A Method for Recursive Least Squares Filtering Based Upon an Inverse *QR* Decomposition

S. Thomas Alexander, *Senior Member, IEEE*, and Avinash L. Ghirnikar, *Member, IEEE*

Abstract—**A new computationally efficient algorithm for recursive least squares filtering is derived, which is based upon an inverse *QR* decomposition. The method solves directly for the time-recursive least squares filter vector, while avoiding the highly serial backsubstitution step required in previously derived direct *QR* approaches. Furthermore, the method employs orthogonal rotation operations to recursively update the filter, and thus preserves the inherent stability properties of *QR* approaches to recursive least squares filtering. The results of simulations over extremely long data sets are also presented, which suggest stability of the new time-recursive algorithm. Finally, parallel implementation of the resulting method is briefly discussed, and computational wavefronts are displayed.**

I. Introduction

ADAPTIVE filters minimizing the least squares (LS) error criterion have been applied in recent years to many problems in signal processing [1]–[5]. However, the widespread acceptance of recursive LS (RLS) filters has been impeded by a sometimes unacceptable numerical performance in limited precision environments. This degradation of performance is especially noticeable for the family of techniques collectively known as "fast" RLS filters. These fast algorithms are typically characterized by an absence of matrix multiplications and require $O(N)$ ("on the order of N") operations per sampling interval, where N is the number of coefficients in the filter. Unfortunately, finite precision implementations of these fast RLS filters have sometimes been observed to be numerically unstable [5]–[7], although recent work suggests that some fast algorithms may indeed be stabilized [8], [9].

These numerical problems associated with fast RLS filters have thus stimulated research into applying stable orthogonal linear algebraic transformations to the original RLS problem [1], [10]–[12]. A motivation for considering these transformations comes from numerical linear algebra, in which matrix transformations, or decompositions, are used to triangularize the multiplying matrix in a set of simultaneous linear equations. This triangularization is sometimes called the Cholesky factorization [13] of the original least squares data matrix.

Manuscript received November 2, 1990; revised October 15, 1991. This work was supported in part by the National Science Foundation under Grant MIP-8552571.

S. T. Alexander is with the Electrical and Computer Engineering Department, North Carolina State University, Raleigh, NC 27695-7911.

A. L. Ghirnikar was with the Electrical and Computer Engineering Department, North Carolina State University, Raleigh, NC 27695-7911. He is now with ViTel Communications Corporation, Santa Clara, CA 95054.

IEEE Log Number 9203348. .

One popular method for producing such a triangular structure is the *QR* decomposition using Givens rotations [15], which allows the unknown solution vector (i.e., the LS weight vector) to be found by backsubstitution. In the RLS case, the *QR* decomposition transforms the original RLS problem (which uses data covariance values) into a problem that uses only transformed data values. This, in turn, causes the numerical dynamic range of the transformed computational problem to be reduced to one-half the dynamic range of the original RLS problem [14], which has immediate benefit concerning stability considerations. Numerous researchers have observed that the time-recursive filtering problem exhibits more stable properties when implemented in the *QR* decomposition form [1], [10], [12].

Another important benefit of *QR*-based approaches is that the rotations computations are easily mapped onto systolic array structures for a parallel implementation [1], [16]. However, if the LS weight vector is desired at every time iteration then backsubstitution steps must be performed at each time interval to solve for the LS weights. Backsubstitution may be implemented on either a parallel two-dimensional array [17] or a linear bidirectional array [1], [16]. However, the two-dimensional array can become quite large for long filter lengths and consequently requires a substantial area for VLSI implementation. While simpler in structure, the linear array suffers more time delay since it requires approximately $2N$ additionally clock cycles to compute the N filter coefficients [1]. These reasons have made backsubstitution a costly operation to perform, and thus research interest has recently turned toward finding approaches that will allow time-recursive *QR*-based solutions for the weight vector without implementing backsubstitution.

This paper derives such a method, which is called the inverse *QR* algorithm. The derivation of the inverse *QR* method in this paper is done by a different method and from a different perspective than previous work on inverse factorizations [13]. Additionally, the current work extends the results of [13] to explicitly include the time-recursive updating necessary for recursive LS filtering, and an exponential forgetting factor is included for use in nonstationary environments.

The remainder of this paper is organized as follows. Section II reviews the problem of LS filtering using the direct *QR* decomposition. The backsubstitution step for the filter coefficients is denoted. Section III derives the

basic form for the LS weight vector update, and illustrates the natural appearance of the inverse Cholesky factor. Section IV, derives the entire inverse *QR* algorithm, and shows how a key result allows the highly stable Givens rotation method to be used in the implementation. Section V presents simulations that suggest long-term stability of the method, and also discusses parallelization of the inverse *QR* algorithm. An array structure is also discussed, which demonstrates the pipelined operation of the algorithm as well. Finally, a comparison is made between the inverse *QR* algorithm and another recently derived method that does not requires backsubstitution [19].

II. RLS Filtering by Direct *QR* Decomposition

To establish the notation for this paper, let the reference signal sequence be $d(i)$, and let the data signal used by the least squares filter be $x(i)$. In both cases, the index i runs from $1 \leq i \leq n$, where n is the current time value. Let the N-length vector $\boldsymbol{w}(n)$ be the RLS filter vector computed at time n, where the elements of $\boldsymbol{w}(n)$ are given by

$$\boldsymbol{w}(n) = [w_1(n), w_2(n), \cdots, w_N(n)]^T. \tag{2.1}$$

The notation $\hat{d}(i|n)$ will be used to denote the prediction of $d(i)$ using the transversal filter $\boldsymbol{w}(n)$, and is given by

$$\hat{d}(i|n) = \sum_{k=1}^{N} w_k(n)x(i-k+1). \tag{2.2}$$

The error incurred from predicting $d(i)$ using $\hat{d}(i|n)$ in (2.2) is therefore given by

$$e(i|n) = d(i) - \sum_{k=1}^{N} w_k(n)x(i-k+1). \tag{2.3}$$

The problem considered in this paper is the standard RLS problem [1], [2] of finding the least squares filter $\boldsymbol{w}(n)$, which minimizes the cumulative squared error $\epsilon(n)$ defined by

$$\epsilon(n) = \sum_{i=1}^{n} \lambda^{n-i} e^2(i|n) \tag{2.4}$$

where $e(i|n)$ is given by (2.3); and $0 \ll \lambda < 1$ is an exponential weighting factor. The motivation for considering *QR* decompositions is easily seen by expressing the previously described problem in a matrix–vector notation. Define the n-component vectors $\boldsymbol{e}(n)$ and $\boldsymbol{d}(n)$ as

$$\boldsymbol{e}(n) = [e(1|n), e(2|n), \cdots, e(n|n)]^T \tag{2.5a}$$

$$\boldsymbol{d}(n) = [d(1), d(2), \cdots, d(n)]^T. \tag{2.5b}$$

Using (2.5b), (2.3), and (2.1) in (2.5a) then gives

$$\boldsymbol{e}(n) = \boldsymbol{d}(n) - \boldsymbol{X}(n)\boldsymbol{w}(n) \tag{2.6}$$

where $\boldsymbol{X}(n)$ is the $n \times N$ data matrix defined by

$$\begin{bmatrix} x(1) & 0 & \cdots & 0 \\ x(2) & x(1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x(n-1) & x(n-2) & \cdots & x(n-N) \\ x(n) & x(n-1) & \cdots & x(n-N+1) \end{bmatrix}. \tag{2.7}$$

Next, denote the Euclidean norm of a vector $\boldsymbol{a}$ as $\|\boldsymbol{a}\| = \sqrt{\boldsymbol{a}^T\boldsymbol{a}}$, and define the diagonal matrix $\boldsymbol{\Lambda}(n)$ as

$$\boldsymbol{\Lambda}(n) = \text{diag}\,[\sqrt{\lambda^{n-1}}, \sqrt{\lambda^{n-2}}, \cdots, \sqrt{\lambda}, 1]. \tag{2.8}$$

These definitions now allow the cumulative squared error in (2.4) to be written as

$$\epsilon(n) = \|\boldsymbol{\Lambda}(n)\boldsymbol{d}(n) - \boldsymbol{\Lambda}(n)\boldsymbol{X}(n)\boldsymbol{w}(n)\|^2. \tag{2.9}$$

