ELSEVIER Linear Algebra and its Applications 278 (1998) 11–36

LINEAR ALGEBRA AND ITS APPLICATIONS

Transformation techniques for Toeplitz and Toeplitz-plus-Hankel matrices II. Algorithms

Georg Heinig [a,*,1], Adam Bojanczyk [b]

[a] *Kuwait University, Department of Mathematics, P.O Box 5969, Safat 13060, Kuwait*
[b] *Cornell University, School of Electrical Engineering, Ithaca, NY 14853-3801, USA*

Received 9 August 1996; accepted 12 October 1997

Submitted by P. Van Dooren

Abstract

This paper is a continuation of [G. Heinig, A. Bojanczyk, Linear Algebra Appl. 254 (1997) 193–226] where transformations mapping Toeplitz and Toeplitz-plus-Hankel matrices into generalized Cauchy matrices were studied. In the present paper fast algorithms for LU-factorization and inversion of generalized Cauchy matrices are discussed. It is shown that the combination of transformation pivoting techniques leads to algorithms for indefinite Toeplitz and Toeplitz-plus-Hankel matrices that are more stable than the classical ones. Special attention is paid to the symmetric and hermitian cases.

Keywords: Toeplitz matrix; Cauchy matrix; Fast algorithm

1. Introduction

It is well known that the classical fast algorithms for the solutions of linear systems with a Toeplitz or Toeplitz-plus-Hankel coefficient matrix are unstable if the matrix has ill-conditioned principal submatrices. The first idea to overcome this difficulty was to apply look-ahead strategies similar to look-ahead Lanczos methods (see, for example, [6,11]). One drawback of the look-ahead

* Corresponding author.

[1] Supported by research project SM-100 of Kuwait University.

0024-3795/98/$19.00
PII: S0024-3795(97)10043-X

methods are that they are not always fast. A simple example for such a situation is a matrix of the form

$$T = \begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix} + E,$$

where I is the identity and E is a Toeplitz matrix with a small norm. Furthermore, a reliable step size estimator is rather expensive.

As an alternative, during the last years transformation based algorithms have been proposed in [13,9,21] and other papers. These algorithms are always fast and provide good numerical results if they are combined with pivoting. In [16] and the present paper several transformation based algorithms are discussed and compared. All algorithm are based on the transformation into generalized Cauchy matrices. The paper [16] [2] is dedicated to the transformation, whereas the present paper describes mainly the solution of the resulting systems with a generalized Cauchy coefficient matrix. Note that there are other possibilities for transformation. The transformation into Vandermonde-like matrices (after extension) is discussed in [14,15].

To begin with let us recall the definition of a (generalized) Cauchy matrix. Let $c = (c_i)_1^n$ and $d = (d_j)_1^n$ be fixed n-tuples of complex numbers and $A = [a_{ij}]_1^n$ a given matrix. Then the *Cauchy rank* of A with respect to c and d is, by definition, the rank r of the matrix

$$\nabla_{c,d}(A) = [(c_i - d_j)a_{ij}]_1^n.$$

If r is small compared with the order of the matrix A, then A will be called (generalized) *Cauchy matrix*. Cauchy matrices in the classical sense are matrices with $(c_i - d_j)a_{ij} = 1$ for all i and j and have, therefore, Cauchy rank one.

Cauchy matrices appear directly in many applications, for example in rational interpolation (see [12,2]). The motivation for the present paper is, however, the observation that they appear indirectly as the result of transformations of the form $A \to \mathscr{C}_1 A \mathscr{C}_2$ of Toeplitz and Toeplitz-plus-Hankel matrices and their generalizations where $\mathscr{C}_1$, $\mathscr{C}_2$ are related to discrete Fourier or real trigonometric transformations. This was first observed in [12,13] for complex Fourier transformations and in [9] and [21] for some real trigonometric transformations. In our paper [16] we presented an account of different transformations transforming Toeplitz and Toeplitz-plus-Hankel matrices into Cauchy matrices. All these transformations can be carried out in an efficient and stable way. [3]

[2] We refer to this paper as Part I.

[3] Actually the efficiency of the FFT depends on n. The common algorithms assume that n is a product of small primes. In [14] an approach is presented with more freedom in the choice of the length of the FFT.

One of the pleasant properties of Cauchy matrices is that permutations of columns and rows preserve the Cauchy structure, which fails to be true for Toeplitz and Toeplitz-plus-Hankel matrices. This property of Cauchy matrices provides additional freedom in how matrix decomposition algorithms based on Gaussian elimination can be applied. It turns out that instead of dealing with Toeplitz and Toeplitz-plus-Hankel matrices, it is worthwhile, from the numerical point of view, to deal with the corresponding Cauchy matrices. This is the subject of this paper.

Fast inversion algorithms for Cauchy matrices were proposed for the first time in [17]. The algorithms there are of Levinson type and are based on the "bordering method". In [7,8] algorithms of Schur type for matrices with "recursive structure" were presented that also include fast algorithms for Cauchy matrices. The Schur type algorithms are in particular convenient for parallel processing. Concerning more references in this development we refer to the survey paper [20]. In [12] both Levinson and Schur type (called type-I and type-II, respectively) algorithms were derived from the interpolation interpretation of Cauchy systems of equations. Based on the observation that the Cauchy rank is inherited by Schur complementation it was observed in [10,19] independently that Gaussian elimination can be carried out in an efficient way considering only the generators, which are the vectors with the help of which the matrix $\nabla_{c,d}(A)$ is "generated". This result is implicitly also contained in [12] (compare Theorem 5.1 and Proposition 5.1). The corresponding Gauss–Schur algorithms was presented in [12,9]. The difference between the Algorithm GKO in [9] and the type-II algorithm in [12] is that the type-II algorithm in [12] computes the LU-factorization of the Cauchy matrix and its inverse in parallel, while the Algorithm GKO in [9] computes only the LU-factorization of the original matrix (by the same formulas as the type-II algorithm) and uses backward substitution for the solution of the corresponding systems of equations.

Gaussian elimination can be combined with partial pivoting which leads, as numerical experiments show, in practice to fairly stable algorithms. However, as it was observed in [23], if one applies the fast algorithm then it can happen, at least theoretically, that there is a growth in the generators during the computation. According to [22] in all examples in [23] the growth can be avoided by a proper choice of the generators. Moreover, in [21] an orthogonalization technique for the generators combined with some approximately complete, fast pivoting is proposed, and it is shown that by applying this technique the growth becomes moderate.

In the first part of this paper attention was paid to transformations preserving properties like realness and symmetry. Once the Cauchy matrix is real and symmetric, the question is how to carry out pivoting while preserving

the structure of the matrix. This question is one of the main topics of this paper. [4]

Let us sketch the contents of the paper. In Section 2 we consider algorithms for general Cauchy matrices. First we present the Gauss–Schur type algorithm *LU-CAUCHY* for the LU-factorization of a Cauchy matrix in the form how it was designed in [9] and in other form derived in [12]. We also present a type-I (Levinson-type) algorithm *UL-INVCAUCHY* for the UDL-factorization of the inverse matrix with a pure matrix derivation (and correcting a mistake in [12]).

The magnitude of the residual error produced by the algorithm *UL-INVCAUCHY* indicates that the algorithm may not be backward stable. However, the relative error in the solution vector x is of the same order of magnitude as that produced by Algorithm *LU-CAUCHY* and proportional to the condition number of the matrix A. This situation is similar to that in the case of Levinson and Schur methods applied to a linear system with positive definite Toeplitz coefficient matrix. For more discussion on stability of Levinson and Schur-type algorithms we refer to [1].

In Section 3 we discuss first the specifics of symmetric and hermitian Cauchy matrices and after that we adopt the Bunch–Kaufman–Parlett method for symmetric pivoting to the case of symmetric and hermitian Cauchy matrices. In Section 4 we discuss the application of this to matrices appearing as the result of transformations of Toeplitz and Toeplitz-plus-Hankel presented in [16]. This concerns all types of complex transformations discussed in [16] and the sine-I transformation as an example of a real trigonometric transformation. The implementation for the other real transformations would be similar, but it is slightly more complicated.

The results of [21] suggest that the mixed sine-I/cosine-I transformation should not be used for transformation since a substantial growth of the elements during the computation could be expected. A better mixed transformation is the combination of cosine-II and sine-IV which is the transformation proposed in [21].

In Section 4 we present the results of numerical experiments. Experiments were carried out for complex hermitian Toeplitz-plus-Hankel and for real symmetric Toeplitz systems of linear equations. The results are compared with those obtained with the fast look-ahead solver of Chan/Hansen [6]. The comparison indicates that the transformation based algorithms provide more accurate results in the case of several subsequent ill-conditioned principal submatrices.

[4] After the the first draft of the paper was completed we learned that Kailath and Olshevsky developed independently similar ideas in connection with symmetric pivoting for Cauchy matrices (see [18]).

