UCLA
COMPUTATIONAL AND APPLIED MATHEMATICS

A Look-Ahead Levinson Algorithm for General Toeplitz Systems

Tony F. Chan
Per Christian Hansen

May 1990

CAM Report 90-11

Department of Mathematics
University of California, Los Angeles
Los Angeles, CA. 90024-1555

A Look-Ahead Levinson Algorithm for General Toeplitz Systems[*]

Tony F. Chan[†] Per Christian Hansen[‡]

March 26, 1991

Abstract

A variety of algorithms are available for efficient numerical solution of Toeplitz systems: classical "fast algorithms", such as those due to Levinson [31], Trench [37] and Bareiss [3], as well as the more recent "asymptotically superfast algorithms" due to de Hoog [17], Ammar & Gragg [1] and others. For the special class of symmetric positive definite Toeplitz matrices, the classical "fast algorithms" are known to be weakly numerically stable [8, 15], but otherwise all these methods are potentially numerically unstable. General Toeplitz systems do occur frequently in many signal processing applications, and there is a need for algorithms which are numerically stable and can exploit the Toeplitz structure.

In this paper we present an extension of Levinson's algorithm that is guaranteed to be weakly stable for a large class of general Toeplitz matrices, namely those that do not have many consecutive ill-conditioned leading principal submatrices. The new algorithm adapts itself to the given Toeplitz matrix by skipping over all the ill-conditioned leading principal submatrices encountered during the solution process. This is done by a look-ahead strategy that monitors the condition of the leading principal submatrices and, if necessary, switches to a block step of suitable size.

The overhead of the look-ahead algorithm is typically small compared to the classical Levinson algorithm, and in addition a reliable condition number estimate is produced.

1 Introduction

Toeplitz matrices of large dimensions occur frequently in a variety of signal processing applications, such as linear prediction [39], harmonic retrieval problems [26], ARMA spectral estimation [22], computation of eigenvector oriented spectrums [21], and eigenfilter problems [10, 28, 30]. Surveys of applications can be found in [8, 18, 19, 29]. All these

[*]Both authors are supported by a NATO Collaborative Research Grant 5-2-05/RG900098 and by the National Science Foundation under contracts NSF-DMS87-14612 and ASC-9003002. The first author is also supported by the Army Research Office under contract DAAL03-88-K-0085 and by the Dept. of Energy under contract DE-FG-03-87-ER-25037. The second author is supported by a travel grant from Knud Højgaards Fond.

[†]Department of Mathematics, University of California, Los Angeles, 405 Hilgard Ave., California 90024,USA. Email: chan@math.ucla.edu.

[‡]UNI•C (Danish Computing Center for Research and Education), Building 305, Technical University of Denmark, DK-2800 Lyngby, Denmark. Email: unipch@wuli.uni-c.dk.

problems involve the solution of Toeplitz systems of linear equations, often during eigenvalue computations by inverse iterations [38]. Due to the size of these Toeplitz matrices and the need for fast signal processing algorithms, it is necessary to exploit the special structure of the matrix when solving systems of equations with Toeplitz matrices. However, the numerical stability of the algorithm (i.e., its sensitivity to rounding errors) should preferably not be sacrificed for a fast algorithm.

Fast algorithms for solving Toeplitz systems have been around for many years. Most of these algorithms solve the Toeplitz system in $O(n^2)$ operations, where n is the order of the matrix, and recently also so-called "asymptotically superfast algorithms" with a computational complexity of $O(n \log^2 n)$ have appeared. These methods can be categorized into two main classes: (i) those that implicitly compute a factorization of the Toeplitz matrix itself, and (ii) those that implicitly compute a factorization of the inverse of the Toeplitz matrix. In the first class we find algorithms such as those due to Bareiss [3] and Brent et al. [5, 6], while the algorithms by Levinson [31], Durbin [20], Trench [37], Bitmead & Anderson [4], Morf [33], de Hoog [17], and Ammar & Gragg [1] all belong to the second class. For more details about these and other algorithms, see for example the surveys in [7] and [19].

Common for all these Toeplitz algorithms is that they are potentially numerically unstable for general Toeplitz matrices. Numerical stability has only been proved for the $O(n^2)$-algorithms when applied to symmetric positive definite Toeplitz matrices [8, 15], and only in the sense that for well-conditioned problems one is guaranteed to compute a solution which is close to the exact solution. This is called "weak stability" in [9]. Otherwise, the algorithms may break down by a division by zero or—even worse—the algorithms may give arbitrarily inaccurate solutions without any warning of this event. The reason for this is that all the algorithms implicitly involve the inversion of principal submatrices of the Toeplitz matrix. Unless the Toeplitz matrix is well-conditioned, symmetric and positive definite, and an $O(n^2)$-algorithm is used, we cannot guarantee that all these submatrices are well-conditioned. If one or more ill-conditioned submatrices are encountered during the solution process, then the accuracy of the computed solution deteriorates due to rounding errors caused by these ill-conditioned submatrices.

A third class of $O(n^2)$-algorithms for Toeplitz matrices is the class of iterative methods, such as the preconditioned conjugate gradient method [35]. If a suitable preconditioner, e.g. a circulant one, is used then this method has good convergence properties for symmetric positive definite Toeplitz matrices [11, 13]. However, the convergence properties for nonsymmetric or indefinite matrices are not clear, and it is difficult to get good preconditioners for general Toeplitz matrices. Therefore, these iterative methods do not provide a practical alternative to the methods mentioned above for general Toeplitz matrices.

In view of these facts, the only algorithms that are guaranteed to be numerically stable for general Toeplitz matrices are the classical factorization methods such as LU and LDL^T factorizations with pivoting [23, §3.4 & §4.4]. Unfortunately, these algorithms require $O(n^3)$ operations to solve the Toeplitz system. Therefore, it is important to develop more efficient algorithms for general Toeplitz matrices.

The algorithm presented in this paper is a step in this direction. We focus on one particular algorithm from class (ii), namely Levinson's algorithm [2, 31], and we present an extended version of the algorithm which is guaranteed to be weakly stable for a much broader class of Toeplitz matrices than those being symmetric and positive definite. Our tool for presenting the algorithm and for discussing its numerical properties is matrix analysis and numerical linear algebra, mainly because this is a good way to analyze stability issues, to discuss condition numbers, etc. Introductions to Toeplitz solvers from a linear algebraic point of view can be found in [19] and [23, §4.7].

The classical Levinson algorithm is basically a recursive sequence of updating the solutions to increasingly larger systems of equations involving all the leading principal submatrices in their natural order. Our look-ahead algorithm is essentially a block-version of this method, where the block size is adjusted adaptively in each step to ensure that the leading principal submatrix involved in the particular step is as well-conditioned as possible. We have borrowed the idea for this algorithm from a similar algorithm we designed for symmetric indefinite Toeplitz matrices [14]. Our new algorithm takes advantage of the fact that, during the Levinson algorithm, one can compute inexpensive estimates of the condition number of the leading principal submatrices. Hence, our algorithm can "look ahead" and always choose locally the best conditioned leading principal submatrix, in order to "skip over" all the ill-conditioned ones. In addition, it provides the user with a reliable accuracy estimate for the computed solution. The algorithm is numerically stable for all those Toeplitz matrices that have at most $p_{\max} - 1$ ill-conditioned leading principal submatrices, where $p_{\max}$ is the maximum block-size allowed by the user. Our algorithm is particularly well suited for eigenvalue computations by means of inverse iterations, such as the algorithms in [10, 16, 38].

The overhead of the new algorithm is difficult to predict, because it depends on the number of ill-conditioned submatrices encountered. A matrix of order $n = 120$ with 39 singular leading principal submatrices requires an overhead of 41 % including condition estimation. For matrices with one ill-conditioned leading principal submatrix, which typically appear in Toeplitz eigenvalue computations [27, 38], the overhead is about 20 %. For matrices with no ill-conditioned leading submatrices, the overhead is less than 10 % for matrices of order greater than 60. Notice that this last overhead is primarily due to the necessary condition estimation in each step.

The idea of skipping certain leading principal submatrices is a natural one and, in fact, has been used before by several authors [18, 24, 36, 40]. In all these algorithms, the decision for taking a block-step is based solely on the size of the prediction error, which appears as a denominator in the algorithms. The prediction error is zero if and only if the corresponding leading principal submatrix is singular. In [18, 24, 40] a block step is taken if a true zero prediction error is encountered. Moreover, these algorithms make explicit use of the fact that the leading principal submatrix is singular, and therefore need further development before they can be used in finite precision arithmetic. Sweet [36] uses a simple threshold-type test and takes a block step when the prediction error is less than a pre-set threshold. However, an ill-conditioned leading submatrix is not guaranteed to reveal itself by a small prediction error, and his algorithm is therefore not guaranteed to detect the need for a block step. Our algorithm is based on a more reliable—and still efficient—test for ill-conditioned submatrices.

Fortran implementations of our algorithms have also been developed [25], and they are available from both authors.

The paper is organized as follows. In Section 2, we review the classical Levinson algorithm for nonsymmetric matrices, while the new look-ahead Levinson algorithm is presented in Section 3. In Sections 4 and 5 we give some important details concerning the practical implementation of the extended algorithm. Finally, in Section 6 we present numerical results. Our notation is the following. Capital letters, boldface small letters, and regular small letters denote matrices, vectors, and scalars, respectively. I_k denotes the identity matrix of order k, while E_k denotes the $k \times k$ exchange matrix $E_k = \text{antidiag}(1, 1, \ldots, 1)$, which satisfies $E_k^2 = I_k$. The superscript "T" denotes matrix and vector transposition, and $\|\cdot\|_2$ denotes the matrix 2-norm which is equal to the largest singular value of the matrix [23, §2.5.3].

