A CORDIC-BASED QR-RLS MULTICHANNEL LATTICE FILTER

João Gomes *Victor Barroso*

Instituto Superior Técnico – Instituto de Sistemas e Robótica
Av. Rovisco Pais, 1049–001 Lisboa, Portugal
{jpg,vab}@isr.ist.utl.pt

ABSTRACT

This paper presents an array-based algorithm for multichannel (multiple-input, single-output) lattice filtering. The filter is formed by an array of units that are adapted locally and in parallel using a set of recursions that closely match those for single-channel lattice filters. The design, based on a known step-by-step modular decomposition approach, allows for unequal filter lengths to be specified for different input channels. Individual units are updated using a square-root (QR) recursive-least squares (RLS) algorithm in array form that exhibits highly favorable numerical behavior. This type of filter may be mapped onto hardware by resorting to the CORDIC *algorithm to implement Givens rotations. A procedure based on three* CORDIC *steps is proposed to handle complex data that arise in several applications of multichannel filtering. The algorithm is compared by simulation with plain RLS and another multichannel RLS lattice algorithm over randomly-generated channels, and shown to significantly outperform them under low-precision fixed-point arithmetic.*

1. INTRODUCTION

Adaptive lattice filters are widely used in many areas of signal processing due to their robust numerical behavior and linear increase in computational complexity with filter order [1]. In [2] a key correspondence was established between RLS update expressions and the 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.

This work addresses the extension of the above link with Kalman filtering to the multichannel case, where a single discrete-time sequence is to be estimated from samples observed over a period of time in a set of parallel channels. To avoid dealing with inversion or Cholesky decomposition of covariance matrices, which are difficult to implement in hardware or fixed-point processors, the step-by-step approach of [3] is used. In this type of modular decomposition the full multichannel time/order update is expressed as a set of interrelated single-channel problems whose recursions are purely scalar. Forward/backward linear prediction is central to the development of single-channel lattice filters, and extending that concept to the subproblems that arise in modular decomposition provides the basis for the proposed multichannel algorithm.

Having established the properties of the least-squares (LS) problems addressed in individual units and how the basic single-channel lattice recursions should be adapted to them, one can readily derive equivalent dynamic systems whose Kalman filter equations coincide with RLS update expressions. Specifically, an array-form algorithm based on 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. As in other multichannel filters [3, 4], the algorithm requires $O(ML^2)$ operations per time step for L input channels and M filter coefficients per channel.

The actual mapping from prearrays to postarrays at each time step in scalar lattice units is performed by a complex Givens rotation. To avoid evaluating an inverse square root when computing the Givens matrix parameters [5], a CORDIC implementation suitable for fixed-point hardware is proposed. Complex data are handled by a modification of the two-step technique developed in [6] for systolic arrays, by adding a third CORDIC step that performs phase counter-rotation.

The performance of this algorithm is compared by simulation with that of plain RLS and the multichannel RLS lattice algorithm proposed in [3]. As expected, both lattice algorithms outperform plain RLS under fixed-point arithmetic, but the array form exhibits particularly stable behavior even for precisions as low as 12 bit (and even lower).

2. MULTICHANNEL LATTICE STRUCTURE

Throughout the paper, vectors and matrices will be represented by lowercase boldface and uppercase boldface letters, respectively. The notations $(\cdot)^T$ and $(\cdot)^*$ stand for transpose and complex conjugate transpose (hermitian). Modulo-L indexing (in the range $1,\dots L$ rather than $0,\dots L-1$ as usual), is denoted by $(\cdot)_L$.

In the multichannel setup L input channels convey discrete-time signals $u^{(i)}$, $1 \le i \le L$, 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

This work was supported by Fundação para a Ciência e a Tecnologia (ISR/IST plurianual funding) through the POS–Conhecimento Program that includes FEDER funds.

exponentially-weighted LS cost function

$$J(n) = \|\mathbf{d}(n) - \mathbf{U}(n)\mathbf{w}\|^2 \,, \tag{2}$$

with

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

and $\mathbf{\Lambda}(n) = \mathrm{diag}(\lambda^n, \ldots \lambda^0)$. In RLS algorithms advantage is taken of the structure of $\mathbf{U}(n)$ to incrementally adjust $\mathbf{w}$ at each time step.

