A NUMERICAL COMPARISON OF LOOK-AHEAD LEVINSON AND SCHUR ALGORITHMS FOR NON-HERMITIAN TOEPLITZ SYSTEMS

MARLIS HOCHBRUCK*

Abstract. In two recent papers we proposed different types of look-ahead algorithms for recursively computing Padé forms located on two adjacent rows of the Padé table of a formal Laurent series. These methods can be interpreted as look-ahead Levinson and Schur algorithms for solving general non-Hermitian Toeplitz linear systems of equations.

In this paper, we discuss look-ahead strategies and options for solving Toeplitz systems. The main aim of this paper is a numerical comparison of different types of look-ahead methods and of various options for solving a Toeplitz system. We compare our implementation of look-ahead Levinson and Schur algorithms with the classical algorithms and with the look-ahead Levinson solver provided by Hansen and Chan.

1. Introduction. The solution of a general square linear system of dimension N requires $O(N^3)$ arithmetic operations. However, if the coefficient matrix is structured, *e.g.*, Hankel, Cauchy, Vandermonde, or Toeplitz, then there exist fast algorithms which require only $O(N^2)$ operations. In this paper, we consider the special case of linear systems with Toeplitz coefficient matrices $\mathbf{T} = \mathbf{T}_N$. Toeplitz matrices are characterized by having identical entries on each diagonal. The fast algorithms for this particular application can be classified into two categories: *Levinson* and *Schur algorithms*. Levinson algorithms [26, 28] compute an inverse LDU decomposition of $\mathbf{T}$. This inverse LDU decomposition allows one to solve not only the Toeplitz system of dimension N but as a byproduct also all leading principal subsystems, which is important in many practical applications. The dominating arithmetic operations of Levinson algorithms are inner products and SAXPYs. In contrast, Schur algorithms [4, 27] compute directly an LDU decomposition of $\mathbf{T}$. From a computational point of view, the main difference between the two types of algorithms is that Schur algorithms require only SAXPYs but no inner products. On a parallel computer with $O(N)$ processors, one can therefore solve a Toeplitz linear system with a Schur algorithm in $O(N)$ execution time. Another way of further reducing the operations of fast $O(N^2)$ Schur algorithms is due to the possibility of applying recursive doubling and fast $O(N \log N)$ polynomial multiplications. Such algorithms are called *superfast*; see *e.g.*, [1, 7].

However, all the algorithms we just mentioned and which we refer to as *classical* can break down due to a division by zero when they are applied to general non-Hermitian or to Hermitian indefinite Toeplitz matrices. While a division by zero may be a very rare event in practice, much more serious problems for implementations in finite precision arithmetic are instabilities arising from divisions by numbers of small absolute value. Such instabilities are called *near breakdowns*.

Recently, a number of extensions of the classical algorithms, so called *look-ahead methods* which can avoid breakdowns as well as near-breakdowns have been proposed, *e.g.*, [6, 9, 10, 14, 16, 17, 22, 29]. The problem of curing near breakdowns in classical algorithms can be tackled from different sides, namely from a purely linear algebra setting [6, 16], from displacement rank approaches [23, 25], from an interpretation in terms of formally biorthogonal polynomials [9, 10], and from its relation to Padé forms

* Mathematisches Institut, Universität Tübingen, Auf der Morgenstelle 10, D–72076 Tübingen, Germany. E-mail: marlis@na.uni-tuebingen.de

in two adjacent rows of the Padé table of a formal Laurent series [14, 17, 22, 29]. Some of the fast look-ahead Schur algorithms can also be extended to superfast look-ahead methods [14, 15, 16]. From a Padé theory point of view, Levinson algorithms compute denominators of Padé forms and Schur algorithms work with numerators and residuals of Padé forms.

Here we present numerical experiments with two look-ahead algorithms for computing Padé forms in adjacent rows of the Padé table which we proposed recently in [17, 22]. We combined these algorithms with procedures for solving linear systems with non-Hermitian Toeplitz coefficient matrices. We have written a C package with different variants of look-ahead Levinson and Schur algorithms combined with various options for solving the Toeplitz systems. The aim of this paper is to report timings and achieved accuracies for the algorithms of [17, 22] and to compare these to other existing methods. However, to the best of our knowledge, the only implementation of a look-ahead method available on the network is the look-ahead Levinson algorithm by Chan and Hansen [6], so that we could compare our results only to the package [18] containing the algorithms from [6]. In order to illustrate the additional work one has to perform when instabilities in the classical algorithms arise, we also present results obtained with the classical algorithms. It turns out that already for moderate dimensions the overhead of our look-ahead algorithms is negligible but the gain of accuracy is significant.

For a detailed description of look-ahead algorithms and for further references we refer to [14, 16, 17, 22].

The paper starts with providing notation and basic definitions used in this paper. It then briefly recalls the classical Levinson and Schur algorithm written in polynomial form in Section 3. After a sketch of different look-ahead algorithms in Section 4, we discuss look-ahead strategies in Section 5. We continue with a survey on various options for solving Toeplitz systems from the output we obtain from look-ahead Levinson and Schur algorithms. In Section 7 we analyze the complexity of these different options. Finally we present numerical examples with our C implementation of all the different algorithms in Section 8. A C package containing the look-ahead Toeplitz solvers developed in [17, 22] is available from the author.

2. Basic definitions. In the following,

$$\begin{aligned}\mathcal{L}_l &:= \{h \in \mathcal{L}\,;\ \mu_k = 0 \text{ if } k < l\},\\ \mathcal{L}_m^* &:= \{h \in \mathcal{L}\,;\ \mu_k = 0 \text{ if } k > m\},\\ \mathcal{P}_m &:= \{p \in \mathcal{L}\,;\ p \text{ polynomial of degree at most } m\}\end{aligned}$$

denote subsets of $\mathcal{L}$, the set of formal Laurent series with complex coefficients,

$$h(\zeta) := \sum_{k=-\infty}^{\infty} \mu_k \zeta^k, \tag{2.1}$$

and $\mathcal{P}$, the set of all polynomials. The formal projection of $h \in \mathcal{L}$ into $\mathcal{L}_l \cap \mathcal{L}_m^*$ is denoted by

$$\Pi_{l:m} h(\zeta) := \sum_{j=l}^{m} \mu_j \zeta^j$$

and the mth coefficient of h is written as $\Pi_m h(\zeta) := \mu_m$. We write $h(\zeta) = O_+(\zeta^l)$ if $h(\zeta) \in \mathcal{L}_l$.

Unless indicated otherwise, $\|\cdot\| = \|\cdot\|_2$ always denote the Euclidean or spectral norm.

For $m \in \mathbb{Z}$ and $n \in \mathbb{N}$, where $\mathbb{Z}$ is the set of all integers and $\mathbb{N}$ is the subset of all nonnegative integers, an (m,n) *Padé form* of h is any pair $(p,q) \in \mathcal{L}_m^* \times (\mathcal{P}_n \backslash \{0\})$ satisfying

$$h(\zeta)q(\zeta) - p(\zeta) = O_+(\zeta^{m+n+1}) \in \mathcal{L}_{m+n+1}. \tag{2.2}$$

The series $e \in \mathcal{L}_0$ defined implicitly by

$$h(\zeta)q(\zeta) - p(\zeta) = \zeta^{m+n+1}e(\zeta) \tag{2.3}$$

is called the *residual* of (p,q).

Padé approximations can be listed in a Padé table, where we let the m-axis point to the bottom and the n-axis point to the right. In this paper, we deal with Padé forms in the (-1)st and the 0th row of the Padé table. A key point to our look-ahead algorithms is the use of regular $(0,n)$ Padé forms. A $(0,n)$ Padé form (p_n, q_n) is *regular* if and only if the Toeplitz matrix

$$\mathbf{T}_n := \begin{bmatrix} \mu_0 & \cdots & \mu_{-n+1} \\ \vdots & \ddots & \vdots \\ \mu_{n-1} & \cdots & \mu_0 \end{bmatrix} \in \mathbb{C}^{n,n} \tag{2.4}$$

is nonsingular. We call a $(0,n)$ Padé form well-conditioned iff $\mathbf{T}_n$ is *well conditioned.*

The classical algorithms work with *column-regular* (*column well-conditioned*) pairs, which are characterized by $\mathbf{T}_n$ and $\mathbf{T}_{n+1}$ being nonsingular (well conditioned), see [14, 16, 17] for details and other equivalent definitions of (column) regular pairs.

According to their location with respect to (p_n, q_n), the neighbors of a $(0,n)$ Padé form (p_n, q_n) are denoted by arrows *i.e.*, we denote by $(p_n^{\nwarrow}, q_n^{\nwarrow})$ a $(-1, n-1)$ Padé form, and by $(p_n^{\uparrow}, q_n^{\uparrow})$ a $(-1, n)$ Padé form of h. The corresponding residuals are denoted in the same way. The following picture shows the location of these Padé forms and illustrates the notation.

$$\begin{array}{|lll|} \hline -1 & & q_n^{\nwarrow} \quad q_n^{\uparrow} \\ 0 & & q_n \\ & & \\ & n-1 & n \\ \hline \end{array} \tag{2.5}$$

If (p_n, q_n) is a regular Padé form, then we call n a regular index. In this case, the Padé forms shown in the picture can be computed by solving linear systems with coefficient matrix $\mathbf{T}_n$. They are uniquely determined up to scaling. Once we have q_n, we obtain $p_n = \Pi_{-\infty:0}(hq_n)$ and $e_n = \Pi_{n+1:\infty}(hq_n)$ from formal projections, and similar for the other Padé forms. Setting

$$q_n(\zeta) = \sum_{j=0}^{n} \rho_{j,n}\zeta^j$$

and using the analogous notation for $q_n^{\uparrow}$ and $q_n^{\nwarrow}$, we have the Yule-Walker equations

$$\mathbf{T}_n \begin{bmatrix} \rho_{1,n} \\ \vdots \\ \rho_{n,n} \end{bmatrix} = -\rho_{0,n} \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_n \end{bmatrix}, \qquad \mathbf{T}_n \begin{bmatrix} \rho_{0,n}^{\uparrow} \\ \vdots \\ \rho_{n-1,n}^{\uparrow} \end{bmatrix} = -\rho_{n,n}^{\uparrow} \begin{bmatrix} \mu_{-n} \\ \vdots \\ \mu_{-1} \end{bmatrix}$$

and

$$\mathbf{T}_n \begin{bmatrix} \rho_{0,n}^{\nwarrow} \\ \vdots \\ \rho_{n-1,n}^{\nwarrow} \end{bmatrix} = e_n^{\nwarrow}(0) \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}.$$

If $\mathbf{T}_n$ is regular, *i.e.*, if the $(0,n)$ Padé form is regular, then the coefficients of the right-hand side vectors cannot vanish. Hence we can use the normalizations

$$\rho_{0,n} = 1, \quad \rho_{n,n}^{\uparrow} = 1, \quad e_n^{\nwarrow}(0) = 1. \tag{2.6}$$

For abbreviation we define

$$\varepsilon_{n,n} = e_n(0) \quad \text{and} \quad \pi_{0,n} = p_n(\infty) = \Pi_0(p_n),$$

and we use the analogous notation for the neighbors of the $(0,n)$ Padé form.