2 The classical Levinson algorithm

Since our new algorithm is basically an extension of the classical Levinson algorithm for nonsymmetric matrices, let us first summarize this algorithm following the notation in [23, §4.7]. For a formulation of the Levinson algorithm in terms of Szegö polynomials, see e.g. [1, §5]. We write the Toeplitz system as $T_n \mathbf{x} = \mathbf{b}$, where the real $n \times n$ Toeplitz matrix and the corresponding right-hand side are given by

$$T_n = \begin{pmatrix} \rho_0 & \rho_1 & \rho_2 & \cdots & \rho_{n-1} \\ \sigma_1 & \rho_0 & \rho_1 & \cdots & \rho_{n-2} \\ \sigma_2 & \sigma_1 & \rho_0 & \cdots & \rho_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma_{n-1} & \sigma_{n-2} & \sigma_{n-3} & \cdots & \rho_0 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \beta_n \end{pmatrix}. \tag{1}$$

Notice that we write σ_i instead of the standard notation ρ_{-i}, because this makes our notation clearer. We recall that a Toeplitz matrix as well as its inverse are persymmetric, i.e. $E_n T_n E_n = T_n^T$ and $E_n T_n^{-1} E_n = (T_n^{-1})^T$. At the kth stage, the Levinson algorithm has recursively computed the solution $\mathbf{x}_k$ to the order-k problem $T_k \mathbf{x}_k = \mathbf{b}_k \equiv (\beta_1, \ldots, \beta_k)^T$ and, simultaneously, the solutions $\mathbf{y}_k$ and $\mathbf{z}_k$ to the following two Yule-Walker-problems of order k:

$$T_k^T \mathbf{y}_k = -\mathbf{r}_k \equiv -(\rho_1, \ldots, \rho_k)^T, \qquad T_k \mathbf{z}_k = -\mathbf{s}_k \equiv -(\sigma_1, \ldots, \sigma_k)^T. \tag{2}$$

The next step is then to solve the three problems $T_{k+1}\mathbf{x}_{k+1} = \mathbf{b}_{k+1}$, $T_{k+1}^T \mathbf{y}_{k+1} = -\mathbf{r}_{k+1}$, and $T_{k+1}\mathbf{z}_{k+1} = -\mathbf{s}_{k+1}$. To illustrate this step, we first factorize the matrices T_{k+1} and T_{k+1}^T as follows:

$$T_{k+1} = \begin{pmatrix} T_k & E_k \mathbf{r}_k \\ \mathbf{s}_k^T E_k & \rho_0 \end{pmatrix} = \begin{pmatrix} I_k & \mathbf{0} \\ \mathbf{s}_k^T E_k T_k^{-1} & 1 \end{pmatrix} \begin{pmatrix} T_k & E_k \mathbf{r}_k \\ \mathbf{0}^T & \gamma^{(k)} \end{pmatrix} \tag{3}$$

$$T_{k+1}^T = \begin{pmatrix} T_k^T & E_k \mathbf{s}_k \\ \mathbf{r}_k^T E_k & \rho_0 \end{pmatrix} = \begin{pmatrix} I_k & \mathbf{0} \\ \mathbf{r}_k^T E_k (T_k^{-1})^T & 1 \end{pmatrix} \begin{pmatrix} T_k^T & E_k \mathbf{s}_k \\ \mathbf{0}^T & \gamma^{(k)} \end{pmatrix}. \tag{4}$$

We note that the bottom right element in the rightmost factor of both T_{k+1} and T_{k+1}^T is the same, namely the *prediction error* $\gamma^{(k)}$. From our linear algebraic point of view, $\gamma^{(k)}$ is the Schur complement of T_k in T_{k+1}, and it is given by

$$\gamma^{(k)} \equiv \rho_0 - \mathbf{s}_k^T E_k T_k^{-1} E_k \mathbf{r}_k = \rho_0 - \mathbf{s}_k^T (T_k^{-1})^T \mathbf{r}_k = \rho_0 + \mathbf{s}_k^T \mathbf{y}_k. \tag{5}$$

Now, write the solutions sought as $\mathbf{x}_{k+1} = \begin{pmatrix} \mathbf{u}_k \\ \alpha^{(k)} \end{pmatrix}$, $\mathbf{y}_{k+1} = \begin{pmatrix} \mathbf{v}_k \\ \eta^{(k)} \end{pmatrix}$ and $\mathbf{z}_{k+1} = \begin{pmatrix} \mathbf{w}_k \\ \phi^{(k)} \end{pmatrix}$. Using the factorizations in (3) and (4), it is easy to see that $\mathbf{x}_{k+1}$, $\mathbf{y}_{k+1}$ and $\mathbf{z}_{k+1}$ are the solutions to the three systems

$$\begin{pmatrix} T_k & E_k\mathbf{r}_k \\ \mathbf{0}^T & \gamma^{(k)} \end{pmatrix} \begin{pmatrix} \mathbf{u}_k \\ \alpha^{(k)} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_k \\ \beta_{k+1} - \mathbf{s}_k^T E_k \mathbf{x}_k \end{pmatrix} \tag{6}$$

$$\begin{pmatrix} T_k^T & E_k\mathbf{s}_k \\ \mathbf{0}^T & \gamma^{(k)} \end{pmatrix} \begin{pmatrix} \mathbf{v}_k \\ \eta^{(k)} \end{pmatrix} = \begin{pmatrix} -\mathbf{r}_k \\ -\rho_{k+1} - \mathbf{r}_k^T E_k \mathbf{y}_k \end{pmatrix} \tag{7}$$

$$\begin{pmatrix} T_k & E_k\mathbf{r}_k \\ \mathbf{0}^T & \gamma^{(k)} \end{pmatrix} \begin{pmatrix} \mathbf{w}_k \\ \phi^{(k)} \end{pmatrix} = \begin{pmatrix} -\mathbf{s}_k \\ -\sigma_{k+1} - \mathbf{s}_k^T E_k \mathbf{z}_k \end{pmatrix}. \tag{8}$$

Solving these systems by backsubstitution, we see that the new vectors $\mathbf{x}_{k+1}$, $\mathbf{y}_{k+1}$ and $\mathbf{z}_{k+1}$ are given by the following updates to $\mathbf{x}_k$, $\mathbf{y}_k$ and $\mathbf{z}_k$:

$$\mathbf{x}_{k+1} = \begin{pmatrix} \mathbf{x}_k \\ 0 \end{pmatrix} + \begin{pmatrix} E_k\mathbf{y}_k \\ 1 \end{pmatrix} \alpha^{(k)} \tag{9}$$

$$\mathbf{y}_{k+1} = \begin{pmatrix} \mathbf{y}_k \\ 0 \end{pmatrix} + \begin{pmatrix} E_k\mathbf{z}_k \\ 1 \end{pmatrix} \eta^{(k)} \tag{10}$$

$$\mathbf{z}_{k+1} = \begin{pmatrix} \mathbf{z}_k \\ 0 \end{pmatrix} + \begin{pmatrix} E_k\mathbf{y}_k \\ 1 \end{pmatrix} \phi^{(k)}, \tag{11}$$

where the scalars $\alpha^{(k)}$, $\eta^{(k)}$ and $\phi^{(k)}$ are computed by

$$\alpha^{(k)} = (\beta_{k+1} - \mathbf{s}_k^T E_k \mathbf{x}_k)/\gamma^{(k)} \tag{12}$$

$$\eta^{(k)} = (-\rho_{k+1} - \mathbf{r}_k^T E_k \mathbf{y}_k)/\gamma^{(k)} \tag{13}$$

$$\phi^{(k)} = (-\sigma_{k+1} - \mathbf{s}_k^T E_k \mathbf{z}_k)/\gamma^{(k)}. \tag{14}$$

For efficiency, the prediction error $\gamma^{(k)}$ should not be computed via its definition (5). Instead, it is computed recursively along with the $\mathbf{x}_{k+1}$, $\mathbf{y}_{k+1}$ and $\mathbf{z}_{k+1}$. We shall not derive the recursion formula here, but instead refer to [23, §4.7]. The recursion starts with $\gamma^{(0)} = \rho_0$, $\eta^{(0)} = -\rho_1/\rho_0$ and $\phi^{(0)} = -\sigma_1/\rho_0$, and then $\gamma^{(k+1)}$ is given by:

$$\gamma^{(k+1)} = (1 - \eta^{(k)}\phi^{(k)})\gamma^{(k)}. \tag{15}$$

It is easy to show that the classical Levinson algorithm for nonsymmetric Toeplitz matrices requires $3n^2 - 2n - 1$ multiplications or divisions. For symmetric matrices, $\mathbf{r}_k = \mathbf{s}_k$, $\mathbf{y}_k = \mathbf{z}_k$, $\eta^{(k)} = \phi^{(k)}$, and the computational effort reduces to $2n^2 - 2$ multiplications or divisions.

Because of the division by the prediction error $\gamma^{(k)}$ in (12), (13) and (14), numerical instability will occur if one or several successive $\gamma^{(k)}$ encountered during the recursive process become numerically small. We stress that this numerical instability is also present for formulations based on polynomials—it is not associated with the matrix formulation used here. We also stress that for general Toeplitz matrices, one or more $|\gamma^{(k)}|$ may be arbitrarily small even if the matrix is well-conditioned, i.e. if $|\gamma^{(n)}|$ is not small. The only matrices, for which numerical instability in the classical Levinson algorithm is guaranteed no to occur, are well-conditioned symmetric positive definite matrices, for which there is a monotonic decrease of the $|\gamma^{(k)}|$ so they cannot be smaller than $|\gamma^{(n)}|$. To avoid a small $|\gamma^{(k)}|$ for general Toeplitz matrices we could incorporate pivoting, but this would destroy the Toeplitz structure. This motivates us to perform a block-step instead, as described in the next section.

3 The look-ahead Levinson algorithm

Let us now describe our extension of the classical Levinson algorithm. Assume, at stage k, that we have encountered an ill-conditioned T_{k+1}. We therefore wish to perform a *p-step*, i.e. to skip ahead from T_k to a well-conditioned T_{k+p}. Assume for the moment that p is given; in Section 4 we address the question of how to actually detect an ill-conditioned T_{k+1} and how to choose p. Let us define the *shifted* vectors $\mathbf{r}_{k,i} \equiv (\rho_{1+i}, \rho_{2+i}, \ldots, \rho_{k+i})^T$ and $\mathbf{s}_{k,i} \equiv (\sigma_{1+i}, \sigma_{2+i}, \ldots, \sigma_{k+i})^T$ (in particular $\mathbf{r}_{k,0} = \mathbf{r}_k$ and $\mathbf{s}_{k,0} = \mathbf{s}_k$) and the $k \times p$ matrices $R_p \equiv (\mathbf{r}_{k,0}, \ldots, \mathbf{r}_{k,p-1})$ and $S_p \equiv (\mathbf{s}_{k,0}, \ldots, \mathbf{s}_{k,p-1})$. Then we can write

$$T_{k+p} = \begin{pmatrix} T_k & E_k R_p \\ S_p^T E_k & T_p \end{pmatrix} = \begin{pmatrix} T_k & E_k\mathbf{r}_k & E_k\mathbf{r}_{k,1} & E_k\mathbf{r}_{k,2} & \cdots & E_k\mathbf{r}_{k,p-1} \\ \mathbf{s}_k^T E_k & \rho_0 & \rho_1 & \rho_2 & \cdots & \rho_{p-1} \\ \mathbf{s}_{k,1}^T E_k & \sigma_1 & \rho_0 & \rho_1 & \cdots & \rho_{p-2} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{s}_{k,p-1}^T E_k & \sigma_{p-1} & \sigma_{p-2} & \sigma_{p-3} & \cdots & \rho_0 \end{pmatrix}. \quad (16)$$