Since the matrix $\boldsymbol{\Lambda}(n)\boldsymbol{X}(n)$ is $n \times N$, then there exists an $n \times n$ orthogonal matrix $\boldsymbol{Q}(n)$ such that [1], [15]

$$\boldsymbol{Q}^T(n)\boldsymbol{\Lambda}(n)\boldsymbol{X}(n) = \begin{bmatrix} \boldsymbol{R}(n) \\ \boldsymbol{0} \end{bmatrix} \tag{2.10}$$

where $\boldsymbol{R}(n)$ is an $N \times N$ upper triangular matrix, sometimes called the Cholesky factor [13], [15]; and $\boldsymbol{0}$ is an $(n-N) \times N$ matrix of zeros. The pervasive use of matrix symbols $\boldsymbol{Q}$ and $\boldsymbol{R}$ is transformations such as (2.10) leads to the common name of "*QR* decomposition." Performing a similar transformation on the weighted reference vector $\boldsymbol{\Lambda}(n)\boldsymbol{d}(n)$ gives

$$\boldsymbol{Q}^T(n)\boldsymbol{\Lambda}(n)\boldsymbol{d}(n) = \begin{bmatrix} \boldsymbol{z}(n) \\ \boldsymbol{v}(n) \end{bmatrix} \tag{2.11}$$

where $\boldsymbol{z}(n)$ is an $N \times 1$ vector; and $\boldsymbol{v}(n)$ is an $(n-N) \times 1$ vector. Since $\boldsymbol{Q}(n)$ is orthogonal, then premultiplying each vector within the norm defined in (2.9) by the matrix $\boldsymbol{Q}(n)$ does not change the value of the norm, giving

$$\epsilon(n) = \|\boldsymbol{Q}(n)\boldsymbol{\Lambda}(n)\boldsymbol{d}(n) - \boldsymbol{Q}(n)\boldsymbol{\Lambda}(n)\boldsymbol{X}(n)\boldsymbol{w}(n)\|^2. \tag{2.12}$$

Substituting (2.10) and (2.11) into (2.12) then gives the desired form

$$\epsilon(n) = \left\| \begin{bmatrix} \boldsymbol{z}(n) - \boldsymbol{R}(n)\boldsymbol{w}(n) \\ \boldsymbol{v}(n) \end{bmatrix} \right\|^2. \tag{2.13}$$

It is straightforward [1], [15] to see that the norm in (2.13) is minimized if the top partition on the right of (2.13) is set equal to zero, which is equivalent to

$$\boldsymbol{R}(n)\boldsymbol{w}(n) = \boldsymbol{z}(n). \tag{2.14}$$

Equation (2.14) can then be solved by backsubstitution to find $\boldsymbol{w}(n)$.

Equation (2.14) forms the basis for direct *QR* methods for solving the RLS problem. To see this, assume that $\boldsymbol{R}(n-1)$ and $\boldsymbol{z}(n-1)$ are available from the previous iteration. If $\boldsymbol{R}(n-1)$ can be easily updated to $\boldsymbol{R}(n)$ and $\boldsymbol{z}(n-1)$ can be easily updated to $\boldsymbol{z}(n)$, then (2.14) may be obtained quite easily. It may be shown [1], [18] that such an update may be found in a straightforward manner since there exists an $(N+1) \times (N+1)$ orthogonal matrix $\boldsymbol{T}(n)$ such that

$$\boldsymbol{T}(n)\begin{bmatrix} \sqrt{\lambda}\boldsymbol{R}(n-1) \\ \boldsymbol{x}^T(n) \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}(n) \\ \boldsymbol{0}^T \end{bmatrix} \tag{2.15}$$

where $\boldsymbol{x}^T(n) = [x(n), x(n-1), \cdots, x(n-N+1)]$ is the vector incorporating the new data sample collected at time n. It can also be shown [1], [18] that the same matrix $\boldsymbol{T}(n)$ may be used to update $z(n-1)$ to $z(n)$, giving

$$\boldsymbol{T}(n)\begin{bmatrix}\sqrt{\lambda}z(n-1)\\ d(n)\end{bmatrix} = \begin{bmatrix}z(n)\\ \zeta(n)\end{bmatrix} \tag{2.16}$$

where $\zeta(n)$ is just the last component in the vector on the right-hand side of (2.16).

It is straightforward to find the matrix $\boldsymbol{T}(n)$, which zeros the last row of the matrix on the left-hand side of (2.15). One method that has received considerable investigation recently in RLS applications is Givens rotations [1], [16], largely due to its ability to be implemented in parallel and systolic structures. Backsubstitution must still be performed if the RLS filter coefficients are needed, and this impedes the parallelization of the algorithm. However, (2.14) also provides the basis of developing another approach, called the inverse QR algorithm, which allows the RLS filter computation without backsubstitutions while still retaining the stability properties of QR algorithms. This derivation is begun in the next section.

III. Filter Update Derivation

The preceding section summarized previous work in QR recursive methods. This section begins the presentation of new results by first deriving the form for the update of the RLS filter, which does not require the highly serial backsubstitution step. Note from (2.14) that the analytical solution for the RLS weight vector $\boldsymbol{w}(n)$ is given by

$$\boldsymbol{w}(n) = \boldsymbol{R}^{-1}(n)z(n) \tag{3.1}$$

It will be assumed that $\boldsymbol{R}^{-1}(n)$ and $z(n-1)$ are available from the previous iteration and that $\boldsymbol{R}^{-1}(n)$ and $z(n)$ must now be obtained. To do this, first express the orthogonal transformation matrix $\boldsymbol{T}(n)$ in (2.15) as the partitioned matrix

$$\boldsymbol{T}(n) = \begin{bmatrix}\boldsymbol{A} & \boldsymbol{b}\\ \boldsymbol{c}^T & \delta\end{bmatrix} \tag{3.2}$$

where $\boldsymbol{A}$ is $N \times N$; $\boldsymbol{b}$ and $\boldsymbol{c}$ are $N \times 1$; and δ is a scalar (the time indices have been dropped for simplicity). Then, substitute (3.2) into (2.15) and perform the partitioned multiplications. The top partition of the result gives

$$\boldsymbol{R}(n) = \sqrt{\lambda}\boldsymbol{A}\boldsymbol{R}(n-1) + \boldsymbol{b}\boldsymbol{x}^T(n). \tag{3.3}$$

Next substitute (3.2) into (2.16) and perform the partitioned multiplications. The top partition of the result gives

$$z(n) = \sqrt{\lambda}\boldsymbol{A}z(n-1) + \boldsymbol{b}d(n). \tag{3.4}$$

Substituting (3.3) and (3.4) into (3.1) then provides

$$\boldsymbol{w}(n) = [\sqrt{\lambda}\boldsymbol{A}\boldsymbol{R}(n-1) + \boldsymbol{b}\boldsymbol{x}^T(n)]^{-1} \cdot [\sqrt{\lambda}\boldsymbol{A}z(n-1) + \boldsymbol{b}d(n)]. \tag{3.5}$$

Note in (3.5) that since $\boldsymbol{R}(n-1)$, $\boldsymbol{x}(n)$, $z(n-1)$, and $d(n)$ are all known at time n, then finding the update for $\boldsymbol{w}(n)$ simply requires finding expressions for the matrix partitions $\boldsymbol{A}$ and $\boldsymbol{b}$. This is done by first recognizing that since $\boldsymbol{T}(n)$ is orthogonal, then [15]

$$\boldsymbol{T}(n)\boldsymbol{T}^T(n) = \boldsymbol{I} \tag{3.6}$$

and substituting (3.2) for $\boldsymbol{T}(n)$ in (3.6) thus gives

$$\boldsymbol{A}\boldsymbol{A}^T + \boldsymbol{b}\boldsymbol{b}^T = \boldsymbol{I} \tag{3.7a}$$

$$\boldsymbol{A}\boldsymbol{c} + \delta\boldsymbol{b} = \boldsymbol{0} \tag{3.7b}$$

$$\delta^2 + \boldsymbol{c}^T\boldsymbol{c} = 1 \tag{3.7c}$$

Equation (3.7b) immediately shows that the unknown vector $\boldsymbol{b}$ in (3.5) may be eliminated by using

$$\boldsymbol{b} = -\frac{1}{\delta}\boldsymbol{A}\boldsymbol{c}. \tag{3.8}$$

The matrix $\boldsymbol{A}$ in (3.5) may next be eliminated by substituting (3.8) into (3.5), and using the matrix inverse identity $(\boldsymbol{X}\boldsymbol{Y})^{-1} = \boldsymbol{Y}^{-1}\boldsymbol{X}^{-1}$ to simplify the result, providing

$$\boldsymbol{w}(n) = \left[\sqrt{\lambda}\boldsymbol{R}(n-1) - \frac{1}{\delta}\boldsymbol{c}\boldsymbol{x}^T(n)\right]^{-1} \cdot \left[\sqrt{\lambda}z(n-1) - \frac{1}{\delta}\boldsymbol{c}d(n)\right]. \tag{3.9}$$