2. Fast factorization algorithms for general Cauchy matrices

In this section we consider general Cauchy matrices $C = [a_{ij}]_1^n$. If the Cauchy rank of C with respect to $c = (c_i)_1^n$ and $d = (d_j)_1^n$ equals r then there exist vectors z_i and $y_j \in \mathbb{C}^r$ satisfying

$$(c_i - d_j)a_{ij} = z_i^{\mathrm{T}} y_j \tag{2.1}$$

The relation (2.1) can be written as a displacement (Sylvester) equation

$$D(c)C - CD(d) = ZY^{\mathrm{T}}, \tag{2.2}$$

where $Z = \mathrm{col}(z_i^{\mathrm{T}})_1^n$ and $Y = \mathrm{col}(y_i^{\mathrm{T}})_1^n$, $D(c) = \mathrm{diag}(c_i)_1^n$, $D(d) = \mathrm{diag}(d_j)_1^n$. Following [20] we call the matrices Y and Z generators of C.

We restrict ourselves to the following two cases:

Case A: $c_i \neq d_j$ for all i and j.

Case B: $c_i = d_i$ for all i and the c_i are pairwise different.

This covers all cases appearing in part I. Clearly, in the case A the generators define C uniquely; in the case B the generators define C only for the off-diagonal entries.

It is obvious that the inverse of a Cauchy matrix is a Cauchy matrix again and has the same Cauchy rank. In fact, (2.2) implies

$$D(d)C^{-1} - C^{-1}D(c) = -XW^{\mathrm{T}}.$$

where X and W are the solutions of the "fundamental" equations $CX = Z$ and $W^{\mathrm{T}}C = Y^{\mathrm{T}}$. In case A the solutions X and W define the matrix C^{-1} completely. In case B one has still to find the diagonal of C^{-1}. This can be done by solving also the equation $Cu = \mathbf{1}$ with $\mathbf{1} = [1\ 1\ \ldots\ 1]^{\mathrm{T}}$. Then the diagonal of C^{-1} is given by

$$\mathrm{diag}\, C^{-1} = u - \left(\sum_{j \neq i} \frac{x_i^{\mathrm{T}} w_j}{c_i - c_j} \right)_{i=1}^{n}.$$

A basic observation which leads to the construction of fast inversion and factorization algorithms is that the Cauchy structure is inherited under Schur complementation. To show this one has to apply the "magic wand" Schur complement formula for a partitioned matrix

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

with nonsingular A_{11} which reads as follows

$$A = \begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & A_{11}^{\#} \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix}, \tag{2.3}$$

where $A_{11}^{\#}$ denotes the Schur complement $A_{11}^{\#} = A_{22} - A_{21}A_{11}^{-1}A_{12}$. If A_{11} is 1×1 then (2.3) just illustrates one step of Gaussian elimination. Repeating these steps one gets an LU-factorization of the matrix A. A block LU-factorization of A is obtained in an analogous way by choosing pivots A_{11} as square submatrices of size $k \geqslant 2$.

Lemma 1 (cf. [10,19]). *Let*

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

be a partitioned Cauchy matrix corresponding to (c,d), $c = (c_1, c_2)$, $d = (d_1, d_2)$ *is the partition of the tuples c and d according to that of C, and let the generators of C be given by*

$$Z = \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix}, \quad Y = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}.$$

Then the Schur complement $C_{11}^{\#}$ *is Cauchy matrix corresponding to* (c_2, d_2) *with generators*

$$Z^{(1)} = Z_2 - C_{21}C_{11}^{-1}Z_1, \qquad Y^{(1)^{\mathrm{T}}} = Y_2^{\mathrm{T}} - Y_1^{\mathrm{T}}C_{11}^{-1}C_{12}.$$

This lemma leads to the following algorithm of LU-factorization of strongly nonsingular Cauchy matrices which, for the case A, is a part of the type-II algorithm described in [12] and coincides, also for case A, with the GKO-algorithm in [9].

Algorithm LU-CAUCHY

1. Initialization: $Z^{(0)} = Z = \begin{bmatrix} z_1^{(0)} \\ Z_2^{(0)} \end{bmatrix}$, $Y^{(0)} = Y = \begin{bmatrix} y_1^{(0)} \\ Y_2^{(0)} \end{bmatrix}$.

In case B: $d^{(0)} = \mathrm{diag}C$.

2. For $k = 0, \ldots, n-1$,
(a) Compute $a^{(k)}$, $l^{(k)} = (l_i^{(k)})_1^{n-k}$, $u^{(k)} = (u_j^{(k)})_1^{n-k}$ by

$$a^{(k)} = \frac{z_1^{(k)\mathrm{T}} y_1^{(k)}}{c_{1+k} - d_{1+k}} \quad \text{(case A)}, \qquad a^{(k)} = d_{11}^{(k)} \quad \text{(case B)}$$

$$l_i^{(k)} = \frac{1}{a^{(k)}} \frac{z_i^{(k)\mathrm{T}} y_1^{(k)}}{c_{i+k} - d_{1+k}}, \qquad u_j^{(k)} = \frac{1}{a^{(k)}} \frac{z_1^{(k)\mathrm{T}} y_j^{(k)}}{c_{1+k} - d_{j+k}}.$$

(b) Compute

$$Z^{(k+1)} = \begin{bmatrix} z_1^{(k+1)\mathrm{T}} \\ Z_2^{(k+1)} \end{bmatrix} = Z_2^{(k)} - l^{(k)} z_1^{(k)\mathrm{T}},$$

$$Y^{(k+1)} = \begin{bmatrix} y_1^{(k+1)\mathrm{T}} \\ Y_2^{(k+1)} \end{bmatrix} = Y_2^{(k)} - u^{(k)} y_1^{(k)\mathrm{T}}.$$

(c) In case B compute $d^{(k+1)} = (d_i^{(k+1)})_1^{n-k-1}$ by,

$$d_i^{(k+1)} = d_{i+1}^{(k)} - a^{(k)} l_i^{(k)} u_i^{(k)}.$$

The vectors $l^{(k)}$ and $u^{(k)}$ form the nonzero sections of rows and columns of the lower and the upper triangular factors of C, respectively.

If one wants to solve a Cauchy linear system then one can use this factorization algorithm followed by backward substitution. Another possibility consists in the recursive computation of the solution of the equations $CX = Z$ and $W^{\mathrm{T}}C = Y^{\mathrm{T}}$ and the application of the formula for the inverse matrix. This leads to the type-I algorithm described in [12] which computes C^{-1}. This way can be preferable if one has to solve several equation with one and the same matrix and different right-hand sides. Multiplication by C^{-1} can be carried out with $O(n\log^2 n)$ operations in general and $O(n\log n)$ for Cauchy matrices obtained from Toeplitz-plus-Hankel matrices after transformation.

The algorithm *LU-CAUCHY* can be combined with usual partial pivoting. Thus the condition concerning the strong nonsingularity of C is not essential.

Let us show how to get a factorization for the inverse matrix A^{-1}. This leads to a Levinson-type algorithm for computing the generators of C^{-1} which is similar to that presented in [17] and coincides with that in [12] for the case A. [5]

The Levinson-type recursions are based on the"bordering" formula for the inverse matrix

$$\begin{aligned} A^{-1} &= \begin{bmatrix} I & -A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & (A_{11}^{\#})^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix} \\ &= \begin{bmatrix} A_{11}^{-1} - URL & -UR \\ -RL & R \end{bmatrix}, \end{aligned} \tag{2.4}$$

where

$$U = A_{11}^{-1}A_{12}, \quad L = A_{21}A_{11}^{-1}, \quad R = (A_{11}^{\#})^{-1}.$$

We apply this formula to Cauchy matrices $C = [a_{ij}]_1^n$ satisfying (2.1). Let C_k $(k = 1, \ldots, n)$ be the principal submatrices which are assumed to be nonsingular. Furthermore we represent C_{k+1} as

$$C_{k+1} = \begin{bmatrix} C_k & q_k \\ p_k^{\mathrm{T}} & s \end{bmatrix}.$$

[5] Let us note that the corresponding recursion formula in [12] is not correct in case some of the d_js coincide.

The upper and lower factors of the UDL-factorization of C^{-1} are obtained from the solutions of the equations

$$C_k u_k = q_k, \qquad l_k^{\mathrm{T}} C_k = p_k^{\mathrm{T}}. \tag{2.5}$$

In order to find a fast algorithm for the factorization of C^{-1} we have to describe the recursion for the vectors q_k and p_k and for the generators X_k and W_k of C_k^{-1}, i.e. for the solutions of the equations $C_k X_k = Z_k$ and $W_k^{\mathrm{T}} C_k = Y_k^{\mathrm{T}}$, and in the case B also for the diagonals of C_k^{-1}.

