A Look-Ahead Schur-Type Algorithm for Solving General Toeplitz Systems

Roland W. Freund
AT&T Bell Laboratories
600 Mountain Avenue, Room 2C–420
Murray Hill, New Jersey 07974–0636, U.S.A.

June 08, 1993

Abstract. The classical fast Toeplitz solvers, such as the Levinson or the Bareiss algorithm, require that all leading principal submatrices be nonsingular, and they are numerically unstable for general Toeplitz systems. We present an extension of the Bareiss algorithm that can be applied to general Toeplitz systems with singular and nonsingular, but ill-conditioned leading principal submatrices.

1 Introduction

Systems of linear equations of the form

$$Tx = b, \quad \text{where} \quad T = [\, t_{i-j} \,]_{i,j=0,\ldots,N} = \begin{bmatrix} t_0 & t_{-1} & \cdots & t_{-N} \\ t_1 & t_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & t_{-1} \\ t_N & \cdots & t_1 & t_0 \end{bmatrix}, \qquad (1)$$

are called *Toeplitz systems*. Here, $t_j \in \mathbb{C}$, $j = -N, -N+1, \ldots, N-1, N$, are complex numbers, and we assume that the coefficient matrix T of (1) is nonsingular. Toeplitz systems arise in many important applications, including time-series analysis, linear prediction, image processing, statistics, probability theory, solution of integral equations, and Padé approximation; see [5, 4] and the references given there. There are classical *fast* algorithms that take advantage of the special Toeplitz structure and require only $\mathcal{O}(N^2)$ operations for the solution of (1), instead of $\mathcal{O}(N^3)$. Fast Toeplitz solvers can be divided into two categories: (i) Levinson-type algorithms [7] that are based on an inverse triangular factorization of T; (ii) Schur-type solvers [6], such as the Bareiss algorithm [1], that are based on a direct triangular factorization of T. While both approaches require $\mathcal{O}(N^2)$ operations on serial computers, they have different computational complexity on parallel and vector machines. In contrast to Levinson-type algorithms that require the calculation of inner products of long vectors, Schur-type methods involve only vector updates. Consequently, on parallel machines with $\mathcal{O}(N)$ processors, Schur-type methods require only $\mathcal{O}(N)$ computing time, as compared to $\mathcal{O}(N \log N)$ computing time for Levinson-type methods; see, e.g., [2].

Unfortunately, the classical Toeplitz solvers require that all leading principal submatrices $T_n := [\, t_{i-j} \,]_{i,j=0,\ldots,n}$, $n = 0, 1, \ldots, N$, of T be nonsingular. If T has exactly singular or nonsingular, yet ill-conditioned leading principal submatrices, then exact breakdowns or near-breakdowns triggering numerical instabilities occur. In [5], Freund and Zha proposed an extension of the classical Levinson algorithm [7] that uses so-called *look-ahead* techniques to skip over singular and ill-conditioned submatrices. In their derivation of this look-ahead Levinson algorithm, they make use of the intimate connection of Toeplitz matrices and *formally biorthogonal polynomials* (FBOPs). It turns out that FBOPs can also be used to devise a look-ahead version of the Bareiss algorithm for solving general Toeplitz systems (1). In this note, we present an outline of the resulting Schur-type Toeplitz solver; for a more detailed description of the algorithm, we refer the reader to [4].

In the sequel, all matrices are allowed to have real or complex entries. As in (1), we always count rows and columns starting with 0; for example, a general $(n+1) \times (m+1)$ matrix is of the form $M = [\, m_{ij} \,]_{\substack{i=0,\ldots,n \\ j=0,\ldots,m}}$. We denote by $M^T := [\, m_{ji} \,]$ the transpose of $M = [\, m_{ij} \,]$, and we use the notation $M_{i_1:i_2, j_1:j_2} := [\, m_{ij} \,]_{\substack{i=i_1,\ldots,i_2 \\ j=j_1,\ldots,j_2}}$ for submatrices of M. Finally, we denote by $\mathcal{P}_n$ the set of all complex polynomials of degree at most n.