2.1 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, 7]. In the scalar case the unit ("block") at stage 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 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 at stage $m-1$, 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 [3] a lattice block is internally formed by an $L \times L$ grid of scalar units. These are regarded as L parallel *chains* of units interacting over L cascaded *stages*. The unit in chain i at stage l, $1 \leq i, l \leq L$, is denoted by (i,l). 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 by 1 in all channels. For convenience, 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 ones,

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

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 $i+1$ (Fig. 1a). 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.

2.2 Order Update

In the interest of space, only a brief outline of the derivation of the multichannel lattice is presented here. The reader is referred to [8] for further details.

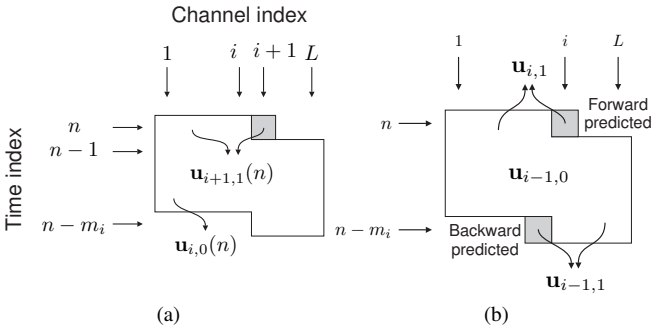

Figure 1: Processing in the first stage ($l = 1$) of a lattice block (a) Time/order update of the input data vector (b) Forward/backward linear prediction for unit $(i,1)$

As in the scalar case, the update recursions for cell (i,l) 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 $\mathbf{u}_{i,l-1}(n)$. Specifically, the upper-rightmost and lower-leftmost elements, labeled in Fig. 1b as "forward predicted" and "backward predicted" for a unit in the first stage, play the same role of the most recent and oldest input samples in a single-channel lattice. Known geometric arguments from the scalar case [1] can then be readily extended to this multichannel setting to conclude that the projection $\hat{\mathbf{d}}_{i,l}(n)$ of $\mathbf{d}(n)$ onto $\mathbf{U}_{i,l}(n)$ may be order updated by considering auxiliary forward and backward prediction error vectors,

$$\hat{\mathbf{d}}_{i,l}(n) = \hat{\mathbf{d}}_{i,l-1}(n) + \kappa_{i,l}(n)\mathbf{b}_{i,l-1}(n) \,, \tag{5}$$

$$\mathbf{b}_{i,l}(n) = \mathbf{b}_{i-1,l-1}(n) + \kappa^b_{i,l}(n)\mathbf{f}_{i,l-1}(n) \,, \tag{6}$$

$$\mathbf{f}_{i,l}(n) = \mathbf{f}_{i,l-1}(n) + \kappa^f_{i,l}(n)\mathbf{b}_{i-1,l-1}(n) \,. \tag{7}$$

In (5)–(7) $\mathbf{f}_{i,l-1}(n)$ and $\mathbf{b}_{i-1,l-1}(n)$ are the vectors of conjugated residuals for weighted LS forward and backward prediction, respectively, of the elements highlighted in Fig. 1b based on $\mathbf{u}_{i-1,l-1}$. The scalars $\kappa_{i,l}$, $\kappa^f_{i,l}$ and $\kappa^b_{i,l}$ are the reflection coefficients. The interconnections between internal units of a lattice block are implied by these expressions [8]. 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 $n-1$.

In principle, the L backward errors in any of the chains could be used for ladder filtering, 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 Fig. 1a). From (5), the recursive expression for the *a posteriori* output error vector $\mathbf{e}_l(n) = \mathbf{d}(n) - \hat{\mathbf{d}}_{L,l}(n)$ to be propagated is

$$\mathbf{e}_l(n) = \mathbf{e}_{l-1}(n) - \kappa_l(n)\mathbf{b}_{L,l-1}(n) \,. \tag{8}$$

2.2.1 Filter Structure

Unequal channel orders are attained by incorporating different channels at distinct points in the processing chain, as discussed in [3, 8]. Assuming that channel indices are 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. Channel i will enter the cascade $m_i - 1$ blocks

