USING A LATTICE ALGORITHM TO ESTIMATE THE KALMAN GAIN VECTOR IN FAST NEWTON-TYPE ADAPTIVE FILTERING

Marc Moonen[1] *Ian Proudler*[2]

[1] ESAT - Katholieke Universiteit Leuven, Kardinaal Mercierlaan 94, 3001 Heverlee - Belgium
[2] DRA, Room E506, St Andrews Road, Malvern, Worcs., WR14 3PS, UK

ABSTRACT

In this paper we consider a recursive least squares (RLS) adaptive filtering problem where the input signal can be modelled as the output of a low order autoregressive (AR) process. We will show how a good estimate of the Kalman gain vector can be obtained using a small least squares lattice (LSL) filter. This estimate can then be used in the normal way to determine the optimum filter coefficients. The resulting adaptive filtering algorithm is similar in concept to the Fast Newton algorithm. The main difference is the use of the LSL instead of a low order covariance domain fast RLS algorithm. The potential advantage of this new algorithm is that, unlike a covariance domain algorithm, a LSL can be implemented in a numerically stable form.

1. INTRODUCTION

It is well known [1] that linear prediction of the input (regression) signal is a fundamental part of any fast RLS algorithm. Mathematically, one is led to a linear prediction problem of the same order as the original adaptive filtering one. Physically, however, there is no reason why the orders of the two problems should be so coupled. The Fast Newton algorithm [3] decouples these two problems and as a result has the convergence speed of RLS and yet nearly the same computational complexity as the LMS algorithm. The algorithm has two potential numerical 'weak spots': the weight-update recursion (see equation (1)) and the (low order) covariance domain RLS algorithm used to solve the linear prediction problem. In this paper we will show how the latter algorithm can be replaced by a (stable) least squares lattice (LSL).

We take as our starting point, the so-called 'inverse updating' RLS algorithm [1, 4] as specified in section 2. In section 3, the principle of the so-called Fast Newton approach [3] is explained by means of graphical representations (signal flow graphs). In section 4, it is shown how a LSL based Fast Newton-type algorithm may be derived. Preliminary simulation results are given in section 5.

2. INVERSE UPDATING RLS

Consider the basic RLS weight-update recursion:

$$\begin{aligned} \underline{w}(k) &= \underline{w}(k-1) + \underline{\kappa}_p(k) \cdot e(k) \\ e(k) &= d(k) - \underline{w}^T(k-1) \cdot \underline{u}_p(k) \end{aligned} \tag{1}$$

where the p-dimensional vector $\underline{w}(k)$ consists of the adaptive filtering weights at time k, $\underline{\kappa}_p(k)$ is the Kalman gain vector, $d(k)$ is the desired signal, and $\underline{u}_p(k)$ is the p-dimensional regression vector

$$\underline{u}_p(k) = \left[\begin{array}{cccc} u(k) & u(k-1) & \ldots & u(k-p+1) \end{array} \right]^T .$$

It is well know [1] that the Kalman gain vector is given by

$$\underline{\kappa}_p(k) = R_p^{-1}(k) \cdot R_p^{-T}(k) \cdot \underline{u}_p(k) \tag{2}$$

where $R_p(k)$ is the (upper triangular) Cholesky factor of the covariance matrix $M_p(k) = \sum_{i=1}^{k} \underline{u}_p(i)\underline{u}_p^T(i)$.

The inverse updating RLS algorithm is based on storing and updating $\underline{w}(k)$ together with $R_p^{-T}(k)$ (lower triangular). A signal flow graph (SFG) is given in **Figure 1** (for a 4th order problem). We assume that the reader is familiar with this algorithm, and give only a brief explanation. Details may be found in [1, 4, 2].

The basic building blocks which will be used in the SFGs are shown at the right. White hexagons represent 2×2 orthogonal (Givens) transformations of the form

$$\left[\begin{array}{c} a' \\ b' \end{array} \right] = \left[\begin{array}{cc} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{array} \right] \left[\begin{array}{c} a \\ b \end{array} \right]$$

where the rotation angle ϕ is one of the inputs. The hexagons in the left-most column of Figure 1 portray the computation of a rotation angle ϕ, such that one of the outputs is forced to zero and the other one is positive. Black squares represent memory cells (delay elements), possibly combined with exponential weighting with a weight factor λ.

