A New Algorithm for Solving Toeplitz Systems of Equations

Frank de Hoog
Division of Mathematics and Statistics
CSIRO
P.O. Box 1965
Canberra City, ACT, 2601, Australia

In memory of James H. Wilkinson

Submitted by Gene H. Golub

ABSTRACT

We present some recurrences that are the basis for an algorithm to invert an $n \times n$ Toeplitz system of linear equations with computational complexity $O(n \log^2 n)$. The recurrences used are closely related to the Zohar-Trench and Bareiss algorithms but do not have any obvious connection with other asymptotically fast algorithms for the inversion of Toeplitz systems.

1. INTRODUCTION

The problem of finding the solution of the system of linear equations

$$T_n \mathbf{y} = \mathbf{h}, \tag{1.1}$$

where $\mathbf{y}, \mathbf{h} \in C^n$ and $T_n \in C^n \times C^n$ is a Toeplitz matrix, arises in many data processing applications, such as the solution of the truncated Weiner-Hopf equations [7, 11]. Since a Toeplitz matrix has the form

$$T_n = (t_{ij}), \qquad t_{ij} = t_{i-j}, \quad i, j = 1, \ldots, n, \tag{1.2}$$

so that the entries along the diagonal and parallel to the diagonal are constant, it is not surprising that the structure of (1.2) can be exploited to obtain efficient algorithms for the solution of (1.1). In fact, if the principal

52 Vanderbilt Ave., New York, NY 10017 0024-3795/87/$3.50

minors of T_n are non zero,

$$\det(T_k) \neq 0, \qquad k = 1, \ldots, n, \tag{1.3}$$

then a number of algorithms [1, 16, 17, 18] are available that will solve (1.1) using only $O(n^2)$ operations rather than the $O(n^3)$ required for Gaussian elimination. For problems for which the condition number of T_k, $k = 1, \ldots, n$, is not large, these algorithms have been found to be an efficient way to solve (1.1). Fortunately, many problems in practice have additional structure. For example, if T_n is well conditioned and positive definite, then T_k, $k = 1, \ldots, n$, are well conditioned and positive definite. If however such additional structure is not present, the above algorithms can become unstable. This difficulty may have been overcome by Sweet [15], who has proposed an $O(n^2)$ algorithm based on orthogonal factorization of T_n.

Recently, a number of asymptotically fast algorithms [3, 4, 13] have been proposed for the solution of (1.1) that require only $O(n \log^2 n)$ operations. Brent, Gustavson, and Yun [4] use the fact that any algorithm for the calculation of entries in the Padé table can be used to calculate the solutions of

$$T_n \mathbf{a}_n = \alpha_n \mathbf{e}_1, \tag{1.4a}$$

$$T_n \mathbf{b}_n = \beta_n \mathbf{e}_n, \tag{1.4b}$$

where

$$\mathbf{e}_1 = (1, 0, \ldots, 0)^T,$$

$$\mathbf{e}_n = (0, 0, \ldots, 1)^T,$$

and $\alpha_n, \beta_n \in C$. As explained in Section 2, $\mathbf{a}_n$ and $\mathbf{b}_n$ completely specify T_n^{-1}. In [4], two fast algorithms are presented for the calculation of entries in the Padé table and both could be used to solve (1.4a,b). Unfortunately the algorithms are not necessarily stable even if T_n is positive definite. The reason is that the algorithms described in [4] really solve Hankel problems and will become unstable if the leading principal submatrices of the corresponding Hankel matrix becomes ill conditioned. For example, if we consider

$$T_4 = \begin{bmatrix} 4 & 1 & 1+\varepsilon & 1 \\ 1 & 4 & 1 & 1+\varepsilon \\ 1+\varepsilon & 1 & 4 & 1 \\ 1 & 1+\varepsilon & 1 & 4 \end{bmatrix},$$

then the corresponding Hankel matrix is

$$H_4 = \begin{bmatrix} 1 & 1+\varepsilon & 1 & 4 \\ 1+\varepsilon & 1 & 4 & 1 \\ 1 & 4 & 1 & 1+\varepsilon \\ 4 & 1 & 1+\varepsilon & 1 \end{bmatrix},$$

and clearly its principal submatrix of order two is ill conditioned if ε is small.

The algorithms given by Bitmead and Anderson [3] and Morf [13] are nearly identical and are based on the concept of the displacement rank of a matrix as developed in [6, 10]. The drawback of this approach is that the hidden constant in the $O(n \log^2 n)$ estimate is very large. Specifically, Sexton [14] estimates that the operation count in the Bitmead-Anderson algorithm is bounded by $6300 n \log^2 n + O(n \log n)$ when T_n is symmetric. Although this may well be an overestimate [2], it would appear that the algorithm in its present state of development is not practical.

