Linear Complexity Parallel Algorithms for Linear Systems of Equations with Recursive Structure*

I. Gohberg
School of Mathematical Sciences
Raymond and Beverly Sackler Faculty of Exact Sciences
Tel Aviv University
Tel Aviv, Israel

T. Kailath
Information Systems Laboratory
Department of Electrical Engineering
Stanford University
Stanford, California

I. Koltracht and P. Lancaster
Department of Mathematics and Statistics
University of Calgary
Calgary, Alberta, Canada

In memory of James H. Wilkinson

Submitted by Jack Dongarra

ABSTRACT

A method of derivation of parallel algorithms for $(N+1)\times(N+1)$ matrices with recursive structure is presented and applied to Toeplitz, Hankel, and other Toeplitz-like matrices. The derived algorithms, executed on $O(N)$ parallel processors, require $O(N)$ arithmetic operations per processor.

1. INTRODUCTION

In the present paper we consider linear systems of equations

$$R\chi = \mathbf{f} \tag{1.1}$$

0024-3795/87/$3.50

where R is an $(N+1)\times(N+1)$ matrix with recursive structure as defined in [7]. We consider strongly regular coefficient matrices, that is, matrices with nonsingular principal leading minors. However, in Section 8 we indicate how to apply algorithms obtained here to matrices with some singular principal leading minors.

In [7] it is shown that matrices of Toeplitz, Hankel, close to Toeplitz, Hilbert, and Vandermonde type that are important in applications have recursive structure, and $O(N^2)$ recursive algorithms are derived. A common feature of these algorithms is that they require computation of inner products at each step of the recursion. Since calculation of the inner product of two vectors of length k on parallel processors requires at least $O(\log k)$ operations per processor, the algorithms of [7] are not well suited for implementation on parallel processors. In this present paper we show how to exclude inner-product computations from algorithms for matrices with recursive structure. This is done by a method that includes extension of the principal leading minors of R and still preserves the recursive structure of R. Algorithms of [7] are accordingly transformed into algorithms for extended vectors and no longer require the computation of inner products, thus allowing parallel implementation.

To illustrate, in the Levinson algorithm for a strongly regular symmetric Toeplitz matrix $R = \{r_{|i-j|}\}_0^N$ we find recursively, for $k = 1, \ldots, N$,

$$\mu_k = -\sum_{j=0}^{k-1} r_{j+1}\gamma_{k-1}(j), \tag{1.2}$$

$$\gamma_k(j) = \frac{1}{1-\mu_k^2}\left[\gamma_{k-1}(j-1) + \mu_k\gamma_{k-1}(k-j-1)\right] \quad \text{for } j = 1, \ldots, k-1, \tag{1.3}$$

$$\gamma_k(0) = \frac{\mu_k}{1-\mu_k^2}\gamma_{k-1}(k-1), \tag{1.4}$$

$$\gamma_k(k) = \frac{1}{1-\mu_k^2}\gamma_{k-1}(k-1), \tag{1.5}$$

starting with $\gamma_0(0) = 1/r_0$. The computed quantities determine the *UDL* factorization of R^{-1} [see (2.18) to follow]. The last vector γ_N also gives the inverse of R via the Gohberg-Semencul formula [6, p. 86].

The extended Levinson algorithm starts with

$$\gamma_0(0) = 1/r_0, \qquad \gamma_0(j) = -r_j/r_0, \quad j = 1,\ldots,N, \tag{1.6}$$

$$\gamma_0(-j) = -r_j/r_0, \quad j = 1,\ldots,N, \tag{1.7}$$

and finds recursively, for $k = 1,\ldots,N$,

$$\gamma_k(j) = \frac{1}{1-\gamma_{k-1}^2(-1)}\left[\gamma_{k-1}(j-1) + \gamma_{k-1}(-1)\gamma_{k-1}(k-j-1)\right],$$

$$j = k-N \ldots N, \quad j \neq 0, k, \tag{1.8}$$

$$\gamma_k(0) = \frac{\gamma_{k-1}(-1)\gamma_{k-1}(k-1)}{1-\gamma_{k-1}^2(-1)}, \tag{1.9}$$

$$\gamma_k(k) = \frac{\gamma_{k-1}(k-1)}{1-\gamma_{k-1}^2(-1)}. \tag{1.10}$$

We see that the inner product μ_k [of Equation (1.2)] is now found as the $\gamma_{k-1}(-1)$th entry of the vector $\gamma_{k-1} = [\gamma_{k-1}(k-N-1),\ldots,\gamma_{k-1}(N)]^T$. The extended vector γ_k consists of three parts. The central part $[\gamma_k(0),\ldots,\gamma_k(k)]^T$, after division by $\gamma_k(k)$, gives the kth column in the upper triangular factor of the *UDL* decomposition of R^{-1}, and the lower part $[\gamma_k(k+1),\ldots,\gamma_k(N)]^T$ with the opposite sign gives the kth column in the lower triangular factor of the *LDU* decomposition of R [see (2.11) and (2.18) below for details]. The upper part $[\gamma_k(k-N),\ldots,\gamma_k(-2)]^T$ is auxiliary.

The solution of the system of equations (1.1) can be computed from the vectors γ_k with a simultaneous recursion involving a delay of only one recursion step. More precisely, starting with $\mathbf{x}_0 = \mathbf{f} + f(0)[\gamma_0 - \mathbf{e}_0]$, calculate recursively for $k = 1,\ldots,N$

$$\chi_k(j) = \chi_{k-1}(j) + \chi_{k-1}(k)\gamma_k(j), \qquad j = 0,\ldots,N, \quad j \neq k, \tag{1.11}$$

$$\chi_k(k) = \chi_{k-1}(k)\gamma_k(k). \tag{1.12}$$

The solution of (1.1) is given by χ_N. This algorithm can be easily extended for simultaneous solution of (1.1) with any number of right-hand sides.

The algorithm (1.6)–(1.12) requires at most $2N+1$ multiplications per processor, and can be executed on $O(N)$ parallel processors (with the number of processors depending on the number of right-hand sides). The storage

requirement here is proportional to the number of processors. Since all quantities required for the solution and factorization of (1.1) are packed into one vector, the recursion (1.6)–(1.12) also allows efficient implementation on pipelined computers.

A parallel algorithm for the LDU decomposition of a positive definite Toeplitz matrix and a consequent solution of (1.1) using either this decomposition or the Gohberg-Semencul formula was presented in [13]. For nonsymmetric Toeplitz matrices a parallel algorithm with lower storage requirements based on the Barreis algorithm (see [2]) was given in [4]. A linear complexity algorithm for the Cholesky decomposition of a general positive definite matrix on a triangular array of processors is presented in [1]. In the case of a Toeplitz matrix, the triangular array reduces to a linear array similar to the one of [13]. These algorithms are of linear complexity, though the coefficients in the expression $O(N)$ are not given. It appears that these coefficients must be greater than $2N$, since all the above algorithms include the LDU decomposition of R as one of their steps, and this decomposition requires at least $2N$ multiplications per processor. In contrast, due to the efficient packing of data into one array, the whole algorithm of (1.6)–(1.12) has the same complexity as the LDU decomposition of R, namely $2N$.

It is a common feature of the extended algorithms for all classes of matrices considered here that together with the UDL factorization of R^{-1}, they also give the LDU factorization of R. Moreover, the computation of the LDU factors of R can be done independently, thus allowing further reduction in complexity and the number of processors.

The algorithms presented here also have a possible advantage in their low storage requirements. It will be seen that the solution of (1.1) will require that only $O(N)$ numbers be stored simultaneously. If required, the storage of triangular factors of R and R^{-1} will be more memory consuming, as $O(N^2)$ numbers must then be stored simultaneously.

In Section 2 the general extension method for matrices with structure is presented. In Sections 3–7 this method is used to obtain parallel algorithms for Toeplitz, Hankel, close to Toeplitz, Hilbert-type, and Vandermonde-type matrices respectively. Since all the algorithms presented find triangular factors of all principal leading minors and their inverses, the use of these algorithms can be recommended only for those matrices for which the conditioning of each principal leading minor is not worse than the conditioning of the matrix R itself. (Positive definite matrices obviously belong to this category.) In case of ill-conditioned or singular principal leading minors the method suggested in Section 8 can be used. This will generally mean a tradeoff between worse conditioning of the linear system to be solved and the improved complexity of computation offered by our algorithm.

In Section 9 we present some numerical experiments comparing roundoff error accumulation for the algorithms presented here with that of the

numerically stable QR algorithm with column pivoting [17, p. 111] and with that of Gaussian elimination with partial pivoting [17, p. 93]. The commonly used (and numerically infamous) Hilbert matrix [17, p. 108] is used as a test matrix whenever possible.

In all of these comparisons the parallel algorithms of this paper display qualities of numerical stability comparable to those of the standard algorithms (see also [18] and [19]). Indeed, in some cases the parallel algorithms demonstrate remarkably good properties in this respect.

The method used here, involving the extension of principal leading minors, originated in the paper by Gohberg and Koltracht [8]. In that paper the problem of numerical solution of Fredholm integral equations was addressed and led to parallel difference schemes for displacement, displacement plus Hankel, and close to displacement kernels. (Phenomena of this kind have been investigated in [20].)

2. PARALLEL ALGORITHMS FOR STRONGLY REGULAR MATRICES

Let $R = \{r_{ij}\}_{i,j=0}^{N}$ be any $(N+1)\times(N+1)$ matrix with nonsingular leading submatrices $R_k^l = \{r_{ij}\}_{i,j=0}^{k}$ for $k = 0, 1, \ldots, N$. Let

$$R = \begin{bmatrix} R_k^l & * \\ R_k^c & * \end{bmatrix}. \tag{2.1}$$

(Here, superscripts l and c denote "leading" and "complementary," respectively.)

Suppose we are given $q_k \times (k+1)$ matrices U_k for $k = 0, 1, \ldots, N$ of the form

$$U_k = \{u_{ij}\}_{i,j=-q_k,0}^{-1,k}, \qquad q_0 \geqslant q_1 \geqslant \cdots \geqslant q_N = 0,$$

and $p_k \times (k+1)$ matrices V_k for $k = 0, 1, \ldots, N$ of the form

$$V_k = \{v_{ij}\}_{i,j=1,0}^{p_k,k}, \qquad p_0 \geqslant p_1 \geqslant \cdots \geqslant p_N = 0.$$

Then we can define extensions of the leading submatrices R_k^l of the form

$$R_k = \begin{bmatrix} I_{q_k} & U_k & 0 & 0 \\ 0 & R_k^l & 0 & 0 \\ 0 & R_k^c & I_{N-k} & 0 \\ 0 & V_k & 0 & I_{p_k} \end{bmatrix} \tag{2.2}$$

and observe that R_k is nonsingular if and only if R_k^l is nonsingular. We also require that the extensions of R_k be nested in the sense that U_k with the last column deleted is equal to $E_k U_{k-1}^l$ (where E_k truncates the first $q_{k-1} - q_k$ rows of U_{k-1}), and V_k with deleted last column is equal to $F_k V_{k-1}$ (where F_k truncates the last $p_{k-1} - p_k$ rows of V_{k-1}). Thus

$$R_k = \begin{bmatrix} I_{q_k} & E_k U_{k-1} & \mathbf{u}_k & 0 & 0 \\ 0 & R_{k-1}^l & \mathbf{r}_k^l & 0 & 0 \\ 0 & R_{k-1}^c & \mathbf{r}_k^c & I_{N-k} & 0 \\ 0 & F_k V_{k-1} & \mathbf{v}_k & 0 & I_{p_k} \end{bmatrix}.$$

Extensions of this kind are called *structured*. We remark that U_k, V_k, or both can have zero dimension, which means that no extension is actually made in the corresponding direction.

Finally, suppose we are given a vector $\mathbf{f} \in \mathbb{C}^{N+1+p_0+q_0}$, $\mathbf{f}^T = [f(-q_0), \ldots, f(-1), f(0), \ldots, f(N), \ldots, f(N+p_0)]$. Define $\mathbf{f}_0^2 = [f(0), \ldots, f(N)]^T$. Let

$$\mathbf{f}_0 = \mathbf{f} = \begin{bmatrix} \mathbf{f}_0^1 \\ \mathbf{f}_0^2 \\ \mathbf{f}_0^3 \end{bmatrix},$$

and then recursively

$$\mathbf{f}_k = \begin{bmatrix} \mathbf{f}_k^1 \\ \mathbf{f}_k^2 \\ \mathbf{f}_k^3 \end{bmatrix} = \begin{bmatrix} E_k \mathbf{f}_{k-1}^1 \\ \mathbf{f}_{k-1}^2 \\ F_k \mathbf{f}_{k-1}^3 \end{bmatrix}, \qquad k = 1, \ldots, N,$$

such that, in particular, $\mathbf{f}_N = [f(0), \ldots, f(N)]^T$. Consistently with the above partitioning we denote by $\mathbf{e}_k \in \mathbb{C}^{N+1+p_k+q_k}$ the vector

$$\mathbf{e}_k = \begin{bmatrix} \mathbf{0} \\ \mathbf{e}_k^2 \\ \mathbf{0} \end{bmatrix},$$

where $\mathbf{e}_k^2 \in \mathbb{C}^{N+1}$, $\mathbf{e}_k^2 = [0, \ldots, 0, 1, 0, \ldots, 0]^T$, with 1 in the kth position.

