# Array-Based QR-RLS Multichannel Lattice Filtering

João Gomes, *Member, IEEE*, and Victor A. N. Barroso, *Senior Member, IEEE*

*Abstract*—An array-based algorithm for multichannel lattice filtering is proposed. The filter is formed by a set of units that are adapted locally and concurrently using recursions that closely match those for single-channel lattice filters. The design, based on a known modular decomposition approach, allows for unequal filter lengths to be specified for different input channels. Individual units are updated using a square-root recursive least-squares (RLS) algorithm in array form that relies mainly on Givens rotations and exhibits highly favorable numerical behavior and a regular structure that is appealing from a hardware implementation perspective. Iterative implementations of Givens rotations using the Newton method and CORDIC processors are examined in the context of fixed-point implementations. A procedure based on three CORDIC steps is proposed to handle complex data that arise in several applications of multichannel filtering. Algorithm initialization issues are also addressed. The array algorithm is compared in simulation with plain RLS, QR-RLS, and two related multichannel lattice algorithms. Its performance is shown to be comparable to that of other QR-decomposition-based algorithms under fixed-point arithmetic. In particular, it retains desirable graceful degradation properties as the numerical precision decreases.

*Index Terms*—Adaptive filtering, lattice algorithms, multichannel filtering, recursive least-squares (RLS) estimation.

## I. INTRODUCTION

**A**DAPTIVE lattice algorithms have been used in channel equalization, system identification, filter banks, speech analysis, and several other filtering applications. Their practical relevance stems from the modularity, simplicity, robust numerical behavior, and linear increase in computational complexity with filter order, making them particularly well suited for hardware implementation. While several adaptation criteria for lattice filters have been published, recursive least-squares (RLS) remains one of the most popular and is adopted in this work as well [1].

In [2], a key correspondence was established between RLS update expressions and Kalman filter equations for some very simple dynamic systems. This provided a rich framework whereby the myriad single-channel RLS lattice and transversal algorithms were shown to be specific instances of known Kalman filter equations in various forms. More importantly, it allowed the extensive literature on Kalman filtering to be directly applied in RLS problems, suggesting novel algorithms with improved performance.

The main objective of this paper is to extend the correspondence with Kalman filtering to the multichannel lattice case, where a single discrete-time sequence is estimated from samples observed over a period of time in a set of parallel channels. Potential applications abound in array processing, communications, acoustic echo cancellation, geophysics and medicine, among others. Complex data arise naturally in several of these problems, and will be assumed henceforth. Parts of this work addressing the derivation of the filter structure and some aspects of the adaptation algorithm can be found in [3] and [4].

Early works on multichannel lattice filtering extended known single-channel algorithms by considering multichannel linear prediction, but some of the required matrix operations are difficult to compute in hardware or low-precision processors and do not lend themselves to parallel implementation. Inversions and other cumbersome matrix operations may be circumvented by QR-type algorithms that directly operate on the data matrix, rather than the sample autocorrelation matrix [5]. The derivation of QR-type algorithms for solving least-squares (LS) problems using Givens rotations was examined in [6] in a general setting; among several algorithms, [6] describes the equivalent of a single-channel lattice filter in array form. Modular decompositions were proposed as an alternative way of avoiding the matrix formulation of multichannel lattice (or transversal) filtering altogether by considering a set of interrelated single-channel problems [7]–[10]. Similar decompositions have been used in transversal filtering as well [11].

Starting from the modular decomposition approach of [9], order-update and time-update recursions based on forward and backward linear prediction are derived here for each of the resulting LS subproblems paralleling the single-channel case. Having established the properties of these subproblems, how they map into individual processing units, and how single-channel lattice recursions should be adapted to them, one can readily derive simple dynamic systems whose Kalman filter equations are equivalent to the basic update expressions. In line with general results in [12] regarding the structure of lattice filters, it was found that several theoretically equivalent formulations could be chosen. A square-root array algorithm based on angle-normalized errors is proposed here due to the anticipated numerical robustness stemming from the stability of Givens rotations and the compressed dynamic range of its internal variables. Explicitly examining the connection of LS subproblems with single-channel lattice theory also clarifies the secondary issue, not addressed in [9], of appropriately initializing the algorithm's variables based on regularization. The same modular decomposition was used in [13] to propose a power-normalized multichannel lattice algorithm, whose performance was not illustrated.

Multichannel QR lattice algorithms for equal channel orders, using both conventional and square-root-free Givens rotations, were presented in [5]. That work was further developed in [14] by removing redundant operations, which also improved the numerical stability. An array-type algorithm of this kind based on multichannel linear prediction for equal channel orders was also given in [15]. In [16] the multichannel QR lattice of [14] was extended to unequal channel orders by combining linear prediction with so-called transition stages that decorrelate some of the input channels so that they can be incorporated at various points of the lattice cascade. The approach differs from modular decomposition, but the resulting filter structures are strikingly similar. When expressed in QR form the algorithm also resembles the one proposed here and, as discussed in Section III-B, will usually require fewer operations. Its structure is less regular, however, which is a relevant factor for hardware implementations. Yet another structurally similar multichannel filter was derived in [17] by representing the basic QR-RLS algorithm as a hierarchical signal flow graph and then applying a sequence of high-level block manipulations in graphical form. In spite of relatively minor differences, all multichannel algorithms mentioned above have $O(ML^2)$ complexity for $L$ channels and $M$ filter coefficients per channel.

More recent results on multichannel QR-based filtering have emphasized the QR-RLS algorithm as a starting point, and then consider block and sequential channel decompositions to obtain fast algorithms for time-series data [18], [19]. Among several variants, this approach leads to order-recursive forms whose complexity is similar to that of previously-mentioned algorithms, but their structure is less regular.

In the proposed algorithm the actual mapping from prearray to postarray at each time step in scalar lattice units is performed by a complex Givens rotation, which in its basic form requires evaluating an inverse square root [20]. Square-root-free (fast) Givens rotations have been applied in lattice filtering [5], but they require formulating a modified lattice update with increased numerical range that somewhat detracts from the original appeal of array algorithms. In [6] the derivation of Givens-based algorithms with and without square-roots is examined, and it is shown that a particular lattice algorithm using fast rotations is actually a variant of the unnormalized *a priori* lattice. Two alternative implementations of the standard Givens rotation suitable for fixed-point arithmetic are considered in this work and compared by simulation. One of them iteratively computes the inverse square root using the Newton method. The simplicity of the iteration and its fast convergence make it particularly appropriate for programmable processors. The other alternative uses CORDIC processors and is directed towards hardware implementations. In the latter, complex data are handled by splitting a Givens rotation into three more elementary operations, each parameterized by a single real angle, that perform phase rotation, real Givens rotation, and phase counter-rotation. This approach is a modified version of the two-step technique developed in [21] for QR-RLS filtering using systolic arrays.

This paper is organized as follows. Section II states the multichannel LS filtering problem, reviews the modular decomposition approach, the global structure of the multichannel lattice filter, and derives the basic order and time recursions for individual units in terms of *a posteriori* errors and angle-normalized errors. Section III establishes the dynamical systems providing the link with Kalman filtering, from which multichannel lattice equations in array form are derived. Section IV describes the approaches for performing complex Givens rotations based on the Newton algorithm and three-step CORDIC processors. Section V examines the impact of the Newton and CORDIC approaches in fixed-point arithmetic and presents simulation results comparing the performance of the proposed algorithm with other multichannel filters. Finally, Section VI summarizes the main results and draws some conclusions.

*Notation:* Throughout the paper, vectors and matrices are represented by lowercase boldface and uppercase boldface letters, respectively. The notations $(\cdot)^T$, $(\cdot)^*$ and $(\cdot)^+$ stand for transpose, complex conjugate transpose (Hermitian) and Moore-Penrose pseudoinverse, respectively. The inner product of two complex vectors is denoted by $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{y}^* \mathbf{x}$. Modulo-$L$ indexing (in the range $1, \ldots L$ rather than $0, \ldots L-1$ as usual), is denoted by $(\cdot)_L$.

## II. MULTICHANNEL LATTICE STRUCTURE

In the multichannel setup $L$ input channels convey discrete-time signals $u^{(i)}$, $1 \leq i \leq L$, that are observed over a period of time and linearly combined to approximate a desired output (reference signal) $d$ at time $n$ as $\hat{d}(n) = \mathbf{w}^* \mathbf{u}(n)$, where the input sample vector $\mathbf{u}(n)$ is given by

$$\mathbf{u}(n) = \begin{bmatrix} \mathbf{u}^{(1)}(n) \\ \cdots \\ \mathbf{u}^{(L)}(n) \end{bmatrix}, \quad \mathbf{u}^{(i)}(n) = \begin{bmatrix} u^{(i)}(n) \\ \cdots \\ u^{(i)}(n - m_i + 1) \end{bmatrix} \quad (1)$$

and the filter order $m_i$ need not be the same for all channels. The optimal coefficient vector $\mathbf{w}$ minimizes the exponentially weighted LS cost function

$$J(n) = \sum_{k=0}^{n} \lambda^{n-k} \left| d(k) - \hat{d}(k) \right|^2 = \|\mathbf{d}(n) - \mathbf{U}(n)\mathbf{w}\|^2 \quad (2)$$

with forgetting factor $0 < \lambda \leq 1$,

$$\mathbf{d}(n) = \boldsymbol{\Lambda}^{\frac{1}{2}}(n) \begin{bmatrix} d^*(0) \\ \cdots \\ d^*(n) \end{bmatrix}, \quad \mathbf{U}(n) = \boldsymbol{\Lambda}^{\frac{1}{2}}(n) \begin{bmatrix} \mathbf{u}^*(0) \\ \cdots \\ \mathbf{u}^*(n) \end{bmatrix} \quad (3)$$

and $\boldsymbol{\Lambda}(n) = \mathrm{diag}(\lambda^n, \ldots \lambda^0)$. The family of RLS algorithms recursively updates the coefficient vector $\mathbf{w}$ to avoid explicitly evaluating at each time step the optimal solution $\mathbf{w} = \mathbf{U}^+(n)\mathbf{d}(n)$ which projects $\mathbf{d}$ onto the column space of $\mathbf{U}$ [22].

### A. Modular Decomposition

The lattice filter is a cascade of blocks, each relying on forward and backward linear prediction to progressively increase the filter order from 0 to a desired value at the end of the cascade [1], [22]. In the scalar case unit $m$ in the lattice addresses the LS problem $J_m(n) = \|\mathbf{d}(n) - \mathbf{U}_m(n)\mathbf{w}_m\|^2$ for input vectors containing the $m$ most recent samples at each time instant. Order recursions are derived by partitioning the input matrix $\mathbf{U}_m$ in
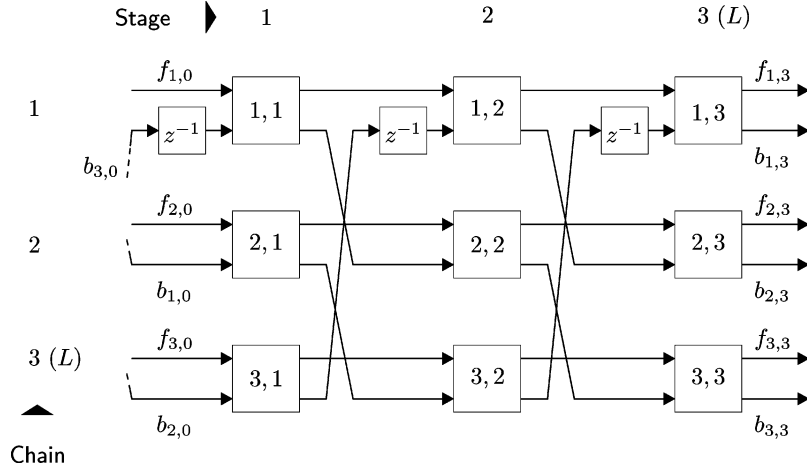
Fig. 1.   Structure of a multichannel lattice block under modular decomposition for $L = 3$.

different ways to separate its columns containing the most recent or oldest samples. The desired LS projection of $\mathbf{d}$ is then expressed in terms of the one performed by unit $m - 1$, as well as forward and backward linear prediction residuals of the leftmost/rightmost columns of the input matrix [1].