In this paper, we present another asymptotically fast $O(n \log^2 n)$ algorithm for the solution of (1.1) which does not appear to make contact with the concepts underlying the algorithms given in [4] and [3, 13]. Since the hidden constant in the asymptotic estimate is moderate and the algorithm is closely related to the Zohar-Trench [16–18] and Bareiss [1] algorithms, it is hoped that the present approach (or a modification of it) will lead to a practical algorithm when n is large.

The paper is organised as follows. In Section 2 preliminary results and notation are presented. The algorithm is then presented in Section 3. Finally, some concluding remarks are made in Section 4.

2. PRELIMINARIES

In the sequel we assume that (1.3) holds. Indeed as the recurrences described will become unstable if one or more of the T_k, $k = 1, \ldots, n$, become nearly singular (i.e. has a large condition number relative to that of T_n), we are really only concerned with the application of the recurrences to matrices for which this is not the case.

The key to many of the $O(n^2)$ algorithms is the fact that the inverse of T_n is completely specified by the solutions of (1.4a,b) (essentially the first and last columns of T_n^{-1}). Specifically,

$$T_n^{-1} = \frac{1}{\beta_n a_{n1}} (L_n U_n - W_n G_n), \tag{2.1}$$

where

$$L_n = \begin{bmatrix} a_{n1} & 0 & \cdots & 0 \\ a_{n2} & a_{n1} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ a_{nn} & a_{nn-1} & \cdots & a_{n1} \end{bmatrix}, \qquad U_n = \begin{bmatrix} b_{nn} & b_{nn-1} & \cdots & b_{n1} \\ 0 & b_{nn} & \cdots & b_{n2} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & b_{nn} \end{bmatrix},$$

$$W_n = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ b_{n1} & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ b_{nn-1} & b_{nn-2} & \cdots & b_{n1} & 0 \end{bmatrix},$$

$$G_n = \begin{bmatrix} 0 & a_{nn} & \cdots & a_{n3} & a_{n2} \\ 0 & 0 & \cdots & a_{n4} & a_{n3} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

is the Gohberg–Semencul [8] formulation of the Trench formula. It is easy to verify that (2.1) can also be rewritten as

$$T_n^{-1} = \frac{1}{\beta_n a_{n1}} (U_n L_n - G_n W_n) \tag{2.2}$$

The advantages associated with (2.1) and (2.2) are fairly clear. Firstly, they allow a representation of the inverse that requires only $O(n)$ storage. Secondly, as pointed out by Jain [9] and others, the factors U_n, L_n, W_n, and Z_n are Toeplitz, and hence the solution $T_n^{-1}\mathbf{h}$ can be calculated in only $O(n \log n)$ operations using the fast Fourier transform.

In what follows, we shall find it convenient to associate a polynomial with a vector and vice versa. Thus, with a vector

$$\mathbf{p} = (p_1, p_2, \ldots)^T$$

we associate the polynomial

$$P(x) = \sum_{j \geq 0} p_{j+1} x^j,$$

and conversely. By $\bar{\mathbf{p}}$ we mean the vector whose components are the complex conjugate of $\mathbf{p}$, and we denote by $\bar{P}(x)$ the polynomial associated with $\bar{\mathbf{p}}$.

Finally, we define the "truncated" vector

$$\mathbf{p}^{(k)} = (p_1, \ldots, p_k)^T$$

and associate with $\mathbf{p}^{(k)}$ the polynomial $P^{(k)}(x)$.

Now consider the polynomials $A_k(x)$ and $B_k(x)$, $k = 1, \ldots, n$, associated with the solutions of

$$T_k \mathbf{a}_k = \alpha_k \mathbf{e}_1, \qquad k = 1, \ldots, n, \tag{2.3a}$$

$$T_k \mathbf{b}_k = \beta_k \mathbf{e}_k, \qquad k = 1, \ldots, n, \tag{2.3b}$$

where α_k and β_k are such that the first and last components of $\mathbf{a}_k$ and $\mathbf{b}_k$ respectively are one (i.e. $a_{k1} = 1 = b_{kk}$). It is well known (see for example [12]) and easy to verify that for $k = 0, \ldots, n-1$,

$$\begin{bmatrix} A_{k+1}(x) \\ B_{k+1}(x) \end{bmatrix} = \begin{bmatrix} 1 & -\gamma_{k+1}x \\ -\omega_{k+1} & x \end{bmatrix} \begin{bmatrix} A_k(x) \\ B_k(x) \end{bmatrix}, \tag{2.4}$$

where

$$A_0(x) = 1, \qquad B_0(x) = 1/x, \qquad \gamma_1 = \omega_1 = 0, \tag{2.5a}$$

$$\gamma_{k+1} = \eta_k/\beta_k, \qquad k > 0, \tag{2.5b}$$

$$\omega_{k+1} = \lambda_k/\alpha_k, \qquad k > 0, \tag{2.5c}$$

$$\alpha_k = \sum_{j=0}^{k-1} t_{-j} a_{kj+1}, \qquad k > 0, \tag{2.5d}$$

$$\beta_k = \sum_{j=1}^{k-1} t_{k-j} b_{kj}, \qquad k > 0, \tag{2.5e}$$

$$\eta_k = \sum_{j=0}^{k-1} t_{k-j} a_{kj+1}, \qquad k > 0, \tag{2.5f}$$

$$\lambda_k = \sum_{j=1}^{k} t_{-j} b_{kj}, \qquad k > 0. \tag{2.5g}$$

The advantage of the scaling of $\mathbf{a}_k$ and $\mathbf{b}_k$ becomes apparent when it is noted that

$$\alpha_{k+1} = \beta_{k+1} = \alpha_k - \frac{\lambda_k \eta_k}{\alpha_k}. \tag{2.6}$$