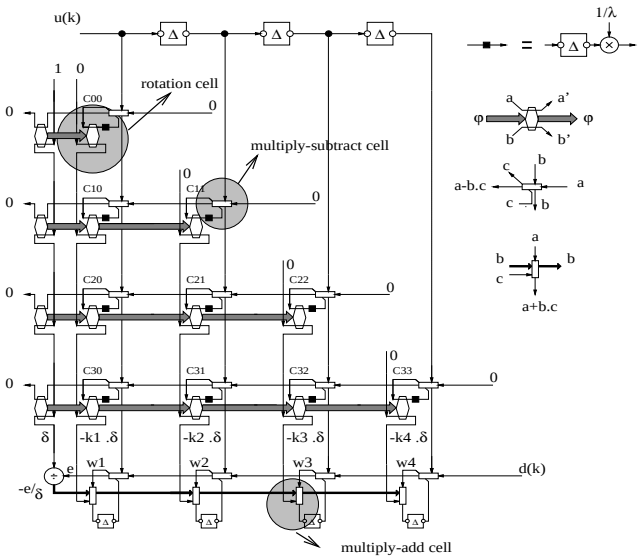

Figure 1. Inverse updating RLS algorithm

The weight-update recursion, formula (1), is performed in the bottom row. The residual $e(k)$ is accumulated from right to left. The Kalman gain vector is produced by the triangular part, up to a scaling $-\delta$, together with δ itself (available at the left, as indicated). The updating of $R_p^{-T}(k-1)$ is done by means of orthogonal transformations, defined at the left-hand side of the array. The vector $-R_p^{-T}(k-1)\cdot\underline{u}_p(k)$ is needed to compute the rotation angles, and this matrix-vector product is also accumulated from right to left. Further SFG details may be found in [2, 5].

3. FAST NEWTON TYPE ALGORITHMS

It is well known that

$$R_p^{-T}(k) = \begin{bmatrix} c_{00}(k) & 0 & \dots & \dots & 0 \\ c_{10}(k) & c_{11}(k) & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ c_{\tilde{p}0}(k) & c_{\tilde{p}1}(k) & \dots & \dots & c_{\tilde{p}\tilde{p}}(k) \end{bmatrix} \quad (3)$$

where $\underline{c}_i(k) = [c_{i0}(k)\dots c_{ii}(k)]$ is the (energy normalised) i-th order backward prediction-error filter coefficient vector and $\tilde{p} = p-1$.

Now if the signal $u(k)$ is modelled as being generated by an AR process of order $m < p$, we have [3] that (for $i > m$) $c_{i,0}(k) = 0$ and (for $i > m$ and $i \geq j > 0$) $c_{i,j}(k) = c_{i-1,j-1}(k-1)$. Hence it is possible, knowing only the solution to the m-th order backward prediction problem, to construct the full p-th order Cholesky factor R_p by extending R_m. This is illustrated in **Figure 2** (for a 9th order problem, with $m = 2$). In the triangular matrix, the entries below the m-th subdiagonal are set equal to zero, and the non-zero diagonals are derived from simple delay lines. This is possible since these coefficients are known given $\underline{c}_m$ and hence

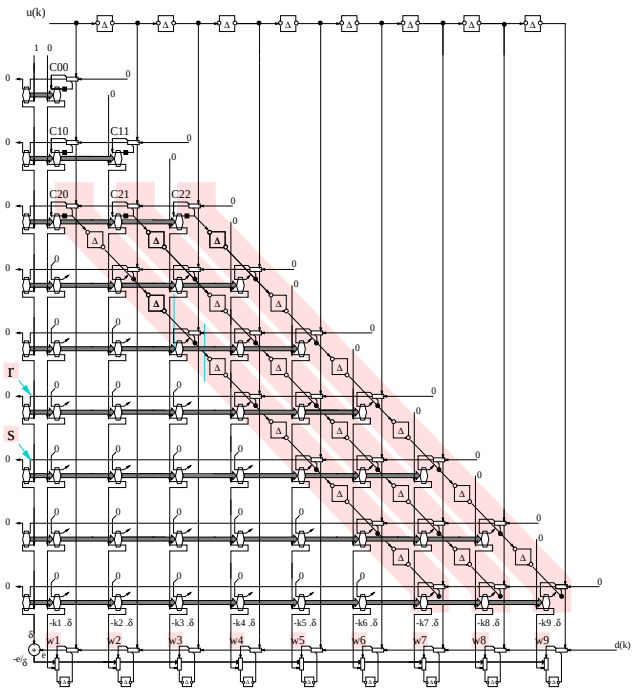

Figure 2. Inverse updating RLS with AR input

need not be stored and updated. The 'backwards only' Fast Newton algorithm [3] results when the time-shift structure now present in R_p^{-T} is used to construct an *O(p)* algorithm for generating the p-th order Kalman gain vector $\underline{\kappa}_p$ from the m-th order one $\underline{\kappa}_m$. The main ideas are illustrated in Figures 3-4-5.