In order to update the $\mathbf{x}_k$, $\mathbf{y}_k$ and $\mathbf{z}_k$ to $\mathbf{x}_{k+p}$, $\mathbf{y}_{k+p}$ and $\mathbf{z}_{k+p}$, we proceed in a similar fashion as in the classical Levinson algorithm, but now using block Gaussian-elimination to solve the three $(k+p) \times (k+p)$ systems $T_{k+p}\mathbf{x}_{k+p} = \mathbf{b}_{k+p}$, $T_{k+p}^T \mathbf{y}_{k+p} = -\mathbf{r}_{k+p}$ and $T_{k+p}\mathbf{z}_{k+p} = -\mathbf{s}_{k+p}$. First, we introduce the matrices Y_p and Z_p as the solutions to the problems

$$T_k^T Y_p = -R_p, \qquad T_k Z_p = -S_p. \quad (17)$$

Then, we factorize T_{k+p} and T_{k+p}^T as

$$T_{k+p} = \begin{pmatrix} I_k & 0 \\ S_p^T E_k T_k^{-1} & I_p \end{pmatrix} \begin{pmatrix} T_k & E_k R_p \\ 0 & \Gamma_p^{(k)} \end{pmatrix} \quad (18)$$

$$T_{k+p}^T = \begin{pmatrix} I_k & 0 \\ R_p^T E_k (T_k^{-1})^T & I_p \end{pmatrix} \begin{pmatrix} T_k^T & E_k S_p \\ 0 & (\Gamma_p^{(k)})^T \end{pmatrix}, \quad (19)$$

where the $p \times p$ matrix $\Gamma_p^{(k)}$ is the Schur complement of T_k, given by

$$\Gamma_p^{(k)} = T_p - S_p^T E_k T_k^{-1} E_k R_p = T_p - S_p^T (T_k^{-1})^T R_p = T_p + S_p^T Y_p. \quad (20)$$

Note that the first column of Y_p and Z_p is simply $\mathbf{y}_k$ and $\mathbf{z}_k$, respectively. Now, write the sought solution as $\mathbf{x}_{k+p} = \begin{pmatrix} \mathbf{u}_k \\ \mathbf{a}_p^{(k)} \end{pmatrix}$, $\mathbf{y}_{k+p} = \begin{pmatrix} \mathbf{v}_k \\ \mathbf{e}_p^{(k)} \end{pmatrix}$ and $\mathbf{z}_{k+p} = \begin{pmatrix} \mathbf{w}_k \\ \mathbf{f}_p^{(k)} \end{pmatrix}$, and let $\mathbf{b}_{k+p} = \begin{pmatrix} \mathbf{b}_k \\ \mathbf{b}_p^{(k)} \end{pmatrix}$, $\mathbf{r}_{k+p} = \begin{pmatrix} \mathbf{r}_k \\ \mathbf{r}_p^{(k)} \end{pmatrix}$ and $\mathbf{s}_{k+p} = \begin{pmatrix} \mathbf{s}_k \\ \mathbf{s}_p^{(k)} \end{pmatrix}$. Here, we have defined the p-vectors $\mathbf{b}_p^{(k)} \equiv (\beta_{k+1}, \ldots, \beta_{k+p})^T$, $\mathbf{r}_p^{(k)} \equiv (\rho_{k+1}, \ldots, \rho_{k+p})^T$ and $\mathbf{s}_p^{(k)} \equiv (\sigma_{k+1}, \ldots, \sigma_{k+p})^T$. Then we use the factorizations in (18) and (19) to obtain the systems

$$\begin{pmatrix} T_k & E_k R_p \\ 0 & \Gamma_p^{(k)} \end{pmatrix} \begin{pmatrix} \mathbf{u}_k \\ \mathbf{a}_p^{(k)} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_k \\ \mathbf{b}_p^{(k)} - S_p^T E_k \mathbf{x}_k \end{pmatrix} \quad (21)$$

$$\begin{pmatrix} T_k^T & E_k S_p \\ 0 & (\Gamma_p^{(k)})^T \end{pmatrix} \begin{pmatrix} \mathbf{v}_k \\ \mathbf{e}_p^{(k)} \end{pmatrix} = \begin{pmatrix} -\mathbf{r}_k \\ -\mathbf{r}_p^{(k)} - R_p^T E_k \mathbf{y}_k \end{pmatrix} \quad (22)$$

$$\begin{pmatrix} T_k & E_k R_p \\ 0 & \Gamma_p^{(k)} \end{pmatrix} \begin{pmatrix} \mathbf{w}_k \\ \mathbf{f}_p^{(k)} \end{pmatrix} = \begin{pmatrix} -\mathbf{s}_k \\ -\mathbf{s}_p^{(k)} - S_p^T E_k \mathbf{z}_k \end{pmatrix}. \tag{23}$$

If we solve these systems by backsubstitution, we see that the new vectors $\mathbf{x}_{k+p}$, $\mathbf{y}_{k+p}$ and $\mathbf{z}_{k+p}$ are given by the following updates to $\mathbf{x}_k$, $\mathbf{y}_k$ and $\mathbf{z}_k$:

$$\mathbf{x}_{k+p} = \begin{pmatrix} \mathbf{x}_k \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} E_k Y_p \\ I_p \end{pmatrix} \mathbf{a}_p^{(k)} \tag{24}$$

$$\mathbf{y}_{k+p} = \begin{pmatrix} \mathbf{y}_k \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} E_k Z_p \\ I_p \end{pmatrix} \mathbf{e}_p^{(k)} \tag{25}$$

$$\mathbf{z}_{k+p} = \begin{pmatrix} \mathbf{z}_k \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} E_k Y_p \\ I_p \end{pmatrix} \mathbf{f}_p^{(k)}, \tag{26}$$

where the vectors $\mathbf{a}_p^{(k)}$, $\mathbf{e}_p^{(k)}$ and $\mathbf{f}_p^{(k)}$ are the solutions to the indefinite systems

$$\Gamma_p^{(k)} \mathbf{a}_p^{(k)} = \mathbf{b}_p^{(k)} - S_p^T E_k \mathbf{x}_k \tag{27}$$

$$(\Gamma_p^{(k)})^T \mathbf{e}_p^{(k)} = -\mathbf{r}_p^{(k)} - R_p^T E_k \mathbf{y}_k \tag{28}$$

$$\Gamma_p^{(k)} \mathbf{f}_p^{(k)} = -\mathbf{s}_p^{(k)} - S_p^T E_k \mathbf{z}_k. \tag{29}$$

It is easy to see that the block step will handle the numerical instability problem as long as p is chosen properly. It would be expensive if we were to solve for the extra $p-1$ vectors in Y_p and Z_p (17) naively, say using Levinson's algorithm, because we would have to choose an a priori number $p_{\max}$ such that $p \le p_{\max}$ and then carry along $p_{\max}-1$ extra systems. Instead, we present a less expensive updating procedure, computing Y_p and Z_p only when necessary. The idea is to write the columns of these two matrices in terms of updates to already computed quantities. This is possible due to the following theorem.

Theorem 1 *Let the p columns of Y_p and Z_p be denoted by $\mathbf{y}_{k,i}$ and $\mathbf{z}_{k,i}$, $i = 0, \ldots, p-1$, each column satisfying $T_k^T \mathbf{y}_{k,i} = -\mathbf{r}_{k,i}$ and $T_k \mathbf{z}_{k,i} = -\mathbf{s}_{k,i}$. Also, let $(\mathbf{v})_j$ denote the j-th component of the vector $\mathbf{v}$, and define the $k \times k$ "upshift" matrix*

$$\Delta_k \equiv \begin{pmatrix} 0 & 1 & & & \\ 0 & 0 & 1 & & \\ & 0 & 0 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 0 & 0 \end{pmatrix}. \tag{30}$$

Then all the vectors $\mathbf{y}_{k,i}$ and $\mathbf{z}_{k,i}$ for $i = 1, \ldots, p-1$ can be computed recursively from $\mathbf{y}_{k,0} = \mathbf{y}_k$ and $\mathbf{z}_{k,0} = \mathbf{z}_k$ by means of the relations:

$$\mathbf{y}_{k,i} = \Delta_k y_{k,i-1} - (\mathbf{y}_{k,i-1})_1 \mathbf{y}_k + c_i \mathbf{g}_k, \qquad i = 1, \ldots, p-1 \tag{31}$$

$$\mathbf{z}_{k,i} = \Delta_k z_{k,i-1} - (\mathbf{z}_{k,i-1})_1 \mathbf{z}_k + d_i \mathbf{h}_k, \qquad i = 1, \ldots, p-1 \tag{32}$$

where we have defined the vectors $\mathbf{g}_k$ and $\mathbf{h}_k$ by

$$\mathbf{g}_k \equiv \frac{1}{\gamma^{(k-1)}} \begin{pmatrix} E_{k-1} \mathbf{z}_{k-1} \\ 1 \end{pmatrix}, \qquad \mathbf{h}_k \equiv \frac{1}{\gamma^{(k-1)}} \begin{pmatrix} E_{k-1} \mathbf{y}_{k-1} \\ 1 \end{pmatrix}, \tag{33}$$

and where c_i and d_i are the i-th components of the right-hand sides of the equations in (28) and (29):

$$\mathbf{c} = -\mathbf{r}_p^{(k)} - R_p^T E_k \mathbf{y}_k, \qquad \mathbf{d} = -\mathbf{s}_p^{(k)} - S_p^T E_k \mathbf{z}_k. \tag{34}$$

Proof. We shall prove the relation (31) for $\mathbf{y}_{k,i-1}$, the relation (32) for $\mathbf{z}_{k,i-1}$ is proved analogously. Consider the following matrix-vector product of order $k+1$:

$$T_{k+1}^T \begin{pmatrix} \mathbf{y}_{k,i-1} \\ 0 \end{pmatrix} = \begin{pmatrix} \rho_0 & \mathbf{s}_k^T \\ \mathbf{r}_k & T_k^T \end{pmatrix} \begin{pmatrix} (\mathbf{y}_{k,i-1})_1 \\ \Delta_k y_{k,i-1} \end{pmatrix} = \begin{pmatrix} \rho_0(\mathbf{y}_{k,i-1})_1 + \mathbf{s}_k^T \Delta_k y_{k,i-1} \\ (\mathbf{y}_{k,i-1})_1 \mathbf{r}_k + T_k^T \Delta_k y_{k,i-1} \end{pmatrix}. \qquad (35)$$

Using $T_k^T \mathbf{y}_{k,i} = -\mathbf{r}_{k,i}$ and Eq. (17) it follows that this vector is also given by

$$T_{k+1}^T \begin{pmatrix} \mathbf{y}_{k,i-1} \\ 0 \end{pmatrix} = \begin{pmatrix} T_k^T & E_k \mathbf{s}_k \\ \mathbf{r}_k^T E_k & \rho_0 \end{pmatrix} \begin{pmatrix} \mathbf{y}_{k,i-1} \\ 0 \end{pmatrix} = \begin{pmatrix} -\mathbf{r}_{k,i-1} \\ \mathbf{r}_k^T E_k \mathbf{y}_{k,i-1} \end{pmatrix}$$

$$= \begin{pmatrix} -\rho_i \\ -\mathbf{r}_{k,i} \end{pmatrix} - \begin{pmatrix} \mathbf{0} \\ -\rho_{k+i} - \mathbf{r}_k^T E_k \mathbf{y}_{k,i-1} \end{pmatrix} = \begin{pmatrix} -\rho_i \\ -\mathbf{r}_{k,i} \end{pmatrix} - \begin{pmatrix} \mathbf{0} \\ -\rho_{k+i} - \mathbf{y}_k^T E_k \mathbf{r}_{k,i-1} \end{pmatrix}. \qquad (36)$$

