Pivoting and Backward Stability of Fast Algorithms for Solving Cauchy Linear Equations *

Tibor Boros
ArrayComm, Inc.
3141 *Zanker Road*
San Jose, CA 95134, *USA*

Thomas Kailath
Information Systems Laboratory
Department of Electrical Engineering
Stanford University, Stanford CA 94305-4055, *USA*

and

Vadim Olshevsky†
Department of Mathematics and Statistics
Georgia State University
Atlanta, GA 30303, *USA*

Abstract

Three fast $O(n^2)$ algorithms for solving *Cauchy linear systems* of equations are proposed. A rounding error analysis indicates that the backward stability of these new Cauchy solvers is similar to that of Gaussian elimination, thus suggesting to employ various *pivoting techniques* to achieve a favorable backward stability. It is shown that Cauchy structure allows one to achieve in $O(n^2)$ operations partial pivoting ordering of the rows and several other judicious orderings *in advance*, without actually performing the elimination. The analysis also shows that for the important class of *totally positive* Cauchy matrices it is advantageous to avoid pivoting, which yields a remarkable backward stability of the suggested algorithms.

It is shown that Vandermonde and Chebyshev-Vandermonde matrices can be efficiently transformed into Cauchy matrices, using Discrete Fourier, Cosine or Sine transforms. This allows us to use the proposed algorithms for Cauchy matrices for rapid and accurate solution of Vandermonde and Chebyshev-Vandermonde linear systems.

The analytical results are illustrated by computed examples.

AMS subject classification: 65F05, 65L20, 15A09, 15A23

Key words: Displacement structure, Cauchy matrix, Vandermonde matrix, fast algorithms, pivoting, rounding error analysis, backward stability, total positivity.

* This work was supported in part by NSF contracts CCR-962811, 9732355 and 0098222
† web: www.cs.gsu.edu/~matvro email: volshevsky@gsu.edu

1 Introduction

1.1 Related facts

Vandermonde and related matrices. Linear systems with Cauchy and Vandermonde coefficient matrices,

$$C(x_{1:n},\, y_{1:n}) = \begin{bmatrix} \frac{1}{x_1-y_1} & \cdots & \frac{1}{x_1-y_n} \\ \vdots & \ddots & \vdots \\ \frac{1}{x_n-y_1} & \cdots & \frac{1}{x_n-y_n} \end{bmatrix}, \qquad V(x_{1:n}) = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}, \tag{1.1}$$

are classical. They are encountered in many applied problems related to polynomial and rational function computations. Vandermonde and Cauchy matrices have many similar properties, among them one could mention the existence of explicit formulas for their determinants and inverses, see, e.g., [BKO99] and references therein. Along with many interesting algebraic properties, these matrices have several remarkable numerical properties, often allowing us much more accurate computations than those based on the use of general (structure-ignoring) algorithms, say Gaussian elimination with pivoting. At the same time, such favorable numerical properties are much better understood for Vandermonde and related matrices (see, for example [BP70], [TG81], [CF88], [Hig87], [Hig88], [Hig90], [RO91], [CR92], [CR93], [V93], [Ty94]), as compared to the analysis of numerical issues related to Cauchy matrices (see [GK90], [GK93]).

The Björck-Pereyra algorithm for Vandermonde systems. In particular, most of the above mentioned papers were devoted to the analysis of numerical properties and extensions of the now well-known Björck-Pereyra algorithm for solving Vandermonde linear systems [BP70], [GVL89]. This algorithm is based on the decomposition of the inverse of a Vandermonde matrix into a product of bidiagonal factors,

$$V^{-1}(x_{1:n}) = U_1^{-1} \,\ldots\, U_{n-1}^{-1}\, L_{n-1}^{-1} \,\ldots\, L_1^{-1}\,, \tag{1.2}$$

where

$$U_k^{-1} = \begin{bmatrix} \alpha_1^{(k)} & \beta_1^{(k)} & & \\ & \ddots & \ddots & \\ & & \ddots & \beta_{n-1}^{(k)} \\ & & & \alpha_n^{(k)} \end{bmatrix}, \qquad L_k^{-1} = \begin{bmatrix} \gamma_1^{(k)} & & & \\ \delta_2^{(k)} & \ddots & & \\ & \ddots & \ddots & \\ & & \delta_n^{(k)} & \gamma_n^{(k)} \end{bmatrix}. \tag{1.3}$$

This description allows one to solve the associated linear systems in only $O(n^2)$ operations, which is by an order of magnitude less than the complexity $O(n^3)$ of general (structure-ignoring) methods. Moreover, the algorithm requires only $O(n)$ locations of memory.

