Q}'_{N-1}(N+2)\cdots\mathbf{Q}'_0(N+2) \\ &= \begin{bmatrix} \cos\theta_N(N+2) \cdots & 0 & \cdots -\sin\theta_N(N+2) \\ \vdots & & \vdots \\ 0 & \mathbf{I}_{N+1} & 0 \\ \vdots & & \vdots \\ \sin\theta_N(N+2) \cdots & 0 & \cdots \cos\theta_N(N+2) \end{bmatrix} \\ &\quad \cdot \begin{bmatrix} \cos\theta_{N-1}(N+2) \cdots 0 \cdots -\sin\theta_{N-1}(N+2) & 0 \\ \vdots & \vdots \\ 0 & \mathbf{I}_N & 0 \\ \vdots & \vdots \\ \sin\theta_{N-1}(N+2) & \cos\theta_{N-1}(N+2) & 0 \\ 0 \quad \cdots 0 \cdots & 0 & 1 \end{bmatrix} \end{aligned}$$

$$\cdots \begin{bmatrix} \cos\theta_0(N+2) & 0 & -\sin\theta_0(N+2) & \cdots 0 \\ 0 & 1 & 0 & \cdots 0 \\ \sin\theta_0(N+2) & 0 & \cos\theta_0(N+2) & \cdots 0 \\ \vdots & \vdots & \vdots & \\ \vdots & \vdots & \vdots & \mathbf{I}_N \\ 0 & 0 & 0 & \end{bmatrix} \tag{9.25}$$

The above procedure should be repeated for each new incoming input signal vector as follows:

$$\begin{aligned} \mathbf{Q}(k)\underline{\mathbf{X}}(k) &= \tilde{\mathbf{Q}}(k) \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{Q}}(k-1) \end{bmatrix} \begin{bmatrix} \mathbf{I}_2 & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{Q}}(k-2) \end{bmatrix} \\ &\cdots \begin{bmatrix} \mathbf{I}_{k-N} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{Q}}(k-N) \end{bmatrix} \underline{\mathbf{X}}(k) = \underbrace{\begin{bmatrix} \mathbf{0} \\ \\ \mathbf{U}(k) \end{bmatrix}}_{N+1} \begin{matrix} \} k-N \\ \\ \} N+1 \end{matrix} \end{aligned} \tag{9.26}$$

where $\mathbf{Q}(k)$ is a $(k+1)$ by $(k+1)$ matrix which represents the overall triangularization matrix via elementary Givens rotations matrices $\mathbf{Q}_i'(m)$ for all $m \leq k$ and $0 \leq i \leq N$.

Since each Givens rotation matrix is orthogonal, then it can easily be proved that $\mathbf{Q}(k)$ is also orthogonal (actually orthonormal), i.e.,

$$\mathbf{Q}(k)\mathbf{Q}^T(k) = \mathbf{I}_{k+1} \tag{9.27}$$

Also, from (9.27), it is straightforward to note that

$$\mathbf{Q}(k) = \tilde{\mathbf{Q}}(k) \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}(k-1) \end{bmatrix} \tag{9.28}$$

where $\tilde{\mathbf{Q}}(k)$ is responsible for zeroing the latest input vector $\mathbf{x}^T(k)$ in the first row of $\underline{\mathbf{X}}(k)$. The matrix $\tilde{\mathbf{Q}}(k)$ is given by

$$\tilde{\mathbf{Q}}(k) = \begin{bmatrix} \cos\theta_N(k) & \cdots & 0 & \cdots & -\sin\theta_N(k) \\ \vdots & & & & \vdots \\ 0 & & \mathbf{I}_{k-1} & & 0 \\ \vdots & & & & \vdots \\ \sin\theta_N(k) & \cdots & 0 & \cdots & \cos\theta_N(k) \end{bmatrix}$$

$$
\cdot \begin{bmatrix}
\cos\theta_{N-1}(k) & \cdots & 0 & \cdots & -\sin\theta_{N-1}(k) & 0 \\
\vdots & & & & \vdots & \vdots \\
0 & & \mathbf{I}_{k-2} & & 0 & 0 \\
\vdots & & & & \vdots & \vdots \\
\sin\theta_{N-1}(k) & \cdots & 0 & \cdots & \cos\theta_{N-1}(k) & 0 \\
0 & \cdots & 0 & \cdots & 0 & 1
\end{bmatrix}
$$

$$
\cdots \begin{bmatrix}
\cos\theta_0(k) & \cdots & 0 & \cdots & -\sin\theta_0(k) & 0 \\
\vdots & & & & \vdots & \vdots \\
0 & & \mathbf{I}_{k-N-1} & & 0 & 0 \\
\vdots & & & & \vdots & \vdots \\
\sin\theta_0(k) & \cdots & 0 & \cdots & \cos\theta_0(k) & 0 \\
 & & \mathbf{0} & & & \mathbf{I}_N
\end{bmatrix}
$$

$$
= \begin{bmatrix}
\prod_{i=0}^{N} \cos\theta_i(k) & \cdots & 0 & \cdots & -\prod_{i=1}^{N} \cos\theta_i(k)\sin\theta_0(k) \\
\vdots & & & & \vdots \\
0 & & \mathbf{I}_{k-N-1} & & 0 \\
\vdots & & & & \vdots \\
\sin\theta_0(k) & & & & \cos\theta_0(k) \\
\vdots & & \vdots & & \vdots \\
\prod_{i=0}^{j-1} \cos\theta_i(k)\sin\theta_j(k) & \cdots & 0 & \cdots & \vdots \\
\vdots & & \vdots & & -\sin\theta_N(k)\prod_{i=1}^{N-1} \cos\theta_i(k)\sin\theta_0(k)
\end{bmatrix}
$$

$$
\begin{bmatrix}
\cdots -\prod_{i=j+1}^{N} \cos\theta_i(k)\sin\theta_j(k) & \cdots & -\sin\theta_N(k) \\
 & & 0 \\
\ddots & & \\
 & \ddots & \mathbf{0} \\
 & & \cos\theta_{N-1}(k) \\
 & & \cos\theta_N(k)
\end{bmatrix} \tag{9.29}
$$

Note that the matrix $\tilde{\mathbf{Q}}(k)$ has the following general form

$$\tilde{\mathbf{Q}}(k) = \begin{bmatrix} * & 0 \cdots & 0 & \cdots 0 & \overbrace{* \cdots *}^{N+1} \\ 0 & & & & \\ \vdots & & \mathbf{I}_{k-N-1} & \mathbf{0} & \\ * & & & * & \\ \vdots & & \mathbf{0} & \ddots & \\ * & & & * & * \end{bmatrix} \Bigg\} N+1 \tag{9.30}$$

where $*$ represents a nonzero element. This structure of $\tilde{\mathbf{Q}}(k)$ is useful for developing some fast QR-RLS algorithms.

Returning to (9.27), we can conclude that

$$\mathbf{Q}(k)\underline{\mathbf{X}}(k) = \tilde{\mathbf{Q}}(k) \begin{bmatrix} x(k) & x(k-1) & \cdots & x(k-N) \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \\ \multicolumn{4}{c}{\lambda^{1/2}\mathbf{U}(k-1)} \end{bmatrix} \tag{9.31}$$

The first Givens rotation angle required to replace $x(k-N)$ by a zero is $\theta_0(k)$ such that

$$\cos\theta_0(k)x(k-N) - \sin\theta_0(k)\lambda^{1/2}u_{1,N+1}(k-1) = 0 \tag{9.32}$$

where $u_{1,N+1}(k-1)$ is the element $(1, N+1)$ of $\mathbf{U}(k-1)$. Then, it follows that

$$\cos\theta_0(k) = \frac{\lambda^{1/2}u_{1,N+1}(k-1)}{u_{1,N+1}(k)} \tag{9.33}$$

$$\sin\theta_0(k) = \frac{x(k-N)}{u_{1,N+1}(k)} \tag{9.34}$$

where

$$u^2_{1,N+1}(k) = x^2(k-N) + \lambda u^2_{1,N+1}(k-1) \tag{9.35}$$

From (9.35), it is worth noting that the $(1, N+1)$ element of $\mathbf{U}(k)$ is the square root of the exponentially weighted input signal energy, i.e.,

$$u^2_{1,N+1}(k) = \sum_{i=0}^{k-N} \lambda^i x^2(k-N-i) \tag{9.36}$$

In the triangularization process, all the submatrices multiplying each column of $\underline{\mathbf{X}}(k)$ are orthogonal matrices and as a consequence the norm of each column in $\underline{\mathbf{X}}(k)$ and $\mathbf{Q}(k)\underline{\mathbf{X}}(k)$ should be the same. This confirms that (9.36) is valid. Also, it can be shown that

$$\sum_{i=1}^{k+1} \underline{x}_{i,j}^2(k) = \sum_{i=1}^{N+2-j} u_{i,j}^2(k) = \sum_{i=1}^{k+1} \lambda^{i-1} x^2(k+2-i-j) \tag{9.37}$$

for $j = 1, 2, \ldots, N+1$.

Now consider that the intermediate calculations of (9.31) are performed as follows:

$$\tilde{\mathbf{Q}}(k) \begin{bmatrix} \mathbf{x}^T(k) \\ \mathbf{0} \\ \lambda^{1/2}\mathbf{U}(k-1) \end{bmatrix} = \mathbf{Q}'_N(k)\,\mathbf{Q}'_{N-1}(k) \cdots \mathbf{Q}'_i(k) \begin{bmatrix} \mathbf{x}'_i(k) \\ \mathbf{0} \\ \mathbf{U}'_i(k) \end{bmatrix} \tag{9.38}$$

where $\mathbf{x}'_i(k) = [x'_i(k)\, x'_i(k-1) \ldots x'_i(k-N+i)\, 0 \ldots 0]$ and $\mathbf{U}'_i(k)$ is an intermediate upper triangular matrix. Note that $\mathbf{x}'_0(k) = \mathbf{x}^T(k)$, $\mathbf{U}'_0(k) = \lambda^{1/2}\mathbf{U}(k-1)$, and $\mathbf{U}'_{N+1}(k) = \mathbf{U}(k)$. In practice, the multiplication by the zero elements in (9.38) should be avoided. We start by removing the increasing $\mathbf{I}_{k-N-1}$ section of $\tilde{\mathbf{Q}}(k)$ (see (9.30)), thereby generating a matrix with reduced dimension denoted by $\mathbf{Q}_\theta(k)$. The resulting equation is

$$\begin{aligned} \mathbf{Q}_\theta(k) \begin{bmatrix} \mathbf{x}^T(k) \\ \lambda^{1/2}\mathbf{U}(k-1) \end{bmatrix} &= \mathbf{Q}'_{\theta_N}(k)\mathbf{Q}'_{\theta_{N-1}}(k) \cdots \mathbf{Q}'_{\theta_i}(k) \begin{bmatrix} \mathbf{x}'_i(k) \\ \mathbf{U}'_i(k) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{0} \\ \mathbf{U}(k) \end{bmatrix} \end{aligned} \tag{9.39}$$

where $\mathbf{Q}'_{\theta_i}(k)$ is derived from $\mathbf{Q}'_i(k)$ by removing the $\mathbf{I}_{k-N-1}$ section of $\mathbf{Q}'_i(k)$ along with the corresponding rows and columns, resulting in the following form

$$\mathbf{Q}'_{\theta_i}(k) = \begin{bmatrix} \cos\theta_i(k) & \cdots & 0 & \cdots & -\sin\theta_i(k) & \cdots & 0 \\ \vdots & & & & \vdots & & \vdots \\ 0 & & \mathbf{I}_i & & 0 & \cdots & 0 \\ \vdots & & & & \vdots & & \vdots \\ \sin\theta_i(k) & \cdots & 0 & \cdots & \cos\theta_i(k) & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \mathbf{I}_{N-i} \\ 0 & \cdots & 0 & \cdots & 0 & & \end{bmatrix} \tag{9.40}$$

The Givens rotation elements are calculated by

$$\cos\theta_i(k) = \frac{[\mathbf{U}_i'(k)]_{i+1,N+1-i}}{c_i} \tag{9.41}$$

$$\sin\theta_i(k) = \frac{x_i'(k-N+i)}{c_i} \tag{9.42}$$

where $c_i = \sqrt{[\mathbf{U}_i'(k)]^2_{i+1,N+1-i} + x_i'^2(k-N+i)}$ and $[\cdot]_{i,j}$ is the (i,j) element of the matrix.