We introduce the notations

$$Z_k = \mathrm{col}(z_i^{\mathrm{T}})_1^k, \quad Y_k = \mathrm{col}(y_j)_1^k,$$
$$D_k(c) = \mathrm{diag}(c_i)_1^k, \quad D_k(d) = \mathrm{diag}(d_j)_1^k.$$

Furthermore, let $\{e_i\}$ stand for the standard basis in $\mathbb{C}^k$.

Lemma 2. (1) *The solutions $u_k = (u_{kj})_1^k$ of the equations* (2.5) *can be computed from the generators X_k, W_k via*

$$u_{kj} = \begin{cases} (d_j - d_{k+1})^{-1} e_j^{\mathrm{T}} X_k (y_{k+1} - W_k^{\mathrm{T}} q_k): \ d_j \neq d_{k+1}, \\ \\ e_j^{\mathrm{T}} X_k W_k^{\mathrm{T}} (D_k(c) - d_{k+1} I_k)^{-1} q_k: \ d_j = d_{k+1} \end{cases} \tag{2.6}$$

where

$$q_k = (D_k(c) - d_{k+1} I_k)^{-1} Z_k y_{k+1}. \tag{2.7}$$

Analogous equalities hold for the solution l_k.

(2) *The generators X_{k+1} for C_{k+1} can be computed from the generators of C_k via*

$$X_{k+1} = \begin{bmatrix} X_k \\ 0 \end{bmatrix} - \frac{1}{\alpha_k} \begin{bmatrix} u_k \\ -1 \end{bmatrix} (z_{k+1}^{\mathrm{T}} - p_k^{\mathrm{T}} X_k), \tag{2.8}$$

where

$$p_k^{\mathrm{T}} = z_{k+1}^{\mathrm{T}} Y_k (c_{k+1} I_k - D_k(d))^{-1}, \quad \alpha_k = a_{k+1,k+1} - p_k^{\mathrm{T}} u_k. \tag{2.9}$$

An analogous equality holds for the generator W_{k+1}.

(3) *The diagonal of C_{k+1}^{-1} can be computed from the diagonal of C_k^{-1} via*

$$\mathrm{diag}\, C_{k+1}^{-1} = \left(\mathrm{diag}\, C_k^{-1} - \alpha_k^{-1}\, \mathrm{diag}(u_{ki} l_{ki})_{i=1}^k, \ \alpha_k^{-1} \right). \tag{2.10}$$

Proof. From the displacement equality

$$D_k(d) C_k^{-1} - C_k^{-1} D_k(c) = -X_k W_k^{\mathrm{T}}$$

we conclude that

$$\begin{aligned} &(D_k(d) - d_{k+1} I_k) C_k^{-1} (D_k(c) - d_{k+1} I_k)^{-1} Z_k y_{k+1} \\ &\quad = (C_k^{-1} - X_k W_k^{\mathrm{T}} (D_k(c) - d_{k+1} I_k)^{-1}) Z_k y_{k+1} = X_k (y_{k+1} - W_k^{\mathrm{T}} q_k). \end{aligned}$$

Taking (2.7) into account we obtain

$$(D_k(d) - d_{k+1})u_k = X_k(y_{k-1} - W_k^{\mathrm{T}} q_k). \tag{2.11}$$

In case when $d_j \neq d_{k+1}$ this leads to the first equality in Lemma 2. In case when $d_j = d_{k+1}$ we obtain

$$0 = e_j^{\mathrm{T}} C_k^{-1} q_k - e_j^{\mathrm{T}} X_k W_k^{\mathrm{T}} (D_k(c) - d_{k+1} I_k)^{-1} q_k.$$

This leads to the second equality in Lemma 2.

The recursion in (2) can be checked immediately and the recursion in (3) follows from (2.4). □

This lemma leads to the following algorithm for the factorization of C^{-1} for a strongly nonsingular Cauchy matrix $C = [a_{ij}]_1^n$.

Algorithm UL-INVCAUCHY

1. Initialization: $X_1 = \frac{1}{a_{11}} Z_1$, $W_1 = \frac{1}{a_{11}} Y_1$, in the case B: $\operatorname{diag} C_1^{-1} = \frac{1}{a_{11}}$.
2. For $k = 1, \ldots, n-1$,
   (a) Compute q_k by (2.7) and p_k by (2.9).
   (b) Compute u_k by (2.6) and analogously l_k.
   (c) Compute α_k by (2.9) and X_{k+1} by (2.8), and analogously W_{k+1}.
   (d) In case B compute $\operatorname{diag} C_{k+1}^{-1}$ by (2.10).

Now the vectors $[-u_k^{\mathrm{T}} \quad 1]^{\mathrm{T}}$ form the nonzero sections of columns of the upper triangular factor and the vectors $[-l_k \quad 1]$ the lower triangular factor of the UDL-factorization of C^{-1}. The numbers α_k correspond to the diagonal factor. In order to get the inverse matrix one has only to store the generators $X = X_n$ and $W = W_n$ and, in case B, also the diagonal $\operatorname{diag} C_n^{-1}$. This leads also to a Levinson-type algorithm for solving Cauchy systems. From the numerical view point it may be less accurate when the nodes are close to each other (part of the reason for this is the formula presence of the division by $(d_j - d_{k+1})$ in the formula (2.6). Furthermore, pivoting is more expensive and the algorithm is not very suitable for parallelization due to inner product calculations. But the Levinson-type algorithm is important because it provides a simple updating formula for the inverse matrix or for the solution of systems of equations.

3. Symmetric pivoting

In this section we describe certain features of fast Cauchy solvers that can be exploited to lower the number of operations when the matrices under consideration are symmetric or hermitian. This will lead us to symmetric (or hermitian) fast Cauchy solvers. We next describe how these symmetric solvers can be applied to the special Cauchy matrices obtained from Toeplitz and Toeplitz-plus-Hankel matrices. We start with some general observations.

3.1. Dependent generators

If the generators Z and Y of the Cauchy matrix depend via a relation

$$Y = ZK \quad \text{or} \quad Y^* = Z^*K, \tag{3.12}$$

where K is an $r \times r$ matrix, then it is sufficient to compute in the algorithm *LU-CAUCHY* only one of the generators. This follows immediately from the fact that the relation (3.12) is inherited by Schur complementation. This also applies to the algorithm *UL-INVCAUCHY*. The following is easily checked.

Proposition 3. (1) *If* $c_i = d_i$ $(i = 1, \ldots, r)$ *and* C *is symmetric then* r *is even and the first relation of* (3.12) *holds for certain* Z *and*

$$K = \begin{bmatrix} 0 & I_{r/2} \\ -I_{r/2} & 0 \end{bmatrix}.$$

(2) *If* $c_i = -\bar{d}_i$ $(i = 1, \ldots, r)$ *and* C *is hermitian then the second relation of* (3.12) *holds for certain* Z *and*

$$K = \begin{bmatrix} I_p & 0 \\ 0 & -I_q \end{bmatrix}.$$

The first assertion follows from the fact that in this case $\nabla_{c,d}(C)$ is skew symmetric. Using symmetric Gaussian elimination, it can be easily shown that a skew-symmetric matrix S has even rank and admits the representation $S = ZKZ^{\mathrm{T}}$, where K has the form as in Proposition 3, case (1). The second assertion follows from the fact that in this case $\nabla_{c,d}(C)$ is hermitian.

For some Cauchy matrices obtained as the result of transformations of Toeplitz-plus-Hankel matrices the generators of the Stein displacement are connected rather then those of the Sylvester displacement. [6] That means we have a relation

$$C - D(c)CD(d) = ZKZ^{\mathrm{T}} \quad \text{or} \quad C - D(c)CD(d) = ZKZ^* \tag{3.13}$$

for a certain $r \times r$ matrix K and a $n \times r$ matrix Z. Matrices satisfying a relation (3.13) are sometimes called *(generalized) Pick matrices*. Clearly, they are also (generalized) Cauchy matrices.

If C satisfies (3.13) then it is not immediately clear how the relation between the generators are inherited in the Schur complement. Our suggestion is to transform the Stein displacement equation (3.13) into a Sylvester displacement

[6] The Stein displacement is often called *discrete time* and the Sylvester displacement *continuous time* Lyapunov displacement (cf. [18]).

equation using linear fractional transformations of the nodes c_i and d_j. The following is easily checked.