Since the condition $\det(T_k) \neq 0$, $k = 1, \ldots, n$, ensures that $\alpha_k \neq 0$ and $\beta_k \neq 0$, $k = 1, \ldots, n$, the recurrences are well defined.

Equations (2.4), (2.5), and (2.6) are essentially the basis of the Zohar–Trench algorithm [16–18] and provide a decomposition of the form (2.1) in $2n^2 + O(n)$ multiplications and a similar number of additions. In addition, if T_n is Hermitian, then $\eta_k = \bar{\lambda}_k$, $B_k(x) = x^{k-1}\bar{A}_k(1/x)$, $k = 1, \ldots, n$, and the computational complexity is reduced by a factor of two.

We shall also require some knowledge of an algorithm due to Bareiss [1] for the solution of (1.1). For a description of this algorithm it is convenient to use the "shift" matrix $Z_k = (z_{ij}^{(k)})$ defined by

$$z_{ij}^{(k)} = \begin{cases} 1 & \text{if} \quad j - i = k, \\ 0 & \text{otherwise.} \end{cases}$$

Thus, premultiplication of a matrix by Z_k shifts the rows of the matrix up k places with zero fill.

The Bareiss algorithm is as follows. Let

$$\begin{aligned} T^{(1)} &:= T =: T^{(-1)}, \\ \mathbf{h}^{(1)} &:= h =: \mathbf{h}^{(-1)}, \end{aligned} \tag{2.7}$$

and for $k = 2, \ldots, n$ define

$$\theta_k = t_{1k}^{(k-1)} / t_{kk}^{(1-k)}, \tag{2.8a}$$

$$\delta_k = t_{k1}^{(1-k)} / t_{kk}^{(k-1)}, \tag{2.8b}$$

$$T^{(k)} = T^{(k-1)} - \theta_k Z_{k-1} T^{(1-k)}, \tag{2.8c}$$

$$T^{(-k)} = T^{(1-k)} - \delta_k Z_{1-k} T^{(k-1)}, \tag{2.8d}$$

$$\mathbf{h}^{(k)} = \mathbf{h}^{(k-1)} - \theta_k Z_{k-1} \mathbf{h}^{(1-k)}, \tag{2.8e}$$

$$\mathbf{h}^{(-k)} = \mathbf{h}^{(1-k)} - \delta_k Z_{1-k} \mathbf{h}^{(k-1)}. \tag{2.8f}$$

The application of (2.8) to (1.1) yields two systems of linear equations:

$$T^{(n)}\mathbf{y} = \mathbf{h}^{(n)}, \tag{2.9a}$$

$$T^{(-n)}\mathbf{y} = \mathbf{h}^{(-n)}. \tag{2.9b}$$

Since the structure of (2.8) is such that

$$t_{ij}^{(k)} = 0, \qquad 0 < j - i < k, \tag{2.10a}$$

$$t_{ij}^{(-k)} = 0, \qquad 0 < i - j < k, \tag{2.10b}$$

it follows that $T^{(n)}$ is lower triangular while $T^{(-n)}$ is upper triangular. Thus, Equations (2.9a,b) can be solved by forward and backward substitution respectively. Furthermore, as the first $n+1-k$ rows of $T^{(k)}$ and the last $n+1-k$ rows of $T^{(-k)}$ retain their Toeplitz structure, the steps (2.8c,d) can be performed quite efficiently. Specifically, the calculation of $T^{(n)}$ and $T^{(-n)}$ requires $2n^2 + O(n)$ multiplications and a similar number of additions.

Our interest in the Bareiss algorithm stems from the fact that the multipliers θ_k, δ_k, $k = 2,\ldots,n$, are equal to the coefficients γ_k, ω_k, $k = 2,\ldots,n$, in the Levinson recurrence (2.4).