The unknown vector $\boldsymbol{c}$ in (3.9) may now be found by examining the bottom partition resulting from substituting (3.2) into (2.15)

$$\sqrt{\lambda}\boldsymbol{c}^T\boldsymbol{R}(n-1) + \delta\boldsymbol{x}^T(n) = \boldsymbol{0}^T. \tag{3.10}$$

Postmultiplying both sides of (3.10) by $\boldsymbol{R}^{-1}(n-1)$ and simplifying gives

$$\boldsymbol{c} = -\frac{\delta\boldsymbol{R}^{-T}(n-1)\boldsymbol{x}(n)}{\sqrt{\lambda}} \tag{3.11}$$

where the "$-T$" notation in (3.11) signifies the transpose of the inverse. Note that since $\boldsymbol{R}^{-1}(n-1)$ is upper triangular, then $\boldsymbol{R}^{-T}(n-1)$ is lower triangular. Next, define the $N \times 1$ vector $\boldsymbol{a}(n)$ as

$$\boldsymbol{a}(n) = \frac{\boldsymbol{R}^{-T}(n-1)\boldsymbol{x}(n)}{\sqrt{\lambda}} \tag{3.12}$$

in which case (3.11) becomes

$$\boldsymbol{c} = -\delta\boldsymbol{a}(n). \tag{3.13}$$

At this point, the definition of $\boldsymbol{a}(n)$ in (3.11) may seem to be little more than a notational convenience. However, implementation of the algorithm in Sections IV and V will show that this intermediate vector $\boldsymbol{a}(n)$ provides the key to parallelization of the inverse QR approach.

Substituting (3.13) into (3.9) now gives

$$\boldsymbol{w}(n) = [\sqrt{\lambda}\boldsymbol{R}(n-1) + \boldsymbol{a}(n)\boldsymbol{x}^T(n)]^{-1} \cdot [\sqrt{\lambda}z(n-1) + \boldsymbol{a}(n)d(n)]. \tag{3.14}$$

To simplify (3.14), recall the matrix inverse lemma [1], [15], restated here: suppose a $p \times p$ matrix $\boldsymbol{M}$ may be

written as

$$\boldsymbol{M} = \boldsymbol{H} + \boldsymbol{g}\boldsymbol{h}^T \tag{3.15a}$$

where $\boldsymbol{H}$ is $p \times p$; and $\boldsymbol{g}$, $\boldsymbol{h}$ are $p \times 1$. Then the inverse $\boldsymbol{M}^{-1}$ is given by

$$\boldsymbol{M}^{-1} = \boldsymbol{H}^{-1} - \frac{\boldsymbol{H}^{-1}\boldsymbol{g}\boldsymbol{h}^T\boldsymbol{H}^{-1}}{1 + \boldsymbol{h}^T\boldsymbol{H}^{-1}\boldsymbol{g}}. \tag{3.15b}$$

Comparing (3.15a) with (3.14), it is seen that the matrix inverse on the right-hand side of (3.14) may be computed by using the substitutions

$$\boldsymbol{H} = \sqrt{\lambda}\boldsymbol{R}(n-1) \tag{3.16a}$$

$$\boldsymbol{g} = \boldsymbol{a}(n) \tag{3.16b}$$

$$\boldsymbol{h} = \boldsymbol{x}(n). \tag{3.16c}$$

Therefore, applying (3.15b) and (3.16) to (3.14), and using (3.12) in the result gives

$$\boldsymbol{w}(n) = \left[\frac{\boldsymbol{R}^{-1}(n-1)}{\sqrt{\lambda}} - \frac{\boldsymbol{R}^{-1}(n-1)\boldsymbol{a}(n)\boldsymbol{a}^T(n)}{\sqrt{\lambda}[1 + \boldsymbol{a}^T(n)\boldsymbol{a}(n)]}\right] \cdot [\sqrt{\lambda}\boldsymbol{z}(n-1) + \boldsymbol{a}(n)d(n)]. \tag{3.17}$$

Two additional definitions made at this point will be very useful in Section IV. Define the scalar $b^2(n)$ as

$$b^2(n) = 1 + \boldsymbol{a}^T(n)\boldsymbol{a}(n) \tag{3.18}$$

and define the $N \times 1$ vector $\boldsymbol{u}(n)$ as

$$\boldsymbol{u}(n) = \frac{\boldsymbol{R}^{-1}(n-1)\boldsymbol{a}(n)}{b(n)\sqrt{\lambda}} \tag{3.19}$$

where $b(n)$ is just the square root of the quantity in (3.18). Multiplying out the terms in (3.17) and using (3.18) and (3.19) in the result then provides

$$\boldsymbol{w}(n) = \boldsymbol{R}^{-1}(n-1)\boldsymbol{z}(n-1) + \frac{\boldsymbol{u}(n)}{b(n)} \cdot [d(n) - \sqrt{\lambda}\boldsymbol{a}^T(n)\boldsymbol{z}(n-1)]. \tag{3.20}$$

However, using (3.12) for $\boldsymbol{a}^T(n)$ in (3.20) gives

$$\boldsymbol{w}(n) = \boldsymbol{R}^{-1}(n-1)\boldsymbol{z}(n-1) + \frac{\boldsymbol{u}(n)}{b(n)} \cdot [d(n) - \boldsymbol{x}^T(n)\boldsymbol{R}^{-1}(n-1)\boldsymbol{z}(n-1)]. \tag{3.21}$$

But since (3.1) is valid for any time argument, it is valid for $n-1$ and thus

$$\boldsymbol{w}(n-1) = \boldsymbol{R}^{-1}(n-1)\boldsymbol{z}(n-1). \tag{3.22}$$

Substituting (3.22) into (3.21) now gives

$$\boldsymbol{w}(n) = \boldsymbol{w}(n-1) + \frac{\boldsymbol{u}(n)}{b(n)}[d(n) - \boldsymbol{x}^T(n)\boldsymbol{w}(n-1)]. \tag{3.23}$$

Finally, (2.3) shows that the quantity in brackets in (3.23) is just $e(n|n-1)$, the error resulting from predicting the reference signal at time n using the filter computed at time $n-1$. Hence, the filter update equation of (3.23) is given by the simple equation

$$\boldsymbol{w}(n) = \boldsymbol{w}(n-1) + \frac{\boldsymbol{u}(n)}{b(n)}e(n|n-1). \tag{3.24}$$

Equation (3.24) has the familiar form of the updated weight vector $\boldsymbol{w}(n)$ being given by a combination of the old weight vector $\boldsymbol{w}(n-1)$ plus a correction term proportional to a "gain" vector $\boldsymbol{u}(n)$ scaled by the prediction error $e(n|n-1)$. The inverse QR algorithm derived in the next section will show that these needed quantities $\boldsymbol{u}(n)$ and $b(n)$ may be derived entirely by rotation-based methods.

IV. Inverse QR Derivation

From Section III, it has been seen that the quantities $\boldsymbol{u}(n)$ from (3.19) and $b(n)$ from (3.18) are all that actually need to be computed in order to update the filter vector $\boldsymbol{w}(n-1)$ to $\boldsymbol{w}(n)$. However, (3.18) and (3.19) show that these needed quantities are both dependent on the $\boldsymbol{a}(n)$ vector defined in (3.12), which, in turn, is dependent upon the previous inverse Cholesky factor $\boldsymbol{R}^{-T}(n-1)$. At the next time increment $n+1$, all of these quantities will be dependent upon $\boldsymbol{R}^{-T}(n)$, and so on. Therefore, the key to developing an efficient update method for $\boldsymbol{u}(n)$ and $b(n)$ depends upon deriving an efficient update for the inverse Cholesky factor from time $n-1$ to time n.

