JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 9, 261–270 (1990)

Linearly Connected Arrays for Toeplitz Least-Squares Problems

A. W. BOJANCZYK

School of Electrical Engineering, Cornell University, Ithaca, New York 14853

R. P. BRENT

Laboratory for Computer Science, Australian National University, GPO Box 4, Canberra, ACT 2601, Australia

AND

F. R. DE HOOG

Division of Mathematics and Statistics, CSIRO, GPO Box 1965, Canberra, ACT 2601, Australia

We present a linearly connected array of $O(n)$ cells that solves the linear least-squares problem for an $(m+1) \times (n+1)$ Toeplitz matrix in time $O(m+n)$. The total storage required is $O(n)$ words, i.e., only a constant per cell. The parallel algorithm described in this paper is based on the sequential QR factorization algorithm for Toeplitz matrices recently developed by the authors.

1. INTRODUCTION

A rectangular Toeplitz matrix is an $(m+1) \times (n+1)$ matrix of the form

$$T = \begin{bmatrix} t_0 & t_{-1} & t_{-2} & \cdot & t_{-n} \\ t_1 & t_0 & t_{-1} & \cdot & t_{-n+1} \\ t_2 & t_1 & t_0 & \cdot & t_{-n+2} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ t_m & t_{m-1} & t_{m-2} & \cdot & t_{m-n} \end{bmatrix}. \qquad (1.1)$$

We are concerned with the least-squares solution of, possibly, an overdetermined system of equations defined by the full rank matrix T of the form (1.1) and the $(m+1)$-dimensional vector b, i.e., the minimization of $\|Tc - b\|_2$ over $c \in R^{n+1}$, where $b \in R^{m+1}$.

The least-squares problem

$$\min \|Tc - b\|_2 \qquad (1.2)$$

can be solved by solving the square system of normal equations

$$T^T Tc = T^T b \qquad (1.3)$$

or by converting the original least-squares problem into an equivalent problem

$$\min \|Q^T Tc - Q^T b\|_2 \qquad (1.4)$$

by premultiplying both T and b by an orthogonal matrix Q such that $Q^T T = R$ and R is upper triangular.

Note that the covariance matrix $T^T T$ is no longer Toeplitz but has low displacement rank and hence can be solved in time $O(n)$ using fast algorithms developed in [4].

In this paper we show that it is possible to solve the least-squares problem (1.2) on a linear array of $O(n)$ processors in time $O(m+n)$ and storage $O(n)$ without computing the covariance matrix. The algorithm we use is based on the QR factorization of the Toeplitz matrix T described in [1] and solves the equivalent problem (1.4).

Our approach to the QR decomposition of Toeplitz matrices offers an improvement over the pioneering work by Sweet [9]. Sweet has proposed an algorithm which computes the QR decomposition of a rectangular Toeplitz matrix in $O(nm)$ multiplications. Sweet's algorithm generates R column by column but, to the authors' knowledge, calculations for computing column k cannot start before calculations for column $k-1$ are finished. Moreover, elements within a column have to be generated sequentially. Thus it seems that Sweet's algorithm, in its present form, is not suitable for parallel computation while our algorithm can be implemented efficiently in parallel.

2. DESCRIPTION OF THE ALGORITHM

We start with a short description of the algorithm for the QR decomposition of Toeplitz matrices. The reader is referred to [1] for a detailed discussion and derivation of this algorithm.

Assume that

$$Q^{\mathrm{T}}T = R, \tag{2.1}$$

where Q is an $(m+1)\times(n+1)$ matrix with orthonormal columns,

$$Q^{\mathrm{T}} = [q_1, q_2, \ldots, q_{n+1}]^{\mathrm{T}} = \begin{bmatrix} q_{1,1} & q_{2,1} & \cdots & q_{m+1,1} \\ q_{1,2} & q_{2,2} & \cdots & q_{m+1,2} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ q_{1,n+1} & q_{2,n+1} & \cdots & q_{m+1,n+1} \end{bmatrix}, \tag{2.2a}$$

and R is an $(n+1)\times(n+1)$ upper triangular matrix,

$$R = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & \cdots & r_{1,n+1} \\ \cdot & r_{2,2} & r_{2,3} & \cdots & r_{2,n+1} \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & r_{n+1,n+1} \end{bmatrix}. \tag{2.2b}$$

The matrices Q^{T} and R are generated row by row starting from the top rows. Each row of R (and Q^{T}) is calculated from the previous row after three implicit modifications of rank 1 to the matrix R, one update and two downdates.

To describe the algorithm we need some auxiliary vectors and matrices. Specifically, let

$$x = [x_1, x_2, \ldots, x_n]^{\mathrm{T}} := [t_m, t_{m-1}, \ldots, t_{m-n+1}]^{\mathrm{T}}$$

be a vector consisting of the first n coefficients of the last row of T,

$$y = [y_1, y_2, \ldots, y_n]^{\mathrm{T}} := [t_{-1}, t_{-2}, \ldots, t_{-n}]^{\mathrm{T}}$$

be a vector consisting of the last n coefficients of the first row of T, and

$$f = [f_1, f_2, \ldots, f_n]^{\mathrm{T}} := [r_{1,2}, r_{1,3}, \ldots, r_{1,n+1}]^{\mathrm{T}}$$

be a vector consisting of the last n coefficients of the first row of R.

Furthermore, let

$$R_t = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & \cdots & r_{1,n} \\ \cdot & r_{2,2} & r_{2,3} & \cdots & r_{2,n} \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & r_{n,n} \end{bmatrix}$$

and

$$R_b = \begin{bmatrix} r_{2,2} & r_{2,3} & r_{2,4} & \cdots & r_{2,n+1} \\ \cdot & r_{3,3} & r_{3,4} & \cdots & r_{3,n+1} \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & r_{n+1,n+1} \end{bmatrix}$$

be respectively the principal $n \times n$ top and bottom submatrices of R.