It will also be convenient to introduce an operator $T_{k,k}$ which acts on vectors of size $N+1+p_{k-1}+q_{k-1}$ by eliminating the top $q_{k-1} - q_k$ and the

bottom $p_{k-1} - p_k$ elements and, in addition, replaces the entry $q_{k-1} + k + 1$ by zero. Thus,

$$T_{k,k} = \begin{bmatrix} 0 & \cdots & 0 & 1 & & & & & & 0 & \cdots & 0 \\ & & & \ddots & & & & 0 & & & & \\ & & & & & 1 & & & & & & \\ & & & & & & 0 & & & & & \\ & & & & & & & 1 & & & & \\ & & & & 0 & & & & \ddots & & & \\ & & & & & & & & & & & \\ 0 & \cdots & 0 & & & & & & & 1 & 0 & \cdots & 0 \end{bmatrix} \tag{2.3}$$

and

$$T_{k,k}\mathbf{f}_{k-1} = [f(-q_k),\ldots,f(-1),f(0),\ldots,f(k-1),0,f(k+1),\ldots,f(N+q_k)]. \tag{2.4}$$

Nestedness of extensions of principal leading minors ensures that the following recursive relation also holds for the extended solutions (see also Equation (2.3) of [7]). Thus, let χ_k and γ_k be the solutions of the equations $R_k\chi_k = \mathbf{f}_k$ and $R_k\gamma_k = \mathbf{e}_k$ for $k = 0,1,\ldots,N$. Then

$$\chi_k = T_{k,k}\chi_{k-1} + \chi_{k-1}(k)\gamma_k. \tag{2.5}$$

Indeed, using the nestedness of R_k in the above sense, it is easily verified that

$$R_k T_{k,k}\chi_{k-1} = \mathbf{f}_k - \chi_{k-1}(k)\mathbf{e}_k,$$

which implies (2.5).

Let us consider now the transposed equation $R^T\chi = \mathbf{f}$ and apply the above technique. The notation of Equation (2.1) is extended to give

$$R = \begin{bmatrix} R_k^l & R_k^d \\ R_k^c & * \end{bmatrix}, \qquad R^T = \begin{bmatrix} (R_k^l)^T & (R_k^c)^T \\ (R_k^d)^T & * \end{bmatrix}.$$

Then write

$$R_k^T = \begin{bmatrix} I_{q_k} & \tilde{U}_k & 0 & 0 \\ 0 & \left(R_k^l\right)^T & 0 & 0 \\ 0 & \left(R_k^d\right)^T & I_{N-k} & 0 \\ 0 & \tilde{V}_k & 0 & I_{p_k} \end{bmatrix}.$$

where $\tilde{U}_k$ and $\tilde{V}_k$ are nested in the same sense as U_k and V_k, respectively. If χ_k and ω_k denote the solutions of the equations $R_k^T\chi_k = \mathbf{f}_k$ and $R_k^T\omega_k = \mathbf{e}_k$ for $k = 0, 1, \ldots, N$ then, for the transposed matrices, we also have

$$\chi_k = T_{k,k}\chi_{k-1} + \chi_{k-1}(k)\omega_k. \tag{2.6}$$

The recursion (2.5) shows that if γ_k are known for $k = 0, 1, \ldots, N$, we can find the solution of the equation

$$R\chi = \mathbf{f} \tag{2.7}$$

on $O(N)$ processors in parallel at the expense of N multiplications and N additions per processor. Similarly, one can solve the transposed system

$$R^T\chi = \mathbf{f} \tag{2.8}$$

provided the vectors ω_k are known. We can see the equations (2.5) and (2.6) as two equations with four unknown vectors (those with subscript k). The idea now is to use the available structure of the matrix R to deduce additional equations for these unknowns by carefully choosing the right-hand sides in (2.7) and (2.8). To be precise, we give the following general

DEFINITION. A strongly regular $(N+1)\times(N+1)$ matrix $R = \{r_{ij}\}_{i,j=0}^N$ is said to have a parallel recursive structure if there exist extensions R_k and R_k^T, $k = 0, 1, \ldots, N$, of the form (1.4) and (1.5) and right-hand sides $\mathbf{g}^\nu, \mathbf{h}^\nu$, $\nu = 1, \ldots, \alpha$, such that for the solutions of the equations

$$R_k\varphi_k^\nu = \mathbf{g}_k^\nu, \qquad \nu = 1, \ldots, \alpha, \tag{2.9}$$

$$R_k^T\psi_k^\nu = \mathbf{h}_k^\nu, \qquad \nu = 1, \ldots, \alpha, \tag{2.10}$$

there exist operators Γ_k and W_k (not necessarily linear) such that, for $k = 1, \ldots, N$,

(1) we have

$$\gamma_k = \Gamma_k\left(\gamma_{k-1}, \omega_{k-1}, \varphi^1_{k-1}, \ldots, \varphi^\alpha_{k-1}, \psi^1_{k-1}, \ldots, \psi^\alpha_{k-1}\right), \tag{2.11}$$

$$\omega_k = W_k\left(\gamma_{k-1}, \omega_{k-1}, \varphi^1_{k-1}, \ldots, \varphi^\alpha_{k-1}, \psi^1_{k-1}, \ldots, \psi^\alpha_{k-1}\right); \tag{2.12}$$

(2) the computation of $\gamma_k(j)$ and $\omega_k(j)$ for each $j = -q_k, \ldots, N + p_k$ requires a number of arithmetic operations bounded by some number β which is independent of j and k.

In what follows we shall use flops to measure the complexity of algorithms. By a flop we understand one multiplication or division and one addition or subtraction: $ab \pm c$ or $a/b \pm c$. (For the justification of this use see [9, p. 32].) Thus the complexity of an algorithm is M flops if it takes no more than M flops per processor to get the solution.

The definition of a matrix with recursive structure implies that for such a matrix the vectors γ_k and ω_k, $k = 1, \ldots, N$, can be computed on $2(\alpha+1)(N + p_0 + q_0)$ parallel processors in $(\beta+1)N+1$ flops. Indeed, at step k we compute γ_k and ω_k in at most β flops per processor and then find φ^ν_k and ψ^ν_k, $\nu = 1, \ldots, \alpha$, via (2.5) and (2.6) in one flop per processor. We remark that there can be an obvious tradeoff between the number of processors and the complexity per processor per step.

In what follows we shall sometimes use more general forms of the above definition. For example, we may consider operators Γ_k and W_k depending on quantities computed in two or more previous steps of the recursion, or on solutions for several different extensions of R_k and R^T_k.

To start the recursions (2.5), (2.6), (2.11), and (2.12) we set

$$\gamma_0 = -r_{00}^{-1}\left[u_{-q_0,0}, \ldots, u_{-1,0}, -1, r_{10}, \ldots, r_{N0}, v_{1,0}, \ldots, v_{p_0,0}\right]^T, \tag{2.13}$$

$$\omega_0 = -r_{00}^{-1}\left[\tilde{u}_{-q_0,0}, \ldots, \tilde{u}_{-1,0}, -1, r_{01}, \ldots, r_{0N}, \tilde{v}_{1,0}, \ldots, \tilde{v}_{p_0,0}\right]^T, \tag{2.14}$$

$$\varphi^\nu_0 = \mathbf{g}^\nu + g^\nu(0)[\gamma_0 - \mathbf{e}_0], \qquad \nu = 1, \ldots, \alpha, \tag{2.15}$$

$$\psi^\nu_0 = \mathbf{h}^\nu + h^\nu(0)[\omega_0 - \mathbf{e}_0], \qquad \nu = 1, \ldots, \alpha. \tag{2.16}$$

Comparing the equations

$$R_k\gamma_k = \mathbf{e}_k \text{ and } R_k^T\omega_k = \mathbf{e}_k, \qquad k = 0,\dots,N,$$

with the equations

$$RU^{-1} = LD \quad \text{and} \quad R^TL^{-T} = U^TD,$$

we also get the triangular factorizations

$$R = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -\gamma_0(1) & 1 & \cdots & 0 \\ -\gamma_0(2) & -\gamma_1(2) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ -\gamma_0(N) & -\gamma_1(N) & \cdots & 1 \end{bmatrix} \begin{bmatrix} \dfrac{1}{\gamma_0(0)} & & & 0 \\ & \dfrac{1}{\gamma_1(1)} & & \\ & & \ddots & \\ 0 & & & \dfrac{1}{\gamma_N(N)} \end{bmatrix}$$

$$\times \begin{bmatrix} 1 & -\omega_0(1) & -\omega_0(2) & \cdots & -\omega_0(N) \\ 0 & 1 & -\omega_1(2) & \cdots & -\omega_1(N) \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}, \tag{2.17}$$

$$R^{-1} = \begin{bmatrix} 1 & \dfrac{\gamma_1(0)}{\gamma_1(1)} & \cdots & \dfrac{\gamma_N(0)}{\gamma_N(N)} \\ 0 & 1 & \cdots & \dfrac{\gamma_N(1)}{\gamma_N(N)} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \gamma_0(0) & & & 0 \\ & \gamma_1(1) & & \\ & & \ddots & \\ 0 & & & \gamma_N(N) \end{bmatrix}$$

$$\times \begin{bmatrix} 1 & 0 & \cdots & 0 \\ \dfrac{\omega_1(0)}{\omega_1(1)} & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ \dfrac{\omega_N(0)}{\omega_N(N)} & \dfrac{\omega_N(1)}{\omega_N(N)} & \cdots & 1 \end{bmatrix}. \tag{2.18}$$

3. TOEPLITZ MATRICES

Let $R = \{r_{i-j}\}_{i,j=0}^{N}$ be a strongly regular Toeplitz matrix. We define the structured extensions of principal leading minors R_k^l via (2.2), setting $p_0 = 0$ and

$$U_k = \begin{bmatrix} r_{k-N} & r_{k-N-1} & \cdots & r_{-N} \\ \vdots & \vdots & & \vdots \\ r_{-2} & r_{-3} & \cdots & r_{-k} \\ r_{-1} & r_{-2} & \cdots & r_{-k-1} \end{bmatrix}, \qquad k = 0,1,\ldots,N, \tag{3.1}$$

and similarly for R_k^T we set $p_0 = 0$ and

$$\tilde{U}_k = \begin{bmatrix} r_{N-k} & r_{N-k+1} & \cdots & r_N \\ \vdots & \vdots & & \vdots \\ r_2 & r_3 & \cdots & r_k \\ r_1 & r_2 & \cdots & r_{k+1} \end{bmatrix}, \qquad k = 0,1,\ldots,N. \tag{3.2}$$

Note that U_k and $\tilde{U}_k$ develop the Toeplitz structure of R in a natural way. Furthermore, the special structure of R_k and R_k^T can be fully expressed in terms of the rotation matrix J_{2N-k+1} of size $2N-k+1$:

$$J_{2N-k+1} = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix}. \tag{3.3}$$

In fact, we have

$$R_k^T = J_{2N-k+1} R_k J_{2N-k+1}, \tag{3.4}$$

which characterizes the persymmetry of the extension R_k. Now let $\mathbf{f} = \mathbf{e}_0$ in (2.5) and (2.6). Then $R_k \varphi_k = \mathbf{e}_0$ and $R_k^T \psi_k = \mathbf{e}_0$. Since $J_{2N-k+1}^2 = I$, we get from (3.4)

$$\varphi_k = J_{2N-k+1}\omega_k, \qquad \psi_k = J_{2N-k+1}\gamma_k, \tag{3.5}$$

where, as before, γ_k and ω_k are defined as the solutions of the equations

$R_k\gamma_k = \mathbf{e}_k$, $R_k^T\omega_k = \mathbf{e}_k$. Substituting (3.5) into (2.5) and (2.6), we get

$$J_{2N-k-1}\gamma_k = T_{k,k}J_{2N-k-2}\gamma_{k-1} + \gamma_{k-1}(-1)\omega_k,$$
$$J_{2N-k-1}\omega_k = T_{k,k}J_{2N-k-2}\omega_{k-1} + \omega_{k-1}(-1)\gamma_k,$$

which can be rewritten in the following way:

$$\gamma_k = T_{k,0}\gamma_{k-1} + \gamma_{k-1}(-1)J_{2N-k+1}\omega_k,$$
$$\omega_k = T_{k,0}\omega_{k-1} + \omega_{k-1}(-1)J_{2N-k+1}\gamma_k,$$

and we use the notation $T_{k,0} = J_{2N-k+1}T_{k,k}J_{2N-k}$. It follows from the strong regularity of R that $\gamma_{k-1}(-1)\omega_{k-1}(-1) \neq 1$. Thus

$$\gamma_k = \Gamma_k(\gamma_{k-1}, \omega_{k-1}),$$
$$\omega_k = W_k(\gamma_{k-1}, \omega_{k-1}),$$

where

$$\Gamma_k(\gamma_{k-1}, \omega_{k-1}) = \frac{1}{1-\gamma_{k-1}(-1)\omega_{k-1}(-1)} \times \left[T_{k,0}\gamma_{k-1} + \gamma_{k-1}(-1)J_{2N-k+1}T_{k,0}\omega_{k-1}\right], \quad (3.6)$$

$$W_k(\gamma_{k-1}, \omega_{k-1}) = \frac{1}{1-\gamma_{k-1}(-1)\omega_{k-1}(-1)} \times \left[T_{k,0}\omega_{k-1} + \omega_{k-1}(-1)J_{2N-k+1}T_{k,0}\gamma_{k-1}\right]. \quad (3.7)$$

We can also write

$$\begin{bmatrix}\gamma_k \\ \omega_k\end{bmatrix} = \frac{1}{1-\gamma_{k-1}(-1)\omega_{k-1}(-1)} \times \begin{bmatrix} I & \gamma_{k-1}(-1)J_{2N-k+1} \\ \omega_{k-1}(-1)J_{2N-k+1} & I \end{bmatrix}\begin{bmatrix} T_{k,0}\gamma_{k-1} \\ T_{k,0}\omega_{k-1}\end{bmatrix}. \quad (3.8)$$

ALGORITHM 3.1. Let $R = \{r_{i-j}\}_{i,j=0}^{N}$ be a strongly regular Toeplitz matrix. Then one can find the solution of the equation

$$R\chi = \mathbf{f} \quad (3.9)$$

in the following way.