In the modular decomposition approach of [9], [10] a sequence of $L$ parallel *chains* of scalar units interact over $L$ *stages*. In each stage, the prediction order is increased by 1 in one of the input channels in such a way that after $L$ stages the overall filter order is increased in all channels. Fig. 1 depicts the structure of one of these multichannel lattice blocks for $L = 3$, whose internal connections will be described in the sequel.

Consider a given lattice block and let $\mathbf{u}_{i,0}$ denote the input vector corresponding to the LS problem solved at the output of the previous lattice block in the $i$-th chain. The actual number of samples in any of the $L$ input channels that make up $\mathbf{u}_{i,0}$ is unimportant for deriving the lattice recursions, and with a minor abuse of notation will still be denoted by $m_1, \ldots m_L$ as in (1). This vector is assumed to be updated up to time $n$ in channels $1, \ldots i$, but only up to time $n - 1$ in the remaining channels, *viz.*

$$\mathbf{u}_{i,0}(n) = \left[ \mathbf{u}^{(1)T}(n) \ldots \mathbf{u}^{(i)T}(n) \mathbf{u}^{(i+1)T}(n-1) \ldots \right.$$
$$\left. \mathbf{u}^{(L)T}(n-1) \right]^T. \quad (4)$$

Unit $(i+1, 1)$ in the first stage of the current lattice block solves a LS problem whose input vector $\mathbf{u}_{i+1,1}(n)$ is obtained from $\mathbf{u}_{i,0}(n)$ by incorporating the most recent sample $u^{(i+1)}(n)$, which increases the order by 1 in channel $i + 1$. The same holds in general when going from $\mathbf{u}_{i,l-1}(n)$ to $\mathbf{u}_{i+1,l}(n)$ for any stage $l$ of the lattice block. Chain indices should be interpreted modulo-$L$ in the range $1, \ldots L$, and when wrap-around occurs the time reference increases to $n + 1$. Fig. 2 depicts this time/order update process for equal orders at the input of stage 1.

### B. Order Update

As in the scalar case, update recursions for cell $(i, l)$ in Fig. 1 are based on forward and backward linear prediction of selected elements of $\mathbf{u}_{i,l}(n)$ from $\mathbf{u}_{i-1,l-1}(n)$ and $\mathbf{u}_{i,l-1}(n)$ (Fig. 3).
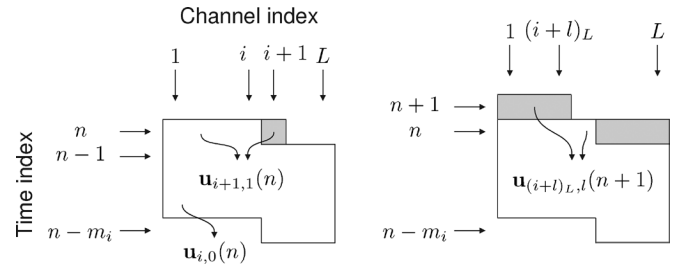


Fig. 2.   Time/order update of the input data vector.

Paralleling (2), cell $(i, l)$ addresses the projection of $\mathbf{d}(n)$, denoted by $\hat{\tilde{\mathbf{d}}}_{i,l}(n)$, onto the column space of

$$\mathbf{U}_{i,l}(n) = \mathbf{\Lambda}^{\frac{1}{2}}(n) \left[ \mathbf{u}_{i,l}(0) \quad \ldots \quad \mathbf{u}_{i,l}(n) \right]^*. \quad (5)$$

The "forward predicted" and "backward predicted" elements highlighted in Fig. 3 may be singled-out from $\mathbf{u}_{i,l}(n)$ as

$$\mathbf{S}_{i,l} \mathbf{u}_{i,l}(n) = \begin{bmatrix} u^{(i)}(n) \\ \mathbf{u}_{i-1,l-1}(n) \end{bmatrix} \quad (6)$$

$$\mathbf{Q}_{i,l} \mathbf{u}_{i,l}(n) = \begin{bmatrix} \mathbf{u}_{i,l-1}(n) \\ u^{(j)}(n - m_j - \tau_{i-l}) \end{bmatrix} \quad (7)$$

where $\mathbf{S}_{i,l}$, $\mathbf{Q}_{i,l}$ are permutation matrices, $j = (i + 1 - l)_L$ and $\tau_{i-l} = 1$ if $i - l < 0$ and 0 otherwise. Then, known geometric arguments from the scalar case [1] are readily extended to this multichannel setting to conclude that the projection $\hat{\mathbf{d}}_{i,l}(n)$ may be order updated by considering auxiliary forward and backward prediction error vectors

$$\hat{\mathbf{d}}_{i,l}(n) = \hat{\mathbf{d}}_{i,l-1}(n) + \kappa_{i,l}(n) \mathbf{b}_{i,l-1}(n) \quad (8)$$

$$\mathbf{b}_{i,l}(n) = \mathbf{b}_{i-1,l-1}(n) + \kappa_{i,l}^b(n) \mathbf{f}_{i,l-1}(n) \quad (9)$$

$$\mathbf{f}_{i,l}(n) = \mathbf{f}_{i,l-1}(n) + \kappa_{i,l}^f(n) \mathbf{b}_{i-1,l-1}(n). \quad (10)$$

In (8)–(10), $\mathbf{b}_{i,l-1}(n)$ is the vector of conjugated residuals for weighted LS backward prediction of $u^{(j)}(k - m_j - \tau_{i-l})$ from $\mathbf{u}_{i,l-1}(k)$ for $0 \leq k \leq n$. Similarly, $\mathbf{f}_{i,l-1}(n)$ is built from conjugated forward prediction residuals of $u^{(i)}(k)$ given $\mathbf{u}_{i-1,l-1}(k)$. The scalars $\kappa_{i,l}$, $\kappa_{i,l}^f$ and $\kappa_{i,l}^b$ are the reflection
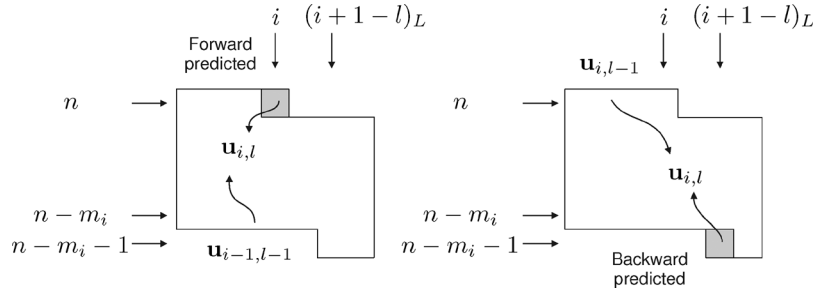
Fig. 3. Forward/backward linear prediction in stage $l > 1$.

coefficients. The pattern of interconnections between the internal units of the lattice block in Fig. 1 is implied by (9), (10). Note that the usual delay at the backward error input is only present in the units of chain $i = 1$, where the data vector $\mathbf{u}_{i-1,l-1}(n) = \mathbf{u}_{L,l-1}(n-1)$ only contains samples up to time $n - 1$. Again paralleling the single-channel lattice, only the last line of (9), (10) will be explicitly needed for cell $(i, l)$ to recursively update its internal parameters over time.

*1) Ladder Filter:* A lattice filter transforms a correlated input data sequence into a new sequence of uncorrelated, in the LS sense, backward prediction errors. These enter a ladder section, providing an orthogonal signal basis that simplifies the actual filtering process.

A single $L \times L$ multichannel lattice block increases the order of the (implicit) LS estimation problem by one sample on $L$ channels, so the unit should output a total of $L$ uncorrelated contributions to an $L$-stage ladder block. In principle, the $L$ backward errors in any of the chains could be used for this purpose, as they form an uncorrelated set. An obvious choice is $b_{L,l-1}$, as the associated data vectors $\mathbf{u}_{L,l-1}$ use the same time reference for the most recent sample in all channels (refer to Figs. 2 or 3). Stage $l$ of the ladder block thus minimizes the analog of (2) for data matrix $\mathbf{U}_{L,l}(n)$ and, from (8), the recursive expression for the *a posteriori* output error vector $\mathbf{e}_l(n) \triangleq \mathbf{d}(n) - \hat{\mathbf{d}}_{L,l}(n)$ to be propagated is

$$\mathbf{e}_l(n) = \mathbf{e}_{l-1}(n) - \kappa_l(n)\mathbf{b}_{L,l-1}(n). \tag{11}$$

### C. Merging Channels

When several $L$-channel lattice blocks are cascaded, all channels are constrained to share the same equivalent filter order. This section discusses how different orders can be specified by cascading lattice blocks with increasing dimension.

Channel indices are assumed to be sorted so that the required filter orders satisfy $m_1 \geq m_2 \ldots \geq m_L$. The filter will have a total of $m_1$ cascaded lattice blocks with increasing size, as some of the channels are only effectively incorporated later in the processing chain. Channel $i$ enters the cascade $m_i - 1$ blocks before the final one, so that the input signal $u^{(i)}$ contributes to a total of $m_i$ blocks as intended. In block $s \geq 1$ the lattice dimension $L_s$ equals the number of channels that satisfy $m_i \geq m_1 - s + 1$. Fig. 4 illustrates the filter structure for orders $m_1 = 4$, $m_2 = 3$, $m_3 = 1$, where, similarly to [9], $L_s \times L_s$ lattice blocks and $1 \times L_s$ ladder sections are denoted by $\mathcal{L}[L_s]$ and $\mathcal{D}[L_s]$, respectively. Each ladder section is fed a scalar prediction error and
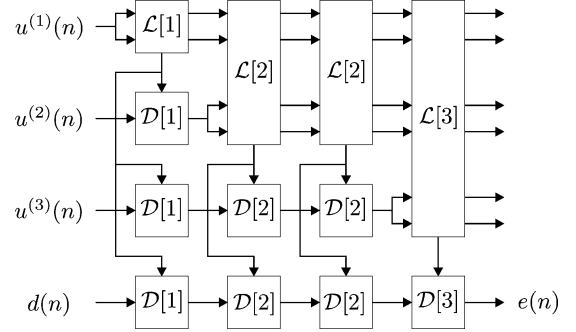


Fig. 4. Multichannel lattice filter for channel orders $m_1 = 4$, $m_2 = 3$, $m_3 = 1$.
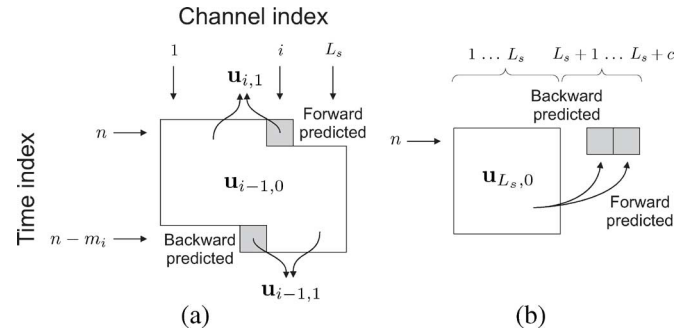


Fig. 5. Graphical depiction of prediction errors $f_{i,0}, b_{i-1,0}$ needed in the first stage $(l = 1)$ of lattice block $s + 1$ that incorporates $c$ new channels. (a) Chain $1 < i \leq L_s$. (b) Chain $L_s + 1 < i \leq L_s + c$.

the $L_s$ intermediate backward residuals $b_{L_s,l-1}(n), 1 \leq l \leq L_s$ from its associated lattice block.