Define three sequences of plane rotations $G(\alpha_1)$, $G(\alpha_2)$, $\ldots$, $G(\alpha_n)$, $G(\beta_1)$, $G(\beta_2)$, $\ldots$, $G(\beta_n)$, and $G(\gamma_1)$, $G(\gamma_2)$, $\ldots$, $G(\gamma_n)$ by the relations

$$G(\alpha_n)G(\alpha_{n-1})\cdots G(\alpha_1)[y, R_t^{\mathrm{T}}]^{\mathrm{T}} = [W^{\mathrm{T}}, 0]^{\mathrm{T}} \tag{2.3a}$$

$$G(\beta_n)G(\beta_{n-1})\cdots G(\beta_1)[x, Z^{\mathrm{T}}]^{\mathrm{T}} = [W^{\mathrm{T}}, 0]^{\mathrm{T}} \tag{2.3b}$$

$$G(\gamma_n)G(\gamma_{n-1})\cdots G(\gamma_1)[f, R_b^{\mathrm{T}}]^{\mathrm{T}} = [Z^{\mathrm{T}}, 0]^{\mathrm{T}}. \tag{2.3c}$$

The transformations $G(\alpha_k)$, $G(\beta_k)$, and $G(\gamma_k)$ are rotations in the plane $(k, k+1)$ and are defined by the cosines and sines of the angles α_k, β_k, and γ_k such that the relations (2.3a), (2.3b), and (2.3c) hold. It can be shown that the relations (2.3a)–(2.3c) uniquely define W, Z, $G(\alpha_1)$, $G(\alpha_2)$, $\ldots$, $G(\alpha_n)$, $G(\beta_1)$, $G(\beta_2)$, $\ldots$, $G(\beta_n)$, and $G(\gamma_1)$, $G(\gamma_2)$, $\ldots$, $G(\gamma_n)$; for details see [1].

The relation (2.3a) is an updating operation following addition of the row y^{T} to R_t while (2.3b) and (2.3c) are downdatings of W and Z following subtraction of rows x^{T} and f^{T}, respectively.

Remark. Premultiplication by $G(\alpha_k)$, $G(\beta_k)$, or $G(\gamma_k)$ alters only rows k and $k+1$. Thus, when applying $G(\alpha_k)$, $G(\beta_k)$, or $G(\gamma_k)$ to a matrix we shall consider only rows k and $k+1$, remembering that the other rows remain unchanged. Also, we shall assume that the dimensions of $G(\alpha_k)$, $G(\beta_k)$, and $G(\gamma_k)$ are such that premultiplication by $G(\alpha_k)$, $G(\beta_k)$, and $G(\gamma_k)$ is feasible.

Relations (2.3a)–(2.3c) are the basis of the recursive procedure for computing rows of the matrix R. In the kth step of the recursion we determine, from (2.3a), the transformation $G(\alpha_k)$ and the kth row of W; from (2.3b), the transformation $G(\beta_k)$ and the kth row of Z; and from (2.3c), the transformation $G(\gamma_k)$ and the kth row of R_b. In outline the kth step of the recursion is the following.

Assume that, prior to execution of the kth step, rows k and $k+1$ of $G(\alpha_{k-1})\cdots G(\alpha_1)[y, R_t^{\mathrm{T}}]^{\mathrm{T}}$ are known. (Note that the $(k+1)$st row of $G(\alpha_{k-1})\cdots G(\alpha_1)[y, R_t^{\mathrm{T}}]^{\mathrm{T}}$ is the kth row of R_t.) The transformation $G(\alpha_k)$ is determined from the kth elements of rows k and $k+1$ in such a way that premultiplication by $G(\alpha_k)$ annihilates the kth element in row $k+1$. As transformations $G(\alpha_{k+1}), \ldots, G(\alpha_n)$ will not alter the kth row of $G(\alpha_k)\cdots G(\alpha_1)[y, R_t^{\mathrm{T}}]^{\mathrm{T}}$, premultiplication by $G(\alpha_k)$ gives also the kth row of W (and the $(k+1)$st row of $G(\alpha_k)\cdots G(\alpha_1)[y, R_t^{\mathrm{T}}]^{\mathrm{T}}$). Now, in the relation (2.3b), we know the kth row of $G(\beta_{k-1})\cdots G(\beta_1)[x, Z^{\mathrm{T}}]^{\mathrm{T}}$ (from step $k-1$) and the kth row of W. This allows us to determine the transformation $G(\beta_k)$ and also the kth row of Z. Similarly, in the relation (2.3c) we know the kth row of $G(\gamma_{k-1})\cdots G(\gamma_1)[f, R_b^{\mathrm{T}}]^{\mathrm{T}}$ (from step $k-1$) and the kth row of Z. Thus we can determine the transformation $G(\gamma_k)$ and the kth row of R_b. But the kth row of R_b (except for its last component) is identical to the $(k+1)$st row of R_t. So we can repeat the whole procedure starting now from rows $k+1$ and $k+2$ of $G(\alpha_k)\cdots G(\alpha_1)[y, R_t^{\mathrm{T}}]^{\mathrm{T}}$. As the first row of $[y, R_t^{\mathrm{T}}]^{\mathrm{T}}$ is known, and the first row of R is given by $\tau_1^{\mathrm{T}}T/|\tau_1|$, where τ_1 is the first column of T, the description of the procedure for computing R is complete.

In order to compute the matrix Q let us define an $(m+1)\times(n+1)$ matrix P with orthogonal columns by the relation

$$G(\gamma_n)G(\gamma_{n-1})\cdots G(\gamma_1)Q^{\mathrm{T}} = P^{\mathrm{T}}. \tag{2.4a}$$

Then it can be shown (detailed description is given in [1]) that the following relation holds:

$$[I_{n+1}, 0]G(\alpha_n)G(\alpha_{n-1})\cdots G(\alpha_1)\begin{bmatrix} 1 & 0^{\mathrm{T}} \\ 0 & Q^{\mathrm{T}} \end{bmatrix} = [I_{n+1}, 0]G(\beta_n)G(\beta_{n-1})\cdots G(\beta_1)\begin{bmatrix} 0^{\mathrm{T}} & 1 \\ P^{\mathrm{T}} & 0 \end{bmatrix}. \tag{2.4b}$$

Here, the transformations $G(\alpha_n)$, $G(\alpha_{n-1})$, $\ldots$, $G(\alpha_1)$, $G(\beta_n)$, $G(\beta_{n-1})$, $\ldots$, $G(\beta_1)$, and $G(\gamma_n)$, $G(\gamma_{n-1})$, $\ldots$, $G(\gamma_1)$ are those defined by the relations (2.3a)–(2.3c).

