An Inverse Factorization Algorithm for Linear Prediction

James Nagy*
Department of Mathematics
North Carolina State University
Raleigh, North Carolina 27695-8205

and

Robert J. Plemmons†
Department of Mathematics and Computer Science
Wake Forest University
P.O. Box 7388
Winston-Salem, North Carolina 27109

Submitted by Biswa Datta

ABSTRACT

A new inverse factorization technique is presented for solving linear prediction problems arising in signal processing. The algorithm is similar to a scheme of Qiao in that it uses the rectangular Toeplitz structure of the data to recursively compute the prediction error and to solve the problem when the optimum filter order has been found. The novelty of the scheme presented here is the use of an inverse factorization scheme due to Pan and Plemmons for solving the linear prediction problem with low computational complexity and without the need for solving triangular systems. We also provide a linear systolic array for solving these problems.

1. INTRODUCTION

In linear prediction, one estimates a sample of a signal by a linear combination of its past values. That is, given a sequence $\{t_i\}_{i=1}^{n}$, one finds

*Research supported by the US Air Force under grant AFOSR-91-0388.
†Research supported by the US Air Force under grant AFOSR-91-0388.

655 Avenue of the Americas, New York, NY 10010 0024-3795/92/$5.00

coefficients $a_1, a_2, \ldots, a_p$ ($p < n$) which minimize the error function

$$E(p) = \sum_{i=1}^{n+p} \left(t_i + a_1 t_{i-1} + \cdots + a_p t_{i-p}\right)^2.$$

The parameter p is problem dependent and is called the order of the predictor.

The linear prediction (LP) problem can be interpreted as finding a vector $a^T(p) = [a_1, a_2, \ldots, a_p]$ which minimizes

$$E(p) = \|T(p)a(p) + x(p)\|_2^2,$$

where $T(p)$ is an $(n + p) \times p$ column circulant matrix whose first column is

$$[0 \quad t_1 \quad \cdots \quad t_n \quad 0 \quad \cdots \quad 0]^T$$

and $x^T(p) = [t_1, \ldots, t_n, 0, \ldots, 0]$. That is, we wish to solve the least squares problem

$$\min \|T(p)a(p) + x(p)\|_2 \tag{1}$$

for the LP vector $a(p)$, where $\|\cdot\|_2$ denotes the usual Euclidean norm.

The solution to a least squares problem such as (1) can be found in many different ways. In particular, consider the normal equations

$$T^T(p)T(p)a(p) = -T^T(p)x(p).$$

The matrix $T^T(p)T(p)$ is now a symmetric, positive definite [provided $T(p)$ has full column rank] $p \times p$ Toeplitz matrix. This system can then be solved using any of the fast [$O(p^2)$] Toeplitz solvers such as Levinson's algorithm [9], or using the more recent superfast [$O(p \log^2 p)$] methods such as that of Ammar and Gragg [3]. But it is well known [9] that the normal equations approach to a least squares problem can produce inaccurate solutions for ill-conditioned problems, due to the squaring of the condition number of $T(p)$.

The most stable methods for solving a least squares problem are based on the QR factorization [9]. Since $T(p)$ is a circulant matrix, one can take advantage of its structure to obtain a fast QR factorization of $T(p)$. The lattice algorithm [7] uses this observation. More recently, Qiao [15] has proposed a more efficient algorithm for solving the LP problem. Qiao's method recursively computes $R(p)$ and $u(p) = Q^T(p)x(p)$, where $T(p) = Q(p)R(p)$, until an optimal p is reached. Then $a(p)$ is computed using back substitution.

In this paper we describe a method which is based on a fast QR factorization algorithm of Bojanczyk, Brent, and de Hoog [5] and the generalization of their work by Chun, Kailath, and Lev-Ari [6]. More specifically, we recursively compute $R^{-1}(p)$ and $u(p) = Q^T(p)x(p)$ using hyperbolic rotations (see [9, Section 12.6.4]) until an optimal p is reached. Then the solution to the LP problem is given by $a(p) = -R^{-1}(p)u(p)$. The numerical complexity of the algorithm proposed in this paper is similar to that of Qiao. But in our algorithm no back solve is required, making it more amenable to parallel processing and to an efficient systolic array implementation.

The question of when an optimal p is attained needs to be addressed. Although this is an important topic in linear prediction, it will not be a main part of our discussion. Rather, our focus will be on solving the recursive least squares problems (1). For a more complete discussion on finding the order of the predictor see [8, 11, 12]. To make our LP algorithm complete, though, we will need some stopping criterion. One of the simplest is based on the fact that, as p increases, $E(p)$ levels off. Moreover, as p increases $T(p)$ can become ill-conditioned. Thus, we would like to find a minimum $p = p_0$ such that, for $p > p_0$, $E(p)$ is relatively flat. One way to test for flatness is the threshold test, which requires that

$$1 - \frac{E(p+1)}{E(p)} < \delta$$

for several consecutive steps [11]. The value of δ and the number of consecutive steps are chosen empirically by the user. This is the stopping criterion used by Qiao [15] and, for consistency in comparing methods, will be the one we employ in our algorithm.

Our algorithm is based on hyperbolic rotations and their relation to rank-1 downdating, which we review in Section 2. In Section 3 we show how $R(p)$ can be recursively computed, and in Section 4 we describe how this is modified to construct $R^{-1}(p)$. In Sections 5 and 6 the LP algorithm is derived. In Section 7 the algorithm is stated and some numerical experiments comparing our method with that of Qiao [14] are discussed. In Section 8 we develop a parallel implementation for the inverse factorization LP algorithm.

2. RANK-1 DOWNDATING

The derivation of our linear prediction algorithm depends on ideas from rank-1 downdating. More specifically, let R be an $n \times n$ upper triangular matrix and let $z \in \Re^n$. Downdating refers to the problem of finding an upper triangular matrix $\hat{R}$ such that

$$\hat{R}^T\hat{R} = R^TR - zz^T. \tag{2}$$

If $\hat{R}$ exists [2, 9], then it can be easily found using hyperbolic rotations.

In this section we describe hyperbolic rotations and how they can be used to introduce zeros into selected entries of a matrix. Once this is done, it follows easily that, by applying a sequence of hyperbolic rotations, we can find $\hat{R}$ satisfying (2). Our discussions on these topics follow those given in Golub and Van Loan [9].

An elementary hyperbolic rotation $H(i, j, \theta)$ is a matrix which is the identity everywhere except $h_{ii} = h_{jj} = c$ and $h_{ji} = h_{ij} = -s$, where $c = c(\theta) = \cosh\theta$ and $s = s(\theta) = \sinh\theta$ for some θ. That is,

$$H(i,j,\theta) = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & \cdots & -s & & \\ & & \vdots & & \vdots & & \\ & & -s & \cdots & c & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix}.$$

Let $b \in \Re^n$, $b^T = [b_1 \ \cdots \ b_i \ \cdots \ b_j \ \cdots \ b_n]$, with $b_i \neq b_j$. If $|b_i| > |b_j|$ and we choose

$$c = \frac{b_i}{\sqrt{b_i^2 - b_j^2}} \quad \text{and} \quad s = \frac{b_j}{\sqrt{b_i^2 - b_j^2}}, \tag{3}$$

then

$$H(i,j,\theta)b = \begin{bmatrix} b_1 & \cdots & \bar{b}_i & \cdots & 0 & \cdots & b_n \end{bmatrix}^T,$$

where $\bar{b}_i = \sqrt{b_i^2 - b_j^2}$. A similar choice for c and s can be made if $|b_j| > |b_i|$. Thus hyperbolic rotations can be used to introduce zeros into a vector.