First, it is readily seen that, because of the time-shift structure (delay lines) in $R_p^{-T}(k-1)$ as well as in $\underline{u}_p(k)$, the (lower part of the) vector $-R_p^{-T}(k-1)\cdot\underline{u}_p(k)$, *i.e.* the computed matrix-vector product, has a shift structure, too. In Figure 2, it is readily verified that, *e.g.*, $s(k) = r(k-1)$ (left-hand column). This is exploited in **Figure 3**, where the lower part of $-R_p^{-T}(k-1)\cdot\underline{u}_p(k)$, starting from its $(m+2)$-nd component, is derived from the computed $(m+1)$-st component, by means of a delay line. The end result is that the *O(p²)* matrix-vector product is now reduced to a (much cheaper) *O(m²)* matrix-vector product.

In a similar fashion, one can exploit the available shift structure to reduce the complexity of the Kalman gain vector computation. Figure 3 is first transformed into **Figure 4**. In Figure 3, the Kalman gain vector $\kappa_p = \kappa$ (resp. $\kappa_{m+1} = \bar{\kappa}$) is produced up to a scaling $-\delta$ (resp. $-\bar{\delta}$), together with the scaling itself (up to a sign) using orthogonal transformations. Given that no coefficient update operation is needed, it is easy to show that multiply-add operations can be used for the update instead provided the scaling in changed from $-\delta$ (resp. $-\bar{\delta}$) to $-\delta^2$ (resp. $-\bar{\delta}^2$), as indicated. In Figure

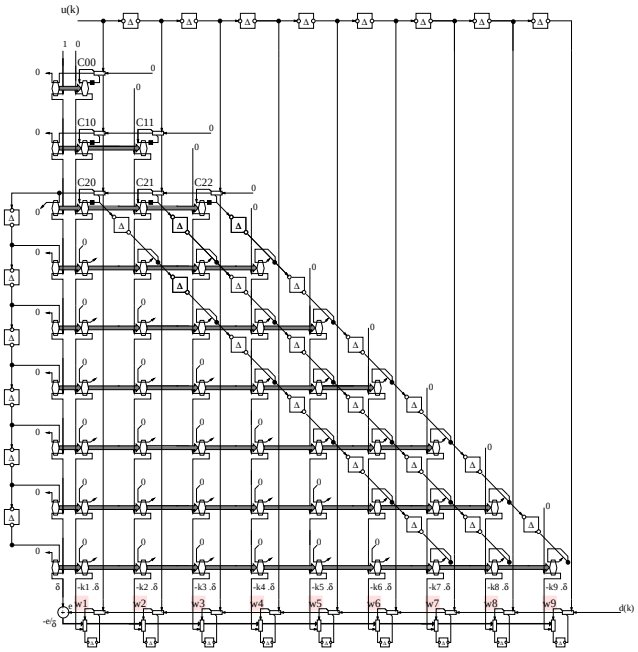

Figure 3. Derivation Fast Newton algorithm (1)

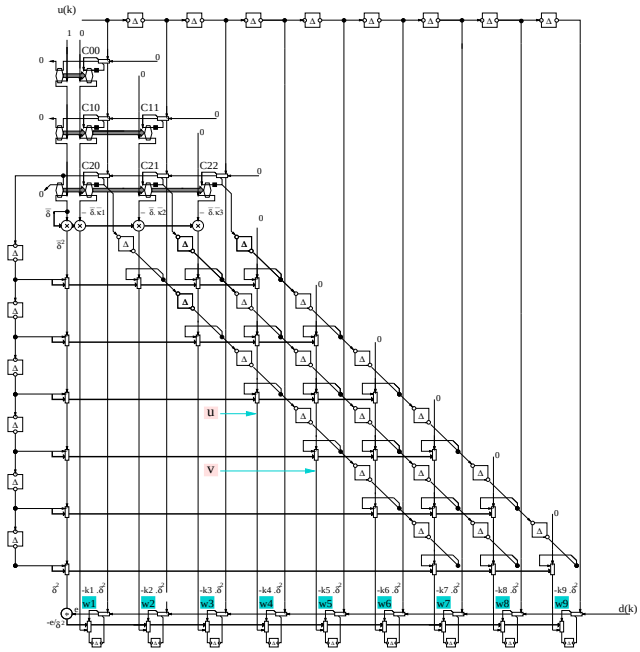

Figure 4. Derivation Fast Newton algorithm (2)