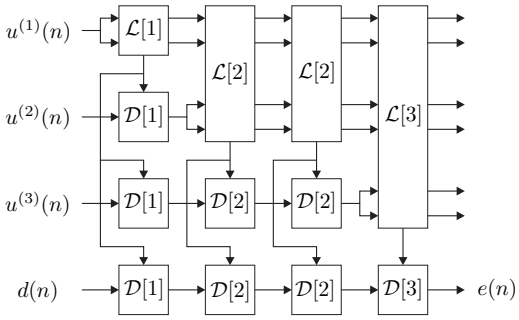

Figure 2: Multichannel lattice filter for channel orders $m_1 = 4$, $m_2 = 3$, $m_3 = 1$

before the final one, so that the input signal $u^{(i)}$ will contribute to a total of m_i blocks as intended. Fig. 2 illustrates the filter structure for orders $m_1 = 4$, $m_2 = 3$, $m_3 = 1$, where, similarly to [3], $L_s \times L_s$ lattice blocks and $1 \times L_s$ ladder sections are denoted by $\mathscr{L}[L_s]$ and $\mathscr{D}[L_s]$, respectively. Each ladder section is fed a scalar prediction error and the L_s backward residuals $b_{L_s,l-1}(n)$, $1 \le l \le L_s$ from its associated lattice block s.

Merging channel i into the lattice section after block s requires knowledge of linear prediction residuals of $u^{(i)}(n)$ given $\mathbf{u}_{L_s,L_s}(n)$ [8]. But this sort of prediction is precisely the purpose of ladder blocks processing the reference input d, so all that is needed is to replicate them for all input channels to be merged downstream, as shown in Fig. 2. While distinct, prediction errors and reflection coefficients in all ladder cells will be denoted by the same symbols e_l, κ_l to keep the notation simple.

3. ARRAY ALGORITHM

The norms of $\mathbf{f}_{i,l}(n)$ and $\mathbf{b}_{i,l}(n)$ in (6)–(7) yield the prediction error energies

$$F_{i,l}(n) = \|\mathbf{f}_{i,l}(n)\|^2\,, \qquad B_{i,l}(n) = \|\mathbf{b}_{i,l}(n)\|^2\,, \quad (9)$$

which are minimized by the optimal reflection coefficients $\kappa^f_{i,l}(n)$, $\kappa^b_{i,l}(n)$. Time update expressions for $F_{i,l}$, $B_{i,l}$ may be derived in terms of *a priori* and *a posteriori* errors, as in the single-channel lattice, and conversion factors γ may also be defined as the ratio between these two types of error. Moreover, the conversion factors of linear prediction errors based on the same vector of input samples are identical, i.e., $f_{i,l}$, $b_{i-1,l} \rightarrow \mathbf{u}_{i-1,l} \rightarrow \gamma_{i-1,l}$.

The array algorithm uses angle-normalized errors $\varepsilon^f_{i,l}$, $\varepsilon^b_{i,l}$, ε_l, that equal their *a posteriori* counterparts divided by the square-root of the associated conversion factors [1]. In the forward case, for example, one has $\varepsilon^f_{i,l}(n) = \gamma^{-1/2}_{i-1,l}(n) f_{i,l}(n)$. When written in terms of these errors, the expressions for the optimal reflection coefficients coincide with the solution of a standard growing-memory LS problem that may be time-updated by RLS [1]. That is the crucial observation that enables individual units to perform a purely local update of their internal state variables (reflection coefficients and prediction error energies).

Although several details have been omitted, the key point to retain is that the concepts underlying the derivation of single-channel lattice algorithms can be directly generalized to the multichannel case using modular decomposition. This includes state-space models that provide the link with Kalman filtering theory on which array algorithms are based. In practice, multichannel expressions are readily obtained from their single-channel counterparts by making the substitutions