In the last step, we have used $\mathbf{r}_k^T E_k \mathbf{y}_{k,i-1} = -\mathbf{y}_k^T T_k E_k \mathbf{y}_{k,i-1} = -\mathbf{y}_k^T E_k T_k^T \mathbf{y}_{k,i-1} = \mathbf{y}_k^T E_k \mathbf{r}_{k,i-1}$. Equating the last k elements of the vectors in (35) and (36), we obtain

$$T_k^T \Delta_k y_{k,i-1} = -\mathbf{r}_{k,i} - (\mathbf{y}_{k,i-1})_1 \mathbf{r}_k - \begin{pmatrix} \mathbf{0} \\ c_i \end{pmatrix}, \qquad i = 1, \ldots, p-1.$$

By multiplication with $(T_k^{-1})^T$ and using $T_k^T \mathbf{y}_{k,i} = -\mathbf{r}_{k,i}$ again, we get

$$\Delta_k y_{k,i-1} = \mathbf{y}_{k,i} + (\mathbf{y}_{k,i-1})_1 \mathbf{y}_k - (T_k^{-1})^T \begin{pmatrix} \mathbf{0} \\ c_i \end{pmatrix}, \qquad i = 1, \ldots p-1.$$

To obtain the expression (31) for $\mathbf{y}_{k,i}$ we must now consider the rightmost term in the above equation. Using one step of the classical Levinson algorithm to go from T_{k-1}^T to T_k^T (cf. Section 2), the solution to this system can simply be written as

$$(T_k^{-1})^T \begin{pmatrix} \mathbf{0} \\ c_i \end{pmatrix} = \frac{c_i}{\gamma^{(k-1)}} \begin{pmatrix} E_k \mathbf{z}_{k-1} \\ 1 \end{pmatrix}.$$

Thus, we have proved (31). □

After the p-step, we have two choices for computing the $\gamma^{(k+p)}$ required in the next step. We can use an updating formula similar to (15), which becomes $\gamma^{(k+p)} = \gamma^{(k)} - \mathbf{d}^T \mathbf{e}_p^{(k)}$, or we can compute $\gamma^{(k+p)}$ by the definition (5), e.g. $\gamma^{(k+p)} = \rho_0 + \mathbf{s}_{k+p}^T \mathbf{y}_{k+p}$. We tried both in our numerical experiments, and the latter (more expensive) formula gives better accuracy in the computed solution for large values of the order n, so we recommend the latter approach to computing $\gamma^{(k+p)}$.

Let us determine the computational effort for performing a p-step at the kth stage. Computation of the extra $2(p-1)$ vectors $\mathbf{y}_{k,i}$ and $\mathbf{z}_{k,i}$ requires $4(p-1)k$ multiplications. Since $\gamma^{(k)}$ is already given by (5), setting up $\Gamma_p^{(k)}$ and the right-hand sides in (27), (28) and (29) requires $(\frac{1}{2}p(p+1) - 1 + 3p)k = (\frac{1}{2}p^2 + \frac{7}{2}p - 1)k$ multiplications. The computational effort involved in a LU-factorization of the indefinite matrix $\Gamma_p^{(k)}$ is $O(p^3)$ multiplications. Thus, the effort in preparing the p-step is

$$\text{preparation effort} = \left(\frac{1}{2}p^2 + \frac{15}{2}p - 5\right) k + O(p^3) \text{ multiplications}. \qquad (37)$$

Solving the three systems in (27), (28) and (29) using the LU-factorization requires $3p^2$ multiplications, while the updating of $\mathbf{x}_{k+1}$, $\mathbf{y}_{k+1}$ and $\mathbf{z}_{k+1}$ in Eqs. (24), (25), and (26) requires additional $3pk$ multiplications, and the re-computation of $\gamma^{(k+p)}$ by (5) (for $p > 1$ only) requires k multiplications. Thus, the effort in updating the solution is

$$\text{updating effort} = \begin{cases} 3k & \text{multiplications,} \quad p = 1 \\ (3p+1)k + O(p^2) & \text{multiplications,} \quad p > 1 \end{cases} . \tag{38}$$

However, note that we save p classical Levinson steps which would have involved approximately $3pk$ multiplications. Hence, the computational *overhead* in the updating phase in only $k + O(p^2)$ multiplications for $p > 1$.

Concerning the numerical stability of the look-ahead algorithm, we shall not go into a detailed discussion here, but only mention that the error analysis can be carried out using exactly the same technique as in [14, 15]. In particular, following [14, Thm. 2], for all the intermediate steps of the look-ahead algorithm we can prove that the residual vector corresponding to the computed $\mathbf{x}_{k+p}$ is guaranteed to be small. Hence, if the Toeplitz matrix T_n is well-conditioned and if all ill-conditioned leading principal submatrices of T_n are avoided during the look-ahead algorithm, then the computed solution $\mathbf{x}_n$ is guaranteed to be close to the exact solution. The conclusion is that the look-ahead algorithm for nonsymmetric matrices is weakly stable [9]).

4 The choice of the block size

For numerical stability and efficiency, it is important to decide *when* to perform a p-step and to chose the *correct* value of the block-size p. Our aim is to choose the block size adaptively in order to ensure the best possible accuracy in the computed solution while, at the same time, choosing $p > 1$ only when necessary, because such block-steps are more expensive (as measured in multiplications) than p usual Levinson steps. Our approach is to let the user choose a maximum block size $p_{\max}$ and then, in each step k, the algorithm adaptively chooses the optimal block-size p within the look-ahead range $p = 1, \ldots, p_{\max}$, i.e. the p that minimizes the influence of rounding errors. The perturbation of the solution, due to rounding errors, when solving the system $T_{k+p}\mathbf{x}_{k+p} = \mathbf{b}_{k+p}$ is proportional to the machine precision times $(\psi_{\min}(T_{k+p}))^{-1}$, where $\psi_{\min}(T_{k+p})$ denotes the smallest singular value of T_{k+p} [23, §2.5.3]. Therefore, we want to find p such that $\psi_{\min}(T_{k+p})$ is as large as possible. Let $s_{\min}$ denote the smallest $\psi_{\min}(T_{k+p})$ accepted so far in all the previous steps. In the kth step, we then choose as block size the smallest p for which

$$\psi_{\min}(T_{k+p}) \geq 0.1\, s_{\min}. \tag{39}$$

The reason for comparing with $s_{\min}$ is that if we have already accepted a leading principal submatrix with a smallest singular value equal to $s_{\min}$, then any better conditioned submatrix will not improve the error in the solution significantly. The factor 0.1 is included in (39) to avoid choosing a too large block size p if $\psi_{\min}(T_{k+p})$ is only slightly smaller than $s_{\min}$ such that T_{k+p} is still well-conditioned. If (39) is not satisfied for any $p \leq p_{\max}$, then we choose as block-size the p for which $\psi_{\min}(T_{k+p})$ is maximum, and only in this case do we update the $s_{\min}$. In this way, we are guaranteed in each step to choose the optimally conditioned T_{k+p} within the allowed look-ahead range $p_{\max}$, and the strategy is numerically stable as long as T_n does not have more than $p_{\max} - 1$ consecutive ill-conditioned leading submatrices.

An important side-effect of using $s_{\min}$ in this fashion to keep track of the smallest accepted $\psi_{\min}(T_{k+p})$ is that it provides us with an almost "free" estimate of the condition of the computed solution. Since the accuracy of the *computed* solution depends on the smallest singular value of any T_{k+p} accepted during the look-ahead Levinson algorithm, we can define the *algorithm condition number* as:

$$\kappa_{\text{Levinson}} \equiv \|T_n\|_2 / \min\{\text{accepted } \psi_{\min}(T_{k+p})\} \approx \|T_n\|_2 / s_{\min}, \tag{40}$$

and then we know that the relative error in the computed solution is proportional to κ_{Levinson} times the machine precision. Another important quantity is the *matrix condition number* $\kappa(T_n) \equiv \|T_n\|_2/\psi_{\min}(T_n) \approx \|T_n\|_2/\psi_{\min}(T_n)$ [23, §2.7.2], which measures the sensitivity of the solution $\mathbf{x}$ to perturbations of the right-hand side $\mathbf{b}$. Due to the definition in (40), we always have $\kappa_{\text{Levinson}} \geq \kappa(T_n)$. When both the algorithm condition number and the matrix condition number are not large, then our new Levinson algorithm has computed a numerically stable solution. On the other hand, if $\kappa_{\text{Levinson}} \gg \kappa(T_n)$ then the maximum block-size $p_{\max}$ was too small to "skip over" a sequence of ill-conditioned leading principal submatrices. Notice that our extended Levinson algorithm returns estimates for both the algorithm condition number κ_{Levinson} and the matrix condition number $\kappa(T_n)$ without increasing the complexity of the algorithm.

In order to keep the computational overhead as small as possible, it is very important to compute an estimate of $\psi_{\min}(T_{k+p})$ efficiently. It is well-known that the smallest singular value $\psi_{\min}(\Gamma_p^{(k)}) = \|(\Gamma_p^{(k)})^{-1}\|_2^{-1}$ of the Schur complement $\Gamma_p^{(k)}$ does not give a reliable estimate of the condition of T_{k+p}, see the discussion in [12]. Our numerical experiments confirmed this: in some situations an ill-conditioned T_{k+p} was not reflected by a small $\psi_{\min}(\Gamma_p^{(k)})$, and a p-step—although necessary—was therefore not performed.