9.2.3 QR-Decomposition RLS Algorithm

The triangularization procedure above discussed can be applied to generate the QR-RLS algorithm that avoids the calculation of the $\mathbf{S}_D(k)$ matrix of the conventional RLS algorithm. The weighted a posteriori error vector can be written as a function of the input data matrix, that is

$$\boldsymbol{\varepsilon}(k) = \begin{bmatrix} \varepsilon(k) \\ \lambda^{1/2}\varepsilon(k-1) \\ \vdots \\ \lambda^{k/2}\varepsilon(0) \end{bmatrix} = \begin{bmatrix} d(k) \\ \lambda^{1/2}d(k-1) \\ \vdots \\ \lambda^{k/2}d(0) \end{bmatrix} - \underline{\mathbf{X}}(k)\mathbf{w}(k) \tag{9.43}$$

By premultiplying the above equation by $\mathbf{Q}(k)$, it follows that

$$\begin{aligned} \boldsymbol{\varepsilon}_q(k) = \mathbf{Q}(k)\boldsymbol{\varepsilon}(k) &= \mathbf{Q}(k)\mathbf{d}(k) - \mathbf{Q}(k)\underline{\mathbf{X}}(k)\mathbf{w}(k) \\ &= \mathbf{d}_q(k) - \begin{bmatrix} \mathbf{0} \\ \mathbf{U}(k) \end{bmatrix}\mathbf{w}(k) \end{aligned} \tag{9.44}$$

where

$$\boldsymbol{\varepsilon}_q(k) = \begin{bmatrix} \varepsilon_{q_1}(k) \\ \varepsilon_{q_2}(k) \\ \vdots \\ \varepsilon_{q_{k+1}}(k) \end{bmatrix}$$

and

$$\mathbf{d}_q(k) = \begin{bmatrix} d_{q_1}(k) \\ d_{q_2}(k) \\ \vdots \\ d_{q_{k+1}}(k) \end{bmatrix}$$

Since $\mathbf{Q}(k)$ is an orthogonal matrix, (9.6) is equivalent to

$$\xi^d(k) = \boldsymbol{\varepsilon}_q^T(k)\boldsymbol{\varepsilon}_q(k) \tag{9.45}$$

because

$$\boldsymbol{\varepsilon}_q^T(k)\boldsymbol{\varepsilon}_q(k) = \boldsymbol{\varepsilon}^T(k)\mathbf{Q}^T(k)\mathbf{Q}(k)\boldsymbol{\varepsilon}(k) = \boldsymbol{\varepsilon}^T(k)\boldsymbol{\varepsilon}(k)$$

The weighted-square error can be minimized in (9.45) by calculating $\mathbf{w}(k)$ such that $\varepsilon_{q_{k-N+1}}(k)$ to $\varepsilon_{q_{k+1}}(k)$ are made zero using a back-substitution algorithm such as

$$w_i(k) = \frac{-\sum_{j=1}^{i} u_{N+1-i,i-j+1}(k)w_{i-j}(k) + d_{q\,k+1-i}(k)}{u_{N+1-i,i+1}(k)} \tag{9.46}$$

for $i = 0, 1, \ldots, N$, where $\sum_{j=i}^{i-1}[\cdot] = 0$. With this choice for $\mathbf{w}(k)$, the minimum weighted-square error at instant k is given by

$$\xi_{\min}^d(k) = \sum_{i=1}^{k-N} \varepsilon_{q_i}^2(k) \tag{9.47}$$

An important relation can be deduced by rewriting (9.44) as

$$\mathbf{d}_q(k) = \begin{bmatrix} \mathbf{d}_{q_1}(k) \\ ---- \\ \mathbf{d}_{q_2}(k) \end{bmatrix} = \begin{bmatrix} d_{q_1}(k) \\ \vdots \\ d_{q_{k-N}}(k) \\ ------ \\ d_{q_{k-N+1}}(k) \\ \vdots \\ d_{q_{k+1}}(k) \end{bmatrix}$$

$$= \begin{bmatrix} \varepsilon_{q_1}(k) \\ \vdots \\ \varepsilon_{q_{k-N}}(k) \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{U}(k) \end{bmatrix} \mathbf{w}(k) \tag{9.48}$$

where $\mathbf{w}(k)$ is the optimum coefficient vector at instant k. By examining (9.31) and (9.44), the right-most side of (9.48) can then be expressed as

$$\begin{bmatrix} \boldsymbol{\varepsilon}_{q_1}(k) \\ \mathbf{d}_{q_2}(k) \end{bmatrix} = \begin{bmatrix} \varepsilon_{q_1}(k) \\ \vdots \\ \varepsilon_{q_{k-N}}(k) \\ \mathbf{d}_{q_2}(k) \end{bmatrix} = \tilde{\mathbf{Q}}(k) \begin{bmatrix} d(k) \\ \lambda^{1/2} \begin{bmatrix} \varepsilon_{q_1}(k-1) \\ \vdots \\ \varepsilon_{q_{k-N-1}}(k-1) \\ \mathbf{d}_{q_2}(k-1) \end{bmatrix} \end{bmatrix} \tag{9.49}$$

Using similar arguments around (9.38)–(9.40), and starting from (9.49), the transformed weighted-error vector can be updated as described below:

$$\tilde{\mathbf{Q}}(k) \begin{bmatrix} d(k) \\ \lambda^{1/2} \begin{bmatrix} \boldsymbol{\varepsilon}_{q_1}(k-1) \\ \mathbf{d}_{q_2}(k-1) \end{bmatrix} \end{bmatrix} = \mathbf{Q}'_N(k)\mathbf{Q}'_{N-1}(k)\cdots\mathbf{Q}'_i(k) \begin{bmatrix} d'_i(k) \\ \boldsymbol{\varepsilon}'_{q_i}(k) \\ \mathbf{d}'_{q_{2i}}(k) \end{bmatrix} \tag{9.50}$$

where $d'_i(k)$, $\boldsymbol{\varepsilon}'_{q_i}(k)$, and $\mathbf{d}'_{q_{2i}}(k)$ are intermediate quantities generated during the rotations. Note that $\boldsymbol{\varepsilon}'_{q_{N+1}}(k) = [\varepsilon_{q_2}(k)\ \varepsilon_{q_3}(k)\ldots\varepsilon_{q_{k-N}}(k)]^T$, $d'_{N+1}(k) = \varepsilon_{q_1}(k)$, and $\mathbf{d}'_{q_{2N+1}} = \mathbf{d}_{q_2}(k)$.

If we delete all the columns and rows of $\tilde{\mathbf{Q}}(k)$ whose elements are zeros and ones, i.e., the $\mathbf{I}_{k-N-1}$ section of $\tilde{\mathbf{Q}}(k)$ with the respective bands of zeros below, above, and on each side of it in (9.30), one would obtain matrix $\mathbf{Q}_\theta(k)$. In this case, the resulting equation corresponding to (9.49) is given by

$$\underline{\mathbf{d}}(k) = \begin{bmatrix} \varepsilon_{q_1}(k) \\ \mathbf{d}_{q_2}(k) \end{bmatrix} = \mathbf{Q}_\theta(k) \begin{bmatrix} d(k) \\ \lambda^{1/2}\mathbf{d}_{q_2}(k-1) \end{bmatrix} \tag{9.51}$$

Therefore, we eliminate the vector $\boldsymbol{\varepsilon}'_{q_{N+1}}(k)$ which is always increasing, such that in real-time implementation the updating is performed through

$$\begin{aligned} \underline{\mathbf{d}}(k) &= \mathbf{Q}_\theta(k) \begin{bmatrix} d(k) \\ \lambda^{1/2}\mathbf{d}_{q_2}(k-1) \end{bmatrix} \\ &= \mathbf{Q}'_{\theta_N}(k)\,\mathbf{Q}'_{\theta_{N-1}}(k)\,\cdots\,\mathbf{Q}'_{\theta_i}(k) \begin{bmatrix} d'_i(k) \\ \mathbf{d}'_{q_{2i}}(k) \end{bmatrix} \end{aligned} \tag{9.52}$$

Another important relation can be derived from (9.44) by premultiplying both sides by $\mathbf{Q}^T(k)$, transposing the result, and postmultiplying the result by the pinning vector

$$\boldsymbol{\varepsilon}_q^T(k)\mathbf{Q}(k) \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \boldsymbol{\varepsilon}^T(k) \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \varepsilon(k) \tag{9.53}$$

Then, from the definition of $\mathbf{Q}(k)$ in (9.28) and (9.29), the following relation is obtained

$$\begin{aligned}\varepsilon(k) &= \varepsilon_{q_1}(k)\prod_{i=0}^{N}\cos\theta_i(k)\\ &= \varepsilon_{q_1}(k)\gamma(k)\end{aligned} \tag{9.54}$$

This relation shows that the a posteriori output error can be computed without the explicit calculation of $\mathbf{w}(k)$. The only information needed is the Givens rotation cosines. In applications where only the a posteriori output error is of interest, the computationally intensive back-substitution algorithm of (9.46) to obtain $\mathbf{w}_i(k)$ can be avoided.

Now, all the mathematical background to develop the QR-RLS algorithm has been derived. After initialization, the Givens rotation elements are computed using (9.41) and (9.42). These rotations are then applied to the information matrix and the desired signal vector, respectively, as indicated in (9.39) and (9.52). The next step is to compute the error signal using (9.54). Finally, if the tap-weight coefficients are required we should calculate them using (9.46). Algorithm 9.1 summarizes the algorithm with all essential computations.

Example 9.1. In this example, we solve the system identification problem described in Sect. 3.6.2 by using the QR-RLS algorithm described in this section.

Solution. In the present example, we are mainly concerned in testing the algorithm implemented in finite precision, since the remaining characteristics (such as: misadjustment and convergence speed) should follow the same pattern of the conventional RLS algorithm. We considered the case where eigenvalue spread of the input signal correlation matrix is 20, with $\lambda = 0.99$. The presented results were obtained by averaging the outcomes of 200 independent runs. Table 9.1 summarizes the results, where it can be noticed that the MSE is comparable to the case of the conventional RLS algorithm (consult Table 5.2). On the other hand, the quantization error introduced by the calculations to obtain $\mathbf{w}(k)_Q$ is considerable. After leaving the algorithm running for a large number of iterations, we found no sign of divergence.

In the infinite-precision implementation, the misadjustment measured was 0.0429. As expected (consult Table 5.1) this result is close to the misadjustment obtained by the conventional RLS algorithm. □

9.3 Systolic Array Implementation

The systolic array implementation of a given algorithm consists of mapping the algorithm in a pipelined sequence of basic computation cells. These basic cells perform their task in parallel, such that in each clock period all the cells are activated. An in-depth treatment of systolic array implementation and parallelization

Algorithm 9.1 QR-RLS algorithm

$\mathbf{w}(-1) = [0\,0\ldots 0]^T, \quad w_0(0) = \frac{d(0)}{x(0)}$

For $k = 1$ to N (Initialization)

Do for $i = 1$ to k

$$w_i(k) = \frac{-\sum_{j=1}^{i} x(j) w_{i-j}(k) + d(i)}{x(0)} \tag{9.11}$$

End

End

$$\mathbf{U}'_0(N+1) = \lambda^{1/2}\underline{\mathbf{X}}(N) \tag{9.12}$$

$$\mathbf{d}'_{q_{20}}(N+1) = [\,\lambda^{1/2}d(N)\;\lambda d(N-1)\ldots\lambda^{(N+1)/2}d(0)]^T$$

For $k \geq N+1$

Do for each k

$\gamma'_{-1} = 1$

$d'_0(k) = d(k)$

$\mathbf{x}'_0(k) = \mathbf{x}^T(k)$

Do for $i = 0$ to N

$$c_i = \sqrt{[\mathbf{U}'_i(k)]^2_{i+1,N+1-i} + x'^2_i(k-N+i)}$$

$$\cos\theta_i = \frac{[\mathbf{U}'_i(k)]_{i+1,N+1-i}}{c_i} \tag{9.41}$$

$$\sin\theta_i = \frac{x'_i(k-N+i)}{c_i} \tag{9.42}$$

$$\begin{bmatrix} \mathbf{x}'_{i+1}(k) \\ \mathbf{U}'_{i+1}(k) \end{bmatrix} = \mathbf{Q}'_{\theta_i}(k) \begin{bmatrix} \mathbf{x}'_i(k) \\ \mathbf{U}'_i(k) \end{bmatrix} \tag{9.39}$$

$$\gamma'_i = \gamma'_{i-1}\cos\theta_i \tag{9.54}$$

$$\begin{bmatrix} d'_{i+1}(k) \\ \mathbf{d}'_{q_{2i+1}}(k) \end{bmatrix} = \mathbf{Q}'_{\theta_i}(k) \begin{bmatrix} d'_i(k) \\ \mathbf{d}'_{q_{2i}}(k) \end{bmatrix} \tag{9.51}$$

End

$\mathbf{d}'_{q_{20}}(k+1) = \lambda^{1/2}\mathbf{d}'_{q_{2N+1}}(k)$

$\mathbf{U}'_0(k+1) = \lambda^{1/2}\mathbf{U}'_{N+1}(k)$

$\gamma(k) = \gamma'_N$

$\varepsilon(k) = d'_{N+1}(k)\gamma(k)$ (9.51)

If required compute

$$\underline{\mathbf{d}}(k) = \begin{bmatrix} d'_{N+1}(k) \\ \mathbf{d}'_{q_{2N+1}}(k) \end{bmatrix} \tag{9.51}$$

$w_0(k) = \frac{\underline{d}_{N+2}(k)}{u_{N+1,1}(k)}$

Do for $i = 1$ to N

$$w_i(k) = \frac{-\sum_{j=1}^{i} u_{N+1-i,i-j+1}(k) w_{i-j}(k) + \underline{d}_{N+2-i}(k)}{u_{N+1-i,i+1}(k)} \tag{9.46}$$

End

End

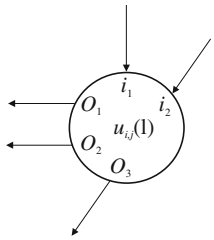
of algorithms is beyond the scope of this text. Our objective in this section is to demonstrate in a summarized form that the QR-RLS algorithm can be mapped in a systolic array. Further details regarding this subject can be found in references [2–4, 12, 13].