To begin the derivation of this update, first premultiply the matrices on each side of (2.15) by the respective transpose of each side, giving

$$[\sqrt{\lambda}\boldsymbol{R}^T(n-1) \quad \boldsymbol{x}(n)]\boldsymbol{T}^T(n)\boldsymbol{T}(n)\begin{bmatrix}\sqrt{\lambda}\boldsymbol{R}(n-1)\\ \boldsymbol{x}^T(n)\end{bmatrix} = [\boldsymbol{R}^T(n) \quad \boldsymbol{0}]\begin{bmatrix}\boldsymbol{R}(n)\\ \boldsymbol{0}^T\end{bmatrix}. \tag{4.1}$$

Since $\boldsymbol{T}(n)$ is orthogonal, then $\boldsymbol{T}^T(n)\boldsymbol{T}(n) = \boldsymbol{I}$ and (4.1) may be expanded to produce

$$\boldsymbol{R}^T(n)\boldsymbol{R}(n) = \lambda\boldsymbol{R}^T(n-1)\boldsymbol{R}(n-1) + \boldsymbol{x}(n)\boldsymbol{x}^T(n). \tag{4.2}$$

Comparing (3.15a) with (4.2), it is seen that the matrix inverse lemma may again be used with the substitutions

$$\boldsymbol{M} = \boldsymbol{R}^T(n)\boldsymbol{R}(n) \tag{4.3a}$$

$$\boldsymbol{H} = \lambda\boldsymbol{R}^T(n-1)\boldsymbol{R}(n-1) \tag{4.3b}$$

$$\boldsymbol{h} = \boldsymbol{g} = \boldsymbol{x}(n). \tag{4.3c}$$

Making substitutions (4.3) in (3.15b) and using the relation $(\boldsymbol{X}\boldsymbol{Y})^{-1} = \boldsymbol{Y}^{-1}\boldsymbol{X}^{-1}$ in the results thus gives

$$\boldsymbol{R}^{-1}(n)\boldsymbol{R}^{-T}(n) = \frac{\boldsymbol{R}^{-1}(n-1)\boldsymbol{R}^{-T}(n-1)}{\lambda} - \frac{\boldsymbol{R}^{-1}(n-1)\boldsymbol{R}^{-T}(n-1)\boldsymbol{x}(n)}{\lambda} \times \left[1 + \frac{\boldsymbol{x}^T(n)\boldsymbol{R}^{-1}(n-1)\boldsymbol{R}^{-T}(n-1)\boldsymbol{x}(n)}{\lambda}\right]^{-1} \frac{\boldsymbol{x}^T(n)\boldsymbol{R}^{-1}(n-1)\boldsymbol{R}^{-T}(n-1)}{\lambda}. \tag{4.4}$$

Equation (4.4) may now be simplified using (3.12) for $\boldsymbol{a}(n)$ and (3.18) for $b(n)$, giving

$$\boldsymbol{R}^{-1}(n)\boldsymbol{R}^{-T}(n) = \frac{\boldsymbol{R}^{-1}(n-1)\boldsymbol{R}^{-T}(n-1)}{\lambda} - \frac{\boldsymbol{R}^{-1}(n-1)\boldsymbol{a}(n)\boldsymbol{a}^T(n)\boldsymbol{R}^{-T}(n-1)}{\lambda b^2(n)}. \tag{4.5}$$

Equation (4.5) may now be simplified further by substituting (3.19) for $\boldsymbol{u}(n)$, giving

$$\boldsymbol{R}^{-1}(n)\boldsymbol{R}^{-T}(n) + \boldsymbol{u}(n)\boldsymbol{u}^T(n) = \frac{\boldsymbol{R}^{-1}(n-1)\boldsymbol{R}^{-T}(n-1)}{\lambda}. \tag{4.6}$$

Equation (4.6) implies the existence of an $(N+1) \times (N+1)$ orthogonal matrix $\boldsymbol{P}(n)$ such that

$$\boldsymbol{P}(n)\begin{bmatrix} \lambda^{-1/2}\boldsymbol{R}^{-T}(n-1) \\ \boldsymbol{0}^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}^{-T}(n) \\ \boldsymbol{u}^T(n) \end{bmatrix}. \tag{4.7}$$

Equation (4.7) may be easily verified by forming the product of each side of (4.7) with its respective transpose, and verifying that (4.6) results.

However, there is a major difference between the orthogonal matrix $\boldsymbol{P}(n)$ in (4.7) and the orthogonal matrix $\boldsymbol{T}(n)$ in (2.15). Note that (2.15) suggests a straightforward way of determining the orthogonal matrix $\boldsymbol{T}(n)$ as a sequence of Givens rotations [1], [15] that sequentially zero the components of $\boldsymbol{x}^T(n)$. However, (4.7) does not necessarily suggest such a straightforward method since specific quantities [i.e., the components of $\boldsymbol{u}^T(n)$] must be introduced into the last row of the right-hand matrix. Therefore, an alternative method for determining the entries of the orthogonal matrix $\boldsymbol{P}(n)$ needs to be developed, and this is done next.

The derivation of $\boldsymbol{P}(n)$ is similar to an approach taken in [13]. Equation (4.7) suggests that $\boldsymbol{P}(n)$ may be partitioned in the following manner:

$$\boldsymbol{P}(n) = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{f} \\ \boldsymbol{g}^T & h \end{bmatrix} \tag{4.8}$$

where $\boldsymbol{E}$ is $N \times N$; $\boldsymbol{f}$ and $\boldsymbol{g}$ are $N \times 1$; and h is a scalar (the time indices have been dropped for ease of notation). Substituting (4.8) into (4.7), performing the multiplications, and equating the resulting matrix partitions then gives the following relations:

$$\frac{\boldsymbol{E}\boldsymbol{R}^{-T}(n-1)}{\sqrt{\lambda}} = \boldsymbol{R}^{-T}(n) \tag{4.9a}$$

$$\frac{\boldsymbol{g}^T\boldsymbol{R}^{-T}(n-1)}{\sqrt{\lambda}} = \boldsymbol{u}^T(n). \tag{4.9b}$$

To find the remaining relations necessary to specify the elements of $\boldsymbol{P}(n)$ recognize that since $\boldsymbol{P}(n)$ is orthogonal, then

$$\boldsymbol{P}(n)\boldsymbol{P}^T(n) = \boldsymbol{I}. \tag{4.10}$$

Substituting (4.8) for $\boldsymbol{P}(n)$ into (4.10) and performing the multiplications then gives the following three relations:

$$\boldsymbol{E}\boldsymbol{E}^T + \boldsymbol{f}\boldsymbol{f}^T = \boldsymbol{I} \tag{4.11a}$$

$$\boldsymbol{E}\boldsymbol{g} + \boldsymbol{f}h = \boldsymbol{0} \tag{4.11b}$$

$$h^2 + \boldsymbol{g}^T\boldsymbol{g} = 1. \tag{4.11c}$$

Equations (4.11b) and (4.11c) may now be manipulated to provide $\boldsymbol{g}$ and h in terms of known quantities. It will also be shown in the following work that explicit expressions for $\boldsymbol{E}$ and $\boldsymbol{f}$ are not necessary to derive the needed elements of $\boldsymbol{P}(n)$.

Proceeding in this manner, substituting (3.19) into (4.9b) shows immediately that

$$\boldsymbol{g} = \frac{\boldsymbol{a}(n)}{b(n)} \tag{4.12}$$

where $\boldsymbol{a}(n)$ has been defined in (3.12). Substituting (4.12) into (4.11c), and then simplifying using (3.18), gives

$$h = \frac{1}{b(n)} \tag{4.13}$$

The unknown vector $\boldsymbol{f}$ may be eliminated by using (4.13) and (4.12) in (4.11b), giving

$$\boldsymbol{f} = -\boldsymbol{E}\boldsymbol{a}(n) \tag{4.14}$$

Next, form the product of $\boldsymbol{P}(n)$ with the augmented vector $[\boldsymbol{a}^T(n), 1]^T$ and use the partitioning in (4.8) for $\boldsymbol{P}(n)$, which produces

$$\boldsymbol{P}(n)\begin{bmatrix} \boldsymbol{a}(n) \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{f} \\ \boldsymbol{g}^T & h \end{bmatrix}\begin{bmatrix} \boldsymbol{a}(n) \\ 1 \end{bmatrix}. \tag{4.15}$$

Finally, substituting (4.12)–(4.14) into (4.15) then provides the desired form

$$\boldsymbol{P}(n)\begin{bmatrix} \boldsymbol{a}(n) \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ b(n) \end{bmatrix}. \tag{4.16}$$

Equation (4.16) is a pivotal result. First, recall from (4.7) that $\boldsymbol{P}(n)$ is the orthogonal matrix necessary to update the inverse Cholesky factor $\boldsymbol{R}^{-T}(n-1)$ to $\boldsymbol{R}^{-T}(n)$. Equation (4.16) now shows that $\boldsymbol{P}(n)$ is also the norm-preserving rotation matrix that will zero the first N elements of the augmented vector $[\boldsymbol{a}^T(n), 1]^T$. Since $\boldsymbol{P}(n)$ is therefore a rotation matrix [15], the elements of $\boldsymbol{P}(n)$ may be determined entirely by an appropriate set of operations using only the known elements of $\boldsymbol{a}(n)$. This is specifically the reason for previously having introduced the vector $\boldsymbol{a}(n)$ in (3.12).

