Fortran Subroutines for General Toeplitz Systems*

Per Christian Hansen† Tony F. Chan‡

September 13, 1990

Abstract

This paper presents Fortran 77 implementations of the extended Levinson algorithm of Chan & Hansen [7, 8] for solving symmetric indefinite and general Toeplitz systems. The algorithms are numerically stable for all Toeplitz matrices that do not have many *consecutive* ill-conditioned leading principal submatrices, and also produce estimates of the algorithm and matrix condition numbers. In contrast, the classical Levinson algorithm is only guaranteed to be numerically stable for symmetric positive definite Toeplitz matrices, and no condition estimate is produced.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*Linear systems*; G.4 [**Mathematical Software**]—*Algorithm analysis.*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Toeplitz systems, Levinson's algorithm, condition estimation

1 Introduction

The availability of efficient and reliable software for solving general Toeplitz systems is essential in a variety of applications, such as signal and image processing, control theory, statistics, and the numerical solution of integral equations. Surveys of Toeplitz systems and their applications can be found in [5, 11, 12, 18]. Fast Toeplitz algorithms that utilize the special structure of the Toeplitz matrix, and therefore require only $O(n^2)$ operations (where n is the order of the matrix), have been around for many years. Recently, so-called superfast algorithms with a complexity of $O(n \log n)$ operations have also appeared, see for example the excellent survey paper by Brent [3]. Fortran software for solving

*Both authors are supported by a NATO Collaborative Research Grant 5-2-05/RG900098. The first author is also supported by the Army Research Office under contract DAAL03-88-K-0085, by the Dept. of Energy uncer contract DE-FG-03-87-ER-25037, and by the National Science Foundation under contract FDP-NSF-ASC-9003002

†UNI•C (Danish Computing Center for Research and Education), Building 305, Technical University of Denmark, DK-2800 Lyngby, Denmark. Email: neupch@uts.uni-c.dk.

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

Toeplitz systems seems to be limited to implementations of the *Levinson algorithm* [20], which is the standard fast Toeplitz algorithm. Subroutines are available from the Argonne Toeplitz package [1], from the IMSL Math/Library [17, §§1.1.13–1.1.14], and from the book *Numerical Recipes* [21, §2.8].

Unfortunately, all the above-mentioned algorithms are only guaranteed to be numerically stable for Toeplitz matrices that are symmetric and positive definite [5]. In some applications, this requirement is indeed satisfied. However, there are also many applications in which the matrix is not symmetric or not guaranteed to be positive definite. One such application is the eigenfilter problem in signal processing [6, 9, 23]. For these general Toeplitz systems, all the above-mentioned algorithms may break down due to the presence of a singular submatrix, or—more seriously—they can produce arbitrarily inaccurate solutions in finite precision arithmetic. Moreover, none of these algorithms provide an estimate of the accuracy of the computed solution, so there is no simple means for detecting an inaccurate solution.

For these reasons, it is necessary to develop fast and numerically stable Toeplitz algorithms which are able to solve general Toeplitz systems and, in addition, able to produce inexpensive estimates of the accuracy of the computed solution. Work along this line is described in the papers [11, 15, 22, 24], but none of these algorithms incorporate numerically reliable techniques for detecting the potential numerical instability. Chan & Hansen [7, 8] recently presented an efficient and reliable extension of the classical Levinson algorithm which avoids the numerical instabilities. Their algorithm is based on a reliable and inexpensive means for detecting any numerical instability caused by an ill-conditioned leading principal submatrix, combined with an efficient scheme for 'skipping over' the instability. They proved that the extended Levinson algorithm is numerically stable for a much broader class of Toeplitz matrices than the symmetric and positive definite ones, namely those general Toeplitz matrices that do not have many consecutive ill-conditioned leading principal submatrices [7, Theorem 2]. The complexity of the new algorithm is still $O(n^2)$, and experiments show that for nonsymmetric Toeplitz matrices with only one ill-conditioned leading principal submatrix, the new algorithms requires about 20 % more multiplications than the classical Levinson algorithm. This computational overhead includes the necessary monitoring of the condition number of all the leading principal submatrices, whose by-product is a reliable accuracy estimated for the computed solution.

The purpose of this paper is to present Fortran 77 implementations of the extended Levinson algorithm. We have developed subroutines for both symmetric and nonsymmetric Toeplitz matrices, and they all include computation of the above-mentioned accuracy estimate. The algorithms are tested numerically on a number of test-matrices. In Sections 2 and 3 we briefly summarize the extended Levinson algorithm. Next, in Section 4 we introduce the Fortran subroutines, and in Section 5 we discuss some implementation issues. Finally, in Section 6 we report the results from our numerical tests.

2 The extended Levinson algorithm

In this section we give a brief summary of the extended Levinson algorithm for nonsymmetric Toeplitz matrices—the simplifications for symmetric matrices are obvious. For details, we refer to the original descriptions of the algorithm in [7, 8]. Our notation for an order-n Toeplitz system $T_n \mathbf{x}_n = \mathbf{b}_n$, which follows that of Golub & Van Loan [14, §4.7],

is the following:

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

Notice that we write σ_i instead of the standard notation ρ_{-i}, because this makes our notation simpler. At the kth stage, the Levinson algorithm has recursively computed the solution $\mathbf{x}_k$ to the order-k system $T_k \mathbf{x}_k = \mathbf{b}_k$ 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 of the classical Levinson algorithm is then to compute the solution to 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 $T_{k+1} \mathbf{z}_{k+1} = -\mathbf{s}_{k+1}$, of order $k+1$, by means of a simple updating of the vectors $\mathbf{x}_k$, $\mathbf{y}_k$ and $\mathbf{z}_k$. This updating technique is numerically stable as long as the leading principal submatrix T_{k+1} is well-conditioned. The updating is, however, a numerically unstable process if T_{k+1} is ill-conditioned. For this reason, the extended Levinson algorithm performings a *block step* from T_k to T_{k+p}, where $p > 1$, whenever it is necessary to skip over one or several consecutive ill-conditioned leading principal submatrices. Assume for the moment that the block size p is known; in Section 3 we return to the practical choice of p.

To summarize a block step of the extended algorithm, define the two matrices

$$R_p \equiv \begin{pmatrix} \rho_1 & \cdots & \rho_p \\ \rho_2 & \cdots & \rho_{p+1} \\ \vdots & \ddots & \vdots \\ \rho_k & \cdots & \rho_{p+k-1} \end{pmatrix}, \qquad S_p \equiv \begin{pmatrix} \sigma_1 & \cdots & \sigma_p \\ \sigma_2 & \cdots & \sigma_{p+1} \\ \vdots & \ddots & \vdots \\ \sigma_k & \cdots & \sigma_{p+k-1} \end{pmatrix}, \tag{3}$$

and let Y_p and Z_p denote the solutions to the two 'extended Yule-Walker-problems':

$$T_k^T Y_p = -R_p, \qquad T_k Z_p = -S_p. \tag{4}$$

Note that the first column of R_p, S_p, Y_p and Z_p is $\mathbf{r}_k$, $\mathbf{s}_k$, $\mathbf{y}_k$ and $\mathbf{z}_k$, respectively. Also, let $\Gamma_p^{(k)}$ denote the Schur complement of T_k in T_{k+p}, which can be computed as