As an alternative to using $\psi_{\min}(\Gamma_p^{(k)})$, we will estimate $\psi_{\min}(T_{k+p}) = \|T_{k+p}^{-1}\|_2^{-1}$ directly. The following expression for T_{k+p}^{-1} is obtained by means of Eqs. (17) and (18):

$$T_{k+p}^{-1} = \begin{pmatrix} T_k^{-1} + E_k Y_p (\Gamma_p^{(k)})^{-1} Z_p^T E_k & E_k Y_p (\Gamma_p^{(k)})^{-1} \\ (\Gamma_p^{(k)})^{-1} Z_p^T E_k & (\Gamma_p^{(k)})^{-1} \end{pmatrix}. \tag{41}$$

Since we are only concerned with identifying the ill-conditioned submatrices T_{k+p} in order to "skip over" them, the accuracy of the estimate of $\|T_{k+p}^{-1}\|_2$ is not so important. What is needed is a reliable and efficient means for detecting when $\|T_{k+p}^{-1}\|_2$ becomes large. A lower bound for $\|T_{k+p}^{-1}\|_2$ is given by:

$$\max\{\|T_k^{-1} + E_k Y_p (\Gamma_p^{(k)})^{-1} Z_p^T E_k\|_2, \|E_k Y_p (\Gamma_p^{(k)})^{-1}\|_2, \|(\Gamma_p^{(k)})^{-1} Z_p^T E_k\|_2, \|(\Gamma_p^{(k)})^{-1}\|_2\}. \tag{42}$$

There are no simple lower bounds for the matrix norms in (42), so instead we use the upper bounds

$$\|T_k^{-1} + E_k Y_p (\Gamma_p^{(k)})^{-1} Z_p^T E_k\|_2 \leq \|T_k^{-1}\|_2 + \|Y_p\|_2 \, \|Z_p\|_2 \, \|(\Gamma_p^{(k)})^{-1}\|_2 \tag{43}$$

$$\|E_k Y_p (\Gamma_p^{(k)})^{-1}\|_2 \leq \|Y_p\|_2 \, \|(\Gamma_p^{(k)})^{-1}\|_2 \tag{44}$$

$$\|(\Gamma_p^{(k)})^{-1} Z_p^T E_k\|_2 \leq \|Z_p\|_2 \, \|(\Gamma_p^{(k)})^{-1}\|_2. \tag{45}$$

The norms of Y_p and Z_p can be bounded by means of their numerically largest elements as follows: $\mu_{(Y)} \equiv \max\{|(Y_p)_{ij}|\} \leq \|Y_p\|_2$ and $\mu_{(Z)} \equiv \max\{|(Z_p)_{ij}|\} \leq \|Z_p\|_2$. Finally, due to our block algorithm we know that T_k is well-conditioned so that $\|T_k^{-1}\|_2$ cannot contribute significantly to a large $\|T_{k+p}^{-1}\|_2$, and we can therefore readily ignore $\|T_k^{-1}\|_2$ in (43). This combination of lower and upper bounds leads to the following heuristic estimate for $\psi_{\min}(T_{k+p}) = \|T_{k+p}^{-1}\|_2^{-1}$ which works well in practice:

$$\psi_{\min}(T_{k+p}) \approx \Psi_{k+p} \equiv \psi_{\min}(\Gamma_p^{(k)}) / \max\{1, \mu_{(Y)}, \mu_{(Z)}, \mu_{(Y)}\mu_{(Z)}\}. \tag{46}$$

For $p = 1$, $\psi_{\min}(\Gamma_k^{(k)})$ is simply $|\gamma^{(k)}|$, and for $p > 1$ the smallest singular value $\psi_{\min}(\Gamma_k^{(k)})$ can be estimated by any efficient condition number estimator in $O(p^2)$ multiplications. Determination of the numerically largest element in Y_p and Z_p requires $2kp$ comparisons at the kth stage, i.e. $O(n^2)$ comparisons for the complete algorithm. We do not think it is possible to get good estimates of the smallest singular values with a lower complexity than this.

A nice consequence of combining this strategy for estimating $\|T_{k+p}^{-1}\|_2$ with the block-size decision based on (39), is that a specific threshold for detection of ill-conditioned submatrices is *not* required. This is important both from a numerical and a practical (i.e., a user's) point of view. In [14] we discussed the numerical performance of the above strategy for estimating $\|T_{k+p}^{-1}\|_2$. Our tests for the nonsymmetric case reported in [25] lead to exactly the same conclusion, namely that the estimate (46) is always within a factor of about ten from the true norm. We shall therefore not pursue this any further here.

5 Practical details

An important detail in a practical implementation of our algorithm is the fact that two block steps may follow immediately after each other. Let p' denote the size of the *previous* p'-step, and let k' denote the corresponding dimension of that leading submatrix $T_{k'}$, such that $k = k' + p'$. In such a situation, neither $\gamma^{(k-1)}$ nor $\mathbf{y}_{k-1}$ or $\mathbf{z}_{k-1}$ is available because we skipped the step $k-1$ and the ill-conditioned matrix T_{k-1} when going from $T_{k'}$ to $T_k = \begin{pmatrix} T_{k'} & E_{k'}R_{p'} \\ S_{p'}^T E_{k'} & T_{p'} \end{pmatrix}$. Hence, we cannot use Eqs. (31) and (32) to compute the vectors $\mathbf{y}_{k,i}$ and $\mathbf{z}_{k,i}$ because we cannot produce the vectors $\mathbf{g}_k$ and $\mathbf{h}_k$ in (33) (which are functions of $\gamma^{(k-1)}$, $\mathbf{y}_{k-1}$ and $\mathbf{z}_{k-1}$). However, in this special situation there is another way to proceed. The idea is to compute $\mathbf{y}_{k,1}$ by updating the vector $\mathbf{y}_{k',1}$ and, if $p > 2$, then compute $\mathbf{g}_k$ such that $\mathbf{y}_{k,i}$ can be computed by means of (31) (and similarly for $\mathbf{h}_k$ and $\mathbf{z}_{k,i}$):

Theorem 2 *In case of two consecutive extended Levinson steps with $k = k' + p'$, the vectors $\mathbf{y}_{k,1}$ and $\mathbf{z}_{k,1}$ are given by*

$$\mathbf{y}_{k,1} = \begin{pmatrix} \mathbf{y}_{k',1} \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} E_{k'} Z_{p'} \\ I_{p'} \end{pmatrix} \mathbf{a}_{(y)}, \qquad \mathbf{z}_{k,1} = \begin{pmatrix} \mathbf{z}_{k',1} \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} E_{k'} Y_{p'} \\ I_{p'} \end{pmatrix} \mathbf{a}_{(z)} \tag{47}$$

where the vectors $\mathbf{a}_{(y)}$ *and* $\mathbf{a}_{(z)}$ *are the solutions to the systems*

$$(\Gamma_{p'}^{(k')})^T \mathbf{a}_{(y)} = - \begin{pmatrix} \rho_{k'+2} \\ \vdots \\ \rho_{k+1} \end{pmatrix} - R_{p'}^T E_{k'} \mathbf{y}_{k',1} \tag{48}$$

$$\Gamma_{p'}^{(k')} \mathbf{a}_{(z)} = - \begin{pmatrix} \sigma_{k'+2} \\ \vdots \\ \sigma_{k+1} \end{pmatrix} - S_{p'}^T E_{k'} \mathbf{z}_{k',1}. \tag{49}$$

If $p > 2$*, then the remaining vectors* $\mathbf{y}_{k,i}$ *and* $\mathbf{z}_{k,i}$*,* $i = 2, \ldots, p-1$ *can be computed from (31) and (32) with the vectors* $\mathbf{g}_k$ *and* $\mathbf{h}_k$ *given by*

$$\mathbf{g}_k = c_1^{-1} (\mathbf{y}_{k,1} - \Delta_k y_k + (\mathbf{y}_k)_1 \mathbf{y}_k), \qquad \mathbf{h}_k = d_1^{-1} (\mathbf{z}_{k,1} - \Delta_k z_k + (\mathbf{z}_k)_1 \mathbf{z}_k) \tag{50}$$

in which c_1 *and* d_1 *are the first component of the vectors* $\mathbf{c}$ *and* $\mathbf{d}$ *defined in (34).*

Proof. We recall that the vector $\mathbf{z}_{k,1}$ is the solution to the system $T_k \mathbf{z}_{k,1} = -(\sigma_2, \ldots, \sigma_{k+1})^T = -(\mathbf{s}_{k',1}^T, \sigma_{k'+2}, \ldots, \sigma_{k+1})^T$. In the previous p'-step we have already computed $\mathbf{z}_{k',1}$ satisfying $T_{k'} \mathbf{z}_{k',1} = -\mathbf{s}_{k',1}$. Since the first k' components of these two right-hand sides are identical, it is immediately clear that we can use an extended Levinson-step as described in Section 3, Eqs. (24)–(29), to obtain $\mathbf{z}_{k,1}$ with $\mathbf{a}_{(z)}$ given by (49). The expression for $\mathbf{y}_{k,1}$ is proved in a similar manner, using that $T_k^T \mathbf{y}_{k,1} = -(\mathbf{r}_{k',1}^T, \rho_{k'+2}, \ldots, \rho_{k+1})^T$ and that $T_{k'}^T \mathbf{y}_{k',1} = -\mathbf{r}_{k',1}$. Eq. (50) follows immediately from Eqs. (31) and (32) with $i = 1$. □

Hence, all we need to do is to save the matrices $Y_{p'}$ and $Z_{p'}$ and the factorization of $\Gamma_{p'}^{(k')}$ from the previous p'-step. Then the vectors $\mathbf{y}_{k,i}$ and $\mathbf{z}_{k,i}$, $i = 1, \ldots, p-1$ can be computed recursively using (31) and (32) and Theorem 2. The effort in computing $\mathbf{y}_{k,1}$, $\mathbf{z}_{k,1}$, $\mathbf{g}_k$ and $\mathbf{h}_k$ this way is only about $4(p'+1)k + O((p')^2)$ multiplications.

In the special situation where we have looked $p_{\max}$ steps ahead, but chosen a $p < p_{\max}$, we could in principle update the remaining $\mathbf{y}_{k',i}$ and $\mathbf{z}_{k',i}$, $i = p+1, \ldots, p_{\max}$ to obtain some of the new vectors $\mathbf{y}_{k,i}$ and $\mathbf{z}_{k,i}$. This is, however, not reliable in general because some of these remaining $\mathbf{y}_{k',i}$ and $\mathbf{z}_{k',i}$ might correspond to ill-conditioned leading submatrices occurring after the current T_{k+p}. Thus, we prefer to recompute all the $\mathbf{y}_{k,i}$ and $\mathbf{z}_{k,i}$ by means of Theorem 2. The complete look-ahead Levinson algorithm, including condition estimation, is summarized in Fig. 1.

A second detail, which is but a minor one, is that the very first leading submatrices $T_1, T_2, \ldots$ of T_n may be ill-conditioned. If this is the case, we cannot perform a p-step, simply because we have no starting vectors to update. Instead, we compute the solutions $\mathbf{x}_p$, Y_p and Z_p by a standard linear-equation solver, such as LU-factorization with pivoting, ignoring the Toeplitz structure of T_p (notice that we need the full matrices Y_p and Z_p because the immediately following step may also be a block step).