Proposition 4. *Let ϕ be a linear fractional transformation, $\phi(\lambda) = (\alpha\lambda - \beta)/(\alpha\lambda + \beta)$, and let $c_i = \phi(x_i)$, $d_j = \phi(y_j)$. Then the Stein displacement equation $C - D(c)CD(d) = R$ is equivalent to the Sylvester displacement equation*

$$D(x)C + CD(y) = \frac{1}{2\alpha\beta} D(\alpha x + \beta\mathbf{1})RD(\alpha y + \beta\mathbf{1}).$$

3.2. Bunch–Kaufman–Parlett pivoting algorithm

Now we discuss the problem of pivoting for the symmetric and hermitian cases. In the symmetric (or hermitian) case by working with one generator only we hope to decrease the number of arithmetic operation required in the unsymmetric case. However, if proper care is not taken, pivoting will destroy the symmetry. In order to keep the symmetry throughout the computation partial pivoting can be replaced by diagonal pivoting. However, simple diagonal pivoting, in general, does not bound the rate of growth in the data. The rate of growth can be bounded if some sort of block diagonal pivoting is used. Several block diagonal pivoting techniques were discussed in the literature for computing factor of (indefinite) symmetric matrices (see [3–5]). The most widely used is the technique presented in [5]. We will refer to this technique as *Bunch–Kaufman–Parlett* pivoting algorithm, or the BKP algorithm for short. This algorithm uses block diagonal pivoting with blocks of order 1 or 2. In what follows we will outline the BKP algorithm for factorization of a general indefinite symmetric matrix, and next we will show how it can be incorporated into fast symmetric Cauchy solvers.

For a given symmetric matrix A, the BKP-algorithm computes a permutation P, a unit lower triangular matrix M and a block diagonal matrix D with block of order 1 or 2 such that

$$P^{\mathrm{T}}AP = MDM^{\mathrm{T}}. \tag{3.14}$$

The permutation matrix P depends on a parameter $0 < \alpha < 1$ which controls the rate of growth of pivots. The rate is minimized for $\alpha = (1 + \sqrt{17})/8$.

The factorization proceeds in stages. In stage $k + 1$ $(k = 0, \ldots, n - 1)$ one operates on the matrix $A^{(k)}$, with $A^{(0)} = A$. Given $A^{(k)}$, a symmetric permutation of rows and columns is applied to $A^{(k)}$ so a satisfactory diagonal pivot block of order 1 or 2 can be determined. If $P^{(k)}$ is such a permutation, and

$$(P^{(k)})A^{(k)}(P^{(k)})^{\mathrm{T}} = \begin{bmatrix} D^{(k)} & E^{\mathrm{T}} \\ E & G \end{bmatrix}, \tag{3.15}$$

with $D^{(k)}$ and G symmetric, then the pivot block is $D^{(k)}$, and $A^{(k+1)}$ is computed as the Schur complement of $D^{(k)}$, i.e.

$$A^{(k+1)} = G - E(D^{(k)})^{-1}E^{\mathrm{T}}.$$

The significant elements of the new columns of M are computed as $E(D^{(i-1)})^{-1}$. The whole process is iterated until all columns of M are determined.

The most important aspect of the method is a procedure for finding the permutation matrix $P^{(k)}$ and determining the order of the pivot block $D^{(k)}$. In [5] this is done in two steps. First, one checks whether a suitable diagonal permutation that would bound pivot element growth exists. If such a pivot cannot be found satisfactory a 2×2 pivot is calculated instead.

A single iteration, that is the transitions from $A^{(k)}$ to $A^{(k+1)}$ is summarized in Algorithm BKP. There, in order to simplify notation, superscripts were dropped. Also, following [5], I_{ij} is used to denote a permutation which interchanges row i with row j.

Algorithm BKP

Let $A = A^{(k)} = [a_{ij}]$ be the current $(n-k) \times (n-k)$ matrix. One iteration of the algorithms computes a permutation P, pivot block D of order s ($s = 1$ or 2), s columns of the lower triangular factor M, and the Schur complement $A^{(k+1)}$ corresponding to D.

1. Initialization: Set $\alpha = (1 + \sqrt{17})/8$.
2. (a) Search for a 1×1 diagonal pivot, i.e. set $s = 1$,
   - (a1) compute $\lambda = |a_{m1}| = \max_{2 \leqslant i \leqslant n-k} |a_{i1}|$,
   - (a2) if $|a_{11}| \geqslant \alpha\lambda$ the element c_{11} is a good pivot element, set $P = I$ and go to (c),
   - (a3) compute $\sigma = \max_{i \neq m} |a_{i1}|$,
   - (a4) if $|a_{11}|\sigma \geqslant \alpha\lambda^2$ the element a_{11} is still a good pivot element, set $P = I$ and go to (c),
   - (a5) if $|a_{mm}| \geqslant \alpha\sigma$ the element a_{mm} is a good pivot element, set $P = I_{1m}$ and go to (c).

   (b) There is not any satisfactory 1×1 diagonal pivot, set $s = 2$ and $P = I_{2m}$.

   (c) Compute
   - (c1) set $PAP^{\mathrm{T}} = \begin{bmatrix} D & E^{\mathrm{T}} \\ E & G \end{bmatrix}$, where D is $s \times s$,
   - (c2) compute the Schur complement $A^{(k+1)}$ of D,
   - (c3) compute the significant elements of the new columns in M as ED^{-1}.

The algorithm easily extends to the case of a hermitian A. A repeated application of Algorithm BKP will produce the desired decomposition (3.14).

The decomposition can be used to solve a linear system of equations $Ax = b$. Given the symmetric decomposition $PCP^T = MDM^T$ this can be accomplished in $MS_r(n) = n^2$ multiplications and $AS_r(n) = n^2$ additions for a real A. For a complex matrix most operations has to be performed in complex arithmetic. A complex multiplication is equivalent to four real multiplications and two real additions. A complex addition is equivalent to two real additions. Thus in the complex case, given the symmetric (or hermitian) decomposition of A, the cost of solving the linear system $Ax = b$ is $MS_c(n) = 4n^2$ multiplications and $AS_c(n) = 4n^2$ additions.

3.3. Symmetric pivoting on generators

Recall that the possibility to speed up the factorization procedure for generalized Cauchy matrices from $O(n^3)$ to $O(n^2)$ complexity is based on the fact that it is sufficient to carry out all operations on the generators. Concerning pivoting the following obvious observation is in order.

If C is a Cauchy matrix with generators Z and Y and P is a permutation matrix then $\tilde{C} = PCP^T$ has the generators PZ and PY and the nodes of $\tilde{C}$ are the corresponding permutations of the original nodes.

The above observation when combined with Algorithm LU-Cauchy and Algorithm BKP leads to Algorithm BKP-Cauchy which is described below.

Let $A_{i:j}$ denote a matrix composed of rows i through j of the matrix A. Similarly, if $J \subset \{1, 2, \ldots, n\}$ then A_J will denote a matrix composed of all rows of A which indices are taken from J (in the increasing order).

Algorithm BKP-Cauchy

Let C be an $n \times n$ Cauchy matrix defined by the generators Y and $Z = YK$, the nodes (c, d) where $c_i \neq c_j$ for $i \neq j$, and the diagonal $\operatorname{diag}(C)$ if $c = d$. The algorithms computes the permutation P, the pivot block D of order s ($s = 1$ or 2) s columns of the lower triangular factor M, and the generators $\hat{Y}$ and $\hat{Z}$, such that $\hat{Z} = \hat{Y}K$.

1. Initialization: Set $\alpha = (1 + \sqrt{17})/8$, and $s = 1$,
2. (a) Look for a 1×1 diagonal pivot,
   - (a1) compute the off diagonal entries of the first column of C, $C_1 = (a_{i1})_1^n = (\operatorname{diag}(c_{2:n}) - d_1 I)^{-1} Y_{2:n} K y_1$, and set $\lambda = |a_{m1}| = \max_{2 \leq k \leq n} |a_{k1}|$,
   - (a2) if $|a_{11}| \geq \alpha\lambda$ set $P = I$, $D = a_{11}$, $E = C_1$, and go to (c),
   - (a3) compute the off diagonal entries of the mth column of C, $C_m = (a_{im})_1^m = (\operatorname{diag}(c_J) - d_m I)^{-1} Y_J K y_m$, where $J = \{1, \ldots, m-1, m+1, \ldots, n\}$, and set $\sigma = \max_{i \neq m} |a_{im}|$,
   - (a4) if $|a_{11}|\sigma \geq \alpha\lambda^2$ set $P = I$, $D = a_{11}$, $E = C_1$, and go to (c),
   - (a5) if $|a_{mm}| \geq \alpha\sigma$ set $P = I_{1m}$, $D = a_{mm}$, $E = C_m$, and go to (c)

(b) Set $s = 2$,
  (b1) $P = I_{2m}$,
  (b2) $D = \begin{bmatrix} a_{11} & a_{1m} \\ a_{m1} & a_{mm} \end{bmatrix}$, and $E = [C_1, C_m]$.
(c) Compute other relevant quantities
  (c2) permute Y, E, b, c, d and $\operatorname{diag}(C)$ according to P,
  (c3) compute the generator $\hat{Y} = Y_{s+1:n} - \hat{E}D^{-1}Y_{1:s}$, where $\hat{E}$ equals E with its first s rows dropped,
  (c4) compute entries in rows $s+1$ through n in the first s columns of the lower triangular factor M as $\hat{E}D^{-1}$,
  (c5) compute $\operatorname{diag}(\hat{C}) = \operatorname{diag}(C) - \operatorname{diag}(ED^{-1}E^{\mathrm{T}})$.

