RSRE
MEMORANDUM No. 4562

ROYAL SIGNALS & RADAR ESTABLISHMENT

QR DECOMPOSITION BASED ALGORITHMS AND ARCHITECTURES FOR LEAST-SQUARES ADAPTIVE FILTERING

Authors: I K Proudler & J G McWhirter

PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
RSRE MALVERN,
WORCS.

RSRE MEMORANDUM No. 4562

CONDITIONS OF RELEASE

0120146 308888

..... DRIC U

..... DRIC Y

Reports quoted are not necessarily available to members of the public or to commercial organisations.

DEFENCE RESEARCH AGENCY

RSRE Memorandum 4562

TITLE: QR DECOMPOSITION BASED ALGORITHMS AND ARCHITECTURES FOR LEAST-SQUARES ADAPTIVE FILTERING.

AUTHORS: I K Proudler and J G McWhirter.

DATE: January 1992

SUMMARY

In this memorandum we show how the method of QR decomposition (QRD) may be applied to the adaptive filtering and beamforming problems. QR decomposition is a form of orthogonal triangularisation which is particularly useful in least squares computations and forms the basis of some very stable numerical algorithms. When applied to the problem of narrowband adaptive beamforming where the data matrix, in general, has no special structure, this technique leads to an architecture which can carry out the required computations in parallel using a triangular array of relatively simple processing elements.

The problem of an adaptive time series filter is also considered. Here the data vectors exhibit a simple time-shift invariance and the corresponding data matrix is of Toeplitz structure. In this case, the triangular processor array is known to be very inefficient. Instead, it is shown how the Toeplitz structure may be used to reduce the computational complexity of the QR decomposition technique. The resulting orthogonal least squares lattice and "fast Kalman" algorithms may be implemented using far fewer processing elements. These "fast" QRD algorithms are very similar to the more conventional ones but, in general, are found to have superior numerical properties.

The more general problem of multi-channel adaptive filtering which arises, for example, in broadband adaptive beamforming can also be solved using the QRD-based least-squares minimisation technique. In this case the data matrix has a block Toeplitz structure which may be exploited to generate an efficient multi-channel fast Kalman or least squares lattice algorithm. The multi-channel least squares lattice algorithm may be implemented using a lattice of the triangular processor arrays and so it constitutes a hybrid solution which encompasses the algorithms and architectures of narrowband adaptive beamforming and adaptive filtering as special cases.

INTENTIONALLY BLANK

CONTENTS

Figures

Tables

INTENTIONALLY BLANK

1 Introduction.

In this memorandum we will show how the method of QR decomposition (QRD) may be applied to the adaptive filtering and beamforming problems. QR decomposition is a form of orthogonal triangularisation which is particularly useful in least squares computations and forms the basis of some very stable numerical algorithms.

In section 2, we show how the method of QR decomposition by Givens rotations may be applied to the problem of narrowband adaptive beamforming where the data matrix, in general, has no special structure. In particular, it is shown how the least squares computation may be carried out in parallel using a triangular array of relatively simple processing elements.

In section 3, we consider the problem of an adaptive time series filter for which the data vectors exhibit a simple time-shift invariance and the corresponding data matrix is of Toeplitz structure. In this case, the triangular processor array described in section 2 is known to be very inefficient. Instead, it is shown how the Toeplitz structure may be used to reduce the computational complexity of the QR decomposition technique. The resulting orthogonal least squares lattice and "fast Kalman" algorithms, derived in section 3, may be implemented using far fewer processing elements. These "fast" QRD algorithms are very similar to the more conventional ones [12] but, in general, are found to have superior numerical properties.

In section 4, we consider the more general problem of multi-channel adaptive filtering which arises, for example, in broadband adaptive beamforming. In this case the data matrix has a block Toeplitz structure which may be exploited to generate an efficient multi-channel fast Kalman or least squares lattice algorithm. The multi-channel least squares lattice algorithm may be implemented using a lattice of the triangular processor arrays discussed in section 2 and so it constitutes a hybrid solution which encompasses the algorithms and architectures of sections 2 and 3 as special cases. The appendix contains listings of the various algorithms in an ALGOL-like code.

The content of this memorandum appears, with slight modifications, as chapter 7 - "The QR Family" - of the book entitled "Adaptive System Identification and Signal Processing Algorithms", edited by S. Theodoridis and N. Kaloupsidis, and published by Prentice-Hall.

2 Narrow-band Beamforming.

2.1 QR Decomposition

A narrow-band beamformer is essentially a spatial filter and the corresponding adaptive beamforming problem may be formulated in terms of least squares minimisation [30]. A fundamental structure in least squares minimisation problems is an adaptive linear combiner of the type illustrated in figure 1. This may be applied directly to the problem of narrowband adaptive beamforming, resulting in the so-called generalised sidelobe canceller [30]. The combined output from an adaptive linear combiner at sample time t_1 is denoted by

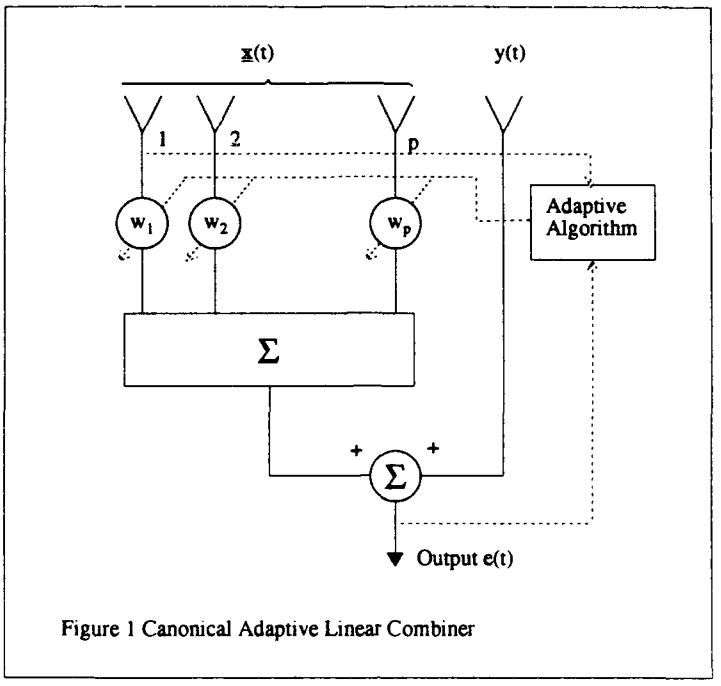


Figure 1 Canonical Adaptive Linear Combiner

$$e(t) = \underline{x}^T(t_i)\underline{w} + y(t_i) \tag{1}$$

where $\underline{x}(t_i)$ is the p-element (complex) vector of auxiliary signals at time t_i and $y(t_i)$ is the corresponding sample of the primary signal. The residual signal power at time n is estimated by the quantity

$$E_n(\underline{w}) = |e(n)|^2 \tag{2}$$

where
$$\underline{e}(n) = B(n)\,[e(1), e(2), \ldots, e(n)]^T \tag{3}$$

and the diagonal matrix

$$B(n) = \text{diag}\,[\beta^{n-1}, \beta^{n-2}, \ldots, 1] \tag{4}$$

constitutes an exponential window with $0 < \beta \leq 1$. From equation (1) it follows that the vector of residuals may be written in the form

$$\underline{e}(n) = X(n)\underline{w} + \underline{y}(n) \tag{5}$$

where

$$X(n) = B(n)\begin{bmatrix} \underline{x}^T(1) \\ \underline{x}^T(2) \\ \vdots \\ \underline{x}^T(n) \end{bmatrix} \quad \text{and} \quad \underline{y}(n) = B(n)\begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(n) \end{bmatrix} \tag{6}$$

X(n) is simply the matrix of all data samples received by the combiner up to time n, and $\underline{y}(n)$ is the corresponding vector of data in the primary or reference channel. For convenience, the matrix B(n) has simply been absorbed into the definition of $\underline{e}(n)$, $\underline{y}(n)$ and X(n).

The least squares weight vector $\underline{w}(n)$ is simply the one which minimises $E_n(\underline{w})$ and the conventional approach [12] to this problem involves explicit computation of the data covariance matrix $M(n) = X^H(n)X(n)$ and, as a result, the condition number of the problem is squared. For a given finite wordlength, this leads to a considerable loss in performance and should be avoided if possible. Note that the computation of $\underline{w}(n)$ is based on all data received up to time n.

An alternative approach to the least-squares estimation problem is the method of QR decomposition which constitutes a form of orthogonal triangularisation and has particularly good numerical properties. It will be generalised here to the case of complex data as required, for example, in narrowband adaptive beamforming. An $(n \times n)$ unitary matrix Q(n) is generated such that

$$Q(n)X(n) = \begin{bmatrix} R(n) \\ O \end{bmatrix} \tag{7}$$

where R(n) is a $p \times p$ upper triangular matrix. Then, since Q(n) is unitary, we have

$$M(n) = X^H(n)X(n) = X^H(n)Q^H(n)Q(n)X(n) = R^H(n)R(n) \tag{8}$$

so that the triangular matrix R(n) is the Cholesky (square-root) factor of the data covariance matrix M(n).

Now, again by virtue of the unitary nature of the matrix Q(n), we have

$$\|\underline{e}(n)\| = \|Q(n)\underline{e}(n)\| = \left\| \begin{bmatrix} R(n) \\ O \end{bmatrix} \underline{w}(n) + \begin{bmatrix} \underline{u}(n) \\ \underline{v}(n) \end{bmatrix} \right\| \tag{9}$$

where

$$\begin{bmatrix} \underline{u}(n) \\ \underline{v}(n) \end{bmatrix} = Q(n)\underline{y}(n) \tag{10}$$

It follows that the least squares weight vector $\underline{w}(n)$ must satisfy the equation

$$R(n)\underline{w}(n) + \underline{u}(n) = \underline{o} \tag{11}$$

and hence

$$E_n(\underline{w}) = ||\underline{v}(n)||^2 \tag{12}$$

Since the matrix R(n) is upper triangular, equation (11) is much easier to solve than the Gauss normal equations [12]. The weight vector $\underline{w}(n)$ may be derived quite simply by a process of back-substitution. Equation (11) is also much better conditioned since the condition number of R(n) is identical to that of the original data matrix X(n), the two being related by a unitary transformation.

2.2 Givens Rotations

The triangularisation process may be carried out using either Householder transformations[10] or Givens rotations[9]. However, the Givens rotation method is particularly suitable for adaptive filtering since it leads to a very efficient algorithm whereby the triangularisation process is recursively updated as each new row of data enters the combiner. Assume that the matrix X(n-1) has already been reduced to triangular form by the unitary transformation

$$Q(n-1)X(n-1) = \begin{bmatrix} R(n-1) \\ O \end{bmatrix} \tag{13}$$

and define the unitary matrix

$$\bar{Q}(n-1) = \begin{bmatrix} Q(n-1) & o \\ o^T & 1 \end{bmatrix} \tag{14}$$

Clearly,

$$\bar{Q}(n-1)X(n) = \bar{Q}(n-1)\begin{bmatrix} \beta X(n-1) \\ x^T(n) \end{bmatrix} = \begin{bmatrix} \beta R(n-1) \\ O \\ x^T(n) \end{bmatrix} \tag{15}$$

and hence the triangularisation of X(n) may be completed by using a sequence of (complex) Givens rotations to eliminate the vector $\underline{x}^T(n)$. Each Givens rotation is an elementary unitary transformation of the form

$$\begin{bmatrix} c & s^* \\ -s & c \end{bmatrix}\begin{bmatrix} 0 \ldots 0, \beta r_i \ldots \beta r_k \ldots \\ 0 \ldots 0, x_i \ldots x_k \ldots \end{bmatrix} = \begin{bmatrix} 0 \ldots 0, r_i' \ldots r_k' \ldots \\ 0 \ldots 0, 0 \ldots x_k' \ldots \end{bmatrix} \tag{16}$$

where

$$c^2 + |s|^2 = 1 \tag{17}$$

and the cosine parameter is assumed to be real without loss of generality. Clearly we require

$$c = \frac{\beta r_i}{\sqrt{\beta^2 r_i^2 + |x_i|^2}} \qquad \text{and} \qquad s = \frac{x_i}{\sqrt{\beta^2 r_i^2 + |x_i|^2}} \tag{18}$$

The sequence of rotations is applied as follows. The p element vector $\underline{x}^T(n)$ is rotated with the first row of $\beta R(n-1)$ so that the leading element of $\underline{x}^T(n)$ is eliminated, and a reduced vector $\underline{x}'^T(n)$ is produced. Note that the first row of $R(n-1)$ is modified in the process. The (p-1)-element reduced vector $\underline{x}'^T(n)$ is then rotated with the second row of $\beta R(n-1)$ so that the leading element of $\underline{x}'^T(n)$ is eliminated, and so on until every element of the data vector has been annihilated. The resulting triangular matrix $R(n)$ then corresponds to a complete triangularisation of the matrix $X(n)$ as defined in equation (7). The corresponding unitary matrix $Q(n)$ is simply given by the recursive expression

$$Q(n) = \hat{Q}(n)\bar{Q}(n-1) \tag{19}$$

where $\hat{Q}(n)$ is a unitary matrix representing the sequence of Givens rotation operations described above, i.e.

$$\hat{Q}(n)\begin{bmatrix} \beta R(n-1) \\ O \\ \underline{x}^T(n) \end{bmatrix} = \begin{bmatrix} R(n) \\ O \\ \underline{o}^T \end{bmatrix} \tag{20}$$

It is not difficult to deduce in addition that

$$\hat{Q}(n)\begin{bmatrix} \beta \underline{u}(n-1) \\ \beta v(n-1) \\ y(n) \end{bmatrix} = \begin{bmatrix} \underline{u}(n) \\ \beta \underline{v}(n-1) \\ \alpha(n) \end{bmatrix} = \begin{bmatrix} \underline{u}(n) \\ y(n) \end{bmatrix} \tag{21}$$

and this shows how the vector $\underline{u}(n)$ can be updated recursively using the same sequence of Givens rotations. The least squares weight vector $\underline{w}(n)$ may then be derived by solving equation (11). The solution is not defined, of course, if $n < p$ but the recursive triangularisation process may, nonetheless, be initialised by setting $R(0) = O$ and $\underline{u}(0) = \underline{o}$.

2.3 Parallel Implementation

The Givens rotation algorithm described above may be implemented in parallel using a triangular processor array of the type illustrated in figure 2 for the case $p = 4$. It comprises three distinct sections - the basic triangular array labelled ABC, the right hand column of cells labelled DE and the final processing cell labelled F.

At time n-1, each cell within the basic triangular array stores one element of the triangular matrix

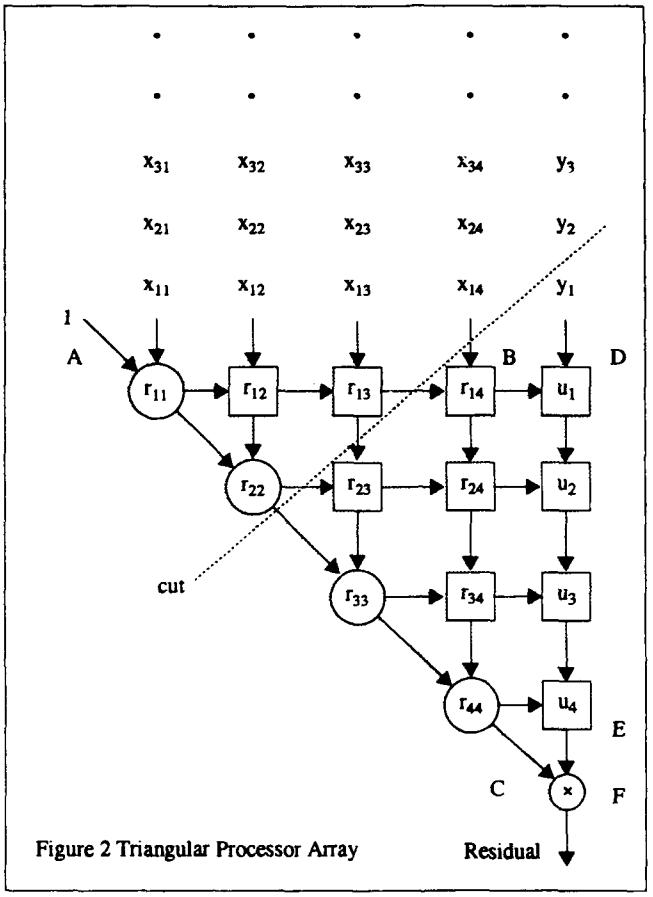


Figure 2 Triangular Processor Array

R(n – 1) and each cell in the right hand column DE stores one element of the corresponding vector $\underline{u}(n-1)$. On the next clock cycle, the data vector $[\underline{x}^T(n), y(n)]$ is input to the top of the array as shown. Each row of cells within the basic triangular array performs a Givens rotation between one row of the stored triangular matrix and a vector of data received from above so that the leading element of the received vector is eliminated. The reduced data vector is then passed downwards through the array. The boundary cell in each row computes the appropriate rotation parameters and passes them on to the right so that the internal cells can apply the same rotation to all other elements in the received data vector. This arrangement ensures that as the input data vector $\underline{x}^T(n)$ moves down through the array it interacts with the previously stored triangular matrix R(n – 1) and undergoes the sequence of rotations $\hat{Q}(n)$ described in section 2.2. All of its elements are thereby eliminated (one on each row of the array) and the updated triangular matrix R(n) is generated and stored in the process.

As the corresponding input sample y(n) moves down through the right hand column of cells, it undergoes the same sequence of Givens rotations interacting with the stored vector $\underline{u}(n-1)$ and thereby

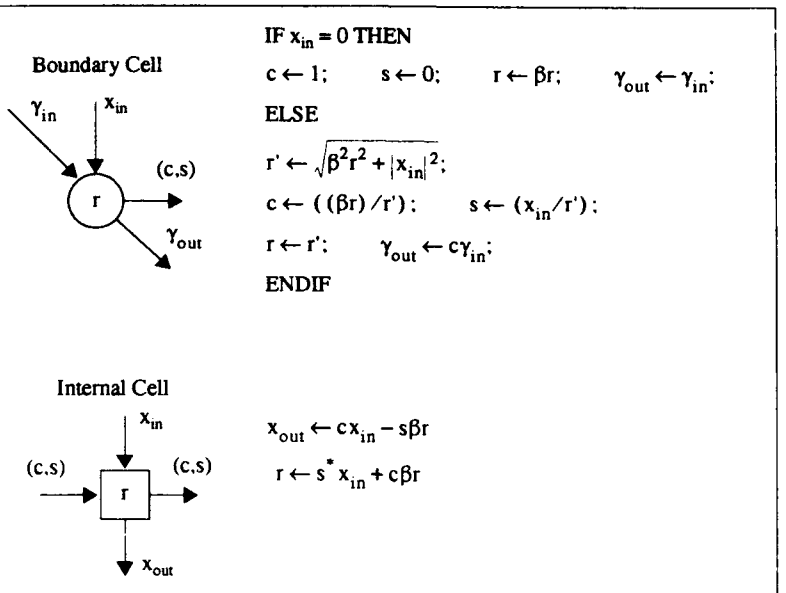


Figure 3 Processing Elements for Triangular QRD Array

generating the updated vector $\underline{u}(n)$. The resulting output, which emerges from the bottom cell in the right hand column, is simply the value of the parameter $\alpha(n)$ in equation (21).

The function of the rotation cells is specified in figure 3 and follows immediately from equations (16) and (18). The boundary cell includes an additional parameter γ which is not required for the basic Givens rotation but will be explained in section 2.5 where the function of the final cell F is also made clear. Since the matrix R is initialised to zero, it can be seen that the elements on its leading diagonal (i.e. the values of r within the boundary cells) may be treated as real variables throughout the computation and so the number of real arithmetic operations performed within the boundary cell is, surprisingly, less than that required for an internal cell.

For simplicity and ease of discussion, we have assumed that all cells of the array in figure 2 operate on the same input data vector during a given clock cycle. The critical path for the array therefore involves 2p+1 cells and the maximum rate at which data vectors can be input is ~ 1/((2p+1)T) where T is the typical processing time for a single cell. When Gentleman and Kung[8] first proposed the triangular array in figure 2, they showed how it could be fully pipelined by introducing latches to store the outputs generated by each cell before they are passed on as inputs to the neighbouring cells. The resulting systolic array can achieve an input clock rate ~ 1/T and only requires nearest neighbour cell interconnections which is highly advantageous for VLSI implementation.

The systolic array may be generated by cutting the diagram in figure 2 along all diagonals parallel

to the one indicated and introducing a storage element where each data path crosses a cut line. Note that these cut lines also intersect the input data paths and so each input data vector enters the triangular array in a skewed or staggered manner. Clearly, figure 2 provides a sufficient description of the parallel algorithm without including the detailed pipelining and timing aspects associated with a systolic or wavefront array. This relative simplicity means that the parallel processor can be represented more easily in block diagrammatic form later in this memorandum.

2.4 Square-Root-Free Version

Gentleman[7] and Hammarling[11] have derived extremely efficient QR decomposition algorithms based on modified Givens rotations which require no square-root operation. The essence of these square-root-free algorithms is a factorisation of the form

$$R(n) = D^{1/2}(n)\bar{R}(n) \tag{22}$$

where

$$D(n) = \text{diag}\left[r_{11}^2(n), r_{22}^2(n), \ldots, r_{pp}^2(n)\right] \tag{23}$$

and $\bar{R}(n)$ is a unit upper triangular matrix. The complex Givens rotation in equation (16) then takes the form

$$\begin{bmatrix} c & s^* \\ -s & c \end{bmatrix} \begin{bmatrix} 0 \ldots 0, \beta\sqrt{d} \ldots \beta\sqrt{d}\bar{r}_k \ldots \\ 0 \ldots 0, \sqrt{\delta}\bar{x}_1 \ldots \sqrt{\delta}\bar{x}_k \ldots \end{bmatrix} = \begin{bmatrix} 0 \ldots 0, \sqrt{d'} \ldots \sqrt{d'}\bar{r}_k' \ldots \\ 0 \ldots 0, 0 \ldots \sqrt{\delta'}\bar{x}_k' \ldots \end{bmatrix} \tag{24}$$

where $\sqrt{d}$ represents an element in the diagonal matrix $D^{1/2}(n)$, and $\bar{r}_k$ denotes an associated off-diagonal element from the matrix $\bar{R}(n)$. Note that each element x_k of the data vector has been expressed in the form $\sqrt{\delta}\bar{x}_k$ where δ is a scaling factor which changes value as a result of the rotation.