1. Start with

$$\gamma_0(0) = \omega_0(0) = 1/r_0; \qquad \gamma_0(j) = \omega_0(-j) = -r_j/r_0, \quad j = \pm 1, \ldots, \pm N;$$

$$\chi_0(0) = f(0)/r_0; \qquad \chi_0(j) = f(j) - f(0)r_j/r_0, \quad j = 1, \ldots, N.$$

2. Compute recursively for $k = 1, \ldots, N$

$$\beta_k = 1 - \gamma_{k-1}(-1)\omega_{k-1}(-1);$$

$$\gamma_k(0) = \beta_k^{-1}\gamma_{k-1}(-1)\omega_{k-1}(k-1), \qquad \gamma_k(k) = \beta_k^{-1}\gamma_{k-1}(k-1),$$

$$\omega_k(0) = \beta_k^{-1}\gamma_{k-1}(-1)\gamma_{k-1}(k-1), \qquad \omega_k(k) = \beta_k^{-1}\omega_{k-1}(k-1);$$

$$\gamma_k(j) = \beta_k^{-1}\left[\gamma_{k-1}(j-1) + \gamma_{k-1}(-1)\omega_{k-1}(k-j-1)\right], \quad j = k-N, \ldots, N, \quad j \neq 0, k,$$

$$\omega_k(j) = \beta_k^{-1}\left[\omega_{k-1}(j-1) + \omega_{k-1}(-1)\gamma_{k-1}(k-j-1)\right], \quad j = k-N, \ldots, N, \quad j \neq 0, k;$$

and starting with $k = 2$,

$$\chi_{k-1}(k-1) = \chi_{k-2}(k-1)\gamma_{k-1}(k-1);$$

$$\chi_{k-1}(j) = \chi_{k-2}(j) + \chi_{k-2}(k-1)\gamma_{k-1}(j), \quad j = 0, \ldots, N, \quad j \neq k-1;$$

3. The vector χ_N computed via

$$\chi_N(N) = \chi_{N-1}(N)\gamma_N(N),$$

$$\chi_N(j) = \chi_{N-1}(j) + \chi_{N-1}(N)\gamma_N(j), \qquad j = 0, \ldots, N-1,$$

gives the solution of (3.9).

This algorithm can be executed on $5N+2$ parallel processors with no more than $2N+1$ flops per processor. Indeed, we can compute β_k, $\gamma_{k-1}(j-1) + \gamma_{k-1}(-1)\omega_{k-1}(k-j-1)$ and $\omega_{k-1}(j-1) + \omega_{k-1}(-1)\gamma_{k-1}(k-j-1)$ for $j = k-N, \ldots, N$, $j \neq 0, k$, simultaneously in one flop and then find $\gamma_k(j)$ and $\omega_k(j)$, $j = k-N, \ldots, N$, in one division. The vectors χ_k are computed with one delay.

It has been seen in equations (2.17) and (2.18) that the vectors γ_k, ω_k, $k = 0,\dots,N$, also yield the *LDU* factorization of R and the *UDL* factorization of R^{-1}. It follows from Algorithm 3.1 that the *LDU* factorization of R can be found independently as follows:

ALGORITHM 3.2. Under the conditions of Algorithm 3.1:

1. Start with

$$\gamma_0(j) = \omega_0(j) = -r_j/r_0, \qquad j = \pm 1,\dots,\pm N.$$

2. Compute recursively for $k = 1,\dots,N$

$$\beta_k = 1 - \gamma_{k-1}(-1)\omega_{k-1}(-1),$$

$$\gamma_k(j) = \beta_k^{-1}[\gamma_{k-1}(j-1) + \gamma_{k-1}(-1)\omega_{k-1}(k-j-1)],$$
$$j = -1,\dots,k-N, \quad j = k+1,\dots,N,$$

$$\omega_k(j) = \beta_k^{-1}[\omega_{k-1}(j-1) + \omega_{k-1}(-1)\gamma_{k-1}(k-j-1)],$$
$$j = -1,\dots,k-N, \quad j = k+1,\dots,N.$$

3. The *LDU* factorization of R is now given by (2.17).

Both Algorithms 3.1 and 3.2 can be further simplified if R is symmetric, i.e., $r_j = r_{-j}$, $j = 1,\dots,N$, in which case $\gamma_k = \omega_k$ for $k = 0,\dots,N$. The symmetric version of the Algorithm 3.2 is also known as the Schur algorithm (see [10], for example).

4. HANKEL MATRICES

Let $R = \{r_{i+j}\}_{i,j=0}^{N}$ be a strongly regular Hankel matrix. The extended principal leading minors R_k are defined via (2.2) by setting $q_0 = 0$ and

$$V_k = \begin{bmatrix} r_{N+1} & r_{N+2} & \cdots & r_{2N-k} \\ r_{N+2} & r_{N+3} & \cdots & r_{2N-k+1} \\ \vdots & \vdots & & \vdots \\ r_{2N-k} & r_{2N-k+1} & \cdots & r_{2N} \end{bmatrix}, \qquad k = 0,\dots,N, \quad (4.1)$$

thus developing the Hankel structure of R in a natural way.

We shall describe two possible choices of operators Γ_k. The first one yields a parallel version of an algorithm originally due to Lanczos (1952). So let $R\gamma_k = \mathbf{e}_k$, and let us introduce for $k = 1, \ldots, N$ three linear operators: $T_{k,0}, T_{k,k}$ which map $\mathbb{C}^{2N-k+2}$ into $\mathbb{C}^{2N-k+1}$, and $T_{k,k+1}$ which maps $\mathbb{C}^{2N-k+3}$ into $\mathbb{C}^{2N-k+1}$, according to the rules

$$T_{k,0}[f(0),\ldots,f(2N-k+2)]^T = [0, f(0),\ldots,f(k-1), f(k+2),\ldots,f(2N-k+2)]^T,$$

$$T_{k,k}[f(0),\ldots,f(2N-k+2)]^T = [f(0),\ldots,f(k-1),0,f(k+1),\ldots,f(2N-k+1)]^T,$$

$$T_{k,k+1}[f(0),\ldots,f(2N-k+3)^T = [f(0),\ldots,f(k-2),0,0,f(k+1),\ldots,f(2N-k+1)]^T$$

It is obvious that

$$R_k T_{k,k}\gamma_{k-1} = \mathbf{e}_{k-1} - \gamma_{k-1}(k)\mathbf{e}_k$$

and

$$R_k T_{k,k+1}\gamma_{k-2} = \mathbf{e}_{k-2} - \gamma_{k-2}(k-1)\mathbf{e}_{k-1} - \gamma_{k-2}(k)\mathbf{e}_k.$$

It also follows from the Hankel structure of R_k that

$$R_k T_{k,0}\gamma_{k-1} = \mathbf{e}_{k-2} - \gamma_{k-1}(k)\mathbf{e}_{k-1} - \gamma_{k-1}(k+1)\mathbf{e}_k.$$

Thus

$$R_k[T_{k,0}\gamma_{k-1} - T_{k,k+1}\gamma_{k-2} + [\gamma_{k-1}(k) - \gamma_{k-2}(k-1)]T_{k,k}\gamma_{k-1}]$$
$$= [\gamma_{k-2}(k) - \gamma_{k-1}(k+1) + \gamma_{k-1}(k)[\gamma_{k-2}(k-1) - \gamma_{k-1}(k)]]\mathbf{e}_k.$$

It follows from the strong regularity of R that

$$\beta_k = \gamma_{k-2}(k) - \gamma_{k-1}(k+1) + \gamma_{k-1}(k)[\gamma_{k-2}(k-1) - \gamma_{k-1}(k)] \neq 0.$$

Thus

$$\gamma_k = \Gamma_k(\gamma_{k-1}, \gamma_{k-2}),$$

where

$$\Gamma_k(\gamma_{k-1}, \gamma_{k-2})$$

$$= \beta_k^{-1}\{T_{k,1}\gamma_{k-1} - T_{k,k+1}\gamma_{k-2} + [\gamma_{k-1}(k) - \gamma_{k-2}(k-1)]T_{k,k}\gamma_{k-1}\}. \tag{4.2}$$

We see that in this case γ_k is expressed in terms of quantities computed in two previous steps of the recursion. This case can be included in the definition of Section 2 by an obvious generalization of Equations (2.11) and (2.12).

ALGORITHM 4.1. Let $R = \{r_{i+j}\}_{i,j=0}^{N}$ be a strongly regular Hankel matrix. Then the solution of the equation

$$R\chi = \mathbf{f} \tag{4.3}$$

can be found in the following way.

1. Start with

$$\gamma_0(0) = \frac{1}{r_0}, \qquad \gamma_0(j) = -\frac{r_j}{r_0}, \quad j = 1,\ldots,2N,$$

$$\gamma_1(0) = -\frac{r_1}{r_0 r_2 - r_1^2}, \qquad \gamma_1(1) = \frac{r_0}{r_0 r_2 - r_1^2},$$

$$\gamma_1(j) = -r_j\gamma_1(0) - r_{j+1}\gamma_1(1), \qquad j = 2,\ldots,2N-1,$$

$$\chi_0(0) = \frac{f(0)}{r_0}, \qquad \chi_0(j) = f(j) - \frac{r_j f(0)}{r_0}, \quad j = 1,\ldots,2N.$$

2. Compute recursively for $k = 2, \ldots, N$

$$\delta_k = \gamma_{k-2}(k-1) - \gamma_{k-1}(k), \qquad \beta_k = \gamma_{k-2}(k) - \gamma_{k-1}(k+1) + \gamma_{k-1}(k)\delta_k,$$

$$\gamma_k(0) = -\beta_k^{-1}\left[\delta_k \gamma_{k-1}(0) + \gamma_{k-2}(0)\right],$$

$$\gamma_k(k-1) = \beta_k^{-1}\left[\gamma_{k-1}(k-2) - \delta_k \gamma_{k-1}(k-1)\right],$$

$$\gamma_k(k) = \beta_k^{-1}\gamma_{k-1}(k-1),$$

$$\gamma_k(j) = \beta_k^{-1}\left[\gamma_{k-1}(j-1) - \gamma_{k-2}(j) - \delta_k \gamma_{k-1}(j)\right], \qquad j = 1, \ldots, k-2,$$

$$\gamma_k(j) = \beta_k^{-1}\left[\gamma_{k-1}(j+1) - \gamma_{k-2}(j) - \delta_k \gamma_{k-1}(j)\right],$$

$$j = k+1, \ldots, 2N-k,$$

$$\chi_{k-1}(j) = \chi_{k-2}(j) + \chi_{k-2}(k-1)\gamma_{k-1}(j),$$

$$j = 0, \ldots, 2N-k, \quad j \neq k-1,$$

$$\chi_{k-1}(k-1) = \chi_{k-2}(k-1)\gamma_{k-1}(k-1).$$

3. The vector χ_N,

$$\chi_N(j) = \chi_{N-1}(j) + \chi_{N-1}(N)\gamma_N(j), \qquad j = 0, \ldots, N-1,$$

$$\chi_N(N) = \chi_{N-1}(N)\gamma_N(N)$$

gives the solution of (4.3).

This algorithm can be executed on $3N+4$ parallel processors at the expense of no more than $2N+1$ flops per processor. Indeed, we can compute simultaneously δ_k, $\gamma_{k-2}(k) - \gamma_{k-1}(k+1)$, and $\gamma_{k-1}(j \pm 1) - \gamma_{k-2}(j)$ for all j, then find $\delta_k \gamma_{k-1}(j)$ and β_k simultaneously, and finally find all $\gamma_k(j)$. The vector χ_{k-1} is computed with one delay.