We conclude this section by giving upper and lower bounds for the computational effort involved in the new algorithm. In the worst case, for each k we will look $p_{\max}$ steps ahead and choose $p = 1$, thus spending a lot of effort on "unnecessary" looking ahead. This will happen, for example, if T_n is an ill-conditioned positive definite matrix for which $\psi_{\min}(T_i) < 0.1\, \psi_{\min}(T_{i-1})$ for all $i = 2, \ldots, n$. According to Eqs. (37) and (38), and taking into account the overhead in estimating $\psi_{\min}(T_{k+p})$ as well as the overhead in Theorem 2, each such step requires about

$$\sum_{p=1}^{p_{\max}} \left(\frac{1}{2}p^2 + \frac{15}{2}p - 5\right) k + 3p_{\max}k + 4(p_{\max}+1)k =$$

$$\left(\frac{p_{\max}^3}{6} + 4p_{\max}^2 + \frac{65}{6}p_{\max} + 4\right) k \text{ multiplications.}$$

Summing over all k we then obtain the following approximate expression for the very pessimistic upper bound

$$\text{maximum effort} = \frac{1}{2}\left(\frac{p_{\max}^3}{6} + 4p_{\max}^2 + \frac{65}{6}p_{\max} + 4\right) n^2 \text{ multiplications.} \qquad (51)$$

For $p_{\max}$ equal to 2, 3 and 4 this amounts to $21.5n^2$, $38.5n^2$ and $61n^2$ multiplications, respectively. As we shall see in the next section, we never obtain this upper bound in practice. The *minimum* computational effort corresponds to matrices where all the $\psi_{\min}(T_{k+1})$ form an increasing sequence, such that only 1-steps are performed, and we only look one step ahead in each stage. For such matrices, Theorem 2 is never used, and Eqs. (37) and (38) lead to $6k$ multiplications in each stage, thus giving a total computational effort of:

$$\text{minimum effort} = 3n^2 \text{multiplications.} \qquad (52)$$

Notice that this is identical to the complexity of the classical Levinson algorithm; i.e., for such matrices there is no overhead. For a typical well-conditioned nonsymmetric Toeplitz matrix, neither the minimum nor the maximum bounds apply, but our experience is that the look-ahead algorithm typically takes a few 2-steps and rarely p-steps with $p > 2$, such that the average computational effort is not much larger than $3n^2$ multiplications.

6 Numerical results

In this section we present some numerical results from applying our look-ahead Levinson algorithm to a series of test problems. These tests were carried out in Matlab [32]. Fortran implementations of the look-ahead Levinson algorithms are also available [25].

We first tested our algorithm on three small nonsymmetric Toeplitz matrices from [36] as listed in Table 1. Also shown in Table 1 are the smallest singular value $\psi_{\min}(T_{i+1})$ of all the leading principal submatrices, and the estimate Ψ_{i+1} computed by means of (46). The table demonstrates how well this estimate is able to track the smallest singular value of all the leading principal submatrices—even when there is a rapid change in the size of $\psi_{\min}(T_{i+1})$. We see that these three matrices require block steps of size 2, 2 and 6, respectively, to skip over the ill-conditioned submatrices. We generated right-hand sides $\mathbf{b}$ to these systems such that the true solution is $\mathbf{x} = (1, 1, \ldots, 1)^T$, and we measure the accuracy of the computed solution $\tilde{\mathbf{x}}$ by means of the relative solution error ρ, defined as

$$\rho \equiv \|\tilde{\mathbf{x}} - \mathbf{x}\|_2 / \|\mathbf{x}\|_2. \qquad (53)$$

Sweet-1				
i	ρ_i	σ_i	$\psi_{\min}(T_{i+1})$	Ψ_{i+1}
0	4		4.00	4.00
1	8	6	2.88	2.67
2	1	$\frac{71}{16}+\epsilon_1$	$3.40\cdot 10^{-8}$	$5.00\cdot 10^{-8}$
3	6	5	0.71	0.23
4	2	3	0.93	0.23
5	3	1	0.79	1.10

Sweet-2				
i	ρ_i	σ_i	$\psi_{\min}(T_{i+1})$	Ψ_{i+1}
0	8		8.00	8.00
1	4	4	4.00	6.00
2	1	$-34+\epsilon_2$	$1.02\cdot 10^{-14}$	$1.39\cdot 10^{-14}$
3	6	5	3.41	3.90
4	2	3	4.51	4.34
5	3	1	3.12	4.69

Sweet-3				
i	ρ_i	σ_i	$\psi_{\min}(T_{i+1})$	Ψ_{i+1}
0	5		5.00	5.00
1	-1	1	5.10	5.20
2	6	-3	5.09	7.86
3	2	12.755	$1.66\cdot 10^{-5}$	$3.73\cdot 10^{-5}$
4	5.697	-19.656	$1.16\cdot 10^{-5}$	$7.97\cdot 10^{-6}$
5	5.850	28.361	$1.78\cdot 10^{-5}$	$7.52\cdot 10^{-6}$
6	3	-7	$8.79\cdot 10^{-6}$	$2.52\cdot 10^{-6}$
7	-5	-1	$3.98\cdot 10^{-5}$	$7.74\cdot 10^{-6}$
8	-2	2	0.23	0.05
9	-7	1	0.35	0.43
10	1	-6	4.03	0.63
11	10	1	1.69	5.22
12	-15	-0.5	3.32	5.26

Table 1: Three test matrices from [36], with $\epsilon_1 = 5\cdot 10^{-8}$ and $\epsilon_2 = 5\cdot 10^{-13}$.

	accuracy ρ		multiplications		
	classical	look-ahead	classical	look-ahead	overhead
Sweet-1	$3.02\cdot 10^{-8}$	$1.08\cdot 10^{-15}$	95	207	118 %
Sweet-2	$5.51\cdot 10^{-2}$	$3.27\cdot 10^{-16}$	95	157	65 %
Sweet-3	$4.01\cdot 10^{-10}$	$3.49\cdot 10^{-14}$	480	859	79 %

Table 2: The relative solution error ρ and the required number of multiplications for the three small test matrices from Table 1.

n	accuracy ρ		multiplications		
	classical	look-ahead	classical	look-ahead	overhead
15	$4.89 \cdot 10^{-3}$	$5.99 \cdot 10^{-16}$	644	1018	58 %
30	$2.99 \cdot 10^{-3}$	$5.38 \cdot 10^{-15}$	2639	3918	48 %
60	$3.88 \cdot 10^{-3}$	$4.95 \cdot 10^{-14}$	10679	15343	43 %
120	$8.03 \cdot 10^{-3}$	$9.16 \cdot 10^{-14}$	42959	60693	41 %

Table 3: Numerical results for shifted KMS test matrices of order n given by Eq. (54).

n	classical	look-ahead	overhead
15	644	805	25.0 %
30	2639	3030	14.8 %
60	10679	11556	8.3 %
120	42959	44821	4.3 %

Table 4: Comparison of the number of multiplications of the classical Levinson algorithms with the average number of multiplications for the look-ahead Levinson algorithm, for matrices with all leading principal submatrices being well-conditioned.

Table 2 compares the accuracy and computational effort of our look-ahead algorithm with the classical Levinson algorithm for the three matrices from Table 1. The classical Levinson algorithm gives poor results, especially for the second matrix. Our look-ahead Levinson algorithm is able to compute the solution to almost full machine precision, which is approximately 10^{-16}, at the cost of some computational overhead. For these small matrices, the overhead is significant, but the relative overhead quickly decreases as the order of the matrices increases.

The next test matrices are the symmetric shifted KMS matrices from [38], given by

$$\rho_0 = 10^{-14}, \qquad \rho_i = \sigma_i = (1/2)^{i-1}, \quad i = 1, \ldots, n-1 \tag{54}$$

for which every third leading principal submatrix T_k, $k = 1, 4, 7, \ldots$ is numerically singular (they would be exactly singular if $\rho_0 = 0$, but then the classical Levinson algorithm would break down due to a division by zero). Again, we generated right-hand sides such that the true solution consists of ones, and we compare the classical and the look-ahead Levinson algorithms in Table 3. Our primary aim is to demonstrate the accuracy of the look-ahead Levinson algorithm, so we did not take into account the symmetry of the problem—numerical results for the special symmetric version of the look-ahead Levinson are given in [14]. Again, at the cost of some overhead, we can compute the solutions to almost full machine precision. The overhead decreases with the order n because a great part of it is linear in n. We notice that these test matrices are very demanding for our algorithm—every third step of the look-ahead Levinson algortihm is a 2-step—so we find that the overhead is quite acceptable.

Next, we applied our algorithm to random nonsymmetric test matrices for which all the leading principal submatrices are well-conditioned, and the average computational overhead is listed in Table 4. Again, we see that the overhead decreases with the order n, and the overhead is less than 10 % for $n > 60$.

Our last tests were done on a class of nonsymmetric test matrices generated such that at least one leading principal submatrix is ill-conditioned, with a controlled degree of ill-conditioning. In order to generate such test matrices, we first generate a random positive nonsymmetric Toeplitz matrix $\bar{T}_n$; i.e., all the elements of $\bar{T}_n$ are positive. Let $\lambda(\bar{T}_k)$ denote the numerically smallest real eigenvalue of the leading principal submatrix $\bar{T}_k$ of $\bar{T}_n$ (due to Perron's theorem [34, p. 216], a positive matrix always has at least one real eigenvalue, which is identical to the spectral radius of the matrix). Thus, for arbitrary k we can set our test matrix to:

$$T_n = \bar{T}_n - (\lambda(\bar{T}_k) - \delta)I_n, \tag{55}$$

where $\delta \geq 0$ is a real parameter, and then we know that the leading principal submatrix T_k of T_n will have an eigenvalue equal to δ. Since the numerically smallest eigenvalue is bounded below by the smallest singular value [23, p. 349], we are guaranteed that the smallest singular value of T_k satisfies $\psi_{\min}(T_k) \leq \delta$, and in this way we can control the ill-conditioning of T_k by means of δ (we emphasize that there may be other ill-conditioned submatrices, too). Matrices of the particular form (55) are likely to occur in eigenvalue computations by means of inverse iterations [16, 38]. Our tests were carried out with the following values of δ,

$$\delta = 0, \quad 10^3\mathbf{u}, \quad 10^6\mathbf{u}, \quad 10^9\mathbf{u}, \quad 1 \tag{56}$$

where $\mathbf{u} = 2.22 \cdot 10^{-16}$ denotes the machine precision, and with the following values of $p_{\max}$:

$$p_{\max} = 1 \text{ (classical Levinson)}, \quad 2, \quad 3, \quad 4. \tag{57}$$

For each δ we generated 300 test matrices: 100 of order $n = 16$, 100 of order $n = 32$, and 100 of order $n = 64$. We chose $k = n/2$ and the right-hand side $\mathbf{b}$ such that the exact solution is $\mathbf{x} = (1, \ldots, 1)^T$.

Fig. 2 illustrates the typical performance of our algorithm by showing four histograms of the relative error ρ as defined in Eq. (53). These four tests were carried out with test matrices with $n = 64$, $\delta = 10^3\mathbf{u}$, and $p_{\max}$ equal to 1, 2, 3, and 4. For $p_{\max} = 1$ (classical Levinson), ρ is never smaller than 10^{-3} due to the ill-conditioned submatrix T_{32}. On the other hand, for $p_{\max} > 1$ the relative error ρ is never larger than 10^{-9}, and typically it is even smaller, about 10^{-12}.