$$\begin{aligned}
\varepsilon^f_l\,,\, \varepsilon^f_{l-1}\,,\, F_l\,,\, F_{l-1}\,, &\longrightarrow \varepsilon^f_{i,l}\,,\, \varepsilon^f_{i,l-1}\,,\, F_{i,l}\,,\, F_{i,l-1}\,,\\
\varepsilon^b_l(n)\,,\, \varepsilon^b_{l-1}(n-1) &\longrightarrow \varepsilon^b_{i,l}(n)\,,\, \varepsilon^b_{i-1,l-1}(n)\,,\\
B_l(n)\,,\, B_{l-1}(n-1) &\longrightarrow B_{i,l}(n)\,,\, B_{i-1,l-1}(n)\,,
\end{aligned}$$

and keeping ε_l and ε_{l-1} unaltered. Table 1 summarizes the array-based QR-RLS algorithm where, generically, a set of related 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. Instead of actual forward reflection coefficients, the algorithm propagates energy-normalized coefficients denoted by p. Moreover, square-root conversion factors $\gamma^{1/2}_{L,l}$ must be propagated as well to obtain the unnormalized *a priori* error at the output of the last ladder block.

Table 1: Summary of the multichannel QR-RLS algorithm in array form (MQR)

Initialization: Determine lattice/ladder block sizes L_s according to Sec. 2.2.1 and choose $0 < \lambda \le 1$. For unit (i,l) in lattice block s set

$$p^{f*}_{i,l}(-1) = p^{b*}_{i,l}(-1) = 0\,, \qquad F^{1/2}_{i,l-1}(-1) = B^{1/2}_{i-1,l-1}(-1) = \sqrt{\delta}\,,$$

for small $\delta > 0$. Set $\boldsymbol{\Theta}^b_{1,l}(-1) = \mathbf{I}_2$, $\varepsilon^b_{L_s,l-1}(-1) = 0$ 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

$$\begin{aligned}
&\varepsilon^f_{i,0}(n) = \varepsilon^b_{i,0}(n) = u^{(i)}(n)\,, && 1 \le i \le L_1\,,\\
&\varepsilon_0(n) = u^{(i)}(n)\,, && L_1 + 1 \le i \le L\,,\\
&\varepsilon_0(n) = d(n)\,, \quad \gamma^{1/2}_{L_1,0}(n) = 1\,, && \text{reference}\,.
\end{aligned}$$

In lattice block $s > 1$ incorporate new channels as described in Sec. 2.2.1.

Lattice update: Compute Givens matrices and update lattice units as

$$\begin{bmatrix} \lambda^{1/2} F^{1/2}_{i,l-1}(n-1) & \varepsilon^f_{i,l-1}(n) \\ \lambda^{1/2} p^{b*}_{i,l}(n-1) & \varepsilon^b_{i-1,l-1}(n) \end{bmatrix} \boldsymbol{\Theta}^f_{i,l} = \begin{bmatrix} F^{1/2}_{i,l-1}(n) & 0 \\ p^{b*}_{i,l}(n) & \varepsilon^b_{i,l}(n) \end{bmatrix}$$

$$\begin{bmatrix} \lambda^{1/2} B^{1/2}_{i-1,l-1}(n-1) & \varepsilon^b_{i-1,l-1}(n) \\ \lambda^{1/2} p^{f*}_{i,l}(n-1) & \varepsilon^f_{i,l-1}(n) \end{bmatrix} \boldsymbol{\Theta}^b_{i,l} = \begin{bmatrix} B^{1/2}_{i-1,l-1}(n) & 0 \\ p^{f*}_{i,l}(n) & \varepsilon^f_{i,l}(n) \end{bmatrix}$$

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

$$\begin{bmatrix} \lambda^{1/2} B^{1/2}_{L_s,l-1}(n-1) & \varepsilon^b_{L_s,l-1}(n) \\ 0 & \gamma^{1/2}_{L_s,l-1}(n) \end{bmatrix} \boldsymbol{\Theta}^b_{1,l} = \begin{bmatrix} B^{1/2}_{L_s,l-1}(n) & 0 \\ \times & \gamma^{1/2}_{L_s,l}(n) \end{bmatrix}$$

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

$$\begin{bmatrix} \lambda^{1/2} p^*_l(n-1) & \varepsilon_{l-1}(n) \end{bmatrix} \boldsymbol{\Theta}^b_{1,l} = \begin{bmatrix} p^*_l(n) & \varepsilon_l(n) \end{bmatrix}$$

Filter output: Obtain the *a priori* filter output from the final ladder block in the reference chain as $\xi_L(n) = \gamma^{-1/2}_{L,L}(n)\varepsilon_L(n)$.