Suppose that a total of $c$ new channels are incorporated into the lattice section after block $s$, with indices $L_s + 1, \ldots L_s + c$. This implies that the underlying data vectors $\mathbf{u}_{i,0}$ at the input of block $s + 1$ contain no samples for channels $L_s + 1, \ldots L_s + c$. It is then straightforward to verify that, for $L_s + 1 \leq i \leq L_s + c$, the residual $f_{i,0}(n) = b_{i,0}(n)$ is just the prediction error of $u^{(i)}(n)$ given the fully updated data vector at the output of the previous lattice block, $\mathbf{u}_{L_s,0}(n)$, regardless of the remaining newly incorporated channels.

Fig. 5(a) depicts the samples involved in the computation of forward/backward residuals $f_{i,0}, b_{i-1,0}$ that are needed for order update in unit $(i, 1)$, $1 < i \leq L_s$, in the first stage of a lattice block. Fig. 5(b) shows a similar diagram for one of the units in a newly incorporated chain. The figure highlights the fact that only simple prediction errors of $u^{(i)}(n)$, $i > L_s$, given $\mathbf{u}_{L_s,0}(n)$ are needed at the input of block $s+1$, in addition to the

$L_s$ forward/backward residuals propagated from block $s$. But an entirely equivalent error based on $\mathbf{u}_{L_s,0}$ is computed by the ladder section for reference signal $d$ (Section II-B-1), implying that identical ladder blocks should process each channel that will be incorporated downstream. When $d$ is replaced by $u^{(i)}$, $i > L_s$, the last line of (11) still provides the order recursion that is needed to gradually propagate the prediction error.

While distinct, prediction errors and reflection coefficients in all ladder cells will be denoted by the same symbols $e_l$, $\kappa_l$ to keep the notation simple. When entering a lattice block, $e_{L_s}$ becomes the zeroth-order error $f_{i,0} = b_{i,0}$ for the appropriate chain $i$. In the remainder of this paper the size of a lattice block will still be represented as $L$ rather than $L_s$, unless otherwise noted, with the understanding that this parameter actually changes along the filter for unequal channel orders.

### D. Time Update

The norms of $\mathbf{f}_{i,l}(n)$ and $\mathbf{b}_{i,l}(n)$ in (9), (10) yield the prediction error energies

$$F_{i,l}(n) = \|\mathbf{f}_{i,l}(n)\|^2, \quad B_{i,l}(n) = \|\mathbf{b}_{i,l}(n)\|^2 \quad (12)$$

which are minimized when the reflection coefficients $\kappa_{i,l}^f(n)$ and $\kappa_{i,l}^b(n)$ are chosen according to

$$\kappa_{i,l}^f(n) = -\frac{\langle \mathbf{f}_{i,l-1}(n), \mathbf{b}_{i-1,l-1}(n) \rangle}{\|\mathbf{b}_{i-1,l-1}(n)\|^2} = -\frac{\Delta_{i,l-1}(n)}{B_{i-1,l-1}(n)} \quad (13)$$

$$\kappa_{i,l}^b(n) = -\frac{\langle \mathbf{b}_{i-1,l-1}(n), \mathbf{f}_{i,l-1}(n) \rangle}{\|\mathbf{f}_{i,l-1}(n)\|^2} = -\frac{\Delta_{i,l-1}^*(n)}{F_{i,l-1}(n)} \quad (14)$$

where the inner product in both numerators was abbreviated to $\Delta_{i,l-1}(n)$. As forward prediction is ultimately a growing-memory LS problem with data matrix $\mathbf{U}_{i-1,l}(n)$ and reference $u^{(i)}(0), \ldots u^{(i)}(n)$, a known time recursion exists for the residual energy $F_{i,l}(n)$ involving the *a priori* error $\eta_{i,l}(n)$ and *a posteriori* error $f_{i,l}(n)$ [22]

$$F_{i,l}(n) = \lambda F_{i,l}(n-1) + f_{i,l}(n)\eta_{i,l}^*(n). \quad (15)$$

Moreover, at time $n$ these errors are known to be related by a real conversion factor in the interval [0,1] that is uniquely determined by the input data matrix $\mathbf{U}_{i-1,l}(n)$ (see Appendix A) and will, hence, be denoted by $\gamma_{i-1,l}(n)$. Similar comments can be made about backward prediction and the *a priori* error $\beta_{i,l}$ and *a posteriori* error $b_{i,l}$. As $\eta_{i,l}$, $f_{i,l}$ and $\beta_{i-1,l}$, $b_{i-1,l}$ share a single data matrix, the same will hold for the conversion factor

$$\gamma_{i-1,l}(n) = \frac{f_{i,l}(n)}{\eta_{i,l}(n)} = \frac{b_{i-1,l}(n)}{\beta_{i-1,l}(n)}. \quad (16)$$

The numerator of (13), (14) is the inner product of residual vectors in two RLS problems that share a common input data matrix, and Appendix A shows that it may be recursively updated in time similarly to $F_{i,l}(n)$, $B_{i,l}(n)$ as

$$\Delta_{i,l}(n) = \lambda \Delta_{i,l}(n-1) + \beta_{i-1,l}(n) f_{i,l}^*(n). \quad (17)$$

In the ladder section, where estimation is based on data matrix $\mathbf{U}_{L,l}$, *a priori* errors $\xi_l$ and *a posteriori* errors $e_l$ are related by $e_l(n)/\xi_l(n) = \gamma_{L,l}(n)$.

### E. Angle Normalization

Published single-channel lattice algorithms have been formulated with *a priori*, *a posteriori* or angle-normalized errors [1], [22]. The latter are used in QR-type array algorithms of primary interest to this work due to their excellent numerical properties. In light of the rather direct multichannel extensions that have been derived so far, the following angle-normalized prediction errors are defined:

$$\epsilon_{i,l}^f(n) = \gamma_{i-1,l}^{1/2}(n)\eta_{i,l}(n) = \gamma_{i-1,l}^{-1/2}(n)f_{i,l}(n) \quad (18)$$

$$\epsilon_{i,l}^b(n) = \gamma_{i,l}^{1/2}(n)\beta_{i,l}(n) = \gamma_{i,l}^{-1/2}(n)b_{i,l}(n) \quad (19)$$

$$\epsilon_l(n) = \gamma_{L,l}^{1/2}(n)\xi_l(n) = \gamma_{L,l}^{-1/2}(n)e_l(n). \quad (20)$$

When written in terms of (18)–(20), the time recursions of Section II-D assume a particularly simple form

$$F_{i,l}(n) = \lambda F_{i,l}(n-1) + \left|\epsilon_{i,l}^f(n)\right|^2 \quad (21)$$

$$B_{i,l}(n) = \lambda B_{i,l}(n-1) + \left|\epsilon_{i,l}^b(n)\right|^2 \quad (22)$$

$$\Delta_{i,l}(n) = \lambda \Delta_{i,l}(n-1) + \epsilon_{i-1,l}^b(n)\epsilon_{i,l}^{f*}(n). \quad (23)$$

As these are entirely analogous to the single-channel case, one can still argue as in [1], [22] that, with suitable initialization of (21)–(23), the reflection coefficients $\kappa_{i,l}^f(n)$ and $\kappa_{i,l}^b(n)$ in (13), (14) optimally project onto each other the normalized error vectors

$$\boldsymbol{\epsilon}_{i,l-1}^f(n) = \boldsymbol{\Lambda}^{\frac{1}{2}}(n)\left[\epsilon_{i,l-1}^f(0) \quad \ldots \quad \epsilon_{i,l-1}^f(n)\right]^* \quad (24)$$

$$\boldsymbol{\epsilon}_{i-1,l-1}^b(n) = \boldsymbol{\Lambda}^{\frac{1}{2}}(n)\left[\epsilon_{i-1,l-1}^b(0) \quad \ldots \quad \epsilon_{i-1,l-1}^b(n)\right]^* \quad (25)$$

and may be time updated by RLS. The adaptation of internal variables in an individual lattice cell is formally independent of other cells, and thus remains amenable to parallel implementation.

### III. ARRAY ALGORITHM

The major goal of the previous section was to establish that the known model for single-channel lattice cells still applies to individual units in a modular multichannel lattice with only minor changes. In practice, multichannel expressions are readily obtained from their single-channel counterparts by making the substitutions

$$\epsilon_l^f, \epsilon_{l-1}^f \longrightarrow \epsilon_{i,l}^f, \epsilon_{i,l-1}^f$$
$$F_l, F_{l-1} \longrightarrow F_{i,l}, F_{i,l-1}$$
$$\epsilon_l^b(n), \epsilon_{l-1}^b(n-1) \longrightarrow \epsilon_{i,l}^b(n), \epsilon_{i-1,l-1}^b(n)$$
$$B_l(n), B_{l-1}(n-1) \longrightarrow B_{i,l}(n), B_{i-1,l-1}(n)$$
$$\kappa_l^f, \kappa_l^b \longrightarrow \kappa_{i,l}^f, \kappa_{i,l}^b \quad (26)$$