Table 9.1 Results of the finite-precision implementation of the QR-RLS algorithm

	$\xi(k)_Q$	$E[\|\|\Delta\mathbf{w}(k)_Q\|\|^2]$
No. of bits	Experiment	Experiment
16	$1.544\ 10^{-3}$	0.03473
12	$1.563\ 10^{-3}$	0.03254
10	$1.568\ 10^{-3}$	0.03254

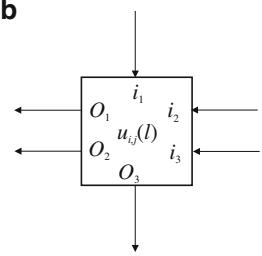
Fig. 9.1 Basic cells: (**a**) Angle processor, (**b**) Rotation processor

a



If i_1=0 then
$O_1 \leftarrow 1, O_2 \leftarrow 0, O_3 \leftarrow i_2$
$u_{i,j} = \lambda u_{i,j}$
Otherwise
$c \leftarrow \sqrt{\lambda^2 u_{i,j}^2(l) + i_1^2}$
$O_1 \leftarrow \cos\theta_{i-1} = \frac{\lambda u_{i,j}(l)}{c}$
$O_2 \leftarrow \sin\theta_{i-1} = \frac{i_1}{c}$
$O_3 \leftarrow i_2 O_1$
$u_{i,j}(l+1) \leftarrow c$
End

b



$O_1 \leftarrow i_2$
$O_2 \leftarrow i_3$
$O_3 \leftarrow i_1 i_2 - i_3 \lambda u_{i,j}(l)$
$u_{i,j}(l+1) \leftarrow i_1 i_3 + i_2 \lambda u_{i,j}(l)$

A Givens rotation requires two basic steps. The first step is the calculation of the sine and cosine which are the elements of the rotation matrix. The second step is the application of the rotation matrix to given data. Therefore, the basic computational elements required to perform the systolic array implementation of the QR-RLS algorithm introduced in the last section are the angle and the rotation processors shown in Fig. 9.1. The angle processor computes the cosine and sine, transferring

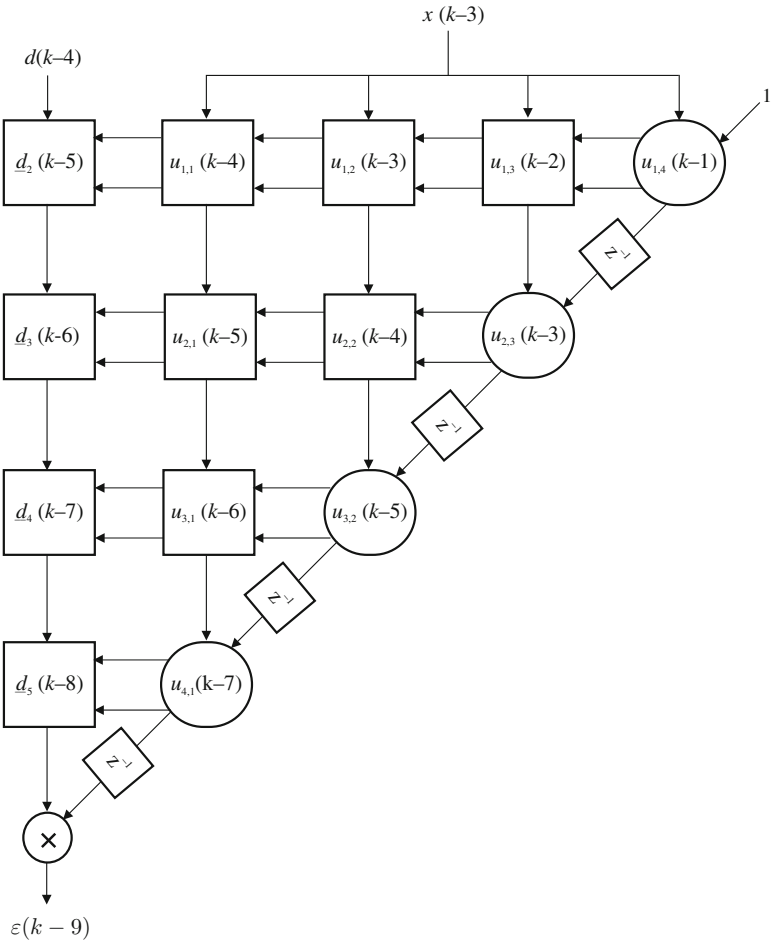


Fig. 9.2 QR-Decomposition systolic array for N=3

the results to outputs 1 and 2, respectively, whereas in output 3 the cell delivers a partial product of cosines meant to generate the error signal as in (9.54). The rotation processor performs the rotation between the data coming from input 1 with the internal element of the matrix $\mathbf{U}(l)$ and transfers the result to output 3. This processor also updates the elements of $\mathbf{U}(l)$ and transfers the cosine and sine values to the neighboring cell on the left.

Now, imagine that we have the upper triangular matrix $\mathbf{U}(k)$ arranged below the row consisting of the new information data vector as in (9.31), or equivalently as in (9.39). Following the same pattern, we can arrange the basic cells in order to compute the rotations of the QR-RLS algorithm as shown in Fig. 9.2, with the

input signal $x(k)$ entering the array serially. In this figure, do not consider for this moment the time indexes and the left-hand side column. The input data weighting is performed by the processors of the systolic array.

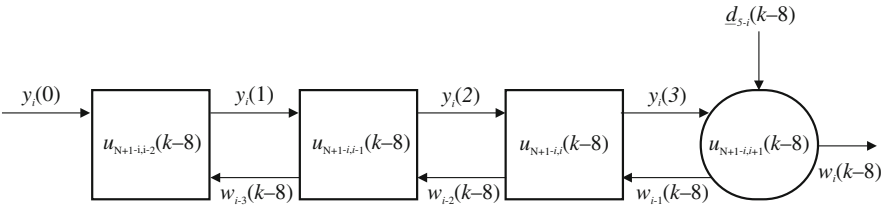
Basically, the computations corresponding to the triangularization of (9.31) are performed through the systolic array shown in Fig. 9.2, where at each instant of time an element of the matrix $\mathbf{U}(k)$ is stored in the basic processor as shown inside the building blocks. Note that these stored elements are skewed in time and are initialized with zero. The left-hand cells store the elements of the vector $\underline{\mathbf{d}}(k)$ defined in (9.51), which are also initialized with zero and updated in each clock cycle. The column on the left-hand side of the array performs the rotation and stores the rotated values of the desired signal vector which are essential to compute the error signal.

In order to allow the pipelining, the outputs of each cell are computed at the present clock period and made available to the neighboring cells in the following clock period. Note that the neighboring cells on the left and below a given cell are performing computations related to a previous iteration, whereas the cells on the right and above are performing the computations of one iteration in advance. This is the pipelining scheme of Fig. 9.2.

Each row of cells in the array implements a basic Givens rotation between one row of $\lambda\mathbf{U}(k-1)$ and a vector related to the new incoming data $\mathbf{x}(k)$. The top row of the systolic array performs the zeroing of the last element of the most recent incoming $\mathbf{x}(k)$ vector. The result of the rotation is then passed to the second row of the array. This second row performs the zeroing of the second-to-last element in the rotated input signal. The zeroing processing continues in the following rows by eliminating the remaining elements of the intermediate vectors $\mathbf{x}'_i(k)$, defined in (9.38), through Givens rotations. The angle processors compute the rotation angles that are passed to each row to perform the rotations.

More specifically, returning to (9.31), at the instant k, the element $x(k-N)$ of $\mathbf{x}(k)$ is eliminated by calculating the angle $\theta_0(k)$ in the upper angle processor. The same processor also performs the computation of $u_{1,N+1}(k)$ that will be stored and saved for later elimination of $x(k-N+1)$, which occurs during the triangularization of $\underline{\mathbf{X}}(k+1)$. In the same period of time, the neighboring rotation processor performs the computation of $u_{1,N}(k-1)$ using the angle $\theta_0(k-1)$ that was received from the angle processor in the beginning of the present clock period k. The modifications to the first row of the $\mathbf{U}(k)$ matrix and to the vector $\underline{\mathbf{d}}(k)$ related to the desired signal are performed in the first row of the array, due to the rotation responsible for the elimination of $x(k-N)$. Note that the effect of the angle $\theta_0(k)$ in the remaining elements of the first row of $\mathbf{U}(k)$ will be felt only in the following iterations, one element each time, starting from the right- to the left-hand side.

The second row of the systolic array is responsible for the rotation corresponding to $\theta_1(l)$ that eliminates the element $x'_1(l-N+1)$ of $\mathbf{x}'_1(l)$ defined in (9.38). The rotation $\theta_1(l)$ of course modifies the remaining nonzero elements of $\mathbf{x}'_1(l)$ generating $\mathbf{x}'_2(l)$, whose elements are calculated by the rotation processor and forwarded to the next row through output 3.



$w_i = 0$ for $i < 0$
Do for $i = 0, 1, \ldots, N$
$y_i(N - i) = 0$
Do for l= $N - i + 1, \ldots, N$
$y_i(l) = y_i(l-1) + u_{N+1-i,i-N+l}(k-8)w_{i-N+l-1}(k-8)$
End

$$w_i(k-8) = \frac{d_{5-i}(k-8) - y_i(3)}{u_{N+1-i,i+1}(k-8)}$$

End

Fig. 9.3 Systolic array and algorithm for the computation of $\mathbf{w}(k)$

Likewise, the $(i+1)$th row performs the rotation $\theta_i(l)$ that eliminates $x_i'(l - N + i)$ and also the rotation in the vector $\underline{\mathbf{d}}(l)$.

In the bottom part of the systolic array, the product of $\varepsilon_{q_1}(l)$ and $\gamma(l)$ is calculated at each clock instant, in order to generate a posteriori output error given by $\varepsilon(l)$. The output error obtained in a given sample period k corresponds to the error related to the input data vector of $2(N+1)$ clock periods before.

The systolic array of Fig. 9.2 exhibits several desirable features such as local interconnection, regularity, and simple control circuitry, that yields a simple implementation. A possible problem, as pointed out in [13], is the need to distribute a single clock throughout a large array, without incurring any clock skew.

The presented systolic array does not allow the computation of the tap-weight coefficients. A solution pointed out in [13] employs the array of Fig. 9.2 by freezing the array and applying an appropriate input signal sequence such that the tap-weight coefficients are made available at the array output $\varepsilon(l)$. An alternative way is to add a systolic array to solve the back-substitution problem [13]. The array is shown in Fig. 9.3 with the corresponding algorithm. The complete computation of the coefficient vector $\mathbf{w}(k)$ requires 2^{N+1} clock samples. In this array, the square cells produce the partial products involved in (9.11). The round cell performs the subtraction of the sum of the product result with an element of the vector $\underline{\mathbf{d}}(k-8)$, namely $\underline{d}_{5-i}(k-8)$. This cell also performs the division of the subtraction result by the element $u_{N+1-i,i+1}(k-8)$ of the matrix $\mathbf{U}(k-8)$. Starting with $i = 0$, the sum of products has no elements and as a consequence the round cell just performs the division $\frac{\underline{d}_{5-i}(k-8)}{u_{N+1-i,i+1}(k-8)}$. On the other hand, for $i = N$ all the square cells are actually taking part in the computation of the sum of products.

Note that in this case, in order to obtain $w_N(k-8)$, the results of all the cells starting from left to right must be ready, i.e., there is no pipelining involved.

Example 9.2. Let us choose a simple example, in order to illustrate how the systolic array implementation works, and compare the results with those belonging to the standard implementation of the QR-RLS algorithm. The chosen order is $N = 3$ and the forgetting factor is $\lambda = 0.99$.

Suppose that in an adaptive-filtering environment, the input signal consists of

$$x(k) = \sin(\omega_0 k)$$

where $\omega_0 = \frac{\pi}{250}$.

The desired signal is generated by applying the same sinusoid to an FIR filter whose coefficients are given by

$$\mathbf{w}_o = [1.0\ 0.9\ 0.1\ 0.2]^T$$

Solution. First consider the results obtained with the conventional QR-RLS algorithm. The contents of the vector $\underline{\mathbf{d}}(k)$ and of the matrix $\mathbf{U}(k)$ are given below for the first four iterations.

Iteration $k = 1$

$$\underline{\mathbf{d}}(k) = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0126 \end{bmatrix} \quad \mathbf{U}(k) = \begin{bmatrix} 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0126 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix} \tag{9.55}$$

Iteration $k = 2$

$$\underline{\mathbf{d}}(k) = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.0364 \\ 0.0125 \end{bmatrix} \quad \mathbf{U}(k) = \begin{bmatrix} 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0251 & 0.0126 & 0.0000 & 0.0000 \\ 0.0125 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix} \tag{9.56}$$

Iteration $k = 3$

$$\underline{\mathbf{d}}(k) = \begin{bmatrix} 0.0000 \\ 0.0616 \\ 0.0363 \\ 0.0124 \end{bmatrix} \quad \mathbf{U}(k) = \begin{bmatrix} 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0377 & 0.0251 & 0.0126 & 0.0000 \\ \boxed{0.0250} & \boxed{0.0125} & 0.0000 & 0.0000 \\ 0.0124 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix} \tag{9.57}$$

Iteration $k = 4$

$$\underline{\mathbf{d}}(k) = \begin{bmatrix} 0.0892 \\ 0.0613 \\ 0.0361 \\ 0.0124 \end{bmatrix} \quad \mathbf{U}(k) = \begin{bmatrix} 0.0502 & 0.0377 & 0.0251 & 0.0126 \\ 0.0375 & 0.0250 & 0.0125 & 0.0000 \\ 0.0249 & 0.0124 & 0.0000 & 0.0000 \\ \boxed{0.0124} & 0.0000 & 0.0000 & 0.0000 \end{bmatrix} \tag{9.58}$$

Iteration $k = 5$

$$\underline{\mathbf{d}}(k) = \begin{bmatrix} 0.1441 \\ 0.0668 \\ 0.0359 \\ 0.0123 \end{bmatrix} \quad \mathbf{U}(k) = \begin{bmatrix} 0.0785 & 0.0617 & 0.0449 & 0.0281 \\ 0.0409 & 0.0273 & 0.0136 & 0.0000 \\ 0.0248 & 0.0124 & 0.0000 & 0.0000 \\ 0.0123 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix} \tag{9.59}$$

The data stored in the systolic array implementation represent the elements of the vector $\underline{\mathbf{d}}(k)$ and of the matrix $\mathbf{U}(k)$ skewed in time. This data is shown below starting from the fourth iteration, since before that no data is available to the systolic array.