In this paper, we consider only well-conditioned linear systems and we assume that the length of the look-ahead steps as well as the total number of such steps are bounded independently of the dimension N. This justifies to consider all computations of order at most $O(k_{\max}^3)$ as negligible, where $k_{\max}$ is the maximal length of look-ahead steps if N is large; see Figure 8.1 below.

3. Classical Levinson and Schur algorithms. Let us first give an interpretation of the classical Levinson and Schur algorithms in terms of Padé approximation; see *e.g.*, [5, 14]. Basically, these algorithms proceed from a column-regular $(0,n)$ Padé form (p_n, q_n) and its upper neighbor $(p_n^{\uparrow}, q_n^{\uparrow})$, a $(-1,n)$ Padé form, to the next column-regular pair. This works exactly as long as the Padé form (p_n, q_n) is column-regular, but the recurrence breaks down when this is not the case.

In Table 3.1, we give the well-known classical algorithms written in our notation. Let us start with commenting on these algorithms. Step 1 is the initialization phase. For the Levinson algorithm, initialization is done with scalars. Here, the degree of the polynomials $q_n^{\uparrow}$ and q_n (denominators of Padé forms) is equal to the index n. The Schur algorithm starts with polynomials in ζ (residuals) or ζ^{-1} (numerators) of degree at most N, if $h \in \mathcal{L}_{-N} \cap \mathcal{L}_N^*$. For this algorithm, the degree of the polynomials decreases with the iteration index n. Note that we use only two polynomials for the Levinson algorithm but four for the Schur algorithm. In Step 2, the Levinson algorithm computes the leading coefficients of $p_n^{\uparrow}$ and e_n, since these are not known here. This requires two inner products of length $n+1$. There is nothing to compute for the Schur algorithm, where these coefficients are available anyway. The Schur parameters are computed in Step 3. They are identical for both methods. Steps 4 and 5 contain the recursions for the denominators of the $(-1,n+1)$ and the $(0,n+1)$ Padé form in the Levinson algorithm and for the numerators and residuals of these Padé forms in the Schur algorithm. Note that all the recurrences are of the same

	Levinson algorithm	Schur algorithm
1.	$q_0^{\uparrow} = q_0 = 1$ $\varepsilon_{0,0}^{\uparrow} = \mu_0$	$p_0^{\uparrow} = \Pi_{-\infty:-1}\, h$, $p_0 = \Pi_{-\infty:0} h$ $e_0^{\uparrow} = \Pi_{0:\infty}\, h$, $e_0 = \Pi_{1:\infty}\, h$
	for $n = 0, 1, \ldots$	
2.	$\pi_{0,n}^{\uparrow} = \Pi_{-1}(hq_n^{\uparrow})$, $\varepsilon_{n,n} = \Pi_{n+1}(hq_n)$	
3.	$\gamma_0^{\uparrow(n)} = -\pi_{0,n}^{\uparrow}/\varepsilon_{n,n}$, $\gamma_0^{(n)} = -\varepsilon_{n,n}/\varepsilon_{n,n}^{\uparrow}$	
4.	$q_{n+1}^{\uparrow} = \zeta q_n^{\uparrow} + q_n \gamma_0^{\uparrow(n)}$	$p_{n+1}^{\uparrow} = \zeta p_n^{\uparrow} + p_n \gamma_0^{\uparrow(n)}$ $e_{n+1}^{\uparrow} = e_n^{\uparrow} + e_n \gamma_0^{\uparrow(n)}$
5.	$q_{n+1} = \zeta q_n^{\uparrow} \gamma_0^{(n)} + q_n$	$p_{n+1} = \zeta p_n^{\uparrow} \gamma_0^{(n)} + p_n$ $e_{n+1} = \zeta^{-1}(e_n^{\uparrow} \gamma_0^{(n)} + e_n)$
6.	$\varepsilon_{n+1,n+1}^{\uparrow} = \varepsilon_{n,n}(1 - \gamma_0^{\uparrow(n)}/\gamma_0^{(n)})$	

TABLE 3.1
Classical Levinson and Schur algorithm

type. This follows from the fact, that denominators, numerators, and residuals satisfy the same recurrence relations, a result which is well-known in Padé theory. Finally, a scalar recursion is needed for updating the leading coefficient of $e_{n+1}^{\uparrow}$ in the Levinson algorithm in Step 6.

For each n, the Levinson algorithm requires two inner products of length $n+1$ in Step 2 and two SAXPYs of length $n+1$ in Steps 4 and 5. In contrast, for $h \in \mathcal{L}_{-N+1} \cap \mathcal{L}_{N-1}^{*}$, which corresponds to solving a Toeplitz system of dimension N, the Schur algorithm requires two SAXPYs of length $N-n$ for p_{n+1} and $e_{n+1}^{\uparrow}$ and two SAXPYs of length $N-n-1$ for $p_{n+1}^{\uparrow}$ and e_{n+1}. This yields an operation count of $2N^2 + O(N)$ flops for both algorithms. Here and in the following, we call an operation of type $a + bc$ with scalars a, b, and c a *flop*.

Clearly, the formulas for the Schur parameters in Step 3 show that both algorithms break down if $\varepsilon_{n,n} = 0$ or $\varepsilon_{n,n}^{\uparrow} = 0$. For an implementation in finite precision arithmetic, it is more likely that they become unstable if one of these quantities have small absolute value. Then we have to look-ahead, for example by one of the methods described in [17, 22], which we sketch in the next section or by the algorithms given in [6, 9, 10, 14, 16].

4. Look-ahead Levinson and Schur algorithms. To include look-ahead into the classical algorithms from Table 3.1, we proposed in [17] and [22] to use the recurrences from the classical algorithm whenever possible and to switch to recurrences based on regular pairs (p_n, q_n) and $(p_n^{\nwarrow}, q_n^{\nwarrow})$ otherwise. Mixing recurrences based on column-regular and on regular pairs has the desirable consequence that when the algorithm is applied to a Toeplitz matrix for which no look-ahead is necessary, then it reduces to the classical algorithm and does not waste any operations. In addition, using regular pairs instead of column-regular ones (as in the classical algorithms) ensures that the algorithms perform look-ahead steps of minimal size. This is not the case for the algorithms we proposed in [16]. When we switch from a classical step to a

Overhead for a block of size $k > 1$	Levinson-type		Schur-type
	SAXPYs	inner products	SAXPYs
Hochbruck [22]	$2k-1$	–	$4k-2$
Gutknecht/Hochbruck [17]	$4k-3$	–	$6k-5$
Freund/Zha [10], Freund [9]	$4k$	–	$6k$
Chan/Hansen [6]	$4k-4$	$(k^2+k)/2$	not proposed

TABLE 4.1
Overhead of different look-ahead algorithms for solving Toeplitz systems.

look-ahead step, then we clearly have $(p_n^{\nwarrow}, q_n^{\nwarrow}) = (p_{n-1}^{\uparrow}, q_{n-1}^{\uparrow})$; see Picture (2.5). After a look-ahead step, the auxiliary Padé form $(p_n^{\nwarrow}, q_n^{\nwarrow})$ has to be computed separately.

The basic idea of performing look-ahead steps in the algorithms developed in [17] is to "jump" from one regular pair to the next one by multiplying the regular pair by a suitable two-point Padé form. The computational operations we have to perform are multiplications by polynomials of low degree, namely the length of the look-ahead step. Therefore, we refer to this type of algorithm as the *product form* of a look-ahead algorithm. In practical situations, these steps are mostly small, say two to four or five. In order to fill the gaps between two regular pairs, so called *underdetermined Padé forms* have been introduced, which can also be computed from a regular pair. The denominators of regular Padé forms and their upper neighbors can be interpreted as *regular formally biorthogonal polynomials* (FBOPs) with respect to a bilinear form defined by its moments μ_j. Denominators of underdetermined Padé forms correspond to *inner formally biorthogonal polynomials*; see [14, 17, 22] for details.

In contrast to these algorithms, the look-ahead recurrences proposed in [22] compute the next regular pair by forming a linear combination of the previous regular pair and underdetermined Padé forms in between. It turns out that these variants are cheaper than the product forms of the look-ahead Levinson and Schur algorithms.

The overhead of the different look-ahead algorithms for computing a complete (inverse) block LDU decomposition is summarized in Table 4.1. With overhead we mean the difference of the number of operations for a look-ahead step of size k and k steps of the respective classical algorithm. The algorithm of Chan and Hansen is the only one which has overhead also in steps of length $k = 1$.

For illustrations of the different look-ahead methods and for details we refer to [17, 22]. Here we only want to give the relation between certain coefficients of Padé forms in two adjacent rows of the Padé table of the Laurent series h and the (inverse) block LDU factorization of the Toeplitz matrix $\mathbf{T} = \mathbf{T}_N$, because these factorizations play an important role for the solution of Toeplitz systems. If we define $\mathbf{R} = \mathbf{R}_N$ and $\mathbf{R}^{\uparrow} = \mathbf{R}_N^{\uparrow}$, where

$$(4.1)\quad \mathbf{R}_N^{\uparrow} := \begin{bmatrix} \rho_{0,0}^{\uparrow} & \rho_{0,1}^{\uparrow} & \cdots & \rho_{0,N-1}^{\uparrow} \\ & \rho_{1,1}^{\uparrow} & \cdots & \rho_{1,N-1}^{\uparrow} \\ & & \ddots & \vdots \\ & & & \rho_{N-1,N-1}^{\uparrow} \end{bmatrix}, \quad \mathbf{R}_N := \begin{bmatrix} \rho_{0,0} & \rho_{1,1} & \cdots & \rho_{N-1,N-1} \\ & \rho_{0,1} & \cdots & \rho_{N-2,N-1} \\ & & \ddots & \vdots \\ & & & \rho_{0,N-1} \end{bmatrix},$$

and $\mathbf{D} = \mathbf{D}_N$ via

$$(4.2)\qquad \mathbf{D} := \mathbf{R}^T \mathbf{T} \mathbf{R}^{\uparrow},$$

then the following theorem holds

THEOREM 4.1. [17, Theorem 8.1] *If the upper triangular matrices* $\mathbf{R}$ *and* $\mathbf{R}^{\uparrow}$ *defined in* (4.1) *contain in their* $(n+1)$*st column* $(0 \le n < N)$ *the coefficients of* n*th regular or inner formally biorthogonal polynomials, where the latter are constructed as described in* [17, Section 7] *or* [22, Theorem 3]*, then the matrix* $\mathbf{D}$ *in* (4.2) *is block diagonal. The blocks of* $\mathbf{D}$ *always start with a regular index and each block is a Toeplitz matrix. Moreover, the* $n \times n$ *principal submatrix* $\mathbf{D}_n$ *of* $\mathbf{D}$ *is nonsingular if and only if* $\mathbf{T}_n$ *is nonsingular.*

The size of a block of $\mathbf{D}$, which corresponds to the length of a look-ahead step, is equal to one plus the number of consecutive singular or ill-conditioned principal submatrices of $\mathbf{T}$. If $\mathbf{T}$ has only a few singular or ill-conditioned principal submatrices, then $\mathbf{D}$ has mostly blocks of size one and a few small blocks.