If we are required to introduce zeros into a matrix, we simply let b be the column of the matrix which contains the element to be zeroed, and construct the hyperbolic rotation as above. We point out that in practice there are more stable ways to choose c and s than (3) [2].

Notice that $H(i, j, \theta)$ is completely defined by the scalars $c = c(\theta)$ and $s = s(\theta)$. Moreover, $H(i, j, \theta)$ is constructed using only two scalars, b_i and b_j. Because of this observation, we will often use

$$[c, s] = [c(\theta), s(\theta)] = \text{hyp}(b_i, b_j)$$

to denote the hyperbolic rotation $H(i, j, \theta)$ which rotates the ith row into the jth row and zeros out b_j.

In the downdating problem, we attempt to find a product of hyperbolic rotations of the form

$$H = H(n, n+1, \theta_n) \cdots H(2, n+1, \theta_2) H(1, n+1, \theta_1)$$

so that

$$H \begin{bmatrix} R \\ z^T \end{bmatrix} = \begin{bmatrix} \hat{R} \\ 0^T \end{bmatrix},$$

where $\hat{R}$ is an upper triangular matrix. It can be shown that if $R^TR - zz^T$ is nonsingular, then such a product of hyperbolic rotations can be found [2].

Now, since $c^2(\theta_i) - s^2(\theta_i) = 1$, it follows that

$$H^T(i, n+1, \theta_i) S H(i, n+1, \theta_i) = S,$$

where

$$S = \begin{bmatrix} I_{n-1} & \\ & -1 \end{bmatrix}.$$

Thus $H^TSH = S$. Using this result, we see that

$$\begin{aligned} R^TR - zz^T &= [R^T \quad z] S \begin{bmatrix} R \\ z^T \end{bmatrix} = [R^T \quad z] H^T S H \begin{bmatrix} R \\ z^T \end{bmatrix} \\ &= [\hat{R}^T \quad 0] \begin{bmatrix} \hat{R} \\ 0^T \end{bmatrix} = \hat{R}^T\hat{R}, \end{aligned}$$

and hence $\hat{R}$ satisfies (2).

3. COMPUTING $R(p+1)$ FROM $R(p)$

Let $\{t_i\}_{i=1}^n$ be a given sequence, and define $x^T(0) = [t_1, \ldots, t_n]$ and $x(p)$, $T(p)$, and $a(p)$ as in Section 1. Suppose $T(p) = Q(p)R(p)$ is the QR factorization of $T(p)$, where $Q(p)$ is $(n+p) \times p$. Observe that when $p = 1$ we have $T^T(1) = [0, t_1, \ldots, t_n]$. Thus

$$Q(1) = \frac{1}{\|T(1)\|_2} T(1) \quad \text{and} \quad R(1) = r_{11} = \|T(1)\|_2 = \|x(0)\|_2.$$

Since $T(p+1)$ has the Toeplitz structure, one can partition $T(p+1)$ as

$$T(p+1) = \begin{bmatrix} T(p) & y(p) \\ 0^T & t_n \end{bmatrix} = \begin{bmatrix} 0 & 0^T \\ x(p) & T(p) \end{bmatrix}, \tag{4}$$

where $y^T(p) = [0, \ldots, t_1, \ldots, t_{n-1}]$ is an $n+p$-vector. The following lemma shows the relationship between $R(p+1)$ and $R(p)$.

LEMMA 1. *Given the QR factorization $T(p) = Q(p)R(p)$, if $T(p+1) = Q(p+1)R(p+1)$ is the QR factorization of $T(p+1)$, then $R(p+1)$ can be partitioned as*

$$R(p+1) = \begin{bmatrix} R(p) & \tilde{z}(p) \\ 0^T & r_{p+1,p+1} \end{bmatrix} = \begin{bmatrix} r_{11} & z^T(p) \\ 0 & R_b(p) \end{bmatrix}, \tag{5}$$

where $z(p)$ and $\tilde{z}(p)$ are p-vectors and $R_b(p)$ is a $p \times p$ upper triangular matrix.

Proof. The proof essentially follows from Lemma 1 in [15]. ■

Since $T(p) = Q(p)R(p)$ and $T(p+1) = Q(p+1)R(p+1)$, using the relation

$$T^T(p+1)T(p+1) = R^T(p+1)R(p+1),$$

we obtain for $R_b(p)$ in (5) the relation

$$R_b^T(p)R_b(p) = R^T(p)R(p) - z(p)z^T(p) \tag{6}$$

where

$$z^T(p) = \frac{x^T(p)T(p)}{r_{11}}. \tag{7}$$

Observe that $z^T(p) = [z^T(p-1)\ z_p]$ for some scalar z_p. Then from Equation (5), to compute $R(p+1)$ from $R(p)$, we need z_p and the last column of $R_b(p)$. Since z_p can be computed from (7), the last column of $R_b(p)$ can be computed, as in Section 2, via hyperbolic rotations. In particular, one can construct $H(p)$, a product of hyperbolic rotations, such that

$$H(p)\begin{bmatrix} R(p) \\ z^T(p) \end{bmatrix} = \begin{bmatrix} R_b(p) \\ 0^T \end{bmatrix}. \tag{8}$$

Thus a recursive algorithm for computing $R(p+1)$ can be derived from Equations (5), (7), and (8).

4. COMPUTING $R^{-1}(p+1)$ FROM $R^{-1}(p)$

In the last section we saw how one can compute $R(p+1)$ from $R(p)$. In this section we modify the approach to generate $R^{-1}(p+1)$ from $R^{-1}(p)$. Notice first that since $R(1) = \|x(0)\|_2$, it follows that $R^{-1}(1) = 1/\|x(0)\|_2$.

If $R(p+1)$ is partitioned as in (5), then $R^{-T}(p+1)$ can be partitioned as

$$R^{-T}(p+1) = \begin{bmatrix} R^{-T}(p) & 0 \\ -\dfrac{1}{r_{p+1,p+1}}\tilde{z}^T(p)R^{-T}(p) & \dfrac{1}{r_{p+1,p+1}} \end{bmatrix} \tag{9}$$

$$= \begin{bmatrix} \dfrac{1}{r_{11}} & 0^T \\ -\dfrac{1}{r_{11}}R_b^{-T}(p)z(p) & R_b^{-T}(p) \end{bmatrix}. \tag{10}$$

From Equations (9) and (10) we notice that if $R^{-T}(p)$ is known, then to compute $R^{-T}(p+1)$ we need $z(p)$ and the last row of $R_b^{-T}(p)$. Since $z(p)$ can be computed from (7), we need only find the last row of $R_b^{-T}(p)$. The following lemma shows how this can be done.