$$\Gamma_p^{(k)} = T_p + S_p^T Y_p, \tag{5}$$

and let E_k denote the $k \times k$ exchange matrix $E_k \equiv \text{antidiag}(1, 1, \ldots, 1)$. Then 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 $\mathbf{z}_k$:

$$\mathbf{x}_{k+p} = \begin{pmatrix} \mathbf{x}_k \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} E_k Y_p \\ I_p \end{pmatrix} \mathbf{a}_p \tag{6}$$

$$\mathbf{y}_{k+p} = \begin{pmatrix} \mathbf{y}_k \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} E_k Z_p \\ I_p \end{pmatrix} \mathbf{e}_p \tag{7}$$

$$\mathbf{z}_{k+p} = \begin{pmatrix} \mathbf{z}_k \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} E_k Y_p \\ I_p \end{pmatrix} \mathbf{f}_p, \tag{8}$$

where I_p is the identity matrix of order p, and the p-vectors $\mathbf{a}_k$, $\mathbf{e}_k$ and $\mathbf{f}_k$ are the solutions to the three indefinite systems

$$\Gamma_p^{(k)} \mathbf{a}_p = (\beta_{k+1}, \ldots, \beta_{k+p})^T - S_p^T E_k \mathbf{x}_k \tag{9}$$

$$(\Gamma_p^{(k)})^T \mathbf{e}_p = \mathbf{c}_p \equiv -(\rho_{k+1}, \ldots, \rho_{k+p})^T - R_p^T E_k \mathbf{y}_k \tag{10}$$

$$\Gamma_p^{(k)} \mathbf{f}_p = \mathbf{d}_p \equiv -(\sigma_{k+1}, \ldots, \sigma_{k+p})^T - S_p^T E_k \mathbf{z}_k. \tag{11}$$

The small $p \times p$ systems in (9)–(11) are solved by LU-factorization with pivoting. For $p = 1$, a block step is identical to one step of the classical Levinson algorithm, in which case $\Gamma_1^{(k)} = \gamma^{(k)}$ is the so-called prediction error.

The extended Levinson algorithm would be too expensive if we were to solve for the matrices Y_p and Z_p in (4) naively, say using Levinson's algorithm to 'carry along' a sufficiently large number of extra systems. Instead, our extended Levinson algorithm computes Y_p and Z_p only when they are required, by means of a simple updating algorithm which produces the columns of Y_p and Z_p one at a time in the order that they are needed, i.e. starting from the left. In most situations, the *previous* step is a classical Levinson step from T_{k-1} to T_k, and then Y_p and Z_p can be computed as follows:

Algorithm 1. Let $\mathbf{y}_{k,i}$ and $\mathbf{z}_{k,i}$ $i = 0, \ldots, p-1$ denote the columns of Y_p and Z_p, respectively. Also, let $(\mathbf{v})_j$ denote the jth component of the vector $\mathbf{v}$, and define the "shifted" vectors with a zero in the last position:

$$\bar{\mathbf{y}}_{k,i} \equiv ((\mathbf{y}_{k,i})_2, \ldots, (\mathbf{y}_{k,i})_k, 0)^T, \qquad \bar{\mathbf{z}}_{k,i} \equiv ((\mathbf{z}_{k,i})_2, \ldots, (\mathbf{z}_{k,i})_k, 0)^T. \tag{12}$$

Then the vectors $\mathbf{y}_{k,i}$ and $\mathbf{z}_{k,i}$ are given by

$$\mathbf{y}_{k,i} = \bar{\mathbf{y}}_{k,i-1} - (\mathbf{y}_{k,i-1})_1 \mathbf{y}_k + (\mathbf{c}_p)_i \mathbf{g}_k, \qquad \mathbf{z}_{k,i} = \bar{\mathbf{z}}_{k,i-1} - (\mathbf{z}_{k,i-1})_1 \mathbf{z}_k + (\mathbf{d}_p)_i \mathbf{h}_k, \tag{13}$$

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

$$\mathbf{g}_k \equiv \frac{1}{\gamma^{(k-1)}} \begin{pmatrix} E_{k-1} \mathbf{z}_{k-1} \\ 1 \end{pmatrix}, \qquad \mathbf{h}_k \equiv \frac{1}{\gamma^{(k-1)}} \begin{pmatrix} E_{k-1} \mathbf{y}_{k-1} \\ 1 \end{pmatrix}. \tag{14}$$

Here, $\gamma^{(k-1)}$ is the prediction error from the previous step. We recall that for $p = 1$, $\gamma^{(k)}$ is computed by means of the updating formula $\gamma^{(k+1)} = (1 - c_1 d_1)\gamma^{(k)}$, while for $p > 1$ higher accuracy is obtained by the following expression $\gamma^{(k+p)} = \rho_0 + \mathbf{s}_{k+p}^T \mathbf{y}_{k+p}$ [7, 8].

In rare cases, two block steps follow immediately after each other. Let p' denote the size of the *previous* block step, and let k' denote the corresponding dimension of that leading principal submatrix, 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 submatrix T_{k-1}) when going from to $T_{k'}$ to T_k. Hence, we cannot use Eq. (13) to compute $\mathbf{y}_{k,i}$ and $\mathbf{z}_{k,i}$ because we cannot produce the vectors $\mathbf{g}_k$ and $\mathbf{h}_k$ in (14). Instead, we *update* the vector $\mathbf{y}_{k',1}$ from the previous block step to get $\mathbf{y}_{k,1}$ and then, if $p > 2$, compute $\mathbf{g}_k$ by means of Eq. (13) (and similarly for $\mathbf{z}_{k,1}$ and $\mathbf{h}_k$):

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

$$\mathbf{y}_{k,1} = \begin{pmatrix} \mathbf{y}_{k',1} \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} E_{k'} Z_{p'} \\ I_{p'} \end{pmatrix} \mathbf{v}, \qquad \mathbf{z}_{k,1} = \begin{pmatrix} \mathbf{z}_{k',1} \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} E_{k'} Y_{p'} \\ I_{p'} \end{pmatrix} \mathbf{w}, \tag{15}$$

where the vectors $\mathbf{v}$ and $\mathbf{w}$ are the solutions to the systems

$$(\Gamma_{p'}^{(k')})^T \mathbf{v} = -\begin{pmatrix} \rho_{k'+2} \\ \vdots \\ \rho_{k+1} \end{pmatrix} - R_{p'}^T E_{k'} \mathbf{y}_{k',1}, \qquad \Gamma_{p'}^{(k')} \mathbf{w} = -\begin{pmatrix} \sigma_{k'+2} \\ \vdots \\ \sigma_{k+1} \end{pmatrix} - S_{p'}^T E_{k'} \mathbf{z}_{k',1}. \tag{16}$$

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 (13) with the vectors $\mathbf{g}_k$ and $\mathbf{h}_k$ given by

$$\mathbf{g}_k = (\mathbf{y}_{k,1} - \bar{\mathbf{y}}_k + (\mathbf{y}_k)_1 \mathbf{y}_k)/(\mathbf{c}_p)_1, \qquad \mathbf{h}_k = (\mathbf{z}_{k,1} - \bar{\mathbf{z}}_k + (\mathbf{z}_k)_1 \mathbf{z}_k)/(\mathbf{d}_p)_1. \tag{17}$$

A minor detail is that the very first leading principal submatrices T_1, T_2, ... may be ill-conditioned. If this is the case, we cannot perform a block step because we have no starting vectors to update. Instead we must compute the solutions $\mathbf{x}_p$, $\mathbf{y}_p$ and $\mathbf{z}_p$ by LU-factorization with pivoting of T_p, ignoring its Toeplitz structure.