The relation (2.4b) gives us a means for computing the kth row of P^{T} from rows $k-1$ and k of Q^{T}. On the other hand, the relation (2.4a) gives us means for computing the $(k+1)$st row of Q^{T} from its kth row and the kth row of P^{T}. Thus, together, the relations (2.4a) and (2.4b) form a recursion for computing Q^{T} row by row, starting from its first row, which is equal to $\tau_1/|\tau_1|$.

Having computed the QR factorization of T, we obtain the least-squares solution x to the problem (1.2) in two additional steps. First, we premultiply the right-hand-side vector b by Q^{T} to get an $(n+1)$ vector a,

$$a = Q^{\mathrm{T}}b. \tag{2.5a}$$

Next, we solve an $(n+1)\times(n+1)$ upper triangular system

$$Rc = a. \tag{2.5b}$$

The complete, sequential, algorithm SLLS for solving the least-squares problem (1.2) is given in Appendix A.

On inspection of Algorithm SLLS, it is easy to see that, except for determination of cosines and sines of the appropriate angles, all operations performed are vector operations. Moreover, every vector operation can be executed componentwise in the direction of increasing indices. This means that vector operations can be pipelined in the sense that the loop for step $k+1$ can start immediately after only a few components of each vector have been computed in step k. Thus, by multiprocessing and pipelining, we can achieve the execution time of order $O(m+n)$ with $O(n)$ processors, instead of $O(mn)$ for the sequential calculations. This is indeed the case, as will be shown later on.

Because R is not Toeplitz it would seem that $(n+1)(n+2)/2$ coefficients of R have to be stored. However, in order to determine successive components of the solution vector, during the backsubstitution process, only one row of the matrix R is needed at a time. But row k of R can be recovered from its last element and row $k+1$ by applying inverse rotations, i.e., by running Algorithm SLLS backward. Thus, by regenerating rows we are able to reduce storage to $O(n)$ at the expense of some extra computation.

The modified ($O(n)$ storage) backsubstitution phase is given in Appendix B.

3. PARALLEL IMPLEMENTATION

In this section we describe a one-dimensional array of processors for computing the solution of the least-squares problem (1.2). We consider the case when the problem "fits" the array, i.e., the dimension n of the problem is equal to the number of processing cells.

We assume that we have at our disposal a synchronous system of linearly connected microprogrammable cells. This allows us to change the set of operations performed by each cell when necessary. Our approach is similar to the one adopted in the PSC project [6].

The complexity of the parallel algorithm is measured in units of time. A *time unit* is the maximal time that is necessary for a processor to perform the most time consuming set of operations (specified below) together with transferring data to and from neighboring processors. A synchronization mechanism allows processors to exchange data at times separated by integer multiples of a time unit.

The parallel implementation consists of the following steps.

Step 1. Given the matrix $T = [t_1, \ldots, t_{n+1}]$ compute the vector

$$r = t_1^{\mathrm{T}}T/\|t_1\|.$$

Step 2. Given the vectors y, x, r, f, compute the transformations $G(\alpha_1), \ldots, G(\alpha_n)$, $G(\beta_1), \ldots, G(\beta_n)$, $G(\gamma_1)$,

$\ldots, G(\gamma_n)$ and the last column of the matrix R, i.e., the vector

$$\rho = \mathrm{Re}_{n+1}.$$

Step 3. Given the vectors t_1, v, u, b, the scalar r_{11}, and the transformations $G(\alpha_1), \ldots, G(\alpha_n), G(\beta_1), \ldots, G(\beta_n), G(\gamma_1), \ldots, G(\gamma_n)$, compute the modified right-hand-side vector a,

$$a = Q^{\mathrm{T}}b,$$

and the first column of Q, i.e., the vector

$$q = t_1/\|t_1\|.$$

Step 4. Given vectors ρ, a and the transformations $G(\alpha_1), \ldots, G(\alpha_n), G(\beta_1), \ldots, G(\beta_n), G(\gamma_1), \ldots, G(\gamma_n)$, compute the solution vector x.

To simplify the description, different arrays will be used for the realization of Steps 1 to 4. In practice the arrays could be combined. It will be shown that each step can be executed on a linear array of at most $n + 1$ processors in time $O(m + n)$ and the storage $O(n)$, yielding a total time of $O(m + n)$ and total storage $O(n)$.

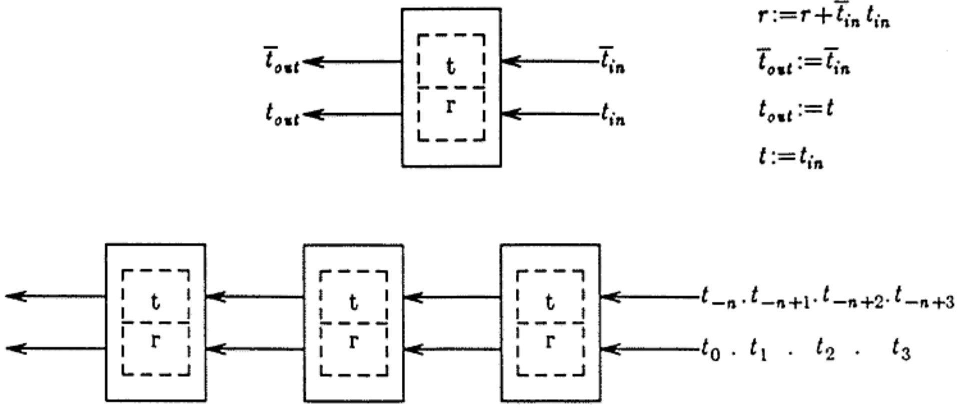
Step 1. There are many possibilities for a realization of the first step as it can be formulated as a convolution computation. A broad discussion of design for the convolution problem is given by Kung [7]; we have chosen design R2 from that paper. The systolic array and its cell definition are depicted in Fig. 3.1. This design is one in which the partial results stay at a cell to accumulate their terms while inputs move in the same direction but at different speeds.