Remarkable accuracy for monotonically ordered nodes. It turns out that along with dramatic speed-up and savings in storage, the BP algorithm often produces a surprisingly high relative accuracy in the computed solution. N.J.Higham analyzed in [Hig87] the numerical performance of the BP algorithm and identified a class of Vandermonde matrices, viz., those for which the nodes can be strictly orderded,

$$0 < x_1 < x_2 < ... < x_n, \tag{1.4}$$

with a favorable forward error bound,

$$|a - \hat{a}| \le 5nuV(x_{1:n})^{-1}\,|f| + O(u^2), \tag{1.5}$$

for the solution $\hat{a}$ computed by the BP algorithm. Here u denotes the unit roundoff in the standard model of floating point arithmetic, and the operations of comparison, and of taking the absolute value of a matrix, are understood in a *componentwise* sense. It was further shown in [BKO99] that under the condition (1.4) the BP algorithm is not only forward, but also backward stable,

$$|\Delta V| \le 12n^2 uV(x_{1:n}) + O(u^2). \tag{1.6}$$

Here the computed solution $\hat{a}$ is the exact solution of a nearby system $(V(x_{1:n}) + \Delta V)\hat{a} = f$.

The Björck-Pereyra-type algorithm for Cauchy matrices. The above analytic error bounds indicate that the accuracy of the BP algorithm can be much higher than could be expected from the condition number of the

coefficient matrix. Such a high accuracy motivated several authors to extend the BP algorithm to several other classes of matrices, see, e.g., [TG81], [Hig88], [RO91]. All these generalizations were developed for Vandermonde related structures. In a recent paper [BKO99] a similar decomposition

$$C^{-1}(x_{1:n}, y_{1:n}) = U_1^{-1} \ldots U_{n-1}^{-1} \, L_{n-1}^{-1} \ldots L_1^{-1} \, , \tag{1.7}$$

was written down for Cauchy matrices, thus leading to a Björck-Pereyra-*type* algorithm which will be referred to as the BP-type algorithm. This algorithm requires $O(n^2)$ operations and $O(n)$ locations of memory. It was further shown in [BKO99] that the following configuration of the nodes,

$$y_n < ... < y_1 < x_1 < ... < x_n \, , \tag{1.8}$$

is an appropriate analog of (1.4) for Cauchy matrices, allowing to prove that the error bounds associated with the BP-type algorithm are entirely similar to (1.5) and (1.6), viz.,

$$|a - \hat{a}| \leq 5(2n+1)uC(x_{1:n}, y_{1:n})^{-1} \, |f| + O(u^2). \tag{1.9}$$

$$|\Delta C| \leq 20n(2n-1)uC(x_{1:n}, y_{1:n}) + O(u^2), \tag{1.10}$$

It is an interesting fact that the conditions (1.4) and (1.8) imply the *total positivity*[1] of $V(x_{1:n})$ and $C(x_{1:n}, y_{1:n})$, resp. Totally positive matrices are usually extremely ill-conditioned, so that the Gaussian elimination procedure often fails to compute even one correct digit in the computed solution. The bound (1.9) indicates that also in such "difficult" cases the BP-type algorithm can produce, for special right hand sides, all possible relative precision, see, e.g., [BKO99] for a discussion and numerical illustrations.

Limitations for non-totally-positive matrices. Of course, reordering of the nodes $\{x_k\}$ and $\{y_k\}$ is equivalent to row and column permutation of $C(x_{1:n}, y_{1:n})$, respectively. Therefore if the two sets of nodes are separated from each other,

$$y_k < x_j, \qquad 1 \leq k, j \leq n, \tag{1.11}$$

the remarkable error bounds (1.9) (1.10) suggest to reorder the nodes monotonically as in (1.8), and to apply the BP-type algorithm of [BKO99].

However, numerical experiments show that in the generic case, i.e., when (1.11) do not hold, the monotonic ordering does not guarantee a satisfactory accuracy, and the corresponding backward error of the *fast* BP-type algorithm of [BKO99] (and of the use of the explicit inversion formula as well) may be worse than that of the *slow* Gaussian elimination with pivoting. Employing other orderings of the nodes (for example, the partial pivoting ordering of the rows of $C(x_{1:n}, y_{1:n})$) does not seem to essentially improve the backward stability. A heuristic explanation for this fact can be drawn from the observation that the usual aim of pivoting is to reduce the size of the factors in the LU factorization of a matrix. Therefore it improves the stability properties of the Gaussian elimination procedure, for which the backward error bound involves the product $|\hat{L}| \cdot |\hat{U}|$ (computed factors). In contrast, an examination of the error analysis of the BP-type algorithm in [BKO99] indicates that the corresponding backward bound involves the quantity

$$|L_1| \cdot \ldots \cdot |L_{n-1}| \cdot |U_{n-1}| \cdot \ldots \cdot |U_1|, \tag{1.12}$$

which because of a non-cancellation property (i.e., $|M| \cdot |N| \geq |MN|$) can be much higher than the more attractive quantity $|L| \cdot |U|$. In the totally positive case (1.8), the entries of L_k and U_k in (1.12) are all nonnegative, thus allowing one to remove the moduli and to replace (1.12) by just $C(x_{1:n}, y_{1:n})$, cf. with the favorable bound (1.10). Unfortunately, in the general case, the *bidiagonal* structure of the L_k and U_k in (1.3) does not allow one to remove the moduli, and to easily deduce from it a satisfactory backward error bound. These limitations suggest that in one's attempts to design a backward stable Cauchy solver it can be better to develop new algorithms rather than to look for stabilizing techniques for the Björck-Pereyra-type algorithm.