TABLE I
KALMAN VARIABLES FOR NORMALIZED FORWARD/BACKWARD PREDICTION AND LADDER FILTERING

| Kalman variable | Forward prediction | Backward prediction | Ladder filtering |
|---|---|---|---|
| Observation | $\lambda^{-n/2}\epsilon_{i,l-1}^{f\,*}(n)$ | $\lambda^{-n/2}\epsilon_{i-1,l-1}^{b\,*}(n)$ | $\lambda^{-n/2}\epsilon_{l-1}^{*}(n)$ |
| Measurement "vector" (gain) | $\epsilon_{i-1,l-1}^{b\,*}(n)$ | $\epsilon_{i,l-1}^{f\,*}(n)$ | $\epsilon_{L,l-1}^{b\,*}(n)$ |
| *A priori* state estimate | $-\lambda^{-n/2}\kappa_{i,l}^{f}(n-1)$ | $-\lambda^{-n/2}\kappa_{i,l}^{b}(n-1)$ | $\lambda^{-n/2}\kappa_{l}(n-1)$ |
| State error covariance matrix | $\lambda^{-1}B_{i-1,l-1}^{-1}(n)$ | $\lambda^{-1}F_{i,l-1}^{-1}(n)$ | $\lambda^{-1}B_{L,l-1}^{-1}(n)$ |
| *A priori* output error | $\lambda^{-n/2}\gamma_{i-1,l-1}^{1/2}(n)\eta_{i,l}^{*}(n)$ | $\lambda^{-n/2}\gamma_{i-1,l-1}^{1/2}(n)\beta_{i,l}^{*}(n)$ | $\lambda^{-n/2}\gamma_{L,l-1}^{1/2}(n)\xi_{l}^{*}(n)$ |
| Conversion factor | $\dfrac{\gamma_{i-1,l-1}(n)}{\gamma_{i-1,l}(n)}$ | $\dfrac{\gamma_{i-1,l-1}(n)}{\gamma_{i,l}(n)}$ | $\dfrac{\gamma_{L,l-1}(n)}{\gamma_{L,l}(n)}$ |

and keeping $\epsilon_l$, $\epsilon_{l-1}$, and $\kappa_l$ unchanged. This should be viewed in the broader context of the discussion in [12], which argues that in order-recursive filters the structure and update relations are largely decoupled. A particular multichannel decomposition strategy induces specific interconnections of elementary (single-channel) building blocks, and these may be updated by any algorithm of a relatively large selection of variants. In light of this, the correspondences above follow directly from [9]. However, providing a compact geometric derivation as in Section II is useful to fully justify the exact initialization procedure developed in Section III-A. From (26) guidelines are extracted for building an unforced state-space model for which the basic Kalman filter equations are equivalent to the RLS recursion for $k_{i,l}^{f}(n)$ [22]

$$x(n+1) = \lambda^{-1/2}x(n)$$
$$y(n) = \epsilon_{i-1,l-1}^{b*}(n)x(n) + \nu(n) \qquad (27)$$

where $x(n)$ is the state variable, $y(n)$ is the observation and $\nu(n)$ is white noise with zero mean and unit variance. Table I generalizes the one given in [22], indicating how the Kalman variables for this state-space model relate to RLS variables for forward prediction defined previously. Models for backward prediction and ladder filtering in the multichannel case may be similarly derived, and their correspondences with RLS are also listed in the table. All lines, except for the last two, follow directly from the equivalence between Kalman and RLS filtering. The expression for the *a priori* Kalman output error for forward prediction, $\alpha(n)$, can be verified as in [22], by using (18) and the top three lines of Table I to obtain

$$\alpha(n) = \lambda^{-n/2}\gamma_{i-1,l-1}^{1/2}(n)$$
$$\times \left( \eta_{i,l-1}^{*}(n) + \kappa_{i,l}^{f}(n-1)\beta_{i-1,l-1}^{*}(n) \right). \qquad (28)$$

The term inside parenthesis is computed by the $(i,l)$ unit before the reflection coefficient is updated and equals the *a priori* error $\eta_{i,l}^{*}(n)$. The *a posteriori* Kalman output error $e(n)$ is derived by first noting that the *a posteriori* state estimate is $-\lambda^{-n/2}\kappa_{i,l}^{f}(n)$, and then proceeding similarly to $\alpha(n)$ to obtain $e(n) = \lambda^{-n/2}\gamma_{i-1,l-1}^{-1/2}(n)f_{i,l}^{*}(n)$. In Kalman filter theory the conversion factor is defined as $\alpha(n)/e(n)$.

Having established the correspondences of Table I, all that remains to be done is to substitute them in the standard Kalman filter equations in array form to obtain the desired lattice recursions. Generically, a set of variables is updated by building a

prearray, applying a Givens rotation to zero-out one of its elements, and then reading the new values from other positions in the postarray. Being a type of square-root formulation of the Kalman filter, this algorithm propagates $F_{i,l}^{1/2}$ and $B_{i,l}^{1/2}$ rather than the energies $F_{i,l}$, $B_{i,l}$. Moreover, instead of actual reflection coefficients, the algorithm propagates a set of related variables $p_{i,l}^{f*}$, $p_{i,l}^{b*}$, $p_{l}^{*}$ such that

$$\kappa_{i,l}^{f}(n) = -p_{i,l}^{f}(n)B_{i-1,l-1}^{-1/2}(n) \qquad (29)$$
$$\kappa_{i,l}^{b}(n) = -p_{i,l}^{b}(n)F_{i,l-1}^{-1/2}(n) \qquad (30)$$
$$\kappa_{l}(n) = p_{l}(n)B_{L,l-1}^{-1/2}(n). \qquad (31)$$

This is not problematic if, as is often the case, estimation errors are the only relevant quantities to be extracted from the adaptive filter. Because the algorithm computes normalized errors, the square-root conversion factors $\gamma_{L,l}^{1/2}$ must be propagated as well to obtain the unnormalized *a priori* error at the output of the last ladder block as $\xi_{L}(n) = \gamma_{L,L}^{-1/2}(n)\epsilon_{L}(n)$. Similarly to the single-channel case, updating $\gamma_{L,l}^{1/2}$ simply amounts to using extended ladder arrays where an additional line allows the required variables to be read directly. Table II summarizes the multichannel array algorithm.

At each stage $l$ the forward, backward and ladder arrays can be evaluated in parallel for all chains.[1] At time $n$, the computations are sequentially performed for the various stages of a lattice block, proceeding downstream until all blocks in the filter have been updated. From an implementation point of view, it makes sense to group the variables used to update an individual lattice cell into a single local data structure or hardware unit. This means that lattice cell $(i,l)$ should store $F_{i,l-1}^{1/2}$, $B_{i-1,l-1}^{1/2}$, $p_{i,l}^{f*}$, $p_{i,l}^{b*}$, and, for $i=1$, the conversion factor[2] $\gamma_{L,l}^{1/2}$. All ladder cells in stage $l$ (processing the reference signal and unmerged input channels) read the updated Givens matrix $\mathbf{\Theta}_{1,l}^{b}$ from lattice cell $(1,l)$ and locally store $p_{l}^{*}$.

### A. Initialization

At time $n$, the order recursion for the conversion factor and normalized errors is initialized at the input of the first level of

---

[1]Strictly speaking this is not quite true in the algorithm of Table II because, for improved efficiency, ladder cells at stage $l$ reuse variables from unit $(1,l)$.

[2]As discussed in Section II-C, the prediction order is 0 for newly incorporated channels at the input of block $s+1$, hence the data matrix for ladder filtering $\mathbf{U}_{L_{s+1},0}$ coincides with $\mathbf{U}_{L_s,L_s}$ at the output of the previous block and the conversion factor can be simply propagated between blocks as $\gamma_{L_{s+1},0} = \gamma_{L_s,L_s}$.

TABLE II
SUMMARY OF THE MODULAR MULTICHANNEL QR-RLS LATTICE ALGORITHM IN ARRAY FORM

**Initialization:** Determine lattice/ladder block sizes $L_s$ according to Sec. II-C. For unit $(i, l)$ in lattice block $s$ set

$$p_{i,l}^{f\,*}(-1) = 0, \qquad\qquad p_{i,l}^{b\,*}(-1) = 0,$$

$$F_{i,l-1}^{1/2}(-1) = \sqrt{\delta\lambda^{-2}}, \qquad\qquad B_{i-1,l-1}^{1/2}(-1) = \sqrt{\delta\lambda^{-m_{i,l}-2}},$$

for small $\delta > 0$, $0 < \lambda \le 1$, and $m_{i,l}$ given by (38)–(39). Set $\boldsymbol{\Theta}_{1,l}^b(-1) = \mathbf{I}_2$, $\epsilon_{L_s,l-1}^b(-1) = 0$ throughout chain 1 and $p_l^*(-1) = 0$ in the units of all ladder blocks.

**Update recursions:** At time $n$ set the filter input as

$$\epsilon_{i,0}^f(n) = \epsilon_{i,0}^b(n) = u^{(i)}(n), \qquad\qquad 1 \le i \le L_1,$$

$$\epsilon_0(n) = u^{(i)}(n), \qquad\qquad L_1 + 1 \le i \le L,$$

$$\epsilon_0(n) = d(n), \quad \gamma_{L_1,0}^{1/2}(n) = 1, \qquad\qquad \text{reference}.$$

In lattice block $s > 1$ incorporate new channels as described in Sec. II-C.

*Lattice update:* Compute Givens matrices and update lattice units as

$$\begin{bmatrix} \lambda^{1/2} F_{i,l-1}^{1/2}(n-1) & \epsilon_{i,l-1}^f(n) \\ \lambda^{1/2} p_{i,l}^{b\,*}(n-1) & \epsilon_{i-1,l-1}^b(n) \end{bmatrix} \boldsymbol{\Theta}_{i,l}^f = \begin{bmatrix} F_{i,l-1}^{1/2}(n) & 0 \\ p_{i,l}^{b\,*}(n) & \epsilon_{i,l}^b(n) \end{bmatrix}$$

$$\begin{bmatrix} \lambda^{1/2} B_{i-1,l-1}^{1/2}(n-1) & \epsilon_{i-1,l-1}^b(n) \\ \lambda^{1/2} p_{i,l}^{f\,*}(n-1) & \epsilon_{i,l-1}^f(n) \end{bmatrix} \boldsymbol{\Theta}_{i,l}^b = \begin{bmatrix} B_{i-1,l-1}^{1/2}(n) & 0 \\ p_{i,l}^{f\,*}(n) & \epsilon_{i,l}^f(n) \end{bmatrix}$$

In chain $i = 1$ use the precomputed (delayed) Givens matrix to update the second row of the backward prediction postarray above, then update the Givens matrix, prediction energy and conversion factor as

$$\begin{bmatrix} \lambda^{1/2} B_{L_s,l-1}^{1/2}(n-1) & \epsilon_{L_s,l-1}^b(n) \\ 0 & \gamma_{L_s,l-1}^{1/2}(n) \end{bmatrix} \boldsymbol{\Theta}_{1,l}^b = \begin{bmatrix} B_{L_s,l-1}^{1/2}(n) & 0 \\ \times & \gamma_{L_s,l}^{1/2}(n) \end{bmatrix}$$

Entry $\times$ equals $b_{L_s,l-1}^*(n) B_{L_s,l-1}^{-1/2}(n)$, but need not be evaluated.

*Ladder update:* Ladder arrays share their first row with chain 1. Use the updated Givens matrix $\boldsymbol{\Theta}_{1,l}^b$ to compute the second row

$$\begin{bmatrix} \lambda^{1/2} p_l^*(n-1) & \epsilon_{l-1}(n) \end{bmatrix} \boldsymbol{\Theta}_{1,l}^b = \begin{bmatrix} p_l^*(n) & \epsilon_l(n) \end{bmatrix}$$

**Filter output:** Obtain the *a priori* or *a posteriori* filter output from the final ladder block in the reference chain

$$\xi_L(n) = \gamma_{L,L}^{-1/2}(n)\epsilon_L(n), \qquad\qquad e_L(n) = \gamma_{L,L}^{1/2}(n)\epsilon_L(n).$$

lattice/ladder blocks for zeroth-order prediction $\gamma_{L,0}^{1/2}(n) = 1$, $\epsilon_{i,0}^f(n) = \epsilon_{i,0}^b(n) = u^{(i)}(n)$, and $\epsilon_0(n) = u^{(i)}(n)$ or $\epsilon_0(n) = d(n)$, as appropriate in each chain.