The preceeding derivations may now be summarized succinctly by combining (4.7) and (4.16) into the single equivalent equation

$$\boldsymbol{P}(n)\begin{bmatrix} \boldsymbol{a}(n) & \lambda^{-1/2}\boldsymbol{R}^{-T}(n-1) \\ 1 & \boldsymbol{0}^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{R}^{-T}(n) \\ b(n) & \boldsymbol{u}^T(n) \end{bmatrix}. \tag{4.17}$$

Equation (4.17) is a very concise high-level statement of the inverse *QR* algorithm. From a linear algebra viewpoint, it clearly illustrates the following steps for updating the LS weight vector without backsubstitution:

1) Compute the intermediate vector $\boldsymbol{a}(n)$ using (3.12).

2) Form the $(N + 1) \times (N \times 1)$ partitioned matrix, which has first column $[\boldsymbol{a}^T(n), 1]^T$ and remaining N columns given by $[\lambda^{-1/2}\boldsymbol{R}^{-1}(n-1)\ \boldsymbol{0}]^T$. [This is the partitioned matrix shown on the left-hand side of (4.17)].

3) Perform the sequence of matrix multiplications $\boldsymbol{P}(n)$ (i.e., rotations), which will zero the first N elements of the first column of the partitioned matrix in 2) above. This produces the matrix on the right-hand side of (4.17).

4) When this zeroing has been completed, the $(N + 1)$st row of the resulting matrix contains $b(n)$ in the first column and $\boldsymbol{u}^T(n)$ in the remaining N columns. These may be used in (3.24) to update $\boldsymbol{w}(n-1)$ to $\boldsymbol{w}(n)$. Additionally, when the zeroing has been completed, the $N \times N$ upper right-hand submatrix contains the inverse Cholesky factor needed for the next time iteration.

The preceding has demonstrated the linear algebraic structure for computing the recursive LS weight vector without backsubstitution. A closer examination of (4.17) will next demonstrate how an algorithmic implementation of the inverse *QR* method may be derived. This algorithmic description will be the basis for parallelizing the method. There are two parts to this development of the algorithm: 1) updating the angle parameters; and 2) updating the inverse Cholesky factor. These are described next.

A. Updating the Angle Parameters

This section will demonstrate how the $2N$ angle parameters $c_i(n)$ and $s_i(n)$ can be computed from only a knowledge of $\boldsymbol{a}(n)$. Consider the operation in which an $(N + 1) \times (N + 1)$ rotation matrix $\boldsymbol{P}^{(1)}(n)$ must be constructed to zero only the first element $a_1(n)$ in (4.16) while preserving the Euclidean norm of the resulting vector. Mathematically

$$\boldsymbol{P}^{(1)}(n)\begin{bmatrix} a_1(n) \\ a_2(n) \\ \vdots \\ a_N(n) \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ a_2(n) \\ \vdots \\ a_N(n) \\ b^{(1)}(n) \end{bmatrix} \tag{4.18}$$

where the definition

$$b^{(i)}(n) = \left[1 + \sum_{k=1}^{i} a_i^2(n)\right]^{1/2} \tag{4.19}$$

has been used. The superscript notation in (4.19) indicates that $b^{(i)}(n)$ is the result of i rotations. An equivalent recursive form of (4.19) that will be useful in the parallelization of the algorithm is

$$b^{(i)}(n) = \{[b^{(i-1)}(n)]^2 + a_i^2(n)\}^{1/2}, \qquad b^{(0)}(n) = 1. \tag{4.20}$$

It is straightforward from (3.18) to see that this notation implies

$$b(n) = b^{(N)}(n) = \left[1 + \sum_{k=1}^{N} a_i^2(n)\right]^{1/2}. \tag{4.21}$$

Since none of the other components $a_2(n)$ through $a_N(n)$ are changed in the rotation (4.18), it is easy to show $\boldsymbol{P}^{(1)}(n)$ must have the form

$$\boldsymbol{P}^{(1)}(n) = \begin{bmatrix} c_1(n) & 0 & \cdots & 0 & -s_1(n) \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \boldsymbol{I} & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ s_1(n) & 0 & \cdots & 0 & c_1(n) \end{bmatrix} \tag{4.22}$$

where the $\boldsymbol{I}$ notation on the right-hand side of (4.22) signifies ones along the diagonal of the matrix and zeros elsewhere. Substituting (4.22) into (4.18) next shows the following relations must be satisfied by the angle parameters $c_1(n)$ and $s_1(n)$:

$$c_1(n)a_1(n) - s_1(n) = 0 \tag{4.23a}$$

$$s_1(n)a_1(n) + c_1(n) = b^{(1)}(n). \tag{4.23b}$$

Solving the simultaneous equations in (4.23) for $c_1(n)$ and $s_1(n)$, and using (4.20) in the result, then gives

$$s_1(n) = \frac{a_1(n)}{b^{(1)}(n)} \tag{4.24a}$$

$$c_1(n) = \frac{b^{(0)}(n)}{b^{(1)}(n)}. \tag{4.24b}$$

To identify the recursive formula for the remaining angle parameter computations, consider the operation of the rotation matrix $\boldsymbol{P}^{(2)}(n)$ on the vector on the right-hand side of (4.18) to zero the second component while preserving its Euclidean norm. Using the notation in (4.20), the elements of $\boldsymbol{P}^{(2)}(n)$ must therefore satisfy

$$\boldsymbol{P}^{(2)}(n)\begin{bmatrix} 0 \\ a_2(n) \\ a_3(n) \\ \vdots \\ a_N(n) \\ b^{(1)}(n) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ a_3(n) \\ \vdots \\ a_N(n) \\ b^{(2)}(n) \end{bmatrix}. \tag{4.25}$$

Using a procedure exactly as in (4.18)–(4.20) shows that $\boldsymbol{P}^{(2)}(n)$ must have the form

$$\boldsymbol{P}^{(2)}(n) = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & c_2(n) & 0 & \cdots & 0 & -s_2(n) \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \boldsymbol{I} & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & s_2(n) & 0 & \cdots & 0 & c_2(n) \end{bmatrix} \tag{4.26}$$

where $c_2(n)$ and $s_2(n)$ are given by

$$s_2(n) = \frac{a_2(n)}{b^{(2)}(n)} \tag{4.27a}$$

$$c_2(n) = \frac{b^{(1)}(n)}{b^{(2)}(n)}. \tag{4.27b}$$

In the matrix representation of (4.16), $\boldsymbol{P}(n)$ is thus seen to be a sequence of N-ordered matrix multiplications

$$\boldsymbol{P}(n) = \boldsymbol{P}^{(N)}(n) \cdots \boldsymbol{P}^{(2)}(n)\boldsymbol{P}^{(1)}(n) \tag{4.28}$$

where, similar to (4.22) and (4.26), the ith matrix $\boldsymbol{P}^{(i)}(n)$ is a diagonal matrix, except for $c_i(n)$ at elements (i, i) and $(N+1, N+1)$, and $s_i(n)$ at element $(N+1, i)$ and $-s_i(n)$ at element $(i, N+1)$. Extending the results of (4.27) show that the angle parameters $c_i(n)$ and $s_i(n)$, $1 \le i \le N$, may be iteratively calculated by the following:

$$b^{(i)}(n) = \{[b^{(i-1)}(n)]^2 + a_i^2(n)\}^{1/2} \tag{4.29a}$$

$$s_i(n) = \frac{a_i(n)}{b^{(i)}(n)} \tag{4.29b}$$

$$c_i(n) = \frac{b^{(i-1)}(n)}{b^{(i)}(n)}. \tag{4.29c}$$

B. Updating the Inverse Cholesky Factor

The $c_i(n)$ and $s_i(n)$ rotation parameters are also used to update the inverse Cholesky factor $\boldsymbol{R}^{-T}(n-1)$ to $\boldsymbol{R}^{-T}(n)$ in preparation for the next time iteration. Denote the elements of $\boldsymbol{R}^{-T}(n-1)$ and $\boldsymbol{R}^{-T}(n)$ as $r_{ij}(n-1)$ and $r_{ij}(n)$, respectively. The manner in which the update is done may easily be seen by examining the effect of $\boldsymbol{P}^{(1)}(n)$ upon columns 2 through $N+1$ of the matrix in (4.17). Substituting (4.22) for $\boldsymbol{P}^{(1)}(n)$ into (4.17), and performing the resulting multiplications, gives

$$\boldsymbol{P}^{(1)}(n)\begin{bmatrix} \lambda^{-1/2}\boldsymbol{R}^{-T}(n-1) \\ \boldsymbol{0}^T \end{bmatrix} = \begin{bmatrix} r_{11}(n) & 0 & 0 & \cdots & 0 \\ \dfrac{r_{21}(n-1)}{\sqrt{\lambda}} & \dfrac{r_{22}(n-1)}{\sqrt{\lambda}} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \dfrac{r_{N1}(n-1)}{\sqrt{\lambda}} & \dfrac{r_{N2}(n-1)}{\sqrt{\lambda}} & \dfrac{r_{N3}(n-1)}{\sqrt{\lambda}} & \cdots & \dfrac{r_{NN}(n-1)}{\sqrt{\lambda}} \\ u_1^{(1)}(n) & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{4.30}$$

where

$$u_1^{(1)}(n) = \lambda^{-1/2}s_1(n)r_{11}(n-1) \tag{4.31}$$

and $r_{11}(n)$ is the (1, 1) element of updated inverse Cholesky factor, given by

$$r_{11}(n) = \lambda^{-1/2}c_1(n)r_{11}(n-1). \tag{4.32}$$

Note that the affect of $\boldsymbol{P}^{(1)}(n)$ has been to update the first row of the (lower triangular) inverse Cholesky factor and to change the first coefficient in $\boldsymbol{u}(n)$. Multiplying the result in (4.30) by $\boldsymbol{P}^{(2)}(n)$ from (4.26) will show the recursive pattern needed by the inverse QR algorithm

$$\boldsymbol{P}^{(2)}(n)\boldsymbol{P}^{(1)}(n)\begin{bmatrix} \lambda^{-1/2}\boldsymbol{R}^{-T}(n-1) \\ \boldsymbol{0}^T \end{bmatrix} = \begin{bmatrix} r_{11}(n) & 0 & 0 & \cdots & 0 \\ r_{21}(n) & r_{22}(n-1) & 0 & \cdots & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ \dfrac{r_{N1}(n-1)}{\sqrt{\lambda}} & \dfrac{r_{N2}(n-1)}{\sqrt{\lambda}} & \dfrac{r_{N3}(n-1)}{\sqrt{\lambda}} & \cdots & \dfrac{r_{NN}(n-1)}{\sqrt{\lambda}} \\ u_1^{(2)}(n) & u_2^{(2)}(n) & 0 & \cdots & 0 \end{bmatrix}. \tag{4.33}$$

Multiplication by $\boldsymbol{P}^{(2)}(n)$ has updated the second row of the inverse Cholesky factor and changed the value of the first two coefficients of $\boldsymbol{u}(n)$. In general, the multiplication by $\boldsymbol{P}^{(i)}(n)$ will update the ith row of the inverse Cholesky factor and change the value of the first i coefficients of $\boldsymbol{u}(n)$. This pattern suggests the following method for updating the entire inverse Cholesky factor and all coefficients of $\boldsymbol{u}(n)$:

Initialize: $u_j^{(k)}(n) = 0$, $1 \le j \le N$, $k < j$.

For $1 \le i \le N$, $1 \le j \le i$

$$r_{ij}(n) = \lambda^{-1/2} c_i(n) r_{ij}(n-1) - s_i(n) u_j^{(i-1)}(n) \tag{4.34a}$$

$$u_j^{(i)}(n) = c_i(n) u_j^{(i-1)}(n) + \lambda^{-1/2} s_i(n) r_{ij}(n-1). \tag{4.34b}$$

The desired quantities $r_{ij}(n)$ and $u_j(n) = u_j^{(N)}(n)$ are produced after the N iterations on i in (4.34).

All the necessary parameters have now been obtained to implement the inverse *QR* algorithm. A summary in "pseudocode" of the entire algorithm, containing a simple initialization procedure, appears in the Appendix. The structure of the algorithm in the Appendix is to aid in coding the algorithm in a high-level language for simulation purposes. Some observations regarding the algorithm, as shown in the Appendix, will suggest possible alternatives for an actual implementation. This is briefly discussed in the next section.

V. Simulations and Parallel Implementation

A. Simulations

In the present study, the inverse *QR* algorithm was simulated in two configurations: 1) systems identifications, and 2) linear prediction filtering. All short-term simulations exhibited the rapid convergence of RLS methods. Therefore, the emphasis of the simulations in this section is to investigate the long-term stability of the inverse *QR* method. The long-term simulations described in this section are similar to other long-term simulations performed to investigate the stability properties of other RLS methods [8], [9], [20].

A simulation was performed using the inverse *QR* filter in the systems identification mode. The data signal $x(n)$ was generated as an uncorrelated unit variance sequence, and $d(n)$ was constructed to be

$$d(n) = x(n-2) + 0.01v(n) \tag{5.1}$$

where $v(n)$ is an uncorrelated zero-mean unit-variance Gaussian sequence. An $N = 5$ length filter was used to compute the estimate

$$\hat{d}(n) = \sum_{k=1}^{5} w_k(n) x(n-k+1). \tag{5.2}$$

In this configuration, the optimal solution is therefore $w_3(n) = 1$ and all other $w_i(n) = 0$. First, a simulation of 1 000 000 time iterations was run using a value of $\lambda = 0.98$. The inverse *QR* algorithm remained stable for the entire simulation and the filter estimates had only the small variance caused by the value of forgetting a factor less than one. Next, a simulation of 1 000 000 time iterations was run using a value of $\lambda = 1$. Again, the inverse *QR* algorithm remained stable for the entire simulation and the filter estimates converged exactly to their correct values and did not vary. This is to be expected when using $\lambda = 1$.

In the linear prediction configuration the data sequence $x(n)$ was generated using the first-order autoregressive model

$$x(n) = 0.9x(n-1) + v(n) \tag{5.3}$$

where $v(n)$ was a zero-mean uncorrelated Gaussian driving sequence with variance scaled such that $\sigma_x^2 = 1$. As before, an $N = 5$ length filter was used to compute the linear prediction

$$\hat{x}(n) = \sum_{k=1}^{5} w_k(n) x(n-k). \tag{5.4}$$

Thus, the inverse *QR* filter coefficient $w_1(n)$ should converge to a mean value of $w_1(n) = 0.9$, while the other $w_i(n)$ should have mean values of zero. An extended simulation of the inverse *QR* linear prediction filter using 1 000 000 time samples of $x(n)$ in (5.3) was then performed. Using a λ value of 0.99 and a filter length of $N = 5$, the inverse *QR* algorithm converged in the mean to these values and remained stable for all 1 000 000 samples of $x(n)$. The only variance was that due to the linear prediction process. Finally, a simulation of 1 000 000 time iterations was run using a value of $\lambda = 1$. Again, the inverse *QR* linear prediction filter converged and remained stable for the entire simulation, with bounded variance due only to the linear prediction process.

While these simulation results do not rigorously prove stability of the inverse *QR* algorithm, they do suggest the algorithm has outstanding numerical properties and thus is an excellent candidate for applications requiring stable performance. These conclusions are similar to conclusions drawn in other works [8], [9], [20] on stabilizing other RLS algorithms.