1.2 Main results

New algorithms. In this paper we develop several alternatives to the BP-type algorithm, all based on factorizations

$$C(x_{1:n}, y_{1:n}) = L_1 \cdot \ldots \cdot L_{n-1} \cdot U_{n-1} \cdot \ldots \cdot U_1,$$

[1] Totally positive matrices are those for which the determinant of every submatrix is positive, see the monographs [GK50] and [K72].

where the factors L_k, U_k are of the form (diagonals with one nonzero row or column)

$$L_k = \begin{bmatrix} l_{11}^{(k)} & & & & \\ & \ddots & & & \\ & & l_{kk}^{(k)} & & \\ & & \vdots & \ddots & \\ & & l_{nk}^{(k)} & & l_{nn}^{(k)} \end{bmatrix}, \qquad U_k = \begin{bmatrix} u_{11}^{(k)} & & & & \\ & \ddots & & & \\ & & u_{kk}^{(k)} & \cdots & u_{kn}^{(k)} \\ & & & \ddots & \\ & & & & u_{nn}^{(k)} \end{bmatrix}. \tag{1.13}$$

Backward error bounds. We produce an error analysis for these new methods, obtaining backward and residual bounds also involving the quantity (1.12), but now with factors of the form (1.13). In contrast to the bidiagonal factors (1.3) of the BP-type algorithm of [BKO99], the sparsity pattern of the factors (1.13) immediately implies the equality

$$|L||U| = |L_1| \cdot ... \cdot |L_{n-1}||U_{n-1}| \cdot ... \cdot |U_1|,$$

resulting in pleasing backward bounds of the form

$$|\Delta C| \leq d_n u |L||U| + O(u^2), \tag{1.14}$$

for the new algorithms. Here the computed solution $\hat{a}$ is the exact solution of a nearby system $(C(x_{1:n}, y_{1:n}) + \Delta C)\hat{a} = f$. These bounds are similar to the well-known bounds for Gaussian elimination,

$$|\Delta R| \leq 2\gamma_n |\hat{L}||\hat{U}|, \qquad \text{where} \qquad \gamma_n = \frac{nu}{1-nu}, \tag{1.15}$$

see, e.g., p. 175 in [Hig96]. Here $\hat{L}$ and $\hat{U}$ denote the *computed* triangular factors.

Different pivoting techniques. This resemblance between the backward error bounds (1.14) and (1.15) suggests to employ different row and column pivoting techniques to reduce the size of $|L|$ and $|U|$ and to stabilize the new algorithms for Cauchy systems.

There is a variety of possible pivoting techniques that can be incorporated into the new algorithms without increasing their $O(n^2)$ complexity, including partial row and partial column pivoting, Gu's pivoting (a variation of complete pivoting for Cauchy-like matrices [Gu95]), and others.

Total positivity. There are classes of matrices for which it is advantageous not to pivot. For example, for totally positive matrices, the exact triangular factors have only positive entries. C.De Boor and A.Pinkus pointed out in [DBP77] that if the entries of the computed factors, $\hat{L}$ and $\hat{U}$ remain nonnegative, than the backward error of Gaussian elimination *without pivoting* is pleasantly small,

$$|\Delta R| \leq 3\gamma_n R, \tag{1.16}$$

see also p.176 in [Hig96]. It turns out that the same recommendation to avoid pivoting can be made for the fast Cauchy solvers proposed here, and moreover because the corresponding error bounds (1.14) involve the *exact* triangular factors, in the case of total positivity we have $|L| \cdot |U| = C(x_{1:n}, y_{1:n})$, implying that a remarkable backward stability of the same form (1.16) is guaranteed for the new fast algorithms without any additional assumptions on the *computed* triangular factors.

1.3 Outline of the paper

It is well-known that for structured matrices the Gaussian elimination procedure can be speeded-up. In the next section we exploit the *displacement structure* of Cauchy matrices to specify two such algorithms. Then in the section 3 we exploit the quasi-Cauchy structure of the Schur complements of $C(x_{1:n}, y_{1:n})$ to derive one more algorithm for solving Cauchy linear equations. Then in section 4 we perform a rounding error analysis for these algorithms, obtaining backward and residual bounds similar to those for Gaussian elimination. This analogy allows us in Section 5 to carry over the stabilizing techniques known for Gaussian elimination to the new Cauchy solvers. The numerical properties of new algorithms are illustrated in section 6 by a variety of examples. Then in section 7 we show how Vandermonde and Chebyshev-Vandermonde matrices can be efficiently transformed into Cauchy matrices by using Discrete Fourier, Cosine or Sine transforms, thus allowing us to use the proposed Cauchy solvers for the rapid solution of Vandermonde and Chebyshev-Vandermonde linear systems.