Observe when the elements of the $\mathbf{U}(k)$ appear stored at the systolic array. For example, consider the highlighted elements. In particular, the element $(4, 1)$ at instant $k = 4$ appears stored in the systolic array at instant $k = 10$, whereas the elements $(3, 1)$ and $(3, 2)$ at instant $k = 3$ appear stored in the systolic array at instants $k = 8$ and $k = 7$, respectively. Following the same line of thought, it is straightforward to understand how the remaining elements of the systolic array are calculated.

Iteration $k = 4$

$$\begin{bmatrix} 0. \\ 0. \\ 0. \\ 0. \end{bmatrix} \quad \begin{bmatrix} 0. & 0. & 0. & 0.0126 \\ 0. & 0. & 0. & \\ 0. & 0. & & \\ 0. & & & \end{bmatrix} \tag{9.60}$$

Iteration $k = 5$

$$\begin{bmatrix} 0. \\ 0. \\ 0. \\ 0. \end{bmatrix} \quad \begin{bmatrix} 0. & 0. & 0.0251 & 0.0281 \\ 0. & 0. & 0.0126 & \\ 0. & 0. & & \\ 0. & & & \end{bmatrix} \tag{9.61}$$

Iteration $k = 6$

$$\begin{bmatrix} 0. \\ 0. \\ 0. \\ 0. \end{bmatrix} \quad \begin{bmatrix} 0. & 0.0377 & 0.0449 & 0.0469 \\ 0. & 0.0251 & 0.0125 & \\ 0. & 0.0126 & & \\ 0. & & & \end{bmatrix} \tag{9.62}$$

Iteration $k = 7$

$$\begin{bmatrix} 0. \\ 0. \\ 0. \\ 0. \end{bmatrix} \quad \begin{bmatrix} 0.0502 & 0.0617 & 0.0670 & 0.0686 \\ 0.0377 & 0.0250 & 0.0136 & \\ 0.0251 & \boxed{0.0125} & & \\ 0.0126 & & & \end{bmatrix} \tag{9.63}$$

Iteration $k = 8$

$$\begin{bmatrix} 0.0892 \\ 0.0616 \\ 0.0364 \\ 0.0126 \end{bmatrix} \quad \begin{bmatrix} 0.0785 & 0.0870 & 0.0913 & 0.0927 \\ 0.0375 & 0.0273 & 0.0148 & \\ \boxed{0.0250} & 0.0124 & & \\ 0.0125 & & & \end{bmatrix} \tag{9.64}$$

Iteration $k = 9$

$$\begin{bmatrix} 0.1441 \\ 0.0613 \\ 0.0363 \\ 0.0125 \end{bmatrix} \quad \begin{bmatrix} 0.1070 & 0.1141 & 0.1179 & 0.1191 \\ 0.0409 & 0.0297 & 0.0160 & \\ 0.0249 & 0.0124 & & \\ 0.0124 & & & \end{bmatrix} \tag{9.65}$$

Iteration $k = 10$

$$\begin{bmatrix} 0.2014 \\ 0.0668 \\ 0.0361 \\ 0.0124 \end{bmatrix} \quad \begin{bmatrix} 0.1368 & 0.1430 & 0.1464 & 0.1475 \\ 0.0445 & 0.0319 & 0.0170 & \\ 0.0248 & 0.0123 & & \\ \boxed{0.0124} & & & \end{bmatrix} \tag{9.66}$$

Iteration $k = 11$

$$\begin{bmatrix} 0.2624 \\ 0.0726 \\ 0.0359 \\ 0.0124 \end{bmatrix} \quad \begin{bmatrix} 0.1681 & 0.1737 & 0.1768 & 0.1778 \\ 0.0479 & 0.0340 & 0.0180 & \\ 0.0246 & 0.0123 & & \\ 0.0123 & & & \end{bmatrix} \tag{9.67}$$

It is a good exercise for the reader to examine the elements of the vectors and matrices in (9.60)–(9.67) and detect when these elements appear in the corresponding vectors $\underline{\mathbf{d}}(k)$ and matrices $\mathbf{U}(k)$ of (9.55)–(9.59). □

9.4 Some Implementation Issues

Several articles related to implementation issues of the QR-RLS algorithm such as the elimination of square root computation [14], stability and quantization error analyses [15–18] are available in the open literature. In this section, some of these results are briefly reviewed.

The stability of the QR-RLS algorithm is the first issue to be concerned when considering a real implementation. Fortunately, the QR-RLS algorithm implemented in finite precision was proved stable in the bounded input/bounded output sense in [16]. The proof was based on the analysis of the bounds for the internal recursions of the algorithm [16, 17]. From another study based on the

quantization-error propagation in the finite-precision implementation of the QR-RLS algorithm, it was possible to derive the error recursions for the main quantities of the algorithm, leading to the stability conditions of the QR-RLS algorithm [18]. The convergence on average of the QR-RLS algorithm can be guaranteed if the following inequality is satisfied [18]:

$$\lambda^{1/2} \parallel \tilde{\mathbf{Q}}_Q(k) \parallel_2 \leq 1 \tag{9.68}$$

where the two norm $\parallel \cdot \parallel_2$ of a matrix used here is the square root of the largest eigenvalue and the notation $[\cdot]_Q$ denotes the finite-precision version of $[\cdot]$. Therefore,

$$\parallel \tilde{\mathbf{Q}}_Q(k) \parallel_2 = \text{MAX}_i \sqrt{\cos_Q^2 \theta_i(k) + \sin_Q^2 \theta_i(k)} \tag{9.69}$$

where $\text{MAX}_i[\cdot]$ is the maximum value of $[\cdot]$. The stability condition can be rewritten as follows:

$$\lambda \leq \frac{1}{\text{MAX}_i \left[\cos_Q^2 \theta_i(k) + \sin_Q^2 \theta_i(k)\right]} \tag{9.70}$$

It can then be concluded that keeping the product of the forgetting factor and the maximum eigenvalue of the Givens rotations smaller than unity is a sufficient condition to guarantee the stability.

For the implementation of any adaptive algorithm, it is necessary to estimate quantitatively the dynamic range of all internal variables of the algorithm in order to determine the length of all the registers required in the actual implementation. Although this issue should be considered in the implementation of any adaptive-filtering algorithm, it is particularly relevant in the QR-RLS algorithms due to their large number of internal variables. The first attempt to address this problem was reported in [17], where expressions for the steady-state values of the cosines and sines of the Givens rotations were determined, as well as the bounds for the dynamic range of the information stored in the processing cells. The full quantitative analysis of the dynamic range of all internal quantities of the QR-RLS algorithm was presented in [18] for the conventional and systolic-array forms. For fixed-point implementation, it is important to determine the internal signal with the largest energy such that frequent overflow in the internal variables of the QR-RLS algorithm can be avoided. The first entry of the triangularized information matrix can be shown to have the largest energy [18] and its steady-state value is approximately

$$u_{0,0}(k) \approx \frac{\sigma_x}{\sqrt{1-\lambda}} \tag{9.71}$$

where σ_x^2 is the variance of the input signal.

The procedure to derive the results above discussed consists of first analyzing the QR-RLS algorithm for ideal infinite-precision implementation. The second step is modeling the quantization errors and deriving the recursive equations that include the overall error in each quantity of the QR-RLS algorithm [18]. Then conditions to

guarantee the stability of the algorithm could be derived. A further step is to derive closed-form solutions to the mean-squared values of the deviations in the internal variables of the algorithm due to finite-precision operations. The main objective in this step is to obtain the excess mean-square error and the variance of the deviation in the tap-weight coefficients. Analytical expressions for these quantities are not very simple unless a number of assumptions about the input and reference signals are assumed [18]. However, they are useful to the designer.

9.5 Fast QR-RLS Algorithm

For the derivation of the fast QR-RLS algorithms, it is first necessary to study the solutions of the forward and backward prediction problems. As seen in Chaps. 7 and 8, the predictor solutions were also required in the derivation of the lattice-based and the fast transversal RLS algorithms.

A family of fast QR-RLS algorithms can be generated depending on the following aspects of their derivation:

- The type of triangularization applied to the input signal matrix, taking into consideration the notation adopted in this book where the first element of the data vectors corresponds to the most recent data. The upper triangularization is related to the updating of forward prediction errors, whereas the lower triangularization involves the updating of backward prediction errors.
- The type of error utilized in the updating process, namely, if it is a priori or a posteriori error.

Table 9.2 shows the classification of the fast QR-RLS algorithms indicating the references where the specific algorithms can be found. Although these algorithms are comparable in terms of computational complexity, those based on backward prediction errors (which utilize lower triangularization of the information matrix) are numerically stable when implemented in finite precision. This good numerical behavior is related to backward consistency and minimal properties inherent to these algorithms [20].

In this section, we start with the application of the QR decomposition to the lower triangularization of the input signal information matrix. Then, the decomposition is applied to the backward and forward prediction problems. This type of triangularization is related to the updating of backward prediction errors.

A fast QR-RLS algorithm is derived by performing the triangularization of the information matrix in this alternative form, namely by generating a lower triangular

Table 9.2 Classification of the fast QR-RLS algorithms

	Prediction	
Error type	Forward	Backward
A priori	[9]	[10, 11]
A posteriori	[4]	[8, 19]

matrix, and by first applying the triangularization to the backward linear prediction problem. Originally, the algorithm to be presented here was proposed in [5] and later detailed in [7] and [8]. The derivations are quite similar to those presented for the standard QR-RLS algorithm. Therefore, we will use the previous results in order to avoid unnecessary repetition. In order to accomplish this objective while avoiding confusion, the following notations are, respectively, used for the triangularization matrix and the lower triangular matrices $\mathcal{Q}$ and $\mathcal{U}$. These matrices have the following forms

$$\mathcal{U}(k) = \begin{bmatrix} 0 & 0 & \cdots & 0 & u_{1,N+1} \\ 0 & 0 & \cdots & u_{2,N} & u_{2,N+1} \\ \vdots & & & \vdots & \vdots \\ u_{N+1,1} & u_{N+1,2} & \cdots & u_{N+1,N} & u_{N+1,N+1} \end{bmatrix} \tag{9.72}$$

$$\begin{aligned} \tilde{\mathcal{Q}}(k) = & \begin{bmatrix} \cos\theta_N(k) & \cdots & 0 & \cdots & -\sin\theta_N(k) & \mathbf{0} \\ \vdots & & & & \vdots & \vdots \\ 0 & & \mathbf{I}_{k-N-1} & & 0 & \vdots \\ \vdots & & & & \vdots & \vdots \\ \sin\theta_N(k) & \cdots & 0 & \cdots & \cos\theta_N(k) & \mathbf{0} \\ & & \mathbf{0} & & & \mathbf{I}_N \end{bmatrix} \\ & \cdot \begin{bmatrix} \cos\theta_{N-1}(k) & \cdots & 0 & \cdots & -\sin\theta_{N-1}(k) & 0 \\ \vdots & & & & \vdots & \vdots \\ 0 & & \mathbf{I}_{k-N} & & 0 & \vdots \\ \vdots & & & & \vdots & \vdots \\ \sin\theta_{N-1}(k) & \cdots & 0 & \cdots & \cos\theta_{N-1}(k) & 0 \\ 0 & \cdots & 0 & \cdots & 0 & \mathbf{I}_{N-1} \end{bmatrix} \\ & \cdots \begin{bmatrix} \cos\theta_0(k) & \cdots & 0 & \cdots & -\sin\theta_0(k) \\ \vdots & & & & \vdots \\ 0 & & \mathbf{I}_{k-1} & & 0 \\ \vdots & & & & \vdots \\ \sin\theta_0(k) & \cdots & 0 & \cdots & \cos\theta_0(k) \end{bmatrix} \end{aligned} \tag{9.73}$$

The triangularization procedure has the following general form

$$\begin{aligned} \mathcal{Q}(k)\underline{\mathbf{X}}(k) = \tilde{\mathcal{Q}}(k) & \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \tilde{\mathcal{Q}}(k-1) \end{bmatrix} \begin{bmatrix} \mathbf{I}_2 & \mathbf{0} \\ \mathbf{0} & \tilde{\mathcal{Q}}(k-2) \end{bmatrix} \\ & \cdots \begin{bmatrix} \mathbf{I}_{k-N} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathcal{Q}}(k-N) \end{bmatrix} \underline{\mathbf{X}}(k) \end{aligned}$$

$$= \begin{bmatrix} \mathbf{0} \\ \underbrace{\mathcal{U}(k)}_{N+1} \end{bmatrix} \begin{matrix} \} k-N \\ \} N+1 \end{matrix} \tag{9.74}$$

where $\mathcal{Q}(k)$ is a $(k+1)$ by $(k+1)$ matrix which represents the overall triangularization matrix.