A repeated application of Algorithm BKP-Cauchy will produce the desired decomposition (3.14). The cost of the algorithm will depend on the order $n \times r$ of generators, and on how often the step (a3) followed by (c) are executed. If (a3) is always performed and is always followed by (c), then, for a real matrix, it is maximal and is $\frac{3}{2}n^2r$ multiplications and $\frac{3}{2}n^2r$ additions. On the other hand if (a2) is always followed by (b) or (c) then the cost is only n^2r multiplications and n^2r additions. The cost of Algorithm BKP for a complex matrix is roughly four times that for a real matrix.

Estimates of the cost (measured by the number of real multiplications) of computing the solution of a symmetric $n \times n$ linear system $Cx = b$ given the generators of C are given in Table 1.

3.4. Computation of generators based on DFT

In this section we provide some more detailed description of generators for Cauchy matrices appearing in Section 2, [16] as the result of transformed Toeplitz matrices.

For a complex number ξ, let $\mathscr{F}(\xi)$ denote the matrix

$$\frac{1}{\sqrt{n}}\left[c_i^{j-1}\right]_1^n$$

where c_i are the nth roots of ξ (in certain unspecified order). We set $\mathscr{F}(1) = \mathscr{F}$. $\mathscr{F}$ is the matrix of the DFT and $\mathscr{F}(\xi) = \mathscr{F}\operatorname{diag}(1, c_1, \ldots, c_1^{n-1})$.

Table 1
Cost of solving $Cx = b$

Matrix	RHS	Cost (real multiplications)	
C	b	Max	Min
Complex	Complex	$4(\frac{3}{2}n^2r + n^2)$	$4(n^2r + n^2)$
Real	Complex	$\frac{3}{2}n^2r + 2n^2$	$n^2r + 2n^2$
Real	Real	$\frac{3}{2}n^2r + n^2$	$n^2r + n^2$

The matrix-vector multiplication $\mathscr{F}(\xi)a$ can be accomplished via FFT in $5n\log n$ and $2.5n\log n$ real arithmetic operations for a complex and real vector a, respectively (assuming that $n = 2^t$ for some positive integer t). These kind of matrix-vector multiplications compute generators of the Cauchy matrices considered in Section 2, part I. As only a small number of operations of the type $\mathscr{F}(\xi)a$ are performed in transforming Toeplitz into Cauchy matrices, we will not include the cost of these matrix-vector operations in the overall estimation of the amount of solving the resulting Cauchy linear systems.

Let $T = [a_{i-j}]$ and $H = [s_{i+j}]$ be $n \times n$ Toeplitz and Hankel matrices, respectively. We will now list generators corresponding to the transformations considered in Sections 2.1–2.4, part I.

(a) `HTfftHC` – Hermitian Toeplitz to complex hermitian Cauchy: Suppose that $a_{-k} = \bar{a}_k$ and $c_i = d_i = \omega_j = \exp(2\pi \mathrm{i} j/n)$ $(\mathrm{i} = \sqrt{-1})$ According to Theorem 4, Part I,

$$\tilde{C} = \mathscr{F} T \mathscr{F}^*$$

is a hermitian Cauchy matrix. More important for our purposes is the fact that the matrix $C = n\mathrm{i}/2\,\mathrm{diag}(\omega)\tilde{C}$ is a symmetric Cauchy matrix with $c = d$. The quantities defining C can be derived from the vectors

$$a' = \left[\frac{a_0}{2}, a_1, \ldots, a_{n-1}\right]^{\mathrm{T}}, \quad a'' = [na_0, (n-1)a_1, \ldots, a_{n-1}]^{\mathrm{T}}. \tag{3.16}$$

The generator Y and the diagonal of C are given by

$$\mathrm{diag}(C) = \frac{n\mathrm{i}}{2}\mathrm{diag}(\omega) f, \tag{3.17}$$

$$Y = \begin{bmatrix} e_1 & 1 \\ \vdots & \vdots \\ e_n & 1 \end{bmatrix}, \tag{3.18}$$

where

$$e = \mathrm{Im}\,\mathscr{F} a' \quad \text{and} \quad f = \mathrm{Re}\,\mathscr{F} a''. \tag{3.19}$$

Finally, $Z = YK$ with $K = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$.

Since C is symmetric and $c = d$ Algorithm BKP-Cauchy can be readily applied to C.

(b) `HTfftSC` – Hermitian Toeplitz to real symmetric Cauchy: Let ω, e, f, $\mathrm{diag}(C)$, Y, K and $\tilde{C}$ be as in (a). The fractional transformation

$$c_j = \frac{\zeta + \omega_j}{\zeta - \omega_j} \cdot \mathrm{i} \tag{3.20}$$

maps the complex node vector ω into a real node vector $c = (c_j)$. The matrix

$$C = \operatorname{diag}(c_j + \mathrm{i})\tilde{C}\operatorname{diag}(c_j - \mathrm{i})$$

is a real symmetric Cauchy matrix. The relevant quantities defining C are the same as in the case (a) except for the nodes and the diagonal elements, which are now c and $d = c$, and $\operatorname{diag}(C) = \operatorname{diag}(c_j^2 + 1)f$.

(c) `STfftSC` – Symmetric Toeplitz to symmetric Cauchy: Let $\xi = \mathrm{i}$ and $\rho = (\rho_j)_1^n$ with $\rho_j = \exp((4j+1)\pi\mathrm{i}/2n)$. Then, according to Theorem 7, Part I,

$$\tilde{C} = \mathscr{F}(\mathrm{i})T\mathscr{F}(\mathrm{i})^{\mathrm{T}}$$

is a symmetric Cauchy matrix. The fractional transformation (3.20) transforms ρ into a real vector c. Now, the matrix

$$C = \frac{1}{2\mathrm{i}}\operatorname{diag}(1 - c_j)\tilde{C}\operatorname{diag}(1 - c_j)$$

is a symmetric Cauchy matrix with the entries

$$c_{ij} = \frac{e_i + e_j}{c_i + c_j},$$

where

$$e = \mathscr{F}(\mathrm{i})a' + \mathscr{F}(-\mathrm{i})a' \tag{3.21}$$

and a' is defined by (3.16). Thus the matrix C is defined by the nodes c and $d = -c$, and the generators Y and $Z = YK$ where

$$Y = \begin{bmatrix} e_1 & 1 \\ \vdots & \vdots \\ e_n & 1 \end{bmatrix} \quad \text{and} \quad K = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Algorithm BKP-Cauchy can be now applied to C. The cost of solving $Tx = b$ is analogous to that in (a) for a complex T, and that in (b) for a real T.

Remark 5. Note from (3.21) that if T is real then e is also real. Thus we have an alternative (to that one in (b)) transformation that takes a real symmetric Toeplitz matrix into a real symmetric Cauchy matrix. We will refer to this transformation in the section on numerical experiments as `RTfftSC`.

(d) `HTpHfftHC` – Hermitian Toeplitz-plus-Hankel to hermitian Cauchy: We now consider the case when the Toeplitz-plus-Hankel $A = T + H$ matrix is hermitian. As remarked in Section 3, Part I, if $\xi = -\eta = \mathrm{i}$ then

$$C = \mathscr{F}(\xi)A\mathscr{F}(\eta)^{\mathrm{T}}$$

is hermitian. The transformation from A to C is based on the vectors

$$t_{\mathrm{col}} = \left[\frac{t_0}{2}, t_1, \ldots, t_{n-1}\right]^{\mathrm{T}}, \quad t'_{\mathrm{col}} = \left[n\frac{t_0}{2}, (n-1)t_1, \ldots, t_{n-1}\right]^{\mathrm{T}},$$

$$t_{\mathrm{row}} = \left[\frac{t_0}{2}, t_{-1}, \ldots, t_{-n+1}\right]^{\mathrm{T}}, \quad t'_{\mathrm{row}} = \left[n\frac{t_0}{2}, (n-1)t_{-1}, \ldots, t_{-n+1}\right]^{\mathrm{T}}$$

to transform T, and on the vectors

$$h_{\mathrm{row}} = \left[\frac{s_{n-1}}{2}, s_n, \ldots, s_{2n-2}\right]^{\mathrm{T}}, \quad h_{\mathrm{col}} = \left[\frac{s_{n-1}}{2}, s_{n-2}, \ldots, s_0\right]^{\mathrm{T}},$$

to transform H. These vectors are transformed via $\mathscr{F}(\xi)$ or $\mathscr{F}(\eta)$. The transformed vectors are combined according to Theorem 9, Part I to give $n \times 4$ generators Z and Y of C. The generators satisfy $\bar{Z} = YK$ for

$$K = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix}.$$

The hermitian version of Algorithm BKP-Cauchy can then be applied.