2 Schur-type algorithms for Cauchy matrices.

2.1. Displacement structure. By now, the *displacement structure* approach is well-known to be useful in the development of various types of fast algorithms for structured matrices, including Toeplitz, Hankel, Toeplitz-plus-Hankel, Vandermonde, Chebyshev-Vandermonde, and several others. This approach is based on a convenient way to capture each of the above particular structures by specifying suitable *displacement operators* $\nabla(\cdot) : \mathbf{C}^{n\times n} \to \mathbf{C}^{n\times n}$. In this paper we shall use operators of the form

$$\nabla_{\{F,A\}}(R) = FR - RA, \tag{2.1}$$

for various choices of (sparse) matrices $\{F, A\}$. Let $\alpha := \mathrm{rank}\nabla_{\{F,A\}}(R)$, then one can factor

$$\nabla_{\{F,A\}}(R) = FR - RA = GB^T, \tag{2.2}$$

where both matrices on the right-hand side of (2.2) have only α columns each : $G, B \in \mathbf{C}^{n\times\alpha}$. The number $\alpha = \mathrm{rank}\nabla_{\{F,A\}}(R)$ is called $\{F, A\}$-*displacement rank* of R, and the pair $\{G, B\}$ is called $\{F, A\}$-*generator* of R. The displacement rank measures the complexity of R, because all its n^2 entries are described by a smaller number $2\alpha n$ entries of its generator $\{G, B\}$. We refer to surveys [KS95], [HR84], [O97], [D01] for more complete information on displacement, different approaches and further references. Here we restrict ourselves only with Cauchy matrices. The next lemma recalls their displacement structure.

Lemma 2.1 *Let $\nabla_{\{F,A\}}(\cdot) : \mathbf{C}^{n\times n} \to \mathbf{C}^{n\times n}$ be defined by (2.1) with*

$$F = D_x = \mathrm{diag}\{x_1, ..., x_n\}, \qquad A = D_y = \mathrm{diag}\{y_1, ..., y_n\}. \tag{2.3}$$

Then the Cauchy matrix has displacement,

$$\nabla_{\{F,A\}}(C(x_{1:n}, y_{1:n})) = \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}^T \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}. \tag{2.4}$$

2.2. Structure of Schur complements. Let $R = R_1$ in (2.2) be partitioned : $R_1 = \begin{bmatrix} d_1 & u_1 \\ l_1 & R_{22}^{(1)} \end{bmatrix}$, and denote by $R_2 = R_{22}^{(1)} - \frac{1}{d_1} l_1 u_1$ its Schur complement. Then if the matrices F_1 and A_1 in (2.2) are lower and upper triangular, resp., say,

$$F_1 = \begin{bmatrix} f_1 & 0 \\ * & F_2 \end{bmatrix}, \qquad A_1 = \begin{bmatrix} a_1 & * \\ 0 & A_2 \end{bmatrix}, \tag{2.5}$$

then the $\{F_2, A_2\}$-*displacement rank* of the Schur complement is less than or equal to α, so that we can write

$$F_2 \cdot R_2 - R_2 \cdot A_2 = G_2 \cdot B_2^T \qquad \text{with} \qquad \text{some} \qquad G_2, B_2 \in \mathbf{C}^{(n-1)\times\alpha}. \tag{2.6}$$

The latter fact was first observed by L.A.Sakhnovich in [S76] (see also [S86]) and by M.Morf in [M80]. See, e.g., [KS95] and the references therein for various formulas showing how to run the *generator recursion* $\{G_1, B_1\} \to \{G_2, B_2\}$. The next lemma (cf. with [GO94a], [GKO95], [KO95b]) provides one particular form for the generator recursion.

Lemma 2.2 *Let $R_1 = \begin{bmatrix} d_1 & u_1 \\ l_1 & R_{22}^{(1)} \end{bmatrix}$ satisfy the displacement equation (2.2) with triangular F_1, A_1 partitioned as in (2.5). If the* (1,1) *entry d_1 of R_1 is nonzero, then the Schur complement $R_2 = R_{22}^{(1)} - \frac{1}{d_1} l_1 u_1$ satisfies the displacement equation (2.6) with*

$$\begin{bmatrix} 0 \\ G_2 \end{bmatrix} = G_1 - \begin{bmatrix} 1 \\ \frac{1}{d_1} l_1 \end{bmatrix} \cdot g_1, \qquad \begin{bmatrix} 0 & B_2 \end{bmatrix} = B_1 - \begin{bmatrix} 1 & \frac{1}{d_1} u_1^T \end{bmatrix} \cdot b_1, \tag{2.7}$$

where g_1 and b_1 denote the top rows of G_1 and B_1, resp.

The standard Gaussian elimination procedure computes the LU factorization using $O(n^3)$ flop. The above lemma allows us to exploit the structure of R_1 to compute this factorization in only $O(n^2)$ flops, as described next.