As usual the multiplication by zero elements can be avoided by replacing $\tilde{\mathcal{Q}}(k)$ by $\mathcal{Q}_\theta(k)$, where the increasing $\mathbf{I}_{k-N-1}$ section of $\tilde{\mathcal{Q}}(k)$ is removed very much like in (9.38) and (9.39). The resulting equation is

$$\mathcal{Q}_\theta(k) \begin{bmatrix} \mathbf{x}^T(k) \\ \lambda^{1/2}\mathcal{U}(k-1) \end{bmatrix} = \mathcal{Q}'_{\theta_N}(k)\mathcal{Q}'_{\theta_{N-1}}(k)\cdots\mathcal{Q}'_{\theta_i}(k) \begin{bmatrix} \mathbf{x}'_i(k) \\ \mathcal{U}'_i(k) \end{bmatrix} \tag{9.75}$$

where $\mathcal{Q}'_{\theta_i}(k)$ is derived from $\mathcal{Q}'_i(k)$ by removing the $\mathbf{I}_{k-N-1}$ section of $\mathcal{Q}'_i(k)$ along with the corresponding rows and columns, resulting in the following form

$$\mathcal{Q}'_{\theta_i}(k) = \begin{bmatrix} \cos\theta_i(k) & \cdots & 0 & \cdots & -\sin\theta_i(k) & \cdots & 0 \\ \vdots & & & & \vdots & & \vdots \\ 0 & & \mathbf{I}_{N-i} & & 0 & \cdots & 0 \\ \vdots & & & & \vdots & & \vdots \\ \sin\theta_i(k) & \cdots & 0 & \cdots & \cos\theta_i(k) & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \mathbf{I}_i \\ 0 & \cdots & 0 & \cdots & 0 & & \end{bmatrix} \tag{9.76}$$

The Givens rotation elements are calculated by

$$\cos\theta_i(k) = \frac{[\mathcal{U}'_i(k)]_{N+1-i,i+1}}{c_i} \tag{9.77}$$

$$\sin\theta_i(k) = \frac{x'_i(k-i)}{c_i} \tag{9.78}$$

where $c_i = \sqrt{[\mathcal{U}'_i(k)]^2_{N+1-i,i+1} + x'^2_i(k-i)}$, and $[\cdot]_{i,j}$ denotes the (i,j) element of the matrix.

9.5.1 Backward Prediction Problem

In the backward prediction problem, the desired signal and vector are respectively

$$d_b(k+1) = x(k-N) \tag{9.79}$$

$$\mathbf{d}_b(k+1) = \begin{bmatrix} x(k-N) \\ \lambda^{1/2}x(k-N-1) \\ \vdots \\ \lambda^{\frac{k-N}{2}}x(0) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{9.80}$$

The reader should note that in the present case an extra row was added to the vector $\mathbf{d}_b(k+1)$. For example, the dimension of $\mathbf{d}_b(k+1)$ is now $(k+2)$ by 1. The backward-prediction-error vector is given by

$$\begin{aligned} \boldsymbol{\varepsilon}_b(k+1) &= \mathbf{d}_b(k+1) - \underline{\mathbf{X}}(k+1)\mathbf{w}_b(k+1) \\ &= [\underline{\mathbf{X}}(k+1)\ \mathbf{d}_b(k+1)] \begin{bmatrix} -\mathbf{w}_b(k+1) \\ 1 \end{bmatrix} \end{aligned} \tag{9.81}$$

The triangularization matrix $\mathcal{Q}(k+1)$ of the input data matrix can be applied to the backward prediction error resulting in

$$\mathcal{Q}(k+1)\boldsymbol{\varepsilon}_b(k+1) = \mathcal{Q}(k+1)\mathbf{d}_b(k+1) - \begin{bmatrix} \mathbf{0} \\ \mathcal{U}(k+1) \end{bmatrix} \mathbf{w}_b(k+1) \tag{9.82}$$

or equivalently

$$\boldsymbol{\varepsilon}_{bq}(k+1) = \mathbf{d}_{bq}(k+1) - \begin{bmatrix} \mathbf{0} \\ \mathcal{U}(k+1) \end{bmatrix} \mathbf{w}_b(k+1) \tag{9.83}$$

From equations and (9.81) and (9.83), it follows that

$$\begin{aligned} \boldsymbol{\varepsilon}_{bq}(k+1) &= \mathcal{Q}(k+1)[\underline{\mathbf{X}}(k+1)\ \mathbf{d}_b(k+1)] \begin{bmatrix} -\mathbf{w}_b(k+1) \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} & \varepsilon_{bq_1}(k+1) \\ \mathbf{0} & \varepsilon_{bq_2}(k+1) \\ & \vdots \\ & \varepsilon_{bq_{k-N+1}}(k+1) \\ \mathcal{U}(k+1) & \mathbf{x}_{q_3}(k+1) \end{bmatrix} \begin{bmatrix} -\mathbf{w}_b(k+1) \\ 1 \end{bmatrix} \end{aligned} \tag{9.84}$$

Also note that

$$[\underline{\mathbf{X}}(k+1)\ \mathbf{d}_b(k+1)] = \underline{\mathbf{X}}^{(N+2)}(k+1) \tag{9.85}$$

where $\underline{\mathbf{X}}^{(N+2)}(k+1)$ is an extended version of $\underline{\mathbf{X}}(k+1)$, with one input signal information vector added. In other words, $\underline{\mathbf{X}}^{(N+2)}(k+1)$ is the information matrix that would be obtained if one additional delay was added at the end of the delay line.

In order to avoid increasing vectors in the algorithm, $\varepsilon_{bq_1}(k+1)$, $\varepsilon_{bq_2}(k+1), \ldots,$ $\varepsilon_{bq_{k-N}}(k+1)$ can be eliminated in (9.84) through Givens rotations, as follows:

$$\begin{aligned}\mathbf{Q}_b(k+1)\boldsymbol{\varepsilon}_{bq}(k+1) &= \mathbf{Q}_b(k+1)\begin{bmatrix} & & \varepsilon_{bq_1}(k+1) \\ & \mathbf{0} & \varepsilon_{bq_2}(k+1) \\ & & \vdots \\ & & \varepsilon_{bq_{k-N+1}}(k+1) \\ \mathcal{U}(k+1) & & \mathbf{x}_{q_3}(k+1) \end{bmatrix}\begin{bmatrix} -\mathbf{w}_b(k+1) \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ & ||\boldsymbol{\varepsilon}_b(k+1)|| \\ \mathcal{U}(k+1) & \mathbf{x}_{q_3}(k+1) \end{bmatrix}\begin{bmatrix} -\mathbf{w}_b(k+1) \\ 1 \end{bmatrix} \end{aligned} \tag{9.86}$$

Note that by induction $[\mathcal{U}]_{N+1-i,i+1}(k+1) = ||\boldsymbol{\varepsilon}_{b,i}(k+1)||$, where $||\boldsymbol{\varepsilon}_{b,i}(k+1)||^2$ corresponds to the least-square backward prediction error of an $(i-1)$th-order predictor.

9.5.2 Forward Prediction Problem

In the forward prediction problem, the following relations are valid[1]:

$$d_f(k) = x(k+1) \tag{9.87}$$

$$\mathbf{d}_f(k) = \begin{bmatrix} x(k+1) \\ \lambda^{1/2}x(k) \\ \vdots \\ \lambda^{\frac{k+1}{2}}x(0) \end{bmatrix} \tag{9.88}$$

$$\boldsymbol{\varepsilon}_f(k) = \mathbf{d}_f(k) - \begin{bmatrix} \underline{\mathbf{X}}(k) \\ \mathbf{0} \end{bmatrix}\mathbf{w}_f(k) \tag{9.89}$$

where $d_f(k)$ is the desired signal, $\mathbf{d}_f(k)$ is the desired signal vector, and $\boldsymbol{\varepsilon}_f(k)$ is the error signal vector.

[1]The reader should note that here the definition of forward prediction error is slightly different from that used in Chaps. 7 and 8, where in the present case we are using the input and desired signals one step ahead. This allows us to use the same information matrix as the conventional QR-Decomposition algorithm of Sect. 9.2.3.

Now, we can consider applying the QR decomposition, as was previously done in (9.74) to the forward prediction error above defined. It should be noted that in the present case an extra row was added to the vectors $\boldsymbol{\varepsilon}_f(k)$ and $\mathbf{d}_f(k)$, as can be verified in the following relations:

$$\boldsymbol{\varepsilon}_f(k) = \left[\begin{array}{c|c} \mathbf{d}_f(k) & \underline{\mathbf{X}}(k) \\ & \mathbf{0} \end{array}\right] \begin{bmatrix} 1 \\ -\mathbf{w}_f(k) \end{bmatrix} \tag{9.90}$$

and

$$\begin{aligned} \boldsymbol{\varepsilon}_{fq}(k) &= \begin{bmatrix} \mathcal{Q}(k) & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \left[\begin{array}{c|c} \mathbf{d}_f(k) & \underline{\mathbf{X}}(k) \\ & \mathbf{0} \end{array}\right] \begin{bmatrix} 1 \\ -\mathbf{w}_f(k) \end{bmatrix} \\ &= \begin{bmatrix} \varepsilon_{fq_1}(k) & \\ \vdots & \mathbf{0} \\ \varepsilon_{fq_{k-N}}(k) & \\ \mathbf{x}_{q_2}(k) & \mathcal{U}(k) \\ \lambda^{\frac{k+1}{2}} x(0) & \mathbf{0} \end{bmatrix} \begin{bmatrix} 1 \\ -\mathbf{w}_f(k) \end{bmatrix} \end{aligned} \tag{9.91}$$

Note that:

$$\left[\begin{array}{c|c} \mathbf{d}_f(k) & \underline{\mathbf{X}}(k) \\ & \mathbf{0} \end{array}\right] = \underline{\mathbf{X}}^{(N+2)}(k+1) \tag{9.92}$$

which is an order extended version of $\underline{\mathbf{X}}(k+1)$ and has dimension $(k+2)$ by $(N+2)$.

In order to recursively solve (9.91) without dealing with ever-increasing matrices, a set of Givens rotations are applied in order to eliminate $\varepsilon_{fq_1}(k)$, $\varepsilon_{fq_2}(k)$, $\ldots$, $\varepsilon_{fq_{k-N}}(k)$, such that the information matrix that premultiplies the vector $[1 \;\; -\mathbf{w}_f(k)]^T$ is triangularized. The Givens rotations can recursively be obtained by

$$\begin{aligned} \mathbf{Q}_f(k) &= \tilde{\mathbf{Q}}_f(k) \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_f(k-1) \end{bmatrix} \\ &= \tilde{\mathbf{Q}}_f(k) \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{Q}}_f(k-1) \end{bmatrix} \cdots \begin{bmatrix} \mathbf{I}_{k-N-1} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{Q}}_f(N+1) \end{bmatrix} \end{aligned} \tag{9.93}$$

where $\tilde{\mathbf{Q}}_f(k)$ is defined as

$$\tilde{\mathbf{Q}}_f(k) = \begin{bmatrix} \cos\theta_f(k) & \cdots & 0 & \cdots & -\sin\theta_f(k) \\ \vdots & & & & \vdots \\ 0 & & \mathbf{I}_k & & 0 \\ \vdots & & & & \vdots \\ \sin\theta_f(k) & \cdots & 0 & \cdots & \cos\theta_f(k) \end{bmatrix} \tag{9.94}$$

If in each iteration, the above rotation is applied to (9.91), we have

$$\begin{aligned}\boldsymbol{\varepsilon}'_{fq}(k) &= \tilde{\mathbf{Q}}_f(k)\begin{bmatrix}1 & \mathbf{0}\\ \mathbf{0} & \mathbf{Q}_f(k-1)\end{bmatrix}\begin{bmatrix}\varepsilon_{fq_1}(k) & \\ \vdots & \mathbf{0}\\ \varepsilon_{fq_{k-N}}(k) & \\ \mathbf{x}_{q_2}(k) & \mathcal{U}(k)\\ \lambda^{\frac{k+1}{2}}x(0) & \mathbf{0}\end{bmatrix}\begin{bmatrix}1\\ -\mathbf{w}_f(k)\end{bmatrix}\\ &= \tilde{\mathbf{Q}}_f(k)\begin{bmatrix}\varepsilon_{fq_1}(k) & \\ 0 & \mathbf{0}\\ \vdots & \\ 0 & \\ \mathbf{x}_{q_2}(k) & \mathcal{U}(k)\\ \lambda^{1/2}||\boldsymbol{\varepsilon}_f(k-1)|| & \mathbf{0}\end{bmatrix}\begin{bmatrix}1\\ -\mathbf{w}_f(k)\end{bmatrix}\\ &= \begin{bmatrix}0 & \\ \vdots & \mathbf{0}\\ 0 & \\ \mathbf{x}_{q_2}(k) & \mathcal{U}(k)\\ ||\boldsymbol{\varepsilon}_f(k)|| & \mathbf{0}\end{bmatrix}\begin{bmatrix}1\\ -\mathbf{w}_f(k)\end{bmatrix}\end{aligned} \tag{9.95}$$

where

$$\cos\theta_f(k) = \frac{\lambda^{1/2}||\boldsymbol{\varepsilon}_f(k-1)||}{\sqrt{\lambda||\boldsymbol{\varepsilon}_f(k-1)||^2 + \varepsilon^2_{fq_1}(k)}} \tag{9.96}$$

$$\sin\theta_f(k) = \frac{\varepsilon_{fq_1}(k)}{\sqrt{\lambda||\boldsymbol{\varepsilon}_f(k-1)||^2 + \varepsilon^2_{fq_1}(k)}} \tag{9.97}$$

and $||\boldsymbol{\varepsilon}_f(k)||$ is the norm of the forward prediction error vector shown in (9.91). This result can be shown by evoking the fact that the last element of $\boldsymbol{\varepsilon}'_{fq}(k)$ is equal to $||\boldsymbol{\varepsilon}_f(k)||$, since $||\boldsymbol{\varepsilon}'_{fq}(k)|| = ||\boldsymbol{\varepsilon}_{fq}(k)|| = ||\boldsymbol{\varepsilon}_f(k)||$, because these error vectors are related through unitary transformations.