2 Formally biorthogonal polynomials

The Toeplitz matrix T in (1) induces a bilinear form $\langle \cdot, \cdot \rangle : \mathcal{P}_N \times \mathcal{P}_N \longmapsto \mathbb{C}$ defined by $\langle \lambda^i, \lambda^j \rangle := t_{i-j}$, $i, j = 0, 1, \ldots, N$. A monic polynomial $\varphi_n(\lambda) = \lambda^n + \cdots$ of degree n is called a *regular* (right) FBOP if it is orthogonal (with respect to $\langle \cdot, \cdot \rangle$) to all polynomials of lower degree, i.e., $\langle \psi, \varphi_n \rangle = 0$ for all

$\psi \in \mathcal{P}_{n-1}$, and if φ_n is uniquely determined by this condition. Similarly, a monic polynomial $\psi_n(\lambda) = \lambda^n + \cdots$ is called a *regular* (left) FBOP if it is the unique polynomial satisfying $\langle \psi_n, \varphi \rangle = 0$ for all $\varphi \in \mathcal{P}_{n-1}$. It turns out that regular FBOPs φ_n and ψ_n of degree n exist if, and only if, the leading principal submatrix T_{n-1} of T is nonsingular. Thus, nonexistence of regular FBOPs is equivalent to a breakdown of the classical fast Toeplitz solvers. Similarly, near-breakdowns, i.e., nonsingular, yet ill-conditioned submatrices T_{n-1} correspond to "ill-conditioned" regular FBOPs.

The basic idea behind both the look-ahead Levinson algorithm proposed in [5] and the look-ahead Schur-type algorithm sketched in the present paper then is to compute only the "well-conditioned" regular FBOPs, and to complement these with suitably chosen *inner* quasi-FBOPs that satisfy only relaxed biorthogonality conditions. This process generates two sequences of monic polynomials

$$\{\varphi_n\}_{n=0}^N \quad \text{and} \quad \{\psi_n\}_{n=0}^N, \tag{2}$$

where, for each n, φ_n and ψ_n is either a pair of regular FBOPs, or a pair of inner quasi-FBOPs. It turns out that the sequences (2) can be generated by means of short recurrences, which are extensions of the classical Szegö recursions for the case that regular FBOPs exist for all n; see [5] for details.

The look-ahead Levinson algorithm in [5] utilizes the polynomials (2) to generate an inverse triangular block-factorization of T of the form

$$\begin{aligned} V^T T U = D, \quad &\text{where} \quad V,\, U \quad \text{are unit upper triangular,} \\ &\text{and} \quad D = \operatorname{diag}\left(D^{(0)}, D^{(1)}, \ldots, D^{(K)}\right). \end{aligned} \tag{3}$$

More precisely, the nth columns of the triangular factors U and V in (3) just consist of the coefficients of the polynomials φ_n and ψ_n, respectively. Furthermore, the beginning of each block $D^{(l)}$, $l = 0, 1, \ldots, K$, of the block-diagonal matrix D in (3) corresponds to a pair of regular FBOPs φ_{n_l} and ψ_{n_l}, and $D^{(l)}$ is a square block of size $n_{l+1} - n_l$, where n_{l+1} and n_l denote the degrees of two consecutive pairs of regular FBOPs.

Rewriting the decomposition (3), we obtain a direct triangular block-factorization of T of the form

$$T = S D^{-1} R^T, \quad \text{where} \quad S = V^{-T} D \quad \text{and} \quad D^{-1} R^T = U^{-1}. \tag{4}$$

Furthermore, D is the block-diagonal matrix from (3), and R, S are block lower triangular matrices. It turns out that the matrices R and S can be expressed directly in terms of the polynomials (2). More precisely, we have

$$\begin{aligned} R &= [\, r_{jn} \,]_{j,n=0,\ldots,N}\,, \quad \text{where} \quad r_{jn} := \langle \psi_n, \lambda^j \rangle, \\ S &= [\, s_{jn} \,]_{j,n=0,\ldots,N}\,, \quad \text{where} \quad s_{jn} := \langle \lambda^j, \varphi_n \rangle. \end{aligned} \tag{5}$$

Using (5) and the recurrences from [5] for generating the polynomials (2), one can derive simple recursions for computing the columns of the matrices R and S in the triangular block-factorization (4) of T; see [4] for details. In the next section, we sketch the resulting algorithm.

3 The algorithm

First, we introduce some further notation. In our algorithm, we also need to update the columns of auxiliary strictly lower triangular matrices P and Q defined by

$$P = [\, p_{jn} \,]_{j,n=0,\ldots,N}\,, \quad \text{where} \quad p_{jn} := \begin{cases} 0 & \text{if } j \le n, \\ \langle \widehat{\varphi}_n, \lambda^j \rangle & \text{if } j > n, \end{cases}$$

$$Q = [\, q_{jn} \,]_{j,n=0,\ldots,N}\,, \quad \text{where} \quad q_{jn} := \begin{cases} 0 & \text{if } j \le n, \\ \langle \lambda^j, \widehat{\psi}_n \rangle & \text{if } j > n. \end{cases}$$