2.3. Speed-up of the Gaussian elimination procedure. The first step of Gaussian elimination procedure applied to a matrix R_1 is described by

$$R_1 = \begin{bmatrix} d_1 & u_1 \\ l_1 & R_{22}^{(1)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1}{d_1} l_1 & I \end{bmatrix} \cdot \begin{bmatrix} d_1 & u_1 \\ 0 & R_2 \end{bmatrix}, \tag{2.8}$$

where $R_2 = R_{22}^{(1)} - \frac{1}{d_1} l_l u_1$ is Schur complement of (1,1) entry d_1 in the matrix R_1. This step provides the first column $\begin{bmatrix} 1 \\ \frac{1}{d_1} l_1 \end{bmatrix}$ of L and the first row $\begin{bmatrix} d_1 & u_1 \end{bmatrix}$ in the LU factorization of R_1. Proceeding with R_2 similarly, after $n-1$ steps one obtains the whole LU factorization.

Algorithms that exploit the displacement structure of a matrix to speed-up the Gaussian elimination procedure are called *Schur-type algorithms*, because the classical Schur algorithm [S17] was shown (see, e.g., [LAK86], [KS95]) to belong to the class. Instead of computing the $(n-k+1)^2$ *entries* of the Schur complement R_k, one has to compute only $\alpha(n-k+1)$ entries of its $\{F_k, A_k\}$-generator $\{G_k, B_k\}$, which requires much less computations. To run the generator recursion (2.7) as well as to write down the corresponding entries of the L and U factors, one needs only to specify how to recover the first row and column of R_k from its generator $\{G_k, B_k\}$. For a Schur complement $R_k = \begin{bmatrix} r_{ij}^{(k)} \end{bmatrix}_{k \le i,j \le n}$ of a Cauchy matrix this is easy to do :

$$r_{i,j}^{(k)} = \frac{g_i^{(k)} \cdot b_1^{(k)}}{x_i - y_j}, \tag{2.9}$$

where $\{ \begin{bmatrix} g_k^{(k)} & \dots & g_n^{(k)} \end{bmatrix}^T, \begin{bmatrix} b_k^{(k)} & \dots & b_n^{(k)} \end{bmatrix} \}$ designates the corresponding generator.

2.4. Triangular factorization for Cauchy matrices. In the rest of this section we formulate two Schur-type algorithms for $C(x_{1:n}, y_{1:n})$. The first version is an immediate implementation of (2.7) and (2.9), and its MATLAB code is given next.

Algorithm 2.3 (Schur-type-Cauchy) *Complexity:* $6n^2 + O(n)$ *flops.*

```
function [L,U] = Cauchy_GS (x,y)
% triangular factorization of the Cauchy matrix C(x_{1:n}, y_{1:n}).
        n=max(size(x)); L=eye(n,n); U=zeros(n,n);
        % Generators %
        g=ones(n,1); b=ones(n,1);
        for k=1:n-1
        %   Computing the k-th row of L and U   %
            U(k,k) = ( g(k) * b(k) ) / ( x(k) - y(k) );
            for j=k+1:n
                L(j,k) = (( g(j) * b(k) ) / ( x(j) - y(k) )) / U(k,k);
                U(k,j) = (( g(k) * b(j) ) / ( x(k) - y(j) ));
            end
        %   Computing the new generators   %
            for j=k+1:n
                g(j) = g(j) - g(k) * L(j,k);
                b(j) = b(j) - b(k) * U(k,j) / U(k,k);
            end
        end
        U(n,n) = ( g(n) * b(n) ) / ( x(n)-y(n) );
return
```

2.5. Direct generator recursion. In fact the algorithm 2.3 is valid for the more general class of Cauchy-like matrices, see, e.g., [GKO95] [KO95b] for details and applications. However for the special case of ordinary Cauchy matrices we can exploit the fact that the corresponding displacement rank is equal to one, to formulate a more specific Schur-type algorithm, based on the next Lemma.

Lemma 2.4 *Let* R_k *in*

$$\mathrm{diag}(x_k, ..., x_n) R_k - R_k \mathrm{diag}(y_k, ..., y_n) = G_k B_k = \begin{bmatrix} g_k^{(k)} & \dots & g_n^{(k)} \end{bmatrix}^T \begin{bmatrix} b_k^{(k)} & \dots & b_n^{(k)} \end{bmatrix}$$

be the successive Schur complements of the Cauchy matrix $C(x_{1:n}, y_{1:n})$. Then the generator recursion (2.7) can be specialized to

$$\begin{bmatrix} g_{k+1}^{(k+1)} \\ \vdots \\ g_n^{(k+1)} \end{bmatrix} = \begin{bmatrix} \frac{x_{k+1}-x_k}{x_{k+1}-y_k} g_{k+1}^{(k)} \\ \vdots \\ \frac{x_n-x_k}{x_n-y_k} g_n^{(k)} \end{bmatrix}, \qquad \begin{bmatrix} b_{k+1}^{(k+1)} & \cdots & b_n^{(k+1)} \end{bmatrix} = \begin{bmatrix} \frac{y_{k+1}-y_k}{y_{k+1}-x_k} b_n^{(k)} & \cdots & \frac{y_n-y_k}{y_n-x_k} b_n^{(k)} \end{bmatrix}. \tag{2.10}$$