Regarding the time initialization of internal variables in lattice units, under the assumption $u^{(i)}(n) = 0$ for $n < 0$ these may be chosen heuristically as $p_{i,l}^{f\,*}(-1) = p_{i,l}^{b\,*}(-1) = p_l^*(-1) = 0$, and $F_{i,l}^{1/2}(-1) = B_{i,l}^{1/2}(-1) = \sqrt{\delta}$, where $\delta$ is a small positive constant [22]. More formally, one may consider modified LS cost functions where a judiciously-chosen regularization term enables exact initial values to be derived [1]. In the multichannel case the proposed regularized form for an LS problem with data matrix $\mathbf{U}_{i,l}$ is

$$\arg\min_{\mathbf{w}} \left\{ \lambda^{n+1}\mathbf{w}^*\boldsymbol{\Pi}_{i,l}\mathbf{w} + \|\mathbf{d}'(n) - \mathbf{U}_{i,l}(n)\mathbf{w}\|^2 \right\} \quad (32)$$

with suitably chosen reference $\mathbf{d}'$ for each specific problem (forward/backward prediction or filtering), as described in previous

sections. In the single channel case, a convenient regularization matrix for an LS problem with input $u(n), \dots u(n-m+1)$ is

$$\lambda^{n+1}\boldsymbol{\Pi} = \delta\mathrm{diag}(\lambda^{n-1}, \dots \lambda^{n-m}). \quad (33)$$

As shown in [1], the geometrical intuition behind forward/backward prediction is preserved under LS regularization, all update formulas remaining valid. The main effect of regularization according to (33) is to provide a well-defined nonzero value for the residual norms at time $-1$, which can then be recursively updated in time as described previously. The multichannel extension of (33) appearing in (32) is similarly related to the time indices of the elements of $\mathbf{u}_{i,l}(n)$ shown schematically in Fig. 3. Its explicit form is tedious but straightforward, and will be omitted here. This choice of $\boldsymbol{\Pi}_{i,l}$ ensures that the same permutation matrices $\mathbf{S}_{i,l}$, $\mathbf{Q}_{i,l}$ in (6), (7) partition it as

$$\mathbf{S}_{i,l}\lambda^{n+1}\boldsymbol{\Pi}_{i,l}\mathbf{S}_{i,l}^T = \mathrm{diag}(\delta\lambda^{n-1}, \lambda^{n+1}\boldsymbol{\Pi}_{i-1,l-1}) \quad (34)$$

$$\mathbf{Q}_{i,l}\lambda^{n+1}\boldsymbol{\Pi}_{i,l}\mathbf{Q}_{i,l}^T = \mathrm{diag}(\lambda^{n+1}\boldsymbol{\Pi}_{i,l-1}, \delta\lambda^{n-m_j-\tau_{i-l}-1}). \quad (35)$$

Given the compatible partitioning of $\mathbf{U}_{i,l}$ and $\mathbf{\Pi}_{i,l}$ expressed in (6), (7) and (34), (35), the same steps used in the single-channel case [1] may be repeated here to relate the original LS problem (32) with lower-order problems involving forward/backward prediction. Unsurprisingly, order recursions (8)–(10) still hold, but in the resulting expressions the singled-out elements in (34) and (35) are added to $F_{i,l-1}(n)$ and $B_{i,l-1}(n)$, respectively. This implies the following initial values for prediction error energies

$$F_{i,l-1}^{1/2}(-1) = \sqrt{\delta\lambda^{-2}}, \quad B_{i,l-1}^{1/2}(-1) = \sqrt{\delta\lambda^{-m_j-\tau_i-l-2}}. \tag{36}$$

Equivalently, the internal energy variables of unit $(i, l)$ should be set as

$$F_{i,l-1}^{1/2}(-1) = \sqrt{\delta\lambda^{-2}}, \quad B_{i-1,l-1}^{1/2}(-1) = \sqrt{\delta\lambda^{-m_{i,l}-2}} \tag{37}$$

with

$$m_{i,l} \triangleq m_{(i-l)_L} + \tau_{(i-1)_L-l}. \tag{38}$$

The remaining variables are initialized with zero, as in the heuristic scheme mentioned above. Recall also that the same symbol was adopted in all lattice blocks for channel orders appearing in (38). In practice, $m_i$ in block $s$ should be set as the number of previous lattice blocks that include chain $i$

$$m_i = s - \min_{1\leq r\leq s, L_r\geq i} r, \quad i \leq L_s. \tag{39}$$

### B. Computational Complexity

Each scalar unit in the algorithm of Table II executes exactly the same abstract operations as one in a single-channel array lattice [6]. These will be denoted below by $G$ (computing Givens parameters; scaling a positive parameter by $\lambda$ and updating it) and $R$ (rotating a pair of complex values, one of which is scaled by $\lambda$). In the worst-case scenario of equal channel orders complexity increases by a factor of $L$ relative to a single-channel lattice with the same total order, as there are $L$ times more full-blown lattice units in the multichannel filter.

The multichannel filter can also be readily compared with [16], which uses essentially the same high-level block structure (see Fig. 4). In the former, updating a block with dimension $L_k$ requires[3] $2L_k^2(G) + 2L_k^2(R)$ updates in the lattice part and $L_k(L+1-L_k)(R)$ in the associated ladder blocks. In [16] the upper $L_k$ elements in the rightmost column of two matrices with dimension $(L_k + L + 1) \times (L_k + 1)$ and $(L_k + L_{k+1}) \times (L_k + 1)$ must be annihilated (and other entries rotated), requiring $2L_k(G) + L_k(L_k + L_{k+1} + L)(R)$ updates. From [6, Table IV] $G$ and $R$ are broken down[4] into elementary real sums $(A)$, products $(M)$, divisions $(D)$ and square-roots $(Q)$ as $G = 4(A) + 7(M) + 2(D) + 1(Q)$ and $R = 8(A) + 14(M)$. A comparative analysis for several values of $L$ and $L_k$ reveals

[3]For convenience, updating of the conversion factor $\gamma^{1/2}$ in chain 1 by a single real product is ignored. The same is done for the algorithm of [16].

[4]Specifically, $G$ accounts for the "angle computer" steps $(2(A) + 4(M) + 2(D) + 1(Q))$ and rotation of a single real value $(2(A) + 3(M))$, while $R$ accounts for the two-step "rotator" in [6, Table IV].

that the proposed algorithm tends to use slightly fewer sums and products than [16], but more divisions and square-roots.

A comparison with the original modular algorithm of [9] can be made at the level of a single unit, as both filters are structurally identical. Most of its steps, listed also in [4, Table 2], require $4(A) + 4(M)$ operations (reflection coefficient updates require an additional $2(D)$ each, while energy updates are real and use $2(A) + 3(M)$), for a total of $28(A) + 30(M) + 4(D)$ in lattice cells and $12(A) + 12(M) + 2(D)$ in ladder cells. On the other hand, the previous analysis shows that the proposed algorithm requires $2(G) + 2(R) = 24(A) + 42(M) + 4(D) + 2(Q)$ in lattice cells and $1(R) = 8(A) + 14(M)$ in ladder cells. Assigning equal weight to all operations, the algorithm of [9] uses 62 (respectively, 26) operations per lattice (respectively, ladder) unit, versus 72 (22) for the one of Table II. This implies that the latter is less complex only in cases where there are at least 2.5 times more ladder units than lattice units. This might happen when the order for a few of the input channels is much larger than for the remaining ones.

## IV. GIVENS ROTATIONS

The array algorithm relies on a set of unitary matrices $\mathbf{\Theta}$ to annihilate the (1, 2) entry of each prearray in Table II, generating time-updated values in the postarray. For complex data this requirement is satisfied by a complex Givens rotation of the form [22]

$$\mathbf{\Theta} = \begin{bmatrix} c & -s \\ s^* & c \end{bmatrix}. \tag{40}$$

In a prearray with top row $[x\ y]$, where $x$ is real and positive, the sine and cosine parameters are given by [20]

$$c = \left(1 + |\tau|^2\right)^{-1/2}, \quad s = c\tau, \quad \tau = yx^{-1} \tag{41}$$

which requires computing one inverse square root. Because this operation is cumbersome to implement in hardware or programmable fixed-point processors, square-root-free fast Givens rotations have been proposed to circumvent it at the expense of an increase in the numerical range of some variables [5]. Here, two iterative implementations of the original rotation are examined to better preserve the numerical robustness of the array-based lattice algorithm.

### A. Newton Algorithm

The function $c$ in (41) has a continuous derivative with respect to $|\tau|$ and may be iteratively evaluated using the Newton-Raphson algorithm. Due to its quadratic convergence speed, this is one of the preferred numerical methods of computing a wide range of functions including square root, inverse, and inverse square root [23]. Evaluating $c(\tau)$ amounts to finding the positive zero of the function $f(c) = 1 + |\tau|^2 - 1/c^2$, which yields the iteration

$$c_{k+1} = c_k - \frac{f(c_k)}{f'(c_k)} = \frac{c_k}{2}\left(3 - \left(1 + |\tau|^2\right)c_k^2\right). \tag{42}$$

In the lattice algorithm of Table II the parameter $\tau$ is a ratio between instantaneous and accumulated errors at time $n$, and

hence will usually satisfy $|\tau|^2 < 1$ except for small $n$ and during transients in the input/reference sequences. When the Newton iteration is initialized fairly close to the desired value, e.g., $c_0 = 1/2$, the cosine parameter $c$ will be approximated to an accuracy of about $10^{-6}$ in 5–10 iterations. Moreover, when implemented in a fixed-point DSP the simplicity of the Newton iteration (42) will likely allow intermediate values to be stored in internal extended-precision registers used for multiply-accumulate (MAC) operations.

To achieve high numerical accuracy in fixed-point arithmetic, few bits should be allocated to the integer part of $\tau$, but in turn this increases the likelihood of overflow during transient events. Whenever $x \ll |y|$ in (41) a scaling factor $K$ (typically a power of two) could be determined so that $\tau' = y/(Kx)$ does not overflow. Then the modified Newton iteration

$$c_{k+1} = \frac{c_k}{2}\left(3 - \left(K^{-2} + |\tau'|^2\right)c_k^2\right) \qquad (43)$$

converges to a fixed point $c'$ such that $c = c'/K$, $s = c'\tau'$.

### B. CORDIC Algorithm

In hardware implementations, the CORDIC algorithm constitutes an attractive way of approximately zeroing the required prearray element through a number of rotations involving mainly shifts and additions [23]. Specifically, a vector with (real) coordinates $(x, y)$ undergoes a series of rotations over angles $\theta_k$, satisfying $\tan\theta_k = \pm 2^{-k}$, $k \geq 0$, such that $(\sqrt{x^2 + y^2}, 0)$ results. Although this would seem to require an infinite number of iterations, under fixed-point arithmetic there is no point in increasing the index $k$ beyond the available number of bits because $2^{-k}$ would then be represented as 0. An elementary CORDIC rotation involving only shifts and sums is given by

$$(x_{k+1}, y_{k+1}) = (x_k - y_k\tan\theta_k, y_k + x_k\tan\theta_k). \qquad (44)$$

After the last iteration, the resulting vector is multiplied by the constant gain $G = \prod_k \cos\theta_k$ which compensates for a small amplification in every simplified rotation (44).

In addition to *vectoring mode*, described above, the CORDIC processor can also operate in *rotating mode*, applying to its input vector a prespecified sequence of rotations. To annihilate the (1, 2) element of a two-column real prearray its first line is processed in vectoring mode and the resulting angles are used in rotating mode for all remaining rows. This is formally equivalent to right multiplication of the prearray by a real Givens matrix.