3 Condition estimation and block size

The block size p must be chosen in each step so as to ensure optimal numerical stability, with the constraint that p be as small as possible in order to reduce the computational overhead involved in a block step, as compared to p classical Levinson steps. A detailed discussion of these aspects can be found in [7, 8]. In the extended Levinson algorithm we employ the following adaptive strategy. Let $p_{\max}$ be the maximum block-size set by the user. Also, let $\tilde{\psi}_{\min}(T_{k+p})$ denote an estimate of the smallest singular value of T_{k+p}, and let $s_{\min}$ denote the smallest $\tilde{\psi}_{\min}(T_i)$ accepted so far in all the previous steps. Then, in the kth step, we choose as block size the smallest value of p for which

$$\tilde{\psi}_{\min}(T_{k+p}) \geq 0.1\, s_{\min}, \qquad p = 1, \ldots, p_{\max}. \tag{18}$$

Our reason for comparing with $s_{\min}$ is that if we have already accepted a leading principal submatrix with a smallest singular value approximately equal to $s_{\min}$, then any better conditioned submatrix will not improve the error in the computed solution significantly. The factor 0.1 is included in (18) to ensure that a block step is taken only if it is really necessary—i.e., $\tilde{\psi}_{\min}(T_{k+p})$ must be smaller than $0.1\, s_{\min}$ in order to 'trigger' a block step. If (18) is not satisfied for any $p \leq p_{\max}$, then we choose as block-size the p for which $\tilde{\psi}_{\min}(T_{k+p})$ 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 submatrix T_{k+p} within the look-ahead range $p_{\max}$. The block strategy is therefore numerically stable as long as T_n does not have more than $p_{\max} - 1$ consecutive ill-conditioned leading principal submatrices.

The estimate $\tilde{\psi}_{\min}(T_{k+p})$ that we use in the extended Levinson algorithm must be reliable and yet computationally inexpensive. The following estimate is derived in [7, 8]:

$$\tilde{\psi}_{\min}(T_{k+p}) = \tilde{\psi}_{\min}(\Gamma_p^{(k)}) / \max\{1, \mu_{(Y)}, \mu_{(Z)}, \mu_{(Y)}\mu_{(Z)}\}, \tag{19}$$

where $\tilde{\psi}_{\min}(\Gamma_p^{(k)})$ is an estimate of the smallest singular value of the Schur complement $\Gamma_p^{(k)}$, while $\mu_{(Y)}$ and $\mu_{(Z)}$ are the numerically largest elements of Y_p and Z_p, i.e.

$$\mu_{(Y)} \equiv \max\{|(Y_p)_{ij}|\}, \qquad \mu_{(Z)} \equiv \max\{|(Z_p)_{ij}|\}. \tag{20}$$

The estimate $\tilde{\psi}_{\min}(\Gamma_p^{(k)})$ can be computed by a Linpack-style condition estimator using the LU-factorization of $\Gamma_p^{(k)}$, which is needed in order to solve Eqs. (9)–(11). For $p = 1$, $\tilde{\psi}_{\min}(T_{k+1})$ is simply $|\gamma^{(k)}|$, the absolute value of the prediction error, while $\mu_{(Y)} = \|\mathbf{y}_k\|_\infty$ and $\mu_{(Z)} = \|\mathbf{z}_k\|_\infty$.

From this description of the extended Levinson algorithm it is evident that the accuracy of the computed solution is proportional to the largest condition number of all the leading principal submatrices T_{k+p} involved in the block steps. Following [10, §2.4.1], we therefore define the *algorithm condition number* as $\|T_n\|_2$ divided by the smallest singular value Ψ of all the leading principal submatrices involved in the block steps (but not in those steps that were 'skipped over'). We can readily use the following estimate

$$\|T_n\|_2 \approx |\rho_0| + \max\{\|\mathbf{r}_{n-1}\|_1, \|\mathbf{s}_{n-1}\|_1\}, \tag{21}$$

and it follows that the quantity

$$\kappa_{\text{Levinson}} \equiv (|\rho_0| + \max\{\|\mathbf{r}_{n-1}\|_1, \|\mathbf{s}_{n-1}\|_1\})/\Psi \tag{22}$$

is a good estimate of the algorithm condition number. Notice that κ_{Levinson} can be computed with almost no computational overhead. Also notice that without any extra overhead, we can compute the estimate

$$\kappa(T_n) \equiv (|\rho_0| + \max\{\|\mathbf{r}_{n-1}\|_1, \|\mathbf{s}_{n-1}\|_1\})/\tilde{\psi}_{\min}(T_n) \tag{23}$$

of the usual condition number of the Toeplitz matrix T_n.

The two condition number estimates κ_{Levinson} and $\kappa(T_n)$ are useful diagnostic tools when evaluating the accuracy of the computed solution. If κ_{Levinson} and $\kappa(T_n)$ are of the same order of magnitude, then the extended Levinson algorithm succeeded in computing a solution which is as reliable as can be expected. On the other hand, if $\kappa_{\text{Levinson}} \gg \kappa(T_n)$ then the maximum block-size $p_{\max}$ was not large enough to allow the extended Levinson algorithm to 'skip over' a sequence of more than $p_{\max}-1$ consecutive ill-conditioned leading principal submatrices. We believe that it is very rare in applications that $p_{\max}$ is greater than, say, 5. We also note that the sequence of estimates $\tilde{\psi}_{\min}(T_i)$, $i = 1, \ldots, n$, if returned to the user from the program, may give additional information about the Toeplitz matrix.

We conclude this section by presenting in Fig. 1 an algorithmic description of the extended Levinson algorithm, including the condition estimation.

4 The Fortran subroutines

The extended Levinson algorithm, as described in Sections 2 and 3, has been implemented in Fortran 77 in double precision. We provide subroutines for symmetric as well as nonsymmetric Toeplitz matrices, because a substantial saving in computing time can be achieved by taking the symmetry into account. In addition to subroutines that allow