LEMMA 2.1. *For $k = 2,\ldots,n$, let $\alpha_k, \beta_k, \gamma_k, \omega_k$ be defined by* (2.4)–(2.6) *and θ_k, δ_k be defined by* (2.8). *Then*

$$\delta_k = \gamma_k, \qquad k = 2,\ldots,n,$$

and

$$\theta_k = \omega_k, \qquad k = 2,\ldots,n.$$

Proof. Consider the solution of (1.4a,b) where α_n and β_n have been chosen such that

$$a_{n1} = b_{nn} = 1.$$

From (2.4),

$$a_{nn} = -\gamma_n$$

and

$$b_{n1} = -\omega_n.$$

Now apply the Bareiss algorithm to (1.4a). Clearly,

$$\mathbf{h}^{(n)} = \alpha_n \mathbf{e}_1,$$

$$\mathbf{h}^{(-n)} = \alpha_n (1, -\delta_2, \ldots, -\delta_n)^T,$$

and hence from (2.9a,b)

$$a_{n1} = \alpha_n / t_{11}^{(n)} = 1,$$

$$a_{nn} = -\alpha_n \delta_n / t_{nn}^{(-n)} = -\gamma_n.$$

Similarly, on applying the Bareiss algorithm to (1.4b) we obtain

$$b_{n1} = -\beta_n \theta_n / t_{11}^{(n)} = -\omega_n,$$

$$b_{nn} = \beta_n / t_{nn}^{(-n)} = 1.$$

From (2.6), $\alpha_n = \beta_n$, and hence

$$\alpha_n = t_{11}^{(n)} = t_{nn}^{(-n)} = \beta_n.$$

Thus

$$a_{nn} = -\delta_n = -\gamma_n$$

and

$$b_{nn} = -\theta_n = -\omega_n.$$

We have therefore established the result when $k = n$. However, θ_k, δ_k depend only on t_j, $-k < j < k$, and are unchanged if the Bareiss algorithm is applied to a problem of dimension $n+1$. As n is arbitrary, the result follows for $k = 2, \ldots, n$. ■

We conclude this section by deriving a recurrence relationship for some polynomials associate with the Bareiss algorithm. Let

$$M_k(x) = \sum m_{kj} x^j, \qquad k = 1, \ldots, n,$$

and

$$V_k(x) = \sum v_{kj} x^j, \qquad k = 1, \ldots, n,$$

be *rational* functions defined by

$$M_1(x) := \sum_{|j| < n} t_{-j} x^j =: V_1(x)$$

and

$$M_k(x) = M_{k-1}(x) - \omega_k x^{k-1} V_{k-1}(x), \tag{2.11a}$$

$$V_k(x) = V_{k-1}(x) - \gamma_k x^{1-k} M_{k-1}(x). \tag{2.11b}$$

From Lemma 2.1 and Equations (2.8), (2.10), and (2.11) we find that

$$t_{ij}^{(k)} = m_{k,j-i}, \qquad i \leqslant n - k + 1,$$

$$t_{ij}^{(-k)} = v_{k,j-i}, \qquad i \geqslant k,$$

$$m_{kj} = 0, \qquad 0 < j < k,$$

$$v_{kj} = 0, \qquad -k < j < 0,$$

and

$$\gamma_k = v_{k-1,1-k}/m_{k-1,0},$$

$$\omega_k = m_{k-1,k-1}/v_{k-1,0}.$$

For $k = 1, \ldots, n$, we can now write

$$M_k(x) = x^k Q_k(x) + S_k(1/x),$$

$$V_k(x) = P_k(x) + x^{-k} R_k(1/x),$$

where $P_k(x), Q_k(x), R_k(x), S_k(x)$ are *polynomials* defined by

$$P_0(x) := \sum_{j \geq 0} t_{-j} x^j \tag{2.12a}$$

$$Q_0(x) := \sum_{j > 0} t_{-j} x^j, \tag{2.12b}$$

$$R_0(x) := \sum_{j > 0} t_j x^j, \tag{2.12c}$$

$$S_0(x) := \sum_{j \geq 0} t_j x^j; \tag{2.12d}$$

$$\begin{bmatrix} P_k(x) \\ Q_k(x) \end{bmatrix} := \begin{bmatrix} 1 & -\gamma_k \\ -\omega_k & 1 \\ \hline x & x \end{bmatrix} \begin{bmatrix} P_{k-1}(x) \\ Q_{k-1}(x) \end{bmatrix}, \tag{2.13a}$$

$$\begin{bmatrix} R_k(x) \\ S_k(x) \end{bmatrix} := \begin{bmatrix} \frac{1}{x} & -\frac{\gamma_k}{x} \\ -\omega_k & 1 \end{bmatrix} \begin{bmatrix} R_{k-1}(x) \\ S_{k-1}(x) \end{bmatrix}; \tag{2.13b}$$

and

$$\gamma_k = \frac{R_{k-1}(0)}{S_{k-1}(0)} = \frac{r_{k-1,0}}{s_{k-1,0}}, \tag{2.14a}$$

$$\omega_k = \frac{Q_{k-1}(0)}{P_{k-1}(0)} = \frac{q_{k-1,0}}{p_{k-1,0}}. \tag{2.14b}$$