Note, that (4.2) is equivalent to

$$\mathbf{T}^{-1} = \mathbf{R}^{\uparrow}\mathbf{D}^{-1}\mathbf{R}^{T}, \tag{4.3}$$

and therefore, the upper triangular matrices $\mathbf{R}$ and $\mathbf{R}^{\uparrow}$ containing the coefficients of denominators of Padé forms in the 0th and (-1)st row of the Padé table of h define an inverse block LDU decomposition of $\mathbf{T}$. A block LDU decomposition of $\mathbf{T}$ is obtained from coefficients of residuals and numerators of Padé forms in these rows of the Padé table. To be more precise, we have

THEOREM 4.2. [14, Theorem 7.1] *Let the assumptions of Theorem* 4.1 *be valid and define*

$$\mathbf{L} = \mathbf{T}^{T}\mathbf{R} \quad \textit{and} \quad \mathbf{L}^{\uparrow} = \mathbf{T}\mathbf{R}^{\uparrow}. \tag{4.4}$$

Then, $\mathbf{L}$ *and* $\mathbf{L}^{\uparrow}$ *are block lower triangular matrices where* $\mathbf{L}$ *contains in its* $(n+1)$*st column the coefficients of the numerator* p_n *of a* $(0,n)$ *(underdetermined) Padé form and* $\mathbf{L}^{\uparrow}$ *contains in its* $(n+1)$*st column the coefficients of the residual* $e_n^{\uparrow}$ *of a* $(-1,n)$ *(underdetermined) Padé form* $(0 \le n < N)$*. Moreover,*

$$\mathbf{T} = \mathbf{L}^{\uparrow}\mathbf{D}^{-1}\mathbf{L}^{T} \tag{4.5}$$

is a block LDU decomposition of $\mathbf{T}$*.*

An equivalent expression for the block diagonal matrix $\mathbf{D}$ defined in (4.2) is obtained from (4.4):

$$\mathbf{D} = \mathbf{R}^{T}\mathbf{L}^{\uparrow} = \mathbf{L}^{T}\mathbf{R}^{\uparrow}. \tag{4.6}$$

Writing the nth step of the classical algorithm compactly as

$$\begin{bmatrix} p_{n+1} & p_{n+1}^{\uparrow} \\ q_{n+1} & q_{n+1}^{\uparrow} \end{bmatrix} = \begin{bmatrix} p_n & p_n^{\uparrow} \\ q_n & q_n^{\uparrow} \end{bmatrix} \begin{bmatrix} 1 & \gamma_0^{\uparrow(n)} \\ \zeta\gamma_0^{(n)} & \zeta \end{bmatrix}, \tag{4.7}$$

and defining

$$\mathbf{L}^{\uparrow(\mathrm{J})} := \mathbf{J}\mathbf{L}^{\uparrow}\mathbf{J} \quad \text{and} \quad \mathbf{R}^{\uparrow(\mathrm{J})} := \mathbf{J}\mathbf{R}^{\uparrow}\mathbf{J},$$

where $\mathbf{J}$ denotes the antidiagonal unit matrix or reflection matrix of order N, we can summarize the classical Schur algorithm in matrix form as

$$[\ \mathbf{T}\ \big|\ \mathbf{T}\ ] \prod_{n=0}^{N-2} \mathbf{C}_n = [\ \mathbf{L}\ \big|\ \mathbf{L}^{\uparrow(\mathrm{J})}\ ], \tag{4.8}$$

where

$$\mathbf{C}_n := \left[\begin{array}{c|c|c|c} \mathbf{I}_{n+1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{N-n-1} & \gamma_0^{\uparrow(n)}\mathbf{I}_{N-n-1} & \mathbf{0} \\ \mathbf{0} & \gamma_0^{(n)}\mathbf{I}_{N-n-1} & \mathbf{I}_{N-n-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_{n+1} \end{array}\right], \tag{4.9}$$

with the Schur parameters $\gamma_0^{(n)}$ and $\gamma_0^{\uparrow(n)}$ from Table 3.1. Likewise, the Levinson algorithm becomes

$$[\ \mathbf{I}\ |\ \mathbf{I}\ ] \prod_{n=0}^{N-2} \mathbf{C}_n = [\ \mathbf{R}\ |\ \mathbf{R}^{\uparrow(\mathrm{J})}\ ]. \tag{4.10}$$

These representations were derived in [16] as a reformulation of the Bareiss algorithm [4] and of look-ahead algorithms based on column-regular pairs. In [21], we have generalized it to cover the look-ahead methods from [17, 22] also, which is a little more complicated because the algorithms use column-regular as well as regular pairs. The reason for the difficulties is that the coefficients of the upper left neighbor of a regular pair is in general not contained in the block factorization of $\mathbf{T}$. However, since we do not want to describe these algorithms in detail here again, we do not present the matrix formulation for the look-ahead case in this paper. We only want to mention this formulation, because it leads to an efficient way of solving Toeplitz systems with the Schur algorithm, as we will see in Section 6.4. For detail we refer to [21].

5. Look-ahead strategies. A crucial point in designing look-ahead algorithms is a criterion for determining the length of a look-ahead step. The main purpose of this criterion is to determine the well-conditioned submatrices of $\mathbf{T}$ in order to compute the column-regular or regular pairs used in our algorithms. From Table 4.1 it becomes clear that look-ahead steps are more expensive than classical steps, so the aim of the criterion is also to avoid look-ahead steps whenever possible.

Since all our algorithms involve the solution of one or two small linear systems, mostly of dimension one, the weakest criterion one has to ensure is to guarantee that the coefficient matrices of all the linear systems which arise are nonsingular. In exact arithmetic, this would ensure that the look-ahead strategy exactly determines the nonsingular principal submatrices of $\mathbf{T}$ and that the length of a look-ahead step corresponds to one plus the number of consecutive singular principal submatrices, see Theorem 4.1. In practice, when the computations are done in finite precision arithmetic, we will require that these matrices are well-conditioned. To achieve this task, we propose to keep the smallest singular value of these coefficient matrices above some tolerance. For the algorithms in product form of [17] we check

$$\sigma_{\min}(\mathbf{M}_{k,n}) > \mathrm{tol}(n), \tag{5.1}$$

where $\mathbf{M}_{k,n}$ is the matrix defined in formula (6.5) of [17]. For the algorithms involving underdetermined Padé forms described in [22] we test on

$$\min\{\sigma_{\min}(\mathbf{L}^{\uparrow(l)}), \sigma_{\min}(\mathbf{L}^{(l)})\} > \mathrm{tol}(n), \tag{5.2}$$

where $n = n_l$ is the index of the last well-conditioned pair. By Theorem 4.2, $\mathbf{L}^{\uparrow(l)}$ is the coefficient matrix in (4.1) of [22], and $\mathbf{L}^{(l)}$ is the coefficient matrix in (4.4) of [22].

Note that at this point we allow the tolerances to depend on the index n. If we check if $n+k$ is a regular index, then $\mathbf{M}_{k,n}$ is a $(2k-1)\times(2k-1)$ matrix and $\mathbf{L}^{\uparrow(l)}$ and $\mathbf{L}^{(l)}$ are $k\times k$ matrices.

For the solution of Toeplitz systems, we implemented the checks (5.1) and (5.2) for different tolerances $\mathrm{tol}(n)$ and found out that a value of the order 0.1 gives satisfactory results for this application. Additionally, we restricted the length of a look-ahead step to $k_{\max}$. If the checks failed for each $1\le k\le k_{\max}$, we set k to the value for which the left-hand side of (5.1) or (5.2) attained its maximum. Only if this maximum is below a tolerance which is of the order of machine precision, the algorithm fails and should be restarted with a larger value of $k_{\max}$. However, this did not happen in any of the examples presented in Section 8.

To provide a heuristic explanation, we first recall two well-known inversion formulas. The first is due to Heinig.

THEOREM 5.1. [20, Remark 1.1] *Let (p,q) be a regular (m,n) Padé form of $h\in\mathcal{L}$, $(p^{\nwarrow},q^{\nwarrow})$ be an $(m-1,n-1)$, normalized by (2.6). Then $\mathbf{T}_{m,n}$ is nonsingular and*

$$
\begin{aligned}
\mathbf{T}_{m,n}^{-1} = {} & \begin{bmatrix} \rho_{n-1}^{\nwarrow} & \rho_{n-2}^{\nwarrow} & \cdots & \rho_0^{\nwarrow} \\ & \rho_{n-1}^{\nwarrow} & \ddots & \vdots \\ & & \ddots & \rho_{n-2}^{\nwarrow} \\ \mathbf{0} & & & \rho_{n-1}^{\nwarrow} \end{bmatrix} \begin{bmatrix} \rho_0 & & & \mathbf{0} \\ \rho_1 & \rho_0 & & \\ \vdots & \ddots & \ddots & \\ \rho_{n-1} & \cdots & \rho_1 & \rho_0 \end{bmatrix} \\
& - \begin{bmatrix} \rho_n & \rho_{n-1} & \cdots & \rho_1 \\ & \rho_n & \ddots & \vdots \\ & & \ddots & \rho_{n-1} \\ \mathbf{0} & & & \rho_n \end{bmatrix} \begin{bmatrix} 0 & & & \mathbf{0} \\ \rho_0^{\nwarrow} & 0 & & \\ \vdots & \ddots & \ddots & \\ \rho_{n-2}^{\nwarrow} & \cdots & \rho_0^{\nwarrow} & 0 \end{bmatrix}.
\end{aligned} \tag{5.3}
$$

The next inversion formula based on column-regular pairs is due to Gohberg and Semencul [12].