B. Parallel Implementation

Finally, some observations may be made concerning the parallel implementation of the inverse *QR* algorithm, which would be very beneficial regarding real-time applications. A detailed discussion of parallelizing the algorithm in the Appendix has been performed in [21] and some of the major results will be summarized here. One recognition from the Appendix is that the update of the ith row of $\boldsymbol{R}^{-T}(n-1)$ may begin after $a_i(n)$ has been computed. For example, consider the case of $i = 1$ in the Appendix. After the new $x(n)$ has been acquired and $a_1(n)$

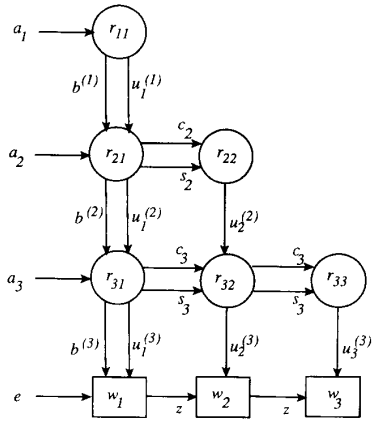


Fig. 1. Example inverse QR array for $N = 3$ showing locations of computed quantities listed in the Appendix.

computed, the remainder of the $i = 1$ loop from (7)–(11) computes $b^{(1)}(n)$, $s_1(n)$, $c_1(n)$, $r_{11}(n)$, and $u_1^{(1)}(n)$. Examination for the case $i = 2$ shows that all these $i = 1$ values are used in the calculation of the parameters having $i = 1$. After $a_2(n)$ has been computed, the remaining parameters computed at $i = 2$ are $b^{(2)}(n)$, $s_2(n)$, $c_2(n)$, $r_{21}(n)$, $r_{22}(n)$, $u_1^{(2)}(n)$, and $u_2^{(2)}(n)$.

This parameter computation and passing to the next stage can be effectively computed by the systolic array structure shown in Fig. 1 for the case of $N = 3$. The location within the array is shown for all parameters needed by the inverse QR algorithm. The round circles in Fig. 1 are "rotation" elements that compute (8)–(11) in the Appendix, and the rectangular boxes represent processors that compute the weights using (13) in the Appendix. The rotation elements in the first column also compute $b^{(i)}(n)$ using (7) and pass this parameter to the next row. Similarly, the w_1 weight processor uses the prediction error and the final $b^{(N)}(n)$ to compute the value $z(n)$ in (12), which is common to all weight processors. Omitted from Fig. 1 are the inner product arrays for computing the $a_i(n)$ and $e(n|n-1)$. Details on these arrays may be found in [21].

The computation of the various parameters in the array of Fig. 1 propagate according to the wavefront shown in the series of Figs. 2–7. The basic structure of the array in Fig. 1 is reproduced, but only the parameters directly involved in the computations at a specific clock cycle are shown. The clock cycles are referenced to the computation of $a_1(n)$. For example, suppose $a_1(n)$ has been computed during clock cycle 0. Then one clock cycle later (at clock cycle 1), $a_1(n)$ is present on the input line to the (1, 1) rotation processor (denoted by heavy black arrow). Using this input, this processor performs the computations (7)–(11) in the Appendix. (It should be mentioned that the processors may require a number of "subcycles" to compute their various parameters.) The shading of the (1, 1) processor signifies that it is active during clock cycle 1.

The heavy black arrows in Fig. 3 next show that $a_2(n)$ is now available during the next clock cycle 2 (heavy black arrow), as are the $b^{(1)}(n)$ and $u_1^{(1)}(n)$ outputs from the (1, 1) processor. These have been passed from the (1, 1) processor and are now used by the (2, 1) processor to perform the computations for $i = 2, j = 1$ in (7)–(11) of the Appendix. (It should be mentioned that the processors may require a number of "subcycles" to compute their various parameters.) The shading of the (1, 1) processor signifies that it is active during clock cycle 1.

The heavy black arrows in Fig. 3 next show that $a_2(n)$ is now available during the next clock cycle 2 (heavy black arrow), as are the $b^{(1)}(n)$ and $u_1^{(1)}(n)$ outputs from the (1,

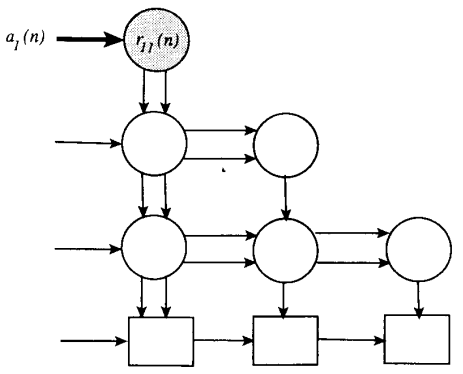


Fig. 2. Computations at clock cycle 1.

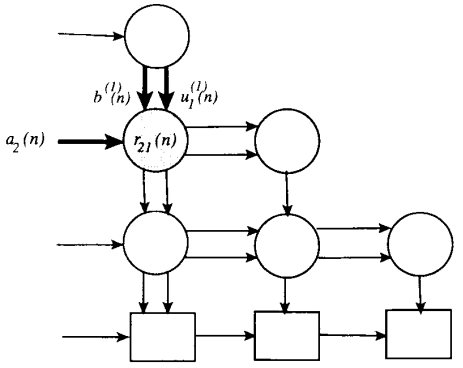


Fig. 3. Computations at clock cycle 2.

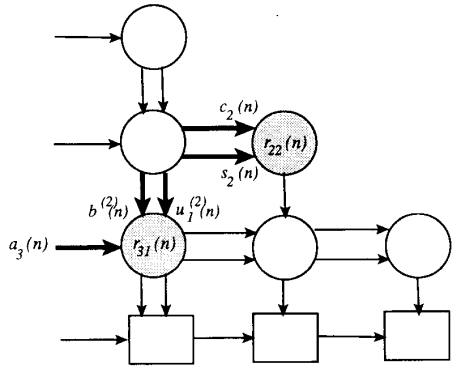


Fig. 4. Computations at clock cycle 3.

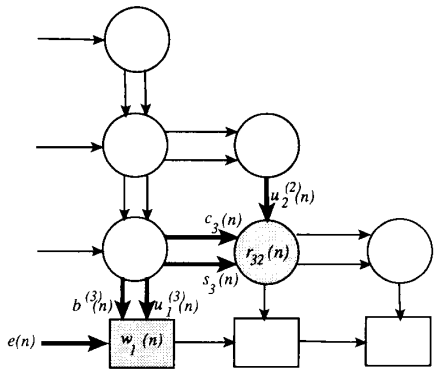


Fig. 5. Computations at clock cycle 4.

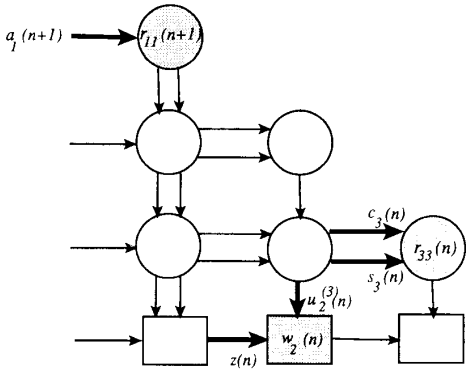


Fig. 6. Computations at clock cycle 5.

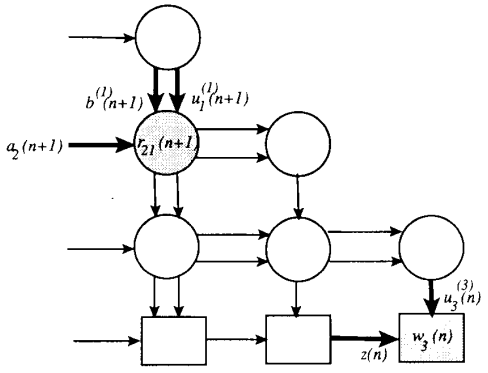


Fig. 7. Computations at clock cycle 6.

1) processor. These have been passed from the (1, 1) processor and are now used by the (2, 1) processor to perform the computations for $i = 2, j = 1$ in (7)–(11) of the Appendix. The light arrow on the a_1 input signifies no new computation is available on that input. In a similar manner, Figs. 4–7 show the computed inputs and occupied processors for clock cycles 3–6 and demonstrate the computational wavefronts as they propagate through the array. Important events to note are the first weight update (clock cycle 4 in Fig. 5) and computation of $a_1(n + 1)$ (clock cycle 5 in Fig. 6). After initialization, the resulting throughput for the array is thus seen to be N filter coefficients every $N + 1$ clock cycles.

Recently, a systolic array structure that may be fully parallelized has been derived [19]. While offering increased performance above the N filter coefficients per $N + 1$ cycles performance of the inverse QR array presented here, the computational elements of the array in [19] are also somewhat more complex than those in the inverse QR array. The storage structures needed in the rotation elements of the inverse QR array are simple registers, thus providing VLSI simplicity. No FIFO queues, as required by the elements in the systolic array of [19], are needed. The presence of the queues in [19] is necessitated by the bidirectional flow and data reflection necessary to achieve fully parallel performance. In this regard, the inverse QR array possesses somewhat of a tradeoff: reduced performance versus reductions in VLSI area, storage, and timing requirements. The selection of one structure over the other will be dependent upon the application and implementation constraints.