init: let T_k = best conditioned T_i, $i = 1 : p_{\max}$
solve $T_k \mathbf{x}_k = \mathbf{b}_k$, $T_k^T \mathbf{y}_k = -\mathbf{r}_k$ and $T_k \mathbf{z}_k = -\mathbf{z}_k$
by LU-factorization with partial pivoting
$\gamma^{(k)} = \rho_0 + \mathbf{s}_k^T \mathbf{y}_k$
$s_{\min} = \tilde{\psi}_{\min}(T_k)$
$\Psi = \tilde{\psi}_{\min}(T_k)$
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 Algorithm 2
else
set up Y_p and Z_p using Algorithm 1
endif
set up the systems in Eqs. (9)–(11)
compute the estimate $\tilde{\psi}_{\min}(T_{k+p})$ by (19)
if $\tilde{\psi}_{\min}(T_{k+p}) > 0.1\, s_{\min}$ goto *update*
end
let T_{k+p} = best conditioned T_{k+i}, $i = 1 : p_{\max}$
if $\tilde{\psi}_{\min}(T_{k+p}) < s_{\min}$ then $s_{\min} = \tilde{\psi}_{\min}(T_{k+p})$
update: solve (9)–(11) for updates $\mathbf{a}_p$, $\mathbf{e}_p$ and $\mathbf{f}_p$
by LU-factorization with partial pivoting
if $\tilde{\psi}_{\min}(T_{k+p}) < \Psi$ then $\Psi = \tilde{\psi}_{\min}(T_{k+p})$
compute $\gamma^{(k+p)}$
end
end
$\kappa_{\text{Levinson}} = (|\rho_0| + \max\{\|\mathbf{r}_{n-1}\|_1, \|\mathbf{s}_{n-1}\|_1\})/\Psi$
$\kappa(T_n) = (|\rho_0| + \max\{\|\mathbf{r}_{n-1}\|_1, \|\mathbf{s}_{n-1}\|_1\})/\tilde{\psi}_{\min}(T_n)$

Figure 1: The extended Levinson algorithm, including condition estimation.

	W	Y	Z
DSYTCL	n	n	–
DGETCL	$2n$	n	n
DSYTE2	$2n$	$2n$	–
DGETE2	$4n$	$2n$	$2n$
DSYTEP	$p_{\max}(p_{\max}^2+14)/3+2(p_{\max}^2-1)$	$(p_{\max}+2)n$	–
DGETEP	$p_{\max}(p_{\max}^2+20)/3+2(p_{\max}^2-1)$	$(p_{\max}+2)n$	$(p_{\max}+2)n$

Table 1: The dimensions of the work-space arrays W, Y, and Z. Note that the dimension of W is independent of n for D__TEP.

the user to specify an arbitrary maximum block-size $p_{\max}$, we also provide subroutines for the two special cases $p_{\max} = 2$ and $p_{\max} = 1$ (the latter being identical to the classical Levinson algorithm except for the additional condition estimation). Hence, the following subroutines are provided:

- routines for symmetric and nonsymmetric Toeplitz matrices,
- routines for $p_{\max} = 1$, $p_{\max} = 2$, and general $p_{\max}$.

If no a priori information is available about the Toeplitz system, we recommend using the subroutines for $p_{\max} = 2$ because they are more efficient than those for general $p_{\max}$. A large algorithm condition number estimate, namely $\kappa_{\text{Levinson}} \gg \kappa(T_n)$, will indicate that a larger $p_{\max}$ was indeed required to solve the Toeplitz system, and the routine which allows a general $p_{\max}$ can then be called.

In the naming convention for the subroutines we follow the convention from LAPACK [2]: the first character denotes the precision, the next two characters denote the type of matrix, and the last three characters denote the algorithm. Hence, the subroutine names are of the form DXXYYY, where D means double precision, and where:

XX	SY	symmetric matrix
	GE	general matrix
YYY	TCL	Toeplitz matrix, classical Levinson algorithm ($p_{\max} = 1$)
	TE2	Toeplitz matrix, extended Levinson algorithm with $p_{\max} = 2$
	TEP	Toeplitz matrix, extended Levinson algorithm with general $p_{\max}$.

The number of input and output parameters to the subroutines depends on the particular routine. Most of the parameters are self-explanatory. The output-parameter PSTEPS is the number of block steps used to solve the system, the output SIGMIN is an array holding the estimates $\tilde{\psi}(T_k)$, $k = 1, \ldots, n$ computed by (19), and the logical input parameter WANTSM controls whether SIGMIN should be used or not. Below, we give a complete list of all the parameters:

Input:	N	the order n of the Toeplitz matrix
	R	the first row of the Toeplitz matrix
	S	the first column of the Toeplitz matrix (routines DGE___ only)
	B	the right-hand side $\mathbf{b}_n$

$p_{\max}$	3	4	5	6	7	8	9	10
DSYTEP	39	70	113	170	243	334	445	578
DGETEP	45	78	123	182	257	350	463	598

Table 2: The length of the work-space array W in D__TEP for some values of $p_{\max}$.

	PMAX	the maximum block-size $p_{\max}$ (routines ___TEP only)
	WANTSM	logical: true - store all singular value estimates in SIGMIN false - do not use SIGMIN
	W	work-space array whose length is given in Table 1
	Y,Z	work-space arrays whose dimensions are given in Table 1 (Z is required by routines DGE___ only)
Output:	X	the solution $\mathbf{x}_n$ to the Toeplitz system
	CONDA	the algorithm condition number estimate κ_{Levinson} (22)
	CONDM	the matrix condition number estimate $\kappa(T_n)$ (23)
	PSTEPS	the number of block steps used to solve the Toeplitz system
	SIGMIN	a one-dimensional array of length n holding all the estimates $\tilde{\psi}_{\min}(T_i)$ for $i = 1, \ldots, n$, with negative entries for those submatrices that were 'skipped over'
	IFAIL	an integer with the following meaning: 0 - normal value, no errors detected 1 - illegal input parameters (N $\leq$ 0 or PMAX $\leq$ 0) 2 - the process was stopped with $k < n$ because $p_{\max}$ consecutive Schur complements are exactly singular.

The value IFAIL = 2 occurs only if $p_{\max}$ exactly singular consecutive submatrices are encountered, which is extremely rare in applications. Such matrices cannot be handled by our codes.

The work-space arrays Y and Z are used for storage of data associated with the 'extended Yule-Walker problems' in Eq. (4). The one-dimensional work-space array W is used for auxiliary storage. The dimensions of these arrays depend on the particular subroutine; Table 1 lists all the combinations. As an aid to the user, Table 2 shows the required length of array W for D__TEP for $p_{\max} = 3, \ldots, 10$.

The output parameters CONDA, CONDM, PSTEPS, and SIGMIN are useful diagnostic tools. If CONDA $\approx$ CONDM, then the subroutine succeeded in computing a solution which is as reliable as can be expected, and PSTEPS shows how many block steps were actually used to solve the problem. Moreover, by inspecting SIGMIN it is fairly easy to estimate the maximum block-size $p_{\max}$ required to obtain a given accuracy. If CONDA $\gg$ CONDM, then a larger $p_{\max}$ is required, and inspection of SIGMIN will usually reveal the necessary value of PMAX.

5 Implementation notes

To simplify the Fortran codes, we have based our implementations of the extended Levinson algorithm on the Level 1 BLAS [19]. We are aware that this strategy involves some

computational overhead; however, we feel that a well-structured implementation is more important than a slightly faster subroutine, and use of the BLAS routines indeed makes the implementation clearer. If optimal speed is required on a particular computer, it is fairly straightforward to tune the routine to that specific computer, either by means of specialized BLAS-routines or by substituting in-line code for these routines.