For each combination of δ and $p_{\max}$, we computed the maximum relative error $\rho_{\max} \equiv \max\{\rho\}$ over all 300 test matrices. Fig. 3 shows a plot of $\rho_{\max}$ as a function of δ and $p_{\max}$. For $p_{\max} = 1$ (classical Levinson), the maximum relative error $\rho_{\max}$ is approximately proportional to δ^{-1} as expected, while $\rho_{\max}$ is almost independent of δ for $p_{\max} > 1$. We see that the look-ahead Levinson algorithm is *always* more accurate than the classical algorithm, even in the case $\delta = 1$ (where typically all the leading submatrices all well-conditioned). For $\delta < 1$, the look-ahead Levinson algorithm is actually much more accurate than the classical Levinson algorithm, with a maximum relative error of about 10^{-10} almost independent of δ.

In Fig. 4 we show the average number of p-steps required in our experiments. The average number of 2-steps is only about 1.5, which is the 'price' we have to pay for ensuring that we can handle these indefinite Toeplitz matrices. The average number of 3-steps and 4-steps are much smaller, less than 0.4.

Finally, in Table 5 we compare the number of multiplications required in our new algorithm with those of the classical Levinson algorithm. Each entry in the table shows,

n	$p_{\max}$ 1		2		3		4	
16	735	735	918	1107	981	1292	1071	1563
32	3007	3007	3381	3792	3518	4215	3698	4800
64	12159	12159	12914	14014	13223	15242	13587	16526

Table 5: The average (left) and maximum (right) number of multiplications.

for each combination of n and $p_{\max}$, the average number of multiplications over all 300 test matrices (left) and the maximum number of multiplications (right) over the same 300 test matrices. We do not show the dependency on δ because, due to the uniform way in which we generate the test matrices, the average multiplication count for $p_{\max} > 1$ is almost independent of δ—although somewhat smaller for $\delta = 1$ than for $\delta < 1$—and completely independent of δ for $p_{\max} = 1$ (classical Levinson). We notice that the multiplication count for our new method with $p_{\max} \leq 4$ is never more than twice the multiplication count for the classical Levinson algorithm, the average ratio in fact being only 1.2. Also, note that the average ratio decreases as n increases, from 1.35 (for $n = 16$) to 1.09 (for $n = 64$). Hence, for this particular class of matrices—which are likely to occur in eigenvalue computations by means of inverse iterations [16, 38]—the complexity of our new algorithm, including condition estimation, is about 20% larger than that of the classical Levinson algorithm.

Acknowledgements

We would like to thank the referees for their very detailed comments and their constructive criticism of our paper.

References

[1] G. S. Ammar & W. B. Gragg, *Superfast solution of real positive definite Toeplitz systems*, SIAM J. Matrix Anal. Appl. **9** (1988), 61–76.

[2] O. B. Arushanian, M. K. Samarin, V. V. Voevodin, E. E. Tyrtyshnikov, B. S. Garbow, J. M. Boyle, W. R. Cowell & K. W. Dritz, *The Toeplitz package users' guide*, Report ANL-83-16, Argonne National Laboratory, Illinois, 1983.

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

[4] R. R. Bitmead & B. D. O. Anderson, *Asymptotically fast solution of Toeplitz and related systems of linear equations*, Lin. Alg. Appl. **34** (1980), 103–116.

[5] R. P. Brent, F. G. Gustavson & D. Y. Y. Yun, *Fast solution of Toeplitz systems of equations and computation of Padé approximants*, J. Algorithms **1** (1980), 259–295.

[6] R. P Brent, H. T. Kung & F. T. Luk, *Some linear-time algorithms for systolic arrays*; in R. E. A. Mason (Ed.), *Information Processing 83*, North-Holland, 1983, pp. 865–876.

[7] R. P. Brent, *Old and new algorithms for Toeplitz systems*; in F. T. Luk (Ed.), *Advanced Algorithms and Architectures for Signal Processing III*, Proc. SPIE **975** (1988), 2–9.

[8] J. R. Bunch, *Stability of methods for solving Toeplitz systems of equations*, SIAM J. Sci. Stat. Comput. **6** (1985), 349–364.

[9] J. R. Bunch, *The weak and strong stability of algorithms in numerical linear albegra*, Lin. Alg. Appl. **88/89** (1987), 49–66.

[10] J. A. Cadzow & T.-C. Chen, *Application of Toeplitz eigenvalue decomposition in digital filter synthesis*, IEEE **ASSP-37** (1989), 655–664.

[11] R. H. Chan & G. Strang, *Toeplitz equations by conjugate gradients with circulant preconditioner*, SIAM J. Sci. Stat. Comput. **10** (1989), 104–119.

[12] T. F. Chan, *On the existence and computation of LU-factorizations with small pivots*, Math. Comp. **42** (1984), 535–547.

[13] T. F. Chan, *An optimal circulant preconditioner for Toeplitz systems*, SIAM J. Sci. Stat. Comput. **9** (1988), 766–771.

[14] T. F. Chan & P. C. Hansen, *A look-ahead Levinson algorithm for indefinite Toeplitz systems*, SIAM J. Matrix Anal. Appl., to appear.

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

[16] G. Cybenko & C. F. Van Loan, *Computing the minimum eigenvalue of a symmetric positive definite Toeplitz matrix*, SIAM J. Sci. Stat. Comput. **7** (1987), 123–131.

[17] F. R. de Hoog, *A new algorithm for solving Toeplitz systems of equations*, Lin. Alg. Appl. **88/89** (1987), 122-138.

[18] P. Delsarte, Y. V. Genin & Y. G. Kamp, *A generalization of the Levinson algorithm for Hermitian Toeplitz matrices with any rank profile*, IEEE **ASSP-33** (1985), 964–971.

[19] C. J. Demeure & L. L. Scharf, *Linear statistical models for stationary sequences and related algorithms for Cholesky factorization of Toeplitz matrices*, IEEE **ASSP-35** (1987), 29–42.

[20] J. Durbin, *The fitting of time-series models*, Rev. Inst. Internat. Statist. **28** (1960), 233–244.

[21] T. S. Durrani & K. C. Sharman, *Extraction of an eigenvector oriented spectrum from the MESA coefficients*, IEEE **ASSP-30** (1982), 649–651.

[22] B. Friedlander, *Instrumental variable models for ARMA spectral estimation*, IEEE **ASSP-31** (1983), 404–415.

[23] G. H. Golub & C. F. Van Loan, *Matrix Computations*, 2. Edition, Johns Hopkins University Press, 1989.

[24] M. J. C. Gover & S. Barnett, *Inversion of Toeplitz matrices which are not strongly non-singular*, IMA J. Numer. Anal. **5** (1985), 101-100.

[25] P. C. Hansen & T. F. Chan, *Fortran subroutines for general Toeplitz systems*, CAM Report 90-21, Dept. of Mathematics, UCLA; submitted to ACM Trans. Math. Software.

[26] M. H. Hayes & M. A. Clements, *An efficient algorithm for computing Pisarenko's harmonic decomposition using Levinson's recursion*, IEEE **ASSP-34** (1986), 485–491.

[27] Y. H. Hu & S.–Y. Kung, *Toeplitz eigensystem solver*, IEEE **ASSP-33** (1985), 1264–1271.

[28] G. Gueguen, *Linear prediction in the singular case and the stability of eigen models*; in Proc. 1981 IEEE Int. Conf. Acoust., Speech, Signal Processing, Atlanta, GA, 881–885.

[29] T. Kailath *A view of three decades of linear filtering theory*, IEEE **IT-20** (1974), 145–181.

[30] S. Y. Kung, *A Toeplitz approximation method and some applications*; in Proc. 1981 Int. Symp. Math. Theory of Networks and Systems, Santa Monica, CA, 262–266.

[31] N. Levinson, *The Wiener rms (root means square) error criterion in filter design and prediction*, J. Math. Phys. **25** (1947), 261–278.

[32] C. B. Moler, J. N. Little & S. Bangert, *Pro-Matlab User's Guide*, The MathWorks, Mass., 1987.

[33] M. Morf, *Doubling algorithms for Toeplitz and related equations*, Proc. IEEE Internat. Conf. on Acoustics, Speech and Signal Processing, Denver, Colorado (April 1980), 954–959.

[34] J. M. Ortega, *Matrix Theory*, Plenum Press, 1987.

[35] G. Strang, *A proposal for Toeplitz matrix calculations*, Stud. Appl. Math. **74** (1986), 171–176.

[36] D. R. Sweet, *The use of pivoting to improve the numerical performance of Toeplitz solvers*; in J. M. Speiser (Ed.), *Advanced Algorithms and Architectures for Signal Processing*, Proc. SPIE **696** (1986), 8–18.

[37] W. F. Trench, *An algortihms for the inversion of finite Toeplitz matrices*, J. SIAM **12** (1964), 515–522.

[38] W. F. Trench, *Numerical solution of the eigenvalue problem for Hermitian Toeplitz matrices*, SIAM J. Matrix Anal. Appl. **10** (1989), 135–146.

[39] D. W. Tufts & R. Kumaresan, *Estimation of frequencies of multiple sinusoids: making linear prediction perform like maximum likelihood*, Proc. IEEE **70** (1982), 975–984.

[40] E. E. Tyrtyshnikov, *New cost-efficient and fast algortihms for special classes of Toeplitz systems*, Sov. J. Numer. Anal. Math. Modelling **3** (1988), 63–76.

```
init.:   let $T_k$ = best conditioned $T_i$, $i = 1 : p_{\max}$
         solve $T_k \mathbf{x}_k = \mathbf{b}_k$, $T_k^T Y_k = -R_k$ and $T_k Z_k = -S_k$ e.g. by LU-factorization
         the first column of $Y_k$ and $Z_k$ is $\mathbf{y}_k$ and $\mathbf{z}_k$, respectively
         $\gamma^{(k)} = \rho_0 + \mathbf{s}_k^T \mathbf{y}_k$
         $s_{\min} = \psi_{\min}(T_k)$
loop:    for $i = k : n-1$
             for $p = 1 : \min(p_{\max}, n-k)$
                 if last step was a block-step
                     set up $Y_p$ and $Z_p$ using Theorem 2
                 else
                     set up $Y_p$ and $Z_p$ using Theorem 1
                 endif
                 set up the systems in (27)–(29)
                 estimate $\psi_{\min}(T_{k+p})$ by (46)
                 if $\psi_{\min}(T_{k+p}) > 0.1\, s_{\min}$ goto update
             end
             let $T_{k+p}$ = best conditioned $T_{k+i}$, $i = 1 : p_{\max}$
             $s_{\min} = \psi_{\min}(T_{k+p})$
update:      solve (27)–(29) for updates $\mathbf{a}_p^{(k)}$, $\mathbf{e}_p^{(k)}$ and $\mathbf{f}_p^{(k)}$ e.g. by LU-factorization
             update $\mathbf{x}_{k+p}$, $\mathbf{y}_{k+p}$ and $\mathbf{z}_{k+p}$ by (24)–(26)
             if $p = 1$
                 $\gamma^{(k+1)} = (1 - \eta^{(k)}\phi^{(k)})\gamma^{(k)}$
             else
                 $\gamma^{(k+p)} = \rho_0 + \mathbf{s}_{k+p}^T \mathbf{y}_{k+p}$
             endif
         end
         $\kappa_{\text{Levinson}} = \| T_n \|_2 / s_{\min}$
         $\kappa(T_n) = \| T_n \|_2 / \psi_{\min}(T_n)$
```

Figure 1. The look-ahead Levinson algorithm.