The *LDU* factorization of R is given by:

ALGORITHM 4.2. Under the conditions of Algorithm 4.1:

1. Start with

$$\gamma_0(j) = -\frac{r_j}{r_0}, \qquad j = 1, \ldots, 2N,$$

$$\gamma_1(j) = -r_j\gamma_1(0) - \frac{r_0 r_{j+1}}{r_0 r_2 - r_1^2}, \qquad j = 2, \ldots, 2N-1.$$

2. Compute recursively for $k = 2, \ldots, N$

$$\delta_k = \gamma_{k-2}(k-1) - \gamma_{k-1}(k),$$

$$\beta_k = \gamma_{k-2}(k) - \gamma_{k-1}(k+1) - \gamma_{k-1}(k)\delta_k,$$

$$\gamma_k(j) = \beta_k^{-1}\left[\gamma_{k-1}(j+1) - \gamma_{k-2}(j) - \delta_k\gamma_{k-1}(j)\right], \qquad j = k+1, \ldots, 2N-k.$$

3. The *LDU* factorization of R is given by (2.17).

Another algorithm for Hankel matrices includes the solutions of the equations $R_k\varphi_k = \mathbf{e}_0$ together with the calculation of the γ_k. It is straightforward to check that

$$R_k\left[T_{k,0}\varphi_{k-1} + \varphi_{k-1}(k)T_{k,k}\gamma_{k-1}\right] = -\left[\varphi_{k-1}(k)\gamma_{k-1}(k) + \varphi_{k-1}(k+1)\right]\mathbf{e}_k.$$

Therefore if

$$\lambda_k = -\varphi_{k-1}(k)\gamma_{k-1}(k) - \varphi_{k-1}(k+1) \neq 0 \tag{4.4}$$

then

$$\gamma_k = \frac{1}{\lambda_k}\left[T_{k,0}\varphi_{k-1} + \varphi_{k-1}(k)T_{k,k}\gamma_{k-1}\right].$$

If $\lambda_k = 0$ we can find γ_k in terms of γ_{k-2} and φ_{k-2} in the following way:

$$\gamma_k = \frac{1}{\mu_k}\left(T_{k,0}T_{k-1,0}\varphi_{k-2} + \varphi_{k-2}(k)T_{k,k}T_{k-1,k-1}\gamma_{k-2} + \frac{\nu_k}{\varphi_{k-2}(k)}T_{k,k}T_{k-1,0}\varphi_{k-2}\right),$$

where

$$\nu_k = -\left[\varphi_{k-2}(k+1) + \varphi_{k-2}(k)\gamma_{k-2}(k-1)\right], \tag{4.5}$$

$$\mu_k = -\left[\varphi_{k-2}(k+2) + \varphi_{k-2}(k)\gamma_{k-2}(k) - \varphi_{k-2}(k+1)\nu_k\right]. \tag{4.6}$$

Thus, in this case, we get the following expression for Γ_k:

$$\Gamma_k(\gamma_{k-1}, \gamma_{k-2}, \varphi_{k-1}, \varphi_{k-2})$$

$$= \begin{cases} \dfrac{1}{\lambda_k}\left[T_{k,0}\varphi_{k-1} + \varphi_{k-1}(k)T_{k,k}\gamma_{k-1}\right] & \text{if} \quad \lambda_k \neq 0, \\ \dfrac{1}{\mu_k}\left[T_{k,0}T_{k-1,0}\varphi_{k-2} + \varphi_{k-2}(k)T_{k,k}T_{k-1,k-1}\gamma_{k-2}\right. & \\ \qquad \left. + \dfrac{\nu_k}{\varphi_{k-2}(k)}T_{k,k}T_{k-1,0}\varphi_{k-2}\right] & \text{if} \quad \lambda_k = 0, \end{cases}$$

where λ_k, μ_k, and ν_k are defined in (4.4), (4.5), and (4.6). Again, γ_k is expressed in terms of quantities computed in two previous steps of the recursion.

ALGORITHM 4.3. Under the conditions of Algorithm 4.1:

1. Start with $\varphi_0 = \gamma_0$ and χ_0 as defined in Algorithm 4.1.
2. Compute recursively for $k = 1, \ldots, N$ as follows:

If $\varphi_{k-1}(k) \neq 0$ then

$$\lambda_k = -\left[\varphi_{k-1}(k)\gamma_{k-1}(k) + \varphi_{k-1}(k+1)\right],$$

$$\gamma_k(0) = \lambda_k^{-1}\varphi_{k-1}(k)\gamma_{k-1}(0), \qquad \gamma_k(k) = \lambda_k^{-1}\varphi_{k-1}(k-1),$$

$$\gamma_k(j) = \lambda_k^{-1}\left[\varphi_{k-1}(j-1) + \varphi_{k-1}(k)\gamma_{k-1}(j)\right], \qquad j = 1, \ldots, k-1,$$

$$\gamma_k(j) = \lambda_k^{-1}\left[\varphi_{k-1}(j+1) + \varphi_{k-1}(k)\gamma_{k-1}(j)\right], \qquad j = k+1, \ldots, 2N-k,$$

$$\varphi_k(k) = \varphi_{k-1}(k)\gamma_k(k),$$

$$\varphi_k(j) = \varphi_{k-1}(j) + \varphi_{k-1}(k)\gamma_k(j), \qquad j = 0, \ldots, 2N-k, \quad j \neq k,$$

$$\chi_k(k) = \chi_{k-1}(k)\gamma_k(k),$$

$$\chi_k(j) = \chi_{k-1}(j) + \chi_{k-1}(k)\gamma_k(j), \qquad j = 0, \ldots, N, \quad j \neq k.$$

If $\varphi_{k-1}(k)=0$ then

$$\nu_{k+1}=-\left[\varphi_{k-1}(k+2)+\varphi_{k-1}(k+1)\gamma_{k-1}(k)\right],$$

$$\mu_{k+1}=-\left[\varphi_{k-1}(k+3)+\varphi_{k-1}(k+1)\gamma_{k-1}(k+1)-\varphi_{k-1}(k+2)\nu_{k+1}\right],$$

$$\gamma_{k+1}(0)=\mu_{k+1}^{-1}\varphi_{k-1}(k+1)\gamma_{k-1}(0),$$

$$\gamma_{k+1}(1)=\mu_{k+1}^{-1}\left(\varphi_{k-1}(k+1)\gamma_{k-1}(1)+\frac{\nu_{k+1}}{\varphi_{k-1}(k+1)}\varphi_{k-1}(0)\right),$$

$$\gamma_{k+1}(k)=\mu_{k+1}^{-1}\left(\varphi_{k-1}(k-2)+\frac{\nu_{k+1}}{\varphi_{k-1}(k+1)}\varphi_{k-1}(k-1)\right),$$

$$\gamma_{k+1}(k+1)=\mu_{k+1}^{-1}\varphi_{k-1}(k-1),$$

$$\gamma_{k+1}(j)=\mu_{k+1}^{-1}\left(\varphi_{k-1}(j-2)+\varphi_{k-1}(k+1)\gamma_{k-1}(j)+\frac{\nu_{k+1}}{\varphi_{k-1}(k+1)}\varphi_{k-1}(j-1)\right),\qquad j=2,\ldots,k-1,$$

$$\gamma_{k+1}(j)=\mu_{k+1}^{-1}\left(\varphi_{k-1}(j+2)+\varphi_{k-1}(k+1)\gamma_{k-1}(j)+\frac{\nu_{k+1}}{\varphi_{k-1}(k+1)}\varphi_{k-1}(j+1)\right),\qquad j=k+2,\ldots,2N-k-1,$$

$$\varphi_{k+1}(k)=\varphi_{k-1}(k+1)\gamma_{k+1}(k),\qquad \varphi_{k+1}(k+1)=\varphi_{k-1}(k+1)\gamma_{k+1}(k+1),$$

$$\varphi_{k+1}(j)=\varphi_{k-1}(j)+\varphi_{k-1}(k+1)\gamma_{k+1}(j),$$

$$j=0,\ldots,2N-k-1,\quad j\neq k,k+1,$$

$$\chi_{k+1}(0)=\chi_{k-1}(0)+\chi_{k-1}(k+1)\gamma_{k+1}(0),$$

$$\chi_{k+1}(k)=\chi_{k-1}(k+1)\gamma_{k+1}(k)-\frac{\chi_{k-1}(k)}{\varphi_{k-1}(k+1)}\varphi_{k-1}(k-1),$$

$$\chi_{k+1}(k+1)=\chi_{k-1}(k+1)\gamma_{k+1}(k+1)-\frac{\chi_{k-1}(k)}{\varphi_{k-1}(k+1)}\varphi_{k-1}(k+2),$$

$$\chi_{k+1}(j) = \chi_{k-1}(j) + \chi_{k-1}(k+1)\gamma_{k+1}(j) - \frac{\chi_{k-1}(k)}{\varphi_{k-1}(k+1)}\varphi_{k-1}(j-1), \qquad j = 1, \ldots, k-1,$$

$$\chi_{k+1}(j) = \chi_{k-1}(j) + \chi_{k-1}(k+1)\gamma_{k-1}(j) - \frac{\chi_{k-1}(k)}{\varphi_{k-1}(k+1)}\varphi_{k-1}(j+1), \qquad j = k+2, \ldots, N.$$

This algorithm has a similar complexity to Algorithm 4.1, but it can be more efficient for the computation of φ_N. It is easy to see that $\varphi_k(k-1) \neq 0$ if and only if the subprincipal leading minors $R_k^s = \{r_{i+j-1}\}_{i,j=1}^k$ are nonsingular. The knowledge of φ_N can be important if $\varphi_N(N-1) \neq 0$, because in this case we can use the Gohberg-Semencul formula (see [6, p. 86]) to write R^{-1} as follows:

$$R^{-1} = \frac{1}{\gamma_N(0)} \left\{ \begin{bmatrix} \gamma_N(0) & & 0 \\ \vdots & \ddots & \\ \gamma_N(N) & \cdots & \gamma_N(0) \end{bmatrix} \begin{bmatrix} \varphi_N(N) & \cdots & \varphi_N(0) \\ & \ddots & \vdots \\ 0 & & \varphi_N(N) \end{bmatrix} \right.$$

$$- \begin{bmatrix} 0 & & & & & 0 \\ \varphi_N(0) & \cdot & & & & \\ \cdot & \cdot & \cdot & & & \\ \cdot & & \cdot & \cdot & & \\ \varphi_N(N-1) & \cdot & \cdot & \cdot & \varphi_N(0) & 0 \end{bmatrix}$$

$$\left. \times \begin{bmatrix} 0 & \gamma_N(N) & \cdot & \cdot & \cdot & \gamma_N(1) \\ & \cdot & \cdot & & & \cdot \\ & & \cdot & \cdot & & \cdot \\ & & & \cdot & \cdot & \gamma_N(N) \\ 0 & & & & & 0 \end{bmatrix} \right\} J, \quad (4.7)$$

where J is the $(N+1)\times(N+1)$ rotation matrix defined in (3.3).

5. CLOSE-TO-TOEPLITZ MATRICES

According to the definition given in [5] an $(N+1)\times(N+1)$ matrix R is called close to Toeplitz if it admits the representation

$$R = \sum_{i=1}^{\alpha} L^i U^i \tag{5.1}$$

where L^i are lower triangular Toeplitz matrices

$$L^i = \begin{bmatrix} r_0^i & 0 & \cdots & 0 \\ r_1^i & r_0^i & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ r_N^i & r_{N-1}^i & \cdots & r_0^i \end{bmatrix} \tag{5.2}$$

and U^i are upper triangular Toeplitz matrices

$$U^i = \begin{bmatrix} s_0^i & s_1^i & \cdots & s_N^i \\ 0 & s_0^i & \cdots & s_{N-1}^i \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & s_0^i \end{bmatrix}. \tag{5.3}$$

It is clear that the minors R_k will have the same structure as R, namely

$$R_k^l = \sum_{i=1}^{\alpha} \left(L_k^i\right)^l \left(U_k^i\right)^l.$$

Let us introduce α extensions of R_k^l,

$$R_k^j = \sum_{i=1}^{\alpha} \begin{bmatrix} 0 & 0 & 0 \\ 0 & \left(L_k^i\right)^l & 0 \\ 0 & \left(L_k^i\right)^C & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & \left(U_k^i\right)^l & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & \left(U_k^j\right)^C & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I \end{bmatrix}, \qquad j = 1, \ldots, \alpha \tag{5.4}$$

where

$$\left(U_k^j\right)^C = \begin{bmatrix} s_{N-k}^j & & \cdots & s_N^j \\ \vdots & \vdots & & \vdots \\ s_1^j & s_2^j & \cdots & s_{k+1}^j \end{bmatrix} \tag{5.5}$$

and solutions of the equations $R_k^j \gamma_k^j = \mathbf{e}_k$, $R_k^j \varphi_k^{i,j} = \mathbf{r}_i$, where $\mathbf{r}_i = [0,\ldots,0,r_0^i,\ldots,r_N^i]^T$. Let $T_{k,0}$ map C^{2N-k+1} into C^{2N-k} according to the rule

$$T_{k,0}[f(k-N-1),\ldots,f(-1),f(0),\ldots,f(N)]^T$$
$$= [f(k-N-1),\ldots,f(-2),0,f(0),\ldots,f(N-1)]^T. \tag{5.6}$$

Then it is straightforward to check that

$$\gamma_k^j = T_{k,0}\gamma_{k-1}^j + \sum_{i=1}^{\alpha} \gamma_{k-1}^i(-1)\varphi_k^{i,j}. \tag{5.7}$$

Using (2.5), we get

$$\left[1 - \sum_{i=1}^{\alpha} \gamma_{k-1}^i(-1)\varphi_{k-1}^{i,j}(k)\right]\gamma_k^j = T_{k,0}\gamma_{k-1}^j + \sum_{i=1}^{\alpha} \gamma_{k-1}^i(-1)T_{k,k}\varphi_{k-1}^{i,j},$$

where $T_{k,k}$ is defined in (2.4). It is easy to see [7] that $1 \neq \sum_{i=1}^{\alpha}\gamma_{k-1}^i(-1)\varphi_{k-1}^{i,j}(k)$. Thus, in the close-to-Toeplitz case, we have α operators Γ_k^j, $j = 1,\ldots,\alpha$, defined by

$$\Gamma_k^j\left(\gamma_{k-1}^1,\ldots,\gamma_{k-1}^{\alpha},\varphi_{k-1}^{1,j},\ldots,\varphi_{k-1}^{\alpha,j}\right) = \frac{1}{1-\sum_{i=1}^{\alpha}\gamma_{k-1}^i(-1)\varphi_{k-1}^{i,j}(k)}$$
$$\times\left[T_{k,0}\gamma_{k-1}^j + \sum_{i=1}^{\alpha}\gamma_{k-1}^i(-1)T_{k,k}\varphi_{k-1}^{i,j}\right], \tag{5.8}$$

and so the recursive structure of a close to Toeplitz matrix is defined via α different extensions of each principal leading minor. Also, this case can be included in the definition of Section 2 by an obvious generalization of (2.11).