4. CORDIC IMPLEMENTATION

The array algorithm relies on a sequence of unitary matrices $\mathbf{\Theta}$ to annihalate the $(1,2)$ entry of each prearray in Table 1, generating time-updated values in the postarray. For complex data this requirement is satisfied by a complex Givens rotation of the form [7]

$$\mathbf{\Theta} = \begin{bmatrix} c & -s \\ s^* & c \end{bmatrix}. \tag{10}$$

In a prearray with top row $[x \quad y]$, where x is real and positive, the sine and cosine parameters are given by [5]

$$c = \left(1+|\tau|^2\right)^{-1/2}, \qquad s = c\tau, \qquad \tau = yx^{-1}, \tag{11}$$

which requires computing one inverse square root. In hardware implementations or programmable fixed-point processors, where this operation is cumbersome, the CORDIC algorithm constitutes an attractive way of approximately zeroing the required prearray element through a number of rotations involving mainly shifts and additions [9]. Specifically, a vector with (real) coordinates (x,y) undergoes a series of rotations over angles θ_k, satisfying $\tan\theta_k = \pm 2^{-k}$, $k \geq 0$, such that vector $(\sqrt{x^2+y^2},0)$ results. Under fixed-point arithmetic the index k is only increased up to the available number of bits. Simplified elementary rotations introduce a small amplification, which is globally compensated after the last one by applying the constant gain $G = \prod_k \cos\theta_k$.

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 annihalate the $(1,2)$ element of a two-column real prearray its first line is processed in vectoring mode and the resulting bit stream is used in rotating mode for all remaining rows.

A modification to this method is proposed in [6] when the element of the prearray to be annihalated is complex. To make it real its column is first multiplied by a suitable 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 [6] 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. This corresponds to an array-type update rule of the form

$$[\mathbf{L}(n-1) \quad \mathbf{u}(n)]\,\mathbf{\Theta} = [\mathbf{L}(n) \quad \mathbf{0}], \tag{12}$$

where $\mathbf{L}$ denotes the lower-triangular Cholesky factor and $\mathbf{\Theta}$ is the product of unitary matrices that perform prerotation/Givens annihalation for all the entries of $\mathbf{u}(n)$.

The setup for lattice filtering is somewhat different from (12), as a single element is to be annihalated 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 ϕ is the phase of the $(1,2)$ element, and

$$\mathbf{C}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{13}$$

Table 2: Summary of the multichannel lattice algorithm of [3] (MKL)

Initialization: Determine lattice/ladder block sizes L_s according to Sec. 2.2.1 and choose $0 < \lambda \leq 1$. For unit (i,l) in lattice block s set

$$\kappa^f_{i,l}(-1) = \kappa^b_{i,l}(-1) = 0\,, \qquad F_{i,l-1}(-1) = B_{i-1,l-1}(-1) = \delta\,,$$

for small $\delta > 0$. Set $\beta_{L_s,l-1}(-1) = b_{L_s,l-1}(-1) = 0$ throughout chain 1 and $k_l(-1) = 0$ in the units of all ladder blocks.

Update recursions: At time n set the filter input as

$$\begin{aligned} &\eta_{i,0}(n) = f_{i,0}(n) = \beta_{i,0}(n) = b_{i,0}(n) = u^{(i)}(n)\,, && 1 \leq i \leq L_1\,,\\ &\xi_0(n) = e_0(n) = u^{(i)}(n)\,, && L_1+1 \leq i \leq L\,,\\ &\xi_0(n) = e_0(n) = d(n)\,, && \text{reference}\,. \end{aligned}$$

Incorporate new channels as described in Sec. 2.2.1.