Also, it is worthwhile to recall that in (9.95) the relation $[\mathcal{U}]_{N+1-i,i+1}(k) = ||\boldsymbol{\varepsilon}_{b,i}(k)||$ is still valid (see (9.86)). Also, by induction, it can easily be shown from (9.91) that:

For $k = 0, 1, \ldots, N$

$$||\boldsymbol{\varepsilon}_f(k)|| = \lambda^{\frac{k+1}{2}}x(0)$$

for $k = N + 1$

$$||\boldsymbol{\varepsilon}'_{fq}(k)|| = ||\boldsymbol{\varepsilon}_f(k)|| = \sqrt{\lambda^{k+1}x^2(0) + \varepsilon^2_{fq_1}(k)}$$

for $k = N + 2$

$$\begin{aligned}||\boldsymbol{\varepsilon}_f(k)|| &= \sqrt{\lambda^{k+1}x^2(0) + \lambda\varepsilon^2_{fq_1}(k-1) + \varepsilon^2_{fq_1}(k)} \\ &= \sqrt{\lambda||\boldsymbol{\varepsilon}_f(k-1)||^2 + \varepsilon^2_{fq_1}(k)}\end{aligned}$$

for $k > N + 2$

$$||\boldsymbol{\varepsilon}_f(k)||^2 = \lambda||\boldsymbol{\varepsilon}_f(k-1)||^2 + \varepsilon^2_{fq_1}(k) \tag{9.98}$$

In the present case, it can be assumed that the partial triangularization can be performed at each iteration as follows:

$$\begin{bmatrix} 0 & \\ 0 & \mathbf{0} \\ \vdots & \\ 0 & \\ \mathbf{x}_{q_2}(k) & \mathcal{U}(k) \\ ||\boldsymbol{\varepsilon}_f(k)|| & \mathbf{0} \end{bmatrix} = \tilde{\mathbf{Q}}_f(k) \begin{bmatrix} \tilde{\mathcal{Q}}(k) & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} x(k+1) & \mathbf{x}^T(k) \\ \mathbf{0} & \mathbf{0} \\ \lambda^{1/2}\mathbf{x}_{q_2}(k-1) & \lambda^{1/2}\mathcal{U}(k-1) \\ \lambda^{1/2}||\boldsymbol{\varepsilon}_f(k-1)|| & \mathbf{0} \end{bmatrix} \tag{9.99}$$

Now we can eliminate $\mathbf{x}_{q_2}(k)$ through a set of rotations $\mathbf{Q}'_f(k+1)$ such that

$$\mathcal{U}^{(N+2)}(k+1) = \mathbf{Q}'_f(k+1) \begin{bmatrix} \mathbf{x}_{q_2}(k) & \mathcal{U}(k) \\ ||\boldsymbol{\varepsilon}_f(k)|| & \mathbf{0} \end{bmatrix} \tag{9.100}$$

where the superscript $(N + 2)$ in the above matrices denotes rotation matrices applied to data with $(N + 2)$ elements.

From the above equation, we can realize that $\mathbf{Q}'_f(k+1)$ consists of a series of rotations in the following order

$$\mathbf{Q}'_f(k+1) = \begin{bmatrix} \mathbf{I}_N & \mathbf{0} & & \\ \mathbf{0} & \cos\theta'_{f_1}(k+1) & -\sin\theta'_{f_1}(k+1) & \\ & \sin\theta'_{f_1}(k+1) & \cos\theta'_{f_1}(k+1) & \end{bmatrix}$$

$$\cdots \begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots\cdots & 0 \\ 0 & \cos\theta'_{f_N}(k+1) & 0 & \cdots & 0 \cdots 0 & -\sin\theta'_{f_N}(k+1) \\ \vdots & 0 & & & & 0 \\ \vdots & & & \mathbf{I}_{N-1} & \vdots & \vdots \\ 0 & \sin\theta'_{f_N}(k+1) & 0 & \cdots & 0 \cdots 0 & \cos\theta'_{f_N}(k+1) \end{bmatrix}$$

$$\cdot \begin{bmatrix} \cos\theta'_{f_{N+1}}(k+1) & 0 \cdots & 0 & \cdots 0 & -\sin\theta'_{f_{N+1}}(k+1) \\ 0 & & & & 0 \\ \vdots & & \mathbf{I}_N & & \vdots \\ \vdots & & & & \vdots \\ \sin\theta'_{f_{N+1}}(k+1) & 0 \cdots & 0 & \cdots 0 & \cos\theta'_{f_{N+1}}(k+1) \end{bmatrix} \tag{9.101}$$

where the rotation entries of $\mathbf{Q'}_f(k+1)$ are calculated as follows:

$$\begin{aligned} \mu_i &= \sqrt{\mu_{i-1}^2 + x_{q_2 i}^2(k)} \\ \cos\theta'_{f_{N+2-i}}(k+1) &= \frac{\mu_{i-1}}{\mu_i} \\ \sin\theta'_{f_{N+2-i}}(k+1) &= \frac{x_{q_2 i}(k)}{\mu_i} \end{aligned} \tag{9.102}$$

for $i = 1, \ldots, N+1$, where $\mu_0 = ||\boldsymbol{\varepsilon}_f(k)||$. Note that μ_{N+1} is the norm of the weighted backward prediction error $||\boldsymbol{\varepsilon}_{b,0}(k+1)||$, for a zero-order predictor (see (9.86)). The quantity $x_{q_2 i}(k)$ denotes the ith element of the vector $\mathbf{x}_{q_2}(k)$.

Since the above rotations, at instant k, are actually completing the triangularization of $\underline{\mathbf{X}}^{(N+2)}(k+1)$, it follows that

$$\tilde{\mathcal{Q}}^{(N+2)}(k+1) = \begin{bmatrix} \mathbf{I}_{k-N} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q'}_f(k+1) \end{bmatrix} \tilde{\mathbf{Q}}_f(k) \begin{bmatrix} \tilde{\mathcal{Q}}(k) & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \tag{9.103}$$

If the pinning vector, $[1\ 0 \ldots 0]^T$, is postmultiplied on both sides of the above equation, we obtain the following relation

$$\begin{aligned} \tilde{\mathcal{Q}}^{(N+2)}(k+1) \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} &= \begin{bmatrix} \mathbf{I}_{k-N} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q'}_f(k+1) \end{bmatrix} \tilde{\mathbf{Q}}_f(k) \begin{bmatrix} \tilde{\mathcal{Q}}(k) & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} \gamma^{(N+2)}(k+1) \\ 0 \\ \vdots \\ \mathbf{r}^{(N+2)}(k+1) \end{bmatrix} \Bigg\} N+2 \\ &= \begin{bmatrix} \mathbf{I}_{k-N} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q'}_f(k+1) \end{bmatrix} \tilde{\mathbf{Q}}_f(k) \begin{bmatrix} \gamma(k) \\ 0 \\ \vdots \\ \mathbf{r}(k) \\ 0 \end{bmatrix} \Bigg\} N+1 \end{aligned} \tag{9.104}$$

where $\mathbf{r}^{(N+2)}(k)$ and $\mathbf{r}(k)$ are vectors representing the last nonzero elements in the first column of $\tilde{\mathcal{Q}}^{(N+2)}(k)$ and $\tilde{\mathcal{Q}}(k)$, respectively, as can be seen in (9.73). Now, we can proceed by taking the product involving the matrix $\tilde{\mathbf{Q}}_f(k)$ resulting in the following relation

$$\begin{matrix} 1 \\ k-N-1 \\ N+1 \end{matrix}\left\{\begin{bmatrix} \gamma(k)\cos\theta_f(k) \\ 0 \\ \vdots \\ \mathbf{r}(k) \\ \gamma(k)\sin\theta_f(k) \end{bmatrix}\right. = \begin{bmatrix} \mathbf{I}_{k-N-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q'}_f^T(k+1) \end{bmatrix} \left.\begin{bmatrix} \gamma^{(N+2)}(k+1) \\ 0 \\ \vdots \\ \mathbf{r}^{(N+2)}(k+1) \end{bmatrix}\right\} \begin{matrix} 1 \\ k-N-1 \\ N+2 \end{matrix} \tag{9.105}$$

Since our interest is to calculate $\mathbf{r}(k+1)$, the above equation can be reduced to

$$\mathbf{Q'}_f(k+1)\begin{bmatrix} \mathbf{r}(k) \\ \gamma(k)\sin\theta_f(k) \end{bmatrix} = \mathbf{r}^{(N+2)}(k+1) \tag{9.106}$$

where the unused $k-N$ rows and columns were deleted and $\mathbf{r}(k+1)$ is the last $N+1$ rows of $\mathbf{r}^{(N+2)}(k+1)$. Now, since we have $\mathbf{r}(k+1)$ available as a function of known quantities, it is possible to calculate the angles of the reduced rotation matrix $\mathcal{Q}_\theta(k+1)$ using the following relation.

$$\begin{bmatrix} \gamma(k+1) \\ \mathbf{r}(k+1) \end{bmatrix} = \mathcal{Q}_\theta(k+1)\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{9.107}$$

By examining the definition of $\mathcal{Q}_\theta(k+1)$ in (9.75) and (9.76), it is possible to conclude that it has the following general form (see (9.29) and (9.30) for similar derivation)

$$\mathcal{Q}_\theta(k+1) = \left.\overbrace{\begin{bmatrix} * & * & \cdots & * \\ * & * & & \\ \vdots & & \ddots & \\ * & * & \cdots & * \end{bmatrix}}^{N+1}\right\} N+1 \tag{9.108}$$

where $*$ represents a nonzero element, with the first column given by

$$\begin{bmatrix} \prod_{i=0}^{N} \cos\theta_i(k+1) \\ \prod_{i=0}^{N-1} \cos\theta_i(k+1)\sin\theta_N(k+1) \\ \vdots \\ \prod_{i=0}^{j-1} \cos\theta_i(k+1)\sin\theta_j(k+1) \\ \vdots \\ \sin\theta_0(k+1) \end{bmatrix} \tag{9.109}$$

Although $\gamma(k+1)$ is not known, referring back to (9.107) and considering that each angle θ_i is individually responsible for an element in the vector $\mathbf{r}(k+1)$, it is possible to show that (9.107) can be solved by the following algorithm:

Initialize $\gamma'_0 = 1$
For $i = 1$ to $N+1$ calculate

$$\sin\theta_{i-1}(k+1) = \frac{r_{N+2-i}(k+1)}{\gamma'_0} \tag{9.110}$$

$$\begin{aligned} \gamma'^2_1 &= \gamma'^2_0[1-\sin^2\theta_{i-1}(k+1)] \\ &= \gamma'^2_0 - r^2_{N+2-i}(k+1) \end{aligned} \tag{9.111}$$

$$\cos\theta_{i-1}(k+1) = \frac{\gamma'_1}{\gamma'_0} \tag{9.112}$$

$$\gamma'_0 = \gamma'_1 \tag{9.113}$$

After computation is finished make $\gamma(k+1) = \gamma'_1$.

In the fast QR-RLS algorithm, we first calculate the rotated forward prediction error as in (9.99), followed by the calculation of the energy of the forward prediction error using (9.98) and the elements of $\tilde{\mathbf{Q}}_f(k)$ given in (9.96) and (9.97), respectively. The rotation entries of $\mathbf{Q}'_f(k+1)$ are calculated using the relations of (9.102), which in turn allow us to calculate $\mathbf{r}^{(N+2)}(k+1)$ through (9.106). Given $\mathbf{r}^{(N+2)}(k+1)$, the rotation angles θ_i can be calculated via (9.110)–(9.112). The remaining equations of the algorithm are the joint-processor section and the computation of the forward prediction error given by (9.51) and (9.54), respectively.

The resulting Algorithm 9.2 is almost the same as the hybrid QR-lattice algorithm of [8]. The main difference is the order the of computation of the angles θ_i. In [8] the computation starts from θ_N by employing the relation

$$\gamma(k+1) = \sqrt{1 - ||\mathbf{r}(k+1)||^2} \tag{9.114}$$

Algorithm 9.2 Fast QR-RLS algorithm based on a posteriori backward prediction error

Initialization

$||\boldsymbol{\varepsilon}_f(-1)|| = \delta \quad \delta$ small

All cosines with 1 (use for $k \leq N+1$)

All other variables with zero.