LEMMA 2. *Let* $b(p) = R^{-T}(p)z(p)$, *and let*

$$H(p) = H(p, p+1, \theta_p) \cdots H(2, p+1, \theta_2) H(1, p+1, \theta_1)$$

be a product of hyperbolic rotations, where $H(i, p+1, \theta_i)$ *rotates the* i*th row into the* $(p+1)$*st row, such that*

$$H(p)\begin{bmatrix} b(p) \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ \gamma(p) \end{bmatrix} \tag{11}$$

with $\gamma(p) = \sqrt{1 - \|b(p)\|_2^2}$. *Then*

$$H(p)\begin{bmatrix} R(p) \\ z^T(p) \end{bmatrix} = \begin{bmatrix} R_b(p) \\ 0^T \end{bmatrix}$$

and

$$H(p)\begin{bmatrix} R^{-T}(p) \\ 0^T \end{bmatrix} = \begin{bmatrix} R_b^{-T}(p) \\ h^T(p) \end{bmatrix}$$

for some vector $h(p)$.

Proof. See Theorem 8 in [13]. ∎

We now describe how the transformation matrix $H(p)$ in Lemma 2 can be constructed. Suppose

$$H(p-1) = H(p-1, p, \theta_{p-1}) \cdots H(2, p, \theta_2) H(1, p, \theta_1)$$

is known and we wish to find $H(p)$. Notice that

$$b(p) = R^{-T}(p)z(p) = \begin{bmatrix} R^{-T}(p-1) & 0 \\ -\frac{1}{r_{pp}}\tilde{z}^T(p-1)R^{-T}(p-1) & \frac{1}{r_{pp}} \end{bmatrix} \times \begin{bmatrix} z(p-1) \\ z_p \end{bmatrix} = \begin{bmatrix} b(p-1) \\ b_p \end{bmatrix},$$

where

$$b_p = e_p^T R^{-T}(p)z(p). \tag{12}$$

Thus

$$\begin{bmatrix} b(p) \\ 1 \end{bmatrix} = \begin{bmatrix} b(p-1) \\ b_p \\ 1 \end{bmatrix}.$$

It follows that

$$H(p-1, p+1, \theta_{p-1}) \cdots H(1, p+1, \theta_1) \begin{bmatrix} b(p-1) \\ b_p \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ b_p \\ \gamma(p-1) \end{bmatrix},$$

where $\gamma(p-1) = \sqrt{1 - b_1^2 - b_2^2 - \cdots - b_{p-1}^2}$. Thus to determine $H(p)$, we need only find $H(p, p+1, \theta_p)$. That is, we need to find c_p and s_p such that

$$\begin{bmatrix} c_p & -s_p \\ -s_p & c_p \end{bmatrix} \begin{bmatrix} b_p \\ \gamma(p-1) \end{bmatrix} = \begin{bmatrix} 0 \\ \gamma(p) \end{bmatrix} \tag{13}$$

and $c_p^2 - s_p^2 = 1$. Using the notation of Section 2, we can write this as $[c_p, s_p] = \text{hyp}(\gamma(p-1), b_p)$. Then it follows that

$$H(p) = H(p, p+1, \theta_p) \cdots H(2, p+1, \theta_2) H(1, p+1, \theta_1).$$

Note that $H(p)$ is completely determined by c_k and s_k, $k = 1, \ldots, p$. Thus at the pth stage of the algorithm we need only compute c_p and s_p which satisfy the above conditions.

5. AN ALGORITHM TO COMPUTE $R^{-T}(p+1)$

In the last section we showed how c_p and s_p, which completely determine $H(p)$, can be recursively computed. Lemma 2 then indicates how the last row of $R_b^{-T}(p)$ can be computed in $O(p^2)$ multiplications. Direct application of Lemma 2, however, is not required. In this section we show how the last row of $R_b^{-T}(p)$ can be constructed in $O(p)$ multiplications. The scheme in turn gives the last row of $R^{-T}(p+1)$.

We know $H(p-1) = H(p-1, p, \theta_{p-1}) \cdots H(1, p, \theta_1)$ satisfies

$$H(p-1)\begin{bmatrix} R^{-T}(p-1) \\ 0^T \end{bmatrix} = \begin{bmatrix} R_b^{-T}(p-1) \\ h^T(p-1) \end{bmatrix}.$$

Furthermore,

$$\begin{bmatrix} R_b^{-T}(p) \\ h^T(p) \end{bmatrix} = H(p)\begin{bmatrix} R^{-T}(p) \\ 0^T \end{bmatrix} = H(p)\begin{bmatrix} R^{-T}(p-1) & 0 \\ l^T(p) & \\ 0^T & 0 \end{bmatrix}$$

$$= H(p, p+1, \theta_p)\begin{bmatrix} R_b^{-T}(p-1) & 0 \\ l^T(p) & \\ h^T(p-1) & 0 \end{bmatrix},$$

where $l^T(p)$ is the last row of $R^{-T}(p)$. Let $l_b^T(p)$ be the last row of $R_b^{-T}(p)$, and suppose that $h^T(p-1)$ and $l^T(p)$ are given. Then $h^T(p)$ and $l_b^T(p)$ can be computed as

$$\begin{bmatrix} l_b^T(p) \\ h^T(p) \end{bmatrix} = \begin{bmatrix} c_p & -s_p \\ -s_p & c_p \end{bmatrix}\begin{bmatrix} l^T(p) & \\ h^T(p-1) & 0 \end{bmatrix}. \tag{14}$$

Thus the last row of $R^{-T}(p+1)$ is

$$l^T(p+1) = \begin{bmatrix} -\frac{1}{r_{11}} l_b^T(p) z(p) & l_b^T(p) \end{bmatrix}. \tag{15}$$

The following algorithm summarizes the above observations.

ALGORITHM 1. Given $x^T(0) = [t_1, \ldots, t_n]$, this algorithm recursively computes $R^{-T}(p)$, where $T(p) = Q(p)R(p)$, for each p.

$l(1) = 1/\|x(0)\|_2$
$z(1) = x^T(0)T(0)/\|x(0)\|_2$
$\gamma(0) = 1$
for $p = 1, 2, \ldots,$ until stop
$b_p = e_p^T R^{-T}(p) z(p)$
$[c_p, s_p] = \text{hyp}(\gamma(p-1), b_p)$
$\gamma(p) = -s_p b_p + c_p \gamma(p-1)$

$$\begin{bmatrix} l_b^T(p) \\ h^T(p) \end{bmatrix} := \begin{bmatrix} c_p & -s_p \\ -s_p & c_p \end{bmatrix} \begin{bmatrix} l^T(p) & \\ h^T(p-1) & 0 \end{bmatrix}$$

$l^T(p+1) = \left[-l_b^T(p) z(p)/\|x(0)\|_2 \;\; l_b^T(p)\right]$
$z_{p+1} = e_{p+1}^T T^T(p+1) x(p+1)/\|x(0)\|_2$
$z^T(p+1) = [z^T(p) \;\; z_{p+1}]$

The vector $l^T(p)$ is the last row of $R^{-T}(p)$.