The square-root-free algorithm may also be implemented using the type of triangular processor array depicted in figure 2 but with the rotation cells as defined in figure 4. In this case, the boundary cells store and update elements of D(n) (i.e. the squares of the diagonal elements of R(n)) while the internal cells store and update the off-diagonal elements of $\bar{R}(n)$. The function of these cells may be deduced in a straightforward manner starting from equations (24) and (17) and noting that the values of c and s need not be computed explicitly. Note that the scale quantities δ are updated only at the boundary cells and passed diagonally from one row to the next. Note also that the parameters d and δ in each boundary cell may be treated as real variables throughout the computation assuming that they are assigned real values initially. For the purposes of normal least squares processing, the δ parameter is initialised to unity so that on input to the array, $\bar{x}(n) = x(n)$. However, as pointed out by Gentleman[7], the δ parameter associated with each input vector may be assigned an arbitrary initial value which serves to weight that row of data accordingly. The square-root-free array may thus be used to perform a general weighted least squares computation.

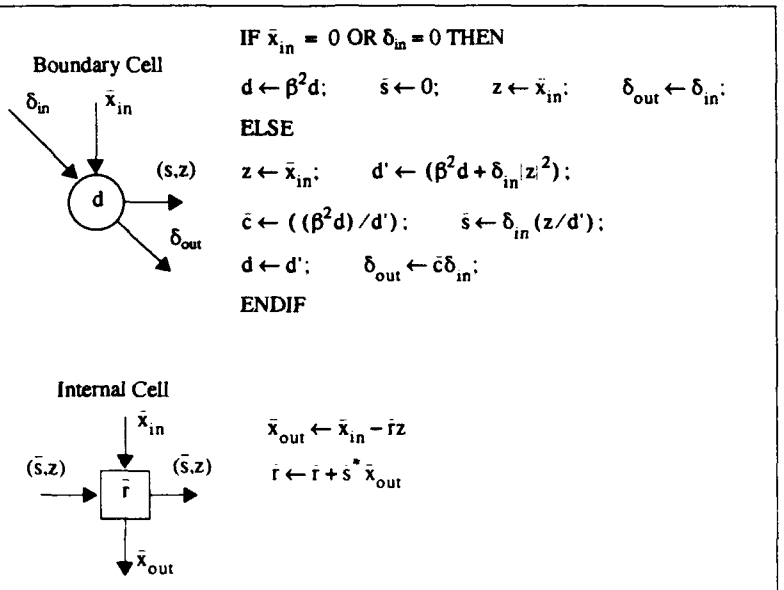


Figure 4 Square-root-free Processing Elements

As with the conventional Givens rotation algorithm, cells in the right hand column perform the same function as those internal to the triangular array. Thus, allowing for the factorisation shown in equation (22), these cells update elements of the vector $\bar{u}(n)$, where

$$\underline{u}(n) = D^{1/2}(n)\bar{u}(n) \tag{25}$$

It follows from equations (22), (25), and (11) that the weight vector $\underline{w}(n)$ is given by

$$\bar{R}(n)\underline{w}(n) + \bar{u}(n) = \underline{o} \tag{26}$$

and may be obtained, as before, by a simple process of back substitution. Figure 4 specifies the cell functions required for one particular version of the square-root-free algorithm. This version, which requires 2 multiplications and 2 additions per cycle to be performed in each internal cell, was first suggested by Golub and reported by Gentleman[7]. We have found it to be particularly stable and accurate throughout an extensive programme of finite wordlength adaptive filtering and beamforming computations[25][32]. These observations are supported by the work of Ling, Manolakis and Proakis[15] who derived an equivalent algorithm (the "error-feedback" algorithm) based on a recursive form of the modified Gram-Schmidt orthogonalisation procedure - see section 2.8.

The "square-root-free" algorithm described above is in fact an algorithm for *calculating* the re-

Method of Calculation		Method of Application
Using square-roots (SQ)	X	Feedforward (FF)
Square-root-free (SF)		Feedback of output (FB)
		Feedback of stored parameter (MFB)

Table 1 Options for Implementing a Givens Rotation

quired planar rotation. In the QRD-based least squares minimisation problem these rotations also have to be *applied* to various vectors. In section 2.3 the "natural" implementation of the Givens rotation (involving the computation of square-roots) was applied to various vectors in a feedforward manner: the two components of the rotated vector are calculated independently on the basis of the two input components. The square-root-free algorithm presented above applies the rotation in a feedback mode: one output is now dependent on one input and the other output. It follows that there exists another feedback algorithm where the parameter that is fed back is the stored quantity $\bar{r}$ - see figure 4 - rather than the cell output $\bar{x}_{out}$.

Combined with the two methods for calculating the rotation parameters, the feedforward/feedback choice results in six different variations: see table 1. The computer simulations of section 3.8 show that, of these four possible variations, the one that is equivalent to the RMGS error-feedback algorithm (square-root-free with feedback) performs the best in terms of numerical stability. It is worth emphasising that the basic architecture (figure 2) is not affected by the choice of rotation technique so that the only difference between the various options is a change of PE's.

2.5 Direct Residual Extraction

In many least squares problems, and particularly in adaptive noise cancellation, the least-squares weight vector $\underline{w}(n)$ is not the principal object of interest. Of more direct concern is the corresponding residual, since this constitutes the noise-reduced output signal from the adaptive combiner[32]. In this section, we will show how the "a-posteriori" least squares residual

$$e(n, n) = \underline{x}^T(n)\underline{w}(n) + y(n) \tag{27}$$

which depends on the most up-to-date weight vector $\underline{w}(n)$, may be obtained directly from the type of processor array described in section 2.3 without computing $\underline{w}(n)$ explicitly[18].

In order to proceed, it is necessary to consider the structure of the $n \times n$ matrix $\hat{Q}(n)$ which, from equation (20), may be expressed in the form

$$\hat{Q}(n) = \begin{bmatrix} A(n) & O & \underline{a}(n) \\ O & I & \underline{o} \\ \underline{b}^T(n) & \underline{o}^T & \gamma(n) \end{bmatrix} \tag{28}$$

where A(n) is a p × p matrix, $\underline{a}(n)$ and $\underline{b}(n)$ are p × 1 vectors and $\gamma(n)$ is a scalar. Now $\hat{Q}(n)$ is given by

$$\hat{Q}(n) = \prod_{i=1}^{p} G_i(n) \tag{29}$$

where the G_i are elementary n×n rotation matrices of the form

$$G_i(n) = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & c_i(n) & \cdots & s_i^*(n) \\ & & & & 1 & \\ & & & & & \ddots \\ & & & & & & 1 \\ & & & -s_i(n) & \cdots & c_i(n) \end{bmatrix} \qquad (i = 1, 2, 3, \ldots p) \tag{30}$$

and all off-diagonal elements are zero except those in the (i,n) and (n,i) locations. It follows directly that

$$\gamma(n) = \prod_{i=1}^{p} c_i(n) \tag{31}$$

i.e. $\gamma(n)$ is the product of the cosine terms in the p rotations represented by $\hat{Q}(n)$. Multiplying both sides of equation (20) by $\hat{Q}^H(n)$ and noting that the matrix $\hat{Q}(n)$ is unitary, it is now easy to deduce that

$$x^T(n) = \underline{a}^H(n)R(n) \tag{32}$$

and similarly, from equation (21) we have

$$y(n) = \underline{a}^H(n)\underline{u}(n) + \gamma(n)\alpha(n) \tag{33}$$

Substituting equations (32) and (33) into equation (27) leads to the expression

$$e(n, n) = \underline{a}^H(n)R(n)\underline{w}(n) + \underline{a}^H(n)\underline{u}(n) + \gamma(n)\alpha(n) \tag{34}$$

and, from equation (11), it follows immediately that

$$e(n, n) = \gamma(n)\alpha(n) \tag{35}$$

Now, as noted in section 2.3, when the conventional Givens rotation algorithm is employed, the parameter $\alpha(n)$ is generated quite naturally within the triangularisation process and simply emerges

from the bottom cell of the right hand column in figure 2; because of its relationship with the a-posteriori residual $e(n, n)$, as given in equation (35), the parameter $\alpha(n)$ is known as the angle-normalised residual. The scalar $\gamma(n)$, as given by equation (31), may also be computed very simply. The product of cosine terms is generated recursively by the parameter γ as it passes from cell to cell along the chain of boundary processors. The simple product required to form the a-posteriori residual as given in equation (35) is computed by the final processing cell F in figure 2.

The direct residual extraction technique obviously avoids a lot of unnecessary computation provided that the weight vector is not required explicitly. As a result, the overall processing architecture is greatly simplified. There is no need for a separate back-substitution processor or any sophisticated control circuitry to ensure that the contents of the triangular array are input to the back-substitution processor in the correct sequence. Consequently, it is much easier to maintain a regular pipelined data flow. A less obvious, but arguably more important, advantage of direct residual extraction is the improved numerical stability which it offers. This derives from the fact that computing the weight vector explicitly requires the solution of a linear inverse problem which may be ill-conditioned in circumstances where the optimum weight vector is not well defined. The least squares residual, however, is always well defined and can be computed reliably. Note, for example, that the correct (zero) residual is obtained even during the first few processing cycles when the data matrix is not full rank and the corresponding weight vector cannot be uniquely defined. This type of unconditional stability, which may be contrasted with that of the traditional RLS algorithm (see Haykin [12]) and avoids the need for "persistent excitation", is extremely important in the context of real time signal processing.

The a-posteriori residual may be computed in a similar manner when the square-root-free algorithm is employed. The square-root-free algorithm delivers from the bottom cell in the right-hand column a scalar $\bar{e}(n)$ given by

$$\delta^{1/2}(n)\bar{e}(n) = \alpha(n) \tag{36}$$

where $\delta^{1/2}(n)$ is the scaling parameter appropriate to the p^{th} row at time n and it has been assumed that $\delta(n)$ is initialised to unity on input to the array. From equations (35) and (36) it follows that

$$e(n, n) = \gamma(n)\delta^{1/2}(n)\bar{e}(n) \tag{37}$$

where $\gamma(n)$ is the product of cosine terms which arise in the conventional Givens rotation algorithm. However, $\delta(n)$, as computed by the boundary processors for the square-root-free algorithm, is simply the product of all the $\bar{c}$ terms and it can easily be shown that this is equivalent to the product of the conventional cosine parameters squared. Thus,

$$\delta(n) = \gamma^2(n) \tag{38}$$

and

$$e(n, n) = \delta(n)\bar{e}(n) \tag{39}$$

Hence the a-posteriori residual e(n,n) may also be obtained directly from the square-root-free processor array, using a final multiplier cell F as illustrated in figure 2.

A second form of residual, which occurs naturally in least squares algorithms such as the least squares lattice (see Haykin [12]) is the a priori residual, denoted e(n, n – 1). Defined in terms of the previously computed weight vector $\underline{w}(n-1)$ and the latest data $[\underline{x}^T(n), y(n)]$, it takes the form

$$e(n, n-1) = \underline{x}^T(n)\underline{w}(n-1) + y(n). \tag{40}$$

This residual may also be obtained directly from the triangular processor array as we now show. By substituting the decomposition for $\hat{Q}(n)$ given in equation (28) in the time update relation (equation (20)) we have

$$\underline{b}^T(n)\beta R(n-1) + \gamma(n)\underline{x}^T(n) = \underline{o} \tag{41}$$

Eliminating the vector $\underline{x}^T(n)$ between equations (40) and (41) we find

$$e(n, n-1) = (-\underline{b}^T(n)\beta R(n-1)\underline{w}(n-1) + \gamma(n)y(n)) / \gamma(n) \tag{42}$$

Again by using the decomposition given in equation (28), this time with equation (21), an other relationship can be obtained:

$$b^T(n)\beta u(n-1) + \gamma(n)y(n) = \alpha(n) \tag{43}$$

By eliminating the term $\gamma(n)y(n)$ between equation (42) and (43) we have that

$$e(n, n-1) = (-\underline{b}^T(n)\beta R(n-1)\underline{w}(n-1) + \alpha(n) - b^T(n)\beta u(n-1)) / \gamma(n) \tag{44}$$

and, from equation (11), it follows immediately that

$$e(n, n-1) = \alpha(n)/\gamma(n) \tag{45}$$

Thus the a priori residual is computed from the same quantities as the a posteriori residual. It follows immediately from equations (35) and (45) that they are related by the expression

$$e(n, n) = \gamma^2(n)e(n, n-1) \tag{46}$$

Note that by eliminating the term $\gamma(n)$ between equations (35) and (45) we find that

$$\alpha(n) = \sqrt{e(n, n-1)e(n, n)} \tag{47}$$

so that the angle normalised residual $\alpha(n)$ can be viewed as the geometric mean of the a priori and the a posteriori residuals.

From equations (38), (39) and (46) we see that

$$\bar{e}(n) = e(n, n-1) \tag{48}$$

so that the scalar which emerges naturally from the bottom cell in the right-hand column of the square-root-free processor array is the corresponding a priori residual. Finally, note that

$$\delta(n) = \gamma^2(n) = \frac{e(n, n)}{e(n, n-1)} \tag{49}$$

and hence we see that $\delta(n)$ is equal to the so-called likelihood variable (see Haykin [12]).

2.6 Weight freezing and flushing

It has been shown that if a data vector $[\underline{x}^T(n), y(n)]$ is input to the triangular processor array in figure 2, the corresponding a-posteriori least squares residual e(n,n) emerges from the final cell F. In order to achieve this result, the array performs two distinct functions:

(1) It generates the updated triangular matrix R(n) and corresponding vector $\underline{u}(n)$ (or D(n), $\bar{R}(n)$ and $\bar{u}(n)$ for the square-root-free algorithm) and hence, implicitly, the updated weight vector $\underline{w}(n)$.

(2) It acts as a simple filter which applies the updated weight vector to the input data according to equation (27).

If the array is subsequently "frozen" in order to suppress any further update of the stored values, but allowed to function normally in all other respects, it will continue to perform the filtering operation without affecting the implicit weight vector $\underline{w}(n)$. Thus, in response to an input of the form $[\underline{x}^T, y]$, the frozen network will produce the output residual

$$e = \underline{x}^T \underline{w}(n) + y \tag{50}$$

Equation (50) may be verified directly by considering the frozen array as a combination of basic matrix operators. Consider first the basic triangular array ABC in figure 2. In frozen mode, the boundary and internal cells perform the reduced processing functions defined in figure 5. Now consider the effect of the simplified network upon a row vector $\underline{x}^T$, input from above in the usual manner. This will give rise to a column vector $\underline{z}$ which emerges from the right. It is straightforward to verify that the input vector $\underline{x}$ is related to the output vector $\underline{z}$ by means of the matrix transformation

$$\underline{x} = R^T \underline{z} \tag{51}$$

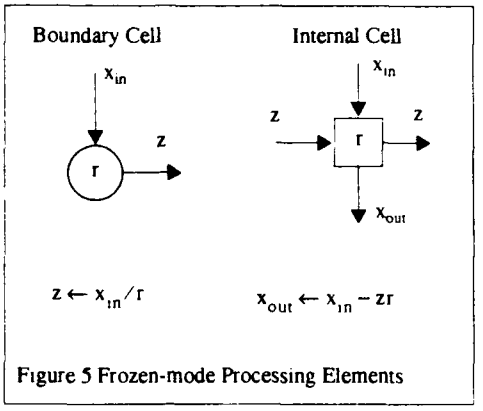


Figure 5 Frozen-mode Processing Elements

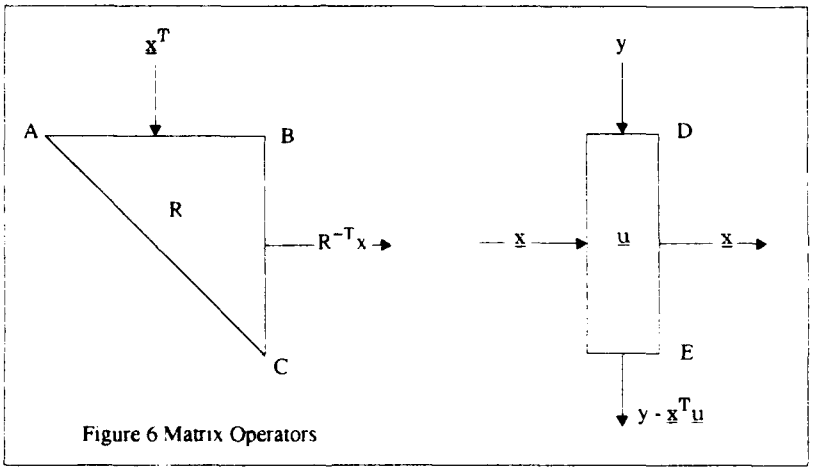


Figure 6 Matrix Operators

where R is the upper triangular matrix stored within the array. For example, it is clear that

$$z_1 = x_1/r_{11} \quad \text{and} \quad z_2 = (x_2 - r_{12}z_1)/r_{22} \tag{52}$$

i.e $$x_1 = r_{11}z_1 \quad \text{and} \quad x_2 = r_{12}z_1 + r_{22}z_2 \tag{53}$$

Assuming that R is non-singular (i.e. no diagonal element of R is zero) it follows that

$$z = R^{-T}x \tag{54}$$

and so the frozen triangular array may be regarded as an R^{-T} matrix operator as depicted schematically in figure 6.

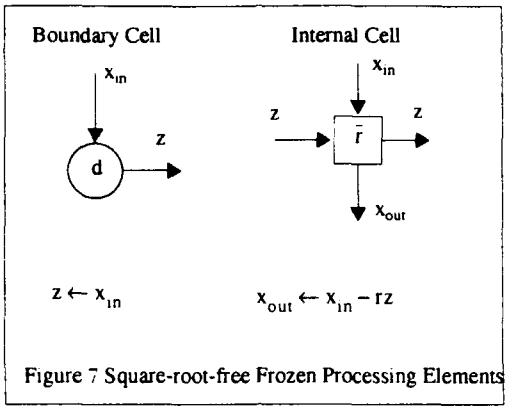


Figure 7 Square-root-free Frozen Processing Elements

Now consider the right hand column of cells DE in figure 2. It is easy to show that, in frozen mode, the effect of this array upon a column vector $\underline{x}$ input from the left and a scalar y input from the top is to produce the scalar output $y - \underline{x}^T\underline{u}$ which emerges from the bottom cell. The vector $\underline{x}$ also emerges unchanged from the right as depicted in figure 6. It follows immediately that if the network in figure 2 is frozen at time n, its effect upon a vector $[\underline{x}^T, y]$ input from the top is to produce the scalar output $y - x^T R^{-1}(n)u(n)$ which emerges from the column of internal cells DE. From equation (11), it can be seen that this is precisely the frozen residual defined in equation (50).

When the square-root-free algorithm is employed, the array in figure 2 may be frozen very simply by setting the forget factor $\beta = 1$ and initialising the parameter δ to zero for any input vector which is to be processed in the frozen mode. As pointed out in section 2.4, this has the effect of assigning zero weight to that vector within the overall least squares computation and so the processing does not affect any values stored within the array. This property can be verified quite easily by inspecting the square-root-free cells in figure 4 and the resulting operations for the frozen cells are shown in figure 7. It follows from the discussion above that if a vector $[\underline{x}^T, y]$ is input to the top of the frozen square-root-free array at time n, the output which emerges from the internal cell E will be $y - x^T R^{-1}(n)u(n)$ and from equation (26) it can be seen that this is again the frozen residual defined in equation (50). Note that the parameter δ_{in} for the final processing cell must be set equal to one to avoid suppressing the output residual from the frozen square-root-free network if this technique is used.

Having established that the function of a frozen triangular array is given by equation (50), it is easy to see how the least squares weight vector, if required, may also be obtained without performing a back-substitution. Let $\underline{h}_i$ denote the p element vector whose only non-zero element occurs in the i^{th} location: it follows that the effect of inputting the sequence of "impulse" vectors $[\underline{h}_i, 0]$ (i=1,2....p) to the frozen array is to produce the sequence of output values $w_i(n)$ (i=1,2....p) and so the weight vector $\underline{w}(n)$ may be extracted from the array without the need for any additional hardware. This technique, which amounts to measuring the impulse response of the system, is generally referred to as "weight flushing"[32].

2.7 Parallel Weight Extraction

The technique of weight flushing presented in section 2.6 requires that the adaptive filtering be temporarily suspended whilst the sequence of impulse vectors is fed into the array. A p^{th} order system would therefore have to suspend its data processing for p time instants. Although it is conceivable that this weight flushing process could be carried out at a higher clock rate, in between the processing of the data, in a high data rate environment this option is unlikely to be viable and an alternative technique is required. As we show below, the weight vector can be produced in parallel with the adaption process, in several ways, by the addition of extra hardware.

Before describing how the weight vector can be generated, we first consider an important property of the $\hat{Q}$ matrix upon which the parallel weight extraction techniques (and also some other important techniques like MVDR beamforming [20]) depend. We have seen, in section 2.2, that the matrix $\hat{Q}(n)$ updates the triangular matrix $R(n-1)$ to $R(n)$ by rotating in the new data vector at time n. Rather surprisingly, the matrix $\hat{Q}(n)$ can also be used to update the triangular matrix $R^{-H}(n-1)$ to $R^{-H}(n)$. Consider the following identity:

$$\begin{bmatrix} \beta R^{H}(n-1) & O & \underline{x}^{*}(n) \end{bmatrix} \hat{Q}^{H}(n)\hat{Q}(n) \begin{bmatrix} \beta^{-1}R^{-H}(n-1) \\ O \\ \underline{o}^{T} \end{bmatrix} = I \tag{55}$$

Let

$$\hat{Q}(n) \begin{bmatrix} \beta^{-1}R^{-H}(n-1) \\ O \\ \underline{o}^{T} \end{bmatrix} = \begin{bmatrix} T(n) \\ O \\ \underline{z}^{T}(n) \end{bmatrix} \tag{56}$$

so that, by means of equations (20) and (55), we have:

$$\begin{bmatrix} R^{H}(n) & O & \underline{o} \end{bmatrix} \begin{bmatrix} T(n) \\ O \\ \underline{z}^{T}(n) \end{bmatrix} = I \tag{57}$$

and clearly,

$$T(n) = R^{-H}(n) \tag{58}$$

Thus equation (56) becomes:

$$\hat{Q}(n) \begin{bmatrix} \beta^{-1}R^{-H}(n-1) \\ O \\ \underline{o}^{T} \end{bmatrix} = \begin{bmatrix} R^{-H}(n) \\ O \\ \underline{z}^{T}(n) \end{bmatrix} \tag{59}$$

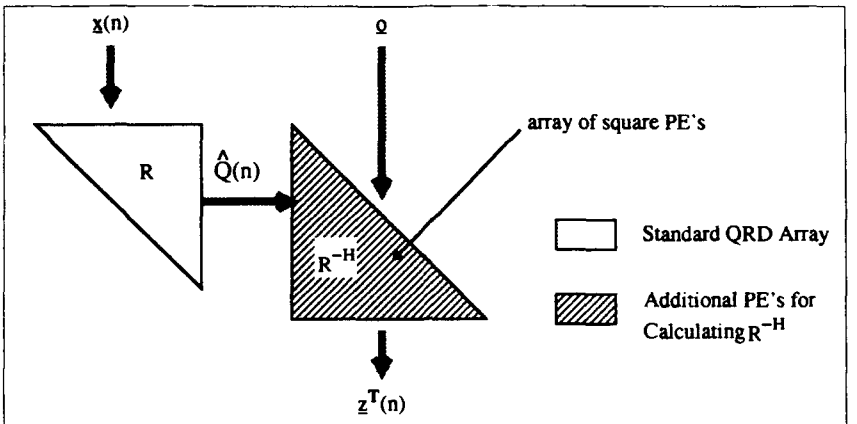


Figure 8 Updating the Inverse Triangular Matrix

and we see that the same rotations that update the matrix R also update the matrix R^{-H}.