THEOREM 5.2. [20, Theorem 1.2] *Let (p,q) be a column-regular (m,n) Padé form of $h\in\mathcal{L}$, and $(p^{\uparrow},q^{\uparrow})$ be an $(m-1,n)$ Padé form of h, normalized by (2.6). Then $\pi_0=\epsilon^{\uparrow}(0)\neq 0$, $\mathbf{T}_{m,n+1}$ is nonsingular,*

$$
\begin{aligned}
\mathbf{T}_{m,n+1}^{-1} = {} & \frac{1}{\pi_0} \begin{bmatrix} \rho_0 & & & \mathbf{0} \\ \rho_1 & \rho_0 & & \\ \vdots & \ddots & \ddots & \\ \rho_n & \cdots & \rho_1 & \rho_0 \end{bmatrix} \begin{bmatrix} \rho_n^{\uparrow} & \rho_{n-1}^{\uparrow} & \cdots & \rho_0^{\uparrow} \\ & \rho_n^{\uparrow} & \ddots & \vdots \\ & & \ddots & \rho_{n-1}^{\uparrow} \\ \mathbf{0} & & & \rho_n^{\uparrow} \end{bmatrix} \\
& - \frac{1}{\pi_0} \begin{bmatrix} 0 & & & \mathbf{0} \\ \rho_0^{\uparrow} & 0 & & \\ \vdots & \ddots & \ddots & \\ \rho_{n-1}^{\uparrow} & \cdots & \rho_0^{\uparrow} & 0 \end{bmatrix} \begin{bmatrix} 0 & \rho_n & \cdots & \rho_1 \\ & 0 & \ddots & \vdots \\ & & \ddots & \rho_n \\ \mathbf{0} & & & 0 \end{bmatrix},
\end{aligned} \tag{5.4}
$$

and

$$
\begin{aligned}
\mathbf{T}_{m,n}^{-1} = \; & \frac{1}{\pi_0}
\begin{bmatrix} \rho_0 & & & \mathbf{0} \\ \rho_1 & \rho_0 & & \\ \vdots & \ddots & \ddots & \\ \rho_{n-1} & \cdots & \rho_1 & \rho_0 \end{bmatrix}
\begin{bmatrix} \rho_n^{\uparrow} & \rho_{n-1}^{\uparrow} & \cdots & \rho_1^{\uparrow} \\ & \rho_n^{\uparrow} & \ddots & \vdots \\ & & \ddots & \rho_{n-1}^{\uparrow} \\ \mathbf{0} & & & \rho_n^{\uparrow} \end{bmatrix} \\
& - \frac{1}{\pi_0}
\begin{bmatrix} \rho_0^{\uparrow} & & & \mathbf{0} \\ \rho_1^{\uparrow} & \rho_0^{\uparrow} & & \\ \vdots & \ddots & \ddots & \\ \rho_{n-1}^{\uparrow} & \cdots & \rho_1^{\uparrow} & \rho_0^{\uparrow} \end{bmatrix}
\begin{bmatrix} \rho_n & \rho_{n-1} & \cdots & \rho_1 \\ & \rho_n & \ddots & \vdots \\ & & \ddots & \rho_{n-1} \\ \mathbf{0} & & & \rho_n \end{bmatrix}.
\end{aligned} \tag{5.5}
$$

From the inversion formulas of Theorems 5.1 and 5.2 we have the upper bounds

$$\|\mathbf{T}_n^{-1}\| \leq \|\mathbf{T}_n^{-1}\|_F \leq 2n \, \|\mathbf{q}_n^{\nwarrow}\| \, \|\mathbf{q}_n\| \tag{5.6}$$

and

$$\max\{\|\mathbf{T}_n^{-1}\|, \|\mathbf{T}_{n+1}^{-1}\|\} \leq \max\{\|\mathbf{T}_n^{-1}\|_F, \|\mathbf{T}_{n+1}^{-1}\|_F\} \leq \frac{2n}{|\pi_{0,n}|} \|\mathbf{q}_n\| \, \|\mathbf{q}_n^{\uparrow}\|, \tag{5.7}$$

where

$$\mathbf{q}_n = [\ \rho_{0,n} \ \cdots \ \rho_{n,n}\ ]^T, \quad \mathbf{q}_n^{\uparrow} = [\ \rho_{0,n}^{\uparrow} \ \cdots \ \rho_{n,n}^{\uparrow}\ ]^T, \quad \mathbf{q}_n^{\nwarrow} = [\ \rho_{0,n}^{\nwarrow} \ \cdots \ \rho_{n-1,n}^{\nwarrow}\ ]^T.$$

Therefore, $|\pi_{0,n}|$ and the norms of $\mathbf{q}_n$, $\mathbf{q}_n^{\uparrow}$ in a generic step and the norms of $\mathbf{q}_n^{\nwarrow}$ and $\mathbf{q}_n$ in a look-ahead step yield explicit upper bounds on the inverse of the principal submatrix $\mathbf{T}_n$. Since the linear systems with coefficient matrices $\mathbf{M}_{k,n}$, $\mathbf{L}^{\uparrow(l)}$, and $\mathbf{L}^{(l)}$ that we have to solve in the various algorithms contain in their right-hand sides coefficients of residuals and numerators of Padé forms, we can conclude that these right-hand sides are small if the norms of the coefficient vectors $\mathbf{q}_n^{\nwarrow}$ and $\mathbf{q}_n$ are sufficiently small, because we assumed that $\mathbf{T}_N$ itself is well conditioned. Then the checks (5.1) and (5.2) guarantee that the solutions of these linear systems cannot be large. From the triangle inequality we then conclude that the norms of $\mathbf{q}_{n+k}^{\nwarrow}$ and $\mathbf{q}_{n+k}$ cannot grow too much. Recall that for $n = n_l$ and $k = 1$ we have $\pi_0 = \mathbf{L}^{(l)} = \mathbf{L}^{\uparrow(l)}$.

Hence, a possible strategy to decide whether a pair of $(0, n)$ and $(-1, n)$ Padé forms is column well-conditioned could be based on the two inequalities

$$|\pi_0| > \mathrm{tol}(n) \qquad \text{and} \qquad \max\{\|\mathbf{q}_n\|, \|\mathbf{q}_n^{\uparrow}\|\} \leq \mathrm{Tol}(n).$$

This would guarantee that

$$\max\{\|\mathbf{T}_n^{-1}\|, \|\mathbf{T}_{n+1}^{-1}\|\} \leq 2n \, \frac{[\mathrm{Tol}(n)]^2}{\mathrm{tol}(n)}. \tag{5.8}$$

For a well-conditioned pair one would require

$$\max\{\|\mathbf{q}_n^{\nwarrow}\|, \|\mathbf{q}_n\|\} \leq \mathrm{Tol}(n),$$

which yields the upper bound

$$\|\mathbf{T}_n^{-1}\| \leq 2n \, [\mathrm{Tol}(n)]^2. \tag{5.9}$$

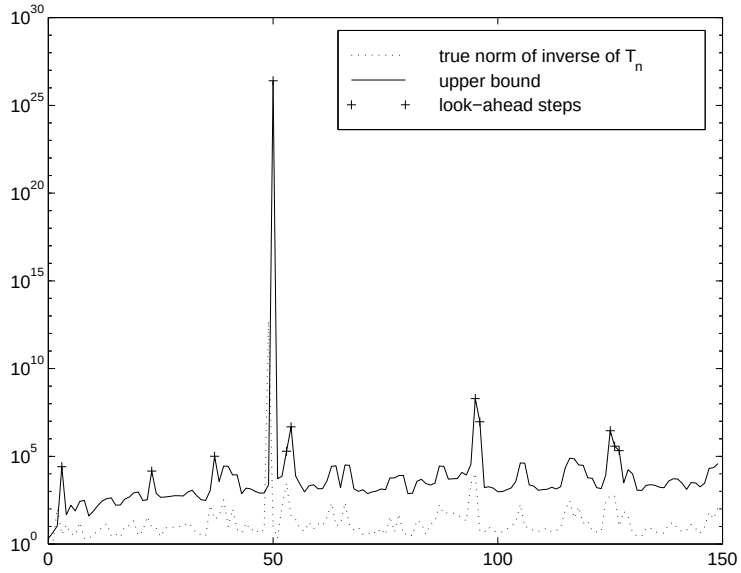

FIG. 5.1. $\|\mathbf{T}_n^{-1}\|$ *versus* n, *upper bounds* (5.6) *and* (5.7).

However, computing the norms is quite expensive and the bounds (5.8) and (5.9) turned out to be fairly pessimistic in practice. Extensive numerical tests have shown that the heuristic argument we explained above is justified in practice and that the cheap tests are sufficient to achieve good accuracy.

In order to manifest this statement, we created a random Toeplitz matrix of dimension 150 with a very ill-conditioned leading principal submatrix of dimension 50. Examples with an ill-conditioned principal submatrix of dimension n can be constructed from a random Toeplitz matrix by first computing the eigenvalues of the nth principal submatrix and then subtracting a suitable scalar multiple of the identity from the original Toeplitz matrix. For the test example of dimension 150, we monitor for each column well-conditioned index the upper bound (5.6) versus n. For every n that is not column well-conditioned, we computed the pair $\mathbf{q}_n^{\nwarrow}$, $\mathbf{q}_n$, no matter whether it was well-conditioned or not. Of course, we used only the well-conditioned pairs to proceed in the algorithm. Figure 5.1 shows the true spectral norms of $\mathbf{T}_n^{-1}$ versus n in a logarithmic plot (dotted line), the upper bound (5.6) if n is column well-conditioned and the upper bound (5.7) otherwise (solid line). The + marks indicate that a look-ahead step was performed at this index. The picture clearly shows that the simple bounds correctly detect ill-conditioned submatrices.

A different look-ahead strategy was proposed by Freund and Zha [10]. It is based on the Schur complement of $\mathbf{T}_{n+k}$. In a number of numerical tests we found no significant difference in the numerical behavior of our algorithms when implementing the tests from [10]. We therefore omit the details here and refer the reader to the original paper [10].

In contrast to these strategies, which involve only local information, Chan and Hansen [6] suggest to estimate the condition numbers of *all* principal submatrices of $\mathbf{T}_N$. This makes the algorithm more expensive because it requires computational overhead compared to the classical algorithm even if no look-ahead steps are taken.

If one does not want to set the tolerances before starting the algorithm, then one can determine the singular values of all matrices which arise in the linear systems for the recurrence coefficients for steps of length $1 \le k \le k_{\max}$ and then set the tolerance to the optimal value within these initial steps. A similar choice was proposed by Van Barel and Bultheel [29].