The array consists of $n + 1$ cells arranged from left to right. Every cell has two registers, t and r. Register r of the cell $i - 1$ accumulates the partial result of the inner product $r_i = \bar{t}_1^{\mathrm{T}} t_i$. The data vectors $\bar{t} = [t_{-n}, \ldots, t_0, \cdots t_m]$ and $t = [t_0, \ldots, t_m]$ enter the array through the rightmost cell and move from right to left, but the latter moves at half the speed of the $\bar{t}$'s; i.e., each t stays inside every cell it passes for one extra cycle. This produces a delay corresponding to the "shift down" operation and guarantees that terms in cell i are accumulated correctly. Cell i is initialized when it receives the first component of the t datastream, i.e., at time $2(i - 1)$. It is easy to see that after $m + 2n$ cycles every inner product r_i, $i = 1, \ldots, n + 1$, is known and stored in register r of cell i.

We still have to divide each r_i by $(r_1)^{1/2}$. One way of achieving this is by shifting r_1 from the leftmost cell to the right through all cells and performing the division by $(r_1)^{1/2}$ while r_1 moves to the right. As r_1 has to pass through all cells, this operation requires an additional $n + 1$ units of time. Thus the overall time for computing the vector r is $m + 3n + 1$ units.

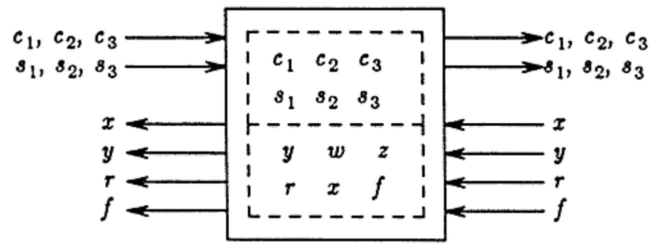
Step 2. *Remark.* Steps 2 and 3 implement the triangularization phase. For the sake of clarity we describe the process of generating the last column of R and the process of modifying the new right-hand-side vector a separately, although it is possible to combine these two processes.

The array for computing the last column of R consists of $n + 1$ processors $P_0, P_1, \ldots, P_n$, arranged from left to right. All processors are identical except for the leftmost processor P_0. The processor P_0 generates all transformations $G(\alpha)$, $G(\beta)$, $G(\gamma)$, while remaining processors propagate them. Each processor has 12 registers, 6 for storing parameters of the three transformations $G(\alpha)$, $G(\beta)$, $G(\gamma)$ (this can be decreased to 3) and 6 for storing components of the intermediate vectors y, x, w, r, f, z. Data flow in both directions between adjacent processors, but parameters of transformations flow from left to right and the intermediate results flow from right to left. The cells are illustrated in Fig. 3.2.



$\bar{t}_{out}$
t_{out}
t
r
$\bar{t}_{in}$
t_{in}
$r := r + \bar{t}_{in}\, t_{in}$
$\bar{t}_{out} := \bar{t}_{in}$
$t_{out} := t$
$t := t_{in}$
$t_{-n} \cdot t_{-n+1} \cdot t_{-n+2} \cdot t_{-n+3}$
$t_0 \cdot t_1 \cdot t_2 \cdot t_3$

FIGURE 3.1



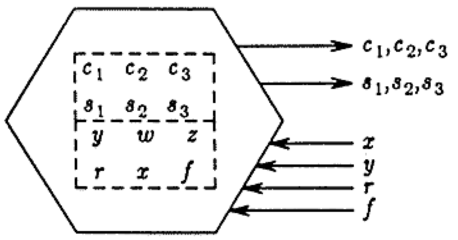


FIGURE 3.2

Before iterating, we feed data vectors

$$x = [t_m, t_{m-1}, \ldots, t_{m-n+1}]^{\mathrm{T}}$$

$$y = [t_{-1}, t_{-2}, \ldots, t_{-n}]^{\mathrm{T}}$$

$$r = [r_{11}, r_{12}, \ldots, r_{1n}]^{\mathrm{T}}$$

$$f = [r_{12}, r_{13}, \ldots, r_{1n+1}]^{\mathrm{T}}$$

into the array so that the kth components are stored in the corresponding registers of the $(k-1)$st cell. Cell k is initialized at time k and is active on alternate cycles only. This delay is caused by the fact that data can travel only with unit speed and cannot be broadcast to all cells at the same unit of time as we do not have global communication.

When active, the processor P_0 executes the short program shown in Fig. 3.3. Similarly, when active, each processor other than P_0 executes the program shown in Fig. 3.4. Having completed the execution of their programs, the processors finish a cycle by transferring data to and from neighboring processors. The vectors x, y, r, f move from cell to cell from left to right. The parameters of transformations c_1, s_1, c_2, s_2, c_3, s_3, which are generated in the leftmost cell, move in the opposite direction meeting consecutive components of the transformed vectors x, y, r, f in the appropriate order for the following cycle.

When the parameters of the kth transformations reach cell $n-k$, the cell executes its program for the last time and then is disabled. Then the registers c_1, s_1, c_2, s_2, c_3, s_3 contain the parameters of the transformations $G(\alpha_k)$, $G(\beta_k)$, $G(\gamma_k)$ and the register r contains the last element of the $(k+1)$st row of R (or equivalently the $(k+1)$st element of the last column of R).

It is easy to see that after n iterations all processors are disabled. As each processor is active only every second cycle, Step 2 requires $2n-1$ units of time.

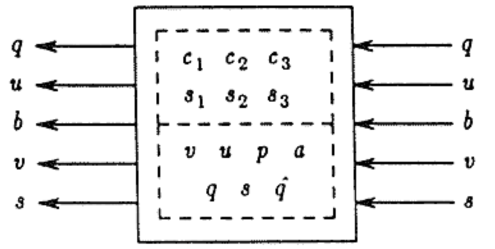
Step 3. The array for modifying the right-hand-side vector also consists of $n+1$ processors $P_0, P_1, \ldots, P_n$. All processors are identical except for the rightmost processor P_n. The processor P_n computes the first column of the matrix Q while the processor P_{n-i} computes the $(i+1)$st column of Q. In addition all processors form the inner products of the vector b with these columns of the matrix Q

```
begin
  $w := (y^2+r^2)^{1/2}$;
  $c_1 := y/w$;
  $s_1 := r/w$;
  $z := (w^2-x^2)^{1/2}$;
  $c_2 := x/w$;
  $s_2 := z/w$;
  $r := (z^2-f^2)^{1/2}$;
  $c_3 := f/z$;
  $s_3 := r/z$;
end
```

FIG. 3.3. Program executed by P_0.

```
begin
  $w := c_1y+s_1r$;
  $y := -s_1+c_1r$;
  $z := (w-xc_2)/s_2$;
  $x := -s_2x+c_2z$;
  $r := (z-fc_3)/s_3$;
  $f := -s_3f+c_3r$;
end
```

FIG. 3.4. Program executed by P_k, $k > 0$.