In order to solve the small indefinite systems (9)–(11) involving $\Gamma_p^{(k)}$, and to solve the small Toeplitz system $T_k \mathbf{x}_k = \mathbf{b}_k$ in the very first step, we need an efficient linear algebra subroutine for solving a small, general system of linear algebraic equations. Since the matrices $\Gamma_p^{(k)}$ and T_k are not guaranteed to be positive definite, pivoting must be employed, and therefore we cannot use the old factorization when going from the submatrix $\Gamma_p^{(k)}$ of order p to the next $\Gamma_{p+1}^{(k)}$ of order $p+1$. This computational overhead is insignificant as long as $p_{\max}$ is small. For $p > 2$, we use the Linpack subroutine `DGEFA`, and for the most common case $p = 2$ we have written a subroutine `DGESL2` based on Cramer's rule (which is more efficient and as accurate as Gaussian elimination for 2×2 systems).

We also need a Linpack-style condition estimator for estimating the smallest singular value of the small matrices factorized by means of `DGEFA`. There are, however, two drawbacks of the Linpack subroutine `DGECO` for doing this:

- it divides the $\psi_{\min}$ estimate with an estimate of the norm of the matrix in order to produce a condition number estimate, whereas we need the $\psi_{\min}$ estimate,
- it may involve a lot of unnecessary scaling to avoid overflow.

As a consequence, we *modify* the Linpack subroutine `DGECO` in two ways: we skip the division with the matrix norm, and we incorporate the modification of the scaling as described by Grimes & Lewis [16]. The name of the new subroutine is `DGESM`.

For the special case of estimating the smallest singular value of symmetric and general 2×2 matrices, we have written special-purpose subroutines `DSYSV2` and `DGESV2`, which compute both the largest and the smallest singular values of a 2×2 matrix. These subroutines are based on the algorithm USVD from [4]. Notice that for a *symmetric* 2×2 *Toeplitz matrix*, the singular values are easy to compute: they are simply $|\rho_0 \pm \rho_1|$.

For completeness, we mention that the following subroutines from the Level 1 BLAS are required: `IDAMAX`, `DASUM`, `DAXPY`, `DDOT`, and `DSCAL`.

6 Numerical tests

All our numerical tests were carried out in double precision on an Alliant FX/8 with IEEE arithmetic and with a machine precision $\mathbf{u} \approx 2.2 \cdot 10^{-16}$. Neither parallel nor vector option was used.

Let us first introduce the test matrices that we have used to test the extended Levinson algorithm. Sweet [22] gives four small test matrices (one symmetric and three nonsymmetric) with ill-conditioned leading principal submatrices, as listed in Table 3. Trench [23] mentions the so-called KMS matrices $T_n^{(KMS)}$ which are symmetric Toeplitz matrices with elements given by

$$\rho_0 = \epsilon, \qquad \rho_i = \sigma_i = (1/2)^{i-1}, \quad i = 1, \ldots, n-1. \tag{24}$$

	$T_6^{(S_1)}$	$T_6^{(S_2)}$		$T_6^{(S_3)}$		$T_{13}^{(S_4)}$			
i	$\rho_i = \sigma_i$	ρ_i	σ_i	ρ_i	σ_i	ρ_i	σ_i	ρ_{7+i}	σ_{7+i}
0	20	4	4	8	8	5	5	-5	-1
1	15	8	6	4	4	-1	1	-2	2
2	$2.5+\epsilon$	1	$\frac{71}{15}+\epsilon$	1	$-34+\epsilon$	6	-3	-7	1
3	6	6	5	6	5	2	12.755	1	-6
4	1	2	3	2	3	5.697	-19.656	10	1
5	-2	3	1	3	1	5.850	28.361	-15	-0.5
6						3	-7		

Table 3: The four test matrices from [22], where ϵ is a small multiple times the machine precision. $T_6^{(S_1)}$, $T_6^{(S_2)}$ and $T_6^{(S_3)}$ have one ill-conditioned 3×3 leading principal submatrix, and $T_{13}^{(S_4)}$ has 5 consecutive leading principal submatrices of order 4–8.

Here, ϵ is a small multiple of the machine precision. If $\rho_0 = 0$, then every third leading principal submatrix $T_1^{(KMS)}$, $T_4^{(KMS)}$, $T_7^{(KMS)}$, ... of $T_n^{(KMS)}$ is exactly singular. Finally, we use the test matrices $T_n^{(\delta,q)}$ from [7, 8], which are 'random' symmetric or nonsymmetric Toeplitz matrices modified such that the smallest singular value of the leading $q \times q$ principal submatrix $T_q^{(\delta,q)}$ is of the order δ. These test matrices are generated by first generating a random Toeplitz matrix $\bar{T}_n$ with positive elements, then computing the numerically smallest real eigenvalue λ of the submatrix $\bar{T}_q$, and finally setting

$$T_n^{(\delta,q)} = \bar{T}_n - (\lambda - \delta) I_n. \tag{25}$$

The positivity of $\bar{T}_q$ ensures the existence of λ. The matrix $T_n^{(\delta,q)}$ has a leading principal submatrix of order q whose smallest singular value can be controlled by means of δ. For more details, cf. [7, 8].

It is crucial for the reliability of the extended Levinson algorithm that the singular value estimates $\tilde{\psi}_{\min}(T_{k+p})$ in Eq. (19) are good estimates, because they are used to detect ill-conditioned leading principal submatrices T_{k+p}. Our first series of numerical tests demonstrates the quality of the estimates $\tilde{\psi}_{\min}(T_{k+p})$. To measure this quality experimentally, we define

$$\phi_i \equiv \max\{\phi_{\min}(T_i)/\tilde{\phi}_{\min}(T_i), \tilde{\phi}_{\min}(T_i)/\phi_{\min}(T_i)\}, \tag{26}$$

where $\phi_{\min}(T_i)$ is the smallest singular value of T_i. We generated 500 test matrices of type $T_{200}^{(\delta,50)}$; i.e. 200×200 nonsymmetric 'random' Toeplitz matrices whose 50×50 leading principal submatrix has a smallest singular value of the order δ. The experiments were carried out in double precision on an Alliant FX/8, with a machine precision $\mathbf{u} \approx 2.2 \cdot 10^{-16}$, and we used the following values of δ:

$$\delta = 10^{-15}, \quad 10^{-13}, \quad 10^{-11}, \quad 10^{-9}, \quad 10^{-7}. \tag{27}$$

For each δ we generated 100 test matrices and computed all the ϕ_i according to (26) for $i = 1, \ldots, n$. This gives a total of 50,000 values of ϕ_i. Most of these ϕ_i correspond to well-conditioned submatrices T_i with a smallest singular value *always* in the range 10^{-3} to 1, while 500 ϕ_i correspond to the ill-conditioned submatrices $T_{50}^{(\delta,50)}$ with δ given by (27).

ϕ_i	δ					
	$\geq 10^{-6}$	10^{-7}	10^{-9}	10^{-11}	10^{-13}	10^{-15}
1.00–1.78	6,538	0	0	0	0	1
1.78–3.16	8,517	0	0	0	0	4
3.16–5.62	13,374	4	3	2	9	3
5.62–10.0	20,771	60	60	47	55	12
10.0–17.8	29,895	32	32	43	35	12
17.8–31.6	18,177	4	5	8	1	12
31.6–56.2	2,070	0	0	0	0	14
56.2–100	145	0	0	0	0	12
100–178	7	0	0	0	0	7
178–316	4	0	0	0	0	7
178–562	2	0	0	0	0	4
562–1000	0	0	0	0	0	3
10^3–10^5	0	0	0	0	0	9
$> 10^5$	0	0	0	0	0	0

Table 4: Histograms of the 'quality measure' ϕ_i for 500 nonsymmetric test matrices $T_{200}^{(\delta,50)}$. E.g., the entry 6,538 in the upper left corner means that 6,538 submatrices—among all the submatrices with a smallest singular value greater than 10^{-6}—had a 'quality factor' in the range 1.00–1.78. The machine precision is $\mathbf{u} = 2.2 \cdot 10^{-16}$. The largest value of ϕ_i for $\delta = 10^{-15}$ is $1.7 \cdot 10^4$.