A modification to this method is proposed in [21] when the element of the prearray to be annihilated is complex. To make it real its column is first multiplied by a complex exponential, and then the CORDIC procedure is applied in the manner described above. Actually, this prerotation can itself be implemented by CORDIC, resulting in an elegant algorithm with two similar steps. The method of [21] is developed for updating the QR decomposition of the RLS data correlation matrix by appending a new input vector to the triangular Cholesky factor and then using a sequence of two-step rotations to zero-out all its elements and in the process update the factor. The setup for lattice filtering is somewhat different, as a single element is to be annihilated and the remaining entries in that column of the postarray

contain relevant updated variables. It is, therefore, necessary to undo the prerotation by applying a symmetric postrotation after the CORDIC step. Prerotation and CORDIC are, respectively, equivalent to right multiplication of the two-column prearray by $\mathbf{D}(\phi) = \mathrm{diag}(1, e^{-j\phi})$, where $\phi$ is the phase of the (1, 2) element, and

$$\mathbf{C}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \qquad (45)$$

Overall, the complex prearray is thus multiplied by the product of unitary prerotation-CORDIC-postrotation matrices

$$\boldsymbol{\Theta}' = \mathbf{D}(\phi)\mathbf{C}(\theta)\mathbf{D}(-\phi) = \begin{bmatrix} \cos\theta & -e^{j\phi}\sin\theta \\ e^{-j\phi}\sin\theta & \cos\theta \end{bmatrix} \qquad (46)$$

which, unlike $\mathbf{D}(\phi)\mathbf{C}(\theta)$ in [21], conforms to the structure of the general complex Givens matrix (40). In summary, the following practical procedure is used to update the $m \times 2$ arrays in Table II, denoted here by $\begin{bmatrix} d & \epsilon \\ \mathbf{1} & \mathbf{r} \end{bmatrix} \rightarrow \begin{bmatrix} d' & 0 \\ \mathbf{l}' & \mathbf{r}' \end{bmatrix}$

$$[(\epsilon', 0), \{\phi_k\}] = \mathrm{vecmode}\,(\mathrm{Re}\{\epsilon\}, \mathrm{Imag}\{\epsilon\})$$
$$(\mathbf{q}_R, \mathbf{q}_I) = \mathrm{rotmode}\,(\mathrm{Re}\{\mathbf{r}\}, \mathrm{Imag}\{\mathbf{r}\}, \{\phi_k\})$$
$$[(d', 0), \{\theta_k\}] = \mathrm{vecmode}(d, \epsilon')$$
$$(\mathbf{l}'_R, \mathbf{q}'_R) = \mathrm{rotmode}\,(\mathrm{Re}\{\mathbf{l}\}, \mathbf{q}_R, \{\theta_k\})$$
$$(\mathbf{l}'_I, \mathbf{q}'_I) = \mathrm{rotmode}\,(\mathrm{Imag}\{\mathbf{l}\}, \mathbf{q}_I, \{\theta_k\})$$
$$(\mathbf{r}'_R, \mathbf{r}'_I) = \mathrm{rotmode}\,(\mathbf{q}'_R, \mathbf{q}'_I, \{-\phi_k\})$$
$$\mathbf{l}' = \mathbf{l}'_R + j\mathbf{l}'_I, \mathbf{r}' = \mathbf{r}'_R + j\mathbf{r}'_I.$$

## V. SIMULATION RESULTS

The performance of the proposed modular QR-based algorithm (denoted by MQR) is illustrated in simulation and selectively compared with that of plain RLS, QR-RLS (implemented according to [14, Table II]), the original modular algorithm of Glentis and Kalouptsidis [9] (denoted by MGK, and also listed in [4, Table 2]), and the multichannel lattice algorithm of Yang [16] (QR-MLSL(K)). MGK is a type of error-feedback form where reflection coefficients are directly updated in time, rather than indirectly by dividing a crosscorrelation by an accumulated residual energy. Such feedback schemes are thought to exhibit improved numerical stability, at least in the single-channel case [22]. The filter is still structured as shown in Figs. 1 and 4, with lattice unit $(i, l)$ storing and updating $F_{i,l-1}, B_{i-1,l-1}, \kappa_{i,l}^f, \kappa_{i,l}^b$. Conversion factors are absent, as the algorithm explicitly computes *a priori* and *a posteriori* errors. The particular implementation of plain RLS used in this paper is the one designated by version II in [24, Table 13.2], which exhibits improved numerical behavior at the cost of moderate added redundancy.

The algorithms were hand-coded in C in a 32-bit Intel processor. Taking advantage of native integer data types with standard (32 bit) and extended (64 bit) precision, low-level arithmetic functions were developed for true fixed-point arithmetic with variable integer/fractional part lengths, up to a total word length of 32 bits. Floating-point versions of the algorithms were generated by recompiling the same high-level code (making optional adjustments such as computing Givens rotation parameters in closed form), linking with modified low-level arithmetic
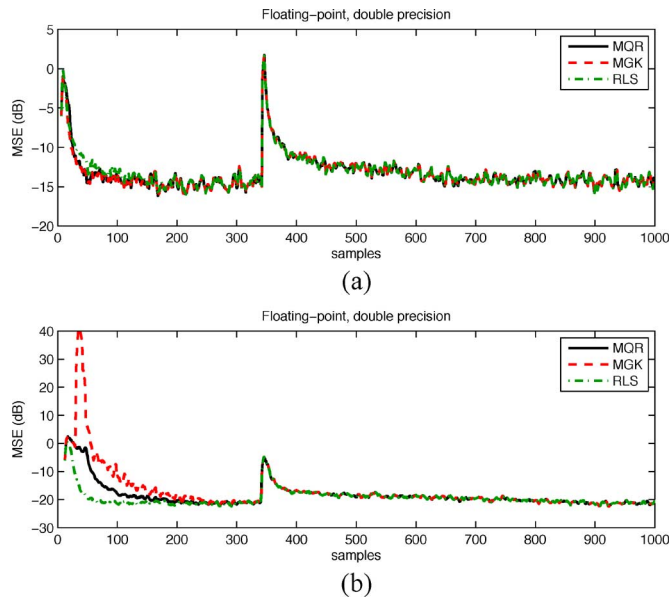
Fig. 6. Learning curves for lattice (MQR, MGK) and plain RLS filtering under 64-bit floating-point arithmetic. (a) AR(6,3,2) channel, RLS order 6,3,2. (b) MA(6,3,2) channel, RLS order 18,9,6.

functions using floating-point operations, and checking the results against MATLAB code.

*AR and MA scenarios*: A random sequence of symbols from the set $\{1, j, -1, -j\}$ is filtered by three randomly-generated parallel channels, complex white noise with identical variance is added to each signal so that the mean SNR is 20 dB, and the resulting multichannel waveform is rescaled for unitary peak magnitude. At time $n = 341$ the channel changes abruptly to an independent realization. In the AR (MA) scenario, the channels are independent single-channel AR (MA) filters with orders 6, 3, and 2 whose poles (zeros) are randomly placed inside the disk $|z| < 0.95$. The RLS filters for AR and MA channels have order 6, 3, 2 and 18, 9, 6, respectively, and the same forgetting factor $\lambda = 0.99$ is used for all. In MA the input signals are differentially delayed with respect to the reference prior to entering each RLS filter to account for 6, 3, 2 (equivalent) anticausal samples.

Fig. 6 shows the *a priori* mean-square error (MSE) performance of RLS, MQR and MGK, averaged over 50 Monte Carlo runs using double-precision floating-point arithmetic. QR-RLS and QR-MLSL(K) provided virtually identical results to MQR, and for clarity are not depicted in the figure. In this particular experiment Givens rotations were calculated in closed form using (41), rather than the iterative methods of Section IV. Bearing in mind their theoretical equivalence, the similarity of results obtained with the various algorithms is not surprising. The MSE in lattice filters, however, tends to overshoot, the effect becoming more noticeable as the filter order increases. Fig. 7 shows similar learning curves for fixed-point arithmetic, and also the steady-state MSE as a function of word length, averaged over iterations 800–1000. Regarding MQR, curves are shown for Givens rotations implemented with the Newton method (MQRN) and CORDIC (MQRC). All units in a given lattice/ladder block share the same numerical format (integer/fraction length), which was manually chosen to cover the empirical floating-point range with as many fractional bits

as possible. The common format used for all internal variables in plain RLS was chosen similarly, but it was found that up to two extra bits were needed in the integer part to avoid saturation when switching to fixed-point arithmetic. Table III lists the integer part lengths that were used for all algorithms in the three test cases described in this section. Steady-state MSE plots include results for QR-RLS using Newton-based rotations (QR-RLSN), but the curves have been omitted in learning plots where they nearly coincide with MQRN. The QR-MLSL(K) algorithm was found to provide even closer results to MQR over all examined precisions using either Newton or CORDIC rotations. Its performance is only explicitly documented in the DFE scenario below, again to avoid cluttering the plots.

Plain RLS is actually more numerically sensitive than the plots suggest, as these do not account for the fact that it failed to converge in about 15% of trials for MA and 2% for AR even with 32-bit precision. Multichannel estimation problems are often ill-conditioned in the absence of noise, reflecting the ambiguity of estimating the reference with similar reliability from different subsets of input channels. This is also the case for the simulated scenarios considered here, and thus increasing the SNR (not shown) degrades the numerical conditioning, expands the dynamic range of the RLS algorithm's internal variables, and ultimately yields poorer performance in fixed point. By contrast, rotation-based and lattice algorithms were able to cope better with such ill-posed LS problems.

Regarding MQRN, the algorithm stops when the difference between consecutive iterates in (42) is lower than $10^{-6}$ or a maximum of 10 Newton steps have been performed. Under floating-point arithmetic 5–6 iterations are typically carried out when $c_0 = 1/2$, whereas the maximum of 10 is often reached when operating in fixed point. MQRN clearly outperforms the CORDIC-based MQRC in Fig. 7 at low precisions, although the difference between them using 8-bit arithmetic is not really meaningful, as both algorithms display large steady-state MSE. Similar behavior was observed in Newton versus CORDIC rotations for QR-RLS and QR-MLSL(K).

The MGK algorithm achieved the best residual MSE performance at low precisions in the AR scenario, which is similar to the one reported in [9], although its convergence time clearly increased relative to floating-point benchmarks. This behavior is exacerbated in the two other MA-like scenarios considered here, i.e., at intermediate precisions the algorithm does not really diverge in the same way that RLS does, but rather strongly overshoots during initialization and then tends to slowly settle down. Note that, as in rotation-based algorithms, abrupt changes in the channel have no obvious negative impact on stability. The poor transient behavior may be related to the choice of computing both *a priori* and *a posteriori* errors, which is not the approach used in known error-feedback single-channel lattice algorithms, where consistency is ensured by propagating conversion factors and a single type of error. As the near cause of the overshooting phenomenon was found to be linked to division by very small prediction error energies, this may possibly be alleviated using thresholding strategies as described in [22, Section 15.11].