3.5. *Computation of generators based on trigonometric transformations*

As was pointed out in Section 3, the trigonometric transformations have the advantage over the DFT that they are real and hence allow to stay in reals when applied to real data. There are numerous fast algorithms for applying trigonometric transformations, see [24–26]. However, as they are all of order $O(n \log n)$ for vectors of length n, a simple implementation based on the complex FFT can be used (in all cases when n is large) as it will only effect lower terms in the cost of solving the resulting Cauchy linear system.

The sine-I transformation in Theorem 9, part I is straightforward to implement and was used in numerical tests. For a Toeplitz-plus-Hankel matrix A this transformation operates on vectors

$$t_{\mathrm{col}} = \left[\frac{t_0}{2}, t_1, \ldots, t_{n-1}\right]^{\mathrm{T}}, \quad t_{\mathrm{row}} = \left[\frac{t_0}{2}, t_{-1}, \ldots, t_{-n+1}\right]^{\mathrm{T}},$$

$$t_{\mathrm{col-row}} = [t_0, t_1 + t_{-1}, \ldots, t_{n-1} + t_{-n+1}]^{\mathrm{T}},$$

$$t'_{\mathrm{col-row}} = [nt_0, (n-1)(t_1 + t_{-1}), \ldots, t_{n-1} + t_{-n+1})]^{\mathrm{T}}$$

to transform T, and on the vectors

$$h_{\mathrm{col}} = \left[\frac{s_{n-1}}{2}, s_n, \ldots, s_{2n-2}\right]^{\mathrm{T}}, \quad h_{\mathrm{row}} = \left[\frac{s_{n-1}}{2}, s_{n-2}, \ldots, s_0\right]^{\mathrm{T}},$$

$$s_{\mathrm{col-row}} = [s_{n-1}, s_{n-2} + s_n, \ldots, s_0 + s_{2n-2}]^{\mathrm{T}},$$

$$s'_{\mathrm{col-row}} = [ns_{n-1}, (n-1)(s_{n-2} + s_n), \ldots, s_0 + s_{2n-2}]^{\mathrm{T}}$$

to transform H. The transformed vectors are combined according to Theorem 9, Part I to give $n \times 4$ generators Z and Y of C. The generators satisfy $\bar{Z} = YK$ for

$$K = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

The hermitian version of Algorithm BKP-Cauchy can then be applied. We will refer to this transformation as `HTpHsineHC`.

As stated in Theorem 9, Part I, for a real symmetric Toeplitz matrix, the transformed matrix is a 2×2 block diagonal with blocks that are symmetric and each having Cauchy rank 2. The symmetric version of Algorithm BKP-Cauchy can be applied to each of the two blocks. This leads to a potential saving of 50% in the cost of solving a real symmetric Toeplitz system of linear equations. We will refer to this transformation as `STsineSC`.

4. Numerical experiments

Numerical tests for all transformation based algorithms were conducted on a MacIntosh PowerBook 180 computer using MATLAB 4.1. The relative machine precision was eps $= 2.2e-16$.

In the tests we generated indefinite Toeplitz or Toeplitz-plus-Hankel matrices as follows. A matrix $\hat{A}$ was first generated randomly. Next $\hat{A}$ was modified to obtain another matrix A according to the formula

$$A = \hat{A} - (\lambda + \text{eps}^{\alpha})I_n,$$

where λ was an eigenvalue of $\hat{A}$, eps the machine relative precision, and $0 < \alpha < 1$. Such modification does not change the underlying structure of the original Toeplitz or Toeplitz-plus-Hankel matrix. By varying α we could influence the magnitude of the condition number of A. We also varied the dimension n of A.

The right-hand side vector b in the linear system $Ax = b$ was chosen in such a way so the true solution x was a vector of ones. This was because such a vector does not reflect potential ill-conditioning of the matrix and hence may expose numerical deficiencies of the solvers.

In the tests we measured the residual error res$x$ in the computed solution $\tilde{x}$ as follows

$$\text{res}x = \frac{||A\tilde{x} - b||}{||A|| \cdot ||\tilde{x}|| + ||b||}.$$

We also measured the relative error res$C$ in the symmetric (or hermitian) decomposition of C as

$$\operatorname{res} C = \frac{||C - LML^{\mathrm{T}}||}{||C||},$$

and the relative error $\operatorname{res} A$ in the decomposition of A as

$$\operatorname{res} A = \frac{||A - FLML^{\mathrm{T}}G||}{||A||},$$

where F and G were the transformations that related the matrix A with the matrix C. The ratio of $\operatorname{res} A$ and $\operatorname{res} C$ measures the degradation of accuracy caused by the transformation from A to C and back to A. For the algorithm LU-CAUCHY, $\operatorname{res} C$ and $\operatorname{res} A$ are defined in an analogous way. For the algorithm UL-INVCAUCHY, $\operatorname{res} C$ and $\operatorname{res} A$ would have to be computed for C^{-1}, hence only $\operatorname{res} x$ was recorded.

4.1. Hermitian Toeplitz-plus-Hankel

Solving systems of equations with a complex hermitian Toeplitz-plus-Hankel coefficient matrix we compared the following algorithms.

ALG 1.1. Use `HTpHfftHC` to transform the matrix into a hermitian Cauchy matrix. Then apply algorithm *LU-CAUCHY* in combination with algorithm *BKP-Cauchy* for pivoting.

ALG 1.2. Instead of `HTpHfftHC` use the real transformation `HTpHsineHC`. (The advantage is that the resulting Cauchy matrix is real if the original matrix is real). Then apply *LU-CAUCHY* in combination with *BKP-Cauchy* as in ALG 1.1.

ALG 1.3. Apply the transformation `HTpHfftHC` and algorithm *LU-CAUCHY* but instead of *BKP-Cauchy* pivoting as in ALG 1.1, use the standard partial pivoting.

ALG 1.4. Use `HTpHfftHC`. Instead of algorithm *LU-CAUCHY* apply algorithm *UL-INVCAUCHY* combined with *BKP-Cauchy* pivoting.

ALG 1.3 was included in order to check whether symmetric pivoting may cause less accurate results than standard partial pivoting. ALG 1.4 was included in order to check whether the algorithm *UL-INVCAUCHY* may produce less accurate results than *LU-CAUCHY*. Representative results are given in Table 2.

The table suggests that accuracy of the symmetric pivoting on the Cauchy matrix C is comparable to that of unsymmetric pivoting. The accuracy of the decomposition of A is slightly lower than that of C. This is caused by the transformation of A into C, and possibly could be improved by a more careful implementation of FFT-like transformations. The residual error of the solution is comparable to that one of the decomposition of A.

Table 2
Complex Hermitian Toeplitz-plus-Hankel A

Size	Cond		ALG 1.1	ALG 1.2	ALG 1.3	ALG 1.4
80	3.8526×10^{4}	res x	3.1655×10^{-15}	9.3151×10^{-14}	9.5612×10^{-15}	5.5622×10^{-14}
		res A	6.9822×10^{-15}	5.0378×10^{-15}	7.1205×10^{-14}	
		res C	5.3745×10^{-16}	5.0378×10^{-15}	6.4230×10^{-15}	
	1.6275×10^{7}	res x	2.7303×10^{-15}	1.1814×10^{-13}	1.0432×10^{-14}	2.3952×10^{-13}
		res A	6.9938×10^{-15}	1.3900×10^{-14}	8.3806×10^{-14}	
		res C	4.7229×10^{-16}	1.3900×10^{-14}	1.4163×10^{-14}	
	2.6182×10^{15}	res x	3.2986×10^{-15}	1.2818×10^{-13}	7.8452×10^{-15}	3.3623×10^{-13}
		res A	3.9141×10^{-16}	9.2986×10^{-14}	7.1720×10^{-14}	
		res C	7.8609×10^{-16}	1.6000×10^{-14}	7.0866×10^{-15}	
120	2.9335×10^{3}	res x	7.3268×10^{-15}	2.8937×10^{-13}	5.1263×10^{14}	3.1281×10^{-13}
		res A	1.8614×10^{-14}	5.5601×10^{-13}	1.6654×10^{13}	
		res C	2.0574×10^{-15}	4.0819×10^{-14}	1.2077×10^{14}	
	2.9691×10^{6}	res x	6.9222×10^{-15}	2.9267×10^{-13}	5.1182×10^{-14}	3.8811×10^{-13}
		res A	1.8270×10^{-14}	5.4112×10^{-13}	1.7444×10^{-13}	
		res C	1.4279×10^{-15}	4.0414×10^{-14}	1.4117×10^{-14}	
	2.0951×10^{15}	res x	5.5690×10^{-15}	3.2705×10^{-13}	5.4942×10^{-14}	4.8662×10^{-13}
		res A	1.7484×10^{-14}	5.1699×10^{-13}	1.7495×10^{-13}	
		res C	2.0453×10^{-15}	3.9838×10^{-14}	1.4298×10^{-14}	
150	3.6807×10^{3}	res x	1.1590×10^{-14}	2.4397×10^{-13}	2.8876×10^{-14}	2.1523×10^{-13}
		res A	2.8423×10^{-14}	5.5647×10^{-13}	3.0573×10^{-13}	
		res C	1.0333×10^{-14}	3.8046×10^{-15}	2.8126×10^{-14}	
	3.5337×10^{6}	res x	1.0409×10^{-14}	2.3590×10^{-13}	2.2918×10^{-14}	5.7461×10^{-13}
		res A	2.6824×10^{-14}	3.5981×10^{-13}	2.6781×10^{-13}	
		res C	8.5036×10^{-14}	4.7821×10^{-15}	1.4756×10^{-14}	
	2.8970×10^{15}	res x	8.1843×10^{-15}	2.0285×10^{-13}	1.4991×10^{-14}	6.1183×10^{-13}
		res A	2.6317×10^{-14}	3.6961×10^{-13}	2.4262×10^{-13}	
		res T	2.9684×10^{-15}	3.9237×10^{-15}	3.0188×10^{-14}	