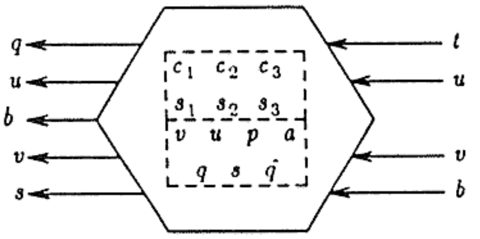


FIGURE 3.5

which the processors compute. Specifically, the processor P_{n-i} calculates the inner product of b and the $(i+1)$st column of Q, i.e., the $(i+1)$st components of the new right-hand side.

Prior to execution of Step 3 the parameters of transformations $G(\alpha_i)$, $G(\beta_i)$, $G(\gamma_i)$ are stored in the registers of the $(i-1)$st cell. (Note that this is the way in which the parameters are stored after Step 2 is completed. Thus we can start Step 3 without any I/O delays.) Data vectors t_1, v, u, b enter the array through the rightmost processor P_n (see Fig. 3.5). Before the computation starts, the register $\hat{q}$ of each processor is set to zero (this corresponds to the fact that in stage 3 of the algorithm SLLS from Section 2, the components of the vector q are shifted to the right before they are combined with the components of the vector v). The modified vectors v, u, q, s move from right to left.

On each cycle the processor P_n executes the program shown in Fig. 3.6.

```
begin
    $p := c_1v+s_1\hat{q}$;
    $v := s_1v+c_1\hat{q}$;
    $p := (p-uc_2)/s_2$;
    $u := -s_2u + c_2p$;
    $\hat{q} := t/r$;
    $a := a+b\hat{q}$;
    $s := \hat{q}$;
    $q := (p-sc_3)/s_3$;
    $s := -s_3s+c_3q$;
end
```

FIG. 3.6. Program executed by P_n.

Similarly, each processor P_i, $i = n-1, \ldots, 0$, executes the program shown in Fig. 3.7. Having completed the execution of its program, each processor shifts the modified vectors v, u, q, s and the vector b to the processor on its left and simultaneously receives new data for processing from the processor on its right.

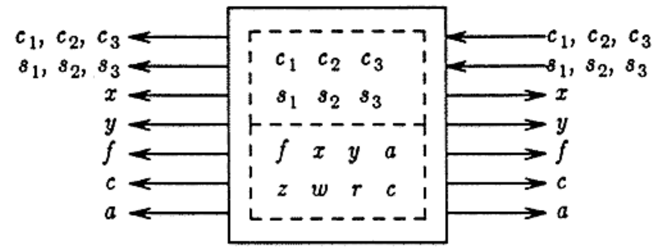
As the processor P_0 is active on the nth cycle and has to process vectors of length $m+1$, Step 3 requires $n+m+2$ units of time.

When the computation is finished the processor P_i still contains the parameters of the transformations $G(\alpha_i)$, $G(\beta_i)$, $G(\gamma_i)$ and in addition the $(n-i+1)$st component of the new right-hand-side vector a. At this point we are ready to start Step 4.

Step 4. Step 4 is similar in structure to Step 2 but is run backward. In Step 4 we regenerate the matrix R using only its last column and the transformations $G(\alpha)$, $G(\beta)$, $G(\gamma)$. The elements of R are regenerated in the exact order required by the Kung–Leiserson array for backsubstitution. Thus the regeneration and backsubstitution processes can be overlapped.

```
begin
    $p := c_1v+s_1\hat{q}$;
    $v := -s_1v+c_1\hat{q}$;
    $p := (p-uc_2)/s_2$;
    $u := -s_2u+c_2p$;
    $\hat{q} := q$;
    $a := a+b\hat{q}$;
    $q := (p-sc_3)/s_3$;
    $s := -s_3s+c_3q$;
end
```

FIG. 3.7. Program executed by P_i, $i < n$.



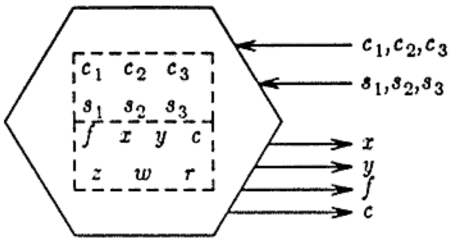


FIGURE 3.8

The array consists of $n+1$ processors $P_0, P_1, \ldots, P_n$, which are identical except for the leftmost processor P_0 (see Fig. 3.8). All processors regenerate rows of R. Additionally, processors $P_1, P_2, \ldots, P_n$ compute partial sums of the inner products in the backsubstitution process while the processor P_0 performs the division which gives the components of the solution vector.

Before iterating, we store the ith components of the vectors

$$\rho = [r_1, r_2, \ldots, r_{n+1}]$$
$$a = [a_1, a_2, \ldots, a_{n+1}]$$

in the registers of the processor P_{i-1}. The parameters of the transformations $G(\alpha_i)$, $G(\beta_i)$, $G(\gamma_i)$ are stored in the processor P_{i-1}. (Note that this is exactly how these quantities are left in the processors after Steps 2 and 3 are completed.)

Cell i is initialized at time i and is active on alternate cycles. The parameters of transformations and the partial sum of the inner products a flow from right to left. The intermediate results f, x, y and the components of the solution vector c flow from left to right.