VI. Summary

This paper has derived a new highly stable RLS algorithm for adaptively updating the LS weight vector. The method, called the inverse QR algorithm, avoids the highly serial backsubstitution step necessary in many previous direct QR methods for finding the LS weight vector. Simulations were presented that strongly suggest the method preserves the rapid initial convergence of RLS, as well as maintaining the long-term stability properties of orthogonal rotation methods. Finally, an outline of the parallel implementation was considered, and a comparison with an alternative implementation was presented.

Appendix
Summary of Inverse QR Algorithm

INITIALIZE:

$$\boldsymbol{w}(0) = \mathbf{0},\ \boldsymbol{x}(0) = \mathbf{0},\ \boldsymbol{R}^{-T}(0) = \delta \boldsymbol{I}(\delta \gg 1) \tag{1}$$

For $n = 1$ **to** *nfinal* **do:**

$$\boldsymbol{x}^T(n) = [x(n), x(n-1), \cdots, x(n-N+1)] \tag{2}$$

$$e(n|n-1) = d(n) - \boldsymbol{w}^T(n-1)\boldsymbol{x}(n) \tag{3}$$

$$u_j^{(k)}(n) = 0, \qquad 1 \le j \le N, \qquad k < j. \tag{4}$$

$$b^{(0)}(n) = 1 \tag{5}$$

For $i = 1$ **to** N **do:**

$$a_i(n) = \lambda^{-1/2} \sum_{j=1}^{i} r_{ij}(n-1)x(n-j+1) \tag{6}$$

$$b^{(i)}(n) = \sqrt{[b^{(i-1)}(n)]^2 + a_i^2(n)} \tag{7}$$

$$s_i(n) = a_i(n)/b^{(i)}(n) \tag{8}$$

$$c_i(n) = b^{(i-1)}(n)/b^{(i)}(n) \tag{9}$$

For $j = 1$ to i do:

$$r_{ij}(n) = \lambda^{-1/2} c_i(n) r_{ij}(n-1) - s_i(n) u_j^{(i-1)}(n) \tag{10}$$

$$u_j^{(i)}(n) = c_i(n) u_j^{(i-1)}(n) + \lambda^{-1/2} s_i(n) r_{ij}(n-1) \tag{11}$$

End j loop

End i loop

$$z(n) = e(n|n-1)/b^{(N)}(n) \tag{12}$$

For $k = 1$ to N do:

$$w_k(n) = w_k(n-1) + z(n) u_k^{(N)}(n) \tag{13}$$

End k loop

End n loop

ACKNOWLEDGMENT

The authors gratefully acknowledge the many valuable suggestions made by Prof. R. J. Plemmons of Wake Forest University.

REFERENCES

[1] S. Haykin, *Adaptive Filter Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1986.

[2] S. T. Alexander, *Adaptive Signal Processing*. New York: Springer-Verlag, 1986.

[3] G. Carayannis, D. G. Manolakis, and N. Kalouptsidis, "A fast sequential algorithm for least-squares filtering and prediction," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-31, pp. 1394–1402, Dec. 1983.

[4] D. D. Falconer and L. Ljung, "Application of fast Kalman estimation to adaptive equalization," *IEEE Trans. Commun.*, vol. COM-26, pp. 1439–1446, Oct. 1978.

[5] J. M. Cioffi and T. Kailath, "Fast RLS transversal filters for adaptive filtering," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 304–337, June 1984.

[6] J. M. Cioffi, "Limited precision effects for adaptive filtering," *IEEE Trans. Circuits Syst.*, vol. CAS-34, pp. 821–833, July 1987.

[7] P. Fabre and C. Gueguen, "Improvement of fast recursive least squares algorithms via normalization: a comparative study," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 296–308, Apr. 1986.

[8] J. L. Botto and G. V. Moustakides, "Stabilizing the fast Kalman algorithms," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 1342–1348, Sept. 1989.

[9] D. T. M. Slock and T. Kailath, "Numerically stable fast transversal filters for recursive least squares adaptive filtering," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 39, pp. 92–114, Jan. 1991.

[10] J. M. Cioffi, "The fast adaptive ROTOR's RLS algorithm," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp. 631–653, Apr. 1990.

[11] P. A. Regalia and M. G. Bellanger, "On the duality between fast *QR* methods and lattice methods in least squares adaptive filtering," *IEEE Trans. Signal Processing*, vol. 39, pp. 879–891, Apr. 1991.

[12] H. Leung and S. Haykin, "Stability of recursive QRD-LS algorithms using finite precision systolic array implementation," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 760–763, May 1989.

[13] C. T. Pan and R. J. Plemmons, "Least squares modifications with inverse factorizations: parallel implications," *J. Comput., Appl. Math.*, vol. 27, pp. 109–127, 1989.

[14] C. M. Rader and A. O. Steinhardt, "Hyperbolic Householder transformations," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 1589–1602, Dec. 1986.

[15] G. H. Golub and C. Van Loan, *Matrix Computations*. Baltimore: Johns Hopkins Press, 1983.

[16] W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic arrays," *Proc. SPIE, Int. Soc. Opt. Eng.*, vol. 298, pp. 19–26, 1981.

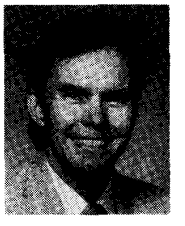
[17] S. Y. Kung, *VLSI Array Processor*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[18] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders, "Methods for modifying matrix factorizations," *Math Comput.*, vol. 28, pp. 505–535, 1974.

[19] A. P. Varvisiotis and S. Theodoridis, "A pipelined structure for *QR* adaptive LS system identification," *IEEE Trans. Signal Processing*, vol. 39, no. 8, pp. 1920–1923, Aug. 1991.

[20] G. E. Bottomley and S. T. Alexander, "A novel approach for stabilizing recursive least squares filters," *IEEE Trans. Signal Processing*, vol. 39, no. 8, pp. 1770–1779, Aug. 1991.

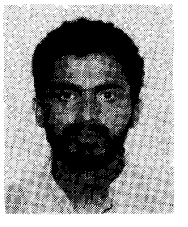
[21] A. L. Ghirnikar, S. T. Alexander, and R. J. Plemmons, "A parallel implementation of the inverse *QR* adaptive filter," *Comput. Elec. Eng.*, vol. 18, no. 3/4, pp. 291–300, 1992.



S. Thomas Alexander (S'74–A'77–M'82–SM'85) was born in Jackson, TN. He received the B.S. degree in physics from Tennessee Technological University, the M.S. degree from the University of Tennessee, and the Ph.D. degree in electrical engineering from North Carolina State University in 1971, 1975, and 1982, respectively.

From 1976 to 1978 he was with the Naval Ocean Systems Center, San Diego, CA, and since 1982 he has been with the Department of Electrical and Computer Engineering, North Carolina State University, where he is currently an Associate Professor.

Dr. Alexander served as an Associate Editor for the IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING from 1986 to 1988 and as the Editor-in-Chief of the IEEE ASSP MAGAZINE from 1987 to 1989. He is a past member of the Spectrum Estimation and Modeling Technical Committee of the SP Society and is currently a member of the Digital Signal Processing Technical Committee. In 1986, he received a Presidential Young Investigator Award from the National Science Foundation in the area of adaptive filtering. He is the author of the text *Adaptive Signal Processing*. In 1989 he received the Outstanding Teacher Award in the Electrical and Computer Engineering Department and in 1990 was elected to the Academy of Outstanding Teachers at North Carolina State University.



Avinash L. Ghirnikar (S'83–M'89) was born in Durgapur, India, on August 25, 1963. He received the B.S. degree in electrical engineering from the Indian Institute of Technology, Bombay, in 1985 and the M.S. and Ph.D. degrees in electrical and computer engineering, both from North Carolina State University, in 1985 and 1990, respectively.

Since November 1990, he has been with ViTel Communications Corporation in Santa Clara, CA, developing signal processing algorithms for multimedia applications. His research interests include adaptive signal processing algorithms, their properties in finite precision, and their real-time implementations for communications applications.

Dr. Ghirnikar is a member of the IEEE Signal Processing and Communication Societies.