Table 3
Cost of solving $Ax = b$

	Multiplications	
	Max	Min
ALG 1.1	$28n^2$	$20n^2$
ALG 1.2	$28n^2$	$20n^2$
ALG 1.3	$36n^2$	$36n^2$
ALG 1.4	$52n^2$	$52n^2$

The cost of the three methods (excluding the cost of the transformations) can be derived from Table 1 with $r = 4$. These costs (measured by the number of real multiplications) are summarized in Table 3.

4.2. Real symmetric Toeplitz

Solving systems of equations with a real symmetric Toeplitz coefficient matrix we compared the following algorithms.

ALG 2.1. Use `RTfftSC` [7] to transform the matrix into a real symmetric Cauchy matrix. Then apply algorithm LU-CAUCHY in combination with algorithm BKP-Cauchy for pivoting.

ALG 2.2. Use `HTfftSC` for the transformation into a real symmetric Cauchy matrix. Then proceed as in ALG 2.1. Apply BKP-Cauchy as in ALG 1.1.

ALG 2.3. Apply the transformation `HTfftHC` to transform the matrix into a complex hermitian matrix. Then proceed as in ALG 2.1.

ALG 2.4. Apply the transformation of the nonsymmetric standard choice in Section 2.1 of Part I. (This is the type-II algorithm proposed in [12] where a general Toeplitz matrix is transformed into a complex matrix with the Cauchy rank 2). Then proceed as in ALG 2.1.

ALG 2.5. Use `RTsineSC` to transform the matrix into a direct sum of two real symmetric Cauchy matrices of about half the size. Then proceed as in ALG 2.1 for the two matrices.

ALG 2.6. A look-ahead Levinson algorithm `dsytep` [8] for solving symmetric (possibly not positive definite) Toeplitz systems of linear equations from [6].

The algorithms ALG 2.1–2.5 are all based on the transformation approach described earlier in this paper. Algorithm ALG 2.6 on the other hand represents another approach known as a *look-ahead* technique developed in [6] for stabilizing the Levinson method when some intermediate submatrices of a symmetric Toeplitz matrix are ill-conditioned, see also [11] and references therein. In the look-ahead approach the user selects the length k of the look-ahead step. Ideally this length should equal the maximal number of consecutive ill-conditioned leading submatrices. The extent of the look-ahead step determines the cost of the method. For the extreme cases $k = 0$ and $k = n - 1$ the look-ahead method becomes the classical Levinson method and the classical Gaussian elimination with partial pivoting, respectively.

It is not difficult to construct well-conditioned symmetric Toeplitz matrices with a prescribed number l of consecutive ill-conditioned leading submatrices. An obvious example is a matrix of the form

[7] See Section 3.4.

[8] This FORTRAN code was run on am IBM RS6000 workstation.

Table 4
Real Symmetric Toeplitz T

Size	Cond		ALG 2.1	ALG 2.2	ALG 2.3	ALG 2.4	ALG 2.5	ALG 2.6
80	2.1×10^{1}	res x	5.4×10^{-15}	2.8×10^{-16}	2.2×10^{-16}	9.3×10^{-5}	1.7×10^{-14}	1.5×10^{-2}
		err x	1.1×10^{-14}	5.6×10^{-16}	4.5×10^{-16}	1.8×10^{-14}	3.5×10^{-14}	3.1×10^{-2}
		res C	5.5×10^{-16}	1.1×10^{-16}	2.2×10^{-16}	2.7×10^{-15}	2.2×10^{-17}	
		res T	1.2×10^{-14}	4.3×10^{-15}	1.4×10^{-15}	1.7×10^{-14}	3.9×10^{-14}	
	2.4×10^{8}	res x	1.5×10^{-8}	1.5×10^{-16}	1.6×10^{-16}	2.0×10^{-16}	1.9×10^{-15}	6.21×10^{-12}
		err x	1.8×10^{-1}	8.4×10^{-9}	3.5×10^{-9}	1.4×10^{-8}	8.5×10^{-8}	2.22×10^{-6}
		res C	1.4×10^{-9}	3.5×10^{-16}	6.7×10^{-16}	1.2×10^{-14}	1.0×10^{-17}	
		res T	8.1×10^{-8}	4.1×10^{-15}	1.3×10^{-15}	1.4×10^{-14}	3.3×10^{-14}	
	8.2×10^{14}	res x	2.2×10^{-8}	1.8×10^{-16}	1.4×10^{-16}	2.8×10^{-16}	2.0×10^{-15}	3.7×10^{-10}
		err x	1.2×10^{0}	1.5×10^{-2}	1.4×10^{-2}	3.2×10^{-2}	7.0×10^{-2}	1.0×10^{0}
		res C	7.0×10^{-9}	3.6×10^{-16}	6.5×10^{-16}	1.3×10^{-14}	8.8×10^{-18}	
		res T	9.7×10^{-8}	4.0×10^{-15}	1.3×10^{-15}	1.7×10^{-14}	3.4×10^{-14}	
120	6.6×10^{2}	res x	2.2×10^{-11}	1.8×10^{-15}	7.3×10^{-15}	2.3×10^{13}	4.2×10^{-14}	5.9×10^{-11}
		err x	2.1×10^{-10}	8.8×10^{-14}	9.3×10^{-14}	8.0×10^{-13}	1.2×10^{-13}	1.8×10^{-10}
		res C	8.5×10^{-11}	4.8×10^{-16}	2.8×10^{-14}	4.4×10^{-15}	1.0×10^{-17}	
		res T	2.2×10^{-10}	6.4×10^{-14}	3.2×10^{-14}	3.5×10^{-13}	1.2×10^{-13}	
	2.2×10^{7}	res x	1.8×10^{-11}	2.2×10^{-15}	2.4×10^{-15}	3.2×10^{-13}	6.5×10^{-14}	8.1×10^{-13}
		err x	4.0×10^{-5}	4.1×10^{-9}	4.1×10^{-9}	4.8×10^{-9}	1.6×10^{-9}	2.6×10^{-8}
		res C	1.2×10^{-7}	2.7×10^{-16}	4.1×10^{-15}	1.9×10^{-14}	1.5×10^{-17}	
		res T	5.1×10^{-9}	3.3×10^{-14}	3.0×10^{-14}	6.5×10^{-13}	1.7×10^{-13}	
	7.7×10^{14}	res x	9.4×10^{-8}	2.1×10^{-15}	2.3×10^{-15}	3.2×10^{-13}	6.4×10^{-14}	2.5×10^{-13}
		err x	3.8×10^{-2}	5.5×10^{-3}	6.0×10^{-3}	2.0×10^{-2}	1.7×10^{-3}	1.7×10^{-3}
		res C	1.9×10^{-4}	3.3×10^{-16}	4.0×10^{-15}	9.6×10^{-15}	2.1×10^{-18}	
		res T	1.0×10^{5}	3.3×10^{-14}	3.0×10^{-14}	6.5×10^{-13}	1.7×10^{-13}	

Table 4 (*continued*)