4, it is then readily verified that, *e.g.*, $v(k) = u(k-1)$. This is exploited in **Figure 5**, where the middlemost part of the Kalman gain vector computation is replaced by a delay line. Figure 5 is essentially a representation of a square-root covariance domain, backwards-only Fast Newton algorithm, with $O(p+m^2)$ complexity. In [3], a 'fast' (FTF-type) algorithm is substituted for the $O(m^2)$ part, to further reduce complexity.

4. LS LATTICE BASED FAST NEWTON

The $(m+1)$-dimensional triangular part at the top in Figure 5 implements an m-th order backward prediction filter (as indicated), producing an (energy normalized) backward prediction error. This residual is then fed into the delay line at the left. It is also seen that the shaded multiply-add chain in the 4th column implements a filter with coefficients which are -up to a time delay- equal to the backward prediction filter coefficients, but in reverse order. Reversing the order of the coefficients is known to map the backward prediction filter into the corresponding forward prediction filter (upon convergence) [1].

A least squares lattice (LSL) is known to produce forward as well as backward residuals, albeit without explicitly computing the filter coefficients [1]. The middlemost part of the Kalman Gain vector may therefore be computed using the LSLs as indicated in **Figure 6** : First, a LSL is substituted for the topmost $(m+1$-dimensional) triangular part of Figure 5, which indeed produces the same backward residual (together with a forward residual, which is ignored). The lattice filter coefficients are then copied into a second lattice filter, which is fed with the m-th order backward residuals, and which produces (at the 'forward residual output', *cfr.* reversed coefficients) the signal needed for the Kalman gain vector computation. The quantity δ turns out to be a conversion parameter between apriori and aposteriori values and is not needed if aposteriori residuals are avialable from the LSLs.

The LSL-based scheme of Figure 6 is an *approximate* scheme, not only because it is based on the Fast Newton approximation, but in addition because of the following :

- The 'original' forward and backward prediction filters (shaded in Figure 5) have the same set of coefficients only *up to time delays* and provided the LSL has converged. The LSL-scheme therefore produces acceptable approximations only in the *slowly time-varying* case (approximately time-invariant in an interval of length $O(m)$) after the LSL is close to convergence.
- An estimate of the middlemost part of the Kalman gain vector is produced. However, the missing parts are not required if the adaptive filtering problem is suitably *re-formulated* such that the corresponding weights are (approximately) zero and thus need not be calculated. This can be achieved by embedding the p-th order filtering problem into a larger $p+2m+1$ problem, of which the first $m+1$ and last m coefficients will be ignored. Figure 6 shows a 4th order problem. Note that the top-left input signal then has to be $u(k+m+1)$ instead of $u(k)$. In other words, the algorithm has a latency of $m+1$ time samples.