Equations (2.12)–(2.14) can be used as a basis for calculating the γ_k, ω_k, $k = 2, \ldots, n$, required in the Levinson recurrence (2.4). If we take into account that only $n - k$ coefficients need to be retained in $P_k(x)$, $Q_k(x), R_k(x), S_k(x)$ during the computation, then the calculation is equivalent to the Bareiss algorithm. Thus $2n^2 + O(n)$ multiplications are required to calculate γ_k, ω_k, $k = 2, \ldots, n$, and a further $n^2 + O(n)$ multiplications are required to calculate $A_n(x), B_n(x)$ using (2.4). This makes a total of $3n^2 + O(n)$ multiplications, which is to be compared with $2n^2 + O(n)$ multiplications required to implement (2.4)–(2.6). However, Equations (2.12)–(2.14) are completely uncoupled from (2.4), and this fact will be exploited.

3. A DOUBLING STRATEGY FOR TOEPLITZ SYSTEMS

In this section we describe a fast algorithm for the calculation of $\mathbf{a}_n, \mathbf{b}_n$ the solutions of (1.4a,b). This then provides the decomposition (2.1) for the inverse of T_n^{-1}, which can then be used to calculate the solution of (1.1).

From (2.4),

$$\begin{bmatrix} A_k(x) \\ B_k(x) \end{bmatrix} = \begin{bmatrix} 1 & -\gamma_k x \\ -\omega_k & x \end{bmatrix} \cdots \begin{bmatrix} 1 & -\gamma_1 x \\ -\omega_1 & x \end{bmatrix} \begin{bmatrix} 1 \\ 1/x \end{bmatrix},$$

where γ_k, ω_k, $k = 1, \ldots, n$, can be calculated using (2.12)–(2.14) (note that $\gamma_1 = \omega_1 = 0$). Let

$$\begin{bmatrix} C_k(x) & xD_k(x) \\ E_k(x) & xF_k(x) \end{bmatrix} := \begin{bmatrix} 1 & -\gamma_k x \\ -\omega_k & x \end{bmatrix} \cdots \begin{bmatrix} 1 & -\gamma_1 x \\ -\omega_1 & x \end{bmatrix}. \tag{3.1}$$

Clearly, $C_k(x), D_k(x), E_k(x), F_k(x)$ are polynomials of degree $k-1$, and

$$A_k(x) = C_k(x) + D_k(x), \tag{3.2a}$$

$$B_k(x) = E_k(x) + F_k(x). \tag{3.2b}$$

Thus, a knowledge of $C_n(x), D_n(x), E_n(x), F_n(x)$ immediately yields the required $A_n(x), B_n(x)$.

Our aim will be to apply a doubling (or divide and conquer) technique to the product (3.1). On examining (2.12)–(2.14) it is clear that γ_j, ω_j, $j = 1, \ldots, k$, depend only on the first k coefficients of $P_0(x)$, $Q_0(x)$, $R_0(x)$, and $S_0(x)$ (i.e. depend only on $\mathbf{p}_0^{(k)}$, $\mathbf{q}_0^{(k)}$, $\mathbf{r}_0^{(k)}$, and $\mathbf{s}_0^{(k)}$). When we wish to emphasize this dependence we shall write it explicitly. Furthermore, since $\gamma_{j+l}, \omega_{j+l}$, $j = 1, \ldots, k$, depend on $\mathbf{p}_l^{(k)}, \mathbf{q}_l^{(k)}, \mathbf{r}_l^{(k)}, \mathbf{s}_l^{(k)}$ in exactly the same way as γ_j, ω_j, $j = 1, \ldots, k$, depend on $\mathbf{p}_0^{(k)}, \mathbf{q}_0^{(k)}, \mathbf{r}_0^{(k)}, \mathbf{s}_0^{(k)}$, we may write

$$\begin{bmatrix} C_k\left(x, \mathbf{p}_l^{(k)}, \mathbf{q}_l^{(k)}, \mathbf{r}_l^{(k)}, \mathbf{s}_l^{(k)}\right) & xD_k\left(x, \mathbf{p}_l^{(k)}, \mathbf{q}_l^{(k)}, \mathbf{r}_l^{(k)}, \mathbf{s}_l^{(k)}\right) \\ E_k\left(x, \mathbf{p}_l^{(k)}, \mathbf{q}_l^{(k)}, \mathbf{r}_l^{(k)}, \mathbf{s}_l^{(k)}\right) & xF_k\left(x, \mathbf{p}_l^{(k)}, \mathbf{q}_l^{(k)}, \mathbf{r}_l^{(k)}, \mathbf{s}_l^{(k)}\right) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & -\gamma_{l+k} x \\ -\omega_{l+k} & x \end{bmatrix} \cdots \begin{bmatrix} 1 & -\gamma_{l+1} x \\ -\omega_{l+1} & x \end{bmatrix}.$$