6. Options for solving Toeplitz systems. We have seen that look-ahead Levinson and Schur algorithms yield block LDU decompositions of $\mathbf{T}^{-1}$ and $\mathbf{T}$, respectively, at least if we compute additionally inner formally biorthogonal polynomials (FBOPs) or numerators and denominators of underdetermined Padé forms and the block diagonal matrix $\mathbf{D}$. Clearly, when one of these LDU decompositions is known, it is easy to solve a Toeplitz system

$$\mathbf{T}\mathbf{x} = \mathbf{b} \tag{6.1}$$

in roughly N^2 flops. However, this is just one way of applying the outcome of our algorithms to this task. There are several other options, which we want to compare here with these two, see also [16].

6.1. Inverse block LDU decomposition. All our look-ahead Levinson algorithms and also the algorithms in [6, 10] generate directly an inverse block LDU decomposition (4.3) if we compute not only regular formally biorthogonal polynomials but also fill the gaps with inner formally biorthogonal polynomials. This was shown in Theorem 4.1. These decompositions can be used to update the solutions of a nested sequence of Toeplitz systems of order n, $n = 1, 2, \ldots, N$:

$$\mathbf{T}_n \mathbf{x}_n = \mathbf{b}_n. \tag{6.2}$$

Clearly, a solution of the subsystem of order n is only computed if $\mathbf{T}_n$ is well conditioned, which is equivalent to n being a well-conditioned index which corresponds to a well-conditioned Padé form. Let us assume that $n = n_l$ and $n + k = n_{l+1}$ are well conditioned indices and denote the $(n+k) \times (n+k)$ principal submatrices $\mathbf{R}_{n+k}$ and $\mathbf{R}^{\uparrow}_{n+k}$ of $\mathbf{R}$ and $\mathbf{R}^{\uparrow}$, respectively, by

$$\mathbf{R}_{n+k} = \begin{bmatrix} \mathbf{R}_n & \mathbf{B} \\ \mathbf{0} & \mathbf{R}^{(l)} \end{bmatrix} \qquad \mathbf{R}^{\uparrow}_{n+k} = \begin{bmatrix} \mathbf{R}^{\uparrow}_n & \mathbf{B}^{\uparrow} \\ \mathbf{0} & \mathbf{R}^{\uparrow(l)} \end{bmatrix},$$

where l denotes the index of the block starting with the regular index $n = n_l$. The $(n+k)$th principal submatrix of the block diagonal matrix $\mathbf{D}$ is written as

$$\mathbf{D}_{n+k} = \begin{bmatrix} \mathbf{D}_n & \mathbf{0} \\ \mathbf{0} & \mathbf{D}^{(l)} \end{bmatrix},$$

so that $\mathbf{T}_n^{-1} = \mathbf{R}^{\uparrow}_n \mathbf{D}_n^{-1} \mathbf{R}_n^T$. If $\mathbf{x}_n$ is the solution of (6.2), and if we write

$$\mathbf{b}_{n+k} = \begin{bmatrix} \mathbf{b}_n \\ \mathbf{c} \end{bmatrix},$$

then a solution of

$$\mathbf{T}_{n+k} \mathbf{x}_{n+k} = \mathbf{b}_{n+k}$$

is given by

$$\begin{aligned}\mathbf{x}_{n+k} &= \begin{bmatrix} \mathbf{R}_n^{\uparrow} & \mathbf{B}^{\uparrow} \\ \mathbf{0} & \mathbf{R}^{\uparrow(l)} \end{bmatrix} \begin{bmatrix} \mathbf{D}_n^{-1} & \mathbf{0} \\ \mathbf{0} & (\mathbf{D}^{(l)})^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{R}_n^T & \mathbf{0} \\ \mathbf{B}^T & (\mathbf{R}^{(l)})^T \end{bmatrix} \begin{bmatrix} \mathbf{b}_n \\ \mathbf{c} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{x}_n \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{B}^{\uparrow} \\ \mathbf{R}^{\uparrow(l)} \end{bmatrix} \mathbf{y}, \end{aligned} \tag{6.3}$$

where

$$\mathbf{y} = (\mathbf{D}^{(l)})^{-1}(\mathbf{B}^T\mathbf{b}_n + (\mathbf{R}^{(l)})^T\mathbf{c}).$$

Therefore, $\mathbf{x}_{n+k}$ can be updated from $\mathbf{x}_n$ by using only those columns of $\mathbf{R}$ and $\mathbf{R}^{\uparrow}$ that belong to the current block. This update procedure costs k inner products for computing $\mathbf{B}^T\mathbf{b}_n$ plus k SAXPYs for the update, if we consider all computations of order at most $O(k^3)$ as negligible. The most important advantage of applying a procedure which uses only quantities available from the current block is that we do not have to store the complete factorization. Hence, the storage requirement is only $O(N)$.

Another equation for $\mathbf{y}$ can be obtained from the equality $\mathbf{D}^{(l)} = (\mathbf{R}^{(l)})^T\mathbf{L}^{\uparrow(l)}$, cf. (4.6). From the Toeplitz structure we have

$$\mathbf{T}_{n+k} = \begin{bmatrix} \mathbf{T}_n & \mathbf{U} \\ \mathbf{V} & \mathbf{T}_k \end{bmatrix},$$

which yields

$$\begin{aligned}\mathbf{y} &= (\mathbf{L}^{\uparrow(l)})^{-1}\left((\mathbf{R}^{(l)})^{-T}\mathbf{B}^T\mathbf{b}_n + \mathbf{c}\right) \\ &= (\mathbf{L}^{\uparrow(l)})^{-1}\left((\mathbf{R}^{(l)})^{-T}\begin{bmatrix} \mathbf{B}^T & (\mathbf{R}^{(l)})^T \end{bmatrix}\mathbf{T}_{n+k}\begin{bmatrix} \mathbf{x}_n \\ \mathbf{0} \end{bmatrix} - \mathbf{V}\mathbf{x}_n + \mathbf{c}\right) \\ &= (\mathbf{L}^{\uparrow(l)})^{-1}(\mathbf{c} - \mathbf{V}\mathbf{x}_n), \end{aligned} \tag{6.4}$$

because the first n columns of the product $\begin{bmatrix} \mathbf{B}^T & (\mathbf{R}^{(l)})^T \end{bmatrix}\mathbf{T}_{n+k}$ are zero since $\mathbf{R}^T\mathbf{T} = \mathbf{L}^T$ is block upper triangular by Theorem 4.2. For the symmetric positive definite case, (6.4) is well-known; see [13, Section 4.7.3]. In the look-ahead case, the formula was derived differently in [6] and [10, Section 6.1].

The inverse block LDU decomposition can also be generated as a byproduct of the look-ahead Schur algorithm, but, unless these columns are used anyway to control the look-ahead step size via (5.8) or (5.9), this increases the costs by about 50%, namely for applying the Levinson recursion (without computation of the inner products). Here, the key point is that the recurrence coefficients are identical for Levinson and for Schur-type algorithms.

6.2. Block LDU decomposition. The primary output of our look-ahead Schur algorithms are the matrices $\mathbf{L}$ and $\mathbf{L}^{\uparrow}$ of the LDU decomposition (4.5) of $\mathbf{T}$. Again, the full decomposition requires to compute underdetermined Padé forms. Due to the Toeplitz structure of the blocks of $\mathbf{D}$, this matrix also contains only quantities from $\mathbf{L}$ and $\mathbf{L}^{\uparrow}$. We refer to Section 4 for details. The inverse of $\mathbf{D}$ is not needed for solving (6.1).

When using the LDU decomposition (4.5) for computing the solution of the linear system (6.1) one can perform the forward elimination for the first linear system in

$$\mathbf{L}^{\uparrow}\mathbf{y} = \mathbf{b}, \quad \text{where } \mathbf{L}^T\mathbf{x} = \mathbf{D}\mathbf{y}, \tag{6.5}$$

with $O(N)$ storage. This involves solving linear systems with the diagonal blocks $\mathbf{L}^{\uparrow(l)}$, which are just scalars in generic steps. If we use the Schur-type algorithm of [22], where we have computed a regular pair from a previous regular pair and underdetermined Padé forms, then the QR or LU decomposition of $\mathbf{L}^{\uparrow(l)}$ is available anyway, because $\mathbf{L}^{\uparrow(l)}$ is just the coefficient matrix of one of the linear systems we have to solve for the recurrence coefficients of the new regular pair [22, Corollary 6]. But one still has to store the complete triangular part of $\mathbf{L}$ for solving the second system in (6.5) by backsubstitution. However, it is also possible to solve (6.1) with the Schur algorithm with $O(N)$ storage and without additionally running the Levinson recurrences, as we will see below.

6.3. Inversion formulas. Theorems 5.1 and 5.2 gave explicit expressions for the inverse of a Toeplitz matrix in terms of a pair of denominators of Padé forms. The look-ahead Levinson algorithm produces this last pair directly. If $N-1$ is a column-regular index, *i.e.*, if $\mathbf{T}_{N-1}$ and $\mathbf{T}_N$ are regular (or well conditioned), we can apply the Gohberg-Semencul formula (5.4) for $m = 0$ and $n = N-1$. If this is not the case, then by assumption, N is a regular index. We then have to compute the denominators of the regular pair of $(-1, N-1)$ and $(0, N)$ Padé forms in order to apply Heinig's inversion formula (5.3) for $m = 0$ and $n = N$.

In a Schur-type algorithm, the denominators are usually not computed, but as mentioned above, we could compute them additionally at the cost of some extra work.

Applying the inversion formula is an $O(N \log N)$ process, if we use fast Fourier transformation techniques.

6.4. Another approach complementing the Schur algorithm. Since $\mathbf{T} = \mathbf{L}^{\uparrow}(\mathbf{R}^{\uparrow})^{-1}$, we can write (6.1) as

$$\mathbf{L}^{\uparrow}\mathbf{y} = \mathbf{b}, \quad \text{where } \mathbf{x} = \mathbf{R}^{\uparrow}\mathbf{y}. \tag{6.6}$$

As we have explained above, the first, block lower triangular system for $\mathbf{y}$ can be solved by forward substitution without storing $\mathbf{L}^{\uparrow}$. Moreover, in the look-ahead Schur-type algorithm of [17], which computes regular pairs from regular pairs without using underdetermined Padé forms, it is sufficient to compute the numerators of right underdetermined Padé forms, *i.e.*, columns of $\mathbf{L}^{\uparrow}$. There is no need for computing left underdetermined Padé forms and residuals of right underdetermined Padé forms.

If we have additionally run the recurrences for the Levinson algorithm, then the solution $\mathbf{x}$ is obtained by just multiplying the temporary vector $\mathbf{y}$ by the upper triangular matrix $\mathbf{R}^{\uparrow}$. Since this involves only quantities from the current block, the storage requirement is only $O(N)$ again.