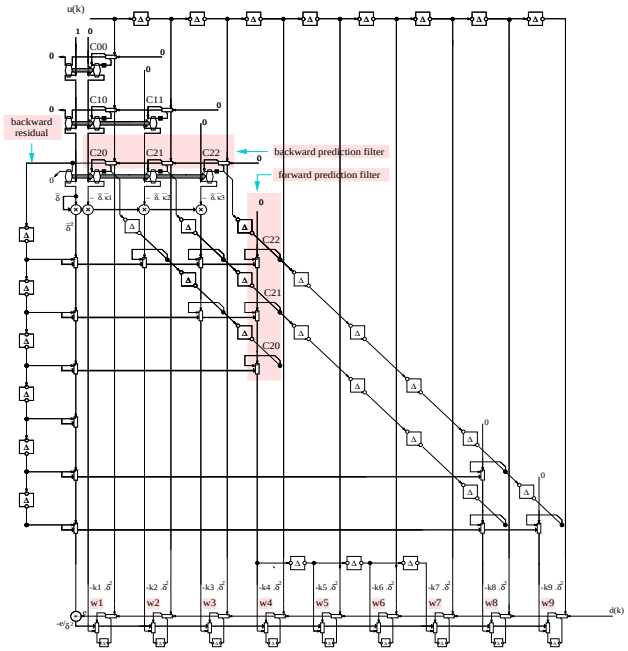

Figure 5. Fast Newton-type algorithm

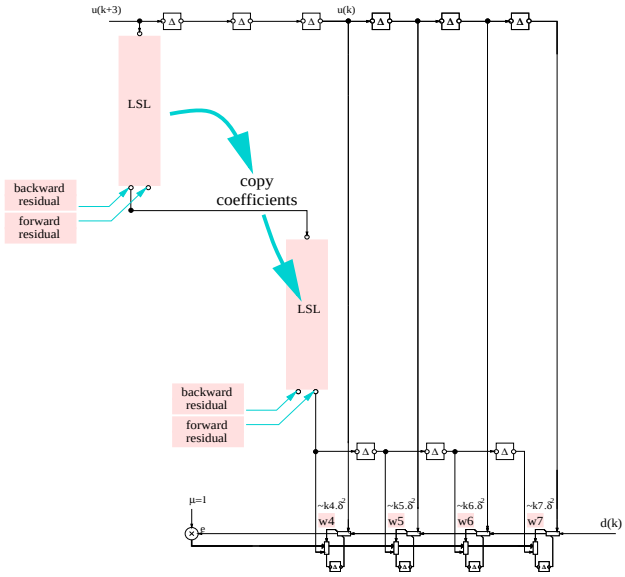

Figure 6. LSL-based Fast Newton algorithm

5. SIMULATIONS

Figure 7 shows the convergence rate of the LSL Fast Newton-type adaptive filtering algorithm. For comparison, the performance of the 'forward-backward' Fast Netwon algorithm [3], the RLS and the normalised LMS algorithms are also shown. The scenario is that of a system identification problem. The target system is 20th order FIR filter (p=20) with an input signal derived from a 6th order AR process. There is additive noise of amplitude 10^{-7}. The Fast Newton-type algorithms both use the correct AR order (m=6) and a forget factor of 0.983. The data matrix had a condition number of 55. In order to avoid initialisation problems, the algorithms were run until they converged and then the FIR filter was changed (at t= 1200). The plot in figure 7 shows the various apriori estimation errors averaged over 5 Monte Carlo runs. It is clear that both Fast Newton-type algorithms perform as well as the exact RLS algorithm. The LMS algorithm is slow to converge because of the high condition number.

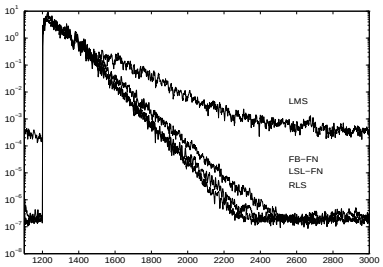

Figure 7. Convergence rate

6. ACKNOWLEDGEMENTS

Marc Moonen is a Research Associate with the F.W.O.-Vlaanderen (Flemish Fund for Science and Research). His research was carried out in the frame of the Concerted Research Action MIPS ('Model-based Information Processing Systems') of the Flemish Government and the IT-project ITA/GBO/T23 of the Flemish I.W.T. ('Integrating Signal Processing Systems'). This work was undertaken whilst Ian Proudler was a Visiting Fellow (grant no. F/96/41) at K.U. Leuven.

REFERENCES

[1] S. Haykin, *Adaptive Filter Theory*, 2nd Edition, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[2] M. Moonen, J.G McWhirter, *A systolic array for recursive least squares by inverse updating.* Electronics Letters, Vol. 29 (1993), No. 13, pp 1217-1218.

[3] G.V. Moustakides and S. Theodoridis,"Fast Newton Transversal Filters - A New Class of Adaptive Estimation Algorithms", IEEE Trans. SP-39(6), p.2184-2193.

[4] C.T. Pan, R.J. Plemmons, *Least squares modifications with inverse factorization: parallel implications.* J. Computational and Applied Mathematics, Vol. 27, No. 1-2, 1989, pp. 109-127.

[5] I.K. Proudler, J.G. McWhirter, M. Moonen, G. Hekstra, *Formal derivation of a systolic array for recursive least squares estimation.* IEEE Trans. CAS II, Vol. 43, Nr. 3, March 1996, pp 247-254.