Do for each $k \geq 0$

$$\begin{bmatrix} \varepsilon_{fq_1}(k) \\ \mathbf{x}_{q_2}(k) \end{bmatrix} = \mathcal{Q}_\theta(k) \begin{bmatrix} x(k+1) \\ \lambda^{1/2}\mathbf{x}_{q_2}(k-1) \end{bmatrix} \tag{9.99}$$

$$||\boldsymbol{\varepsilon}_f(k)||^2 = \lambda||\boldsymbol{\varepsilon}_f(k-1)||^2 + \varepsilon^2_{fq_1}(k) \tag{9.98}$$

$$\sin\theta_f(k) = \frac{\varepsilon_{fq_1}(k)}{||\boldsymbol{\varepsilon}_f(k)||} \tag{9.97}$$

$$\mu_0 = ||\boldsymbol{\varepsilon}_f(k)||$$

Do for $i = 1$ to $N+1$

$$\mu_i = \sqrt{\mu^2_{i-1} + x^2_{q_2 i}(k)} \tag{9.102}$$

$$\cos\theta'_{f_{N+2-i}}(k+1) = \frac{\mu_{i-1}}{\mu_i} \tag{9.102}$$

$$\sin\theta'_{f_{N+2-i}}(k+1) = \frac{x_{q_2 i}(k)}{\mu_i} \tag{9.102}$$

End

$$\mathbf{r}^{(N+2)}(k+1) = \mathbf{Q}'_f(k+1) \begin{bmatrix} \mathbf{r}(k) \\ \gamma(k)\sin\theta_f(k) \end{bmatrix} \tag{9.106}$$

$\mathbf{r}(k+1) =$ last $N+1$ elements of $\mathbf{r}^{(N+2)}(k+1)$

$\gamma'_0 = 1$

Do for $i = 1$ to $N+1$

$$\sin\theta_{i-1}(k+1) = \frac{r_{N+2-i}(k+1)}{\gamma'_0} \tag{9.110}$$

$$\gamma'^2_1 = \gamma'^2_0 - r^2_{N+2-i}(k+1) \tag{9.111}$$

$$\cos\theta_{i-1}(k+1) = \frac{\gamma'_1}{\gamma'_0} \tag{9.112}$$

$\gamma'_0 = \gamma'_1$

End

$\gamma(k+1) = \gamma'_1$

Filter evolution

$$\begin{bmatrix} \varepsilon_{q_1}(k+1) \\ \mathbf{d}_{q_2}(k+1) \end{bmatrix} = \mathcal{Q}_\theta(k+1) \begin{bmatrix} d(k+1) \\ \lambda^{1/2}\mathbf{d}_{q_2}(k) \end{bmatrix} \tag{9.51}$$

$$\varepsilon(k+1) = \varepsilon_{q_1}(k+1)\gamma(k+1) \tag{9.54}$$

End

This algorithm is closely related to the normalized lattice algorithm (see [8]). Some key results are needed to establish the relation between these algorithms. For example it can be shown that the parameter $\gamma(k, N+1)$ of the lattice algorithms corresponds to $\gamma^2(k)$ in the fast QR algorithm.

In Problem 17, it is proved that the elements of $\mathbf{r}(k+1)$ in (9.106) correspond to normalized backward prediction a posteriori errors of distinct orders [8]. This is the explanation for the classification of Algorithm 9.2 in Table 9.2 as one which updates the a posteriori backward prediction errors.

Table 9.3 Results of the finite-precision implementation of the fast QR-RLS algorithm

	$\xi(k)_Q$
No. of bits	Experiment
16	$1.7\ 10^{-3}$
12	$2.0\ 10^{-3}$
10	$2.1\ 10^{-3}$

Example 9.3. In this example, the system identification problem described in Sect. 3.6.2 is solved using the QR-RLS algorithm described in this section. We implemented the fast QR-RLS algorithm with finite precision.

Solution. The main objective of this example is to test the stability of the fast QR-RLS algorithm. For that we run the algorithm implemented with fixed-point arithmetic. The wordlengths used are 16, 12, and 10 bits, respectively. We force the rotations to be kept passive. In other words, for each rotation the sum of the squares of the quantized sine and cosine are kept less or equal to one. Also, we test γ_1' to prevent it from becoming less than zero. With these measures, we did not notice any sign of divergence in our experiments. Table 9.3 shows the measured MSE in the finite-precision implementation, where the expected MSE for the infinite-precision implementation is 0.0015. The analysis of these results shows that the fast QR-RLS has low sensitivity to quantization effects and is comparable to the other stable RLS algorithms presented in this text. □

9.6 Conclusions and Further Reading

Motivated by the numerically well conditioned Givens rotations, two types of rotation-based algorithms were presented in this chapter. In both cases the QR decomposition implemented with orthogonal Givens rotations were employed. The first algorithm is computationally intensive (order N^2) and is mainly useful in applications where the input signal vector does not consist of time delayed elements. The advantages of this algorithm are its numerical stability and its systolic array implementation. The second class of algorithms explores the time-shift property of the input signal vector which is inherent to a number of applications, yielding the fast QR-RLS algorithms with order N numerical operations per output sample.

It should be noticed that the subject of QR-decomposition-based algorithms is not fully covered here. A complete approach to generating fast QR-RLS algorithm using lattice formulation is known [21–24]. In [21], the author applied QR decomposition to avoid inversion of covariance matrices in the multichannel problem employing lattice RLS formulation. A full orthogonalization of the resulting algorithm was later proposed in [23]. By using different formulations, the works of [22, 23], and [24] propose virtually identical QR-decomposition-based lattice RLS algorithms.

In terms of computational complexity, the fast QR-RLS algorithm presented in this chapter is more efficient. Although not discussed here, a solution to compute the adaptive-filter weights from the internal quantities of the fast QR-RLS algorithm is currently available [25].

Another family of algorithms employing QR decomposition are those that replace the Givens rotation by the Householder transformation [1]. The Householder transformation can be considered an efficient method to compute the QR decomposition and is known to yield more accurate results than the Givens rotations in finite-precision implementations. In [26], the fast Householder RLS adaptive-filtering algorithm was proposed and shown to require computational complexity on the order of $7N$. However, no stability proof for this algorithm exists so far. In another work, the Householder transformation is employed to derive a block-type RLS algorithm that can be mapped on a systolic-block Householder transformation [27]. In [28], by employing the Householder transformation, a QR-based LMS algorithm was proposed as a numerically stable and fast converging algorithm with $O[N]$ computational complexity.

A major drawback of the conventional QR-RLS algorithm is the backsubstitution algorithm which is required for computing the weight vector. In a systolic array, it can be implemented as shown in this chapter, through a bidirectional array that requires extra clock cycles. Alternatively, a two-dimensional array can be employed despite being more computationally expensive [13]. An approach called inverse QR method can be used to derive a QR-based RLS algorithm such that the weight vector can be calculated without backsubstitution [29, 30]; however, no formal proof of stability for this algorithm is known.

The QR decomposition has also been shown to be useful for the implementation of numerically stable nonlinear adaptive-filtering algorithms. In [31], a QR-based RLS algorithm for adaptive nonlinear filtering has been proposed.

Some performance evaluations of the QR-RLS and fast QR-RLS algorithms are found in this chapter where these algorithms were employed in some simulation examples.

9.7 Problems

1. If we consider each anti-diagonal element of $\lambda^{\frac{1}{2}}\mathbf{U}(k)$ as a scaling constant d_i, and we divide the input signal vector initially by a constant δ, we can derive a QR-decomposition algorithm without square roots as described below:

   The first two rows to be rotated are

   $$\begin{array}{cccc} \delta\tilde{x}(k) & \delta\tilde{x}(k-1) & \cdots & \delta\tilde{x}(k-N) \\ d_1\lambda^{1/2}\tilde{u}_{1,1}(k-1) & d_1\lambda^{1/2}\tilde{u}_{1,2}(k-1) & \cdots & d_1 \end{array}$$

   where $d_1 = \lambda^{1/2}u_{1,N+1}(k-1)$. The parameter δ can be initialized with 1.

Applying the Givens rotation to the rows above results in

$$\begin{matrix} \delta' x_1'(k) & \delta' x_1'(k-1)\cdots & \delta' x_1'(k-N+1) & 0 \\ d_1'\tilde{u}_{1,1}'(k) & d_1'\tilde{u}_{1,2}'(k)\cdots & d_1'\tilde{u}_{1,N}'(k) & d_1' \end{matrix}$$

where

$$\begin{aligned} & d_1'^2 = d_1^2 + \delta^2 \tilde{x}^2(k-N) \\ & c = \frac{d_1^2}{d_1^2+\delta^2\tilde{x}^2(k-N)} \\ & \delta'^2 = \frac{d_1^2\delta^2}{d_1^2+\delta^2\tilde{x}^2(k-N)} \\ & s = \frac{\delta^2\tilde{x}(k-N)}{d_1^2+\delta^2\tilde{x}^2(k-N)} \\ & x_1'(k-N+i) = \tilde{x}(k-N+i) - \tilde{x}(k-N)\lambda^{1/2}\tilde{u}_{1,N-i+1}(k-1) \\ & \tilde{u}_{1,N-i+1}'(k) = c\lambda^{1/2}\tilde{u}_{1,N+1-i}(k-1) + s\tilde{x}(k-N+i). \end{aligned}$$

The same procedure can be used to triangularize completely the input signal matrix.

(a) Using the above procedure derive a QR-RLS algorithm without square roots.
(b) Compare the computational complexity of the QR-RLS algorithms with and without square roots.
(c) Show that the triangularized matrix $\tilde{\mathbf{U}}(k)$ is related with $\mathbf{U}(k)$ through

$$\mathbf{U}(k) = \mathbf{D}'\tilde{\mathbf{U}}(k)$$

where $\mathbf{D}'$ is a diagonal matrix with the diagonal elements given by d_i' for $i = 1, 2, \ldots, N+1$.

2. Since $\mathbf{Q}^T(k)\mathbf{Q}(k) = \mathbf{I}_{k+1}$, the following identity is valid for any matrix $\mathbf{A}$ and $\mathbf{B}$:

$$\mathbf{C}^T\mathbf{D} = \mathbf{A}^T\mathbf{B} \text{ for } \mathbf{Q}(k)\,[\mathbf{A} \mid \mathbf{B}] = [\mathbf{C} \mid \mathbf{D}]$$

where $\mathbf{Q}(k), \mathbf{A}, \mathbf{B}, \mathbf{C}$,and $\mathbf{D}$ have the appropriate dimensions. By choosing $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and $\mathbf{D}$ appropriately, derive the following relations.

(a) $\mathbf{U}^T(k)\mathbf{U}(k) = \lambda\mathbf{U}^T(k-1)\mathbf{U}(k-1) + \mathbf{x}(k)\mathbf{x}^T(k)$
(b) $\mathbf{p}_D(k) = \lambda\mathbf{p}_D(k-1) + \mathbf{x}(k)d(k)$
where $\mathbf{p}_D(k) = \Sigma_{i=0}^{k}\lambda^k\mathbf{x}(i)d(i)$
(c) $\mathbf{U}^T(k)\mathbf{U}^{-T}(k)\mathbf{x}(k) = \mathbf{x}(k)$
where $\mathbf{U}^{-T}(k) = \left[\mathbf{U}^{-1}(k)\right]^T$
(d) $\mathbf{p}_D^T(k)\mathbf{U}^{-1}(k)\mathbf{U}^{-T}(k)\mathbf{x}(k) + \varepsilon_{q1}(k)\gamma(k) = d(k)$.

3. Partitioning $\mathbf{Q}_\theta(k)$ as follows:

$$\mathbf{Q}_\theta(k) = \begin{bmatrix} \gamma(k) & \mathbf{q}_\theta^T(k) \\ \mathbf{q}_\theta'(k) & \mathbf{Q}_{\theta r}(k) \end{bmatrix}$$

show from (9.51) and (9.39) that

$$\mathbf{q}_\theta^T(k)\lambda^{1/2}\mathbf{U}(k-1) + \gamma(k)\mathbf{x}^T(k) = \mathbf{0}^T$$
$$\mathbf{q}_\theta^T(k)\lambda^{1/2}\mathbf{d}_{q2}(k-1) + \gamma(k)d(k) = \varepsilon_{q1}(k).$$

4. Using the relations of the previous two problems and the fact that $\mathbf{U}(k)\mathbf{w}(k) = \mathbf{d}_{q2}(k)$, show that

   (a) $e(k) = \frac{\varepsilon_{q1}(k)}{\gamma(k)}$
   (b) $\varepsilon(k) = e(k)\gamma^2(k)$
   (c) $\varepsilon_{q1}(k) = \sqrt{\varepsilon(k)e(k)}$.