Recall from the description of the QRD systolic array in section 2.3 that the sine and cosine parameters that define the rotation matrix $\hat{Q}$ are passed from left to right across the array. Thus the matrix R^{-H} can be calculated by appending an array of internal processing elements to the right of the basic QRD array: the QRD array stores R and outputs the rotation parameters that specify $\hat{Q}$ as R is updated from time instant to time instant; the rotation parameters are then used by the new array which stores and updates the matrix R^{-H} by rotating it against a vector of zeros. The matrix R^{-H} is lower triangular and hence the new array will also be lower triangular (see figure 8).

Armed with this property of the $\hat{Q}$ matrix, we can now show how the weight vector can be calculated in parallel with the adaption process [29]. Consider the following augmented data matrix:

$$X_+(n) = \begin{bmatrix} X(n) & y(n) \end{bmatrix} \tag{60}$$

The upper-triangular matrix resulting from a QR decomposition will then have the form:

$$R_+(n) = \begin{bmatrix} R(n) & u(n) \\ 0^T & \varepsilon(n) \end{bmatrix} \tag{61}$$

where $\varepsilon(n) = \|v(n)\|$ is the square-root of the filtering error power for y(n). It is easy to show that the inverse of an upper triangular matrix is another upper triangular matrix; thus let

$$R_+^{-1}(n) = \begin{bmatrix} T'(n) & s(n) \\ 0^T & t(n) \end{bmatrix} \tag{62}$$

where $T'(n)$ is a $(p+1)\times(p+1)$ upper triangular matrix, $\underline{s}(n)$ is a p-dimensional vector, and $t(n)$ is a scalar. Then

$$I = R_+^{-1}(n)R_+(n) = \begin{bmatrix} T'(n) & \underline{s}(n) \\ \underline{o}^T & t(n) \end{bmatrix} \begin{bmatrix} R(n) & \underline{u}(n) \\ \underline{o}^T & \varepsilon(n) \end{bmatrix} = \begin{bmatrix} T'(n)R(n) & T'(n)\underline{u}(n) + \varepsilon(n)\underline{s}(n) \\ \underline{o}^T & t(n)\varepsilon(n) \end{bmatrix} \tag{63}$$

$$\text{hence} \qquad R_+^{-1}(n) = \begin{bmatrix} R^{-1}(n) & -\varepsilon^{-1}(n)R^{-1}(n)\underline{u}(n) \\ \underline{o}^T & \varepsilon^{-1}(n) \end{bmatrix} = \begin{bmatrix} R^{-1}(n) & \varepsilon^{-1}(n)\underline{w}(n) \\ \underline{o}^T & \varepsilon^{-1}(n) \end{bmatrix} \tag{64}$$

where we have used equation (11) to identify the presence of the least squares weight vector $\underline{w}(n)$. Thus we can obtain the weight vector $\underline{w}(n)$ directly from the right-hand column of the inverse of the augmented triangular matrix $R_+(n)$ (or equivalently, the complex conjugate of the bottom row of $R_+^{-H}(n)$).

The matrix $R_+^{-H}(n)$ can be calculated as shown in figure 8 provided we ensure that the QRD array, on the left-hand side of figure 8, is solving the augmented problem. This requires that the data vector being fed into the array, at time n, is $\underline{x}^T(n)\ y(n)$ and hence that the QRD array is now of dimension $(p+1)\times(p+1)$. The adaptive filtering residual is still available with this architecture since the $(p+1)\times(p+1)$ QRD array can be thought of as consisting of a p^{th} order adaptive filter processor (as shown in figure 2) with the multiplier cell (F in figure 2) replaced by an additional circular processing element. Clearly this additional circular processing element can be modified, if necessary, to calculate the adaptive filtering residual.

The parallel weight extraction technique described above is based on the fact that the rotation matrix $\hat{Q}(n)$, as calculated by the basic QRD array on the left-hand side of figure 8, can be used to update the matrix $R^{-H}(n-1)$. An alternative technique for generating $\hat{Q}(n)$ [1][21][22] leads to a different architecture for calculating the optimum weight vector. This technique relies on the fact that the rotation matrix $\hat{Q}(n)$ is completely specified by the relevant p rotation angles (see section 2.2) and hence may be reconstructed from knowledge of these angles. As we show below, the relevant angles can be recovered from the bottom row of the matrix $\hat{Q}(n)$; further more, this row vector can itself be calculated, indirectly, based on knowledge of the matrix $R^{-H}(n-1)$.

From equations (28) and (29), we have

$$\hat{Q}^T(n)\pi_n = \begin{bmatrix} b(n) \\ \underline{o} \\ \gamma(n) \end{bmatrix} \tag{65}$$

$$= \left[-s_1^* \prod_{i=2}^{p} c_i, \ \ldots \ -s_{p-1}^* c_p, \ -s_p^*, \ o^T, \ \prod_{i=1}^{p} c_i \right]^T \tag{66}$$

where the s_i's and c_i's are the sines and cosines of the relevant Givens rotations that compose $\hat{Q}(n)$ and the "pinning" vector π_n is an n-dimensional vector such that

$$\pi_n = [0, 0 \ \ldots \ 0, 1]^T \tag{67}$$

These sines and cosines uniquely determine the matrix $\hat{Q}(n)$ and can be recovered from $\underline{b}(n)$ and $\gamma(n)$ in the following orthogonal manner. A sequence of Givens rotations is used to successively annihilate elements of $\underline{b}(n)$ starting from the top and working down until the pinning vector π_n is produced (see equation (65)).

Let $\gamma = \prod_{i=2}^{p} c_i$, then the first rotation takes the form

$$\begin{bmatrix} c & s^* \\ -s & c \end{bmatrix} \begin{bmatrix} -\gamma s_1^* \\ \gamma c_1 \end{bmatrix} = \begin{bmatrix} 0 \\ g_1 \end{bmatrix} \tag{68}$$

and it is easy to show that

$$g_1 = \pm\gamma \qquad c = \pm c_1 \qquad s = \pm s_1 \tag{69}$$

The ambiguity of sign is immediately resolved by recalling that in the construction of $\hat{Q}_p(n)$ the rotation angle is always chosen such that the cosine is positive. Having applied the first rotation to determine s_1 and c_1, the vector in equation (66) takes the form:

$$\left[0, \ -s_2^* \prod_{i=3}^{p} c_i, \ \ldots \ -s_{p-1}^* c_p, \ -s_p^*, \ o^T, \ \prod_{i=2}^{p} c_i \right]^T \tag{70}$$

and so the next Givens rotation serves to compute s_2 and c_2 in a similar manner, and so on. Note that it is also possible to recover the rotation angles that define $\hat{Q}_p(n)$ from its right-hand column. In particular, we have

$$\hat{Q}_p(n)\pi_n = \begin{bmatrix} \underline{a}(n) \\ \underline{0} \\ \gamma(n) \end{bmatrix} = \begin{bmatrix} s_1 \\ c_1 s_2 \\ \vdots \\ s_p \prod_{i=1}^{p-1} c_i \\ \underline{0} \\ \prod_{i=1}^{p} c_i \end{bmatrix} \tag{71}$$

so that the sines and cosines may be recovered by applying a sequence of Givens rotations which annihilated the elements of the vector $\underline{a}(n)$ starting from the bottom and working upwards. In this case the pair (s_p, c_p) are generated first and (s_1, c_1) last.

Having established that the matrix $\hat{Q}(n)$ can be reconstructed from knowledge of the quantities $\underline{b}(n)$ and $\gamma(n)$, we now proceed to shown how these two quantities can be calculated other than from $\hat{Q}(n)$ itself. Note from equation (41) that

$$\underline{b}(n) = -\beta^{-1}\gamma(n)R^{-T}(n-1)\underline{x}(n) \tag{72}$$

Thus given $R^{-T}(n-1)$ and the new data at time n ($\underline{x}(n)$), we may use equation (72) to calculate the vector $\underline{b}(n)/\gamma(n)$. The value of $\gamma(n)$ can easily be found from the fact that $\hat{Q}(n)$ is orthonormal and hence

$$\underline{b}(n)^2 + \gamma^2(n) = 1 \tag{73}$$

or

$$\gamma(n) = [\,\underline{b}(n)/\gamma(n)^2 + 1\,]^{-1/2} \tag{74}$$

Having calculated $\underline{b}(n)$ and $\gamma(n)$ by this indirect method, we can then calculate $\hat{Q}(n)$ as above and proceed to update the matrix R^{-H}, without the need to explicitly calculate R. Clearly, the augmented data matrix $R_+^{-H}(n)$ can also be updated in this way (with suitably redefined quantities) thus allowing the weight vector $\underline{w}(n)$ to be calculated without the need for the triangular array that performs the QR decomposition of the augmented data matrix. Once the optimum weight vector is known, the adaptive filtering residual could of course be calculated using equation (27) but, since the weight vector may not be well defined this method is less robust than the direct extraction one (see section 2.5).

Another method for extracting the weight vector without interfering with the adaption process also relies on the structure of the $\hat{Q}$ matrix. From equation (72) we see that if a vector $\underline{x}^T(n)$ is annihilated

by Givens rotations against a triangular matrix $\beta R(n-1)$ then the bottom row of the $\hat{Q}$ matrix contains the term $\beta^{-1}R^{-T}(n-1)x(n)$. Now from equation (11) we have that:

$$\underline{w}(n) = -R^{-1}(n)\underline{u}(n) \tag{75}$$

Hence if we use Givens rotations to annihilate the vector $\underline{u}^T(n)$ by rotation against the matrix $R^T(n)$ then the bottom row of the composite rotation matrix will contain the term

$$R^{-1}(n)\underline{u}(n) = -\underline{w}(n) \tag{76}$$

i.e. the negative of the least squares weight vector. A systolic array for implementing this algorithm can be found in reference [31].

2.8 Comparison with Recursive Modified Gram-Schmidt Algorithms

The square-root-free algorithm and architecture with direct residual extraction as described in sections 2.1 to 2.5 was obtained independently by Ling, Manolakis and Proakis [15] based on the method of modified Gram-Schmidt (MGS) orthogonalisation. The MGS algorithm operates on a fixed block of data and is essentially non-recursive. It solves the least squares problem by applying a sequence of linear combinations to the columns of the block data matrix $X(n)$ and transforming it into a matrix $X'(n)$ whose columns are mutually orthogonal. The complete transformation may be represented by means of an upper triangular matrix whose elements correspond to the triangular matrix $R(n)$ which would have resulted from applying a QR decomposition to the data matrix $X(n)$. The process is extended to include the vector of samples $y(n)$ in the primary channel and hence extract from it the component which is orthogonal to the columns of $X(n)$. The bottom element of the resulting vector then corresponds to the a-posteriori least squares residual at time n. The QR decomposition and MGS techniques are clearly related except that the former applies an orthogonal transformation to the data matrix $X(n)$ in order to produce an upper triangular form whereas the latter applies an upper triangular matrix transformation to $X(n)$ in order to produce a matrix with orthogonal columns. Also, the QRD technique may be applied in a convenient row-recursive manner using a sequence of elementary Givens rotations. The most important contribution of Ling, Manolakis and Proakis was to show how the MGS algorithm could also be updated one row at a time thereby generating the recursive modified Gram-Schmidt (RMGS) algorithm which may be implemented using a triangular processing architecture similar to that in figure 2.

Motivated by the desire to update the stored triangular matrix elements directly rather than as a ratio of other updated terms which occur more naturally in the RMGS approach, they manipulated their algorithm further. The resulting update equations were found to involve an important element of feedback which, leeds to an improvement in numerical stability. It is interesting to note that the error feedback RMGS algorithm derived by Ling, Manolakis and Proakis [15] is identical to the form of square-root-free Givens rotation algorithm defined in figures 2 and 4 [14]. It was this relationship which led us to refer to the basic operation in figure 4 as the square-root-free with feedback (SF/FB) Givens rotation

and to identify the underlying feedback mechanism which is inherent to it.

2.9 Comparison with Kalman filter algorithms

The QR decomposition algorithms for recursive least squares processing as defined in figures 2, 3 and 4 operate directly on the basic data matrix $X(n)$ as opposed to the data covariance matrix M(n) and, as noted previously, this has significant numerical advantages. It was also shown in equation (8) that the upper triangular matrix $R(n)$ which is stored and updated within the processor array is analytically identical to the Cholesky square-root factor of the data (or information) covariance matrix. In the nomenclature of Kalman filtering, the QRD algorithm constitutes a numerically stable form of square-root information Kalman filter with unit state-space matrix.

It has recently been shown by Chen and Yao [5] how the algorithm and triangular array architecture may, in fact, be extended to the case of a general square-root information Kalman filter with arbitrary state space matrix. Gaston et al [6] have also shown how the triangular QRD array may be used as the core processor in a general square-root covariance Kalman filter. Note that in Kalman filtering nomenclature the term "covariance" does not refer to forming or using the data covariance matrix $M(n)$. Instead, it denotes an algorithm which is based on updating the matrix $M^{-1}(n)$ (or its Cholesky square root factor $R^{-1}(n)$ in the present context) since this specifies the covariance of the weight vector estimate. In the special case of unit state-space matrix the covariance Kalman filter reduces to a least squares estimation algorithm which makes use of the well-known matrix inversion lemma to perform successive rank-one updates of the matrix $M^{-1}(n)$. This is often referred to as the "Recursive Least Squares" (RLS) algorithm although it is fundamentally different from the QRD-based recursive least squares technique described in this memorandum. For example, as is often pointed out "persistent excitation" is essential if the traditional RLS algorithm is to retain numerical stability due to the $1/\beta$ term which occurs in the associated Riccati equation. However this problem does not arise with the QRD algorithm described in sections 2.1 to 2.5. It would seem sensible, then, to refer to the traditional RLS and the alternative QRD techniques as the "covariance" and "square-root information" recursive least squares algorithms respectively.

Finally, it is worth noting that several authors [1][21][22] have recently developed stable "square-root covariance" least squares algorithms and architectures based on the technique illustrated in figure 8 for updating the "square-root covariance" matrix $R^{-1}(n)$. Their technique, however, makes clever use of a pinning vector to avoid storing and updating the "square-root information" matrix $R(n)$ explicitly and only requires a single triangular processor array as described in section 2.7. It is not clear how their method would extend to the general "square-root covariance" Kalman filtering problem or how it relates to the "square-root covariance" Kalman filter architectures proposed by Gaston et al [6].

3 Adaptive FIR Filtering.

3.1 The QRD Approach.

In section 2 we saw how an adaptive linear combiner could be applied to the problem of narrowband adaptive beamforming. The same linear combiner could be used to construct an adaptive FIR fil-

ter. In this case, the combined output at time t_i is given by the equation

$$e(t_i) = \underline{x}^T(t_i)\underline{w} + y(t_i) = \sum_{j=0}^{p-1} w_j x(t_{i-j}) + y(t_i) \tag{77}$$

which is identical to the narrow-band beamformer case (equation (1)) except that the input vector $\underline{x}(t_i)$ now exhibits a high degree of time-shift invariance. This property manifests itself in the fact that the "data matrix" ($X_p(n)$ of equation (79)) has a Toeplitz structure i.e. each row of the matrix is obtained by shifting the previous row one column to the right and introducing one new data sample. Various algorithms have been devised that take advantage of this redundancy and so reduce the computational load, for a p^{th} order filter, from $O(p^2)$ to $O(p)$ arithmetic operations per sample time (see Haykin [12]). The common basis for these fast algorithms is an efficient technique for solving the least squares linear prediction problem. The concepts of forward and backward linear prediction must both be introduced for this purpose. The adaptive filtering problem can then be solved using quantities already calculated during the linear prediction stages. Unfortunately the majority of these fast algorithms exhibit some form of numerical instability although much work has been done to overcome the numerical problems and various rescue procedures have been developed - see [12].

As we have seen, it is possible to solve a least squares minimisation problem using the technique of QR decomposition. Extensive computer simulations of this algorithm[32] have shown the QRD-based least squares minimisation algorithm to have excellent numerical properties. However, since the recursive QRD algorithm presented in section 2 is designed to solve a general recursive least squares minimisation problem, it requires $O(p^2)$ operations per sample time to generate the solution to a p^{th} order adaptive filter problem. A QRD-based algorithm which is designed for the special case of adaptive filtering and only requires $O(p)$ operations per sample time is thus of considerable interest [13][17][26][33]. In order to simplify the analysis we consider only real signals. The extension to the complex case is straight forward and indeed the algorithms presented in the appendix are for complex signals.

In a least squares adaptive filter of order p, the set of p weights[1] $\underline{w}_p(n)$ at time n is chosen in order to minimise the sum of the squared differences between a reference signal $y(n)$ and a linear combination of the p samples from a data time series $x(t_{n-i})$ $(0 \le i \le p-1)$. Specifically, the measure to be minimised is $\|\underline{e}_p(n)\|$ where:

$$\underline{e}_p(n) = X_p(n)\underline{w}_p(n) + \underline{y}(n) \tag{78}$$

1. In this section we append a subscript to all variables to indicate the order of the problem being solved

$$X_p(n) = B(n) \begin{bmatrix} x(1) & x(0) & \dots & x(2-p) \\ x(2) & x(1) & \dots & x(3-p) \\ \vdots & \vdots & & \vdots \\ x(n) & x(n-1) & \dots & x(n-p+1) \end{bmatrix} \tag{79}$$

and

$$\underline{y}(n) = B(n) \begin{bmatrix} y(1) & \dots & y(n) \end{bmatrix}^T \tag{80}$$

Compared with equations (5) and (6), we see that equations (78) to (80) constitute a standard least squares minimisation problem except for time-shift invariance of the data. We could therefore proceed to solve this least squares minimisation problem via the QR decomposition technique described in section 2.1. In order to do so we must determine an orthogonal matrix $Q_p(n)$ that transforms the matrix $X_p(n)$ into upper triangular form[2] and use the same matrix $Q_p(n)$ to rotate the reference vector $\underline{y}(n)$. i.e.

$$Q_p(n)\, X_p(n) = \begin{bmatrix} R_p(n) \\ O \end{bmatrix} \tag{81}$$

and

$$Q_p(n)\, \underline{y}(n) = \begin{bmatrix} \underline{u}_p(n) \\ v_p(n) \end{bmatrix} \tag{82}$$

It should be noted that once $Q_p(n)$ has been found, the filtering problem has effectively been solved. Knowledge of $Q_p(n)$ means that $\gamma_p(n)$ is known. It also allows the angle normalised residual $\alpha_p(n)$ - the last component of the vector $\underline{v}_p(n)$ - to be calculated and thus the least squares residual may be found (see section 2.5). The $O(p^2)$ QRD-based algorithm for the solution of a p^{th} order least squares minimisation problem as described in section 2 has many desirable features. It operates in the "data domain" and has a time-recursive formulation with time-independent computational requirement and a regular parallel architecture. However the time shift redundancy in the adaptive filtering problem can be used to improve the method further by reducing the computational load from $O(p^2)$ to $O(p)$. The development of fast QRD algorithms for adaptive filtering is based, almost entirely, on the principle of constructing partially triangularised matrices from known quantities and then finding a set of rotations to complete the process. We recall that this was also a key element in the derivation of the time-recursive algorithm in section 2.2. The solution at time n was generated by rotating the new input data for time n into the upper triangular matrix associated with the solution at time (n-1).

Recall that the set of rotations, $Q_p(n)$, required to solve the adaptive filtering problem are entirely dependent on the matrix $X_p(n)$. The matrix $X_p(n)$ can, however, be built up in an order recursive manner

2. In the following, we use shading in order to emphasise the structure of the non-zero elements of a matrix.

by adding extra columns which, because of its Toeplitz structure, consists of one new element and a time-shifted version of the previous column. Consider the following decompositions:

$$X_p(n) = B(n) \begin{bmatrix} x(1) & \dots & x(2-p) \\ \vdots & & \vdots \\ x(n) & \dots & x(n-p+1) \end{bmatrix} \tag{83}$$

$$= \begin{bmatrix} X_{p-1}(n) & \underline{y}^b_{p-1}(n) \end{bmatrix} \tag{84}$$

$$= \begin{bmatrix} \beta^{n-1}x(1) & z^T \\ \underline{y}^f_{p-1}(n) & X_{p-1}(n-1) \end{bmatrix} \tag{85}$$

where[3]
$$\underline{y}^f_{p-1}(n) = B(n-1)\ x(2), \dots, x(n)^T \tag{86}$$

$$\underline{y}^b_{p-1}(n) = B(n)\ x(-p+2), \dots, x(n-p+1)^T \tag{87}$$

and
$$z = \beta^{n-1}\ x(0), \dots, x(-p+2)^T \tag{88}$$

Note from equation (84), that if we had already determined the rotation matrix $Q_{p-1}(n)$ which triangularises the matrix $X_{p-1}(n)$, we could use it to partially triangularise the matrix $X_p(n)$. In doing so we would also have to rotate the vector $\underline{y}^b_{p-1}(n)$ but these are exactly the steps required in the QRD-based solution of the $(p-1)^{st}$ order backward linear prediction problem.

In the $(p-1)^{st}$ order backward linear prediction problem at time n, an estimate of x(n-p+1) is formed from a linear combination of the data {x(n), ..., x(n-p+2)}. The solution to this problem depends on the triangularisation of the matrix $X_{p-1}(n)$ and the transformation of the reference vector $\underline{y}^b_{p-1}(n)$. Hence, knowing the solution to the $(p-1)^{st}$ order backward problem at time n would allow us to construct the partially triangularised matrix $Q_{p-1}(n)X_p(n)$ from known quantities and thus save a large amount of computation. This partially triangularised matrix could then be transformed into the triangular matrix $R_p(n)$ by a sequence of Givens rotations.