6. AN ALGORITHM TO COMPUTE $u(p+1) = Q^T(p+1)x(p+1)$

Suppose $u(p) = Q^T(p)x(p)$ and $H(p)$ are known, and we want to find the vector $u(p+1) = Q^T(p+1)x(p+1)$. From Section 3 we know that $Q^T(1) = (1/r_{11})T^T(1)$. Thus from (7) we have $u(1) = Q^T(1)x(1) = z(1)$. The following lemma shows how $u(p+1)$ and $u(p)$ are related.

LEMMA 3. *Let $Q(p)$ and $Q(p+1)$ be the orthogonal matrices of the QR factorization of $T(P)$ and $T(p+1)$, respectively. If $u(p) = Q^T(p)x(p)$ and $u(p+1) = Q^T(p+1)x(p+1)$ then*

$$u(p+1) = \begin{bmatrix} u(p) \\ u_{p+1} \end{bmatrix}.$$

Proof. See Lemma 2 in [15]. ∎

We seek a way to compute u_{p+1} from the knowledge of $u(p)$ and $H(p)$. To do this, we will modify the work of Chun, Kailath, and Lev-Ari [6], which is based on the displacement structure of the data matrix, for the linear prediction problem.

Let $s \in \mathfrak{R}^k$, and define the upper triangular matrix $U(s)$ as

$$U^T(s) = [s \quad Zs \quad \cdots \quad Z^{p-1}s] \in \mathfrak{R}^{k \times p},$$

where $Z \in \mathfrak{R}^{k \times k}$ is a matrix with ones on its subdiagonal and zeros elsewhere. Let $w_1^T(p) = [r_{11} \; z^T(p)]$ and $w_2^T(p) = [0 \; z^T(p)]$. Then it is easy to see that

$$T^T(p+1)T(p+1) = U^T(w_1)U(w_1) - U^T(w_2)U(w_2),$$

where, to simplify the notation in the following discussion, we make the identification $w_i = w_i(p)$, $i = 1, 2$.

Define a matrix J by

$$J = \begin{bmatrix} I_{p+1} & \\ & \tilde{I}_{p+1} \end{bmatrix}, \qquad \text{where} \quad \tilde{I}_{p+1} = \begin{bmatrix} 1 & \\ & -I_p \end{bmatrix}.$$

Then a matrix B is said to be J-orthogonal if $B^TJB = J$. Using the matrices $U(w_1)$ and $U(w_2)$, one can obtain $R(p+1)$ as follows.

Suppose $\Theta(p)$ is a J-orthogonal matrix such that

$$\Theta(p)\begin{bmatrix} U(w_1) & \dfrac{1}{\|x(0)\|}T^T(p+1) \\ U(w_2) & \dfrac{1}{\|x(0)\|}T^T(p+1) \end{bmatrix} = \begin{bmatrix} R & Q^T \\ 0 & * \end{bmatrix}, \tag{16}$$

where R is upper triangular. Let

$$\Gamma = \begin{bmatrix} U(w_1) & \dfrac{1}{\|x(0)\|}T^T(p+1) \\ U(w_2) & \dfrac{1}{\|x(0)\|}T^T(p+1) \end{bmatrix} \quad \text{and} \quad \Phi = \begin{bmatrix} R & Q^T \\ 0 & * \end{bmatrix}.$$

Setting

$$\Gamma^T\Theta^T(p)J\Theta(p)\Gamma = \Phi^TJ\Phi,$$

it follows that $R(p+1) = R$ and $Q(p+1) = Q$.

Our goal is to find a way to construct $u(p+1) = Q^T(p+1)x(p+1)$. The above observations indicate that if such a $\Theta(p)$ exists, then $u(p+1)$ can be computed. More specifically, suppose $\Theta(p)$ is a J-orthogonal matrix which satisfies (16), and observe that

$$z(p+1) = \frac{1}{\|x(0)\|} T^T(p+1)x(p+1).$$

Then

$$\Theta(p)\begin{bmatrix} z(p+1) \\ z(p+1) \end{bmatrix} = \begin{bmatrix} Q^T(p+1)x(p+1) \\ * \end{bmatrix} = \begin{bmatrix} u(p+1) \\ * \end{bmatrix}.$$

We now describe how $\Theta(p)$ can be constructed. Recall that

$$H(p) = H(p, p+1, \theta_p) \cdots H(2, p+1, \theta_2) H(1, p+1, \theta_1)$$

is a product of hyperbolic rotations and that

$$H(p)\begin{bmatrix} R(p) \\ z^T(p) \end{bmatrix} = \begin{bmatrix} R_b(p) \\ 0^T \end{bmatrix}.$$

Let $\Theta(p) = \tilde{H}_p \cdots \tilde{H}_2 \tilde{H}_1$, where

$$\tilde{H}_k = H(p+1, 2p+2-k, \theta_k) \cdots H(k+2, p+3, \theta_k)$$
$$H(k+1, p+2, \theta_k).$$

Each $\tilde{H}_k$ is J-orthogonal; hence $\Theta(p)$ is J-orthogonal. The following theorem shows that this $\Theta(p)$ satisfies (16).

THEOREM 1.

$$\Theta(p)\begin{bmatrix} U(w_1) \\ U(w_2) \end{bmatrix} = \begin{bmatrix} R(p+1) \\ 0 \end{bmatrix}.$$

Proof. We will show that each $\tilde{H}_k$ will zero the kth superdiagonal of the lower half of

$$\tilde{H}_{k-1} \cdots \tilde{H}_1 \begin{bmatrix} U(w_1) \\ U(w_2) \end{bmatrix}$$

and produce the $(k+1)$th row of $R(p+1)$.

Recall that

$$H(p)\begin{bmatrix} R(p) \\ z^T(p) \end{bmatrix} = H(p, p-1, \theta_p) \cdots H(2, p+1, \theta_2)$$

$$H(1, p+1, \theta_1)\begin{bmatrix} R(p) \\ z^T(p) \end{bmatrix} = \begin{bmatrix} R_b(p) \\ 0^T \end{bmatrix},$$

and observe that:

[1st row of $U(w_1)$] = [1st row of $R(p+1)$].
the Toeplitz structure of $U(w_1)$ implies [2nd row of $U(w_1)$] = [0 1st row of $R(p)$].
1st row of $U(w_2)$] = [0 $z^T(p)$].