Here, $\widehat{\varphi}_n(\lambda) \equiv \lambda^n \varphi(1/\lambda)$ and $\widehat{\psi}_n(\lambda) \equiv \lambda^n \psi(1/\lambda)$ denotes the *reverse* of φ_n and ψ_n, respectively. We use small letters to denote the columns of the matrices R, S, P, and Q:

$$\begin{aligned} R &= [\, r_0 \quad r_1 \quad \cdots \quad r_N \,] = [\, R^{(0)} \quad R^{(1)} \quad \cdots \quad R^{(K)} \,], \\ S &= [\, s_0 \quad s_1 \quad \cdots \quad s_N \,] = [\, S^{(0)} \quad S^{(1)} \quad \cdots \quad S^{(K)} \,], \\ P &= [\, p_0 \quad p_1 \quad \cdots \quad p_N \,], \quad Q = [\, q_0 \quad q_1 \quad \cdots \quad q_N \,]. \end{aligned} \tag{6}$$

Moreover, in (6), the blocks $R^{(l)}$ and $S^{(l)}$, $l = 0, 1, \ldots, K$, are $(N+1) \times (n_{l+1} - n_l)$ matrices conforming with the block structure of the block-diagonal matrix D in (3). The first columns of $R^{(l)}$ and $S^{(l)}$ correspond to regular FBOPs ψ_{n_l} and φ_{n_l}, while the remaining columns of $R^{(l)}$ and $S^{(l)}$ correspond to inner quasi-FBOPs. Finally, we denote by $Z = [\, z_{ij} \,]_{i,j=0,\ldots,N}$, where $z_{j+1,j} = 1$ and $z_{ij} = 0$ if $i \ne j+1$, the $(N+1) \times (N+1)$ down-shift matrix.

After these preliminaries, we can now sketch our look-ahead Schur-type algorithm for computing the triangular block-factorization (4).

Algorithm 1 (Triangular Block-Factorization of General $(N+1)\times(N+1)$ Toeplitz Matrix T.)

INPUT: maximal size $h_{\max}$ of the blocks $D^{(l)}$ of the block-diagonal matrix D in (4); default is $h_{\max}=5$.

0) Determine n such that T_n has the smallest condition number among T_h, $h=0,1,\ldots,h_{\max}-1$.
Set $l=0$, $n_0=0$, $D^{(0)}=T_n$, and

$$\begin{aligned} R^{(0)} &= [\, r_0 \quad r_1 \quad \cdots \quad r_n \,] = (T_{0:n,0:N})^T, \\ S^{(0)} &= [\, s_0 \quad s_1 \quad \cdots \quad s_n \,] = T_{0:N,0:n}, \end{aligned}$$

and, for all $k=0,1,\ldots,n$,

$$\begin{aligned} p_k &= [\, 0 \quad \cdots \quad 0 \quad t_{-(k+1)} \quad \cdots \quad t_{-N} \,]^T, \\ q_k &= [\, 0 \quad \cdots \quad 0 \quad t_{k+1} \quad \cdots \quad t_N \,]^T. \end{aligned}$$

Go to 2).

While $n<N$ do:

1) Decide whether to perform a regular or an inner step, and go to 2) or 3), respectively.

2) (Regular step.) Set $h=n+1-n_l$. If $h=1$, then set

$$\begin{aligned} \tilde{p}_l &= p_{n_l}, \quad \tilde{q}_l = q_{n_l}, \\ \tilde{r}_l &= r_{n_l}, \quad \tilde{s}_l = s_{n_l}, \\ \xi_l &= \frac{1}{r_{nn}}, \quad \gamma_n = \xi_l\, p_{n+1,n}, \quad \eta_l = \frac{1}{s_{nn}}, \quad \delta_n = \eta_l\, q_{n+1,n}, \\ r_{n+1} &= Z r_n - \tilde{p}_l \delta_n, \quad s_{n+1} = Z s_n - \tilde{q}_l \gamma_n, \\ p_{n+1} &= p_n - Z \tilde{r}_l \gamma_n, \quad q_{n+1} = q_n - Z \tilde{s}_l \delta_n. \end{aligned}$$

If $h>1$, then set $\tilde{R}=R_{n_l:n,n_l:n}$, $\tilde{S}=S_{n_l:n,n_l:n}$, compute α_l, β_l, μ_l, ν_l by solving

$$\tilde{R}\,\alpha_l = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \tilde{S}\,\beta_l = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \tilde{S}\,\mu_l = \begin{bmatrix} 0 \\ S_{n_l:n-1,n} \end{bmatrix}, \quad \tilde{R}\,\nu_l = \begin{bmatrix} 0 \\ R_{n_l:n-1,n} \end{bmatrix},$$