Equation (85) allows another partially triangularised version of $X_p(n)$ to be constructed, this time using quantities from the $(p-1)^{st}$ order forward linear prediction problem. The $(p-1)^{st}$ order (forward)

3. Note that the subscript 'p' attached to the vector $\underline{y}^f_p$ is superfluous and that $\underline{y}^f_p = \underline{y}^f_{p-1}$ etc. Its use is merely to preserve symmetry with the vector $\underline{y}^b_p$ for which the subscript is necessary

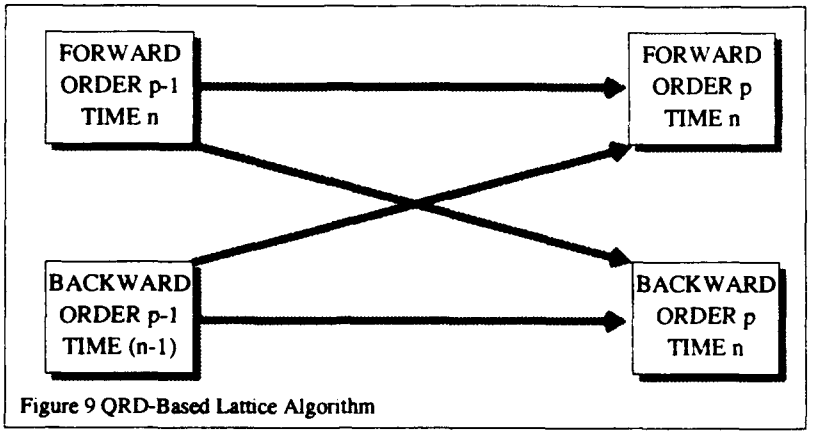


Figure 9 QRD-Based Lattice Algorithm

linear prediction problem, at time n, is defined as the estimation of x(n) based upon the data {x(n-1), ... x(n-p+1)}. This involves the triangularisation of the matrix $X_{p-1}(n-1)$ and the transformation of the relevant reference vector $y^{f}_{p-1}(n)$. We could then use the decomposition given in equation (85) to generate a partial triangularisation of the matrix $X_p(n)$ from known quantities. The formation of the triangular matrix $R_p(n)$ could then be achieved very easily using a sequence of Givens rotations.

It is clear that the two linear prediction problems of order (p-1) are intimately connected to the problem of determining a minimum set of rotations which produce the triangular matrix $R_p(n)$. However, the triangularisation of $X_p(n)$ is central not only to the adaptive filtering problem but also to the p^{th} order linear prediction problems. The rotations which transform the matrix $X_p(n)$ into $R_p(n)$ are used to solve the forward problem at time (n+1) and the backward problem at time n. With a suitable time delay, we can therefore construct an order recursive algorithm for linear prediction and adaptive filtering (figure 9).

It is also possible to develop another type of QRD-based fast algorithm. From the discussion above it is clear that we have a fast method for transforming $R_{p-1}(n-1)$ into $R_p(n)$ via equation (85). This technique allows us to transform the matrix $Q_{p-1}(n-1)$ into $Q_p(n)$ (i.e. we have a time and order update). Equation (84) constitutes a method for transforming $R_p(n)$ into $R_{p-1}(n)$ and hence $Q_p(n)$ into $Q_{p-1}(n)$ (i.e. an order down-date). Thus by combining these two transformations we can achieve an overall time update for the rotation matrix Q_p (figure 10). This transformation does not lead directly to the construction of a fast algorithm since having to calculate the various Q matrices explicitly would require $O(p^2)$ operations and does not represent a reduction in the computational requirement. However we have already seen that the value of this matrix can be inferred from knowledge of its right-hand column (section 2.7). Clearly the transformations that update the matrix Q_p will also perform a time update for this column vector. Then, because we are now dealing with a vector rather than a matrix, the number

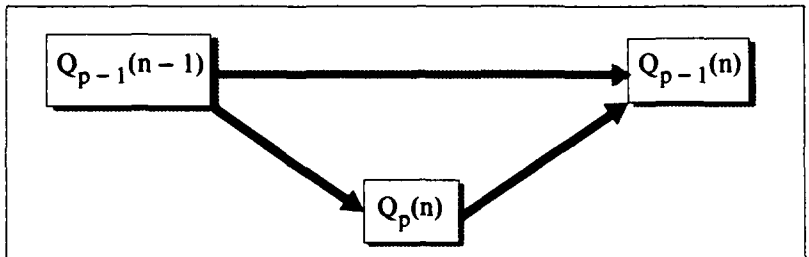


Figure 10 QRD-Based Fast Kalman Algorithm

of computations required to perform this type of update is $O(p)$ and a "fast" algorithm results.

Traditionally "fast" adaptive filtering algorithms fall into two classes: the least-squares lattice algorithms and the "fast Kalman" algorithms. Both types of algorithm solve the p^{th} order linear prediction problem in $O(p)$ operations. The least-squares lattice algorithms do this by solving all of the lower order problems in sequence, whereas the "fast Kalman" algorithms concentrate on a problem of given order and achieve the reduction in computational load by using the time and order update / order downdate technique. Not surprisingly, the two classes of algorithms have tended to be quite different; the fast Kalman algorithms usually calculate the transversal filter coefficients explicitly, whereas the lattice algorithms deal directly with the filter residuals (errors) and calculate reflection coefficients. The fast Kalman algorithms also tend to require fewer arithmetic operations than the lattice algorithms although both are linear in the problem order. The level at which the two different algorithms can be pipelined is different - the lattice algorithms having a higher degree of concurrency. It is also worth noting that the data downdating step which is implicitly required by the fast Kalman algorithms gives cause for concern with regard to numerical stability.

Based on the above classification, we refer to the two fast algorithms derived in this memorandum as the QRD-based least-squares lattice algorithm (figure 9) and the QRD-based fast Kalman algorithm (figure 10). It should be noted, however, that this latter algorithm does not calculate the transversal filter coefficients explicitly; instead it generates the required filter output using the QRD-based method of "direct residual-extraction" discussed in section 2.5. The QRD-based fast Kalman algorithm is also unusual in that it quite naturally produces the solution to all lower order problems whereas fast Kalman algorithms are usually seen as being of "fixed order". This property is a natural consequence of using the QRD technique (see section 3.6). We begin the detailed derivation of these fast algorithms by considering the problem of determining an efficient method for the solution of the p^{th} order forward linear prediction problem.

3.2 Forward Linear Prediction

The p^{th} order forward linear prediction problem, at time n, requires the determination of the vector of filter coefficients $\underline{w}_p^f(n) = [w_{p,0}^f(n), \ldots, w_{p,p-1}^f(n)]^T$ that minimises the total prediction error

$e_p^f(n)$ where

$$\underline{e}_p^f(n) = X_p(n-1)\underline{w}_p(n) + \underline{y}_p^f(n) \tag{89}$$

with $X_p(n-1)$ and $\underline{y}_p^f(n)$ as defined in equations (83) and (86) respectively. In order to solve this least squares problem via the QR decomposition technique we have to determine the rotation matrix $Q_p(n-1)$ that triangularises the data matrix $X_p(n-1)$ and then apply it to the vector $\underline{y}_p^f(n)$ in order to calculate the angle normalised residual $\alpha_p^f(n)$ (cf. equation (21)). We also need to be able to calculate $\gamma_p(n-1)$ in order to generate the a-posteriori prediction residual (see equation (35)). Note also that the triangularisation of $X_p(n-1)$ is exactly what we require in the solution of the p^{th} order adaptive filtering problem at time (n-1) - see equation (78). Consider, therefore, the following composite matrix:

$$M_1 = \begin{bmatrix} \underline{y}_p^f(n) & X_p(n-1) & \underline{y}(n-1) & \pi_{n-1} \end{bmatrix} \tag{90}$$

From equations (14), (19) and (28), we have that

$$Q_p(n-1)\pi_{n-1} = \hat{Q}_p(n-1)\pi_{n-1} = \begin{bmatrix} \underline{a}_p^T(n-1), & o^T, & \gamma_p(n-1) \end{bmatrix}^T \tag{91}$$

It should be clear, therefore, that the vector π_{n-1} in the above matrix (equation (90)) will enable us to calculate $\gamma_p(n-1)$ just as the vector $\underline{y}_p^f(n)$ allows $\alpha_p^f(n)$ to be calculated. Similarly the presence of the vector $\underline{y}(n-1)$ will allow us to calculate $\alpha_p(n-1)$.

Now from equations (84) and (86)we have that

$$M_1 = \begin{bmatrix} \underline{y}_{p-1}^f(n) & X_{p-1}(n-1) & \underline{y}_{p-1}^b(n-1) & \underline{y}(n-1) & \pi_{n-1} \end{bmatrix} \tag{92}$$

where we have used the fact that $\underline{y}_p^f(n) = \underline{y}_{p-1}^f(n)$in order emphasise the appearance of the lower order problem in the decomposition of the matrix M_1. Hence

$$Q_{p-1}(n-1)\, M_1 = \begin{bmatrix} \underline{u}_{p-1}^f(n) & R_{p-1}(n-1) & \underline{u}_{p-1}^b(n-1) & \underline{u}_{p-1}(n-1) & \underline{a}_{p-1}(n-1) \\ \underline{v}_{p-1}^f(n) & O & \underline{v}_{p-1}^b(n-1) & \underline{v}_{p-1}(n-1) & \underline{g}_{p-1}(n-1) \end{bmatrix} = M_2 \tag{93}$$

where $$\underline{g}_{p-1}(n-1) = \begin{bmatrix} o^T, & \gamma_{p-1}(n-1) \end{bmatrix}^T \tag{94}$$

It is easy to show that $\underline{v}_{p-1}^f(n)$ and $\underline{v}_{p-1}^b(n-1)$ must have a time recursive decomposition similar to that given in equation (21) for $\underline{v}_{p-1}(n-1)$. Hence

$$M_2 = \begin{bmatrix} \mu^f_{p-1}(n) & R_{p-1}(n-1) & \underline{\mu}^b_{p-1}(n-1) & \underline{\mu}_{p-1}(n-1) & \underline{a}_{p-1}(n-1) \\ \beta v^f_{p-1}(n-1) & O & \beta \underline{v}^b_{p-1}(n-2) & \beta \underline{v}_{p-1}(n-2) & o \\ \alpha^f_{p-1}(n) & o^T & \alpha^b_{p-1}(n-1) & \alpha_{p-1}(n-1) & \gamma_{p-1}(n-1) \end{bmatrix} \tag{95}$$

Now suppose that we had already calculated a rotation matrix[4], $Q^f_p(n-1)$ say, that rotates the vector $\underline{v}^b_{p-1}(n-2)$ into a form where only the top element is non-zero i.e.

$$\begin{bmatrix} Q^f_p(n-1) & \underline{o} \\ o^T & 1 \end{bmatrix} M_2 =$$

$$\begin{bmatrix} \mu^f_{p-1}(n) & R_{p-1}(n-1) & \underline{\mu}^b_{p-1}(n-1) & \underline{\mu}_{p-1}(n-1) & \underline{a}_{p-1}(n-1) \\ \beta\mu^f_{p-1}(n-1) & o^T & \beta\varepsilon^b_{p-1}(n-2) & \beta\mu_{p-1}(n-2) & 0 \\ \beta\vartheta^f_{p-1}(n-1) & O & o & \beta\underline{\vartheta}_{p-1}(n-2) & o \\ \alpha^f_{p-1}(n) & o^T & \alpha^b_{p-1}(n-1) & \alpha_{p-1}(n-1) & \gamma_{p-1}(n-1) \end{bmatrix} = M_3 \tag{96}$$

The new quantities $\mu^f_{p-1}(n-1)$, $\vartheta^f_{p-1}(n-1)$, $\mu_{p-1}(n-2)$, $\underline{\vartheta}_{p-1}(n-2)$ and $\varepsilon^b_{p-1}(n-2)$ are defined by this operation and we note, by analogy with equation (12), that $\varepsilon^b_{p-1}(n-2)$ is the square-root of the $(p-1)^{st}$ order backward prediction energy at time (n-2).

Now in order to complete the triangularisation of the matrix $X_p(n-1)$ (see equation (92)) all that is required is the annihilation of the single element $\alpha^b_{p-1}(n-1)$. This can be carried out using a single Givens rotation:

$$\hat{Q}^f_p(n)\ M_3 = \begin{bmatrix} u^f_{p-1}(n) & R_{p-1}(n-1) & \underline{u}^b_{p-1}(n-1) & \underline{u}_{p-1}(n-1) & \underline{a}_{p-1}(n-1) \\ \mu^f_{p-1}(n) & o^T & \varepsilon^b_{p-1}(n-1) & \mu_{p-1}(n-1) & \underline{a}_p(n-1) \\ \beta\underline{\vartheta}^f_{p-1}(n-1) & O & o & \beta\underline{\vartheta}_{p-1}(n-2) & o \\ \alpha^f_p(n) & o^T & 0 & \alpha_p(n-1) & \gamma_p(n-1) \end{bmatrix} \tag{97}$$

$$= \begin{bmatrix} \underline{u}^f_p(n) & R_p(n-1) & \underline{u}_p(n-1) & \underline{a}_p(n-1) \\ \underline{v}^f_p(n) & O & \underline{v}_p(n-1) & \underline{g}_p(n-1) \end{bmatrix} \tag{98}$$

4. The notation for the rotation matrices introduced in this memorandum are somewhat arbitrary: in order to solve the p^{th} order *forward* linear prediction problem, we must annihilate quantities from the $(p-1)^{st}$ order ***backward*** prediction problem. Following the nomenclature used for reflection coefficients, the rotation matrices used here are labelled according to the problem to which they relate rather than the quantities they annihilate.

where $a_p(n-1)$ is defined by this operation. The identity in equation (98), and hence the labelling of some of the elements in the bottom row of the right-hand matrix in equation (97), follows by definition (see equation (92)). Note from the above that the "new" quantities $\underline{\vartheta}^f_{p-1}(n-1)$ and $\underline{\vartheta}_{p-1}(n-2)$, introduced in equation (96), are equivalent to existing variables. Indeed, from equations (97) and (98) we have that

$$\begin{bmatrix} \beta\underline{\vartheta}^f_{p-1}(n-1) & \beta\underline{\vartheta}_{p-1}(n-2) \\ \alpha^f_p(n) & \alpha_p(n-1) \end{bmatrix} = \begin{bmatrix} \underline{v}^f_p(n) & \underline{v}_p(n-1) \end{bmatrix} \tag{99}$$

so that, see equation (21),

$$\underline{\vartheta}^f_{p-1}(n) = \underline{v}^f_p(n) \tag{100}$$

and

$$\underline{\vartheta}_{p-1}(n) = \underline{v}_p(n) \tag{101}$$

From the above we see that the sequence of orthogonal transformations shown in equations (93), (96) and (97) solve the p^{th} order forward linear prediction problem. Note, however, that the matrix operated upon by $\hat{Q}^f_p(n)$ in equation (97) consists entirely of quantities that would be available if the $(p-1)^{st}$ order forward and backward problems had already been solved at time n and n-1 respectively. If this assumption were true then we could have constructed this intermediate matrix directly, thereby circumventing the need for the operations as outlined in equations (93) and (96). Only the single Givens rotation of equation (97) would actually need to be performed and so the number of arithmetic operations required would be independent of p: only eight elements, one of which is zero, of the left hand matrix in equation (97) are affected by the required rotation. Having derived a fast method for solving the forward linear prediction problem, we now consider a fast update method for the auxiliary (backward) problem.

3.3 Backward Linear Prediction

The p^{th} order backward linear prediction problem, at time n, requires the determination of the vector of filter coefficients $\underline{w}^b_p(n) = [w^b_{p,0}(n), \ldots, w^b_{p,p-1}(n)]^T$ that minimises the total prediction error $\underline{e}^b_p(n)$ where

$$\underline{e}^b_p(n) = X_p(n)\underline{w}^b_p(n) + \underline{y}^b_p(n) \tag{102}$$

Again the least squares solution to this problem can be found by the method of QR decomposition. It is necessary to determine the rotation matrix $Q_p(n)$ that triangularises the data matrix $X_p(n)$ and then apply it to the vector $\underline{y}^b_p(n)$ in order to calculate $\alpha^b_p(n)$ (cf. equation (21)). We also need to be able to calculate $\gamma_p(n)$ (see equation (35)) in order to generate the a-posteriori prediction residual. Consider, therefore, the following composite matrix and the illustrated decomposition which is a simple extension of equation (85)

$$X_p(n)\ \underline{y}_p^b(n)\ \underline{\pi}_n = \begin{bmatrix} \beta^{n-1}x(1) & \underline{o}^T & 0 & 0 \\ \underline{y}_{p-1}^f(n) & X_{p-1}(n-1) & \underline{y}_{p-1}^b(n-1) & \underline{\pi}_{n-1} \end{bmatrix} = M_4 \tag{103}$$

In equation (103), it has been assumed that the data sequence x(n) is pre-windowed (i.e. x(n) = 0 for $n \leq 0$). Note that this is the only place in the analysis where we require this assumption[5]. Consider the effect of the rotation matrix $Q_{p-1}(n-1)$ on the lower part of the matrix in equation (103):

$$\begin{bmatrix} 1 & \underline{o}^T \\ \underline{o} & Q_{p-1}(n-1) \end{bmatrix} M_4 = \begin{bmatrix} \beta^{n-1}x(1) & \underline{o}^T & 0 & 0 \\ \underline{u}_{p-1}^f(n) & R_{p-1}(n-1) & \underline{u}_{p-1}^b(n-1) & \underline{a}_{p-1}(n-1) \\ v_{p-1}^f(n) & O & \underline{v}_{p-1}^b(n-1) & \underline{g}_{p-1}(n-1) \end{bmatrix} = M_5 \tag{104}$$

As before, all the vectors on the bottom row of the matrix M_5 may be written in terms of their underlying time recursion and thus we obtain the expression:

$$M_5 = \begin{bmatrix} \beta^{n-1}x(1) & \underline{o}^T & 0 & 0 \\ \underline{u}_{p-1}^f(n) & R_{p-1}(n-1) & \underline{u}_{p-1}^b(n-1) & \underline{a}_{p-1}(n-1) \\ \beta\underline{v}_{p-1}^f(n-1) & O & \beta\underline{v}_{p-1}^b(n-2) & \underline{o} \\ \alpha_{p-1}^f(n) & \underline{o}^T & \alpha_{p-1}^b(n-1) & \gamma_{p-1}(n-1) \end{bmatrix} \tag{105}$$

Now suppose that we have already constructed a rotation matrix $Q_p^b(n-1)$ that annihilates the vector $\underline{v}_{p-1}^f(n-1)$ by rotation against the element $\beta^{n-1}x(1)$ i.e.

$$\begin{bmatrix} Q_p^b(n-1) & \underline{o} \\ \underline{o}^T & 1 \end{bmatrix} M_5 = \begin{bmatrix} \beta\epsilon_{p-1}^f(n-1) & \underline{o}^T & \beta\mu_{p-1}^b(n-2) & 0 \\ \underline{u}_{p-1}^f(n) & R_{p-1}(n-1) & \underline{u}_{p-1}^b(n-1) & \underline{a}_{p-1}(n-1) \\ \underline{o} & O & \beta\underline{\vartheta}_{p-1}^b(n-2) & \underline{o} \\ \alpha_{p-1}^f(n) & \underline{o}^T & \alpha_{p-1}^b(n-1) & \gamma_{p-1}(n-1) \end{bmatrix} = M_6 \tag{106}$$

Now let $\hat{Q}_p^b(n)$ be the rotation matrix that annihilates the element $\alpha_{p-1}^f(n)$ by rotation against the element $\beta\epsilon_{p-1}^f(n-1)$. Application of the transformation $\hat{Q}_p^b(n)$ to the above matrix yields the result:

---

5. It is possible to develop a QRD-based fast Kalman algorithm in which $x(n) \neq 0$ for $n \leq 0$. The resulting algorithm is based on much of the pre-windowed version presented here but with some extra computation - see Cioffi [4] for further details. The authors are not aware of any similar work for the QRD-based lattice algorithm.

$$\hat{Q}_p^b(n)\ M_6 = \begin{matrix} \varepsilon_{p-1}^f(n) & o^T & \mu_{p-1}^b(n-1) & \tilde{a}_p(n) \\ \underline{\iota}_{p-1}(n) & R_{p-1}(n-1) & \underline{u}_{p-1}^b(n-1) & \underline{a}_{p-1}(n-1) \\ o & O & \beta\vartheta_{p-1}^b(n-2) & o \\ 0 & o^T & \hat{\alpha}_{p-1}^b(n) & \hat{\gamma}_{p-1}(n) \end{matrix} = M_7 \qquad (107)$$

where the new quantities $\tilde{a}_p(n)$, $\hat{\alpha}_{p-1}^b(n)$ and $\hat{\gamma}_{p-1}(n)$ are defined by this equation. Bearing in mind the underlying data matrix (see equation (103)), recall that we are attempting to create an upper-triangular pxp matrix in the upper left-hand corner on the matrix in equation (107). At present this sub-matrix is not quite triangular but a little thought shows that it is easy to construct a matrix ($\tilde{Q}_p^b(n)$ say) which will complete the required triangularisation. Specifically, let $\tilde{Q}_p^b(n)$ be constructed from a sequence of Givens rotations such that each rotation annihilates one element of the vector $\underline{u}_{p-1}^f(n)$ in turn. Provided we start with the last element of $\underline{u}_{p-1}^f(n)$ and work upwards, the sequence of rotations will not destroy the triangular structure of the matrix $R_{p-1}(n-1)$ although its value will be changed as a result. Thus

$$\tilde{Q}_p^b(n)\ M_7 = \begin{matrix} R_p(n) & \underline{u}_p^b(n) & \underline{a}_p(n) \\ O & v_p^b(n) & g_p(n) \end{matrix} \qquad (108)$$

Note that the matrix $\tilde{Q}_p^b(n)$ only affects the upper part of the partitioned matrix on the left hand side of equation (108). It should therefore be clear that

$$\hat{\alpha}_{p-1}^b(n) = \alpha_p^b(n) \qquad \text{and} \qquad \hat{\gamma}_{p-1}(n) = \gamma_p(n) \qquad (109)$$

Also note that

$$\begin{matrix} \beta\vartheta_{p-1}^b(n-2) \\ \alpha_p^b(n) \end{matrix} = v_p^b(n) \qquad (110)$$

and so the "new" quantity $\vartheta_{p-1}^b(n-2)$, introduced in equation (106), is equivalent to an existing variable. Indeed, from equation (21) we have

$$\vartheta_{p-1}^b(n-1) = v_p^b(n) \qquad (111)$$

Hence the sequence of orthogonal rotations given in equations (104), (106), (107) and (108) solve the p^{th} order backward linear prediction problem. Following the development of the solution to the forward problem in section 3.2, note that the data matrix on the left-hand side of equation (107) could be constructed directly given the solutions to the $(p-1)^{st}$ order forward and backward linear prediction problems at time n and n-1 respectively. Thus the transformations shown in equations (104) and (106)

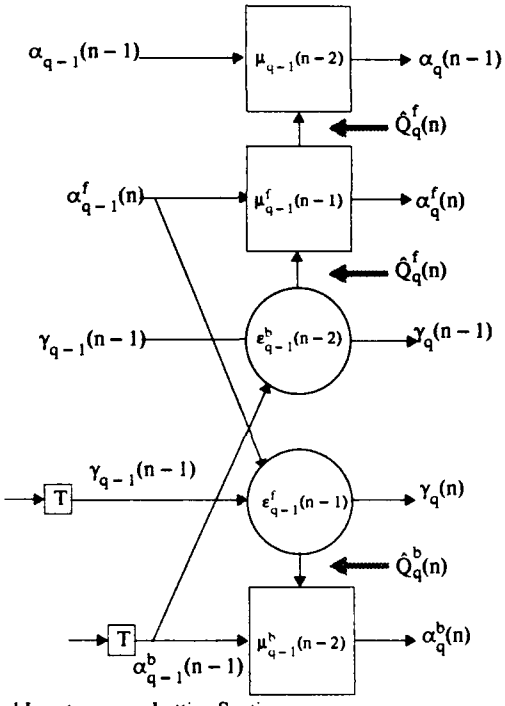


Figure 11 QRD-based Least-squares Lattice Section.

could be by-passed. Furthermore, assuming that we are only interested in the prediction residuals, the transformation shown in equation (108) is not required either, since $\alpha_p^b(n)$ and $\gamma_p(n)$ are both available in the matrix M_7. Thus only the Givens rotation summarised in equation (107) need actually be performed and the number of arithmetic operations required is independent of p: only six elements, one of which is zero, of the matrix M_6 are affected by the rotation.