If $\mathbf{R}^{\uparrow}$ has not been computed, then this simple technique is not applicable. Using the matrix formulation (4.10) of the Levinson algorithm, we find $\mathbf{x}$ according to $\mathbf{R}^{\uparrow}\mathbf{y} = \mathbf{J}\mathbf{R}^{\uparrow(\mathrm{J})}(\mathbf{J}\mathbf{y})$ via

$$\mathbf{x} = \mathbf{J}\left[\begin{array}{c|c} \mathbf{R} & \mathbf{R}^{\uparrow(\mathrm{J})} \end{array}\right]\left[\frac{\mathbf{0}}{\mathbf{J}\mathbf{y}}\right] = \left[\begin{array}{c|c} \mathbf{J} & \mathbf{J} \end{array}\right]\prod_{n=0}^{N-2}\mathbf{C}_n\left[\frac{\mathbf{0}}{\mathbf{J}\mathbf{y}}\right]. \tag{6.7}$$

Again, the storage requirement is only $O(N)$. We have to store all the coefficients of the small two-point Padé forms we used for updating one column-regular or regular pair from the previous one. Moreover, we need the recurrence coefficients for left and right underdetermined Padé forms.

For the Bareiss algorithm applied to a symmetric positive definite $\mathbf{T}$, Delosme and Ipsen [8, 24] derived a formula that is similar to (6.7). They made use of the fact that in this case the matrices $\mathbf{C}_n$ are hyperbolic rotations, and thus the inverse of their product is equal to a diagonally scaled transpose of the product. Thus, both the forward elimination and the backsubstitution can be performed by applying a product of hyperbolic rotations.

7. Operation counts: Schur versus Levinson algorithms. The main difference between a Levinson and a Schur algorithm is that the Levinson algorithm spends about half of the operations on computing inner products but the Schur algorithm requires only SAXPYs. This makes the Schur algorithm highly attractive on parallel computers, in particular on those, where inner products represent a bottleneck of the computation due to the global communication involved.

To summarize the various options for solving the linear systems, the Levinson and the Schur algorithm both require $2N^2 + O(N)$ flops to compute the LDU or inverse LDU factorization of a Toeplitz matrix, because the overhead of our look-ahead algorithms is only of order N by the assumptions made at the end of Section 2. They are satisfied when $\mathbf{T}$ has only a few ill-conditioned principal submatrices, a situation which normally happens in practice.

For the Schur algorithm, one still has the option of computing the LDU decomposition, *i.e.*, additionally running the Levinson recurrences. This costs another $N^2+O(N)$ flops, so that computing the LDU and the inverse LDU decomposition simultaneously does not double the cost, but can be done in $3N^2 + O(N)$ flops.

The cheapest way of solving the linear system with the Levinson algorithm is to apply a suitable inversion formula by exploiting fast Fourier transformation techniques. This is an $O(N \log N)$ process, so that in total, we need $2N^2 + O(N \log N)$ flops to get the solution. Updating the solution from the inverse LDU decomposition via (6.3) costs $N^2+O(N)$ flops, hence we can get the solution with this technique in $3N^2+O(N)$ flops.

For the Schur algorithm, the cheapest solution method is using the LDU decomposition (6.5). The solution process costs again $N^2 + O(N)$ flops, hence the overall process can be done in $3N^2 + O(N)$ flops. However, using the LDU decomposition is only possible if we can store one complete block lower triangular factor of this decomposition. If N is large, this becomes prohibitive. One then has the option to run the Levinson algorithm additionally and use (6.6). The price for this procedure is $4N^2 + O(N)$ flops, since the solution costs again $N^2 + O(N)$ flops. Another option is again to apply an inversion formula, which costs $3N^2 + O(N \log N)$ flops. A cheaper variant stems from the matrix formulation (6.7), where one uses the second equation in (6.6) only implicitly by essentially running the Levinson algorithm backwards. Since the matrices $\mathbf{C}_n$ are of size $2N \times 2N$, computing $\mathbf{x}$ from (6.7) costs $N^2 + O(N)$ flops. Here, $\mathbf{y}$ is taken from the first equation in (6.6) at the cost of $0.5N^2 + O(N)$ flops so that this variant costs $3.5N^2 + O(N)$ flops.

8. Numerical examples. We have implemented all our algorithms in MATLAB and in C, but present only the results from the C implementation. The C code is

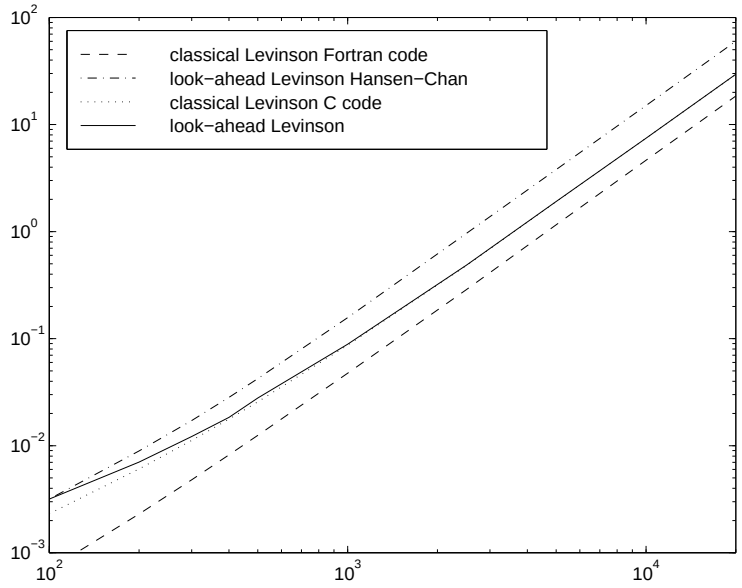

FIG. 8.1. *Execution times in seconds versus the dimension N of the Toeplitz system. Compared are the C implementation of the classical and the look-ahead Levinson algorithm with the* FORTRAN *implementation of the classical and the look-ahead Levinson algorithm from Hansen and Chan.*

available from the author. It uses the Basic Linear Algebra Subroutines (BLAS) and LAPACK routines [2] for computing the smallest singular value and for solving the linear systems which arise in look-ahead steps. For the examples we present here, the linear systems have been solved by Gaussian elimination with partial pivoting. Optionally, the code can solve the linear systems by QR factorization. All our numerical examples were done on an SGI Power Indigo 2 with machine precision $1.11 \cdot 10^{-16}$.

We compare our algorithms with the FORTRAN implementation of the look-ahead Levinson algorithm [6], written by Hansen and Chan [18], which is – to the best of the author's knowledge – the only available implementation of a look-ahead algorithm on the network; see also [11]. Unfortunately, the implementation of Freund and Zha's look-ahead Levinson algorithm announced in [10, Section 9] was not available at the time of this writing. Hence we were not able to add comparisons with this code. In [11], yet another high performance library is announced, which will be developed for distributed memory machines. This library will mainly contain algorithms based on transforming the original Toeplitz matrix to some other structured matrix where pivoting can be applied. It was also not finished yet.

Let us first compare the execution times of the different algorithms. Figure 8.1 shows the execution times for a FORTRAN implementation of the classical Levinson algorithm [3] available from NETLIB, the FORTRAN implementation of the look-ahead Levinson algorithm from Hansen and Chan [18], our C implementation of the classical Levinson algorithm and of our look-ahead Levinson algorithm [22], which uses inner formally biorthogonal polynomials for computing regular pairs. The execution times have been computed for random Toeplitz matrices of dimensions $N = 100, 200, 300, 400, 500, 1000, 2500, 5000, 10000, 20000$ with an ill-conditioned principal submatrix of dimension 50. The maximal size of a look-ahead step was set to 6.

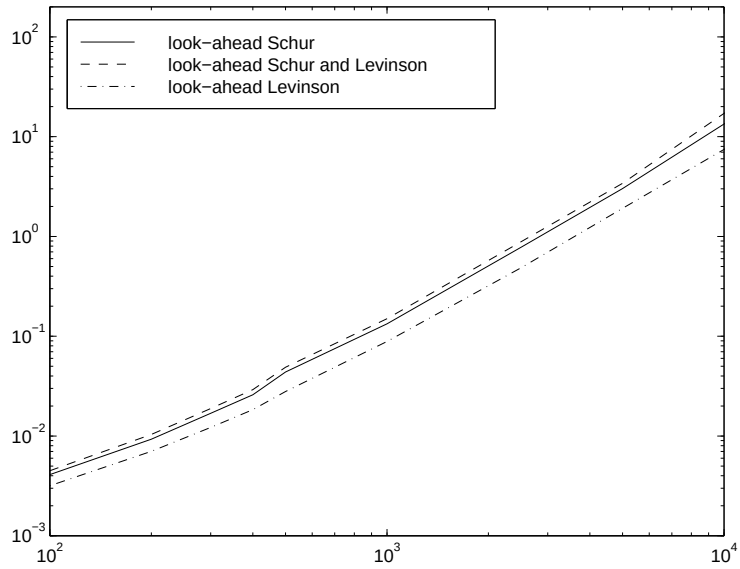

FIG. 8.2. *Execution times in seconds versus the dimension N of the Toeplitz system. Compared are the C implementation of the look-ahead Schur algorithms, the look-ahead Levinson algorithm, and the combined look-ahead Schur and Levinson algorithm.*

The plot is in a double logarithmic scale, where the x-axis shows the dimensions and the y-axis shows the execution times in seconds.

The figure shows that the C implementation is slower than the FORTRAN code from the Argonne package [3]. The main reason is that our C implementation uses the Basic Linear Algebra Subroutines (BLAS) and the FORTRAN code does not. For the Toeplitz solvers, this turns out to be more expensive on our machine because most of the copy operations for long vectors could be done simultaneously with computations if we implemented them by hand. We nevertheless decided to use the BLAS and hope that they are available in an optimized form on most of the machines. Moreover we would like to stress that we have *not* optimized our code with respect to copy operations but only with respect to SAXPY's and inner products. The interesting observation from this figure is that the execution times of the FORTRAN implementation by Hansen and Chan are considerably larger than those of the classical Levinson algorithm although Hansen and Chan also do not use the BLAS' copy function. For large N, the ratio of execution times is about $3.4 : 1$. However, for the C implementation and dimensions larger than about 1000 one cannot even distinguish the two lines for the classical and the look-ahead Levinson algorithm. This shows that the overhead due to look-ahead steps is negligible then. Both classical algorithms gave wrong results because of the ill-conditioned principal submatrix.