Consequently,

$$\begin{bmatrix} C_n\left(x,\mathbf{p}_0^{(n)},\mathbf{q}_0^{(n)},\mathbf{r}_0^{(n)},\mathbf{s}_0^{(n)}\right) & xD_n\left(x,\mathbf{p}_0^{(n)},\mathbf{q}_0^{(n)},\mathbf{r}_0^{(n)},\mathbf{s}_0^{(n)}\right) \\ E_n\left(x,\mathbf{p}_0^{(n)},\mathbf{q}_0^{(n)},\mathbf{r}_0^{(n)},\mathbf{s}_0^{(n)}\right) & xF_n\left(x,\mathbf{p}_0^{(n)},\mathbf{q}_0^{(n)},\mathbf{r}_0^{(n)},\mathbf{s}_0^{(n)}\right) \end{bmatrix}$$

$$= \begin{bmatrix} C_{n-k}\left(x,\mathbf{p}_k^{(n-k)},\mathbf{q}_k^{(n-k)},\mathbf{r}_k^{(n-k)},\mathbf{s}_k^{(n-k)}\right) & xD_{n-k}\left(x,\mathbf{p}_k^{(n-k)},\mathbf{q}_k^{(n-k)},\mathbf{r}_k^{(n-k)},\mathbf{s}_k^{(n-k)}\right) \\ E_{n-k}\left(x,\mathbf{p}_k^{(n-k)},\mathbf{q}_k^{(n-k)},\mathbf{r}_k^{(n-k)},\mathbf{s}_k^{(n-k)}\right) & xF_{n-k}\left(x,\mathbf{p}_k^{(n-k)},\mathbf{q}_k^{(n-k)},\mathbf{r}_k^{(n-k)},\mathbf{s}_k^{(n-k)}\right) \end{bmatrix}$$

$$\times \begin{bmatrix} C_k\left(x,\mathbf{p}_0^{(k)},\mathbf{q}_0^{(k)},\mathbf{r}_0^{(k)},\mathbf{s}_0^{(k)}\right) & xD_k\left(x,\mathbf{p}_0^{(k)},\mathbf{q}_0^{(k)},\mathbf{r}_0^{(k)},\mathbf{s}_0^{(k)}\right) \\ E_k\left(x,\mathbf{p}_0^{(k)},\mathbf{q}_0^{(k)},\mathbf{r}_0^{(k)},\mathbf{s}_0^{(k)}\right) & xF_k\left(x,\mathbf{p}_0^{(k)},\mathbf{q}_0^{(k)},\mathbf{r}_0^{(k)},\mathbf{s}_0^{(k)}\right) \end{bmatrix}. \tag{3.3}$$

Clearly, (3.3) can be used as the basis of a doubling strategy provided that the vectors $\mathbf{p}_k^{(n-k)},\mathbf{q}_k^{(n-k)},\mathbf{r}_k^{(n-k)},\mathbf{s}_k^{(n-k)}$ can be calculated efficiently from $\mathbf{p}_0,\mathbf{q}_0,\mathbf{r}_0,\mathbf{s}_0$. It is easy to verify that

$$\begin{bmatrix} C_k\left(\frac{1}{x}\right) & D_k\left(\frac{1}{x}\right) \\ \frac{1}{x}E_k\left(\frac{1}{x}\right) & \frac{1}{x}F_k\left(\frac{1}{x}\right) \end{bmatrix} = \begin{bmatrix} 1 & -\gamma_k \\ \frac{-\omega_k}{x} & \frac{1}{x} \end{bmatrix} \cdots \begin{bmatrix} 1 & -\gamma_1 \\ \frac{\omega_1}{x} & \frac{1}{x} \end{bmatrix}$$

and

$$\begin{bmatrix} x^{-k}C_k(x) & x^{-k}D_k(x) \\ x^{1-k}E_k(x) & x^{1-k}F_k(x) \end{bmatrix} = \begin{bmatrix} \frac{1}{x} & \frac{-\gamma_k}{x} \\ -\omega_k & 1 \end{bmatrix} \cdots \begin{bmatrix} \frac{1}{x} & \frac{-\gamma_1}{x} \\ -\omega_1 & 1 \end{bmatrix}.$$

Hence, from (2.13a,b) we obtain

$$\begin{bmatrix} P_k(x) \\ Q_k(x) \end{bmatrix} = \begin{bmatrix} C_k\left(\frac{1}{x}\right) & D_k\left(\frac{1}{x}\right) \\ \frac{1}{x}E_k\left(\frac{1}{x}\right) & \frac{1}{x}F_k\left(\frac{1}{x}\right) \end{bmatrix} \begin{bmatrix} P_0(x) \\ Q_0(x) \end{bmatrix} \tag{3.4a}$$

and

$$\begin{bmatrix} R_k(x) \\ S_k(x) \end{bmatrix} = \begin{bmatrix} x^{-k}C_k(x) & x^{-k}D_k(x) \\ x^{1-k}E_k(x) & x^{1-k}F_k(x) \end{bmatrix} \begin{bmatrix} R_0(x) \\ S_0(x) \end{bmatrix}, \tag{3.4b}$$

from which $P_k^{(n-k)}(x), Q_k^{(n-k)}(x), R_k^{(n-k)}(x), S_k^{(n-k)}(x)$ can be calculated by retaining only the first $n-k$ coefficients.