3.4 The QRD Least-squares Lattice Algorithm.

Gathering together the results of sections 3.2 and 3.3 we see that it is possible to utilise various terms from the solution to the $(p-1)^{st}$ order forward and backward linear prediction problems, at time n and (n-1) respectively to generate corresponding terms for the solution to the p^{th} order problems at time n (see figure 11). Note that the processing elements shown in figure 11 are the same as those used in the triangular systolic array described in section 2.3 (i.e. as shown in figure 3). It is possible to show that the corresponding square-root-free implementation may be obtained very simply by substituting the processing elements shown in figure 4.

Given that 0^{th} order linear prediction is trivial, we can thus generate the solution to the p^{th} order problem by iteration in order using a cascade of the sections shown in figure 11. The resultant architecture (figure 12) has a lattice structure and, since the number of operations per stage is independent of p,

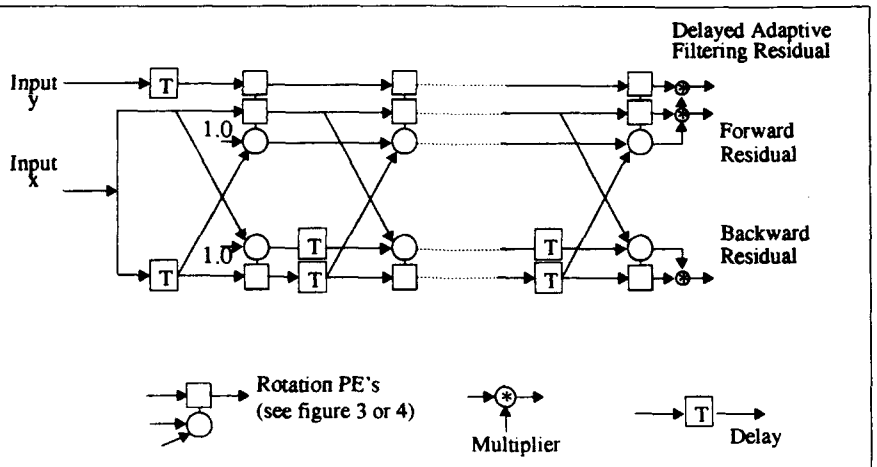
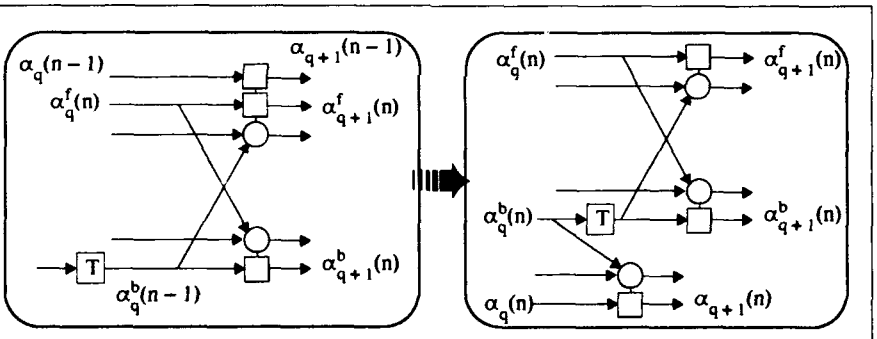


Figure 12 "Delayed" Adaptive Filtering Lattice



a) Delayed Residual

b) Undelayed Residual

Figure 13 Equivalent Order Updates for Joint Process Residual.

O(p) operations are required to solve the p^{th} order problem (see appendix for algorithm listing). Note that by including the adaptive filtering reference vector y(n-1) in the calculation of the p^{th} order forward linear prediction problem (section 3.2) we automatically solve the p^{th} order adaptive filtering problem for time (n-1). The solution to the adaptive filtering problem for time n can be easily derived from the above. Note that in figure 11 the rotation parameters used (in the square processors) to calculate each order update of the delayed joint process residual are evaluated (in the round processors) from the delayed backward residual. Thus by using the undelayed backward residual, we can calculate the rotations required to update the undelayed joint process residual - see figure 13. In this form, a stage of the QRD-based lattice can be seen to be very reminiscent of that of the standard least-squares lattice; the only difference is that the former structure has rotation processors instead of the (reflection coefficient) mul-

tipliers of the conventional form (see Haykin [12] figure 17.2). Although conceptually appealing, the QRD-based lattice filter stage shown in figure 13b is inefficient because there is duplication in the calculations of the rotation parameters for the forward prediction and joint estimation residuals. Any practical implementation would clearly calculate the rotation parameters only once in the joint process estimation channel and store them for use in the next time instant in the forward prediction channel.

In section 2.8 we discussed the Recursive Modified Gram Schmidt (RMGS) algorithm with error feedback. This was proposed by Ling Manolakis and Proakis [15] for the general narrowband adaptive beamforming problem and leads to a triangular array processor equivalent to the one described in figures 2 and 4. Ling and Proakis [16] subsequently developed the technique to produce an efficient RMGS algorithm with error feedback which requires $O(p)$ arithmetic operations per sample time to solve a p^{th} order adaptive FIR filtering problem. Their algorithm also has a lattice structure and, in view of the equivalence referred to above, it is not surprising to find that it corresponds exactly to the QRD-based least squares lattice algorithm derived in this memorandum (assuming that the square-root-free Givens rotations in figure 4 are employed). The RMGS lattice algorithm with error feedback was the first numerically stable least squares lattice algorithm to be developed and it is interesting, therefore, to note this correspondence to an algorithm based entirely on orthogonal rotations.

3.5 The QRD "Fast Kalman" Algorithm.

In this section we expand on the remarks made in section 3.1 about the connection between the forward and backward linear prediction problems for a fixed order and develop the QRD-based fast Kalman algorithm. We take as our starting point the situation where we have solved the p^{th} order forward linear prediction problem for time n and are attempting to update this solution to the next time instant. Specifically, if we could generate the matrix $\hat{Q}_p(n)$ using $O(p)$ arithmetic operations then the linear prediction solution could be updated efficiently.

The original derivation of a QRD-based fast Kalman algorithm was presented by Cioffi [4] although his algorithm differs somewhat from that presented here. The material presented in this section follows that of reference [23] and leads to the same algorithm as derived by Regalia and Bellanger [27]. Both of these derivations were based on Cioffi's work and the approach presented here actually follows the same sequence of ideas used in Cioffi's original paper. The difference in the resulting algorithms is due to the choice of triangular matrix used in the QR decomposition. Here we use an upper, right-hand triangular matrix to conform with the work of Gentleman and Kung [8]. Cioffi on the other hand chose to use a upper, left-hand triangular matrix. This choice results in an algorithm which is slightly more complex than the one derived here. We refer interested readers to the original references for further details. As in the case of the lattice algorithm, the solution to the adaptive filtering problem is updated along with the linear prediction one. In fact, since we will be calculating the matrix $\hat{Q}_p(n)$, there is clearly no need to include the adaptive filtering problem explicitly in the following analysis.

Note from equations (104), (106), (107) and (108) that

$$Q_{p+1}(n) = \tilde{Q}^b_{p+1}(n)\hat{Q}^b_{p+1}(n)\begin{bmatrix} Q^b_{p+1}(n-1) & o \\ o^T & 1 \end{bmatrix}\begin{bmatrix} 1 & o^T \\ o & Q_p(n-1) \end{bmatrix} \tag{112}$$

which can be viewed as a time and order update relationship for the matrix $Q_p(n-1)$. Also from equations (93), (96) and (97) we have

$$Q_{p+1}(n) = \hat{Q}^f_{p+1}(n+1)\begin{bmatrix} Q^f_{p+1}(n) & o \\ o^T & 1 \end{bmatrix}Q_p(n) \tag{113}$$

or, given that the inverse of an orthogonal matrix is just its transpose,

$$Q_p(n) = \begin{bmatrix} Q^f_{p+1}(n) & o \\ o^T & 1 \end{bmatrix}^T [\hat{Q}^f_{p+1}(n+1)]^T Q_{p+1}(n) \tag{114}$$

which is an order down-date relationship for $Q_{p+1}(n)$. Taken together, equations (112) and (114) represent, at least in principle, a means of updating the matrix Q_p from one time instant to the next. However we are really interested in obtaining a time update relationship for the matrix $\hat{Q}_p(n)$, since the matrix Q_p operates on the entire data matrix and would therefore lead to an algorithm which is not time-recursive and requires an ever increasing amount of storage. Furthermore, since explicit evaluation of the relationships derived above would require $O(n^3)$ multiplications, this approach could not lead to a "fast" algorithm.

The observant reader will have noticed that the matrix product $\tilde{Q}^b_{p+1}(n)\hat{Q}^b_{p+1}(n)$ depends on knowledge of the p^{th} forward linear prediction problem at time n (equations (107) and (108)) and thus is, by assumption, is known; whereas the matrix $\hat{Q}^f_{p+1}(n+1)$ depends on the solution to the p^{th} order backward problem at time n. This is somewhat paradoxical given that one of the purposes of trying to calculate $Q_p(n)$ is to calculate this solution! However, as we shall see, it is possible to avoid this paradox by doing the matrix multiplications required by equations (112) and (114) implicitly, thus computing the matrix $\hat{Q}_p(n)$ efficiently and generating a fast algorithm. We do this by constructing only the right-hand column of the matrix $\hat{Q}_p(n)$ and then inferring the whole matrix from this vector (as in section 2.7). This technique reduces the dimension of the problem (from matrices to vectors) and thus scales down the computational requirement; in fact, the evaluation of equations (112) and (114) is actually reduced to $O(p)$ arithmetic operations in this case. Note that the right-hand column of the matrix $\hat{Q}_p(n)$ can be considered to be the result of applying the rotation matrix $\hat{Q}_p(n)$ to the pinning vector π_n defined in equation (67). The occurrence of the pinning vector in the calculations is typical of fast Kalman algorithms.

Returning to equation (112), we see that

$$\begin{bmatrix} a_{p+1}(n) \\ o \\ \gamma_{p+1}(n) \end{bmatrix} = Q_{p+1}(n)\pi_n = \tilde{Q}^b_{p+1}(n)\hat{Q}^b_{p+1}(n) \begin{bmatrix} Q^b_{p+1}(n-1) & o & \begin{matrix} 0 \\ a_p(n-1) \end{matrix} \\ o^T & 1 & \begin{matrix} o \\ \gamma_p(n-1) \end{matrix} \end{bmatrix} \tag{115}$$

and, from equation (113),

$$\begin{bmatrix} a_{p+1}(n) \\ o \\ \gamma_{p+1}(n) \end{bmatrix} = \hat{Q}^f_{p+1}(n+1) \begin{bmatrix} Q^f_{p+1}(n) & o & a_p(n) \\ o^T & 1 & o \\ & & \gamma_p(n) \end{bmatrix} \tag{116}$$

Now from equation (106), it is clear that the vector $\begin{bmatrix} 0 & a_p^T(n-1) & o^T & \gamma_p(n-1) \end{bmatrix}^T$ is unaffected by the matrix $Q^b_{p+1}(n-1)$. Similarly, the vector $\begin{bmatrix} a_p^T(n) & o^T & \gamma_p(n) \end{bmatrix}^T$ is invariant under the action of the matrix $Q^f_{p+1}(n)$ (see equation (96)). Hence, we deduce that

$$\begin{bmatrix} a_{p+1}(n) \\ o \\ \gamma_{p+1}(n) \end{bmatrix} = \tilde{Q}^b_{p+1}(n)\hat{Q}^b_{p+1}(n) \begin{bmatrix} 0 \\ a_p(n-1) \\ o \\ \gamma_p(n-1) \end{bmatrix} \tag{117}$$

and

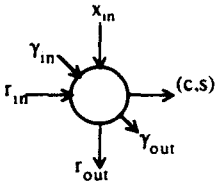
$$\begin{bmatrix} a_{p+1}(n) \\ o \\ \gamma_{p+1}(n) \end{bmatrix} = \hat{Q}^f_{p+1}(n+1) \begin{bmatrix} a_p(n) \\ o \\ \gamma_p(n) \end{bmatrix} \tag{118}$$

Finally, with reference to equation (97), we note that

$$\hat{Q}^f_{p+1}(n+1) \begin{bmatrix} a_p(n) \\ o \\ \gamma_p(n) \end{bmatrix} = \begin{bmatrix} a_p(n) \\ \tilde{a}_{p+1}(n) \\ o \\ \gamma_{p+1}(n) \end{bmatrix} \tag{119}$$

Clearly, if the rotation angle corresponding to $\hat{Q}^f_{p+1}(n+1)$ is Θ then

$$\begin{bmatrix} \tilde{a}_{p+1}(n) \\ \gamma_{p+1}(n) \end{bmatrix} = \begin{bmatrix} \cos\Theta & \sin\Theta \\ -\sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} 0 \\ \gamma_p(n) \end{bmatrix} \tag{120}$$



IF $x_{in} = 0$ THEN

$c \leftarrow 1$; $s \leftarrow 0$; $r_{out} \leftarrow r_{in}$; $\gamma_{out} \leftarrow \gamma_{in}$

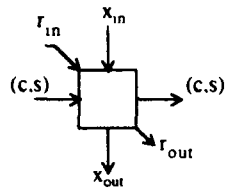
ELSE

$r_{out} \leftarrow \sqrt{r_{in}^2 + x_{in}^2}$;

$c \leftarrow (r_{in}/r_{out})$; $s \leftarrow (x_{in}/r_{out})$;

$\gamma_{out} \leftarrow c\gamma_{in}$;

ENDIF



$x_{out} \leftarrow cx_{in} - sr_{in}$

$r_{out} \leftarrow s^* x_{in} + cr_{in}$

Figure 15 Fast Kalman Processors

and hence the rotation matrix $\hat{Q}_{p+1}^{T}(n+1)$ can be calculated indirectly from the known quantities $a_{p+1}(n)$ and $\gamma_{p+1}(n)$. Thus we can avoid the paradoxical situation of needing to know the solution to the p^{th} order backward linear prediction problem at time n before $\hat{Q}_p(n)$ is known.

Thus by means of equation (117) and we can transform the vector $\begin{bmatrix} a_p^T(n-1) & o^T \\ \gamma_p(n-1) & \end{bmatrix}^T$ into the vector $\begin{bmatrix} a_{p+1}^T(n) & o^T \\ \gamma_{p+1}(n) & \end{bmatrix}^T$ *in O(p) orthogonal operations. Equation (118) then provides the basis* for an *O(p)* method for transforming this latter vector into $\begin{bmatrix} a_p^T(n) & o^T \\ \gamma_p(n) & \end{bmatrix}^T$ and finally $\hat{Q}_p(n)$ can be calculated, again in *O(p)* operations, from $\begin{bmatrix} a_p^T(n) & o^T \\ \gamma_p(n) & \end{bmatrix}^T$ as shown in section 2.7. The resulting algorithm may be implemented using the parallel computing architecture shown in figure 14 with rotation processors as defined in figure 15. Note that these rotation processors are essentially the same as those used in the triangular array and lattice algorithms. The main difference is that the processing elements in figure 15 do not store any internal variables: all variables are either passed into or out of the processing element. In fact if these processing elements are equipped with a storage element and the correct output variable fed back to the relevant input (as shown in the left-hand column of cells in figure 14) then they are essentially identical to the processing elements shown in figure 3.

An interesting consequence of using the QRD approach is that, unlike other fast Kalman algorithms, the QRD-based fast Kalman algorithm not only produces the solution to a given order problem but also that for all lower order problems. This is because the QRD-based approach to least squares minimisation is inherently an order recursive process (see section 3.6). For instance, consider the triangular processor array described in section 2.3. It should be clear that the quantity being passed down to the

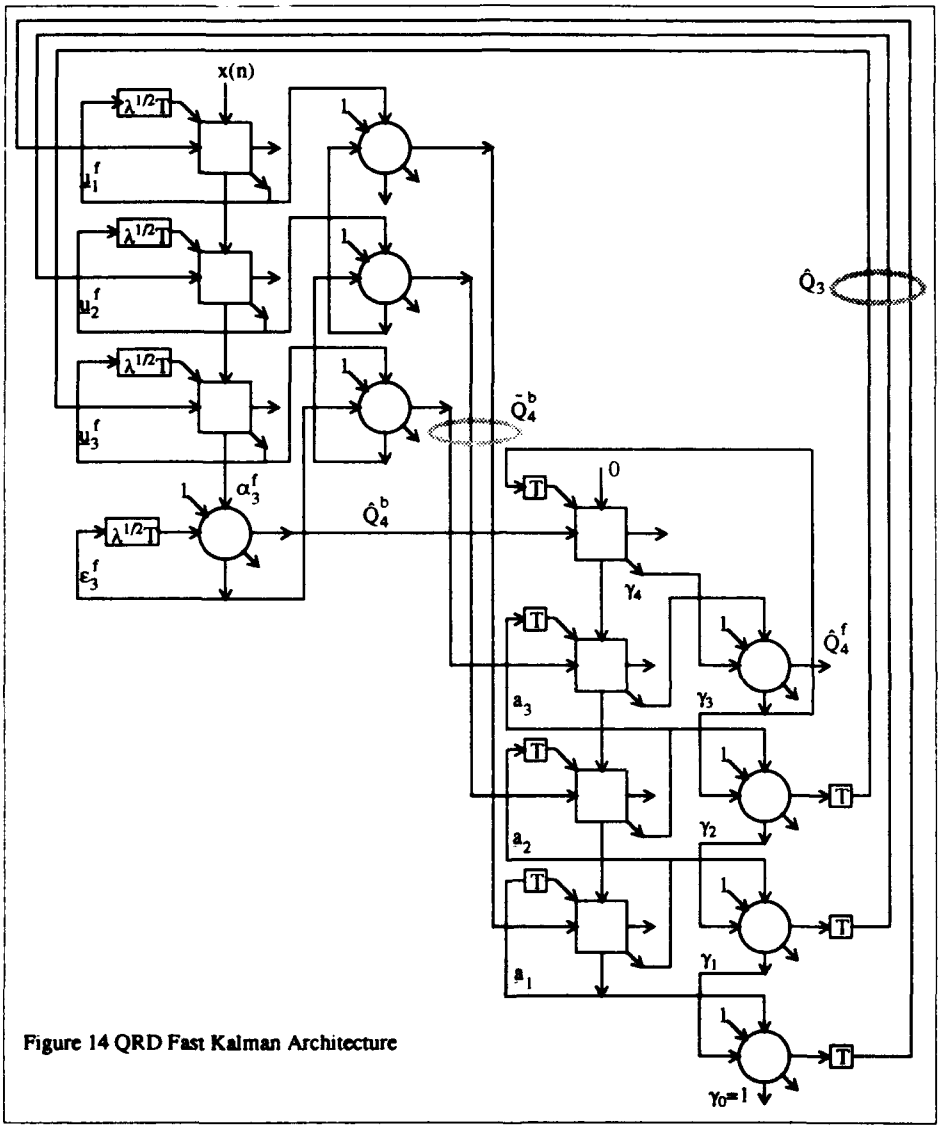


Figure 14 QRD Fast Kalman Architecture

boundary cells (the product of the various cosine terms from earlier rotations) is the quantity $\gamma(N)$ required by the lower order problems. A small amount of thought will also reveal that the angle normalised residual, $\alpha(n)$, for a given lower order problem, is the quantity passed down to a boundary processor from the last internal processor of that particular column (see figure 16).

It is interesting to note that, at first glance, the algorithm pictured in figure 14 appears not to include any quantities related to the backward linear prediction problem; however this is not true. It is possible

to show (see section 3.6) that the vector $\underline{a}_p(n)$ consists of the backward prediction residuals for orders 0 to p-1 normalised by the respective prediction error energy. Indeed Regalia and Bellanger [27] derived their algorithm using these quantities explicitly.

3.6 Physical Interpretation of Fast Algorithm Parameters.

The QRD-based approach to least-squares minimisation is just one of many different ways in which the problem can be solved. However, the quantities used in a QRD-based algorithm appear to be radically different from those to be found in the more familiar approaches. Clearly because the underlying problem is the same, the variables in a QRD-based algorithm must be related to more conventional quantities. In this section we point out some of these relationships along with some other interesting inter-relationships between quantities found in the QRD-based approach. Indeed, the QRD-based approach to least-squares minimisation and linear prediction in particular, offers many useful insights.