*Fractionally-Spaced Decision-Feedback Equalization (DFE)*: In this digital communications scenario the same symbol (reference) sequence considered previously
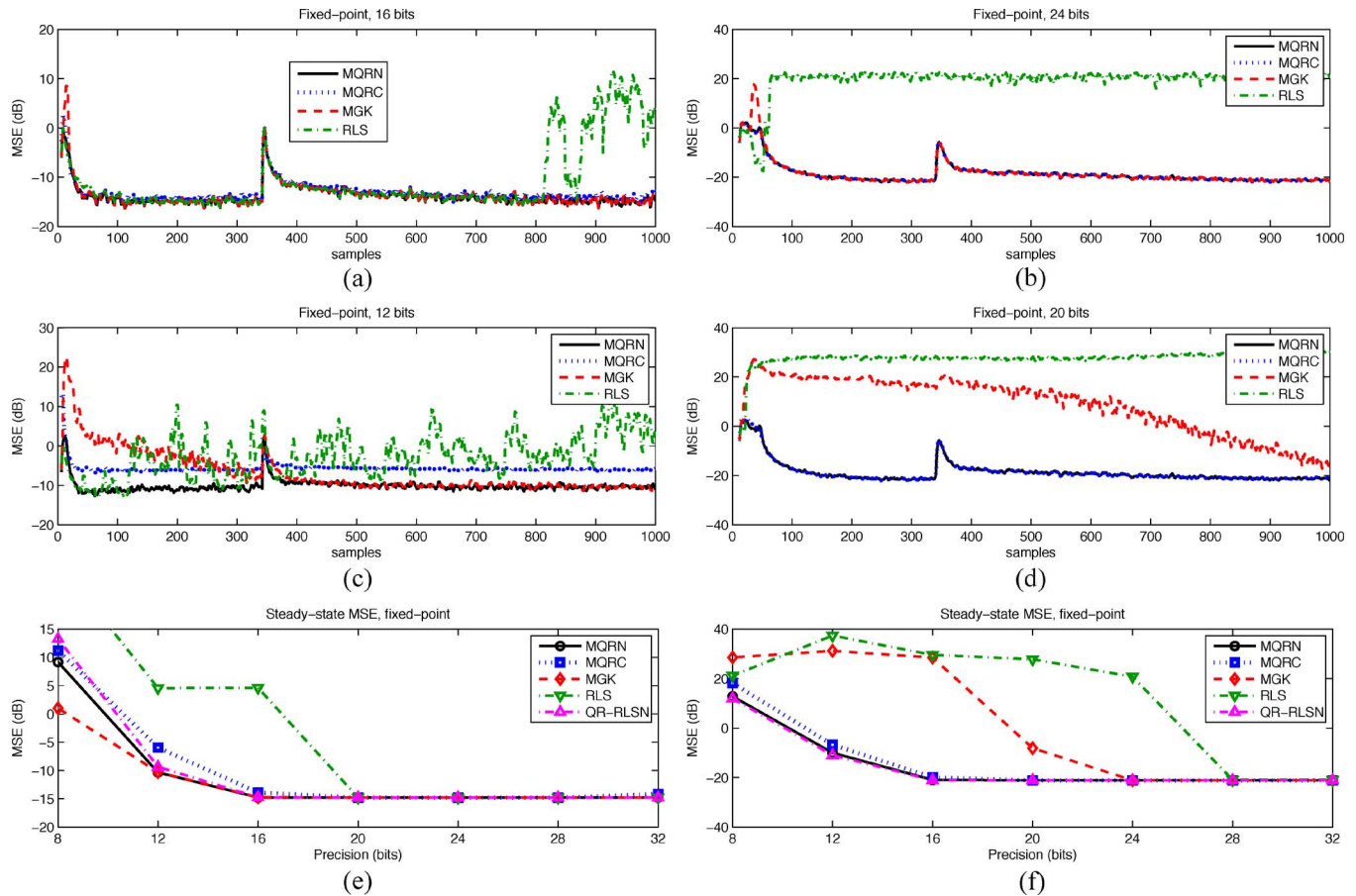
Fig. 7. Fixed-point MSE performance for lattice and transversal filtering. (a), (c) Learning curves for AR(6,3,2) channel, RLS order 6,3,2 using 16, 12 bits. (e) Steady-state MSE for AR channel with variable word length. (b), (d) Learning curves for MA(6,3,2) channel, RLS order 18,9,6 using 24, 20 bits. (f) Steady-state MSE for MA channel.

modulates a train of raised-cosine pulses with 100% rolloff. The continuous pulse-modulated signal is transmitted through a multipath channel with impulse response $\delta(t) + 0.6\delta(t - 1.3T_b) + 0.3\delta(t - 4.6T_b)$, where $T_b$ is the symbol interval. The received signal, contaminated by white Gaussian noise for a mean nominal SNR of 20 dB, is sampled twice per symbol and split into odd and even sample streams at symbol rate. As in MA, these signals are differentially delayed with respect to the reference prior to entering the lattice filter, providing equivalent anticausal samples on both channels as needed. The reference itself, delayed by one sample, is added as a third input channel to be causally processed by the equalizer (feedback filter) when operating in training mode. The lattice DFE merges a feedforward section that filters the input channels and a feedback section that filters the delayed reference. The coefficients for these two sections are typically jointly optimized by a minimum-MSE-like criterion, so the multichannel model of Section II still applies. The orders, chosen by trial and error to yield the best performance, were set to 5, 5, 10, with three anticausal samples on the first two channels. The simulation was run for 2000 symbols.

This LS problem is less well-conditioned than the previous ones, as manifested, e.g., by the greater numerical range of entries in the RLS inverse covariance matrix (see Table III). It is included here primarily to expose differences between QR-type algorithms. Learning curves have been omitted, as they quali-

TABLE III
INTEGER PART LENGTHS (BITS) USED IN THE THREE TEST CASES

|  | AR | MA | DFE |
|---|---|---|---|
| MQR | 4 or 5 | 5 | 6 |
| MGK | 6 or 7 | 6 or 7 | 8 |
| QR-MLSL(K) | 4 or 5 | 5 | 6 |
| RLS | 6 | 7 | 8 |
| QR-RLS | 5 | 5 | 6 |

tatively resemble those of Fig. 7(a), (c) or (b), (d) when transitioning from 20 to 16 bit precision. Fig. 8(a) shows averaged MSE values over iterations 1500–2000 with $\text{SNR} = 20\,\text{dB}$, and Fig. 8(b) repeats the analysis for higher $\text{SNR} = 35\,\text{dB}$, focusing on MQR, QR-MLSL(K), and QR-RLS. The CORDIC version of QR-MLSL(K) (not shown) nearly coincides with the curve for MQRC, similarly to the behavior seen for Newton-based rotations. In this example, QR-RLS has a significant performance advantage over MQR and QR-MLSL(K) at precisions lower than 20 bits, an effect that was partly masked in Fig. 8(a) due to a higher noise floor.

Finally, Fig. 8(c) shows the improvement in MSE, for $\text{SNR} = 35\,\text{dB}$, that results from using extended-precision Newton iterations, as proposed in Section IV-A. Somewhat surprisingly, insignificant gains were obtained for all but the lowest precisions, and even for these the absolute MSE is well above zero, rendering the filter output useless. Similar results were obtained
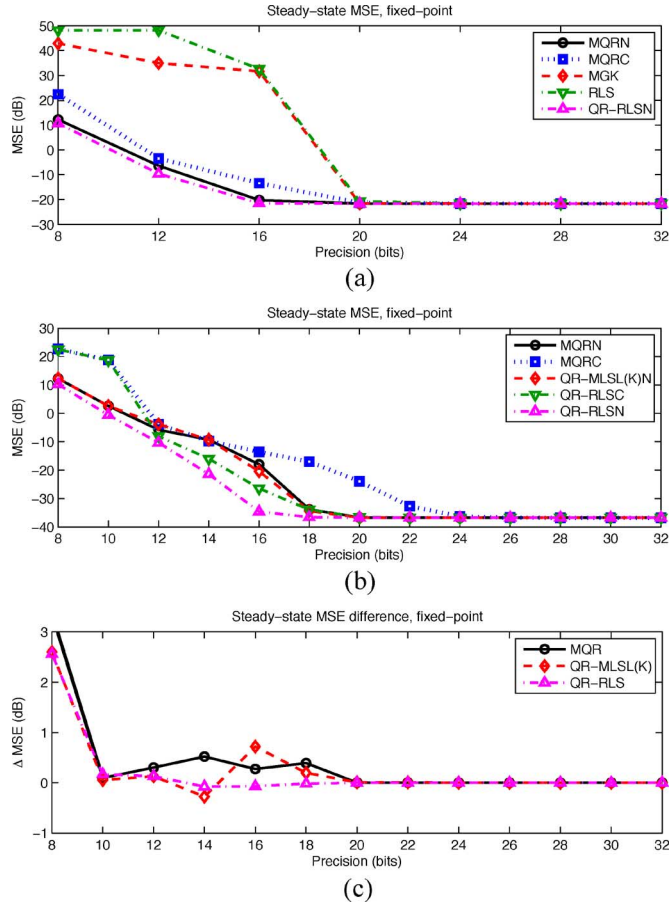
Fig. 8. Fixed-point steady-state MSE in DFE scenario with RLS order 5,5,10. (a) $\mathrm{SNR} = 20$ dB. (b) QR-type algorithms only, $\mathrm{SNR} = 35$ dB. (c) MSE improvement in QR-type algorithms using extended-precision Newton iterations, $\mathrm{SNR} = 35$ dB.

for other tested values of SNR. No definitive explanation for this behavior was found, but intermediate quantizations may introduce a type of randomization that partially compensates for large systematic errors at low precision.

To summarize the results of this simulation study, perhaps the most noteworthy point is that MQR inherits the numerical robustness of QR-type algorithms. Its performance is almost identical to QR-MLSL(K), and while QR-RLS will often attain lower MSE, the difference will usually be small for word lengths of 16–20 bits and higher. Differences in architecture therefore seem to play a minor role in the examined QR-type algorithms. The same trend was observed in an analysis of real data from digital communication experiments in reverberant media, to be reported elsewhere. Being comparable to QR-MLSL(K) both in terms of performance and computational complexity, the MQR filter may very well be the preferred choice for hardware implementation due to the simplicity of its scalar units and highly regular internal block structure, which can be compactly expressed in hardware description languages. Still related to hardware implementation, it would be useful to better understand why Newton-based rotations outperform CORDIC at low/intermediate word lengths. Regarding the MGK algorithm, it is clearly superior to plain RLS, but its performance was found to be inconsistent across test cases and somewhat disappointing for what was expected of an error-feedback form. Clarifying and improving its behavior is beyond

the scope of this work, but it should be noted that similar abrupt changes in the performance of other single-channel error-feedback algorithms have been reported in the technical literature [1].

## VI. Conclusion

An adaptation algorithm in array form for multichannel RLS lattice filtering was presented. The filter is based on a modular decomposition approach and comprises several interconnected units that operate similarly to those found in single-channel lattice filters. The structure is amenable to parallel implementation by adding pipeline registers between stages. As in other algorithms, the complexity for $L$ channels and $M$ coefficients per channel (unequal orders are supported) is $O(ML^2)$. By exploiting a link with Kalman filtering, the update equations were formulated in square-root (QR) array form. Newton and three-step CORDIC algorithms were proposed to carry out complex Givens rotations when the filter is implemented in fixed-point hardware. The Newton-based implementation proved to be slightly more accurate, and will actually execute faster in many programmable architectures. Simulation results show that the algorithm shares the robustness properties of other QR-type variants of RLS, operating reliably over several test cases with fixed-point word lengths under 20 bits. Unlike other considered alternatives, the performance of this algorithm degrades gracefully as the numerical precision decreases.