Lattice update:

$$\begin{aligned} F_{i,l-1}(n) &= \lambda\, F_{i,l-1}(n-1) + f_{i,l-1}(n)\eta^*_{i,l-1}(n)\\ B_{i-1,l-1}(n) &= \lambda\, B_{i-1,l-1}(n-1) + b_{i-1,l-1}(n)\beta^*_{i-1,l-1}(n)\\ \eta_{i,l}(n) &= \eta_{i,l-1}(n) + \kappa^{f\,*}_{i,l}(n-1)\beta_{i-1,l-1}(n)\\ \beta_{i,l}(n) &= \beta_{i-1,l-1}(n) + \kappa^{b*}_{i,l}(n-1)\eta_{i,l-1}(n)\\ \kappa^f_{i,l}(n) &= \kappa^f_{i,l}(n-1) - \frac{b_{i-1,l-1}(n)}{B_{i-1,l-1}(n)}\eta^*_{i,l}(n)\\ \kappa^b_{i,l}(n) &= \kappa^b_{i,l}(n-1) - \frac{f_{i,l-1}(n)}{F_{i,l-1}(n)}\beta^*_{i,l}(n)\\ f_{i,l}(n) &= f_{i,l-1}(n) + \kappa^{f\,*}_{i,l}(n)b_{i-1,l-1}(n)\\ b_{i,l}(n) &= b_{i-1,l-1}(n) + \kappa^{b*}_{i,l}(n)f_{i,l-1}(n) \end{aligned}$$

In chain $i = 1$ use the stored energy $B_{L_s,l-1}(n-1)$ to compute $\kappa^f_{1,l}(n)$ above and only update the residual energy as a final step

$$B_{L_s,l-1}(n) = \lambda\, B_{L_s,l-1}(n-1) + b_{L_s,l-1}(n)\beta^*_{L_s,l-1}(n)$$

Ladder update:

$$\begin{aligned} \xi_l(n) &= \xi_{l-1}(n) - \kappa^*_l(n-1)\beta_{L_s,l-1}(n)\\ \kappa_l(n) &= \kappa_l(n) + \frac{b_{L_s,l-1}(n)}{B_{L_s,l-1}(n)}\xi^*_l(n)\\ e_l(n) &= e_{l-1}(n) - \kappa^*_l(n)b_{L_s,l-1}(n) \end{aligned}$$

Filter output: The *a priori* error $\xi_L(n)$ is directly available from the final ladder block in the reference chain.

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

$$\mathbf{\Theta}' = \mathbf{D}(\phi)\mathbf{C}(\theta)\mathbf{D}(-\phi) = \begin{bmatrix} \cos\theta & -e^{j\phi}\sin\theta \\ e^{-j\phi}\sin\theta & \cos\theta \end{bmatrix}, \tag{14}$$

which, unlike $\mathbf{D}(\phi)\mathbf{C}(\theta)$ in [6], conforms to the structure of the general complex Givens matrix (10). Note that this approach may be applied in other algorithms where complex Givens rotations are used.

5. SIMULATION RESULTS

The performance of the proposed algorithm (denoted by MQR) is illustrated in a simulated moving-average (MA) channel and compared with that of plain RLS and the multichannel algorithm of [3] (denoted by MKL here). The latter is a type of error-feedback form where reflection coefficients are directly updated in time by a hybrid scheme that

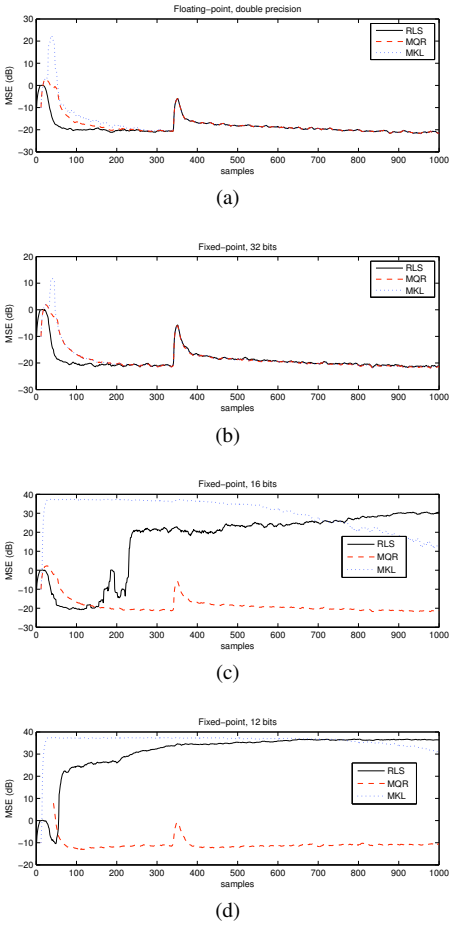

Figure 3: MSE performance for MA channel (a) Floating point: 64 bits (double) (b)–(d) Fixed point: 32, 16, 12 bits