We have already seen that the forward and backward prediction residual powers appear quite naturally in the QRD-based lattice algorithm - albeit in terms of their square-roots: ε_p^f and ε_p^b (see equations (97) and (107)). The lattice algorithm also calculates estimates of the partial correlation coefficients [12]. Consider the term μ_{p-1}^f in equation (97): if the rotation angle corresponding to $\hat{Q}_p^f(n)$ is Θ and we let $c = \cos\Theta$ and $s = \sin\Theta$ then

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} \beta\mu_{p-1}^f(n-1) & \beta\varepsilon_{p-1}^b(n-2) \\ \alpha_{p-1}^f(n) & \alpha_{p-1}^b(n-1) \end{bmatrix} = \begin{bmatrix} \mu_{p-1}^f(n) & \varepsilon_{p-1}^b(n-1) \\ \alpha_p^f(n) & 0 \end{bmatrix} \tag{121}$$

i.e.
$$\mu_{p-1}^f(n) = \beta c\mu_{p-1}^f(n-1) + s\alpha_{p-1}^f(n) \tag{122}$$

where
$$c = \frac{\beta\varepsilon_{p-1}^b(n-2)}{\varepsilon_{p-1}^b(n-1)} \quad \text{and} \quad s = \frac{\alpha_{p-1}^b(n-1)}{\varepsilon_{p-1}^b(n-1)} \tag{123}$$

Combining equations (122) and (123) we find, after some algebra, that

$$b_{p-1}(n) = \beta^2 b_{p-1}(n-1) + \alpha_{p-1}^b(n-1)\alpha_{p-1}^f(n) \tag{124}$$

$$= \sum_{m=1}^{n} \beta^{2(n-m)} \alpha_{p-1}^b(m-1)\alpha_{p-1}^f(m) \tag{125}$$

where
$$b_{p-1}(n) = \varepsilon_{p-1}^b(n-1)\mu_{p-1}^f(n) \tag{126}$$

Now, by comparison with equation (47), we have

$$\alpha^f_{p-1}(n) = \sqrt{e^f_{p-1}(n, n-1)e^f_{p-1}(n, n)} \tag{127}$$

and

$$\alpha^b_{p-1}(n-1) = \sqrt{e^b_{p-1}(n-1, n-2)e^b_{p-1}(n-1, n-1)} \tag{128}$$

Thus the angle normalised linear prediction residuals are identical to the so called rationalised residuals[19]. In particular we see that this interpretation of the angle normalised residuals implies that the right hand side of equation (125) can be viewed as a (weighted) estimate of the crosscorrelation between the normalised forward and backward residuals. Indeed for stationary signal statistics, once a recursive least squares prediction algorithm has converged, the a priori and a posteriori residuals, and hence the angle normalised residuals, are identical. Finally, as the backward residual power $[\varepsilon^b_{p-1}]^2$ is effectively an estimate of the mean square backward residual, we see from equation (126) that

$$\mu^f_{p-1}(n) \approx \frac{[e^b_{p-1}(n-1, n-1)e^f_{p-1}(n, n)]}{(e^b_{p-1}(n-1, n-1))^2} \tag{129}$$

In a similar manner, it is possible to show, again from equation (97), that

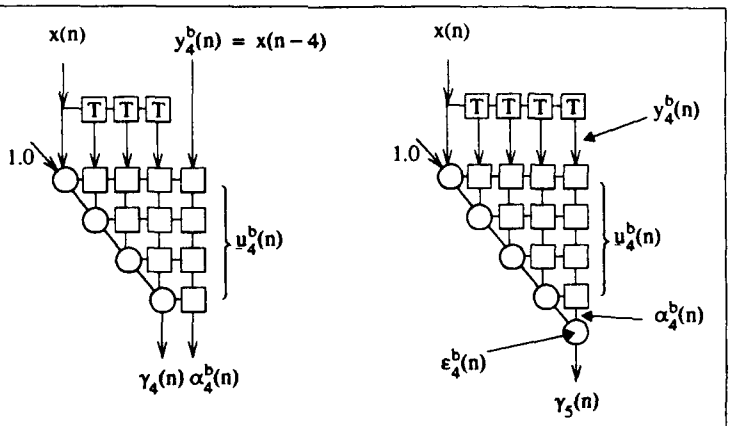
$$\mu_{p-1}(n-1) \approx \frac{[e^b_{p-1}(n-1, n-1)e_{p-1}(n-1, n-1)]}{(e^b_{p-1}(n-1, n-1))^2} \tag{130}$$

and, from equation (107), that

$$\mu^b_{p-1}(n-1) \approx \frac{[e^f_{p-1}(n, n)e^b_{p-1}(n-1, n-1)]}{(e^f_{p-1}(n, n))^2} \tag{131}$$

Thus we see that $\mu^f_{p-1}(n)$, $\mu^b_{p-1}(n-1)$ and $\mu_{p-1}(n-1)$ are estimates of the various PARCOR coefficients.

Next we consider the order-recursive nature of the QRD approach to linear prediction. Note that equations (97) and (98) provide a recursive decomposition of the matrix $R_p(n-1)$. Specifically, and for time n rather (n-1),



a) 4th order Backward Linear Prediction b) 5th order Triangular Array

Figure 16 Order Recursion within the QRD-based LS Algorithm

$$R_p(n) = \begin{bmatrix} R_{p-1}(n) & \underline{u}_{p-1}^b(n) \\ \underline{0}^t & \varepsilon_{p-1}^b(n) \end{bmatrix} \tag{132}$$

This shows that the diagonal elements of the matrix $R_p(n)$ are in fact the square-roots of the backward prediction residual energy terms for each of the sub-order problems. It also indicates why it is sensible for the Givens rotations used in QRD-based linear prediction to ensure that the diagonal elements of the R(n) matrix are always positive. Equation (132) also shows that the off-diagonal parts of the triangular matrix are the various "$\underline{u}$" vectors. This is not surprising considering that the linear prediction problem could be solved using a full pxp systolic array with the time series x(n) fed in via a tapped delay line as indicated in figure 16. It therefore serves to emphasise the fact that a QRD-based approach generates the solution to all sub-order problems as well as the target problem of a given order.

Order recursion also plays an important part in any least-squares lattice algorithm. Traditionally the order recursion for the prediction residuals takes the form [12]

$$\left.\begin{aligned} e_p^f(n, n) &= e_{p-1}^f(n, n) + k_p^f(n) e_{p-1}^b(n-1, n-1) \\ e_p^b(n, n) &= e_{p-1}^b(n-1, n-1) + k_p^b(n) e_{p-1}^f(n, n) \end{aligned}\right\} \tag{133}$$

where $k_p^f(n)$ and $k_p^b(n)$ are the p^{th} order reflection coefficients. Unlike the conventional lattice algorithms, the QRD-based one derived in section 3.4, does not explicitly calculate reflection coefficients; instead the order update for the (angle normalised) residuals takes the form (see equations (97) and (107))

$$\left.\begin{aligned}\alpha_p^f(n) &= c_p^f(n)\alpha_{p-1}^f(n) - s_p^f(n)\beta\mu_{p-1}^f(n-1)\\ \alpha_p^b(n) &= c_p^b(n)\alpha_{p-1}^b(n-1) - s_p^b(n)\beta\mu_{p-1}^b(n-2)\end{aligned}\right\} \tag{134}$$

where

$$\left.\begin{aligned}\mu_{p-1}^f(n) &= c_p^f(n)\beta\mu_{p-1}^f(n-1) + s_p^f(n)\alpha_{p-1}^f(n)\\ \mu_{p-1}^b(n-1) &= c_p^b(n)\beta\mu_{p-1}^b(n-2) + s_p^b(n)\alpha_{p-1}^b(n-1)\end{aligned}\right\} \tag{135}$$

and $c_p^f(n)$, $s_p^f(n)$, $c_p^b(n)$ and $s_p^b(n)$ are the sines and cosines of the transformations $\hat{Q}_p^f(n)$ and $\hat{Q}_p^b(n)$ respectively. Equation (134) appears quite different to the usual lattice equations shown in equation (133); however, note that equation (134) is written in terms of angle normalised residuals and not a-posteriori ones, and that the sines and cosines are functions of the residuals (see equations (97) and (107)). Converting from the angle normalised residuals to the a-posteriori ones using equation (45) and evaluating the sines and cosines we obtain, after some manipulation,

$$\left.\begin{aligned}e_p^f(n, n) &= e_{p-1}^f(n, n) - \frac{\mu_{p-1}^f(n)}{\varepsilon_{p-1}^b(n-1)} e_{p-1}^b(n-1, n-1)\\ e_p^b(n, n) &= e_{p-1}^b(n-1, n-1) - \frac{\mu_{p-1}^b(n-1)}{\varepsilon_{p-1}^f(n)} e_{p-1}^f(n, n)\end{aligned}\right\} \tag{136}$$

from which it follows that

$$\left.\begin{aligned}k_p^f(n) &= -\frac{\mu_{p-1}^f(n)}{\varepsilon_{p-1}^b(n-1)}\\ k_p^b(n) &= -\frac{\mu_{p-1}^b(n-1)}{\varepsilon_{p-1}^f(n)}\end{aligned}\right\} \tag{137}$$

This result is not surprising: the reflection coefficients in equation (133) are defined to be those values that *minimise* the terms $\sum_{m=1}^{n} e_p^f(m, m)$ and $\sum_{m=1}^{n} e_p^b(m, m)$. In other words, the reflection coefficients are the coefficients in a first order least squares minimisation problem. From section 2.1 we know that, when using the QRD technique, the least-squares coefficients are given by

$$\underline{w} = -R^{-1}\underline{u} \tag{138}$$

In the case of a first order problem, both the matrix R and the vector $\underline{u}$ are scalars and for the least-squares minimisation problem shown in equation (133) these quantities equate to those shown in equation (137). This equality can most easily be seen with reference to figure 11. If the least-squares mini-

misation problem is one of first order, then the triangular QRD array (cf. figure 2) will be a single circular processing element and the right-hand column will reduce to a one square processing element. These structures can easily be identified in figure 11 from which the relationships shown in equation (133) can readily be deduced.

As remarked earlier, the QRD-based fast Kalman algorithm is also unusual in that it does not calculate conventional quantities (the optimum coefficients). As described in section 3.5, the QRD-based fast Kalman algorithm calculates the rotation matrix $\hat{Q}_p(n)$. This matrix is then used as shown in equation (21) to compute the vector $\underline{u}_p(n)$ and produce the adaptive filtering residual. However, we note that the vector $\underline{u}_p(n)$ is merely a transformed version of the optimum coefficients $\underline{w}_p(n)$, as discussed above.

Another unusual feature of the algorithm derived in section 3.5 is that it does not appear to make use of any quantities related to the backward prediction problem. This is not true however since the vector $\underline{a}_p(n)$ may be interpreted in terms of the backward prediction residuals. To see the equivalence, consider equations (119),(120) and (123): it is clear that

$$\underline{a}_p(n) = \begin{bmatrix} \underline{a}_{p-1}(n) \\ \gamma_{p-1}(n)\sin\Theta \end{bmatrix} = \begin{bmatrix} \underline{a}_{p-1}(n) \\ \gamma_{p-1}(n)\alpha^b_{p-1}(n) \\ \hline \varepsilon^b_{p-1}(n) \end{bmatrix} = \begin{bmatrix} \underline{a}_{p-1}(n) \\ e^b_{p-1}(n,n) \\ \hline \varepsilon^b_{p-1}(n) \end{bmatrix} \tag{139}$$

and we see that the vector $\underline{a}_p(n)$ consists of the energy normalised backward prediction residuals of order (p-1) and below.

3.7 Weight Extraction from Fast Algorithms

The QRD-based least squares lattice and fast Kalman algorithms presented above are based on the "direct residual extraction" technique and as such produce the adaptive filtering residual without explicitly calculating the optimum weight vector. This is highly desirable in an adaptive filtering context since the residual is the quantity of interest; however in system identification the primary goal is the calculation of the weight vector. The two weight extraction techniques presented for the full QRD-based algorithm - weight flushing (section 2.6) and parallel weight extraction (section 2.7) - are equally applicable to the fast algorithms. Note, however, that neither of the fast algorithms explicitly calculates the triangular matrix R so that the back-substitution method (equation (11)) is not available.

Both the lattice and fast Kalman algorithms use the same processing elements as the full triangular QRD array and can therefore be operated in the frozen mode. If a unit impulse is fed into the frozen filter, its output will clearly be the impulse response of the system i.e. the set of filter weights. Note however that unlike the narrow-band beamformer of section 2, the adaptive filters presented here are not memoryless systems when operated in their frozen mode: the (implicit) tapped delay line must continue to operate. It is therefore necessary to ensure that the tapped delay line is full of zeros before applying the unit impulse. Thus it would require an input time series of length 2p+1 consisting of a single sample

of value unity sandwiched between two sets of p consecutive zero samples.

Although this operation will produce the filters impulse response, it is an invasive procedure - again because of the fact that the system has memory. Having frozen the filter and passed the impulse sequence through it, the state of the filter (the contents of the tapped delay-line) will have been altered compared to the point in time when the filter was frozen. Thus it is not now possible to unfreeze the filter and continue with the adaption process as if nothing had happened. If the system is allowed to adapt from this incorrect filter state then the output will be in error - until sufficient data has been processed so that the error in the filter's state has decayed away. This may well be acceptable in certain situations since this process of converging to the required solution is exactly what happens when the filter is first started. One way in which weight flushing can be made non-invasive is for the contents of the filter delay elements to be stored before the weight flushing begins and the restored before the adaption continues.

Recall from section 2.7, that in order to extract the filter weights in parallel with the adaption process, it is necessary to have available the rotation matrix $\hat{Q}(n)$. This matrix is explicitly calculated in the fast Kalman algorithm and indeed is also available in the lattice algorithm in a disguised form: it can be shown that the rotation matrices $\hat{Q}_i^f(n)$ $(1 \le i \le p)$ are equal to the elementary Givens rotations $G_i(n)$ $(1 \le i \le p)$ that make up the matrix $\hat{Q}(n)$ - see equation (30). Thus it is possible to use the additional hardware shown in figure 8 in conjunction with the fast algorithm (instead of the triangular processor array) to generate the weights. However, the utility of this approach is questionable since the addition of the extra processing means that the computational load of the complete algorithm rises from $O(p)$ to $O(p^2)$ and the advantage of the fast algorithm is lost.

3.8 Computer Simulation.

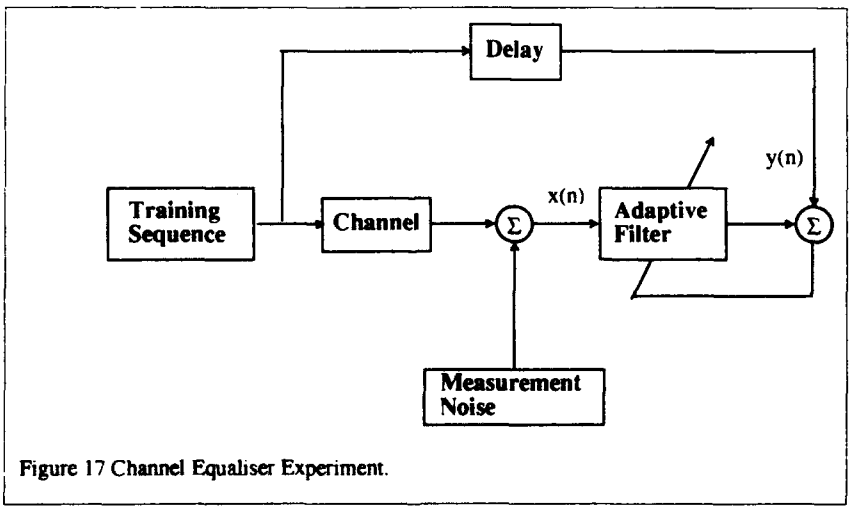


Figure 17 Channel Equaliser Experiment.

The main concern with "fast" algorithms is that they are potentially more sensitive to numerical

errors than their generic couterparts. This is because the fast algorithms exploit some mathematical relationship between various quantities in the generic algorithm in order to reduce the computational load. In the case of the least-squares lattice algorithms, the assumption that the problem has already been solved for the one order allows the solution to the next order problem to be generated efficiently. Now in practice, the calculations can only be done to a finite accuracy so, strictly speaking, the assumptions upon which the fast algorithms are based (e.g. the existence of the solution to the lower order problems) are only approximately true. We can therefore expect to pay some penalty, in terms of numerical stability, for the reduced computational load. The perceived advantage with the fast QRD-based algorithms is, of course, that they should be more robust in the presence of numerical errors than other fast algorithms.

In order to investigate the effects of finite precision on the fast QRD-based algorithms we consider the application of an adaptive filter to a typical channel equalisation problem (figure 17). In particular, we consider[6] the case of an adaptive equaliser applied to a data channel with a "raised cosine" impulse response (equation (140)).

$$h(n) = \begin{cases} \frac{1}{2}\left[1 + \cos\left(\frac{2\pi}{W}(n-2)\right)\right] & n = 1, 2, 3. \\ 0 & \text{otherwise} \end{cases} \qquad (140)$$

By varying the parameter W, the amount of interference between a given symbol and the two either side of it can be changed. This, in effect, controls the eigenvalue spread of the data covariance matrix (see table 2).

An 11^{th} order adaptive QRD-based least-squares adaptive filter is used to equalise the channel response. In the simplest of situations, the equaliser would be trained periodically by transmitting a known sequence and adapting the equaliser with a stored version of this signal as the "reference" signal (see section 3.1). In between these training sessions, with the adaption frozen, the channel can be used for the transmission of data, hopefully with the inter-symbol interference much reduced.

In our computer experiment, the transmission channel is fed with a polar (± 1) pseudorandom training sequence. This sequence, delayed by seven time instants, is used as the reference signal for the adaptive filtering algorithm. The delay is inserted to ensure that the adaptive filter has an impulse response that is symmetrical about the centre tap. A small quantity of "measurement" noise, in the form of a pseudorandom sequence with an approximately gaussian probability distribution function, is added to the channel output. The noise sequence used has zero mean and a variance of 0.001. The "forget factor" β (see equation (4)) was fixed at a value of 0.996, which implies an effective data window (i.e. the duration for which any data vector has an effect on the filter adaption) of 250 time samples.

6. Suggested by S Haykin.

W	$\lvert\lambda_{max}/\lambda_{min}\rvert$
2.9	6.1
3.1	11.2
3.3	21.9
3.5	47.5

Table 2 Eigenvalue Spread

All calculations within the algorithm were performed using limited-precision floating point arithmetic. Only the number of bits in the mantissa is varied during the experiments: the number of bits in the exponent is fixed at eight. No quantities internal to the adaptive filtering algorithm are held to a greater precision than the inputs: the results of *all* arithmetic operations are immediately reduced to the required precision. The numerical performance of most algorithms can be improved by using higher precision for some internal calculations; however it is necessary to have a good understanding of the algorithm and, in particular, to identify the critical intermediate quantities for this method to be used effectively.

The performance of the equaliser is monitored by recording the ensemble-averaged, squared a-priori equalisation error (see equation (40)). This has the advantage that it shows how close to convergence the algorithm is whilst still showing, asymptotically, the least squares equalisation error. The ensemble average is taken over 100 realisations of the experiment. Several experiments were performed using various combinations of parameters and algorithms however only the main results are discussed below. Care should be taken in the interpretation of any computer simulation experiment. In particular, when the numerical stability of an algorithm is being investigated it should be noted [3] that instability is often preceded by a period of apparent stability (e.g. figure 19, 8 bit mantissa plot). Thus simulation experiments can only confirm lower bounds on the sequence length required to cause instability and not "prove" that a system is stable.

Figures 18 and 19 show the basic performance of the fast QRD-based equaliser algorithms, using the square-root, feedforward Givens rotations (SQ/FF), for different values of wordlength and eigenvalue spread. Figure 18 shows that, with double-precision arithmetic, the rate of convergence is more or less insensitive to the different eigenvalue spread settings: as would be expected from a recursive least squares minimisation process. Figure 19 illustrates how the wordlength affects the performance for a fixed eigenvalue spread (W=2.9). There is very little discernable difference between the filters using 12, 16 and 56 (IEEE double-precision) bit mantissas and these filters appear to be well behaved for data sequences of length up to 200. The first sign of any instability appears with a mantissa of 8 bits. Here the filters initially show signs of converging to a stable state but then begin to diverge producing an ever increasing error. The 4 bit systems clearly do not behave in a sensible manner as would be expected from such a short wordlength. Note that there is very little difference between the two fast algorithms (the lattice and the fast Kalman).

Figure 20 shows a comparison of the QRD-based lattice algorithm with the full QRD-based trian-

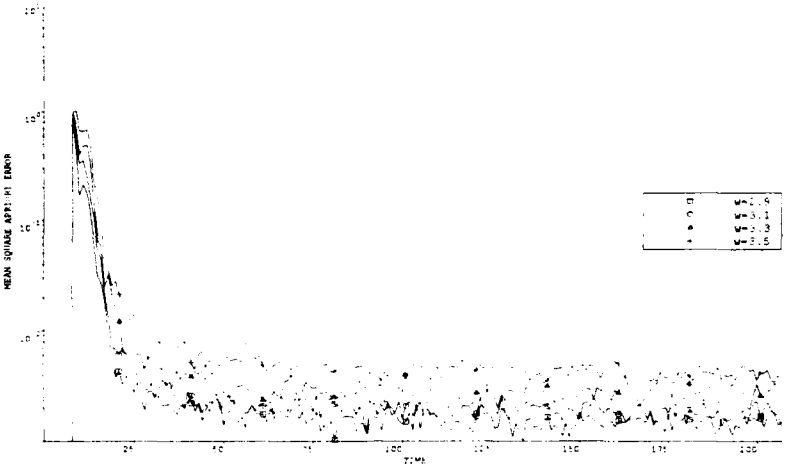


Figure 18(a) SQ/FF QRD Lattice: Effect of Eigenvalue Spread

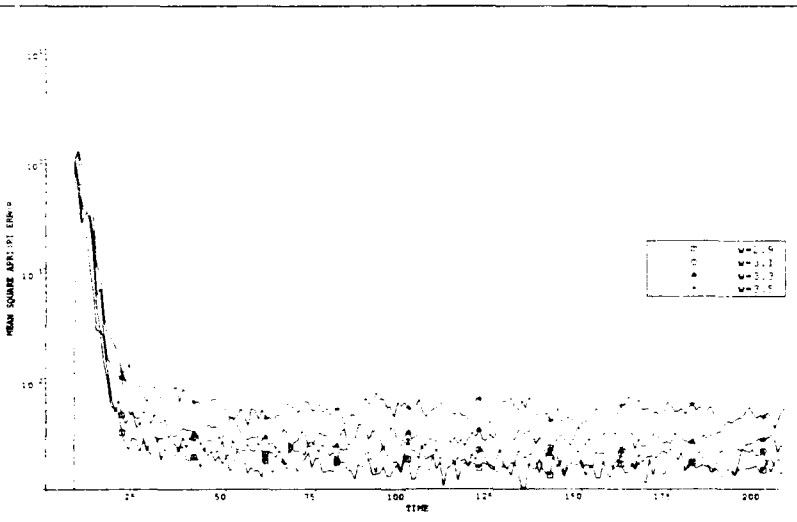


Figure 18(b) QRD fast Kalman: Effect of Eigenvalue Spread.

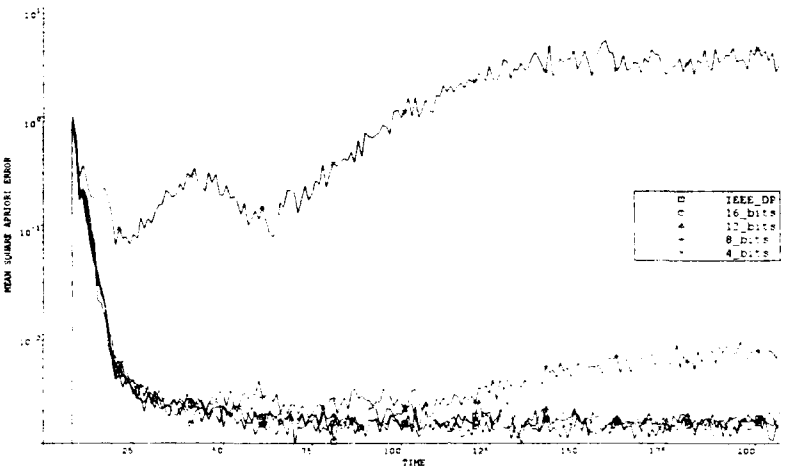


Figure 19(a) SQ/FF QRD Lattice: Effect of Wordlength.