Now apply $\tilde{H}_1 = H(p+1, 2p+1, \theta_1) \cdots H(2, p+2, \theta_1)$ to

$$\begin{bmatrix} U(w_1) \\ U(w_2) \end{bmatrix}$$

and *overwrite*. Then:

[1st row of $U(w_1)$] = [1st row of $R(p+1)$].
[2nd row of $U(w_1)$] = [0 1st row of $R_b(p)$] = [2nd row of $R(p+1)$].
The Toeplitz structure of rows $2, \ldots, p+1$ of $U(w_1)$ is preserved, and the Toeplitz structure of $U(w_2)$ is preserved.
the previous two comments imply that [3rd row of $U(w_2)$] = [0 2nd row of $R(p)$].
[1st row of $U(w_2)$] = [0 $z_1^T(p)$], where $z_1(p)$ is defined by

$$H(1, p+1, \theta_1)\begin{bmatrix} R(p) \\ z^T(p) \end{bmatrix} = \begin{bmatrix} * \\ z_1^T(p) \end{bmatrix}.$$

[Thus, the first superdiagonal of $U(w_2)$ has all zeros.]

Now apply $\tilde{H}_2 = H(p+1, 2p, \theta_2) \cdots H(4, p+3, \theta_2) H(3, p+2, \theta_2)$ to

$$\tilde{H}_1\begin{bmatrix} U(w_1) \\ U(w_2) \end{bmatrix}$$

and *overwrite*. Then:

[1st row of $U(w_1)$] = [1st row of $R(p+1)$].
[2nd row of $U(w_1)$] = [2nd row of $R(p+1)$].
[3rd row of $U(w_1)$] = [0 2nd row of $R_b(p)$] = [3rd row of $R(p+1)$].
The Toeplitz structure of rows $3, \ldots, p+1$ of $U(w_1)$ is preserved, and the Toeplitz structure of $U(w_2)$ is preserved.
The previous two comments imply that [4th row of $U(w_1)$] = [0 3rd row of $R(p)$].
[1st row of $U(w_2)$] = [0 $z_2^T(p)$], where $z_2(p)$ is defined by

$$H(2, p+1, \theta_2)H(1, p+1, \theta_1)\begin{bmatrix} R(p) \\ z^T(p) \end{bmatrix} = \begin{bmatrix} * \\ z_2^T(p) \end{bmatrix}.$$

[Thus the second superdiagonal of $U(w_2)$ has all zeros.]

It is now easy to see that if we continue in this manner, then

$$\Theta(p)\begin{bmatrix} U(w_1) \\ U(w_2) \end{bmatrix} = \begin{bmatrix} R(p+1) \\ 0 \end{bmatrix}. \qquad \blacksquare$$

Next, define $g(p+1)$ by

$$\Theta(p)\begin{bmatrix} z(p+1) \\ z(p+1) \end{bmatrix} = \begin{bmatrix} u(p+1) \\ g(p+1) \end{bmatrix},$$

and suppose $u(p)$ and $g(p)$ are known. Then the following theorem shows how $u(p)$ can be computed recursively.

THEOREM 2. *Let c_k, s_k, $k = 1, \ldots, p$, be the parameters which determine $H(p)$. Then*

$$H(1,2,\theta_p) \cdots H(1, p, \theta_2)H(1, p-1, \theta_1)\begin{bmatrix} z_{p+1} \\ g(p) \end{bmatrix} = \begin{bmatrix} u_{p+1} \\ f(p) \end{bmatrix}$$

with $g^T(p+1) = [f^T(p) \;\; z_{p+1}]$.

Proof. By definition

$$\Theta(p-1) = \prod_{k=p-1}^{1} \left(\prod_{i=p-k}^{1} H(k+i, p+i, \theta_k) \right)$$

and

$$\Theta(p) = \prod_{k=p}^{1} \left(\prod_{i=p-k+1}^{1} H(k+i, p+i+1, \theta_k) \right).$$

Since multiplication on the left by $H(i, j, \theta)$ only acts on the ith and jth rows, we have

$$H(i, j, \theta) H(l, k, \theta) = H(l, k, \theta) H(i, j, \theta),$$

provided $i \neq l, k$ and $j \neq l, k$.

Using this result, observe that

$$\Theta(p) = \left[\prod_{k=p}^{1} H(p+1, 2p+2-k, \theta_k) \right] \times \left[\prod_{k=p-1}^{1} \left(\prod_{i=p-k}^{1} H(k+i, p+i+1, \theta_k) \right) \right].$$

Since

$$\Theta(p-1) \begin{bmatrix} z(p) \\ z(p) \end{bmatrix} = \prod_{k=p-1}^{1} \left(\prod_{i=p-k}^{1} H(k+i, p+i, \theta_k) \right) \begin{bmatrix} z(p) \\ z(p) \end{bmatrix} = \begin{bmatrix} u(p) \\ g(p) \end{bmatrix},$$

it follows that

$$\prod_{k=p-1}^{1} \left(\prod_{i=p-k}^{1} H(k+i, p+i+1, \theta_k) \right) \begin{bmatrix} z(p) \\ z_{p+1} \\ z(p) \\ z_{p+1} \end{bmatrix} = \begin{bmatrix} u(p) \\ z_{p+1} \\ g(p) \\ z_{p+1} \end{bmatrix}.$$

Thus

$$\prod_{k=p}^{1} H(p+1, 2p+2-k, \theta_k) \begin{bmatrix} u(p) \\ z_{p+1} \\ g(p) \\ z_{p+1} \end{bmatrix} = \begin{bmatrix} u(p+1) \\ g(p+1) \end{bmatrix}.$$

Since $H(p+1, 2p+2-k, \theta_k)$ only affects the subvector

$$\begin{bmatrix} z_{p+1} \\ g(p) \end{bmatrix},$$

the result follows. ■

Using Theorem 2, we obtain the following algorithm for computing $u(p)$ recursively.

ALGORITHM 2. Given $x(0)$, c_p, and s_p, which define $H(p)$, $p = 1, \ldots,$ this algorithm recursively computes $u(p)$ fore each p.

$z(1) = x^T(0)T(0)/\|x(0)\|_2$
$u(1) = g(1) = z(1)$
for $p = 1, 2, \ldots$ **until stop**
  $z_{p+1} = e_{p+1}^T T^T(p+1)x(p+1)/\|x(0)\|_2$
  $u_{p+1} = g_{p+1}(p+1) = z_{p+1}$
  for $k = 1, 2, \ldots, p-1$

$$\begin{bmatrix} u_{p+1} \\ g_{p+1-k}(p+1) \end{bmatrix} := \begin{bmatrix} c_k & -s_k \\ -s_k & c_k \end{bmatrix} \begin{bmatrix} u_{p+1} \\ g_{p+1-k}(p) \end{bmatrix}$$

7. A LINEAR PREDICTION ALGORITHM

Algorithms 1 and 2 form the basis for the inverse factorization linear prediction algorithm. Recall that the stopping criterion is to test when the error curve becomes flat. We use the threshold test from [15], which requires

that

$$1 - \frac{E(p+1)}{E(p)} < \delta$$

for a few consecutive steps. The value of δ and the number of consecutive steps are problem dependent. Applying the threshold test at each step does not require much work, since it can be shown [15] that $E(p+1) = E(p) - u_{p+1}^2$. Combining this and Algorithms 1 and 2, we have the following scheme.