propagates both *a posteriori* (f, b, e) and *a priori* (η, β, ξ) errors. Table 2 lists the update equations for MKL using the same notation as in the array algorithm of Table 1 and correcting some minor misprints that appear in [3]. The MKL filter is still structured as described in Secs. 2.2 and 2.2.1, with lattice unit (i,l) internally storing and updating $F_{i,l-1}$, $B_{i-1,l-1}$, $\kappa^f_{i,l}$, $\kappa^b_{i,l}$.

In the simulated scenario a random sequence of symbols from the set $\{1, j, -1, -j\}$ is filtered by three parallel single-input/single-output 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. The channels are independent single-channel MA filters with orders 6, 3 and 2 whose zeros are randomly placed inside the disk $|z| < 0.95$. At time $n = 341$ they change abruptly to independent realizations. The RLS filters have order $m_1 = 18$, $m_2 = 9$, $m_3 = 6$, and forgetting factor $\lambda = 0.99$. The input signals and reference are differentially delayed prior to entering each RLS filter so that the output equivalently accounts for 6, 3, 2 anticausal samples.

Fig. 3a shows the mean-square error (MSE) performance of the three algorithms in 50 Monte Carlo runs using double-precision (64 bits) floating-point arithmetic. In this particular experiment Givens rotations were calculated with (11) rather than CORDIC. Bearing in mind their theoretical equivalence, the similarity of results obtained with the three algorithms is not surprising. The MSE in lattice filters, however, tends to overshoot, the effect being more noticeable as the filter order increases.

Figs. 3b–d show similar results for fixed-point arithmetic using 32, 16 and 12 bits. 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 2 extra bits were needed in the integer part to avoid saturation when switching to fixed-point arithmetic. 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 20% of trials even with 32-bit precision. Unlike MQR and MKL, its behavior further degrades at higher SNR due to ill-conditioning of the input covariance matrix.

6. CONCLUSION

An adaptation algorithm in array form for multichannel RLS lattice filtering was presented. The filter is based on modular decomposition and comprises several interconnected units that operate similarly to those found in single-channel lattice filters. As in other algorithms, the complexity for L channels and M coefficents 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. A three-step CORDIC algorithm was proposed to carry out complex Givens rotations when the filter is implemented in fixed-point hardware. Simulation results show that the algorithm is robust, operating reliably even with low numerical precision. Other alternatives for performing Givens rotations are currently being investigated.

REFERENCES

[1] A. Sayed, *Fundamentals of Adaptive Filtering*. Wiley, 2003.

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

[3] G. Glentis, N. Kalouptsidis, "A highly modular adaptive lattice algorithm for multichannel least squares filtering," *Sig. Proc.*, vol. 46, no. 1, pp. 47–55, 1995.

[4] P. Lewis, "QR-based algorithms for multichannel adaptive least squares lattice filters," *IEEE Trans. on Acoust., Speech and Sig. Proc.*, vol. 38, no. 3, pp. 421–432, Mar. 1990.

[5] G. Golub, C. Van Loan, *Matrix Computations*, 3rd ed. Johns Hopkins Univ. Press, 1996.

[6] C. Rader, "VLSI systolic arrays for adaptive nulling," *IEEE Sig. Proc. Mag.*, vol. 13, no. 4, pp. 29–49, Jul. 1996.

[7] S. Haykin, *Adaptive Filter Theory*, 3rd ed. Prentice-Hall, 1996.

[8] J. Gomes, V. Barroso, "Array-based QR-RLS multichannel lattice filtering," *To be submitted to IEEE Trans. on Sig. Proc.*, 2007.

[9] M. Ercegovac, T. Lang, *Digital Arithmetic*. Morgan Kaufmann, 2003.