ALGORITHM 5.1. Let R be a close to Toeplitz matrix defined by (5.1). Then one can find the solution of the equation

$$R\chi = \mathbf{f} \tag{5.9}$$

in the following way.

1. Start with $\gamma_0^1(0) = \cdots = \gamma_0^\alpha(0) = 1/\sum_{i=1}^{\alpha} r_0^i s_0$ and

$$\gamma_0^1(m) = \cdots = \gamma_0^\alpha(m) = -\gamma_0(0)\sum_{i=1}^{\alpha} s_0^i r_m^i, \qquad m = 1,\ldots,N,$$

$$\gamma_0^j(-m) = -\gamma_0(0)s_m^j, \qquad m = 1,\ldots,N, \quad j = 1,\ldots,\alpha,$$

$$\varphi_0^{i,j}(0) = \gamma_0(0)r^i, \qquad i,j = 1,\ldots,\alpha,$$

$$\varphi_0^{i,j}(m) = r_m^i - \gamma_0(0)r_0^i\sum_{p=1}^{\alpha} s_0^p r_m^p, \qquad i,j = 1,\ldots,\alpha, \quad m = 1,\ldots,N,$$

$$\varphi_0^{i,j}(-m) = -\gamma_0(0)r_0^i s_m^j, \qquad i,j = 1,\ldots,\alpha, \quad m = 1,\ldots,N,$$

$$\chi_0(0) = \gamma_0(0)f(0),$$

$$\chi_0(m) = f(m) - \gamma_0(0)f(0)\sum_{p=1}^{\alpha} s_0^p r_m, \qquad m = 1,\ldots,N.$$

2. Compute recursively for $k = 1,\ldots,N$

$$\mu_k^j = \sum_{p=1}^{\alpha} \gamma_{k-1}^p(-1)\varphi_{k-1}^{p,j}(k), \qquad j = 1,\ldots,\alpha,$$

$$\gamma_k^j(0) = \frac{1}{1-\mu_k^j}\sum_{p=1}^{\alpha} \gamma_{k-1}^p(-1)\varphi_{k-1}^{p,j}(0), \qquad j = 1,\ldots,\alpha,$$

$$\gamma_k^j(k) = \frac{1}{1-\mu_k^j}\gamma_{k-1}^j(k-1),$$

$$\gamma_k^j(m) = \frac{1}{1-\mu_k^j}\left[\gamma_{k-1}^j(m-1) + \sum_{p=1}^{\alpha} \gamma_{k-1}^p(-1)\varphi_{k-1}^{p,j}(m)\right],$$

$$j = 1,\ldots,\alpha, \quad m = k-N,\ldots,N, \quad m \neq 0,k,$$

$$\varphi_k^{i,j}(k) = \varphi_{k-1}^{i,j}(k)\gamma_k^j(k), \qquad i,j = 1,\ldots,\alpha,$$

$$\varphi_k^{i,j}(m) = \varphi_{k-1}^{i,j}(m) + \varphi_{k-1}^{i,j}(k)\gamma_k^j(m),$$

$$i, j = 1, \ldots, \alpha, \quad m = k - N, \ldots, N \quad m \neq k,$$

$$\chi_k(k) = \chi_{k-1}(k)\gamma_k^1(k),$$

$$\chi_k(m) = \chi_{k-1}(m) + \chi_{k-1}(k)\gamma_k^1(m), \qquad m = 0, \ldots, N, \quad m \neq k.$$

3. The vector χ_N gives the solution of (5.9).

This algorithm can be executed on $(\alpha^2 + 2\alpha + 3)N$ parallel processors at the expense of no more than $(\alpha + 2)N$ flops per processor. Indeed, we can also write

$$\varphi_k^{i,j}(m) = \varphi_{k-1}^{i,j}(m) + \frac{1}{1-\mu_k^j}\varphi_{k-1}^{i,j}(k)\left[\gamma_{k-1}^j(m) + \sum_{i=1}^{\alpha} \gamma_{k-1}^p(-1)\varphi_{k-1}^{p,j}(m)\right],$$

so that it takes $\alpha + 1$ flops to calculate $1 - \mu_k^j$ and $\varphi_{k-1}^{i,j}(k)[\gamma_{k-1}^j(m) + \sum_{p=1}^{\alpha}\gamma_{k-1}^p(-1)\varphi_{k-1}^{p,j}(m)]$, and then one more flop to find $\varphi_k^{i,j}(m)$.

We remark that $\varphi_k^{i,j}(m) = \varphi_k^{i,p}(m)$ for $i = 1, \ldots, \alpha$, $\gamma_k^j(m) = \gamma_k^p(m)$, and $\mu_k^j = \mu_k^p$ for all $j, p = 1, \ldots, \alpha$ and $k, m = 0, \ldots, N$.

We also remark that the same algorithms for the transposed matrix R^T will give vectors $R_k^T\omega_k = e_k$. The entries $\gamma_k(m)$ and $\omega_k(m)$ can be found separately, without finding $\gamma_k(m)$ and $\omega_k(m)$ for $m = 0, \ldots, k$, thus giving the *LDU* decomposition of R.

Simpler extensions of principal leading minors of the matrix R will be sufficient if we introduce solutions for some additional right hand sides and compute simultaneously γ_k and ω_k for $k = 0, \ldots, N$. The complexity of the second algorithm will be exactly the same as the complexity of the first one, but the number of processors for the new algorithm will be $(4\alpha + 3)N$ only. First we remark that for $j = 1, \ldots, \alpha$

$$\gamma_k^i(-1) = -\sum_{j=0}^{k} \gamma_k^i(j)s_{j+1}^i.$$

Now let

$$\left(R_k^l\right)^T\left[y_k^i(0), \ldots, y_k^i(k)\right]^T = \left[s_1^i, \ldots, s_{k+1}^i\right]^T.$$

Since the last row of $(R_k^l)^{-T}$ is just $[\gamma_k(0),\dots,\gamma_k(k)]^T$, we get

$$\gamma_k^i(-1) = -y_k^i(k), \qquad i=1,\dots,\alpha.$$

Therefore we can also consider the extensions R_k with $q_0 = p_0 = 0$ for $k = 0,\dots,N$,

$$R_k = \begin{bmatrix} r_{00} & \cdots & r_{0k} & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ r_{k0} & \cdots & r_{kk} & 0 & \cdots & 0 \\ r_{k+1,0} & \cdots & r_{k+1,k} & 1 & & 0 \\ \vdots & & \vdots & & \ddots & \\ r_{N0} & \cdots & r_{Nk} & 0 & & 1 \end{bmatrix}.$$

For $k = 0,\dots,N-1$, let

$$R_k\varphi_k^i = [r_0^i,\dots,r_N^i]^T, \qquad i=1,\dots,\alpha,$$

$$R_K^T\psi_K^i = [s_0^i,\dots,s_N^i]^T, \qquad i=1,\dots,\alpha,$$

$$R_k x_k^i = [r_1^i,\dots,r_N^i,0]^T, \qquad i=1,\dots,\alpha,$$

$$R_k^T y_k^i = [s_1^i,\dots,s_N^i,0]^T, \qquad i=1,\dots,\alpha.$$

Then for $k = 1,\dots,N$

$$\gamma_k = \Gamma_k(\gamma_{k-1}, \varphi_{k-1}^1,\dots,\varphi_{k-1}^\alpha, y_{k-1}^1,\dots,y_{k-1}^\alpha),$$

$$\omega_u = W_k(\omega_{k-1}, \psi_{k-1},\dots,\psi_{k-1}^\alpha, x_{k-1}^1,\dots,x_{k-1}^\alpha),$$

where

$$\Gamma_k = \frac{1}{1+\sum_{i=1}^\alpha y_{k-1}^i(k-1)\varphi_{k-1}^i(k)}\left[T_{k,0}^0\gamma_{k-1} - \sum_{i=1}^\alpha y_{k-1}^i(k-1)T_{k,k}^0\varphi_{k-1}^i\right]$$

$$W_k = \frac{1}{1+\sum_{i=1}^\alpha x_{k-1}^i(k-1)\psi_{k-1}^i(k)}\left[T_{k,0}^0\omega_{k-1} - \sum_{i=1}^\alpha x_{k-1}^i(k-1)T_{k,k}^0\psi_{k-1}^i\right],$$

where now $T^0_{k,0}$ and $T^0_{k,k}$ map $\mathbf{C}^{N+1}$ into itself according to the rule

$$T^0_{k,0}[f(0),\ldots,f(N)]^T = [0, f(0),\ldots,f(N-1)]^T,$$

and

$$T^0_{k,k}[f(0),\ldots,f(N)]^T = [f(0),\ldots,f(k-1),0,f(k+1),\ldots,f(N)]^T.$$

ALGORITHM 5.2. Under the conditions of Algorithm 5.1:

1. Start with

$$\gamma_0(0) = \omega_0(0) = \frac{1}{\sum_{i=1}^{\alpha} r_0^i s_0^i},$$

$$\gamma_0(j) = -\gamma_0(0)\sum_{i=1}^{\alpha} s_0^i r_j^i, \qquad j = 1,\ldots,N,$$

$$\omega_0(j) = -\omega_0(0)\sum_{i=1}^{\alpha} r_0^i s_j^i, \qquad j = 1,\ldots,N,$$

$$\varphi_0^i(0) = \gamma_0(0) r_0^i, \quad x_0^1(0) = \gamma_0(0) r_1^i, \qquad i = 1,\ldots,\alpha,$$

$$\varphi_0^i(j) = r_j^i - \gamma_0(0) r_0^i \sum_{p=1}^{\alpha} s_0^p r_j^p, \qquad j = 1,\ldots,N, \quad i = 1,\ldots,\alpha,$$

$$x_0^i(j) = r_{j+1}^i - \gamma_0(0) r_1^i \sum_{p=1}^{\alpha} s_0^p r_j^p, \qquad j = 1,\ldots,N-1, \quad i = 1,\ldots,\alpha,$$

$$\psi_0^i(0) = \omega_0(0) s_0^i, \quad y_0^i(0) = \omega_0(0) s_1^i, \qquad i = 1,\ldots,\alpha,$$

$$\psi_0^i(j) = s_j^i - \omega_0(0) s_0^i \sum_{p=1}^{\alpha} r_0^p s_j^p, \qquad j = 1,\ldots,N, \quad i = 1,\ldots,\alpha,$$

$$y_0^i(j) = s_{j+1}^i - \omega_0(0) s_1^i \sum_{p=1}^{\alpha} r_0^p s_j^p, \qquad j = 1,\ldots,N-1, \quad i = 1,\ldots,\alpha,$$

$$\chi_0(0) = \gamma_0(0) f(0)$$

$$\chi_0(j) = f(j) - \gamma_0(0) f(0) \sum_{p=1}^{\alpha} s_0^p r_j^p, \qquad j = 1,\ldots,N.$$