ALGORITHM 3. Given $x^T(0) = [t_1, \ldots, t_n]$, this algorithm computes the solution $a(p_0)$ to the least squares problem $\min \|T(p)a(p) + x(p)\|_2$, with $p = p_0 = (\text{optimal } p)$.

$E(1) = \|x(0)\|^2$, $l(1) = 1/\|x(0)\|$,
$z(1) = x^T(0)T(0)/\|x(0)\|$
$u(1) = z(1)$, $g(1) = z(1)$, $\gamma(0) = 1$
$p = 1$
until $1 - E(p+1)/E(p) < \delta$ **for several consecutive steps**
  $b_p = e_p^T R^{-T}(p) z(p)$
  $[c_p, s_p] = \text{hyp}(\gamma(p-1), b_p)$
  $\gamma(p) = -s_p b_p + c_p \gamma(p-1)$

$$\begin{bmatrix} l_b^T(p) \\ h^T(p) \end{bmatrix} := \begin{bmatrix} c_p & -s_p \\ -s_p & c_p \end{bmatrix} \begin{bmatrix} l^T(p) & \\ h^T(p-1) & 0 \end{bmatrix}$$

  $l^T(p+1) = -l_b^T(p) z(p)/\|x(0)\|_2 \; l_b^T(p)$
  $z_{p+1} = e_{p+1}^T T^T(p+1) x(p+1)/\|x(0)\|_2$
  $z^T(p+1) = [z^T(p) \;\; z_{p+1}]$
  $u_{p+1} = g_{p+1}(p+1) = z_{p+1}$
  for $k = 1, 2, \ldots, p-1$

$$\begin{bmatrix} u_{p+1} \\ g_{p+1-k}(p+1) \end{bmatrix} := \begin{bmatrix} c_k & -s_k \\ -s_k & c_k \end{bmatrix} \begin{bmatrix} u_{p+1} \\ g_{p+1-k}(p) \end{bmatrix}$$

  $E(p+1) = E(p) - u_{p+1}^2$
$a(p) = -R^{-1}(p) u(p)$

This algorithm requires $np + 5p^2$ multiplications, where generally $n >> p$. Although Qiao's algorithm takes $np + \frac{5}{2}p^2$ multiplications, his method computes $a(p)$ by using a backsolve, and is thus not amenable to efficient implementation on a systolic array. The method presented in this paper, however, does not require a backsolve, since it recursively constructs $R^{-1}(p)$ instead of $R(p)$.

We now discuss some numerical tests which compare our method with Qiao's [15]. These tests were performed using FORTRAN with double precision

TABLE 1
PREDICTION COEFFICIENTS FROM I.F. AND QIAO METHODS

$n = 100$		$n = 400$	
I.F. method	Qiao method	I.F. method	Qiao method
−2.233225368847	−2.233225368844	−2.259211248247	−2.259211247215
1.564271147812	1.564271147802	1.596641992375	1.596641989163
−0.419212611098	−0.419212611090	−0.427765467145	−0.427765463068
0.112579296451	0.112579296476	0.114419870511	0.114419870103
−0.031104574728	−0.031104574778	−0.029914020105	−0.029914023844
0.007622415344	0.007622415368	0.005887873539	0.005887875789

arithmetic. In each of the experiments, the values used in the threshold test were $\delta = 0.01$, and the number of consecutive time steps was 3. These are the same parameters used by Qiao [14].

The first test consisted of constructing signals

$$x = \left[1 \quad 2 \quad \cdots \quad \frac{n}{2} \quad \frac{n}{2} \quad \cdots \quad 2 \quad 1\right]$$

for $n = 100$ and $n = 400$. These signals are used by Qiao in [14] in order to obtain a matrix $T(p)$ which is ill conditioned. In each case we obtained the same value for the optimal p (which was $p_0 = 6$ for both $n = 100$ and $n = 400$), and the prediction coefficients agreed up to at least 7 decimal places. The results are shown in Table 1, in which we have denoted our inverse factorization algorithm by "I.F. method."

We also compared the two methods using several random signals x whose lengths varied from 50 to 400. In each case both methods obtained the same optimal p values, and the prediction coefficients agreed up to at least 12 decimal places.

8. SYSTOLIC ARRAY IMPLEMENTATION

To obtain a systolic array for the LP problem we will simplify the notation of Algorithm 3. Since the optimal $p_0 = p$ is not known before the computation begins, we will assume that m is an upper bound for p_0. Recall that generally $n >> p_0$, so one could choose $m = n$. Furthermore, in some applications the value of p_0 is fixed before the computation begins [12].

Let l_{ij} denote the i, j entry of $R^{-T}(p_0)$ and notice that $l_b^T = l_{p+1,2:p+1}$. We use the notation $l:k$, for $l \leqslant k$, to indicate the sequence $l, \ldots, k$. Overwriting h and g, we can reformulate Algorithm 3 as follows.

ALGORITHM 3′.

Initialize: $l_{11} = 1/\|x(0)\|$, $z := z(m) = l_{11}T^T(m)x(m)$, $d = -l_{11}z$, $a_1 = l_{11}z_1$, $g = u = z$, $E(1) = \|x(0)\|^2$, $h = 0$, $\gamma = 1$, $p = 1$

while $1 - E(p+1)/E(p) < \delta$ for several consecutive steps

$b_p = l_{p,1:p}z_{1:p}$

$[c_p, s_p] = \text{hyp}(\gamma, b_p)$

$\gamma = -s_p b_p + c_p \gamma$

$$\begin{bmatrix} l_{p+1,2:p+1} \\ h_{1:p} \end{bmatrix} := \begin{bmatrix} c_p & -s_p \\ -s_p & c_p \end{bmatrix} \begin{bmatrix} l_{p,1:p} \\ h_{1:p} \end{bmatrix}$$

$l_{p+1,1} = l_{p+1,2:p+1}d_{1:p}$

$$\begin{bmatrix} u_{p+1:m} \\ g_{1:m-p} \end{bmatrix} := \begin{bmatrix} c_k & -s_k \\ -s_k & c_k \end{bmatrix} \begin{bmatrix} u_{p+1:m} \\ g_{1:m-p} \end{bmatrix}$$

$E(p+1) = E(p) - u_{p+1}^2$

$a_{p+1} = 0$

$a_{1:p+1} := a_{1:p+1} + u_{p+1}l_{p+1,1:p+1}^T$

$p = p + 1$

Notice that in Algorithm 3′, $z(m)$ is computed when all that is needed is $z(p)$. In the parallel implementation, as will be seen, all of $z(m)$ is not computed. We write z in this way because $T^T(m)x(m)$ is a convolution operation, and many efficient systolic arrays exist for convolution [10].

We will describe the systolic array implementation of the LP problem in six steps. The first step is the initialization stage, and the remaining steps compute various parts of the main loop. We then combine the arrays of step 2 through step 6 to obtain a systolic array for the main loop.