Size	Cond		ALG 2.1	ALG 2.2	ALG 2.3	ALG 2.4	ALG 2.5	ALG 2.6
160	6.2×10^{3}	res x	2.6×10^{-12}	1.5×10^{-15}	6.3×10^{-15}	5.7×10^{-15}	1.7×10^{-13}	1.4×10^{-12}
		err x	1.3×10^{-10}	1.8×10^{-13}	3.9×10^{-13}	2.9×10^{-13}	4.6×10^{-13}	2.6×10^{-10}
		res C	4.3×10^{-12}	4.8×10^{-16}	2.9×10^{-14}	5.5×10^{-15}	4.8×10^{-18}	
		res T	6.6×10^{-11}	4.9×10^{-14}	2.6×10^{-14}	2.7×10^{-14}	3.5×10^{-13}	
	5.8×10^{5}	res x	2.4×10^{-14}	7.2×10^{-15}	7.2×10^{-15}	1.9×10^{-14}	7.0×10^{-14}	2.3×10^{-10}
		err x	2.2×10^{-9}	4.4×10^{-11}	7.2×10^{-11}	3.1×10^{-11}	5.1×10^{-11}	8.9×10^{-8}
		res C	1.1×10^{-12}	6.3×10^{-17}	1.6×10^{-15}	8.4×10^{-16}	2.4×10^{-17}	
		res T	1.2×10^{-13}	1.6×10^{-14}	1.8×10^{-14}	2.4×10^{-14}	1.4×10^{-13}	
	5.1×10^{14}	res x	5.2×10^{-1}	2.9×10^{-16}	7.7×10^{-16}	5.1×10^{-15}	2.4×10^{-14}	2.4×10^{-11}
		err x	1.1×10^{0}	1.4×10^{-2}	9.6×10^{-2}	5.2×10^{-1}	2.6×10^{-2}	1.0×10^{0}
		res C	5.0×10^{-1}	7.2×10^{-16}	2.2×10^{-15}	3.1×10^{-14}	9.4×10^{-18}	
		res T	2.3×10^{0}	1.7×10^{-13}	1.6×10^{-14}	3.5×10^{-14}	3.3×10^{-13}	

$$T = \begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix}, \tag{4.1}$$

or its small perturbations, for which $l = n/2$. Now, if the maximal length k of the look-ahead step is smaller than l the look-ahead Levinson algorithm will either break down or produce inaccurate results. On the other hand, in applications where it is unlikely to encounter many consecutive ill-conditioned leading submatrices the computed solution will often have sufficient accuracy. This was confirmed in our numerical experiments where we tested the behavior of the algorithms on matrices with only few consecutive ill-conditioned submatrices.

Results of numerical tests are given in Table 4. The table suggests that accuracy of the symmetric pivoting on the Cauchy matrix C in all but one case is comparable to that of unsymmetric pivoting. The single exception is the method ALG 2.1 which often looses about half of the number of accurate digits. This loss of accuracy is most probably connected with the linear fractional function which maps a unit circle onto a unit interval. However, the related method ALG 2.2 produces an accurate decomposition of C. Reasons for loss of accuracy in ALG 2.1 will have to be investigated further.

The accuracy of the decomposition of T as measured by res T is slightly lower than that of the decomposition of C. This is caused by the transformation of T into C, and possibly could be improved by a more careful implementation of FFT-like transformations. The residual error res$x$ in the solution vector is comparable to that in the decomposition of T. The relative error err$x$ in the solution is proportional to the condition number of T.

On the examples presented, transformation based algorithms ALG 2.2–2.5 compare favorably with the look-ahead algorithm ALG 2.6. In the examples included in Table 4, the matrices had several ill-conditioned intermediate submatrices. For that reason the length of the look-ahead step was chosen to be 16. With this choice of the look-ahead step, the algorithm produced the relative error err$x$ comparable to that produced by the transformation methods, in

Table 5
Cost of solving $Tx = b$

	Real multiplications	
	Max	Min
ALG 2.1	$5n^2$	$4n^2$
ALG 2.1	$5n^2$	$4n^2$
ALG 2.3	$16n^2$	$12n^2$
ALG 2.4	$20n^2$	$20n^2$
ALG 2.5	$2n^2$	$\frac{3}{2}n^2$
ALG 2.6	$\frac{1}{3}n^3$	$2n^2$

most of the cases. The exception was the case represented by the first row in the table when the matrix was a slightly perturbed version of the matrix in (4.1). While the transformation based methods produced accurate solutions, the length of the look-ahead step was too small for ALG 2.6 to find a well conditioned leading submatrix to stabilize the Levinson recursion.

The cost of the five transformation methods (excluding the cost of the transformations) can be derived from Table 1 where $r = 2$ and are summarized in Table 5. Note that except for ALG 2.5 all other transformations transform a real right hand side vector to a complex vector. Thus ALG 2.5 is the least costly among all five transformation algorithms discussed in this paper.

References

[1] A.W. Bojanczyk, R.P. Brent, F.R. de Hoog, D.R. Sweet, On the stability of the Bareiss and related Toeplitz factorization algorithms, SIAM J. Matrix Anal. Appl. 1 (1995) 40–57.
[2] T. Boros, A. Sayed, T. Kailath, Structured matrices and unconstrained rational interpolation problems, Linear Algebra Appl. (to appear).
[3] J.R. Bunch, Analysis of the diagonal pivoting method, SIAM J. Numer. Anal. 8 (1971) 656–680.
[4] J.K. Bunch, L. Kaufman, Some stable methods for calculating inertia and solving symmetric linear systems, Math. Comp. 31 (1977) 162–179.
[5] J. Bunch, L. Kaufman, B. Parlett, Decomposition of a symmetric matrix, Numerische Mathematik 27 (1976) 95–109.
[6] T. Chan, P. Hansen, A look-ahead Levinson algorithm for indefinite Toeplitz systems, SIAM J. Matrix Analysis Appl. 13 (2) (1992) 1079–1090.
[7] I. Gohberg, I. Koltracht, P. Lancaster, Efficient solution of linear systems of equations with recursive structure, Linear Algebra Appl. 80 (1986) 81–113.
[8] I. Gohberg, T. Kailath, I. Koltracht, P. Lancaster, Linear complexity parallel algorithms for linear systems of equations with recursive structure, Linear Algebra Appl. 88/99 (1987) 271–316.
[9] I. Gohberg, T. Kailath, V. Olshevsky, Fast Gaussian elimination with partial pivoting for matrices with displacement structure, Math. Comp. 64 (1995) 1557–1576.
[10] I. Gohberg, V. Olshevsky, Fast state space algorithms for matrix Nehari and Nehari-Takagi interpolation problems, Integral Equations and Operator Theory 20 (1) (1994) 44–83.
[11] M. Gutknecht, Stable row recurrences for the Padé table and generically superfast look-ahead solvers for non-Hermitian Toeplitz systems, Linear Algebra Appl. 188/189 (1993) 351–422.
[12] G. Heinig, Inversion of generalized Cauchy matrices and other classes of structured matrices, in: A. Bojanczyk, G. Cybenko (Eds.), The IMA Volumes in Mathematics and Its Applications, vol. 69, Springer, Berlin, 1992, pp. 63–82.
[13] G. Heinig, Inversion of Toeplitz-like matrices via generalized Cauchy matrices and rational interpolation, Systems and Network: Mathematical Theory and Applications, Akademie Verlag, Berlin, vol. 2, 1994, pp. 707–711.
[14] G. Heinig, Transformation approaches for fast and stable solution of Toeplitz systems and polynomial equations, Proceedings of the workshop on Recent Advances in Applied Mathematics, Kuwait University, 1996, pp. 223–238.
[15] G. Heinig, Solving Toeplitz systems after extension and transformation, Calcolo 33 (1996) 115–129.

[16] G. Heinig, A. Bojanczyk, Transformation techniques for Toeplitz and Toeplitz-plus-Hankel matrices, I: Transformations, Linear Algebra Appl. 254 (1997) 193–226.
[17] G. Heinig, K. Rost, Algebraic Methods for Toeplitz-Like Matrices and Operators, Birkhäuser Verlag, Basel, 1984.
[18] T. Kailath, V. Olshevsky, Diagonal pivoting for partially reconstructable Cauchy-like matrices, with applications to Toeplitz-like linear equations and to boundary rational matrix interpolation problems, Linear Algebra Appl. 254 (1997) 251–302.
[19] T. Kailath, A. Sayed, Fast algorithms for generalized displacement structures, in: H. Kimura, S. Kodama (Eds.), Recent Advances in Mathematical Theory of Systems, Control, Network, and Signal Processing II, Proceedings of the MTNS-91, Mita Press, Japan, 1992, pp. 27–32.
[20] T. Kailath, A. Sayed, Displacement structure: Theory and applications, SIAM Review 37 (3) (1995) 297–386.
[21] Ming Gu, Stable and efficient algorithms for structured systems of linear equations, SIAM J. Matrix Analysis (submitted) .
[22] V. Olshevsky (personal communication).
[23] D. Sweet, R. Brent, Error analysis of a fast partial pivoting method for structured matrices, in: T. Luk (Ed.), Advanced Signal Processing Algorithms, Proceedings of SPIE, vol. 2363, 1995, pp. 266–280.
[24] M. Tasche, Fast algorithms for discrete Chebyshev–Vandermonde transforms and applications, Numerical Algorithms 5 (1993) 453–464.
[25] C. Van Loan, Computational Framework for the Fast Fourier Transform, SIAM, Philadelphia, 1992.
[26] M. Vetterli, H.J. Nussbaumer, Simple FFT and DCT algorithms with reduced number of operations, Signal Processing 6 (1984) 267–278.