The results of these experiments are shown in Table 4. We see that singular values greater than about $10^{-13}\mathbf{u}$ are always estimated within a factor 100, and typically the estimates are much better, approximately within a factor 20 of the true singular values. For singular values smaller than $10^{-13}\mathbf{u}$, the estimates are, in average, within a factor of 100 from the true singular values, but the factor may become as large as 10^5. We conclude that the quantities κ_{Levinson} and $\kappa(T_n)$ as computed by means of Eqs. (22) and (23) are usually good estimates of the condition numbers, except that *very* large condition numbers may be underestimated.

Although the very small singular value estimates $\tilde{\psi}_{\min}(T_i)$ may be off by more than a factor 10^4, what really matters is the algorithm's ability to detect changes in the size of the smallest singular value between consecutive submatrices. Due to the way we generated the test-matrices used in Table 4, we know that $\psi_{\min}(T_{49})$ is never smaller than 10^{-3}, while $\psi_{\min}(T_{50}) \approx \delta \leq 10^{-7}$. I.e., the relative gaps between the smallest singular value of the submatrices T_{49} and T_{50} are at least 10^4. The 'quality measures' listed in Table 4 are always orders of magnitude smaller than these relative gaps. We conclude that the strategy using the estimate $\tilde{\psi}_{\min}(T_i)$ in Eq. (19) is always capable of detecting a major decrease in the sie of the smallest singular value between two consecutive submatrices.

We performed similar experiments with symmetric matrices. Exactly the same conclusions hold for these experiments.

Our next numerical experiments were concerned with the ability of the extended Levinson algorithm to *accurately solve* problems with ill-conditioned leading principal submatrices. Table 5 shows the relative error in the computed solution $\tilde{\mathbf{x}}_n$, i.e. $\|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|_2 / \|\mathbf{x}_n\|_2$,

	$T_6^{(S_1)}$	$T_6^{(S_2)}$	$T_6^{(S_3)}$	$T_{13}^{(S_4)}$	$T_{512}^{(KMS)}$	$T_{2048}^{(KMS)}$
TSLD1	$2.00 \cdot 10^{-2}$	$3.51 \cdot 10^{-2}$	$4.87 \cdot 10^{-1}$	$4.66 \cdot 10^{-10}$	$2.34 \cdot 10^{-2}$	$2.63 \cdot 10^{-2}$
D_TCL	$2.00 \cdot 10^{-2}$	$4.42 \cdot 10^{-3}$	$3.01 \cdot 10^{-1}$	$2.95 \cdot 10^{-10}$	$2.43 \cdot 10^{-2}$	$2.46 \cdot 10^{-2}$
D_TE2	$2.87 \cdot 10^{-16}$	$8.88 \cdot 10^{-15}$	$2.76 \cdot 10^{-16}$	$3.20 \cdot 10^{-10}$	$2.71 \cdot 10^{-14}$	$1.53 \cdot 10^{-13}$
D_TEP	$2.87 \cdot 10^{-16}$	$8.79 \cdot 10^{-16}$	$2.76 \cdot 10^{-16}$	$5.85 \cdot 10^{-14}$	$2.71 \cdot 10^{-14}$	$1.53 \cdot 10^{-13}$

Table 5: The relative error in the computed solution for six test matrices, solved by means of TSLD1 from the Argonne Toeplitz package [1] and our three extended Levinson algorithms. In D_TEP we used $p_{\max} = 6$. Routines of type DSY___ were used for the symmetric matrices $T_6^{(S_1)}$, $T_{512}^{(KMS)}$ and $T_{2048}^{(KMS)}$, while routines of type DGE___ were used for the remaining matrices.

for test matrices of type $T_n^{(S_i)}$ and $T_n^{(KMS)}$. The right-hand side $\mathbf{b}_n$ was chosen such that the exact solution is all ones, $\mathbf{x}_n = (1, 1, \ldots, 1)^T$. We see that once the maximum block-size $p_{\max}$ is large enough, our extended Levinson algorithm is always capable of computing a solution to almost full working precision.

Finallly, we carried out extensive experiments in order to study the efficiency and accuracy of our implementations of the extended Levinson algorithm when solving large-scale Toeplitz systems. In particular, we studied two "extreme" situations for the extended Levinson algorithm, namely when very few block steps are required, and when a large number of block steps is required. Test matrices belonging to the first situation can be generated simply by generating random Toeplitz matrices, while the KMS test-matrices (24) clearly give rise to the second situation. Typical results from these experiments are listed in Tables 6–8. We compare the following six programs:

1 TSLD1 from the Argonne Toeplitz package [1],
2 D_TCL the classical Levinson algorithm with condition estimation
3 D_TE2 the extended Levinson algorithm, specially designed for $p_{\max} = 2$,
4 D_TEP the extended Levinson algorithm with $p_{\max} = 4$,
5 D_TEP the extended Levinson algorithm with $p_{\max} = 6$,
6 D_TEP the extended Levinson algorithm with $p_{\max} = 8$.

Notice that TSLD1 does not take symmetry into account—hence the small overhead in Table 6. In the three tables, "rel. error" denotes the relative error $\|\tilde{\mathbf{x}}_n - \mathbf{x}_n\|_2 / \|\mathbf{x}_n\|_2$ of the computed solution $\tilde{\mathbf{x}}_n$, "time" is the execution time measured in seconds, and "overhead" denotes the computational overhead compared to programs 1 and 2.

From these tables we see that our extended Levinson algorithms in most cases are capable of solving the Toeplitz systems at least as accurately as TSLD1, and much more accurately if there are nearly singular leading principal submatrices. In fact, the solutions computed by all our subroutines are as accurate as can be expected, because the relative error in the solution is within a factor 10 to 100 of CONDA times the machine precision $\mathbf{u} \approx 2.2 \cdot 10^{-16}$. Notice that we obtain higher accuracy for the KMS matrices by taking the symmetry of the Toeplitz matrix $T_n^{(KMS)}$ into account. More experiments with the accuracy of the extended Levinson algorithm are reported in [7, 8].