The nonzero entries of the factors in $C(x_{1:n}, y_{1:n}) = LDU$ are given by $d_k = x_k - y_k$ and

$$\begin{bmatrix} l_{k,k} \\ \vdots \\ l_{n,k} \end{bmatrix} = \begin{bmatrix} \frac{1}{x_k-y_k} g_k^{(k)} \\ \vdots \\ \frac{1}{x_n-y_k} g_n^{(k)} \end{bmatrix}, \qquad \begin{bmatrix} u_{k,k} & \cdots & u_{k,n} \end{bmatrix} = \begin{bmatrix} \frac{1}{x_k-y_k} b_k^{(k)} & \cdots \frac{1}{x_k-y_n} b_n^{(k)} \end{bmatrix} \tag{2.11}$$

The following MATLAB code implements the algorithm based on Lemma 2.4.

Algorithm 2.5 (Schur-type-direct-Cauchy) *Complexity: $6n^2 + O(n)$ flops.*

```
function [L,D,U]= Cauchy_GS_d( x, y)
% triangular factorization of the Cauchy matrix C(x_{1:n}, y_{1:n}).
         n=max(size(x)); L=eye(n,n); D=zeros(n,n); U=eye(n,n);
         % Generators %
         g=ones(n,1); b=ones(n,1);
         for k=1:n-1
              %   Computing the k-th row of L and U   %
             D(k,k) = (x(k)-y(k));
             for j=k:n
                 L(j,k) = g(j)/(x(j) - y(k));
                 U(k,j) = b(j)/(x(k) - y(j));
             end
         %   Computing the new generators   %
             for j=k+1:n
                 g(j) = g(j) *(x(j)-x(k))/(x(j)-y(k));
                 b(j) = b(j) *(y(j)-y(k))/(y(j)-x(k));
             end
         end
         D(n,n) = 1/(x(n)-y(n));
         L(n,n) = g(n);
         U(n,n) = b(n);
return;
```

The fast algorithms 2.3 and 2.5 require $O(n^2)$ locations of memory to store the triangular factors. An algorithm with $O(n)$ storage is described next.

3 Exploiting quasi-Cauchy structure

The concept of *displacement structure* was initiated by the paper [KKM79], where it was first applied to study Toeplitz matrices, using a displacement operator of the form $\nabla_{\{Z,Z^T\}}(R) = R - Z \cdot R \cdot Z^T$, where Z is the lower shift matrix. In this section we make a connection with [LAK86], where the fact that Toeplitz matrices belong to the more general class of matrices with $\nabla_{\{Z,Z^T\}}$-displacement rank 2, was used to introduce the name *quasi-Toeplitz* for such matrices. It is shown that any quasi-Toeplitz matrix R can be represented as a product of three Toeplitz matrices,

$$R = LTU, \tag{3.1}$$

where L is lower and U is upper triangular Toeplitz matrices. Patterning ourselves upon the above definition, and taking Lemma 2.1 as a starting point, we shall refer to matrices with $\{D_x, D_y\}$-displacement rank 1 as *quasi-Cauchy* matrices. The next simple lemma is an analog of (3.1).

Lemma 3.1 *Let D_x and D_y be defined by (2.4). Then the unique solution of the equation*

$$D_x \cdot R - R \cdot D_y = \begin{bmatrix} g_1 & \dots & g_n \end{bmatrix}^T \cdot \begin{bmatrix} b_1 & \dots & b_n \end{bmatrix} \tag{3.2}$$

is given by

$$R = \mathrm{diag}(g_1, g_2, ..., g_n) \cdot C(x_{1:n}, y_{1:n}) \cdot \mathrm{diag}(b_1, b_2, ..., b_n). \tag{3.3}$$

Lemma 3.1 allows us to obtain below an explicit factorization formula for Cauchy matrices. Indeed, by Lemma 2.2 its Schur complement $R_2 = R_{22} - \frac{1}{d_1} l_1 u_1$ in

$$C(x_{1:n},\ y_{1:n}) = \begin{bmatrix} d_1 & u_1 \\ l_1 & R_{22} \end{bmatrix} = \begin{bmatrix} d_1 & 0 \\ l_1 & I \end{bmatrix} \begin{bmatrix} \frac{1}{d_1} & 0 \\ 0 & R_2 \end{bmatrix} \begin{bmatrix} d_1 & u_1 \\ 0 & I \end{bmatrix}$$

also has displacement rank 1, so by Lemma 3.1 its Cauchy structure can be recovered by dropping diagonal factors as shown next.