2. Compute recursively for $k = 1, \ldots, N-1$

$$\mu_k = \sum_{p=1}^{\alpha} y_{k-1}^p(k-1)\varphi_{k-1}^p(k),$$

$$\nu_k = \sum_{p=1}^{\alpha} x_{k-1}^p(k-1)\psi_{k-1}^p(k),$$

$$\gamma_k(0) = -\frac{1}{1+\mu_k}\sum_{p=1}^{\alpha} y_{k-1}^p(k-1)\varphi_{k-1}^p(0),$$

$$\gamma_k(k) = \frac{1}{1+\mu_k}\gamma_{k-1}(k-1),$$

$$\gamma_k(j) = \frac{1}{1+\mu_k}\left[\gamma_{k-1}(j-1) - \sum_{p=1}^{\alpha} y_{k-1}^p(k-1)\varphi_{k-1}^p(j)\right],$$

$$j = 1, \ldots, N, \quad j \neq k,$$

$$\omega_k(0) = -\frac{1}{1+\nu_k}\sum_{p=1}^{\alpha} x_{k-1}^p(k-1)\psi_{k-1}^p(k),$$

$$\omega_k(k) = \frac{1}{1+\nu_k}\omega_{k-1}(k-1) = \gamma_k(k),$$

$$\omega_k(j) = \frac{1}{1+\nu_k}\left[\omega_{k-1}(j-1) - \sum_{p=1}^{\alpha} x_{k-1}^p(k-1)\psi_{k-1}^p(j)\right],$$

$$j = 1, \ldots, N, \quad j \neq k,$$

$$\varphi_k^i(k) = \varphi_{k-1}^i(k)\gamma_k(k), \quad x_k^i(k) = x_{k-1}^i(k)\gamma_k(k), \qquad i = 1, \ldots, \alpha,$$

$$\varphi_k^i(j) = \varphi_{k-1}^i(j) + \varphi_{k-1}^i(k)\gamma_k(j), \qquad i = 1, \ldots, \alpha, \quad j = 0, \ldots, N, \quad j \neq k,$$

$$x_k^i(j) = x_{k-1}^i(j) + x_{k-1}^i(k)\gamma_k(j), \qquad i = 1, \ldots, \alpha, \quad j = 1, \ldots, N, \quad j \neq k,$$

$$\psi_k^i(k) = \psi_{k-1}^i(k)w_k(k), \quad y_k^i(k) = y_{k-1}^i(k)w_k(k), \qquad i = 1, \ldots, \alpha,$$

$$\psi_k^i(j) = \psi_{k-1}^i(j) + \psi_{k-1}^i(k)w_k(j), \qquad i = 1, \ldots, \alpha, \quad j = 0, \ldots, N, \quad j \neq k,$$

$$y_k^i(j) = y_{k-1}^i(j) + y_{k-1}^i(k)w_k(j), \qquad i = 1, \ldots, \alpha, \quad j = 0, \ldots, N, \quad j \neq k,$$

$$\chi_k(k) = \chi_{k-1}(k)\gamma_k(k),$$

$$\chi_k(j) = \chi_{k-1}(j) + \chi_{k-1}(k)\gamma_k(j), \qquad j = 0, \ldots, N, \quad j \neq k.$$

3. Compute

$$\mu_N = \sum_{p=1}^{\alpha} y_{N-1}^p(N-1)\varphi_{N-1}^p(N),$$

$$\gamma_N(0) = -\frac{1}{1+\mu_N}\sum_{p=1}^{\alpha} y_{N-1}^p(N-1)\varphi_{N-1}^p(0),$$

$$\gamma_N(N) = \frac{1}{1+\mu_N}\gamma_{N-1}(N-1),$$

$$\gamma_N(j) = \frac{1}{1+\mu_N}\left[\gamma_{N-1}(j-1) - \sum_{p=1}^{\alpha} y_{N-1}^p(N-1)\varphi_{N-1}^p(j)\right],$$

$$j = 1,\ldots,N-1,$$

$$\chi_N(N) = \chi_{N-1}(N)\gamma_N(N),$$

$$\chi_N(j) = \chi_{N-1}(j) + \chi_{N-1}(N)\gamma_N(j), \qquad j = 0,\ldots,N-1.$$

4. The vector χ_N gives the solution of (5.9).

6. HILBERT-TYPE MATRICES

Matrices of Hilbert type are defined by the relation

$$\operatorname{diag}\{t_0,\ldots,t_N\}R - R\operatorname{diag}\{s_0,\ldots,s_N\} = \sum_{i=1}^{\alpha} \mathbf{g}_i\mathbf{h}_i^T \tag{6.1}$$

(see [11], [7]), where $\{t_j\}_{j=0}^N$ and $\{s_j\}_{j=0}^N$ are two disjoint sets of pairwise different numbers. In this case we consider

$$R_k = \begin{bmatrix} R_k^l & 0 \\ R_k^c & I \end{bmatrix}, \tag{6.2}$$

thus setting $p_k = q_k = 0$ for $k = 0, \dots, N$. It follows from (6.1) and (6.2) that

$$\operatorname{diag}\{t_0, \dots, t_N\} R_k - R_k \operatorname{diag}\{s_0, \dots, s_N\}$$

$$= \sum_{i=1}^{\alpha} \mathbf{g}_i (P_k \mathbf{h}_i)^T + \operatorname{diag}\{0, \dots, 0, t_{k+1} - s_{k+1}, \dots, t_N - s_N\}, \quad (6.3)$$

where P_k is the projection on the first $k+1$ coordinates,

$$P_k [f(0), \dots, f(N)]^T = [f(0), \dots, f(k), 0, \dots, 0]^T.$$

Applying R_k^{-1} to both sides of (6.3), we get

$$R_k^{-1} \operatorname{diag}\{t_0, \dots, t_N\} - \operatorname{diag}\{s_0, \dots, s_N\} R_K^{-1}$$

$$= \sum_{i=1}^{\alpha} \varphi_k^i (P_k \psi_k^i)^T + \operatorname{diag}\{0, \dots, 0, t_{k+1} - s_{k+1}, \dots, t_N - s_N\} R_k^{-1},$$

where φ_k^i and ψ_k^i are solutions of the equations

$$R_k \varphi_k^i = \mathbf{g}_i \quad \text{and} \quad R_k^T \psi_k^i = \mathbf{h}_i.$$

Therefore

$$\operatorname{diag}\{t_k - s_0, \dots, t_k - s_k, t_k - t_{k+1}, \dots, t_k - t_N\} \gamma_k = \sum_{i=1}^{\alpha} \psi_k^i(k) \varphi_k^i.$$

Similarly

$$\operatorname{diag}\{t_0 - s_k, \dots, t_k - s_k, s_{k+1} - s_k, \dots, s_N - s_k\} \omega_k = \sum_{i=1}^{\alpha} \varphi_k^i(k) \psi_k^i.$$

Using (2.5) and (2.6), we get

$$\gamma_k = \Gamma_k(\varphi_{k-1}^1, \dots, \varphi_{k-1}^{\alpha}, \psi_{k-1}^1, \dots, \psi_{k-1}^{\alpha}),$$

$$\omega_k = W_k(\varphi_{k-1}^1, \dots, \varphi_{k-1}^{\alpha}, \psi_{k-1}^1, \dots, \psi_{k-1}^{\alpha}),$$

where

$$\Gamma_k = \frac{t_k - s_k}{\sum_{i=1}^{\alpha} \psi_{k-1}^i(k)\varphi_{k-1}^i(k)} \times \left[\sum_{i=1}^{\alpha} \psi_{k-1}^i(k) \operatorname{diag}\left\{ (s_k - s_0)^{-1}, \ldots, (s_k - s_{k-1})^{-1}, 0, \times (s_k - t_{k+1})^{-1}, \ldots, (s_k - t_k)^{-1} \right\} \varphi_{k-1}^i + \mathbf{e}_k \right]$$

and

$$W_k = \frac{t_k - s_k}{\sum_{i=1}^{\alpha} \psi_{k-1}^i(k)\varphi_{k-1}^i(k)} \times \left[\sum_{i=1}^{\alpha} \varphi_{k-1}^i \operatorname{diag}\left\{ (t_0 - t_k)^{-1}, \ldots, (t_{k-1} - t_k)^{-1}, 0, \times (s_{k+1} - t_k)^{-1}, \ldots, (s_N - t_k)^{-1} \right\} \psi_{k-1}^i + \mathbf{e}_k \right].$$

ALGORITHM 6.1. Let R be a matrix of Hilbert type as defined in (6.1). Then the equation

$$R\chi = \mathbf{f} \tag{6.4}$$

can be solved in the following way.

1. Start with $\gamma_0(0) = \omega_0(0) = 1/r_{00}$ and

$$\gamma_0(j) = -\frac{r_{j0}}{r_{00}}, \quad \omega_0(j) = -\frac{r_{0j}}{r_{00}}, \qquad j = 1, \ldots, N,$$

$$\varphi_0^i(0) = \frac{g_i(0)}{r_{00}}, \quad \psi_0^i(0) = \frac{h_i(0)}{r_{00}}, \quad i = 1, \ldots, \alpha, \qquad \chi_0(0) = \frac{f(0)}{r_{00}},$$

$$\varphi_0^i(j) = g_i(j) - \frac{g_i(0)r_{j0}}{r_{00}}, \quad \psi_0^i(j) = h_i(j) - \frac{h_i(0)r_{0j}}{r_{00}}, \qquad i = 1, \ldots, \alpha, \quad j = 1, \ldots, N,$$

$$\chi_0(j) = f(j) - f(0)\frac{r_{0j}}{r_{00}}, \qquad j = 1, \ldots, N.$$

2. Compute recursively for $k = 1, \dots, N$

$$\gamma_k(k) = \omega_k(k) = \frac{t_k - s_k}{\sum_{i=1}^{\alpha} \psi_{k-1}^i(k)\varphi_{k-1}^i(k)}, \tag{6.5}$$

$$\gamma_k(j) = \gamma_k(k) \frac{\sum_{i=1}^{\alpha} \psi_{k-1}^i(k)\varphi_{k-1}^i(j)}{s_k - s_j}, \qquad j = 0, \dots, k-1,$$

$$\gamma_k(j) = \gamma_k(k) \frac{\sum_{i=1}^{\alpha} \psi_{k-1}^i(k)\varphi_{k-1}^i(j)}{s_k - t_j}, \qquad j = k+1, \dots, N, \tag{6.6}$$

$$\omega_k(j) = \omega_k(k) \frac{\sum_{i=1}^{\alpha} \varphi_{k-1}^i(k)\psi_{k-1}^i(j)}{t_j - t_k}, \qquad j = 0, \dots, k-1,$$

$$\omega_k(j) = \omega_k(k) \frac{\sum_{i=1}^{\alpha} \varphi_{k-1}^i(k)\psi_{k-1}^i(j)}{s_j - t_k}, \qquad j = k+1, \dots, N, \tag{6.7}$$

$$\varphi_k^i(k) = \varphi_{k-1}^i(k)\gamma_k(k), \quad \psi_k^i(k) = \psi_{k-1}^i(k)\omega_k(k), \qquad i = 1, \dots, \alpha, \tag{6.8}$$

$$\varphi_k^i(j) = \varphi_{k-1}^i(j) + \varphi_{k-1}^i(k)\gamma_k(j), \qquad j = 0, \dots, N, \quad i = 1, \dots, \alpha, \quad j \neq k, \tag{6.9}$$

$$\psi_k^i(j) = \psi_{k-1}^i(j) + \psi_{k-1}^i(k)\omega_k(j), \qquad j = 0, \dots, N, \quad i = 1, \dots, \alpha, \quad j \neq k,$$

$$\chi_k(k) = \chi_{k-1}(k)\gamma_k(k),$$

$$\chi_k(j) = \chi_{k-1}(j) + \chi_{k-1}(k)\gamma_k(j), \qquad j = 0, \dots, N, \quad j \neq k.$$

3. The vector χ_N gives the solution of (6.4).

This algorithm can be executed on parallel processors at the expense of no more than $2\alpha N$ flops per processor.

We also remark that the *LDU* decomposition of R can be computed separately, via (6.5)–(6.9) for $j = k+1, \dots, N$.

7. VANDERMONDE-TYPE MATRICES

Let $R = \{r_i^j\}_{i,j=0}^N$, $r_i \neq 0$, be a Vandermonde matrix. Since $\det R = \prod_{0 \leq k < i \leq N}(r_i - r_k)$, R is strongly regular if and only if $r_i \neq r_k$ for $i \neq k$. We first describe a parallel algorithm for the transposed system

$$R^T \chi = \mathbf{f}.$$

The extended minors R_k are defined as in (6.2), setting $p_k = q_k = 0$. Let $R_k\omega_k = \mathbf{e}_k$ and $R_k\psi_k = [1/r_0, 1/r_1, \ldots, 1/r_N]^T$. Then it is easy to show that the following algorithm holds (see the generalization below).

ALGORITHM 7.1. Let $R = \{r_i^j\}_{i,j=0}^N$ be a strongly regular Vandermonde matrix. Then the solution of the equation

$$R^T\chi = \mathbf{f} \tag{7.1}$$

can be found in the following way.

1. Start with

$$\omega_0 = [1, -1, \ldots, -1]^T, \qquad \psi_0 = \left[\frac{1}{r_0}, \frac{1}{r_1} - \frac{1}{r_0}, \ldots, \frac{1}{r_N} - \frac{1}{r_0}\right]^T,$$

$$\chi_0 = [f_0, f_1 - f_0, \ldots, f_N - f_0]^T.$$

2. Compute recursively for $k = 1, \ldots, N$

$$\begin{aligned}
\omega_k(0) &= \frac{1}{r_k\psi_{k-1}(k)}, \\
\omega_k(j) &= -\frac{\psi_{k-1}(j-1)}{r_k\psi_{k-1}(k)}, \qquad j = 1, \ldots, k, \\
\omega_k(j) &= -\frac{r_j\psi_{k-1}(j)}{r_k\psi_{k-1}(k)}, \qquad j = k+1, \ldots, N, \\
\psi_k(0) &= \psi_{k-1}(0) + \frac{1}{r_k}, \\
\psi_k(j) &= \psi_{k-1}(j) - \frac{\psi_{k-1}(j-1)}{r_k}, \qquad j = 1, \ldots, k-1, \\
\psi_k(k) &= -\frac{\psi_{k-1}(k-1)}{r_k}, \\
\psi_k(j) &= \left(1 - \frac{r_j}{r_k}\right)\psi_{k-1}(j), \qquad j = k+1, \ldots, N, \\
\chi_k(j) &= \chi_{k-1}(j) + \chi_{k-1}(k)\omega_k(j), \qquad j = 0, \ldots, N, \quad j \neq k, \\
\chi_k(k) &= \chi_{k-1}(k)\omega_k(k).
\end{aligned} \tag{7.2}$$

3. The last vector χ_N is the solution of (7.1).

Note that the entries of χ_k are computed with one delay after the entries of ω_k are found. Therefore this algorithm can be executed on $4N+3$ parallel processors at the expense of no more than $2N+1$ flops per processor. Indeed, in (7.2) we can compute $r_j\psi_{k-1}(j)$ and $r_k\psi_{k-1}(k)$ simultaneously and then divide them.