## APPENDIX A
## RLS PROPERTIES

Consider the RLS cost function (2) at time $n$. It is assumed that the previous optimal coefficient vector $\mathbf{w}(n-1)$ minimizes $J(n-1)$, making the residual vector orthogonal to the columns of the data matrix $\mathbf{U}(n-1)$ so that

$$\mathbf{U}^*(n-1)\left(\mathbf{d}(n-1) - \mathbf{U}(n-1)\mathbf{w}(n-1)\right) = 0. \quad (47)$$

At time $n$, the reference vector and input matrix are updated as

$$\mathbf{d}(n) = \begin{bmatrix} \lambda^{\frac{1}{2}}\mathbf{d}(n-1) \\ d^*(n) \end{bmatrix}, \quad \mathbf{U}(n) = \begin{bmatrix} \lambda^{\frac{1}{2}}\mathbf{U}(n-1) \\ \mathbf{u}^*(n) \end{bmatrix} \quad (48)$$

and using (47) the orthogonality property is seen to be lost

$$\mathbf{U}^*(n)\left(\mathbf{d}(n) - \mathbf{U}(n)\mathbf{w}(n-1)\right) = \mathbf{u}(n)\xi^*(n) \quad (49)$$

where $\xi(n) = d(n) - \mathbf{w}^*(n-1)\mathbf{u}(n)$ is the *a priori* error. To restore orthogonality at time $n$ the RLS algorithm adds a correction to $\mathbf{w}(n-1)$, yielding

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mathbf{U}^+(n)\boldsymbol{\pi}(n)\xi^*(n) \quad (50)$$

where $\boldsymbol{\pi}(n) = [0 \ldots 0 \ 1]^T$ designates the *pinning vector*. Let

$$e(n) = \left(\mathbf{d}(n) - \mathbf{U}(n)\mathbf{w}(n)\right)^* \boldsymbol{\pi}(n) \quad (51)$$

be the *a posteriori* error. Replacing $\mathbf{w}(n)$ with (50) and simplifying leads to the real conversion factor

$$\gamma(n) = e(n)\xi^{-1}(n) = 1 - \boldsymbol{\pi}^*(n)\mathbf{U}(n)\mathbf{U}^+(n)\boldsymbol{\pi}(n). \quad (52)$$

Consider now two distinct RLS problems with reference vectors $\mathbf{d}_1(n), \mathbf{d}_2(n)$, coefficient vectors $\mathbf{w}_1(n), \mathbf{w}_2(n)$, and a common

data matrix $\mathbf{U}(n)$. Using (48) and (50) the *a posteriori* residual vector at time $n$ satisfies, for $i = 1$ or $2$

$$
\begin{aligned}
\mathbf{e}_i(n) &= \mathbf{d}_i(n) - \mathbf{U}(n)\mathbf{w}_i(n) \\
&= \begin{bmatrix} \lambda^{\frac{1}{2}}\mathbf{e}_i(n-1) \\ \xi_i^*(n) \end{bmatrix} - \begin{bmatrix} \lambda^{\frac{1}{2}}\mathbf{U}(n-1) \\ \mathbf{u}^*(n) \end{bmatrix} \\
&\quad \times \mathbf{U}^+(n)\boldsymbol{\pi}(n)\xi_i^*(n).
\end{aligned}
\tag{53}
$$

Using (47), (53) and the idempotence and Hermitian symmetry of $\mathbf{UU}^+$ leads to a recursive expression for the inner product

$$
\begin{aligned}
\mathbf{e}_j^*(n)\mathbf{e}_i(n) &= \lambda \mathbf{e}_j^*(n-1)\mathbf{e}_i(n-1) + \xi_j(n)\xi_i^*(n) \\
&\quad - \xi_j(n)\boldsymbol{\pi}^*(n)\mathbf{U}(n)\mathbf{U}^+(n)\boldsymbol{\pi}(n)\xi_i^*(n).
\end{aligned}
\tag{54}
$$

By (52) the last term equals $(e_j(n) - \xi_j(n))\xi_i^*(n)$, hence

$$
\langle \mathbf{e}_i(n), \mathbf{e}_j(n) \rangle = \lambda \langle \mathbf{e}_i(n-1), \mathbf{e}_j(n-1) \rangle + e_j(n)\xi_i^*(n).
\tag{55}
$$

### ACKNOWLEDGMENT

### REFERENCES

[1] A. H. Sayed, *Fundamentals of Adaptive Filtering*. New York: Wiley-IEEE, 2003.

[2] A. H. Sayed and T. Kailath, "A state-space approach to adaptive RLS filtering," *IEEE Signal Process. Mag.*, vol. 11, no. 4, pp. 18–60, Jul. 1994.

[3] J. Gomes and V. Barroso, "QR-RLS adaptation of modular multichannel lattice filters," in *Proc. 4th Conf. Telecommun. (ConfTele'03)*, Aveiro, Portugal, Jun. 2003, pp. 135–138.

[4] J. Gomes and V. Barroso, "A CORDIC-based QR-RLS multichannel lattice filter," in *Proc. 15th Europ. Signal Process. Conf. (EU-SIPCO'07)*, Poznań, Poland, Sep. 2007, pp. 1043–1047.

[5] P. S. Lewis, "QR-based algorithms for multichannel adaptive least squares lattice filters," *IEEE Trans. Acoust., Speech Signal Process.*, vol. 38, no. 3, pp. 421–432, Mar. 1990.

[6] F. Ling, "Givens rotation based least squares lattice and related algorithms," *IEEE Trans. Signal Process.*, vol. 39, no. 7, pp. 1541–1551, Jul. 1991.

[7] F. Ling and J. G. Proakis, "A generalized multichannel least squares lattice algorithm based on sequential processing stages," *IEEE Trans. Acoust., Speech Signal Process.*, vol. ASSP-32, no. 2, pp. 381–389, Apr. 1984.

[8] H. Lev-Ari, "Modular architectures for adaptive multichannel lattice algorithms," *IEEE Trans. Acoust., Speech Signal Process.*, vol. ASSP-35, no. 4, pp. 543–552, Apr. 1987.

[9] G.-O. A. Glentis and N. Kalouptsidis, "A highly modular adaptive lattice algorithm for multichannel least squares filtering," *Signal Process.*, vol. 46, no. 1, pp. 47–55, 1995.

[10] G.-O. A. Glentis and N. Kalouptsidis, "Efficient order recursive algorithms for multichannel least squares filtering," *IEEE Trans. Signal Process.*, vol. 40, no. 6, pp. 1354–1374, Jun. 1992.

[11] D. T. M. Slock, L. Chisci, H. Lev-Ari, and T. Kailath, "Modular and numerically stable fast transversal filters for multichannel and multiexperiment RLS," *IEEE Trans. Signal Process.*, vol. 40, no. 4, pp. 784–802, Apr. 1992.

[12] F. Ling, J. G. Proakis, and K. Zhao, "A systematic treatment of order-recursive least-squares algorithms," in *Proc. SPIE Adapt. Signal Process.*, 1991, vol. 1565, pp. 296–306.

[13] G.-O. A. Glentis and C. H. Slump, "A highly modular normalized adaptive lattice algorithm for multichannel least squares filtering," in *Proc. Int. Conf. Acoust., Speech Signal Process. (ICASSP'95)*, Detroit, MI, May 1995, vol. 2, pp. 1420–1423.

[14] B. Yang and J. F. Böhme, "Rotation-based RLS algorithms: Unified derivations, numerical properties, and parallel implementations," *IEEE Trans. Signal Process.*, vol. 40, no. 5, pp. 1151–1167, May 1992.

[15] B. Khalaj, A. H. Sayed, and T. Kailath, "A unified derivation of square-root multichannel least-squares filtering algorithms," in *Proc. Int. Conf. Acoust., Speech Signal Process. (ICASSP'93)*, Minneapolis, MN, Apr. 1993, pp. V-523–V-526.

[16] B. Yang, "A QR multichannel least squares lattice algorithm for adaptive nonlinear filtering," *Int. J. Electron. Commun. (AEÜ)*, vol. 49, no. 4, pp. 171–182, Jul. 1995.

[17] M. Harteneck, J. G. McWhirter, I. K. Proudler, and R. W. Stewart, "Algorithmically engineered fast multichannel adaptive filter based on QR-RLS," *Inst. Elect. Eng. Proc.—Vision, Image Signal Process.*, vol. 146, no. 1, pp. 7–13, Feb. 1999.

[18] A. Rontogiannis and S. Theodoridis, "Multichannel fast QRD-LS adaptive filtering: New technique and algorithms," *IEEE Trans. Signal Process.*, vol. 46, no. 11, pp. 2862–2876, Nov. 1998.

[19] A. L. L. Ramos, J. A. Apolinário, Jr., and S. Werner, "Multichannel fast QRD-RLS adaptive filtering: Block-channel and sequential-channel algorithms based on updating backward prediction errors," *Signal Process.*, vol. 87, no. 7, pp. 1781–1798, 2007.

[20] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD: Johns Hopkins Univ. Press, 1996.

[21] C. M. Rader, "VLSI systolic arrays for adaptive nulling," *IEEE Signal Process. Mag.*, vol. 13, no. 4, pp. 29–49, Jul. 1996.

[22] S. Haykin, *Adaptive Filter Theory*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[23] M. D. Ercegovac and T. Lang, *Digital Arithmetic*, ser. The Morgan Kaufmann Series in Computer Architecture and Design. San Francisco, CA: Morgan Kaufmann, 2003.

[24] S. Haykin, *Adaptive Filter Theory*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1991.

**João Pedro C. P. S. Gomes** (S'95–M'03) was born in Lisbon, Portugal, on June 24, 1970. He received the Diploma, M.Sc., and Ph.D. degrees in electrical engineering from Instituto Superior Técnico (IST), Lisbon, in 1993, 1996, and 2002, respectively.

He joined the Department of Electrical and Computer Engineering, IST, in 1995, where he is currently an Assistant Professor. Since 1994, he has also been a Researcher with the Signal and Image Processing Group, Institute for Systems and Robotics, Lisbon. His current research interests include channel identification and equalization in wireless communications, underwater communications and acoustics, and fast algorithms for adaptive filtering.

**Victor A. N. Barroso** (M'89–SM'00) was born in Lisbon, Portugal, on December 22, 1952. He received the E.E. degree in 1976 and the Ph.D. degree in electrical and computer engineering in 1990, both from the Instituto Superior Técnico (IST), Lisbon.

In 1976, he joined the faculty of the Department of Electrical and Computer Engineering, IST. He was promoted to Assistant Professor in 1990, to Associate Professor in 1993, and to Full Professor in 2004. He received the title of "Professor com Agregação" in Electrical and Computer Engineering in 2003 from the Universidade Técnica de Lisboa—IST. He has taught courses in systems and signal theory, control systems, signal and array processing, computer networks, and communications. He currently heads the Institute for Systems and Robotics (ISR), Lisbon, where he has been a researcher in the Signal and Image Processing Group since 1994. His research interests are in statistical signal/image and array processing, and wireless communications. He has published over 100 journal and conference papers.

Prof. Barroso was Vice-President of the IST's Scientific Council from November 2001 until March 2007. Presently, he is the Director of the Portuguese pole of the Information and Communication Technology Institute (ICTI@Portugal), institution responsible for the management of the collaboration program involving several Portuguese Universities, research and development institutions, and companies with the Carnegie-Mellon University, Pittsburgh, PA. He was an Associate Editor for the IEEE TRANSACTIONS ON SIGNAL PROCESSING and for the IEEE SIGNAL PROCESSING LETTERS. He was the representative of the IEEE Signal Processing Society at the IEEE Wireless Periodicals Working Group in 2000, and the General Chair of the IEEE International Workshop on Signal Processing Advances in Wireless Communications in 2004. He is affiliated with several IEEE societies.