In Figure 8.2 we compare execution times of our C implementation of look-ahead Levinson and Schur algorithms from [22] for the same examples as before. Since the dimensions become as large as 20000, the only option of solving the linear systems from the output of a Schur algorithm is via (6.6). As we discussed in Section 7, there are two ways of computing $\mathbf{x} = \mathbf{R}^{\uparrow}\mathbf{y}$, namely running the Levinson recurrences additionally or computing $\mathbf{x}$ from (6.7) by essentially running the Levinson algorithm

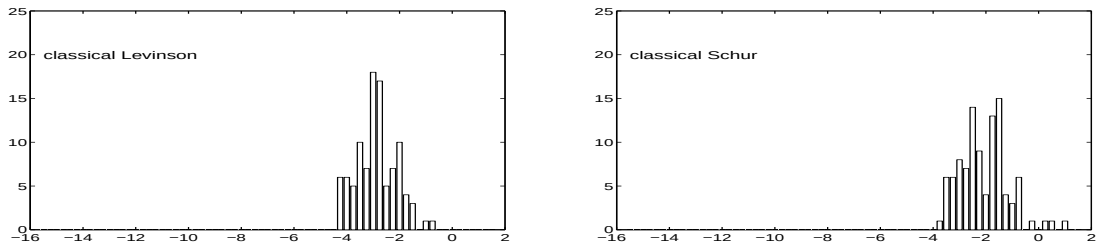

FIG. 8.3. *Histograms of relative errors of the C implementation of the classical Levinson and Schur algorithm.*

backwards. The first variant costs $4N^2 + O(N)$ flops, the second $3.5N^2 + O(N)$, and in fact, 4/3.5 is about the factor between the two curves corresponding to the Schur algorithms in Figure 8.2. The reason that the look-ahead Levinson algorithm is considerably faster than the Schur algorithms is not only that it takes $3N^2 + O(N)$ flops but also that – as we mentioned before – we have not attempted to optimize the number of copy operations in both algorithms. It turns out that our implementation of the Schur algorithm requires about twice as many copy statements, which explains the difference in execution times. We have not plotted the times for the classical Schur algorithm, since the curves are again not distinguishable from those of its look-ahead counterparts for large dimensions N.

For comparing the accuracy of the algorithms we created a set of 100 random test examples of dimension 500 with a very ill-conditioned principal submatrix of dimension 50. We used this test set for all examples. The right-hand sides of the linear systems were computed such that a given random vector was the exact solution. This allowed us to compare the different algorithms by computing the relative errors

$$\frac{\|\mathbf{x}_{exact} - \mathbf{x}_{comput.}\|}{\|\mathbf{x}_{exact}\|}.$$

For our algorithms, we used a tolerance of 0.1 for checking if a regular Padé form is also column-regular, *i.e.*, we enforced $|\pi_{0,n}| > 0.1$. The tolerance for the smallest singular values in (5.1) or (5.2) was 0.3. We present the results in form of histograms, where the x-axis shows the logarithm of the achieved relative error and where the y-axis displays the number of examples which achieved an accuracy in a given interval of length 0.25 in a logarithmic scale. Some of the Toeplitz matrices were themselves ill-conditioned, so that we could not expect to get full accuracy for all of the 100 test examples. In fact, our algorithm produced warnings that the results may be inaccurate for two of the 100 examples.

Figure 8.3 shows the performance of different algorithms when the solution is computed by one of the LDU decompositions from our C implementation of the classical Levinson and Schur algorithm. Clearly, the accuracy is poor because of the very ill-conditioned principal submatrix of dimension 50.

Figure 8.4 shows the achieved accuracy for the FORTRAN implementations of the classical Levinson algorithm [3] and the look-ahead Levinson algorithm [18], where we have set the maximal block size to 6.

In Figure 8.5 we compare our look-ahead Levinson algorithms. For solving the linear systems we used the update formula (6.3). Recall that we denote the algorithm

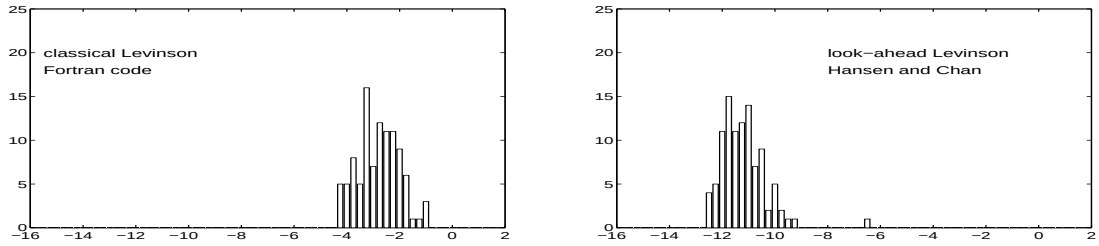

FIG. 8.4. *Histograms of relative errors of* FORTRAN *implementation of the classical Levinson and the look-ahead Levinson algorithm of Hansen and Chan.*

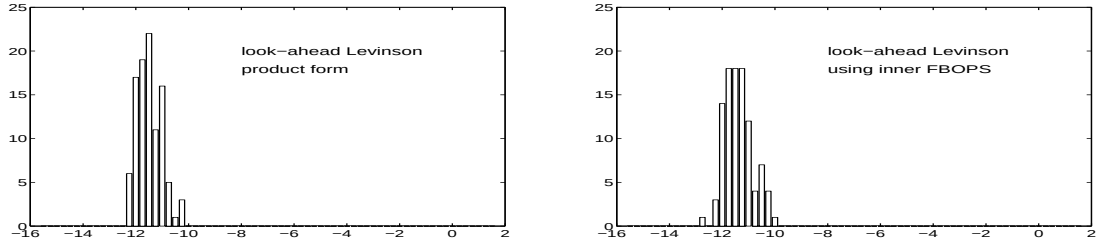

FIG. 8.5. *Histograms of relative errors of our C implementation of the look-ahead Levinson algorithms.*

described in [17] as the product form of our look-ahead Levinson algorithm, because it is based on products of Padé forms with low-order two-point Padé forms. It was shown in Table 4.1 that the algorithm [22] which uses inner formally biorthogonal polynomials has less overhead when we compute the complete LDU decomposition, as we did in all our experiments here. This most efficient look-ahead algorithm gave slightly better results than the Hansen and Chan algorithm. However, as we have seen before, it has much less overhead and shorter execution times.

Figure 8.6 shows the analogous results for our Schur algorithms in product form and those using underdetermined Padé forms. For solving the linear systems we used the LDU decomposition (6.5). Since the denominators of the Padé forms have not been computed the algorithms required $O(N^2)$ storage. The results are very similar to those of our look-ahead Levinson algorithms.

Figure 8.7 shows the relative errors for our two look-ahead Schur algorithms when the solutions of the linear systems are computed via (6.6) without computing $\mathbf{R}^{\uparrow}$ explicitly from the Levinson recurrences but by computing $\mathbf{x}$ via (6.7). These algorithms require only $O(N)$ storage. Finally, Figure 8.8 shows the accuracy for the Schur algorithms when additionally the inverse LDU decomposition is computed, *i.e.*, when we have run the Levinson recurrences but with the recurrence coefficients taken from the Schur algorithm and not by evaluating inner products. The solutions were obtained from (6.6).

The last three Figures 8.6–8.8 containing results from the two variants of look-ahead Schur algorithms combined with different methods for solving the Toeplitz systems look very similar. This leads to the observation that the accuracy is determined mainly by the Schur algorithms itself, the influence of how to compute $\mathbf{x}$ from $\mathbf{y}$ in (6.5) or (6.6) appears to be minor.

So far we used block (inverse) LDU decompositions for solving the Toeplitz sys-

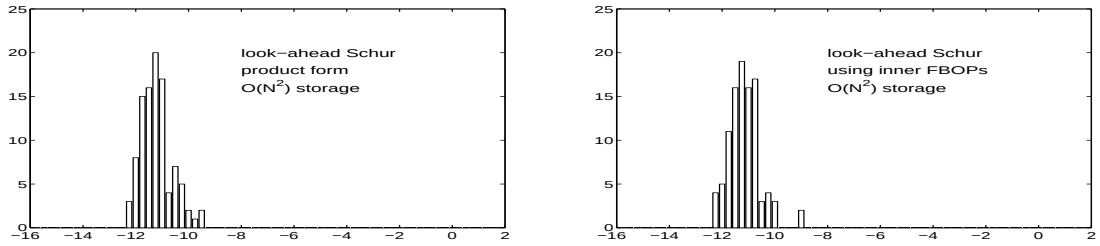

FIG. 8.6. *Histograms of relative errors of our C implementation of the look-ahead Schur algorithms when the solutions are obtained from* (6.5).

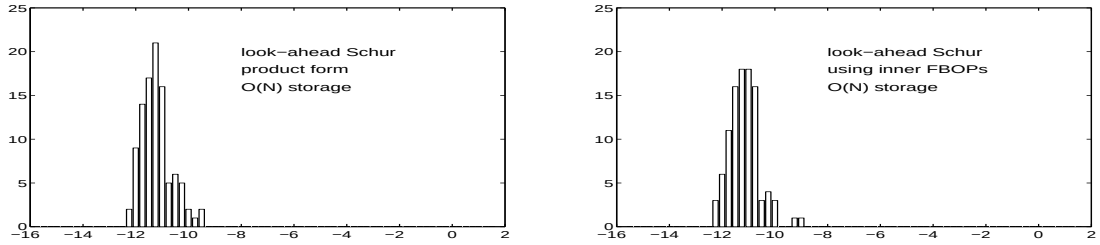

FIG. 8.7. *Histograms of relative errors of our C implementation of the look-ahead Schur algorithms when the solutions are obtained from* (6.6) *with* $\mathbf{x}$ *from* (6.7).

tems. Next we apply one of the inversion formulas from Theorems 5.1 and 5.2. The results are presented in Figure 8.9. As we mentioned before, this solution technique is only applicable when we have run the Levinson algorithm, either alone or combined with the Schur algorithm. Now we can see that applying an inversion formula or using block LDU decompositions makes a difference, although this difference is not significant.

For these algorithms, one can also compute the solution from an LDU decomposition and then apply an inversion formula in order to improve the accuracy by one step of iterative refinement. The achieved accuracy is shown in Figure 8.10, where the intervals for the histograms have now length 0.1 so that the scale of the y-axis remains the same as in the other examples. The accuracy is pretty close to machine precision. Clearly, one could do more than one refinement step to further improve the accuracy.

To summarize, we recommend to use the look-ahead Levinson algorithm [22] which uses inner formally biorthogonal polynomials, optionally combined with one step of iterative refinement, on serial computers. The solution should be computed by using the inverse block LDU decomposition via the update formula (6.3). On parallel machines, Schur algorithms are advantageous. Here the recommendation is again to use the variant of the look-ahead Schur algorithm proposed in [22] since it has less overhead in look-ahead steps than any other variant. If memory facilities allow to store one complete triangular matrix of dimension N, then it is most efficient to compute the solution by using the block LDU decomposition (6.5). Otherwise, using (6.6) and (6.7) is the method of choice for solving the Toeplitz system if only one system has to be solved and no iterative refinement is desired. If one has a Toeplitz system with multiple right-hand sides or wants to apply iterative refinement, then one should additionally run the Levinson recurrences and solve the linear systems via an inversion formula.