Let us consider now the Vandermonde system

$$R\chi = \mathbf{f},$$

and let R_k^l denote the principal leading minors of R. It is easy to see that

$$\lambda_k\varphi_k = \operatorname{diag}\{r_0^{-1},\dots,r_k^{-1}\}\,\gamma_k$$

where $R_k^l\varphi_k = \mathbf{e}_0$, $R_k^l\gamma_k = \mathbf{e}_k$ with $\mathbf{e}_0, \mathbf{e}_k \in \mathbb{C}^{k+1}$, and

$$\lambda_k = \sum_{j=0}^{k} \frac{\gamma_k(j)}{r_j}.$$

Since $\varphi_k(k) = \omega_k(0)$ and $\gamma_k(k) = \omega_k(k)$, we find from (2.5) and Algorithm 7.1 that

$$\varphi_{k-1}(k) = -\frac{1}{\psi_{k-1}(k-1)}, \qquad \lambda_k = -\frac{\psi_{k-1}(k-1)}{r_k}$$

and that

$$\gamma_k(j) = \frac{\gamma_{k-1}(j)}{r_j - r_k}, \qquad j = 0,\dots,k-1.$$

We also remark that

$$\chi_k(k) = \left\langle \left(R_k^{-1}\right)^l \mathbf{f}_k, \mathbf{e}_k \right\rangle = \sum_{j=0}^{k} \omega_k(j) f(j).$$

Thus we get the following algorithm for Vandermonde systems.

ALGORITHM 7.2. Let $R = \{r_i^j\}_{i,j=0}^N$ be a strongly regular Vandermonde matrix. Then the solution of the equation

$$R\chi = \mathbf{f} \tag{7.3}$$

can be found in the following way.

1. Start with ω_0 and ψ_0 defined in Algorithm 7.1 and $\gamma_0(0) = 1$, $\chi_0(0) = f(0)$.
2. Compute recursively for $k = 1, \ldots, N$

$$\omega_k \text{ and } \psi_k$$

via Algorithm 7.1, and

$$\gamma_k(j) = \frac{\gamma_{k-1}(j)}{r_j - r_k}, \qquad j = 0, \ldots, k-1,$$

$$\gamma_k(k) = \omega_k(k).$$

3. Compute simultaneously for $k = 1, \ldots, N$

$$\chi_k(j) = \chi_{k-1}(j) + \frac{\chi_k(k)\gamma_k(j)}{\gamma_k(k)}, \qquad j = 0, \ldots, k-1,$$

$$\chi_k(k) = \sum_{j=0}^{k} \omega_k(j) f(j).$$

4. The last vector χ_N gives the solution of (7.3).

The second step can be executed on $4N+3$ parallel processors at the expense of no more than $2N$ flops per processor. This step will give us the *UDL* decomposition of R^{-1}, which is determined by $[\gamma_k(0), \ldots, \gamma_k(k)]^T$ and $[\omega_k(0), \ldots, \omega_k(k)]^T$ for $k = 1, \ldots, N$ via (2.18). The third step can then be executed on $N+1$ parallel processors at the expense of $2N$ flops per processor also.

Indeed, inner products $\Sigma_{j=0}^k \omega_k(j) f(j)$ for $j = 1, \ldots, N$ can be accumulated simultaneously, giving $\Sigma_{j=1}^i \omega_k(j) f(j)$ and $\chi_k(k) = \Sigma_{j=1}^k \omega_k(j) f(j)$ at step k of the recursion.

Note that if $\mathbf{f} = \mathbf{e}_0$, then we can omit the third step and compute φ_k via

$$\varphi_k(j) = -\frac{r_k\gamma_k(j)}{r_j\psi_{k-1}(k-1)} = -\frac{r_k\gamma_{k-1}(j)}{r_j(r_j-r_k)\psi_{k-1}(k-1)}, \qquad j = 0,\dots,k-1,$$

$$\varphi_k(k) = \frac{1}{r_k\psi_{k-1}(k)}.$$

The Vandermonde matrices belong to a wider class of Vandermonde-type matrices, defined by the relation

$$R\,\mathrm{diag}\{t_0,\dots,t_N\} - ZR = \sum_{i=1}^{\alpha} \mathbf{g}_i\mathbf{h}_i^T \tag{7.4}$$

(see [8], [4]), where

$$Z = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix}$$

and $t_j \neq 0$, $j = 0,\dots,N$. It is easy to see that the unique solution of (7.4) is given by

$$R = \sum_{i=1}^{\alpha} \begin{bmatrix} \mathbf{g}_i & Z\mathbf{g}_i & \cdots & Z^{N-1}\mathbf{g}_i \end{bmatrix} \begin{bmatrix} \mathbf{h}_i^T D^{-1} \\ \mathbf{h}_i^T D^{-2} \\ \vdots \\ \mathbf{h}_i^T D^{-N-1} \end{bmatrix}.$$

Here $D = \mathrm{diag}\{t_0,\dots,t_N\}$, and we use the fact that $Z^N = 0$. In other words,

$$R = \sum_{i=1}^{\alpha} L[\mathbf{g}_i] \begin{bmatrix} \dfrac{h_i(0)}{t_0} & \cdots & \dfrac{h_i(N)}{t_N} \\ \vdots & & \vdots \\ \dfrac{h_i(0)}{t_0^{N+1}} & \cdots & \dfrac{h_i(N)}{t_0^{N+1}} \end{bmatrix}, \tag{7.5}$$

where

$$L[\mathbf{g}_i] = \begin{bmatrix} g^i(0) & & 0 \\ \vdots & \ddots & \\ g^i(N) & \cdots & g^i(0) \end{bmatrix}.$$

Let R_k be given by (6.2) as for Hilbert-type matrices. Then, it is easy to see that

$$R_k D_k \gamma_{k-1} = e_k + \sum_{i=1}^{\alpha} \left[\sum_{j=0}^{k-1} h_i(j) \gamma_{k-1}(j) \right] \mathbf{g}_i,$$

where $D_k[f(0),\ldots,f(N)]^T = [t_0 f(0),\ldots,t_{k-1} f(k-1), 0, f(k)\ldots f(N-1)]^T$. Introducing solutions of the equations $R_k \varphi_k^i = \mathbf{g}_i$, $i = 1,\ldots,\alpha$, we get

$$\gamma_k = D_k \gamma_{k-1} - \sum_{i=1}^{\alpha} \left[\sum_{j=0}^{k-1} h_i(j) \gamma_{k-1}(j) \right] \varphi_k^i. \tag{7.6}$$

For the transposed matrix we have

$$R_k^T S_k \omega_k = t_k \mathbf{e}_k - \sum_{i=1}^{\alpha} \varphi_k^i(k) \mathbf{h}_i,$$

where $S_k \omega_k = [\omega_k(1),\ldots,\omega_k(k), 0, t_{k+1}\omega_k(k+1),\ldots,t_N \omega_k(N)]^T$. Thus, introducing solutions of the equations $R_k^T \psi^i = \mathbf{h}_i$, $i = 1,\ldots,\alpha$, we find that

$$S_k \omega_k = t_k \omega_k - \sum_{i=1}^{\alpha} \varphi_k^i(k) \psi^i. \tag{7.7}$$

It is shown in [4] that $\gamma_k(k) = t_k / \sum_{i=1}^{\alpha} \varphi_{k-1}^i(k) \psi_{k-1}^i(k)$. Since

$$\sum_{j=0}^{k-1} h_i(j) \gamma_{k-1}(j) = \psi_{k-1}^i(k-1) \quad \text{for} \quad i = 1,\ldots,\alpha,$$

it follows from (2.5), (2.6), (7.6), and (7.7) that

$$\gamma_k = \Gamma_k\left(\varphi_{k-1}^1,\ldots,\varphi_{k-1}^{\alpha}, \psi_{k-1}^1,\ldots,\psi_{k-1}^{\alpha}\right),$$

$$\omega_k = W_k\left(\varphi_{k-1}^1,\ldots,\varphi_{k-1}^{\alpha}, \psi_{k-1}^1,\ldots,\psi_{k-1}^{\alpha}\right),$$

where, assuming that $\mathbf{g}_1 = \mathbf{e}_0$,

$$\Gamma_k = \frac{t_k}{\lambda_k} \operatorname{diag}\left\{(t_0 - t_k)^{-1}, \ldots, (t_{k-1} - t_k)^{-1}, \lambda_k^{-1}, \tau_k\right\}$$

$$\times \left(\sum_{i=1}^{\alpha} \psi_{k-1}^i(k) \varphi_{k-1}^i + \mathbf{e}_{k+1} \right)$$

with

$$\lambda_k = \sum_{i=1}^{\alpha} \varphi_{k-1}^i(k) \psi_{k-1}^i(k) \quad \text{and} \quad \tau_k = (Z - t_k I)^{-1},$$

and

$$W_k = \frac{t_k}{\lambda_k} \begin{bmatrix} \mu_k & & & & & & & \\ -1 & 0 & & & & & & \\ & \ddots & \ddots & & & & 0 & \\ & & -1 & 0 & & & & \\ & & & 0 & \lambda_k^{-1} & & & \\ & & & & 0 & -t_{k+1}^{-1} & & \\ & 0 & & & & \ddots & \ddots & \\ & & & & & & 0 & -t_N^{-1} \end{bmatrix}$$

$$\times \sum_{i=1}^{\alpha} \varphi_{k-1}^i(k) \psi_{k-1}^i,$$

with $\mu_k = \varphi_{k-1}^1(k) / \sum_{i=1}^{\alpha} \varphi_{k-1}^i(k) \psi_{k-1}^i(0)$. The assumption that $\mathbf{g}_1 = \mathbf{e}_0$ does not imply any loss of generality [we can always add $\mathbf{e}_0 \mathbf{0}^T$ to the right-hand side in (7.4)] and is immediately satisfied when R is Vandermonde.

ALGORITHM 7.3. Let R be a Vandermonde-type matrix defined in (7.4). Then the equation

$$R\chi = \mathbf{f} \tag{7.8}$$

can be solved in the following way.

1. Start with $\gamma_0 = \omega_0$, χ_0, φ_0^i, ψ_0^i, $i = 1,\dots,\alpha$, given by (2.13)–(2.16).
2. Compute recursively for $k = 1,\dots,N$

$$\gamma_k(k) = \omega_k(k) = \frac{t_k}{\sum_{i=1}^{\alpha} \varphi_{k-1}^i(k)\psi_{k-1}^i(k)},$$

$$\gamma_k(j) = \frac{\gamma_k(k)}{t_j - t_k} \sum_{i=1}^{\alpha} \psi_{k-1}^i(k)\varphi_{k-1}^i(j), \qquad j = 0,\dots,k-1,$$

$$(Z - t_k I)\begin{bmatrix} \gamma_k(k+1) \\ \vdots \\ \gamma_k(N) \end{bmatrix} = \gamma_k(k) \sum_{i=1}^{\alpha} \psi_{k-1}^i(k) \begin{bmatrix} \varphi_{k-1}^i(k+1) \\ \vdots \\ \varphi_{k-1}^i(N) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{7.9}$$

$$\omega_k(j) = -\omega_k(k) \sum_{i=1}^{\alpha} \varphi_{k-1}^i(k)\psi_{k-1}^i(j-1), \qquad j = 1,\dots,k-1,$$

$$\omega_k(j) = -\frac{1}{t_j}\left[\omega_k(k) \sum_{i=1}^{\alpha} \varphi_{k-1}^i(k)\psi_{k-1}^i(j)\right], \qquad j = k+1,\dots,N.$$

φ_k^i, $i = 1,\dots,\alpha$, and χ are computed via (2.5), and

$$\omega_k(0) = \varphi_k^1(k). \tag{7.10}$$

ψ_k^i, $i = 1,\dots,\alpha$, are computed via (2.6).

Note that the back substitution in (7.9) is not parallel. The equation (7.9) can be solved by FFT method in $O(\log k)$ flops. Therefore the whole algorithm can be executed in $O(N \log N)$ flops on parallel processors.