In the sequel we assume that time delays are present where appropriate. That is, each cell of the array is active only when all required data are present. In addition, if an input to a cell is initially zero, we represent this by omitting that input line. For example, Figure 1(a) shows a typical inner product cell which has input values a, b, and c. If c is to be initially zero, the input line for c will not be present, as illustrated in Figure 1(b).

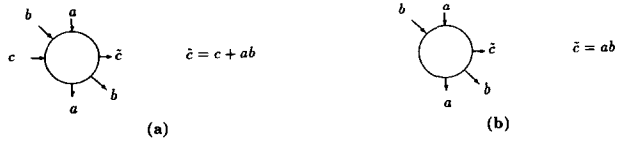


FIG. 1. Example of inner product cells.

Step 1: Computation of l_{11}, z, d, and E_1

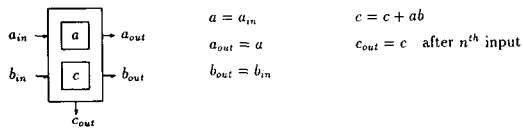
This is the initialization step. A straightforward computation of l_{11} and z would require $m + 1$ inner products. But the circulant structure of $T(m)$ implies that $T^T(m)x(m)$ is a convolution, which can be efficiently implemented with a systolic array. The cells needed for this step are shown in Figure 2.

Some remarks on the cells are needed. The convolution cell in Figure 2(a) is similar to that used for design $R2$ by Kung [10]. Notice that the cell holds a for one clock cycle, while b passes directly through. Furthermore the cell holds c for n clock cycles. That is, only after the nth clock cycle is c passed out of the cell. The inner product cell used in this step is very general, and will be used again in subsequent steps.

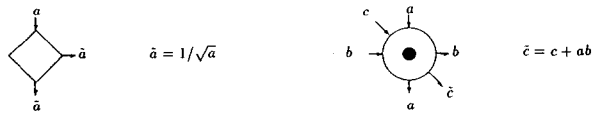
Using these basic cells, a systolic array for the initialization stage can be designed as in Figure 3 (where $m = 5$). Notice that the first row of the input sequence $\{t_i\}$ takes twice as long to move through the convolution cells as the second row of the $\{t_i\}$. It is easy to see that after $n + 3$ time steps l_{11}, z_1, d_1, and E_1 are available, and hence the main loop can begin.

Step 2: Computation of b_p

In this step we need to compute inner products of length p, $p = 1, \ldots, p_0 - 1$. Thus the basic cell needed for this step is the inner product cell shown in Figure 1. The systolic array for this step is shown in Figure 4 for $m = 4$.



(a)



(b) (c)

FIG. 2. Cells for step 1.

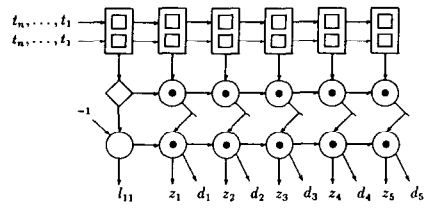


FIG. 3. Array for step 1.

Here we are assuming that the l_{ij} are available. Also notice that the flow of the l_{ij} in this array is consistent with the manner in which they are computed. More specifically, from Algorithm 3′ we see that only l_{pk} is needed to obtain $l_{p+1,k+1}$, $k = 1, \ldots, p$. The boxes in Figure 4 represent this observation.

Step 3: Computation of c_p, s_p, and γ

Given b_p and γ, we need a cell which computes the parameters c_p and s_p for the hyperbolic rotations. This cell is shown in Figure 5. It is similar to the Givens rotation cell used by Alexander, Ghirnikar, and Plemmons [1] for a parallel implementation of inverse QR updating.

Assuming the b_p are available, Figure 5 also shows an array for computing c_p and s_p for $m = 3$. Each cell propagates a copy of c_p and s_p both right and left. This is so that the $(p + 1)$st row of R^{-T} and u_{p+1} can be computed concurrently.

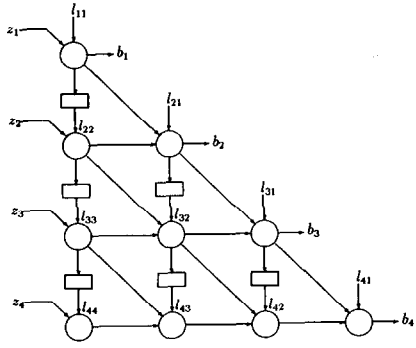


FIG. 4. Array for step 2.

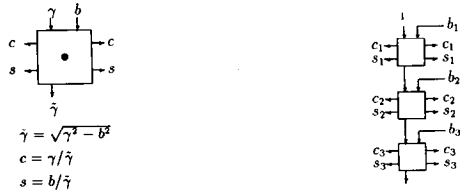


FIG. 5. Cell and array for step 3.

Step 4: Computation of l_{ij}

In this step we need two types of cells. One cell applies the hyperbolic rotation and is shown in Figure 6. The other cell is the inner product cell used in step 2. Figure 7 shows the array used for computing l_{ij} for $m = 5$.

Step 5: Computation of u_{p+1} *and Check for Convergence*

For this step we need a cell which applies the hyperbolic rotations as in step 4, except the direction of flow is reversed. We also need cells to update the error and check for convergence. These cells are shown in Figure 8.

The variable δ is the tolerance in Algorithm 3′, and # is the number of steps required for the convergence test. The value of I indicates whether the optimal p is attained. If $I = 0$ then we continue the algorithm, but if $I = 1$ the algorithm terminates.

The array for this step is illustrated in Figure 9. Note that when the optimal p is reached, the algorithm must output the parameters $a_1, \ldots, a_{p_0}$. We describe how this can be done in the next step.

Step 6: Computation of a_p

The array for this step is similar to that of step 2. The only difference is a slight modification of the inner product cell, which is shown in Figure 10. This cell is the same as that shown in Figure 2(c), except that the output of the accumulated inner product can be directed out of one of two lines. The small circles in Figure 10 represent this. The darkened circle indicates that the line is closed. That is, no values are exited through this line. If the small circle is open, then the line is open. Figure 10 shows that if $I = 0$, then the inner

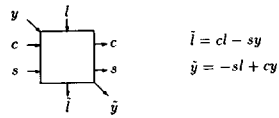


FIG. 6. Cell for step 4.

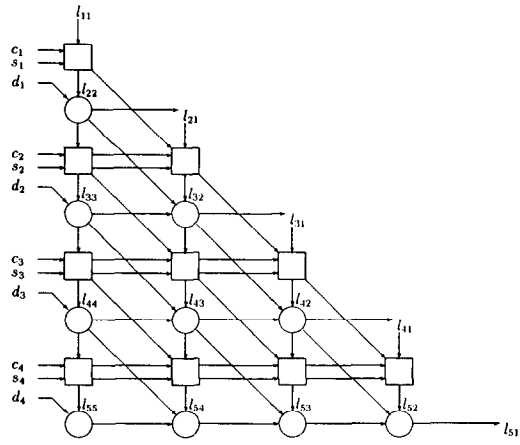


FIG. 7. Array for step 4.

product flows out of the southeast part of the cell, and if $I = 1$, out of the northeast line. The array for this step is shown in Figure 11 for $p_0 = 4$.