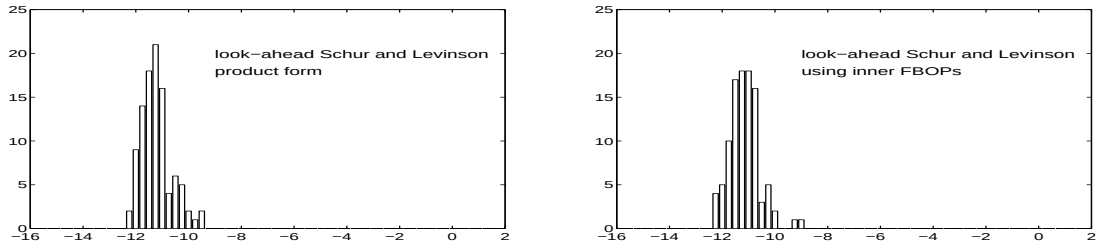

FIG. 8.8. *Histograms of relative errors of our C implementation of the look-ahead Schur algorithms both combined with the corresponding recurrences of the Levinson algorithms.*

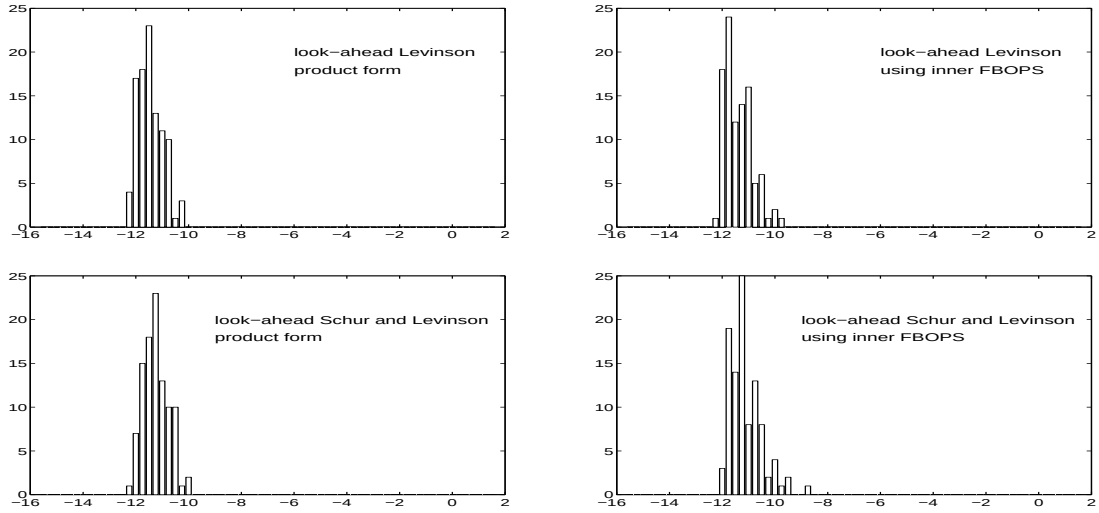

FIG. 8.9. *Histograms of relative errors of our look-ahead algorithms when the solutions are obtained by applying an inversion formula.*

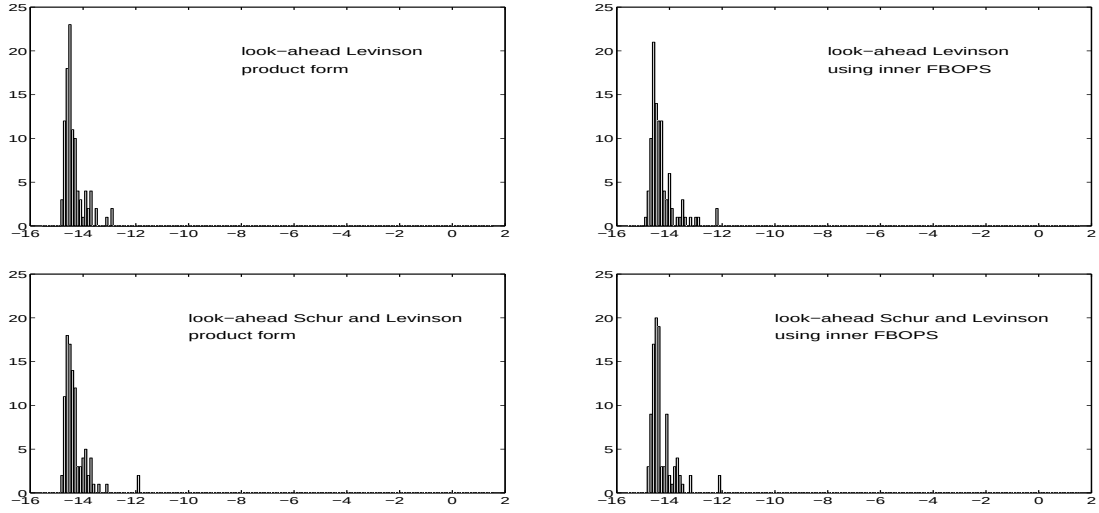

FIG. 8.10. *Histograms of relative errors when the solutions are obtained from an (inverse) LDU decomposition and are improved by one step of iterative refinement.*

Acknowledgement. The author would like to thank Mathias Fröhlich for his tremendous help with implementing the look-ahead Toeplitz solvers in C.

REFERENCES

[1] G. S. Ammar and W. B. Gragg, *The implementation and use of the generalized Schur algorithm*, in Computational and Combinatorial Methods in System Theory, C. Byrnes and A. Lindquist, eds., Elsevier Science Publishers B.V., 1986, pp. 265–279.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, S. Ostrouchov, and D. Sorensen, *LAPACK User's Guide*, SIAM, 1995.

[3] O. B. Arushanian, M. K. Samarin, V. V. Voevoedin, E. E. Tyrtyshnikov, B. S. Garbow, J. M. Boyle, W. R. Coweli, and K. W. Dritz, *The Toeplitz Package Users' Guide*, Technical Report, Argonne National Laboratory, 1983, 1983.

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

[5] A. Bultheel, *Laurent Series and their Padé Approximations*, Birkhäuser, Basel/Boston, 1987.

[6] T. F. Chan and P. C. Hansen, *A look-ahead Levinson algorithm for general Toeplitz systems*, IEEE Trans. Signal Processing, 40 (1992), pp. 1079–1090. .

[7] F. de Hoog, *A new algorithm for solving Toeplitz systems of equations*, Linear Algebra Appl., 88/89 (1987), pp. 123–138.

[8] J. Delosme and I. C. F. Ipsen, *Parallel solution of symmetric positive definite systems with hyperbolic rotations*, Linear Algebra Appl., 77 (1986), pp. 75–111.

[9] R. W. Freund, *A look-ahead Bareiss algorithm for general Toeplitz matrices*, Numer. Math., 68 (1994), pp. 35–69.

[10] R. W. Freund and H. Zha, *Formally biorthogonal polynomials and a look-ahead Levinson algorithm for general Toeplitz systems*, Linear Algebra Appl., 188/189 (1993), pp. 255–304.

[11] K. A. Gallivan, S. Thirumalai, P. Van Dooren, and V. Vermaut, *High performance algorithms for Toeplitz and block Toeplitz matrices*, Linear Algebra Appl., 241–243 (1996), pp. 343–388.

[12] I. Gohberg and A. Semencul, *On the inversion of finite Toeplitz matrices and their continuous analogs (in Russian)*, Mat. Issled., 2 (1972), pp. 201–233.

[13] G. H. Golub and C. F. van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, second ed., 1989.

[14] M. H. Gutknecht, *Stable row recurrences in the Padé table and generically superfast lookahead solvers for non-Hermitian Toeplitz systems*, Linear Algebra Appl., 188/189 (1993), pp. 351–421.

[15] M. H. Gutknecht and M. Hochbruck, *Look-ahead Levinson- and Schur-type recurrences in the Padé table*, Electronic Trans. Numer. Anal., 2 (1994), pp. 104–129.

[16] ———, *Look-ahead Levinson and Schur algorithms for non-Hermitian Toeplitz systems*, Numer. Math., 70 (1995), pp. 181–227.

[17] ———, *Optimized look-ahead recurrences for adjacent rows in the Padé table*, BIT, 36 (1996), pp. 264–286.

[18] P. C. Hansen and T. F. Chan, *FORTRAN subroutines for general Toeplitz systems*, ACM Trans. Math. Softw., 18 (1992), pp. 256–273. Corrigendum, see [19].

[19] ———, *Corrigendum: Algorithm 729. FORTRAN subroutines for general Toeplitz systems*, ACM Trans. Math. Softw., 20 (1994), p. 160.

[20] G. Heinig and K. Rost, *Algebraic Methods for Toeplitz-like Matrices and Operators*, Akademie-Verlag, Berlin, and Birkhäuser, Basel/Stuttgart, 1984.

[21] M. Hochbruck, *The Padé table and its relation to certain numerical algorithms*. Habilitationsschrift, Mathematische Fakultät, Universität Tübingen, Germany, 1996.

[22] ———, *Further optimized look-ahead recurrences for adjacent rows in the Padé table and Toeplitz matrix factorizations*, J. Comput. Appl. Math., 86 (1997), pp. 219–236.

[23] T. Huckle, *Computation of Gohberg-Semencul formulas for a Toeplitz matrix*, Tech. Rep. SCCM-93-14, Computer Science Department, Stanford University, November 1993. Submitted to Linear Algebra Appl.

[24] I. Ipsen, *Some remarks on the generalised Bareiss and Levinson algorithms*, in Numerical Linear Algebra, Digital Signal Processing and Parallel Algorithms, G. H. Golub and P. van Dooren, eds., Springer-Verlag, Berlin, 1990, pp. 189–214.

[25] T. Kailath and A. H. Sayed, *Displacement structure: theory and applications*, SIAM Rev., 37 (1995), pp. 297–386. .

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

[27] I. Schur, *Ueber Potenzreihen, die im Innern des Einheitskreises beschränkt sind, I*, Crelle's J. (J. Reine Angew. Math.), 147 (1917), pp. 205–232.

[28] W. F. Trench, *An algorithm for the inversion of finite Toeplitz matrices*, J. Soc. Indust. Appl. Math., 12 (1964), pp. 515–522.

[29] M. Van Barel and A. Bultheel, *A look-ahead algorithm for the solution of block Toeplitz systems*, Linear Algebra Appl., (1996). Submitted.