Lemma 3.2 *The Cauchy matrix and its inverse can be factored as*

$$C(x_{1:n},\ y_{1:n}) = L_1 \dots L_{n-1}\, D\, U_{n-1} \dots U_1, \tag{3.4}$$

where $D = \mathrm{diag}((x_1 - y_1), ..., (x_n - y_n))$, and

$$L_k = \begin{bmatrix} I_{k-1} & & \\ & \frac{1}{x_k - y_k} & \\ & & \ddots \\ & & \frac{1}{x_n - x_k} \end{bmatrix} \begin{bmatrix} I_{k-1} & & \\ & 1 & \\ & 1 & 1 \\ & \vdots & & \ddots \\ & 1 & & & 1 \end{bmatrix} \begin{bmatrix} I_{k-1} & & \\ & 1 & \\ & & x_{k+1} - x_k \\ & & & \ddots \\ & & & & x_n - x_k \end{bmatrix},$$

$$U_k = \begin{bmatrix} I_{k-1} & & \\ & 1 & \\ & & y_k - y_{k+1} \\ & & & \ddots \\ & & & & y_k - y_n \end{bmatrix} \begin{bmatrix} I_{k-1} & & & & \\ & 1 & 1 & \dots & 1 \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix} \begin{bmatrix} I_{k-1} & & \\ & \frac{1}{x_k - y_{k+1}} & \\ & & \ddots \\ & & & \frac{1}{x_k - y_n} \end{bmatrix}.$$

The representation for the inverse matrix $C(x_{1:n}, y_{1:n})^{-1}$ obtained from the above leads to the following algorithm for solving $C(x_{1:n}, y_{1:n})a = f$.

Algorithm 3.3 (quasi-Cauchy) [2]

Complexity: $6n^2 + O(n)$ *flops.*

```
function a = Cauchy_quasi (x,y,f)
% Solving a Cauchy linear system C(x_{1:n},y_{1:n})a = f.
         n=max(size(x)); a=f;
         fork=1:n-1
            for j=k:n
                a(j) = a(j) * ( x(j) - y(k) );
            end
            for j=k+1:n
                a(j) = a(j) - a(k);
            end
            for j=k+1:n
                a(j) = a(j) / ( x(j) - x(k));
            end
         end
         for k=1:n-1
            a(k) = a(k) / ( x(k) - y(k) );
         end
         a(n) = a(n) * ( x(n) - y(n) );
         for k = n-1:-1:1
            for j=k+1:n
```

[2] Algorithm 3.3 has lower complexity and better error bounds than its earlier variant called Cauchy-2 in [BKO94].

```
            a(j) = a(j) /(y(k)-y(j));
        end
        tmp = 0;
        for j=n:-1:k+1
            tmp = tmp + a(j);
        end
        a(k) = a(k) -tmp;
        for j=k:n
            a(j) = a(j) * ( x(k) - y(j) );
        end
      end
return
```

The reader should note that the multiplication of the central factor of U_k^{-1} by a vector is performed by accumulation of the inner product from the last to the first entry; this order is influenced by the error analysis in the next section.

4 Rounding error analyses

4.1 Stability of the algorithm 2.3.

The algorithm 2.3 is a special case of the more general GKO algorithm [GKO95], which is applicable to the wider class of Cauchy-like matrices. A normwise rounding error analysis for the GKO algorithm appeared in [SB95]. Along with the usual factor $\|L\|\|U\|$ (cf. with (1.15)) the backward error bound of [SB95] involves also a so-called *generator growth* factor of the form

$$\| \operatorname{diag}(\frac{|g_k^{(k)}|\ |b_k^{(k)}|}{g_k^{(k)}\ b_k^{(k)}}) \| \tag{4.1}$$

In the context of [GKO95], [SB95] the quantities $g_k^{(k)}$ and $b_k^{(k)}$ were vectors of size equal to the displacement rank of R; so if the quantity in (4.1) is large, then the backward stability of of the GKO algorithm could be less favorable than that of Gaussian elimination. However, the ordinary Cauchy matrices, considered in the present paper, all have displacement rank 1, so that the constant in (4.1) is unity, suggesting that the backward stability of the algorithm 2.3 is related to that of Gaussian elimination without pivoting.

4.2 Stability of the algorithm 2.5

Theorem 4.1 *Assume that the algorithm 2.5 (Schur-type-direct-Cauchy algorithm) is carried out in floating point arithmetic with a unit roundoff u, and that no overflows were encountered during the computation. Then the computed factors $\hat{L}, \hat{D}, \hat{U}$ of $C(x_{1:n}, y_{1:n}) = LDU$ satisfy $\hat{L} = L + \Delta L$, $\hat{D} = D + \Delta D$, $\hat{U} = U + \Delta U$, where*

$$|\Delta L| \le \gamma_{4n-2}|L|, \qquad |\Delta D| \le u|D|, \qquad |\Delta U| \le \gamma_{4n-2}|U|, \qquad (\gamma_n = \frac{nu}{1-nu}). \tag{4.2}$$

Furthermore, if this computed triangular factorization is used to solve the associated linear system $C(x_{1:n}, y_{1:n})a = f$ then the computed solution $\hat{a}$ solves a nearby system $(C(x_{1:n}, y_{1:n}) + \Delta C)\hat{a} = f$ with

$$|\Delta C| \le ((10n-2)u + O(u^2))|L||DU|, \tag{4.3}$$

and

$$|C(x_{1:n}, y_{1:n})\hat{a} - f| \le ((10n-2)u + O(u^2))|L|\,|DU|\,|\hat{a}|, \tag{4.4}$$

Proof.