5. Show that $\mathbf{U}^T(k)\mathbf{d}_{q2}(k) = \mathbf{p}_D(k)$.
6. Using some of the formulas of the conventional RLS algorithm show that $\gamma^2(k) = 1 - \mathbf{x}^T(k)\mathbf{R}_D^{-1}(k)\mathbf{x}(k)$.
7. The QR-RLS algorithm is used to predict the signal $x(k) = \cos(\pi k/3)$ using a second-order FIR filter with the first tap fixed at 1. Note that we are interested in minimizing the MSE of the FIR output error. Given $\lambda = 0.985$, calculate $y(k)$ and the filter coefficients for the first ten iterations.
8. Use the QR-RLS algorithm to identify a system with the transfer function given below. The input signal is uniformly distributed white noise with variance $\sigma_x^2 = 1$ and the measurement noise is Gaussian white noise uncorrelated with the input with variance $\sigma_n^2 = 10^{-3}$. The adaptive filter has 12 coefficients.

$$H(z) = \frac{1 - z^{-12}}{1 - z^{-1}}$$

   (a) Run the algorithm for $\lambda = 1$, $\lambda = 0.99$, and $\lambda = 0.97$. Comment on the convergence behavior in each case.
   (b) Plot the obtained FIR filter frequency response at any iteration after convergence is achieved and compare with the unknown system.

9. Perform the equalization of a channel with the following impulse response

$$h(k) = \sum_{l=k}^{10}(l-10)[u(k) - u(k-10)]$$

   where $u(k)$ is a step sequence.

   Use a known training signal that consists of a binary $(-1, 1)$ random signal. An additional Gaussian white noise with variance 10^{-2} is present at the channel output.

   (a) Apply the QR-RLS with an appropriate λ and find the impulse response of an equalizer with 50 coefficients.
   (b) Convolve the equalizer impulse response at a given iteration after convergence, with the channel impulse response and comment on the result.

10. In a system identification problem the input signal is generated by an autoregressive process given by

$$x(k) = -1.2x(k-1) - 0.81x(k-2) + n_x(k)$$

where $n_x(k)$ is zero-mean Gaussian white noise with variance such that $\sigma_x^2 = 1$. The unknown system is described by

$$H(z) = 1 + 0.9z^{-1} + 0.1z^{-2} + 0.2z^{-3}$$

The adaptive filter is also a third-order FIR filter. Using the QR-RLS algorithm:

Choose an appropriate λ, run an ensemble of 20 experiments, and plot the average learning curve.

11. The QR-RLS algorithm is applied to identify a 7th-order time-varying unknown system whose coefficients are first-order Markov processes with $\lambda_{\mathbf{W}} = 0.999$ and $\sigma_{\mathbf{W}}^2 = 0.001$. The initial time-varying system multiplier coefficients are

$$\mathbf{w}_o^T = [0.03490 \ -0.01100 \ -0.06864 \ 0.22391 \ 0.55686 \ 0.35798 \ -0.02390 \ -0.07594]$$

The input signal is Gaussian white noise with variance $\sigma_x^2 = 0.7$, and the measurement noise is also Gaussian white noise independent of the input signal and of the elements of $\mathbf{n}_{\mathbf{W}}(k)$, with variance $\sigma_n^2 = 0.01$.

(a) For $\lambda = 0.97$ measure the excess MSE.
(b) Repeat (a) for $\lambda = \lambda_{\text{opt}}$.

12. Suppose a 15th-order FIR digital filter with multiplier coefficients given below is identified through an adaptive FIR filter of the same order using the QR-RLS algorithm. Considering that fixed-point arithmetic is used and for 10 independent runs, calculate an estimate of the expected value of $||\Delta\mathbf{w}(k)_Q||^2$ and $\xi(k)_Q$ for the following case.

Additional noise : white noise with variance	$\sigma_n^2 = 0.0015$
Coefficients wordlength:	$b_c = 16$ bits
Signal wordlength:	$b_d = 16$ bits
Input signal: Gaussian white noise with variance	$\sigma_x^2 = 0.7$
	$\lambda = 0.99$

$\mathbf{w}_o^T = [0.0219360 \ 0.0015786 \ -0.0602449 \ -0.0118907 \ 0.1375379 \ 0.0574545 \ -0.3216703 \ -0.5287203 \ -0.2957797 \ 0.0002043 \ 0.290670 \ -0.0353349 \ -0.0068210 \ 0.0026067 \ 0.0010333 \ -0.0143593]$

Plot the learning curves for the finite- and infinite-precision implementations.

13. Repeat the above problem for the following cases

    (a) $\sigma_n^2 = 0.01, b_c = 9$ bits, $b_d = 9$ bits, $\sigma_x^2 = 0.7, \lambda = 0.98$.
    (b) $\sigma_n^2 = 0.1, b_c = 10$ bits, $b_d = 10$ bits, $\sigma_x^2 = 0.8, \lambda = 0.98$.
    (c) $\sigma_n^2 = 0.05, b_c = 8$ bits, $b_d = 16$ bits, $\sigma_x^2 = 0.8, \lambda = 0.98$.

14. Repeat Problem 12 for the case where the input signal is a first-order Markov process with $\lambda_{\mathbf{x}} = 0.95$.
15. Repeat Problem 9 using the fast QR-RLS algorithm.
16. From (9.74) it is straightforward to show that

$$\underline{\mathbf{X}}(k) = \mathcal{Q}^T(k) \begin{bmatrix} \mathbf{0} \\ \mathcal{U}(k) \end{bmatrix}$$

$$= [\mathcal{Q}_u(k) \;\; \mathcal{Q}_d(k)] \begin{bmatrix} \mathbf{0} \\ \mathcal{U}(k) \end{bmatrix}$$

    where $\mathcal{Q}(k) = [\mathcal{Q}_u(k)\mathcal{Q}_d(k)]^T$.

    (a) Using the above relation show that the elements of $\mathbf{x}_{q_2}(k)$ in (9.95) are given by

$$x_{q_2 i}(k) = [\mathbf{q}_{di}^T(k) \;\; 0]\mathbf{d}_f(k)$$

    where $\mathbf{q}_{di}(k)$ is the ith column of $\mathcal{Q}_d(k)$.

    (b) Show that the a posteriori error vector for an Nth-order forward predictor can be given by

$$\boldsymbol{\varepsilon}_f(k, N+1) = \mathbf{d}_f(k) - \sum_{i=1}^{N+1} x_{q_2 i}(k) \begin{bmatrix} \mathbf{q}_{di}(k) \\ 0 \end{bmatrix}$$

    (c) Can the above expression be generalized to represent the a posteriori error vector for an $(N-j)$th-order forward predictor? See the expression below

$$\boldsymbol{\varepsilon}_f(k, N+1-j) = \mathbf{d}_f(k) - \sum_{i=j}^{N+1} x_{q_2 i}(k) \begin{bmatrix} \mathbf{q}_{di}(k) \\ 0 \end{bmatrix}$$

17. For the fast QR-RLS algorithm, show that the elements of $\mathbf{r}(k+1)$ correspond to a normalized backward prediction a posteriori error defined as

$$r_{N+1-i}(k) = \bar{\varepsilon}_b(k,i) = \frac{\varepsilon_b(k,i)}{||\boldsymbol{\varepsilon}_{b,i}(k)||} = \frac{\varepsilon_{bq_i}(k,i)}{||\boldsymbol{\varepsilon}_{b,i}(k)||} \prod_{j=0}^{i-1} \cos\theta_j(k)$$

where $\prod_{j=0}^{-1} = 1$, and $\varepsilon_b(k,i+1)$ is the a posteriori backward prediction error for a predictor of order i, with $i = 0, 1, \ldots$. Note that $||\boldsymbol{\varepsilon}_{b,i}(k)||^2$ corresponds to $\xi^d_{b_{\min}}(k,i+1)$ used in the lattice derivations of Chap. 7.

References

1. G.H. Golub, C.F. Van Loan, *Matrix Computations*, 2nd edn. (John Hopkins University Press, Baltimore, 1989)
2. W.H. Gentleman, H.T. Kung, Matrix triangularization by systolic arrays. Proc. SPIE, Real Time Signal Process. IV **298**, 19–26 (1981)
3. J.G. McWhirter, Recursive least-squares minimization using a systolic array. Proc. SPIE, Real Time Signal Process. VI **431**, 105–112 (1983)
4. J.M. Cioffi, The fast adaptive ROTOR's RLS algorithm. IEEE Trans. Acoust. Speech Signal Process. **38**, 631–653 (1990)
5. I.K. Proudler, J.G. McWhirter, Y.J. Shepherd, Fast QRD-based algorithms for least squares linear prediction, in *Proceedings of the IMA Conference on Mathematics in Signal Processing*, Warwick, England, Dec 1988, pp. 465–488
6. M.G. Bellanger, The FLS-QR algorithm for adaptive filtering. Signal Process. **17**, 291–304 (1984)
7. M.G. Bellanger, P.A. Regalia, The FLS-QR algorithm for adaptive filtering: The case of multichannel signals. Signal Process. **22**, 115–126 (1991)
8. P.A. Regalia, M.G. Bellanger, On the duality between fast QR methods and lattice methods in least squares adaptive filtering. IEEE Trans. Signal Process. **39**, 879–891 (1991)
9. J.A. Apolinário Jr., P.S.R. Diniz, A new fast QR algorithm based on a priori errors. IEEE Signal Process. Lett. **4**, 307–309 (1997)
10. M.D. Miranda, M. Gerken, A hybrid QR-lattice least squares algorithm using a priori errors. IEEE Trans. Signal Process. **45**, 2900–2911 (1997)
11. A.A. Rontogiannis, S. Theodoridis, New fast QR decomposition least squares adaptive algorithms. IEEE Trans. Signal Process. **46**, 2113–2121 (1998)
12. Z. Chi, J. Ma, K. Parhi, Hybrid annihilation transformation (HAT) for pipelining QRD-based least-square adaptive filters. IEEE Trans. Circ. Syst.-II: Analog Digital Signal Process. **48**, 661–674 (2001)
13. C.R. Ward, P.J. Hargrave, J.G. McWhirter, A novel algorithm and architecture for adaptive digital beamforming. IEEE Trans. Antenn. Propag. **34**, 338–346 (1986)
14. W.H. Gentleman, Least squares computations by Givens transformations without square roots. Inst. Maths. Appl. **12**, 329–336 (1973)
15. W.H. Gentleman, Error analysis of QR decompositions by Givens transformations. Lin. Algebra Appl. **10**, 189–197 (1975)
16. H. Leung, S. Haykin, Stability of recursive QRD-LS algorithms using finite-precision systolic array implementation. IEEE Trans. Acoust. Speech Signal Process. **37**, 760–763 (1989)

17. K.J.R. Liu, S.-F. Hsieh, K. Yao, C.-T. Chiu, Dynamic range, stability, and fault-tolerant capability of finite-precision RLS systolic array based on Givens rotations. IEEE Trans. Circ. Syst. **38**, 625–636 (1991)
18. P.S.R. Diniz, M.G. Siqueira, Fixed-point error analysis of the QR-recursive least squares algorithm. IEEE Trans. Circ. Syst. II: Analog Digital Signal Process. **43**, 334–348 (1995)
19. J.A. Apolinário Jr., M.G. Siqueira, P.S.R. Diniz, On fast QR algorithm based on backward prediction errors: New result and comparisons, in *Proceedings of the First IEEE Balkan Conference on Signal Processing, Communications, Circuits, and Systems*, Istanbul, Turkey, June 2000, CD-ROM, pp. 1–4
20. P.A. Regalia, Numerical stability properties of a QR-based fast least squares algorithm. IEEE Trans. Signal Process. **41**, 2096–2109 (1993)
21. P.S. Lewis, QR-based algorithms for multichannel adaptive least squares lattice filters. IEEE Trans. Acoust. Speech Signal Process. **38**, 421–432 (1990)
22. I.K. Proudler, J.G. McWhirter, T.J. Shepherd, Computationally efficient QR decomposition approach to least squares adaptive filtering. IEE Proc.-Part F **148**, 341–353 (1991)
23. B. Yang, J.F. Böhme, Rotation-based RLS algorithms: Unified derivations, numerical properties, and parallel implementations. IEEE Trans. Signal Process. **40**, 1151–1166 (1992)
24. F. Ling, Givens rotation based least squares lattice and related algorithms. IEEE Trans. Signal Process. **39**, 1541–1551 (1991)
25. M. Shoaib, S. Werner, J.A. Apolinário Jr., T.I. Laakso, Solution to the weight extraction problem in fast QR-decomposition RLS algorithms, in *Proceedings of the IEEE International Conference on Acoustics, Speech, Signal Processing*, Toulouse, France, 2006, pp. III-572–III-575
26. J.M. Cioffi, The fast Householder filters RLS adaptive filter, in *Proceedings of the IEEE International Conference on Acoustics, Speech, Signal Processing*, Albuquerque, NM, 1990, pp. 1619–1622
27. K.J.R. Liu, S.-F. Hsieh, K. Yao, Systolic block Householder transformation for RLS algorithm with two-level pipelined implementation. IEEE Trans. Signal Process. **40**, 946–957 (1992)
28. Z.-S. Liu, J. Li, A QR-based least mean squares algorithm for adaptive parameter estimation. IEEE Trans. Circ. Syst.-II: Analog Digital Signal Process. **45**, 321–329 (1998)
29. A. Ghirnikar, S.T. Alexander, Stable recursive least squares filtering using an inverse QR decomposition, in *Proceedings of the IEEE International Conference on Acoustics, Speech, Signal Processing*, Albuquerque, NM, 1990, pp. 1623–1626
30. S.T. Alexander, A. Ghirnikar, A method for recursive least squares filtering based upon an inverse QR decomposition. IEEE Trans. Signal Process. **41**, 20–30 (1993)
31. M. Syed, V.J. Mathews, QR-Decomposition based algorithms for adaptive Volterra filtering. IEEE Trans. Circ. Syst. I: Fund. Theor. Appl. **40**, 372–382 (1993)