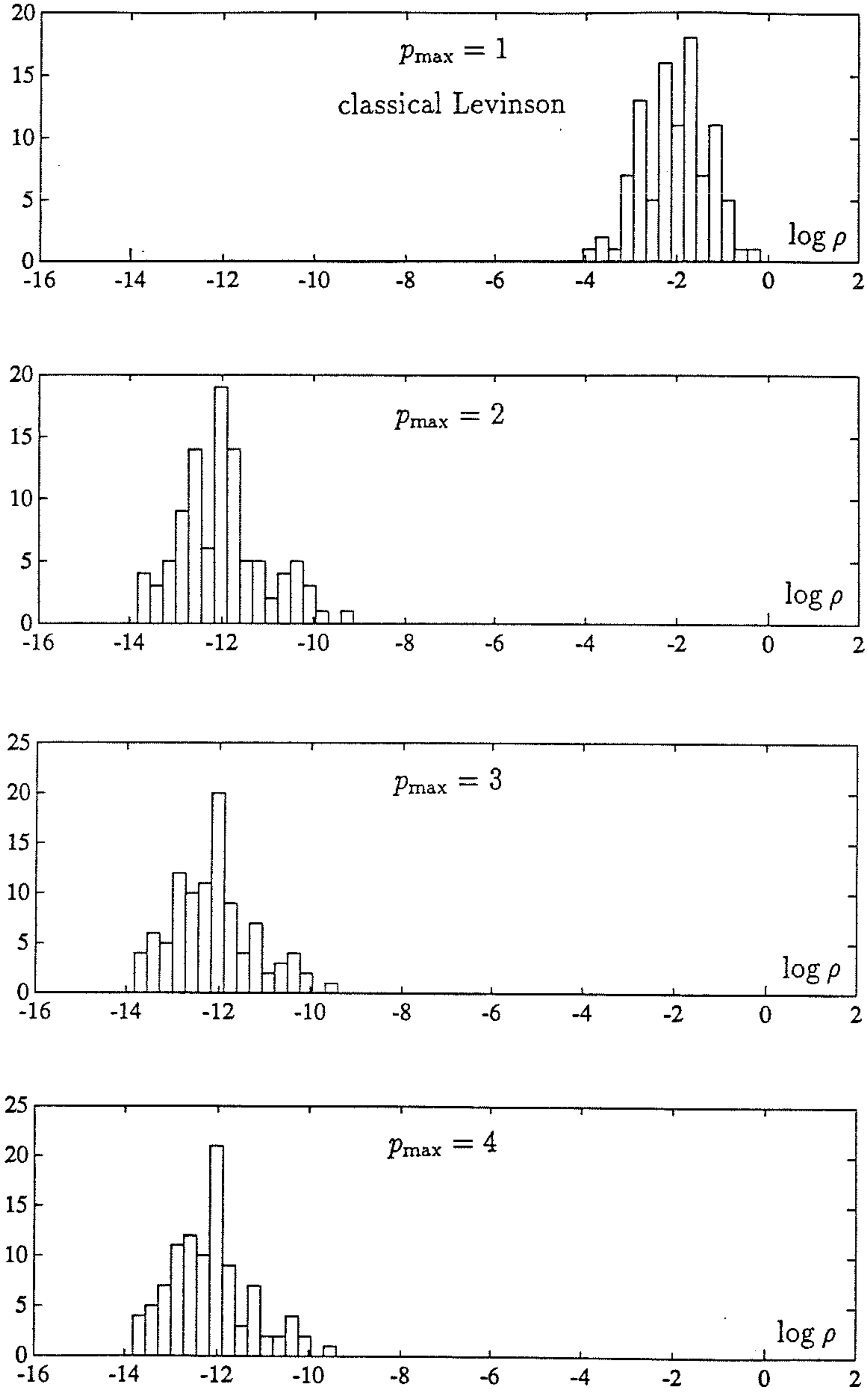

Figure 2. Histograms of the relative error ρ for 100 test matrices with $n = 64$ and $\delta = 10^3\mathbf{u}$, using $p_{\max} = 1, 2, 3$ and 4.

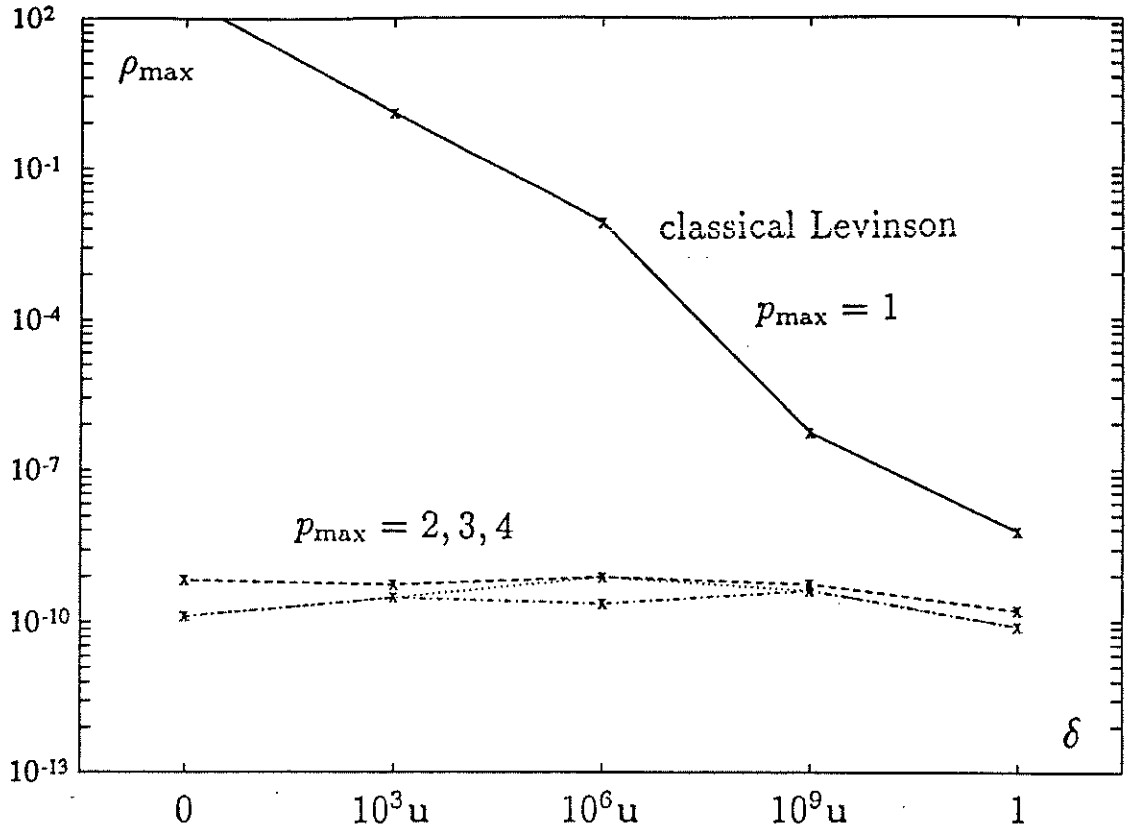

Figure 3. The masimum relative error of the computed solutions.

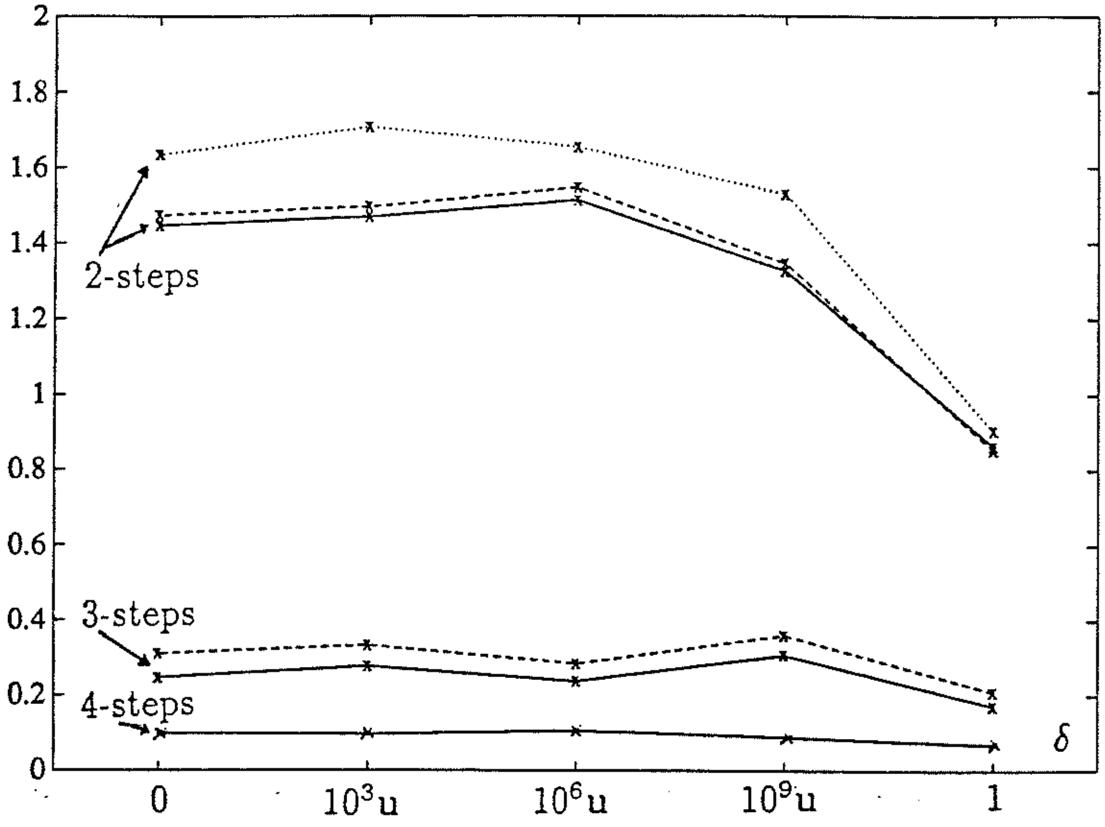

Figure 4. The average number of p-steps for $p_{\max} = 2$ (dotted line), $p_{\max} = 3$ (dashed lines), and $p_{\max} = 4$ (solid lines).