We are now in a position to present an algorithm for the calculation of the product (3.1) when the vectors $\mathbf{p}_0, \mathbf{q}_0, \mathbf{r}_0, \mathbf{s}_0$ are defined by (2.12). For clarity it is presented in a stepwise fashion.

ALGORITHM

1. If $n = 1$, then

$$C_n(x) = C_n(x, \mathbf{p}_0, \mathbf{q}_0, \mathbf{r}_0, \mathbf{s}_0) = 1,$$

$$D_n(x) = D_n(x, \mathbf{p}_0, \mathbf{q}_0, \mathbf{r}_0, \mathbf{s}_0) = -r_{00}/s_{00},$$

$$E_n(x) = D_n(x, \mathbf{p}_0, \mathbf{q}_0, \mathbf{r}_0, \mathbf{s}_0) = -q_{00}/p_{00},$$

$$F_n(x) = F_n(x, \mathbf{p}_0, \mathbf{q}_0, \mathbf{r}_0, \mathbf{s}_0) = 1.$$

2. If $n > 1$,
   let $l = [\log_2 n] - 1$, $k = 2^l$.
   Call algorithm to provide

$$C_k(x) = C_k\left(x, \mathbf{p}_0^{(k)}, \mathbf{q}_0^{(k)}, \mathbf{r}_0^{(k)}, \mathbf{s}_0^{(k)}\right),$$

$$D_k(x) = D_k\left(x, \mathbf{p}_0^{(k)}, \mathbf{q}_0^{(k)}, \mathbf{r}_0^{(k)}, \mathbf{s}_0^{(k)}\right),$$

$$E_k(x) = E_k\left(x, \mathbf{p}_0^{(k)}, \mathbf{q}_0^{(k)}, \mathbf{r}_0^{(k)}, \mathbf{s}_0^{(k)}\right),$$

$$F_k(x) = F_k\left(x, \mathbf{p}_0^{(k)}, \mathbf{q}_0^{(k)}, \mathbf{r}_0^{(k)}, \mathbf{s}_0^{(k)}\right).$$

3. Calculate $\mathbf{p}_k^{(n-k)}, \mathbf{q}_k^{(n-k)}, \mathbf{r}_k^{(n-k)}, \mathbf{s}_k^{(n-k)}$ using (3.4a, b).
4. Call algorithm to provide

$$C_{n-k}\left(x, \mathbf{p}_k^{(n-k)}, \mathbf{q}_k^{(n-k)}, \mathbf{r}_k^{(n-k)}, \mathbf{s}_k^{(n-k)}\right),$$

$$D_{n-k}\left(x, \mathbf{p}_k^{(n-k)}, \mathbf{q}_k^{(n-k)}, \mathbf{r}_k^{(n-k)}, \mathbf{s}_k^{(n-k)}\right),$$

$$E_{n-k}\left(x, \mathbf{p}_k^{(n-k)}, \mathbf{q}_k^{(n-k)}, \mathbf{r}_k^{(n-k)}, \mathbf{s}_k^{(n-k)}\right),$$

$$F_{n-k}\left(x, \mathbf{p}_k^{(n-k)}, \mathbf{q}_k^{(n-k)}, \mathbf{r}_k^{(n-k)}, \mathbf{s}_k^{(n-k)}\right).$$

5. Calculate

$$C_n(x) = C_n\left(x, \mathbf{p}_0^{(n)}, \mathbf{q}_0^{(n)}, \mathbf{r}_0^{(n)}, \mathbf{s}_0^{(n)}\right),$$

$$D_n(x) = D_n\left(x, \mathbf{p}_0^{(n)}, \mathbf{q}_0^{(n)}, \mathbf{r}_0^{(n)}, \mathbf{s}_0^{(n)}\right),$$

$$E_n(x) = E_n\left(x, \mathbf{p}_0^{(n)}, \mathbf{q}_0^{(n)}, \mathbf{r}_0^{(n)}, \mathbf{s}_0^{(n)}\right),$$

$$F_n(x) = F_n\left(x, \mathbf{p}_0^{(n)}, \mathbf{q}_0^{(n)}, \mathbf{r}_0^{(n)}, \mathbf{s}_0^{(n)}\right)$$

using Equation (3.3).

Of course, once $C_n(x)$, $D_n(x)$, $E_n(x)$ and $F_n(x)$ have been calculated, $A_n(x)$ and $B_n(x)$ can be obtained from (3.2a, b).

The potential advantage of the algorithm is that steps 2 and 4, which essentially consist of polynomial multiplication, can be implemented using the fast Fourier transform. There are of course many ways to implement the fast Fourier transform. However, if we use the fast Fourier transform based on powers of 2, then it is not difficult to verify that the above algorithm can be implemented using $5n \log_2^2 n + O(n \log n)$ multiplications and $10n \log_2^2 n + O(n \log n)$ additions.