respectively, and set

$$
\begin{aligned}
P^{(l)} &= [\, Z^{h-1}p_{n_l} \quad Z^{h-2}p_{n_l+1} \quad \cdots \quad Zp_{n-1} \quad p_n \,], \\
Q^{(l)} &= [\, Z^{h-1}q_{n_l} \quad Z^{h-2}q_{n_l+1} \quad \cdots \quad Zq_{n-1} \quad q_n \,], \\
\tilde{p}_l &= P^{(l)}\beta_l, \quad \tilde{q}_l = Q^{(l)}\alpha_l, \\
\tilde{r}_l &= R^{(l)}\alpha_l, \quad \tilde{s}_l = S^{(l)}\beta_l, \\
\gamma_n &= p_{n+1,n}, \quad \delta_n = q_{n+1,n}, \quad \xi_l = \eta_l = 1, \\
r_{n+1} &= Zr_n - \tilde{p}_l\delta_n - R^{(l)}\nu_l, \quad s_{n+1} = Zs_n - \tilde{q}_l\gamma_n - S^{(l)}\mu_l, \\
p_{n+1} &= p_n - Z\tilde{r}_l\gamma_n - ZP^{(l)}\mu_n, \quad q_{n+1} = q_n - Z\tilde{s}_l\delta_n - ZQ^{(l)}\nu_n.
\end{aligned}
$$

Set $n_{l+1} = n+1$, $l = l+1$, $R^{(l)} = S^{(l)} = \emptyset$, and go to 4).

3) (Inner step.) Set

$$
\begin{aligned}
\gamma_n &= \xi_{l-1}\, p_{n+1,n}, \quad \delta_n = \eta_{l-1}\, q_{n+1,n}, \\
r_{n+1} &= Zr_n - \tilde{p}_{l-1}\delta_n, \quad s_{n+1} = Zs_n - \tilde{q}_{l-1}\gamma_n, \\
p_{n+1} &= p_n - Z^{n+2-n_l}\tilde{r}_{l-1}\gamma_n, \quad q_{n+1} = q_n - Z^{n+2-n_l}\tilde{s}_{l-1}\delta_n.
\end{aligned}
$$

4) Set

$$
R^{(l)} = [\, R^{(l)} \quad r_{n+1} \,], \quad S^{(l)} = [\, S^{(l)} \quad s_{n+1} \,],
$$

and update $D^{(l)}$. Set $n = n+1$.

We stress that $h_{\max}$ is the only required input parameter for Algorithm 1. The parameters used for the so-called look-ahead strategy, i.e., the decision in step 1), are determined adaptively in the course of the algorithm based solely on $h_{\max}$; see [5, 4]. For further details, such as the update of $D^{(l)}$ in step 4), we refer the reader to [4].

We note that Algorithm 1 reduces to the classical Bareiss algorithm [1] if only regular steps 2) are performed. Clearly, Algorithm 1 involves only vector updates of long vectors, and as a result, it requires $\mathcal{O}(N)$ computing time on parallel machines with $\mathcal{O}(N)$ processors.

Finally, we remark that Algorithm 1 only generates the inverse block-factorization (4) of T, but not the actual solution x of (1). There are various possibilities to obtain x; see [4]. The most efficient is the following procedure:

$$
\text{Solve} \quad Sc = b \quad \text{for} \quad c, \quad \text{and then set} \quad x = Uc. \tag{7}
$$

Note that, by (4), $T = SU^{-1}$, and thus x in (7) indeed solves (1). Clearly, the forward elimination to compute c in (7) can be done in the course of Algorithm 1, without storing S. Generalizing a recursive back-substitution technique for positive definite Toeplitz matrices due to Delosme and Ipsen [3], it is also possible to obtain x in (7) efficiently, without storing R or explicitly computing U; a detailed description of this procedure can be found in [4].

Acknowledgment: The author is indebted to Ilse Ipsen for bringing the recursive back-substitution in [3] to his attention.

References

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

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

[3] Delosme, J.-M.; Ipsen, I. C. F.: Parallel solution of symmetric positive definite systems with hyperbolic rotations. Linear Algebra Appl. **77** (1986), 75–111.

[4] Freund, R. W.: A look-ahead Bareiss algorithm for general Toeplitz matrices. Technical Memorandum 11274-930106-02, AT&T Bell Laboratories, Murray Hill 1993.

[5] Freund, R. W.; Zha, H.: Formally biorthogonal polynomials and a look-ahead Levinson algorithm for general Toeplitz systems. Linear Algebra Appl. **188** (1993), to appear.

[6] Kailath, T.: A theorem of I. Schur and its impact on modern signal processing. In: Gohberg, I. (ed.): I. Schur Methods in Operator Theory and Signal Processing. Birkhäuser Verlag, Basel 1986, pp. 9–30.

[7] Levinson, N.: The Wiener RMS (root mean square) error criterion in filter design and prediction. J. Math. Phys. **25** (1946), 261–278.