Combining the arrays in steps 2 through 6, we obtain the systolic array for the main loop of the LP algorithm. This is shown in Figure 12 with $p_0 = 3$.

Recall that Algorithm 3 requires $np_0 + 5p_0^2$ multiplications. The numbers of time steps for our systolic array are given by: (1) $n + 3p_0$ for the initialization, and (2) $\frac{1}{2}p_0^2 + \frac{7}{2}p_0$ for the main loop. Thus the set of coefficients $a_1, \ldots, a_{p_0}$ are obtained in a total of only $n + \frac{1}{2}p_0^2 + \frac{13}{2}p_0$ time steps.

9. CONCLUDING REMARKS

In this paper we have presented an inverse factorization method for solving linear prediction problems. The scheme totally avoids the need for solving triangular systems, and is thus amenable to parallel implementation.

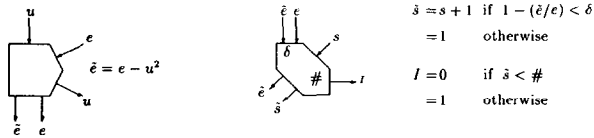


FIG. 8. Cells for step 5.

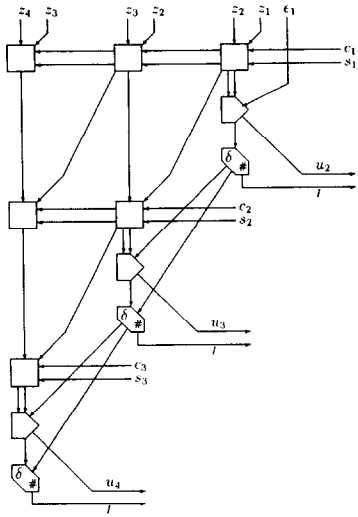


FIG. 9. Array for step 5.

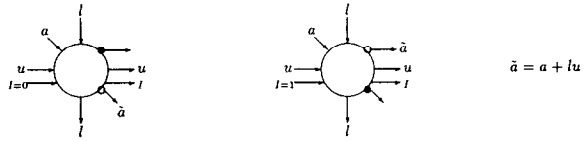


FIG. 10. Cell for step 6.

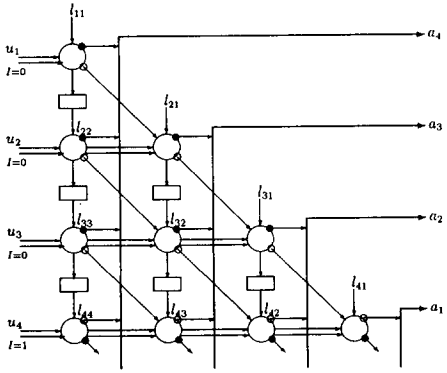


FIG. 11. Array for step 6.

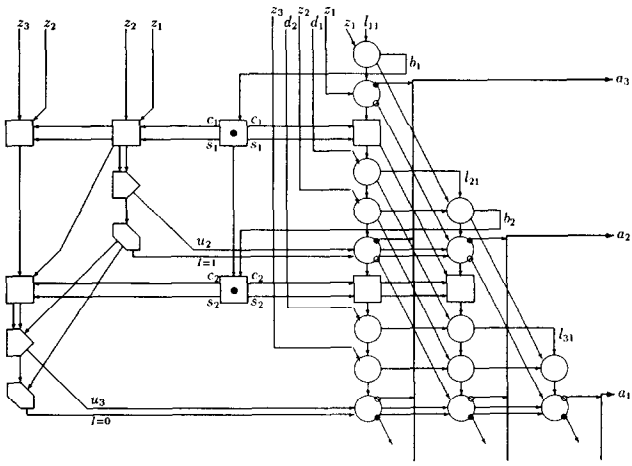


FIG. 12. Final array.

Other applications of this principle are described in [16]. We have shown that our algorithm has low numerical complexity, and can be implemented on a linear systolic array in $n + \frac{1}{2}p_0^2 + \frac{13}{2}p_0$ time steps, where n is the number of data samples and p_0 the optimal order of the predictor.

REFERENCES

1 S. T. Alexander, A. L. Ghirnikar, and R. J. Plemmons, A parallel implementation of the inverse *QR* adaptive filter, Computers in Electrical Engineering, to appear (1992).
2 S. T. Alexander, C.-T. Pan, and R. J. Plemmons, Analysis of a recursive least squares hyperbolic rotation algorithm for signal processing, *Linear Algebra Appl.* 98:3–40 (1988).
3 G. Ammar and W. Gragg, Superfast solution of real positive definite Toeplitz systems, *SIAM J. Matrix Anal. Appl.* 9:61–76 (1988).
4 A. W. Bojanczyk, Systolic implementation of the lattice algorithm for least squares linear prediction problems, *Linear Algebra Appl.* 77:27–42 (1986).
5 A. W. Bojanczyk, R. P. Brent, and F. R. de Hoog, *QR* factorization of Toeplitz matrices, *Numer. Math.* 49:81–94 (1986).
6 J. Chun, T. Kailath, and H. Lev-Ari, Fast parallel algorithms for QR and triangular factorization, *SIAM J. Sci. Statist. Comput.* 8:899–913 (1987).
7 G. Cybenko, A general orthogonalization technique with applications to time series analysis and signal processing, *Math. Comp.* 40:323–336 (1983).

8 P. M. Djurić and S. M. Kay, Model order estimation of 2D autoregressive processes, in *Proceedings of ICASSP*, July 1991.
9 G. H. Golub and C. Van-Loan, *Matrix Computations*, 2nd ed., Johns Hopkins U.P., Baltimore, 1989.
10 H. T. Kung, Why systolic architectures?, *IEEE Trans. Comput.* 15:37–46 (1982).
11 J. Makhoul, Linear prediction: A tutorial review, *Proc. IEEE* 63:561–580 (1975).
12 D. G. Manolakis and J. G. Proakis, *Introduction to Digital Signal Processing*, Macmillan, 1988.
13 C. T. Pan and R. J. Plemmons, Least squares modifications with inverse factorization: Parallel implications, *J. Comput. Appl. Math.* 27:109–127 (1989).
14 S. Qiao, Parallel Implementation of a Linear Prediction Algorithm, Technical Report, Mathematical Sciences Inst., Cornell Univ., Mar. 1987.
15 S. Qiao, Recursive least squares algorithm for linear prediction problems, *SIAM J. Matrix Anal. Appl.* 9:323–328 (1988).
16 M. Moonen and J. Vandewalle, A systolic array for recursive least squares computations, Proc. IEE Conf. on Acoustics, Speech and Sig. Proc., Toronto, Canada, May 1991, pp. 1013–1016.

Received 2 June 1991; final manuscript accepted 31 January 1992