If further structure is available, the above algorithm can be simplified and the operation count reduced. If T_n is Hermitian, then $P_n(x) = \bar{S}_n(x)$, $Q_n(x) = \bar{R}_n(x)$ and $\gamma_k = \bar{\omega}_k$, $k = 1, \ldots, n$. From (2.12), (2.14), (3.1),

$$P_k(x) = \bar{S}_k(x), \qquad k = 1, \ldots, n$$

$$Q_k(x) = \bar{R}_k(x), \qquad k = 1, \ldots, n$$

$$F_k(x) = x^{k-1}\bar{C}_k\left(\frac{1}{x}\right), \qquad k = 1, \ldots, n$$

and

$$E_k(x) = x^{k-1}\bar{D}_k\left(\frac{1}{x}\right), \qquad k = 1, \ldots, n.$$

It is now easy to devise a modification to the algorithm which reduces the complexity to $\frac{5}{2}n \log_2^2 n + O(n \log n)$ multiplications and $5n \log_2^2 n + O(n \log n)$ additions.

4. CONCLUDING REMARKS

We have presented an algorithm for the solution of Toeplitz systems of equations that has complexity $O(n \log^2 n)$ and for which the hidden constant in the complexity estimate is moderate. Specifically, the algorithm requires $5n \log_2^2 n + O(n \log n)$ and $\frac{5}{2} n \log_2^2 n + O(n \log n)$ multiplications for the general and Hermitian cases respectively.

If in addition the equations are real, these complexity estimates can be approximately halved, although of course the multiplication count is still in terms of complex multiplications. However, it is clear that in practice n needs to be quite large in order that algorithms such as the present one become competitive from a complexity point of view.

It has been shown by Sweet [15] that the Bareiss algorithm is as stable as Gaussian elimination for positive definite matrices. Cybenko [5] has established the same result for the Zohar-Trench algorithm. The present algorithm uses the Bareiss algorithm to calculate the coefficients in the Levinson recurrence (2.4) and then calculates the solution of (1.4) using the Levinson recurrence. We therefore expect that our algorithm is also as stable as Gaussian elimination without pivoting for positive definite matrices.

The author wishes to thank Adam Bojanczyk and Richard Brent for a number of helpful discussions during the course of this work.

REFERENCES

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

2 R. R. Bitmead, private communication.

3 R. R. Bitmead and B. D. O. Anderson, Asymptotically fast solutions of Toeplitz and related systems of linear equations, *Linear Algebra Appl.* 34:103–116 (1980).

4 R. P. Brent, F. G. Gustavson, and D. Y. Y. Yun, Fast solutions of Toeplitz systems of equations and computation of Pade approximants, *J. Algorithms* 1:259–295 (1980).

5 G. Cybenko, The numerical stability of the Levinson-Durbin algorithm for Toeplitz systems of equations, *SIAM J. Sci. Statist. Comput.* 1:303–319 (1980).

6 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).

7 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, R.I., 1974.

8 I. C. Gohberg and A. A. Semencul, On the inversion of finite Toeplitz matrices and their continuous analogs, *Mat. Issled.* 2:201–233 (1972).

9 A. K. Jain, An efficient algorithm for a large Toeplitz set of linear equations, *IEEE Trans. Acoust. Speech Signal Process.* 27:612–615 (1979).

10 T. Kailath, S. Y. Kung, and M. Morf, Displacement ranks of matrices and linear equations, *J. Math. Anal. Appl.* 68:395–407 (1979).

11 T. Kailath, A. Viera, and M. Morf, Inverses of Toeplitz operators, innovations and orthogonal polynomials, *SIAM Rev.* 20:106–119 (1978).

12 N. Levinson, The Wiener RMS error criterion in filter design and prediction, *J. Math. Phys.* 25:261–278 (1947).

13 M. Morf, Doubling algorithms for Toeplitz and related equations, in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Denver, 1980, pp. 954–959.

14 H. Sexton, An analysis of an algorithm of Bitmead and Anderson for the inversion of Toeplitz systems, Naval Oceans Systems Center Technical Report No. 756, 1982.

15 D. R. Sweet, Numerical methods for Toeplitz matrices, Ph.D. Thesis, Dept. of Computing Science, Univ. of Adelaide, 1982.

16 W. Trench, An algorithm for the inversion of finite Toeplitz matrices, *J. Soc. Indust. Appl. Math.* 12:512–522 (1966).

17 S. Zohar, The algorithm of W. F. Trench, *J. Assoc. Comput. Mach.* 16:592–601 (1969).

18 S. Zohar, The solution of a Toeplitz set of linear equations, *J. Assoc. Comput. Mach.* 21:272–276 (1974).

Received 1 February 1984; revised 13 September 1985