We note in passing that neither the classical nor the extended Levinson algorithms are—in average–as accurate as LU-factorization with pivoting. This is obviously so, be-

n	program	rel. error	CONDA	CONDM	PSTEPS	time	overhead	
512	1	$2.82 \cdot 10^{-11}$				5.0	0%	
	2	$1.97 \cdot 10^{-10}$	$3.06 \cdot 10^{5}$	$2.43 \cdot 10^{2}$		5.7	14%	0%
	3	$4.89 \cdot 10^{-11}$	$2.04 \cdot 10^{4}$	$2.43 \cdot 10^{2}$	6	6.3	27%	11%
	4	$4.89 \cdot 10^{-11}$	$2.04 \cdot 10^{4}$	$2.43 \cdot 10^{2}$	6	6.9	39%	22%
	5	$4.89 \cdot 10^{-11}$	$2.04 \cdot 10^{4}$	$2.43 \cdot 10^{2}$	6	6.9	39%	22%
	6	$4.89 \cdot 10^{-11}$	$2.04 \cdot 10^{4}$	$2.43 \cdot 10^{2}$	6	7.0	39%	22%
1024	1	$5.39 \cdot 10^{-10}$				19.9	0%	
	2	$1.30 \cdot 10^{-9}$	$2.99 \cdot 10^{7}$	$2.56 \cdot 10^{3}$		22.7	14%	0%
	3	$2.31 \cdot 10^{-10}$	$7.04 \cdot 10^{5}$	$2.56 \cdot 10^{3}$	7	25.9	30%	14%
	4	$4.90 \cdot 10^{-10}$	$7.85 \cdot 10^{4}$	$2.56 \cdot 10^{3}$	10	27.6	39%	22%
	5	$4.90 \cdot 10^{-10}$	$7.85 \cdot 10^{4}$	$2.56 \cdot 10^{3}$	10	27.6	39%	22%
	6	$4.90 \cdot 10^{-10}$	$7.85 \cdot 10^{4}$	$2.56 \cdot 10^{3}$	10	27.6	39%	22%
2048	1	$5.84 \cdot 10^{-8}$				80.5	0%	
	2	$5.35 \cdot 10^{-8}$	$1.46 \cdot 10^{7}$	$3.09 \cdot 10^{3}$		91.0	13%	0%
	3	$1.02 \cdot 10^{-9}$	$2.59 \cdot 10^{5}$	$3.09 \cdot 10^{3}$	6	102.1	27%	12%
	4	$1.02 \cdot 10^{-9}$	$2.59 \cdot 10^{5}$	$3.09 \cdot 10^{3}$	6	110.6	37%	22%
	5	$1.02 \cdot 10^{-9}$	$2.59 \cdot 10^{5}$	$3.09 \cdot 10^{3}$	6	110.6	37%	22%
	6	$1.02 \cdot 10^{-9}$	$2.59 \cdot 10^{5}$	$3.09 \cdot 10^{3}$	6	111.0	38%	22%

Table 6: Large-scale Toeplitz systems with symmetric random test matrices.

n	program	rel. error	CONDA	CONDM	PSTEPS	time	overhead	
512	1	$1.19 \cdot 10^{-10}$				5.0	0%	
	2	$3.85 \cdot 11^{-11}$	$5.70 \cdot 10^{4}$	$5.85 \cdot 10^{1}$		9.0	77%	0%
	3	$5.89 \cdot 11^{-11}$	$5.70 \cdot 10^{4}$	$5.85 \cdot 10^{1}$	5	10.2	102%	14%
	4	$5.89 \cdot 11^{-11}$	$5.70 \cdot 10^{4}$	$5.85 \cdot 10^{1}$	5	11.2	122%	25%
	5	$5.89 \cdot 11^{-11}$	$5.70 \cdot 10^{4}$	$5.85 \cdot 10^{1}$	5	11.0	119%	23%
	6	$5.89 \cdot 11^{-11}$	$5.70 \cdot 10^{4}$	$5.85 \cdot 10^{1}$	5	11.2	122%	25%
1024	1	$3.85 \cdot 10^{-10}$				20.2	0%	
	2	$3.22 \cdot 10^{-10}$	$1.84 \cdot 10^{5}$	$1.15 \cdot 10^{3}$		35.7	77%	0%
	3	$1.89 \cdot 10^{-10}$	$1.22 \cdot 10^{5}$	$1.15 \cdot 10^{3}$	2	41.4	105%	16%
	4	$1.89 \cdot 10^{-10}$	$1.22 \cdot 10^{5}$	$1.15 \cdot 10^{3}$	2	45.0	123%	26%
	5	$1.89 \cdot 10^{-10}$	$1.22 \cdot 10^{5}$	$1.15 \cdot 10^{3}$	2	43.8	117%	23%
	6	$1.89 \cdot 10^{-10}$	$1.22 \cdot 10^{5}$	$1.15 \cdot 10^{3}$	2	43.7	117%	23%
2048	1	$3.20 \cdot 10^{-9}$				80.9	0%	
	2	$2.25 \cdot 10^{-9}$	$1.12 \cdot 10^{6}$	$9.69 \cdot 10^{1}$		143.6	78%	0%
	3	$9.83 \cdot 10^{-10}$	$1.71 \cdot 10^{5}$	$9.69 \cdot 10^{1}$	3	164.1	103%	14%
	4	$7.76 \cdot 10^{-9}$	$1.71 \cdot 10^{5}$	$9.69 \cdot 10^{1}$	3	176.4	118%	23%
	5	$7.76 \cdot 10^{-9}$	$1.71 \cdot 10^{5}$	$9.69 \cdot 10^{1}$	3	175.1	116%	22%
	6	$7.76 \cdot 10^{-9}$	$1.71 \cdot 10^{5}$	$9.69 \cdot 10^{1}$	3	175.1	116%	22%

Table 7: Large-scale Toeplitz systems with nonsymmetric random test matrices.

		DSY___				DGE___			
n	program	rel. error	time	overhead		rel. error	time	overhead	
512	1	$2.34 \cdot 10^{-2}$	5.0	0%		$2.34 \cdot 10^{-2}$	5.1	0%	
	2	$2.43 \cdot 10^{-2}$	5.7	14%	0%	$2.43 \cdot 10^{-2}$	9.0	77%	0%
	3	$2.71 \cdot 10^{-14}$	6.8	36%	20%	$3.54 \cdot 10^{-13}$	11.0	117%	22%
	4	$2.71 \cdot 10^{-14}$	7.6	53%	34%	$3.54 \cdot 10^{-13}$	12.3	144%	37%
	5	$2.71 \cdot 10^{-14}$	7.6	53%	34%	$3.54 \cdot 10^{-13}$	12.1	140%	36%
	6	$2.71 \cdot 10^{-14}$	7.6	53%	34%	$3.54 \cdot 10^{-13}$	12.3	143%	37%
1024	1	$1.53 \cdot 10^{-1}$	20.0	0%		$1.53 \cdot 10^{-1}$	20.3	0%	
	2	$2.28 \cdot 10^{-2}$	22.7	13%	0%	$2.28 \cdot 10^{-2}$	35.8	77%	0%
	3	$3.33 \cdot 10^{-3}$	27.6	38%	22%	$7.83 \cdot 10^{-3}$	44.3	119%	24%
	4	$3.33 \cdot 10^{-3}$	30.3	51%	34%	$7.83 \cdot 10^{-3}$	49.4	144%	38%
	5	$3.33 \cdot 10^{-3}$	30.3	51%	34%	$7.83 \cdot 10^{-3}$	48.7	140%	36%
	6	$3.33 \cdot 10^{-3}$	30.4	52%	34%	$7.83 \cdot 10^{-3}$	48.6	140%	36%
2048	1	$2.63 \cdot 10^{-2}$	80.8	0%		$2.63 \cdot 10^{-2}$	81.3	0%	
	2	$2.46 \cdot 10^{-2}$	90.9	12%	0%	$2.46 \cdot 10^{-2}$	143.7	77%	0%
	3	$1.53 \cdot 10^{-13}$	109.9	36%	21%	$6.90 \cdot 10^{-12}$	175.1	116%	22%
	4	$1.53 \cdot 10^{-13}$	122.0	51%	34%	$6.90 \cdot 10^{-12}$	193.2	138%	34%
	5	$1.53 \cdot 10^{-13}$	122.1	51%	34%	$6.90 \cdot 10^{-12}$	192.4	137%	34%
	6	$1.53 \cdot 10^{-13}$	122.1	51%	34%	$6.90 \cdot 10^{-12}$	192.3	137%	34%