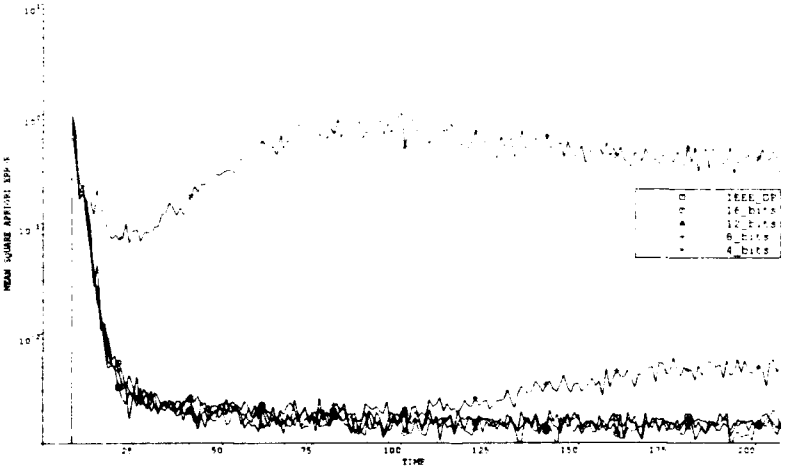


Figure 19(b) QRD fast Kalman: Effect of Wordlength.

gular systolic array version. Four systems are also shown in this figure: they are the "square-root-free with feedback" (SF/FB) forms of the lattice algorithm and the array algorithm along with the SQ/FF versions. This figure shows the case of 4 bit mantissas and a fixed eigenvalue spread setting (W=2.9). This may be considered to be an excessively short wordlength. The reason for this choice is that often finite precision effect are often only manifested after the round-off errors have had time to accumulate [3]. By using a small wordlength, the appearance of such effects occur sooner thus reducing the time necessary to perform the simulation.

In most cases, a p^{th} order RLS adaptive filter will converge within 2p time instants. At this point the a-priori residual will have reached a value primarily determined by the eigenvalue spread and not the wordlength. As round-off errors accumulate, the a-priori error will increase indicating a loss of accuracy in the algorithm. Up to a run length of 10000, the longest simulation run to date by the authors, the SQ/FF lattice algorithm remains stable with 12 bit mantissas. In the case of the SF/FB lattice, the same behaviour is seen using only 4 bit mantissas.

It can be seen, in figure 20, that in the SQ/FF mode the faster, lattice algorithm is only marginally worse than the full triangular array version thus demonstrating that little penalty has been paid in reducing the computational load. As expected, the square-root-free with feedback versions of the algorithms perform better than the basic versions. In this case, there was no discernable difference between the lattice version and the array version in any of the simulations run so far. This would seem to demonstrate the power of the feedback technique in improving the numerical accuracy of these algorithms.

The relative effect of the square-root-free and the feedback techniques can be seen in figure 21. This shows the performance of the lattice algorithms with 4 bit mantissas and fixed eigenvalue spread setting (W=2.9) for the six possible Givens rotation algorithms: SQ/FF, SF/FB, square-root Givens rotations with feedback (SQ/FB), square-root-free feedforward rotations (SF/FF), square-root rotations with the stored parameter fedback (SQ/MFB), and square-root-free rotations with the stored parameter fedback (SF/MFB) - see section 2.4 for more details. From this it can be seen that there is indeed a numerical advantage to avoiding the square-root operation but that the most significant improvement comes about by introducing the "error feedback" - provided that the feedback is applied properly. Figure 21 shows that there is little difference in performance between the "FF" and "MFB" versions whereas the "FB" versions (i.e. the "error feedback" algorithms) are significantly better.

In conclusion, we have found that for both the triangular QRD processor in figure 2 and the least squares lattice filter in figure 12, the best numerical performance over a wide range of computer simulations was obtained using the SF/FB Givens rotations defined in figure 4. The fact that a square-root-free algorithm performs best is not entirely surprising. A closer analysis of the conventional Givens rotation algorithm defined in figure 3 shows that the square root operation is only required in situations where we sum the squares of two numbers and then compute their square root. Avoiding this process could certainly improve the numerical performance. The fact that the feedback form of square-root-free Givens rotation in figure 4 performs best is contrary to the initial expectation of numerical analysts [11]. However, it is entirely consistent with the fact that the corresponding error feedback RMGS lattice al-

This page replaces the original page 52
of RSRE MEMO 4562

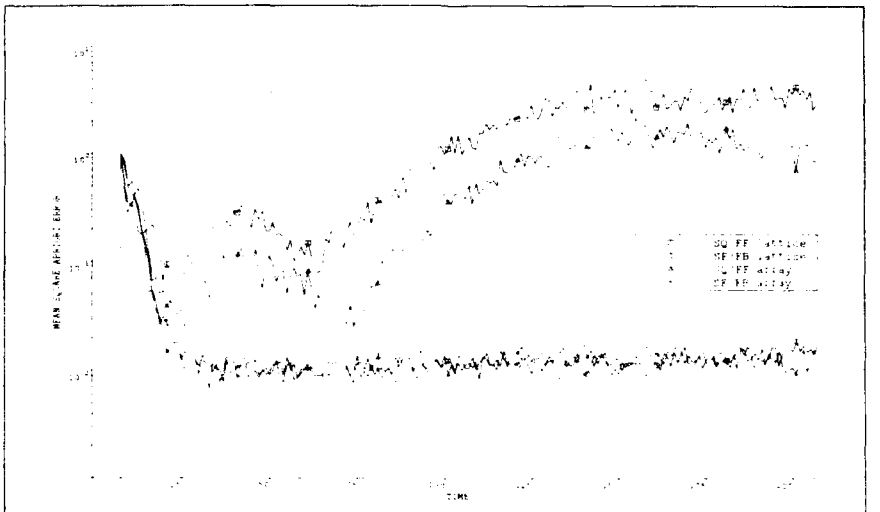


Figure 20 Comparison of Lattice and Array.

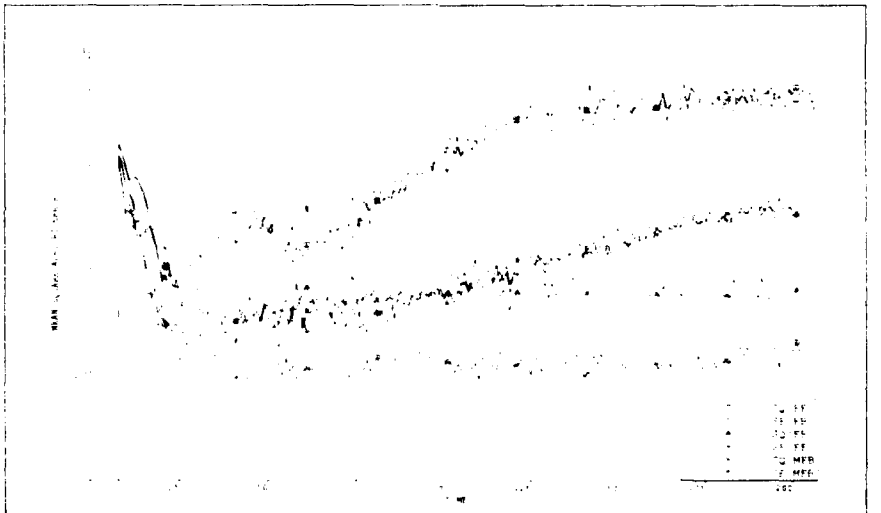


Figure 21 Comparison of Givens Algorithms.

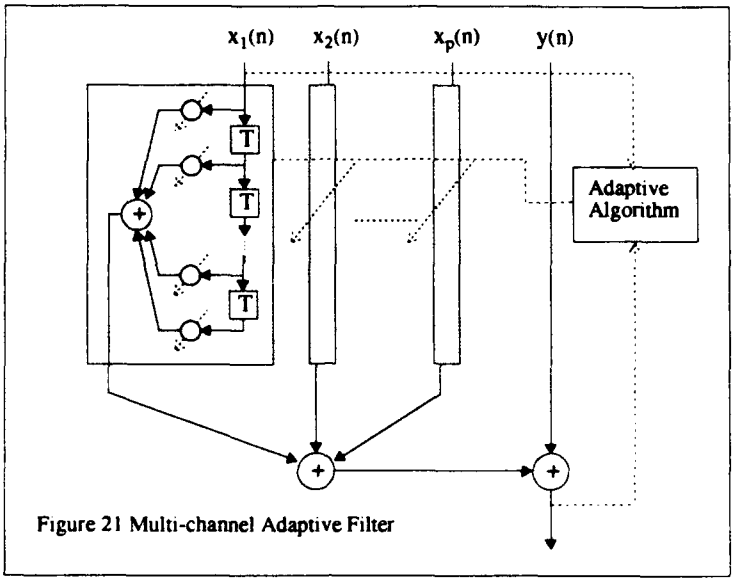


Figure 21 Multi-channel Adaptive Filter

gorithm has also been found to exhibit robust numerical stability. Since Ling, Manolakis and Proakis were first inspired to introduce the error feedback mechanism by analogy with the stability techniques used in control, it would appear that this provides a better way of analysing the numerical stability of other signal processing algorithms.

We have repeated the above experiments with values of the W parameter other than 2.9 and all of the above observations appear to hold essentially independently of the eigenvalue spread.

4 Wide-band Beamforming.

4.1 Multi-channel Adaptive Filters

In a multi-channel least squares adaptive filtering problem at time n, a set of N p-dimensional weight vectors, $\underline{w}_p^{(i)}(n)$ $(0 \le i \le N-1)$, is to be found that minimises the sum of the squared differences between a reference signal y(n) and a linear combination of N samples from each of p data time series $x_i(t_{n-j})$ $(1 \le i \le p, 0 \le j \le N-1)$. This is equivalent to adaptively filtering p separate time series in order to form the best estimate of the reference signal (see figure 21). If the p data sequences come from spatially separate antennae then we have a spatial as well as a temporal filtering problem. In this sense, the multi-channel adaptive filtering problem subsumes both the narrow-band beamforming problem and the (single channel) adaptive filtering problem.

To be specific[7], the measure $E_n(\underline{w}'_N) = \| e_N(n) \|^2$ is to be minimised, where:

$$e_N(n) = X_N(n)\underline{w}'_N + y(n) \qquad (141)$$

$$X_N(n) = B(n) \begin{bmatrix} \underline{x}^T(1) \ldots \underline{x}^T(2-N) \\ \vdots \quad\quad \vdots \\ \underline{x}^T(n) \ldots \underline{x}^T(n-N+1) \end{bmatrix} \tag{142}$$

$$\underline{x}^T(n) = \begin{bmatrix} x_1(n) & x_2(n) & \ldots & x_p(n) \end{bmatrix} \tag{143}$$

$$\underline{w}'_N(n) = \begin{bmatrix} \underline{w}_p^{(0)}(n) \\ \underline{w}_p^{(1)}(n) \\ \vdots \\ \underline{w}_p^{(N-1)}(n) \end{bmatrix}, \tag{144}$$

and

$$\underline{y}(n) = B(n)\,[y(1), \ldots, y(n)]^T \tag{145}$$

Equation (143) serves to define the new vector quantity $\underline{x}(n)$. Note that, apart from the change from scalar to vector quantities, equation (143) is identical to equation (79). Indeed it would be possible to consider the case where the reference signal y(n) is replaced by several such signals. In this case we would have to replace the vector $\underline{y}(n)$ by a matrix. We will not pursue the idea of multichannel joint process estimation any further here but note that a similar situation arises naturally in section 4.2 when we consider multi-channel linear prediction.

The solution of this vector least squares minimisation problem via QR decomposition follows the usual pattern and requires the determination of an orthogonal matrix $Q_N(n)$ that transforms the matrix $X_N(n)$ into upper triangular form. The fact that the matrix $X_N(n)$ is block-Toeplitz allows us to use the ideas developed in section 3 to construct "fast" algorithms. As one might expect, it is possible to derive both a fast Kalman and a lattice algorithm.

4.2 Multi-channel Lattice

The extension of the lattice algorithm presented in section 3.4 to the wide-band beamforming problem is relatively straight forward: the only change required is that certain scalar quantities be replaced by vectors and some vectors be replaced by matrices[24]. The essential features of the derivation presented in section 3.4 carry over exactly. The only point where the derivation of the multi-channel case deviates in any appreciable way from that given in section 3.4 is in the extension to p dimensions of operations on one dimensional objects.

---

7. To be rigorous, we should label quantities involved with a p-channel N-tap multi-channel least squares minimisation problem with two indices (viz p and N). However in the following we will be considering only iterations in the number of taps and not the number of channels. Thus for notational simplicity we will indicate explicitly only the number of taps being considered - the number of channels being assumed to be fixed.

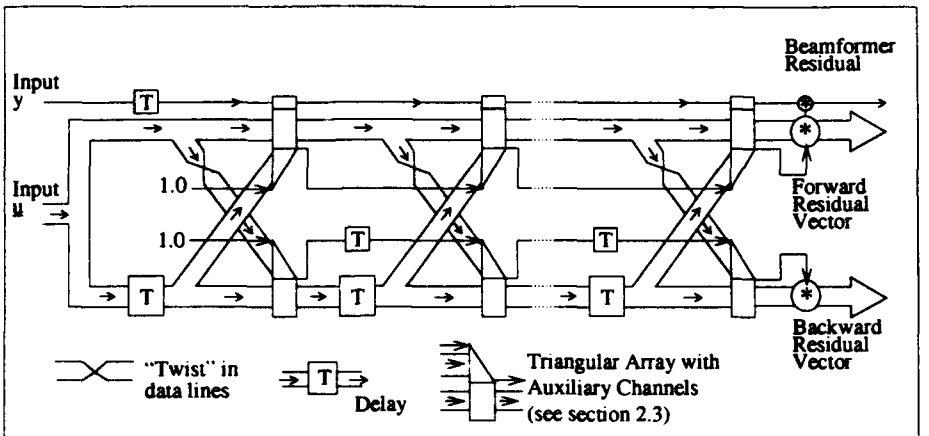


Figure 22 Lattice of Triangular Arrays.

In the solution of the p'" order single-channel forward linear prediction problem it was necessary to determine the rotation matrix $Q_p^f(n-1)$ that annihilated all but one component of the vector $\underline{v}_{p-1}(n-2)$ (see equation (96)). In the N^{th} order multi-channel forward linear prediction problem we have to determine a similar rotation matrix. However in this case, the vector $\underline{v}_{p-1}(n-2)$ is replaced by a suitably defined matrix $V_{N-1}(n-2)$ (with p columns - one for each channel). The equivalent operation to that of $Q_p^f(n-1)$ is to convert $V_{N-1}(n-2)$ into an upper-triangular matrix i.e. to perform a QRD decomposition! Indeed the operation in the single-channel algorithm can be considered to be a QRD decomposition on a vector and the resulting single non-zero component to be a 1×1 triangular matrix. Similarly, the next step in the derivation (cf. equation (97)) consists of calculating the p rotations necessary to perform the recursive QRD update on the above triangular matrix, instead of just one rotation.

The resultant architecture (see figure 22) has a lattice structure where each stage of the lattice contains two triangular systolic arrays. The total number of operations necessary to solve an N^{th} order multi-channel adaptive filtering problem, with p channels, is thus $O(Np^2)$. Note that in figure 22 some of the vector data lines have been "twisted". This merely signifies the fact that data is fed into the triangular arrays in the order specified by the mathematics[8].

The architecture shown in figure 22 is intuitively satisfying for the following reasons. It is well known [28] that the lattice structure of linear prediction algorithms is inherent to the problem. Indeed it is easy to show in more general terms that a p^{th} order linear prediction problem can be solved using a lattice structure which, in effect, solves two least-squares minimisation problems at each stage. These least-squares minimisation problems relate to the determination of the forward and backward reflection coefficients and calculate the order-updated residuals. Specifically, for a multi-channel problem,

8. It is easy to show that this data twist in not necessary since a permutation of the data does not affect the value of the residuals; however we do not pursue this idea any further in this memorandum.

$$\left.\begin{aligned} \underline{e}_p^f(n, n) &= \underline{e}_{p-1}^f(n, n) - K_p^f(n)\underline{e}_{p-1}^b(n-1, n-1) \\ \underline{e}_p^b(n, n) &= \underline{e}_{p-1}^b(n-1, n-1) - K_p^b(n)\underline{e}_{p-1}^f(n, n) \end{aligned}\right\} \quad (146)$$

where $\underline{e}_p^f(n, n)$ and $\underline{e}_p^b(n, n)$ are the p^{th} order forward and backward a-posteriori residual vectors and $K_p^f(n)$ and $K_p^b(n)$ are pxp reflection coefficient matrices, respectively. As we have shown, each of the triangular arrays depicted in figure 22 is capable of performing a recursive least squares minimisation and calculating the residual directly. A close look at figure 22 will show that these triangular arrays operate on the forward and backward residual vectors in precisely the manner required to solve these problems (cf. discussion following equation (137)).

An alternative method [33] for deriving the multi-channel QRD-based lattice algorithm actually begins with the standard multi-channel lattice algorithm and transforms it into a purely orthogonal square-root information algorithm. In the "standard" RLS lattice algorithm, the forward prediction residuals for order p (say) are found by subtracting linear combinations (reflection coefficients) of the $(p-1)^{st}$ order backward prediction residuals from the $(p-1)^{st}$ order forward residuals. A similar relationship holds for the p^{th} order backward residuals. The calculation of the reflection coefficients requires an inversion of the data covariance matrix which is computationally expensive and often ill-conditioned. Using a Cholesky decomposition of the data covariance matrix, Lewis[13] showed how this part of the algorithm could be transformed into a recursive, square-root information process. In this algorithm the reflection coefficients are no longer calculated explicitly: quantities analogous to the vector $\underline{\mu}$ in equation (11) and the vector $\underline{a}$ in equation (91) are calculated instead. Nevertheless the p^{th} order residuals are still calculated in terms of the difference between the $(p-1)^{st}$ order residuals and another term (now a function of the quantities analogous to $\underline{\mu}$ and $\underline{a}$). As the bulk of the calculation is exactly the computation of the reflection coefficients, Lewis proceeded no further with this re-formulation and apparently failed to notice that the "non-orthogonal" part of his algorithm is in fact redundant. Yang and Böhme [33] observed that the adaptive filtering residuals were effectively being produced along with the computation of the vectors $\underline{\mu}$ and $\underline{a}$ - as shown in section 2.5: this observation results in the construction of a purely orthogonal algorithm and is equivalent to that derived from first principles here.

4.3 Multi-channel Fast Kalman Algorithm

The derivation of the multi-channel fast Kalman algorithm is somewhat more difficult than the lattice equivalent. If the various substitutions of scalars for vectors and vectors for matrices are carried out then it is relatively easy to generate a "fast" algorithm. However an operation count will reveal that $O(Np^3)$ operations are required to solve a p channel N^{th} order problem. Assuming that N>p, this is "fast" compared to the $O(N^2p^2)$ operations required when using a generic triangular systolic array fed by tapped delay-lines but falls short of the $O(Np^2)$ operations required by the lattice algorithm. It is possible[2] to generate an $O(Np^2)$ multi-channel fast Kalman algorithm and, as in the single-channel case, the pinning vector is used to allow the inference of a rotation matrix from vector quantities. As before this technique reduces the problem by one dimension and produces an $O(Np^2)$ algorithm.

The "bottle-neck" in the naïve generalisation of the single-channel algorithm (with $O(Np^3)$ opera-

tions) is the calculation equivalent to that shown in equation (108). For convenience we reproduce this equation, in part, below:

$$\tilde{Q}^{b}_{p+1}(n)\begin{bmatrix} \varepsilon^{f}_{p}(n) & \underline{o}^{T} \\ \underline{u}^{f}_{p}(n) & R_{p}(n-1) \\ \underline{o} & O \\ 0 & \underline{o}^{T} \end{bmatrix} = \begin{bmatrix} R_{p+1}(n) \\ O \end{bmatrix} \tag{147}$$

In the multi-channel equivalent, the vector $\underline{u}^{f}_{p}(n)$ is replaced by an Npxp matrix $U^{f}_{N}(n)$ (say) and the scalar $\varepsilon^{f}_{p}(n)$ by a pxp triangular matrix ($E^{f}_{N}(n)$). The operation of annihilating this Npxp matrix - equivalent to a block recursive QRD update - requires $O(Np^{3})$ Givens rotations: it requires $O(p)$ operations to annihilate a p-dimensional vector by rotation against a pxp triangular matrix and the matrix $U^{f}_{N}(n)$ has Np rows.

An alternative procedure for annihilating the matrix $U^{f}_{N}(n)$ is to eliminate it one column at a time rather than one row at a time as in the recursive QRD update (see section 2.2). Each column of $U^{f}_{N}(n)$ has Np components and therefore will require $O(Np)$ operations to annihilate it. If this was all that was required we would have a fast algorithm: $U^{f}_{N}(n)$ has p columns making a total of $O(Np^{2})$ operations. However, it is not sufficient for each column of the matrix $U^{f}_{N}(n)$ just to be annihilated individually: the rotations that annihilate a given column must also be applied to the other columns. Columns that have previously been annihilated clearly will not be affected by this operations but the non-zero ones will be. Thus the rotations that annihilate the first column must be applied to the (p-1) other columns: the rotations that annihilate the second column will have to be applied to (p-2) other columns, and so on. This sequence of steps clearly requires $O(Np^{3})$ operations.

In the above scheme, the column vectors of $U^{f}_{N}(n)$ are subject to, in general, several rotations. This is exactly the situation we faced in the construction of the single-channel fast Kalman algorithm (see equations (112) and (113)). The solution to this problem was to use the pinning vector to effectively condense the rotations down to a single one. Using this idea in the multi-channel case leads to an $O(Np^{2})$ algorithm. The sequence of operations is then as follows: The left-hand column of the matrix $U^{f}_{N}(n)$ is annihilated ($O(Np)$ operations) and this rotation is applied to the second column and a pinning vector ($O(Np)$ operations). The transformed second column is then annihilated ($O(Np)$ operations) and the rotations applied to the transformed pinning vector ($O(Np)$ operations). The doubly transformed pinning vector can then be "unrotated" ($O(Np)$ operations) to generate a rotation that is equivalent to the combined effect of the earlier pair of rotations. This combined rotation is then applied to the third column and another pinning vector and the process continues. At each stage only $O(Np)$ operations are required and thus all p columns of $U^{f}_{N}(n)$ can be annihilated in $O(Np^{2})$ operations.