When active, the processor P_0 executes the following program,

```
begin
  $c := a/r$
  $f := c_3r/s_3;$
  $z := c_3f+s_3r;$
  $x := c_2z/s_2;$
  $w := c_2x+s_2z;$
  $y := c_1w;$
  $r := -s_1w;$
end
```

Similarly, the processor P_i, $i > 0$, executes the following program:

```
begin
  $a := a-rc;$
  $f := -(f-c_3r)/s_3;$
  $z := c_3f+s_3r;$
  $x := -(x-c_2z)/s_2;$
  $w := c_2x+s_2z;$
  $y := c_1w+s_1y;$
  $r := -s_1w+c_1y;$
end
```

It is easy to see that the processor P_0 computes successive components of the solution vector c every second unit of time. Thus Step 4 requires $2n+1$ units of time.

Adding the execution time for Steps 1 to 4, we conclude that the least-squares problem (1.2) can be solved in time $O(n+m)$. Note that all arrays used require only a few registers per cell and that data enter the arrays only through one boundary cell.

APPENDIX A: Algorithm SLLS

Data: $T = [t_1, \ldots, t_{n+1}]$, where t_j is the jth column of the matrix T.

After step k of the triangularization we store the following quantities:

$\hat{y} = [y_{k+1}, \ldots, y_{n+1}]$	$(k+1)$st row of $G(\alpha_k)\cdots G(\alpha_1)[y, R_t^{\mathrm{T}}]^{\mathrm{T}}$
$w = [w_k, \ldots, w_{n+1}]$	kth row of W
$\hat{x} = [x_{k+1}, \ldots, x_{n+1}]$	$(k+1)$st row of $G(\beta_k)\cdots G(\beta_1)[z, Z^{\mathrm{T}}]^{\mathrm{T}}$
$z = [z_{k+1}, \ldots, z_{n+1}]$	kth row of Z
$p = [p_1, \ldots, p_{m+1}]$	kth row of P
$u = [u_1, \ldots, u_{m+1}]$	$(k+1)$st row of $G(\beta_k)\cdots G(\beta_1)[\begin{smallmatrix}0^{\mathrm{T}} & 1\\ P^{\mathrm{T}} & 0\end{smallmatrix}]$
$v = [v_1, \ldots, v_{m+1}]$	$(k+1)$st row of $G(\gamma_k)\cdots G(\gamma_1)[\begin{smallmatrix}1 & 0^{\mathrm{T}}\\ 0 & Q^{\mathrm{T}}\end{smallmatrix}]$
$\hat{f} = [f_{k+1}, \ldots, f_{n+1}]$	$(k+1)$st row of $G(\gamma_k)\cdots G(\gamma_1)[f, R_b^{\mathrm{T}}]^{\mathrm{T}}$
$r_{k+1} = [r_{k+1,k+1}, \ldots, r_{k+1,n+1}]$	$(k+1)$st row of R
$q_{k+1} = [q_{k+1,1}, \ldots, q_{k+1,m+1}]$	$(k+1)$st row of Q^{T}
$s = [s_1, \ldots, s_{m+1}]$	$(k+1)$st row of $G(\gamma_k)\cdots G(\gamma_1)Q^{\mathrm{T}}$

the parameters of the transformations $G(\alpha_1), G(\alpha_2), \ldots, G(\alpha_k), G(\beta_1), G(\beta_2), \ldots, G(\beta_k)$, and $G(\gamma_1), G(\gamma_2), \ldots, G(\gamma_k)$, i.e.,

$$\begin{bmatrix}\cos\alpha_1, \cos\alpha_2, \ldots, \cos\alpha_k\\ \sin\alpha_1, \sin\alpha_2, \ldots, \sin\alpha_k\end{bmatrix}$$

$$\begin{bmatrix}\cos\beta_1, \cos\beta_2, \ldots, \cos\beta_k\\ \sin\beta_1, \sin\beta_2, \ldots, \sin\beta_k\end{bmatrix}$$

$$\begin{bmatrix}\cos\gamma_1, \cos\gamma_2, \ldots, \cos\gamma_k\\ \sin\gamma_1, \sin\gamma_2, \ldots, \sin\gamma_k\end{bmatrix}$$

Initialization:

$$\hat{y} = [y_1, \ldots, y_n]^{\mathrm{T}} := [t_{-1}, t_{-2}, \ldots, t_{-n}]^{\mathrm{T}}$$

$$\hat{x} = [x_1, \ldots, x_n]^{\mathrm{T}} := [t_m, \ldots, t_{m-n+1}]^{\mathrm{T}}$$

$$v = [v_1, \ldots, v_{m+1}]^{\mathrm{T}} := [1, 0, \ldots, 0]^{\mathrm{T}}$$

$$\left(v \text{ is the first row of} \begin{bmatrix}1 & 0^{\mathrm{T}}\\ 0 & Q^{\mathrm{T}}\end{bmatrix}\right)$$

$$u = [u_1, \ldots, u_{m+1}]^{\mathrm{T}} := [0, 0, \ldots, 0]^{\mathrm{T}}$$

$$\left(u \text{ is the first row of} \begin{bmatrix}0^{\mathrm{T}} & 1\\ P^{\mathrm{T}} & 0\end{bmatrix}\right)$$

$$r_{1,1} := (t_1^{\mathrm{T}} t_1)^{1/2}$$

$$\bar{f} := [f_1, \ldots, f_n] := t_1^{\mathrm{T}}[t_2, \ldots, t_{n+1}]/r_{1,1}$$

$$r_1 = [r_{1,1}, \ldots, r_{1,n+1}] := [r_{1,1}, f]$$

$$q_1 = [q_1, \ldots, q_{m+1}]^{\mathrm{T}} := t_1/r_{1,1}$$

(q is the first column of Q)

$$s = [s_1, \ldots, s_{m+1}]^{\mathrm{T}} := q$$

Main loop:

For $k = 1, 2, \ldots, n$ **do**

1. {First update. Compute $[w, \hat{y}] = [e_k, e_{k+1}]^{\mathrm{T}} G(\alpha_k)\cdots G(\alpha_1)[y, R_t^{\mathrm{T}}]^{\mathrm{T}}$}

$$w_k := (y_k^2 + r_{k,k}^2)^{1/2}$$

$$\cos\alpha_k := y_k/w_k, \quad \sin\alpha_k := r_{k,k}/w_k$$

$$\begin{bmatrix}w_{k+1}, \ldots, w_n\\ y_{k+1}, \ldots, y_n\end{bmatrix} := \begin{bmatrix}\cos\alpha_k & \sin\alpha_k\\ -\sin\alpha_k & \cos\alpha_k\end{bmatrix}\begin{bmatrix}y_{k+1}, \ldots, y_n\\ r_{k,k+1}, \ldots, r_{k,n}\end{bmatrix}$$