Table 8: Toeplitz systems with KMS test-matrices, solved by both DSY___ and DGE___. The required number of PSTEPS is 171, 341, and 683, for $n = 512$, 1024, and 2048, respectively. The corresponding pairs of algorithm and condition number estimates, $(\kappa_{\text{Levinson}}, \kappa(T_n))$, computed by DSYTEP, are (2.0,2.0), $(1.40 \cdot 10^{11}, 1.40 \cdot 10^{11})$, and (2.0,2.0).

cause our use of block steps conceptually corresponds to a very limited, 'local' type of pivoting, whereas LU-factorization can pivot throughout the matrix. However, LU-factorization is an $O(n^3)$-process, while the complexity of both the classical and the extended Levinson algorithms is $O(n^2)$.

We now turn to the computational overhead of the extended Levinson algorithms, as compared to `TSLD1`. This overhead is caused by three factors:

1. the Level 1 BLAS routines and the data structures necessary for these routines,
2. the condition estimation necessary in each step,
3. taking a block step whenever necessary.

The overhead caused by the use of the BLAS routines depends on the particular computer as well as the particular version of the BLAS being used. Our tests were carried out on an Alliant FX/8 using the standard Level 1 BLAS from Linpack [13]. Our experiments with a modified version of `DGETCL` without condition estimation (not shown here) show that the BLAS-overhead, as compared to `TSLD1`, is approximately 50 %.

Concerning the overhead involved in the condition estimation, the execution times for `TSLD1` should be compared with those of `DGETCL`, with 50 % subtracted to compensate for the above-mentiond BLAS overhead on the Alliant FX/8. The comparison shows that the condition estimation overhead is approximately 25–30 %. The overhead is mainly due to the comparisons involved in computing $\mu_{(Y)} = \max\{|(Y_p)_{ij}|\}$ and $\mu_{(Z)} = \max\{|(Z_p)_{ij}|\}$, which is implemented using the BLAS routine `IDAMAX`. This overhead is certainly not negligible. On the other hand, we believe that the computational overhead of the extended Levinson algorithms is indeed reasonable, taking into account the advantages of these algorithms, namely the ability to solve a large class of general Toeplitz systems and the accuracy estimation for the computed solution.

Finally, let us consider the computational overhead involved in the block steps themselves. For this purpose, we use the results in Table 8 for Toeplitz systems with KMS test-matrices $T_n^{(KMS)}$. For these test matrices, the extended Levinson algorithm alternates between a classical step and a block step with block size $p = 2$. It is interesting to see that even for such 'demanding' test matrices, the computational overhead of all the block steps is only a small fraction of the total overhead. For example, if we compare the computing times for `DGETE2` in Tables 7 and 8, we see that only about 20 % of the overhead is due to the many block steps. Again, we feel that this overhead is reasonable.

7 Conclusion

We have presented Fortran 77 implementations of the extended Levinson algorithms from [7, 8] for symmetric and nonsymmetric Toeplitz matrices. The subroutines are able to solve efficiently and reliable any Toeplitz system which does not have *many* consecutive ill-conditioned leading principal submatrices. In addition, reliable estimates of the algorithm and matrix condition numbers are produced. In comparison, the classical Levinson algorithm is only guaranteed to be numerically stable for symmetric positive definite Toeplitz matrices.

The computational overhead of these implementations of the extended Levinson algorithms, compared to the highly efficient implementation of the classical Levinson algorithm from the Argonne Toeplitz package [1], is not negligible. For nonsymmetric Toeplitz matrices with few ill-conditioned leading principal submatrices, the overhead is about 100 %. For matrices with many ill-conditioned leading principal submatrices, the overhead is typically about 120 %. On the other hand, the extended Levinson algorithm is able to solve a much larger class of Toeplitz systems than the classical Levinson algorithm, and in addition it produces an accuracy estimate for the computed solution. Moreover, if the extended algorithm fails to compute an accurate solution, as signaled to the user by a large ratio between the estimated algorithm and matrix condition numbers, then the necessary maximum block-size can be found by inspection of the list of smallest singular values of all the leading principal submatrices produced by the algorithm. The extended Levinson algorithm can then be re-run with this new block size.

References

[1] 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, 1983.

[2] C. H. Bischof, J. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling & D. Sorensen, *LAPACK working note # 5, provisional contents*, Report ANL-88-38, Argonne National Laboratory, 1988.

[3] 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.

[4] R. P. Brent, F. T. Luk & C. F. Van Loan, *Computation of the singular value decomposition using mesh-connected processors*, J. VLSI and Computer Systems **1** (1985), 242–270.

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

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

[7] T. F. Chan & P. C. Hansen, *A stable extension of Levinson's algorithm for indefinite Toeplitz systems*, CAM Report 89-20, Dept. of Mathematics, UCLA; to appear in SIAM J. Matrix Anal. Appl.

[8] T. F. Chan & P. C. Hansen, *A stable Levinson algorithm for general Toeplitz systems*, CAM Report 90-11, Dept. of Mathematics, UCLA; submitted to IEEE ASSP.

[9] G. Cybenko & C. F. Van Loan, *Computing the minimum eigenvalue of a symmetric positive definite Toeplitz matrix*, SIAM J. Sci. Stat. Comput. **7** (1986), 123–131.

[10] G. Dahlquist, Å. Björck & N. Anderson, *Numerical Methods*, Prentice Hall, 1974.

[11] 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** (1987), 964–971.

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

[13] J. J. Dongarra, J. R. Bunch, C. B. Moler & G. W. Stewart, *Linpack Users' Guide*, SIAM, 1979.

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

[15] M. J. C. Gover & S. Barnett, *Inversion of Toeplitz matrices which are not strongly non-singular*, IMA J. Numer. Anal. **5** (1985), 101–110.

[16] R. G. Grimes & J. G. Lewis, *Condition number estimation for sparse matrices*, SIAM J. Sci. Stat. Comput. **2** (1981), 384–388.

[17] IMSL Math/Library User's Manual, IMSL Inc., 1987.

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

[19] C. L. Lawson, R. J. Hanson, D. R. Kincaid & F. T. Krogh, *Basic linear algebra subprograms for Fortran use*, ACM Trans. Math. Soft. **5** (1979), 308–323.

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

[21] W. H. Press, B. P. Flannery, S. A. Teukolski & W. T. Vetterling, *Numerical Recipes*, Cambridge University Press, 1987.

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

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

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