A more detailed study of the multi-channel fast Kalman algorithm shows that it is, in effect, just the application of the single channel algorithm many times. It should be clear from the above, that the

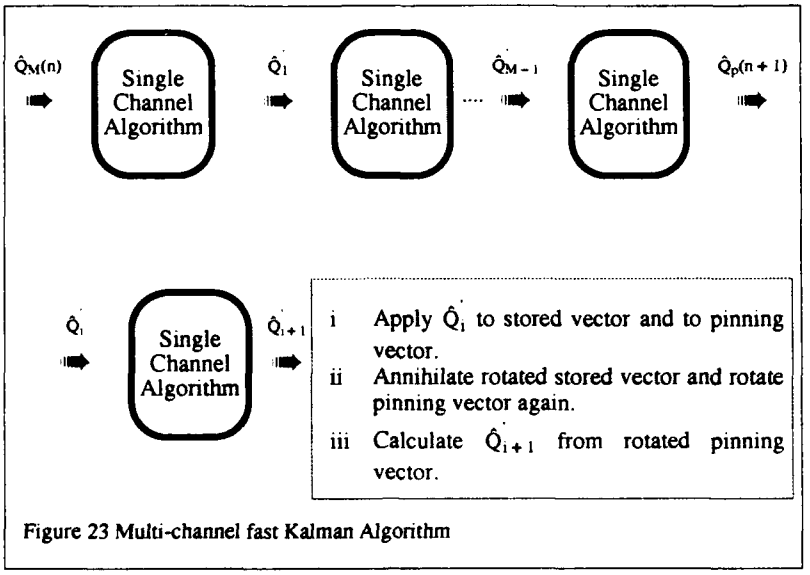


Figure 23 Multi-channel fast Kalman Algorithm

annihilation of one of the columns of the matrix $U'_N(n)$ involves the three steps: the application of a known rotation, annihilation of a vector and subsequent calculation of the combined rotation based on the rotated pinning vector. In the single channel algorithm, exactly the same type of operations are required: the known rotation is $\hat{Q}_p(n)$ and the resultant rotation is $\hat{Q}_p(n+1)$. The multi-channel algorithm thus has the structure shown in figure 23. Each pass of the "single-channel" algorithm requires $O(Np)$ operations (since the matrix $U^f_N(n)$ has columns of dimension Np) and p such passes are required (because the matrix $U^f_N(n)$ has p columns) so the final operation count is $O(Np^2)$.

5 References

[1] S T Alexander and A L Ghirnikar, "A Method for Recursive Least Squares Filtering based upon an Inverse QR Decomposition", *submitted to IEEE trans. SP*, 1991. (See A L Ghirnikar and S T Alexander, "Stable Recursive Least Squares Filtering using an Inverse QR Decomposition", *Proc. IEEE Int. Conf. on ASSP*, pp 1623-1626, Albuquerque, NM, USA, April 1990.)

[2] M G Bellanger and P A Regalia, The FLS-QR Algorithm for Adaptive Filtering: "The Case of Multichannel Signals", *Signal Processing*, vol. 22, no. 2, February 1991.

[3] J M Cioffi, "Finite Precision Effects in Adaptive Filtering", *IEEE trans. CAS*, vol.34, no.7, pp.821-833, July 1987.

[4] J M Cioffi, "The Fast Adaptive Rotors RLS Algorithm", *IEEE trans. ASSP*, vol.38, no.4, pp.631-53, April 1990.

[5] M J Chen and K Yao, "On Realizations of Least Squares Estimation and Kalman Filtering", *Proc. 1st Int. Workshop on Systolic Arrays*, Oxford, pp.161-170, 1986.

[6] F M F Gaston, G W Irwin and J G McWhirter, "Systolic Square-root Covariance Kalman Filtering", *J. VLSI Signal Processing*, vol.2, pp.37-49, 1990.

[7] W M Gentleman, "Least-squares Computations by Givens Transformations without Square-Roots", *J. Inst. Maths. Applic.* vol. 12, pp. 329-336, 1973.

[8] W M Gentleman and H T Kung, "Matrix Triangularisation by Systolic Array", *Proc. SPIE Real Time Signal Processing IV*, vol. 298, 1981.

[9] W Givens, "Computation of Plane Unitary Rotations Transforming a General Matrix to Triangular Form", *J. Soc. Ind. Appl. Math.*, 1958, 6, pp 26-50.

[10] G H Golub, "Numerical Methods for solving Linear Least-Squares Problems", *Num. Math.*, 1965, 7, pp 206-216.

[11] S Hammarling, "A Note on Modifications to the Givens Plane Rotation", *J. Inst. Maths. Applics.*, vol. 13, pp. 215-218, 1974.

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

[13] P S Lewis, "QR-Based Algorithms for Multichannel Adaptive Least Squares Lattice Filters", *IEEE trans. ASSP*, vol.38, no.3, pp.421-32, March 1990.

[14] F Ling, D Manolakis and J G Proakis, "A Flexible, Numerically Robust Array Processing Algorithm and its Relationship to the Givens Transformation", *Proc. IEEE Int. Conf. on ASSP*, April 1986, Tokyo, Japan.

[15] F Ling, D Manolakis and J G Proakis, "A Recursive Modified Gram-Schmidt Algorithm for Least-Squares Estimation", *IEEE Trans. ASSP*, vol.34, no.4, pp. 829-836, Aug. 1986.

[16] F Ling and J G Proakis, "A Generalised Multichannel Least Squares Lattice Algorithm Based on Sequential Processing Stages", *IEEE trans. ASSP*, vol. 32, no. 2, pp. 381-389, Apr. 1987.

[17] F Ling, "Givens Rotation Based Least-squares Lattice and Related Algorithms", *IEEE Trans. SP*, vol.39, no.7, pp.1541-1552, July 1991.

[18] J G McWhirter, "Recursive Least Squares Minimisation using a Systolic Array", *Proc. SPIE Real Time Signal Processing IV*, vol. 431, pp. 105-112, 1983.

[19] J G McWhirter and T J Shepherd, "Least-squares Lattice Algorithm for Adaptive Channel Equalisation", *IEE Proceedings*, vol. 136, Pt. F, no. 6, pp. 532-542, October 1983.

[20] J G McWhirter and T J Shepherd, "Systolic Array Processor for MVDR Beamforming", *IEE Proceedings*, vol. 130, Pt. F, no. 2, pp. 75-80, April 1989.

[21] M Moonen and J Vandewalle, "Recursive Least Squares with Stabilized Inverse Factorization", *IEEE trans. SP*, vol.21, no.1, pp.1-15, January 1990.

[22] C T Pan and R J Plemmons, "Least Squares Modifications with Inverse Factorization: Parallel Implementations", *J. Comput. and Applied Maths.*, vol.27, no.1-2, pp 109-127, 1989.

[23] I K Proudler, J G McWhirter and T J Shepherd, "Fast QRD-based Algorithms for Least Squares

Linear Prediction", *Proc. IMA Conference on Mathematics in Signal Processing*, Warwick, England, 13-15th December 1988.

[24] I K Proudler, J G McWhirter and T J Shepherd, "Computationally Efficient QRD-based Wide-band Beamforming", *Proc. IEEE Int. Conf. on ASSP*, paper no. 348 A13.12, Albuquerque, NM, USA, April 1990.

[25] I K Proudler, J G McWhirter and T J Shepherd, "The QRD-based Least Squares Lattice Algorithm: Some Computer Simulations Using Finite Wordlengths", *Proc. IEEE Int. Symp. on Circuits and Systems*, New Orleans, Lo, USA, 1-3rd May 1990, pp.258-261.

[26] I K Proudler, J G McWhirter and T J Shepherd, "Computationally Efficient, QR Decomposition Approach to Least Squares Adaptive Filtering", *IEE Proceedings*, vol.138, Pt. F, no.4, August 1991, pp.341-353.

[27] P A Regalia and M G Bellanger, "On the Duality Between Fast QR Methods and Lattice Methods in Least Squares Adaptive Filtering", *IEEE Trans. SP*, vol.39, no.4, April 1991.

[28] M J Shensa, "Recursive Least Squares Lattice Algorithms - A Geometric Approach", *IEEE trans. AC*, vol. 26, no. 3, pp. 696-702, June 1981.

[29] T J Shepherd and J Hudson, "Parallel Weight Extraction from a Systolic Adaptive Beamformer", *Proc. IMA Conference on Mathematics in Signal Processing*, Warwick, England, 13-15th December 1988.

[30] B.D. Van Veen and K.M. Buckley, "Beamforming: A Versatile Approach to Spatial Filtering", *IEEE ASSP Magazine*, pp.4-24, April 1988.

[31] A P Varvitsiotis and S Theodoridis, A Pipelined Structure for QR Adaptive LS System Identification, *IEEE Trans. SP*, vol.39, no.8, pp.1920-1923, August 1991.

[32] C R Ward, P J Hargrave, J G McWhirter, A Novel Algorithm and Architecture for Adaptive Digital Beamforming, *IEEE trans. Antennas and Propagation*, vol. AP-34, no.3, 1986, pp.338-346.

[33] B Yang and J F Böhme, "On a Parallel Implementation of the Adaptive Multichannel Least-squares Lattice Filter", *Proc. Int. Symp. on Signals Systems and Electronics*, Erlangen, Fed. Rep. of Germany, Sept. 1989.

6 Appendix

The following algorithms, written in pseudo-ALGOL, are for a narrow-band beamformer and a single channel adaptive filter only. Two narrow-band beamformer algorithms are presented: the first algorithm uses the obvious, feedforward implementation of Givens rotations using square-roots: the second one avoids taking square-roots by calculating transformed quantities and implements the rotations via the feedback algorithm. Both the least squares lattice and fast Kalman algorithms are given but only using the conventional Givens rotations using square-roots. Based on the diagrammatic representations of the algorithms and the interchangeability of the processing elements, it should be possible for the reader to generate any of the "missing" algorithms. In the same spirit, it should be clear how to modify these algorithms to include such aspects as parallel weight-extraction.

The computation count down the right-hand side of the page assumes that the signals being processed are real - although the mathematics is written assuming complex quantities. The complexity of division has been equated to that of square-rooting and multiplication by the exponential weighting factor β is counted as one general purpose multiply - which may not be the case if β is chosen to be of a simple form such as $1 - 2^{-n}$. Note that the algorithms are not optimised: the computational load could be reduced by rewriting the algorithm to take advantage of intermediate quantities common to two or more calculations: such compact forms of the algorithms are not presented here, in order that the regularity of the calculation should not be obscured.

The narrow band beamformer algorithms take as inputs a p-dimensional vector of auxiliary signals $\underline{x}(t)$ and a reference sequence $y(t)$ and calculate the beamformer residual. The adaptive filtering algorithms calculate the filter residual for a p^{th} order system fed with a pre-windowed data sequence $x(t)$ and a reference sequence $y(t)$.

Note: $r_{i,j}(t)$ is the $(i,j)^{th}$ component of the matrix $r(t)$:

$x_i(t)$ is the i^{th} component of the vector $\underline{x}(t)$.

6.1 SQ/FF Narrow-band Beamformer Algorithm

START	add/ subt	mult	sqrt/ divide
INITIALISE {all variables := 0}:			
FOR t FROM 1 DO			
LET $\alpha(t) := y(t)$: $\gamma(t) := 1$:			
FOR i FROM 1 TO p DO			
LET $r_{i,i}(t) := \sqrt{\beta^2 [r_{i,i}(t-1)]^2 + x_i(t)^2}$;	1	3	1
IF $r_{i,i}(t) = 0$ THEN LET $c := 1$: $s := 0$;			
ELSE LET $c := \frac{\beta r_{i,i}(t-1)}{r_{i,i}(t)}$: $s := \frac{x_i(t)}{r_{i,i}(t)}$;	-	1	2
END_IF:			
FOR j FROM i+1 TO p DO			

	add/ subt	mult	sqrt/ divide
LET $x' := (c\underline{x}_j(t) - s r_{i,j}(t-1))$;	1	2	-
$r_{i,j}(t) := (c r_{i,j}(t-1) + s^* \underline{x}_j(t))$;	1	2	-
$\underline{x}_j(t) := x'$			
END_DO;			
LET $\alpha' := (c\alpha(t) - s\underline{u}_i(t-1))$;	1	2	-
$\underline{u}_i(t) := (c\underline{u}_i(t-1) + s^* \alpha(t))$;	1	2	-
$\alpha(t) := \alpha'$; $\gamma(t) := c\gamma(t)$	-	1	-
END_DO; {i loop}			
COMMENT p^{th} order filtered residual COMMENT			
LET $e(t,t) := (\gamma(t)\alpha(t))$	-	1	-
END_DO; {t loop}	p^2+2p	$2p^2+7p+1$	$3p$
FINISH			

6.2 SF/FB Narrow-band Beamformer Algorithm

	add/ subt	mult	sqrt/ divide
START			
INITIALISE {all variables := 0};			
FOR t FROM 1 DO			
LET $e(t,t-1) := y(t)$; $\delta(t) := 1$;			
FOR i FROM 1 TO p DO			
LET $r_{i,i}(t) := (\beta^2 r_{i,i}(t-1) + \delta(t)\, x_i(t)^2)$;	1	3	-
IF $r_{i,i}(t) = 0$ THEN LET $c := 1$; $s := 0$;			
ELSE LET $c := \frac{\beta^2 r_{i,i}(t-1)}{r_{i,i}(t)}$; $s := \frac{\delta(t) x_i(t)}{r_{i,i}(t)}$;	-	2	2
END_IF;			
FOR j FROM i+1 TO p DO			
LET $\underline{x}_j(t) := (\underline{x}_j(t) - r_{i,j}(t-1) x_i(t))$;	1	1	-
$r_{i,j}(t) := (r_{i,j}(t-1) + x_j(t) s^*)$	1	1	-
END_DO;			
LET $e(t,t-1) := (e(t,t-1) - \underline{u}_i(t-1)\underline{x}_i(t))$;	1	1	-
$\underline{u}_i(t) := (\underline{u}_i(t-1) + s^* e(t,t-1))$;	1	1	-
$\delta(t) := c\delta(t)$	-	1	-
END_DO; {i loop}			
COMMENT p^{th} order filtered residual COMMENT			
LET $e(t,t) := (\delta(t) e(t,t-1))$	-	1	-
END_DO; {t loop}	p^2+2p	p^2+7p+1	$2p$
FINISH			

6.3 SQ/FF Lattice Algorithm

START	add/ subt	mult	sqrt/ divide
INITIALISE {all variables := 0}:			
FOR t FROM 1 DO			
LET $\alpha_0^f(t) := x(t)$: $\quad \alpha_0^b(t) := x(t)$:	-	-	-
$\alpha_0(t) := y(t)$: $\quad \gamma_0(t) := 1.0$:	-	-	-
FOR q FROM 1 TO p DO			
LET $\varepsilon_{q-1}^b(t) := \sqrt{\beta^2 (\varepsilon_{q-1}^b(t-1))^2 + \lvert\alpha_{q-1}^b(t)\rvert^2}$:	1	3	1
IF $\varepsilon_{q-1}^b(t) = 0$ THEN LET $c_q^f(t+1) := 1$: $\quad s_q^f(t+1) := 0$:	-	-	-
ELSE LET $c_q^f(t+1) := \dfrac{\beta\varepsilon_{q-1}^b(t-1)}{\varepsilon_{q-1}^b(t)}$: $\quad s_q^f(t+1) := \dfrac{\alpha_{q-1}^b(t)}{\varepsilon_{q-1}^b(t)}$:	-	1	2
END_IF:			
LET $\mu_{q-1}^f(t) := c_q^f(t)\beta\mu_{q-1}^f(t-1) + s_q^{f^*}(t)\alpha_{q-1}^f(t)$:	1	3	-
$\alpha_q^f(t) := c_q^f(t)\alpha_{q-1}^f(t) - s_q^f(t)\beta\mu_{q-1}^f(t-1)$:	1	3	-
$\mu_{q-1}(t) := c_q^f(t+1)\beta\mu_{q-1}(t-1) + s_q^{f^*}(t+1)\alpha_{q-1}(t)$:	1	3	-
$\alpha_q(t) := c_q^f(t+1)\alpha_{q-1}(t) - s_q^f(t+1)\beta\mu_{q-1}(t-1)$:	1	3	-
$\gamma_q(t) := c_q^f(t+1)\gamma_{q-1}(t)$:	-	1	-
COMMENT q-th order forward prediction residual COMMENT			
$e_q^f(t, t) := \gamma_q(t-1)\alpha_q^f(t)$:			
COMMENT q-th order filtered residual COMMENT			
$e_q(t, t) := \gamma_q(t)\alpha_q(t)$:			
LET $\varepsilon_{q-1}^f(t) := \sqrt{\beta^2 (\varepsilon_{q-1}^f(t-1))^2 + \lvert\alpha_{q-1}^f(t)\rvert^2}$:	1	3	1
IF $\varepsilon_{q-1}^f(t) = 0$ THEN LET $c_q^b(t) := 1$: $\quad s_q^b(t) := 0$:	-	-	-
ELSE LET $c_q^b(t) := \dfrac{\beta\varepsilon_{q-1}^f(t-1)}{\varepsilon_{q-1}^f(t)}$: $\quad s_q^b(t) := \dfrac{\alpha_{q-1}^f(t)}{\varepsilon_{q-1}^f(t)}$:	-	1	2
END_IF:			
LET $\mu_{q-1}^b(t-1) := c_q^b(t)\beta\mu_{q-1}^b(t-2) + s_q^{b^*}(t)\alpha_{q-1}^b(t-1)$:	1	3	-
$\alpha_q^b(t) := c_q^b(t)\alpha_{q-1}^b(t-1) - s_q^b(t)\beta\mu_{q-1}^b(t-2)$:	1	3	-
COMMENT q-th order backward prediction residual COMMENT			
$e_q^b(t, t) := \gamma_q(t)\alpha_q^b(t)$:			
END_DO	8	27	6

COMMENT p^{th} order filtered residual COMMENT			
$e_p(t, t) := \gamma_p(t)\alpha_p(t)$;	-	1	-
END_DO			
FINISH	$\overline{8p}$	$\overline{27p+1}$	$\overline{6p}$

6.4 SQ/FF QRD Fast Kalman

START	add/ subt	mult	sqrt/ divide
INITIALISE {all variables := 0};			
FOR q FROM 1 TO p DO			
LET $c_q := 1.0$;	-	-	-
END_DO;			
LET $\gamma_p := 1.0$; -	-	-	
FOR t FROM 1 DO			
LET $\alpha_0^f(t) := x_t(i)$;	-	-	-
FOR q FROM 1 TO p DO			
LET $u_q^f(t) := c_q\beta u_q^f(t-1) + s_q^*\alpha_{q-1}^f(t)$;	p	3p	-
$\alpha_q^f(t) := c_q\alpha_{q-1}^f(t) - s_q\beta u_q^f(t-1)$;	p	3p	-
END_DO;			
COMMENT p^{th} order forward prediction residual COMMENT			
$e_p^f(t, t) := \gamma_p\alpha_p^f(t)$;			
LET $\varepsilon_p^f(t) := \sqrt{\beta^2(\varepsilon_p^f(t-1))^2 + \alpha_p^f(t)^2}$;	1	3	1
IF $\varepsilon_p^f(t) = 0$ THEN LET $c_{p+1}^b := 1.0$; $s_{p+1}^b := 0.0$;	-	-	-
ELSE LET $c_{p+1}^b := \frac{\beta\varepsilon_p^f(t-1)}{\varepsilon_p^f(t)}$; $s_{p+1}^b := \frac{\alpha_p^f(t)}{\varepsilon_p^f(t)}$;	-	1	2
END_IF;			
LET $\gamma_{p+1} := c_{p+1}^b\gamma_p$; $\bar{a}_p := s_{p+1}^b\gamma_p$;	-	2	-
FOR q FROM p TO 1 DO			
LET $\varepsilon_{q-1}^f(t) := \sqrt{(\varepsilon_q^f(t))^2 + u_q^f(t)^2}$;	p	2p	p
IF $\varepsilon_{q-1}^f(t) = 0$ THEN LET $\tilde{c}_q^b := 1.0$; $\tilde{s}_q^b := 0.0$;	-	-	-
ELSE LET $\tilde{c}_q^b := \frac{\varepsilon_q^f(t)}{\varepsilon_{q-1}^f(t)}$; $\tilde{s}_q^b := \frac{u_q^f(t)}{\varepsilon_{q-1}^f(t)}$;	-	-	2p
END_IF;			
LET $\bar{a}_{q-1} := \tilde{c}_q^b a_q + \tilde{s}_q^{b^*} a_q(t-1)$;	p	2p	-

$\underline{a}_{q+1}(t) := \tilde{c}_q^b(t)\underline{a}_q(t-1) - \tilde{s}_q^b(t)\bar{a}_q$;	p	2p	-
END_DO			
LET $\underline{a}_1(t) := a_0$;	-	-	-
LET $\gamma_p := \sqrt{(\gamma_{p+1})^2 + \underline{a}_{p+1}(t)^2}$;	1	2	1
FOR q FROM p TO 1 DO			
LET $\gamma_{q-1} := \sqrt{(\gamma_q)^2 + \underline{a}_q(t)^2}$;	p	2p	p
IF $\gamma_{q-1} = 0$ THEN LET $c_q := 1$; $s_q := 0$;	-	-	-
ELSE LET $c_q := \frac{\gamma_q}{\gamma_{q-1}}$; $s_q := \frac{\underline{a}_q(t)}{\gamma_{q-1}}$;	-	-	2p
END_IF;			
END_DO;			
LET $\alpha_0(t) := y(t)$;	-	-	-
FOR q FROM 1 TO p DO			
LET $\underline{u}_q(t) := c_q\beta\underline{u}_q(t-1) + s_q^*\alpha_{q-1}(t)$;	p	3p	-
$\alpha_q(t) := c_q\alpha_{q-1}(t) - s_q\beta\underline{u}_q(t-1)$;	p	3p	-
END_DO;			
COMMENT p^{th} order filtered residual COMMENT			
$e_p(t, t) := \gamma_p\alpha_p(t)$;	-	1	-
END_DO			
FINISH	8p+4	20p+7	5p+3

INTENTIONALLY BLANK

REPORT DOCUMENTATION PAGE

DRIC Reference Number (if known) .....

Overall security classification of sheet .....UNCLASSIFIED.....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the field concerned must be marked to indicate the classification eg (R), (C) or (S).

Originators Reference/Report No.	Month	Year
MEMO 4562	JANUARY	1992

Originators Name and Location
RSRE, St Andrews Road
Malvern, Worcs WR14 3PS

Monitoring Agency Name and Location

Title
QR DECOMPOSITION BASED ALGORITHMS AND ARCHITECTURES FOR LEAST-SQUARES ADAPTIVE FILTERING

Report Security Classification	Title Classification (U, R, C or S)
UNCLASSIFIED	U

Foreign Language Title (in the case of translations)

Conference Details

Agency Reference	Contract Number and Period

Project Number	Other References

Authors	Pagination and Ref
PROUDLER, I K; McWHIRTER, J G	65

Abstract

In this memorandum we show how the method of QR decomposition (QRD) may be applied to the adaptive filtering and beamforming problems. QR decomposition is a form of orthogonal triangularisation which is particularly useful in least squares computations and forms the basis of some very stable numerical algorithms. When applied to the problem of narrowband adaptive beamforming where the data matrix, in general, has no special structure, this technique leads to an architecture which can carry out the required computations in parallel using a triangular array of relatively simple processing elements.

The problem of an adaptive time series filter is also considered. Here the data vectors exhibit a simple time-shift invariance and the corresponding data matrix is of Toeplitz structure. In this case, the triangular processor array is known to be very inefficient. Instead, it is shown how the Toeplitz structure may be used to reduce the computational complexity of the QR decomposition technique. The resulting orthogonal least squares lattice and "fast Kalman" algorithms may be implemented using far fewer processing elements. These "fast" QRD algorithms are very similar to the more conventional ones but, in general, are found to have superior numerical properties.

Abstract Classification (U,R,C or S)
U

Descriptors

Distribution Statement (Enter any limitations on the distribution of the document)

UNLIMITED

cont/s

INTENTIONALLY BLANK