2. {First downdate. Compute $z = e_k^{\mathrm{T}} Z$ and $\hat{x}^{\mathrm{T}} = e_{k+1}^{\mathrm{T}} G \times (\beta_k)\cdots G(\beta_1)[x, Z^{\mathrm{T}}]^{\mathrm{T}}$}

$$z_k := (w_k^2 - x_k^2)^{1/2}$$

$$\cos\beta_k := x_k/w_k, \quad \sin\beta_k := z_k/w_k$$

$$[z_{k+1}, \ldots, z_n] := ([w_{k+1}, \ldots, w_n] - [x_{k+1}, \ldots, x_n]\cos\beta_k)/\sin\beta_k$$

$$[x_{k+1}, \ldots, x_n] := [-\sin\beta_k \quad \cos\beta_k]\begin{bmatrix}x_{k+1}, \ldots, x_n\\ z_{k+1}, \ldots, x_n\end{bmatrix}$$

3. {Compute $p = Pe_k$, $v = e_{k+1}^{\mathrm{T}} G(\alpha_k)\cdots G(\alpha_1)[\begin{smallmatrix}1 & 0^{\mathrm{T}}\\ 0 & Q^{\mathrm{T}}\end{smallmatrix}]$ and $u = G(\alpha_k)\cdots G(\alpha_1)[\begin{smallmatrix}0^{\mathrm{T}} & 1\\ P^{\mathrm{T}} & 0\end{smallmatrix}]$}

$$\begin{bmatrix}p_1, \ldots, p_{m+1}\\ v_1, \ldots, v_{m+1}\end{bmatrix} := \begin{bmatrix}\cos\alpha_k & \sin\alpha_k\\ -\sin\alpha_k & \cos\alpha_k\end{bmatrix}\begin{bmatrix}v_1, v_2, \ldots, v_{m+1}\\ 0, q_1, \ldots, q_m\end{bmatrix}$$

$$[p_1, \ldots, p_{m+1}] := ([p_1, \ldots, p_{m+1}] - [u_1, \ldots, u_{m+1}]\cos\beta_k)/\sin\beta_k$$

$$[u_1, \cdots, u_{m+1}] := [-\sin\beta_k \quad \cos\beta_k]\begin{bmatrix}u_1, \ldots, u_{m+1}\\ p_1, \ldots, p_{m+1}\end{bmatrix}$$

4. {Second downdate. Compute $r_{k+1} = e_k^{\mathrm{T}}$ and $\hat{f} = e_{k+1}^{\mathrm{T}} g \times (\gamma_k)\cdots G(\gamma_1)[f, R_b^{\mathrm{T}}]^{\mathrm{T}}$}

$$r_{k+1} := (z_k^2 - f_k^2)^{1/2}$$

$$\cos\gamma_k := f_k/z_k, \quad \sin\gamma_k := r_{k+1,k+1}/z_k$$

$$[r_{k+1,k+2}, \ldots, r_{k+1,n+1}] := ([z_{k+1}, \ldots, z_n] - [f_{k+1}, \ldots, f_n]\cos\gamma_k)/\sin\gamma_k$$

$$[f_{k+1}, \ldots, f_n] := [-\sin\gamma_k \quad \cos\gamma_k]\begin{bmatrix}f_{k+1}, \ldots, f_n\\ r_{k+1,k+2}, \ldots, r_{k+1,n+1}\end{bmatrix}$$

5. $\{$Compute $q = Qe_{k+1}$ and $s = e_{k+1}^{\mathrm{T}} G(\gamma_k) \cdots G(\gamma_1) Q^{\mathrm{T}}\}$

$$q^{\mathrm{T}} = [q_{k+1,1}, \dots, q_{k+1,m+1}]$$
$$:= ([p_1, \dots, p_{m+1}] - [s_1, \dots, s_{m+1}]\cos\gamma_k)/\sin\gamma_k$$
$$[s_1, \dots, s_{m+1}] := [-\sin\gamma_k \quad \cos\gamma_k]\begin{bmatrix} s_1, \dots, s_{m+1} \\ q_1, \dots, q_{m+1} \end{bmatrix}$$

end of the k loop

APPENDIX B: Backsubstitution in $O(n)$ Storage

Data:

a	-new rhs
$\rho = [\rho_1, \dots, \rho_{n+1}]$	-last column of R
all transformations	

After step k of the backsubstitution phase store the following quantities:

$\rho = [\rho_1, \dots, \rho_{n-k}]$	first $n-k+1$ elements of the last column of R
$r = [r_{n-k}, \dots, r_{n+1}]$	$(n-k)$th row of R
$\bar{f} = [f_{n-k}, \dots, f_n]$	$(n-k)$th row of $G(\gamma_{n-k-1}) \cdots G(\gamma_1)[f, R_b^{\mathrm{T}}]^{\mathrm{T}}$
$z = [z_{n-k}, \dots, z_{n+1}]$	$(n-k)$th row of Z, $z = e_{n-k}^{\mathrm{T}} G \times (\gamma_{n-k}) \cdots G(\gamma_1)[f, R_b^{\mathrm{T}}]^{\mathrm{T}}$
$\hat{x} = [x_{n-k}, \dots, x_n]$	$(n-k)$th row of $G(\beta_{n-k-1}) \cdots G(\beta_1)[x, Z^{\mathrm{T}}]^{\mathrm{T}}$
$w = [w_{n-k}, \dots, w_n]$	$(n-k)$th row of W, $w = e_{n-k}^{\mathrm{T}} G \times (\beta_{n-k}) \cdots G(\beta_1)[x, Z^{\mathrm{T}}]^{\mathrm{T}}$
$\hat{y} = [y_{n-k}, \dots, y_n]$	$(n-k)$th row of $G(\alpha_{n-k+1}) \cdots G(\alpha_1)[y, R_t^{\mathrm{T}}]^{\mathrm{T}}$

$$\begin{bmatrix} \cos\alpha_1, \cos\alpha_2, \dots, \cos\alpha_{n-k-1} \\ \sin\alpha_1, \sin\alpha_2, \dots, \sin\alpha_{n-k-1} \end{bmatrix}$$
$$\begin{bmatrix} \cos\beta_1, \cos\beta_2, \dots, \cos\beta_{n-k-1} \\ \sin\beta_1, \sin\beta_2, \dots, \sin\beta_{n-k-1} \end{bmatrix}$$
$$\begin{bmatrix} \cos\gamma_1, \cos\gamma_2, \dots, \cos\gamma_{n-k-1} \\ \sin\gamma_1, \sin\gamma_2, \dots, \sin\gamma_{n-k-1} \end{bmatrix}$$