8. THE NON-STRONGLY-REGULAR CASE

If the matrix R is nonsingular but not strongly regular and belongs to one of the classes discussed in Sections 3–7, we can replace the equation $R\chi = \mathbf{f}$ by the equation $R^*R\chi = R^*\mathbf{f}$. It is shown in [7] that if R is Toeplitz, Hankel,

or close to Toeplitz, then R^*R is close to Toeplitz, and if R is of Hilbert or Vandermonde type, then R^*R is of a Hilbert type. Thus Algorithm 4.1 or 6.1 can be applied to R^*R. For example, if R is Toeplitz, then denoting the complex conjugate with an overbar, we have

$$R^*R = L(\mathbf{g}) + U(\mathbf{h}) + L(\mathbf{r}_1)U(\mathbf{s}_1) - L(\mathbf{r}_2)U(\mathbf{s}_2),$$

where $\mathbf{g} = R^*[r_0, \ldots, r_N]^T$, $\mathbf{h} = [0, h_1, \ldots, h_N]^T$ with

$$[h_1, \ldots, h_N]^T = \left(R^l_{N\ 1}\right)^*[\bar{r}_1, \ldots, \bar{r}_N]^T, \qquad \mathbf{r}_1 = [\bar{r}_0, \ldots, \bar{r}_{N}]^T,$$

$$\mathbf{s}_1 = [0, r_{-1}, \ldots, r_{-N}]^T, \qquad \mathbf{r}_2 = [0, \bar{r}_N, \ldots, \bar{r}_1]^T, \qquad \mathbf{s}_2 = [0, r_N, \ldots, r_1]^T.$$

We remark that the computation of initial data for Algorithm 4.1 here requires $O(N)$ flops on $2N+1$ parallel processors.

As mentioned in the introduction, this method means a tradeoff between squaring the conditioning of the original system and the improved complexity of computation offered by our algorithms.

9. NUMERICAL EXAMPLES

In this section we compare results of numerical experiments for the algorithms derived in Sections 3–7 with the most widely used Gaussian elimination with partial pivoting [17, p. 93] and the QR algorithm with column pivoting with its guaranteed numerical stability [17, p. 111].

As test matrices we choose matrices for which the conditioning of each principal leading minor is not worse than the conditioning of the matrix R, and such that R is ill conditioned. One of the best known examples of this kind is the Hilbert matrix $R = \{1/(i+j+1)\}_{i,j=0}^{N}$. This matrix can be used for the testing of Algorithms 3.1, 4.1, 4.2, 5.1, 5.2, and 6.1. Indeed, the Hilbert matrix is also of a Hankel type; thus RJ [where J is defined in (3.3)] is Toeplitz and hence also close to Toeplitz.

For each of comparison we present only the computed (N, N) entry of R^{-1} as a function of the size N and of the algorithm. The results appear in Table 1. The exact value given by the formula

$$(R^{-1})_{N,N} = \frac{[(2N+1)!]^2}{(2N+1)[N!]^4}$$

TABLE 1

HILBERT MATRICES OF SIZES 5 TO 10

	$(R^{-1})_{N,N}$					
Algorithm	$N=5$	6	7	8	9	10
Gauss	44086	693986	9769178	39547369	−161450760	33125049
QR	44086	696342	10399143	39834837	−151806052	40498954
3.1	44085	694605	8768008	39629315	−400982392	−30626402
4.1	44089	693773	9344511	32833552	−78562479	33403400
4.2	44088	693891	9635818	36628650	−108446517	28262606
5.1	44082	694423	8607589	47961848	−476352000	15983355
5.2	44082	694654	8555547	48135835	−457933028	−33041484
6.1	44100	698544	11099084	176679360	2815826112	44914172416
Exact	44100	698544	11099088	176679360	2815827300	44914183600

is shown in the last row of the table. The 2-norm and ∞-norm condition numbers of R are proportional to $e^{3.5(N+1)}$ (see [10], for example).

It is immediately clear that, for these examples, the accumulation of roundoff error in the parallel algorithms is no worse than that in the two standard algorithms.

We note also the surprising numerical stability of Algorithm 6.1 as applied to the Hilbert matrix. It is plausible that this could be explained using the connection with recursion formulas for orthogonal polynomials. But the following discussion suggests that this is not the case.

It is clear that the vectors $[\gamma_k(0),\dots,\gamma_k(k)]^T$, $k=0,\dots,N$ which correspond to the Hilbert matrix $R=\{1/(i+1+j)\}_{i,j=0}^{N}$ are composed of coefficients of Legendre polynomials; thus for $j=0,\dots,k$

$$\int_0^1 x^j\left[\gamma_k(0)+\cdots+\gamma_k(k)x^k\right]dx=\begin{cases}0 & \text{for } j\neq k,\\ 1 & \text{for } j=k.\end{cases}$$

Then the vectors $[\gamma_k(0),\dots,\gamma_k(k)]^T$, $k=0,\dots,N$, can be computed via the Lanczos recursion [15]:

$$\begin{bmatrix}\gamma_k(0)\\ \vdots\\ \gamma_k(k)\end{bmatrix}=\frac{2(4k^2-1)}{k^2}\left(2\begin{bmatrix}0\\ \gamma_{k-1}(0)\\ \vdots\\ \gamma_{k-1}(k-1)\end{bmatrix}-\begin{bmatrix}\gamma_{k-1}(0)\\ \vdots\\ \gamma_{k-1}(k-1)\\ 0\end{bmatrix}-2\begin{bmatrix}\gamma_{k-2}(0)\\ \vdots\\ \gamma_{k-2}(k-2)\\ 0\\ 0\end{bmatrix}\right). \tag{9.1}$$

The value $\gamma_{10}(10)$ can be found from (9.1), and it turns out to be

$$44914184192.$$

We remark that the roundoff error accumulation in Algorithm 6.1 is comparable with that in the recursion (9.1). We also remark that this recursion is in fact equivalent to the Lanczos-type Algorithm 4.1. The only difference is that the coefficient $2(4k^2-1)/k^2$ of (9.1) is computed recursively in Algorithm 4.1, rather than being known exactly. This apparently causes the faster roundoff error accumulation shown in Table 1. Reasons for the unusual numerical stability of Algorithm 6.1 for the Hilbert matrix will be studied elsewhere.

For testing Algorithms 7.1, 7.2, and 7.3 we choose a sequence of ill-conditioned Vandermonde matrices $R_N = \{r_i^j\}_{i,j=0}^{N}$, $N = 3,4,5$, defined by $r_i = 1/10j$, $j = 0,1,2,3,4$, such that

$$R_5 = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \frac{1}{100} & \frac{1}{101} & \cdots & \frac{1}{104} \\ \vdots & \vdots & & \vdots \\ \left(\frac{1}{100}\right)^4 & \left(\frac{1}{101}\right)^4 & \cdots & \left(\frac{1}{104}\right)^4 \end{bmatrix}.$$

The computed (N, N) entry of R_N^{-1} for $N = 3,4,5$ is presented in Table 2 as a function of the matrix size and the algorithm. The exact entry, given by the formula

$$\left(R_N^{-1}\right)_{N,N} = \frac{1}{\prod_{j=0}^{N-1}(r_N - r_j)}$$

(see [7] for example), is shown in the last row of the table.

TABLE 2

VANDERMONDE MATRICES OF SIZES 3, 4, AND 5

	$(R_N^{-1})_{N,N}$		
Algorithm	$N=3$	4	5
Gauss	52542246	−187236501504	426188854001664
QR	52549842	−188273483776	263105164607488
7.1	52540199	−187621146624	517226729832448
7.2	52540199	−187621146624	517226729832448
7.3	52540204	−187621265408	517226985684992
Exact	52540200	−187621225900	517226640486400

Again we note the surprising evidence of numerical stability for Algorithms 7.1–7.3. This supports similar evidence reported in Bjork and Pereyra's paper [3].

In Tables 3 to 5 we compare the parallel algorithms for Toeplitz and close to Toeplitz matrices with the Levinson algorithm, which involves the accumulation of inner products. We choose a version of the Levinson algorithm with the normalization of Algorithm 3.1, described in the introduction. As a test matrix we choose the 10×10 combinatorial matrix

$$R(x) = \{ y + \delta_{ij} x \}_{i,j=0}^{9}, \quad \delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$

TABLE 3

THE COMBINATORIAL MATRIX WITH $x = 10^{-6}$

Levinson	Alg. 3.1	Alg. 5.1	Alg. 5.2	Exact
888938	901461	901461	901462	900000
− 111333	− 100162	− 100162	− 100162	− 100000
− 109144	·	·	·	·
− 10164	·	·	·	·
− 94434	·	·	·	·
− 94846	·	·	·	·
− 87162	·	·	·	·
− 102435	·	·	·	·
− 90749	·	·	·	·
− 97227	− 100162	− 100162	− 100162	− 100000

TABLE 4

THE COMBINATORIAL MATRIX WITH $x = 10^{-7}$

Levinson	Alg. 3.1	Alg. 5.1	Alg. 5.2	Exact
8007973	8628281	8628281	8628282	9000000
− 1564056	− 958697	− 958697	− 958698	− 1000000
− 1413272	·	·	·	·
− 1116907	·	·	·	·
− 507633	·	·	·	·
− 877477	·	·	·	·
− 525685	·	·	·	·
− 704772	·	·	·	·
− 297170	·	·	·	·
− 1000996	− 958697	− 958697	− 958698	− 1000000

TABLE 5

THE COMBINATORIAL MATRIX WITH $x = 10^{-8}$

Levinson	Alg. 3.1	Alg. 5.1	Alg. 5.2	Exact
44739242	60397975	60397975	60397977	90000000
− 22369621	− 6710886	− 6710886	− 6710886	− 10000000
− 22369621	·	·	·	·
0	·	·	·	·
⋮	·	·	·	·
0	− 6710886	− 6710886	− 6710886	− 10000000

The results of numerical examples of Tables 3 to 5 show that, for the example of the combinatorial matrix, the parallel algorithms of Sections 3 and 5 perform better than the classical Levinson algorithm.

The essential difference between the Levinson algorithm and Algorithm 3.1 lies in the fact that the terms $\gamma_k(-1)$ and $\omega_k(-1)$ are computed as certain inner products in the Levinson case and, in contrast, they are computed in Algorithm 3.1 using a recursion of Schur-type (see Algorithm 3.2). Since the top left entry in Tables 3, 4, and 5 is just

$$\prod_{k=1}^{9} (1 - \gamma_k(-1)\omega_k(-1))^{-1},$$

we see that (at least in these examples) the algorithm taking advantage of the Schur recursion achieves higher accuracy than the Levinson method.

REFERENCES

1 H. A. Ahmed, J.-M. Delosme, and M. Morf, Highly concurrent computing structures for matrix arithmetic and signal processing, *IEEE Computer* 15:65–82 (1982).

2 E. H. Bareiss, Numerical solution of linear equations with Toeplitz and vector Toeplitz matrices, *Numer. Math.* 13:404–424 (1969).

3 A. Bjorck and V. Pereyra, Solutions of Vandermonde systems of equations, *Math. Comp.* 24:893–903 (1980).

4 R. P. Brent and F. T. Luk, A systolic array of the linear-time solution of Toeplitz systems of equations, *J. VLSI Comput. Syst.* 1:1–22 (1983).

5 B. Friedlander, M. Morf, T. Kailath, and L. Ljung, New inversion formulas for matrices classified in terms of their distance from Toeplitz matrices, *Linear Algebra Appl.* 27:31–60 (1979).

6 I. C. Gohberg and I. A. Feldman, *Convolution Equations and Projection Methods for Their Solution*, Translations of Mathematical Monographs, Vol. 41, Amer. Math. Soc., Providence, 1974.
7 I. Gohberg, T. Kailath, and I. Koltracht, Efficient solution of linear systems of equation with recursive structure, *Linear Algebra Appl.*, 80:81–113 (1986).
8 I. Gohberg and I. Koltracht, Numerical solution of integral equations, fast algorithms and Krein-Sobolev equations, *Numer. Math.* 47:237–288 (1984).
9 G. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins U.P., Baltimore, 1983.
10 R. T. Gregory and D. L. Karney, *A Collection of Matrices for Testing Computational Algorithms*, Wiley-Interscience, 1969.
11 G. Heinig and K. Rost, *Algebraic Methods for Toeplitz-like Matrices and Operators*, Birkhäuser, 1985.
12 T. Kailath, S.-Y. Kung, and M. Morf, Displacement ranks of matrices and linear equations, *J. Math. Anal. Appl.* 68 (1979).
13 S.-Y. Kung and Y. H. Hu, A highly concurrent algorithm and pipelined architecture for solving Toeplitz systems, *IEEE Trans. Acoust. Speech Signal Process.* 31, No. 1 (1983).
14 C. Lanczos, Solution of systems of linear equations by minimized iterations, *J. Res. Nat. Bur. Standards* 49 (1952).
15 C. Lanczos, *Applied Analysis*, Prentice-Hall, New York, 1957.
16 N. Levinson, Appendix, in N. Wiener, *Extrapolation, Interpolation and Smoothing of Stationary Time Series*, Wiley, New York, 1949.
17 J. H. Wilkinson and C. Reinsch, *Linear Algebra*, Handbook for Automatic Computation, Vol. II, Springer, 1971.
18 A. Bultheel, Error analysis of incoming and outgoing schemes for the trigonometric moment problem, *Proc. Conference on Rational Approximation*, Amsterdam, Springer-Verlag, 1981.
19 G. Cybenko, The numerical stability of Levinson-Durbin algorithm for Toeplitz systems of equations, *SIAM-SISSC* 1 (1980).
20 N. Higham, Error analysis of the Bjork-Pereyra algorithm for solving Vandermonde systems, *Num. Anal.* Rep. 108, 1985.

Received 2 December 1985; revised 29 August 1986