Main loop:

For $i = 0, 1, \dots, n-1$ **do**

6. $\{$Compute the $(n-k+1)$st component of $c\}$

$$c_{n-k+1} := (a_{n-k+1} - [r_{n-k+2}, \dots, r_{n+1}] \times [c_{n-k+2}, \dots, c_{n+1}]^{\mathrm{T}})/r_{n-k+1}$$

7. $\{$Compute $\bar{f} = e_{n-k}^{\mathrm{T}} G(\gamma_{n-k-1}) \cdots G(\gamma_1)[f, R_b^{\mathrm{T}}]^{\mathrm{T}}$ and $z = e_{n-k}^{\mathrm{T}} G(\gamma_{n-k}) \cdots G(\gamma_1)[f, R_b^{\mathrm{T}}]^{\mathrm{T}}\}$

$$r_{n+1} := \rho_{n-k+1}$$
$$f_{n-k} := 0$$
$$[f_{n-k}, \dots, f_n] := -([f_{n-k}, \dots, f_n] - [r_{n-k+1}, \dots, r_{n+1}]\cos\gamma_{n-k})/\sin\gamma_{n-k}$$
$$[f_{n-k}, \dots, f_n] := [\cos\gamma_{n-k}, \sin\gamma_{n-k}]\begin{bmatrix} f_{n-k}, \dots, f_n \\ r_{n-k+1}, \dots, r_{n+1} \end{bmatrix}$$

8. $\{$Compute $\hat{x} = e_{n-k}^{\mathrm{T}} G(\beta_{n-k-1}) \cdots G(\beta_1)[x, Z^{\mathrm{T}}]^{\mathrm{T}}$ and $w = e_{n-k}^{\mathrm{T}} G(\beta_{n-k}) \cdots G(\beta_1)[x, Z^{\mathrm{T}}]^{\mathrm{T}}\}$

$$x_{n-k} := 0$$
$$[x_{n-k}, \dots, x_n] := -([x_{n-k}, \dots, x_n] - [z_{n-k}, \dots, z_n]\cos\beta_{n-k})/\sin\beta_{n-k}$$
$$[w_{n-k}, \dots, w_n] := [\cos\beta_{n-k}, \quad \sin\beta_{n-k}]\begin{bmatrix} x_{n-k}, \dots, x_n \\ z_{n-k}, \dots, z_n \end{bmatrix}$$

9. $\{$Compute $r = e_{n-k}^{\mathrm{T}} R_t$ and $\hat{y} = e_{n-k}^{\mathrm{T}} G(\alpha_{n-k-1}) \cdots G(\alpha_1) \times [y, R_t^{\mathrm{T}}]^{\mathrm{T}}\}$

$$y_{n-k} := 0$$
$$\begin{bmatrix} y_{n-k}, \dots, y_n \\ r_{n-k}, \dots, r_n \end{bmatrix} := \begin{bmatrix} \cos\alpha_{n-k} & \sin\alpha_{n-k} \\ -\sin\alpha_{n-k} & \cos\alpha_{n-k} \end{bmatrix}\begin{bmatrix} w_{n-k}, \dots, w_n \\ y_{n-k}, \dots, y_n \end{bmatrix}$$

end of the k loop

$$r_{n+1} := \rho_1$$
$$c_1 := (a_1 - [r_2, \dots, r_{n+1}][c_2, \dots, c_{n+1}]^{\mathrm{T}})/r_1$$

end of the backsubstitution phase.

Note added in proof. Since this paper was submitted for publication, related work has been published by Chun, Kailath, and Lev-Ari [4] and Cybenko [5]. In particular, Chun *et al.* present a family of QR algorithms for matrices of low displacement rank, and Cybenko presents a generalization of the lattice algorithm for computing the inverse of the triangular matrix R in Eq. (2.1). For a survey of these and other recent results, see Brent [2].

REFERENCES

1. Bojanczyk, A. W., Brent, R. P., and de Hoog, F. R. QR Factorization of Toeplitz matrices. *Numer. Math.* **44** (1986), 81–94.
2. Brent, R. P. Old and new algorithms for Toeplitz systems. In Luk, F. T. (Ed.), *Advanced Algorithms and Architectures for Signal Processing III*, Proceedings SPIE, Vol. 975. SPIE, Bellingham, WA, 1988.
3. Brent, R. P., and Luk, F. T. A systolic array for the linear time solution of Toeplitz systems of equations. *J. VLSI Comput. Systems* **1,** 1 (1983), 1–23.
4. Chum, J., Kailath, T., and Lev-Ari, H. Fast parallel algorithms for the QR and triangular factorization. *SIAM J. Sci. Statist. Comput.* **8** (1987), 899–913.
5. Cybenko, G. Fast Toeplitz orthogonalization using inner products. *SIAM J. Sci. Statist. Comput.* **8** (1987), 734–740.
6. Fisher, A. L., Kung, H. T., Monier, L. M. M., and Dohi, Y. The architecture of a programmable systolic chip. *J. VLSI Comput. Systems* **1,** 2 (1984), 153–169.
7. Kung, H. T. Why systolic architectures? *IEEE Comput. Mag.* **15,** 1 (1982), 37–46.
8. Kung, S. Y., and Hu, Y. H. A highly concurrent algorithm and pipelined architecture for solving Toeplitz systems. *IEEE Trans. Acoust. Speech Signal Process.* **31,** 1 (Feb. 1983).
9. Sweet, D. R. Fast Toeplitz orthogonalization. *Numer. Math.* **43** (1984), 1–21.

Received July 2, 1985