Error in triangular factorization. A straightforward error analysis for the direct generator recursion (2.10) and for (2.11) implies that the computed and the exact columns of L are related by

$$\begin{bmatrix} \hat{l}_{kk} \\ \vdots \\ \hat{l}_{nk} \end{bmatrix} = fl\{ \begin{bmatrix} \frac{1}{x_k - y_k} g_k^{(k)} \\ \vdots \\ \frac{1}{x_n - y_k} g_n^{(k)} \end{bmatrix} \} = \begin{bmatrix} l_{kk}\delta_k^{(k)} \\ \vdots \\ l_{nk}\delta_n^{(k)} \end{bmatrix}$$

so that the componentwise error is nicely bounded :

$$\frac{(1-u)^{3k-2}}{(1+u)^k} \leq \delta_j^{(k)} \leq \frac{(1+u)^{3k-2}}{(1-u)^k}$$

This and similar arguments for U lead to the favorable bounds (4.2).

Error in the computed solution. Standard error analysis, see, e.g., p. 154 in [Hig96], for solving a linear system $\hat{L}\hat{D}\hat{U}a = f$ by forward and backsubstitution yields that the computed solution $\hat{a}$ satisfies

$$(\hat{L} + \hat{\Delta L})(\hat{D} + \hat{\Delta D})(\hat{U} + \hat{\Delta U})\hat{a} = f,$$

where

$$|\hat{\Delta L}| \leq \gamma_n |\hat{L}|, \qquad |\hat{\Delta D}| \leq u|\hat{D}|, \qquad |\hat{\Delta U}| \leq \gamma_n |\hat{U}|. \tag{4.5}$$

The pleasant bound in (4.3) is now deduced from (4.5) and (4.2).

4.3 Stability of the algorithm 3.3

Theorem 4.2 *Assume that the algorithm 3.3 (quasi-Cauchy algorithm) is carried out in floating point arithmetic with unit roundoff u, and that no overflows were encountered during the computation. Then the computed solution $\hat{a}$ solves a nearby system*

$$(C(x_{1:n}, y_{1:n}) + \Delta C)\hat{a} = (L + \Delta L)(D + \Delta D)(U + \Delta U)\hat{a} = f$$

with

$$|C(x_{1:n}, y_{1:n})\hat{a} - f| \leq ((n^2 + 11n - 10)u + O(u^2))|L|\,|DU|\,|\hat{a}|, \tag{4.6}$$

$$|\Delta C| \leq ((n^2 + 11n - 10)u + O(u^2))|L||DU|, \tag{4.7}$$

and

$$|\Delta L| \leq \gamma_{7(n-1)}|L|, \qquad |\Delta D| \leq \gamma_2 |D|, \qquad |\Delta U| \leq \gamma_{n^2+4n-5}|U|, \qquad (\gamma_n = \frac{nu}{1-nu}). \tag{4.8}$$

Proof. Let us recall that algorithm 3.3 solves a Cauchy linear system by computing

$$a = C(x_{1:n}, y_{1:n})^{-1} f = U_1^{-1} \cdot \ldots \cdot U_{n-1}^{-1} \cdot D^{-1} \cdot L_{n-1}^{-1} \cdot \ldots \cdot L_1^{-1} f, \tag{4.9}$$

where the $\{L_i, U_i\}$ are given by (3.4). The proof for (4.7) will be obtained in the following steps.

(i) First we apply the standard error analysis for each elementary matrix-vector multiplication in (4.9) to show that the computed solution $\hat{a}$ satisfies

$$\hat{a} = (U_1^{-1} * \delta U_1) \cdot \ldots \cdot (U_{n-1}^{-1} * \delta U_1) \cdot (D^{-1} * \delta D) \cdot (L_{n-1}^{-1} * \delta L_{n-1}) \cdot \ldots \cdot (L_1^{-1} * \delta L_1) f, \tag{4.10}$$

where the asterisk $*$ denotes the Hadamard (or componentwise) product.

(ii) Next, the obtained bounds for $\delta L_k, \delta D, \delta U_k$ will be used to deduce further bounds for $\Delta L_k, \Delta D, \Delta U_k$, defined by

$$(L_k^{-1} * \delta L_k)^{-1} = L_k + \Delta L_k, \qquad (D^{-1} * \delta D)^{-1} = D + \Delta D, \qquad (U_k^{-1} * \delta U_k)^{-1} = U_k + \Delta U_k.$$

(iii) Finally, inverting (4.10) we shall obtain

$$\hat{a} = (L_1 + \Delta L_1) \cdot \ldots \cdot (L_{n-1} + \Delta L_1) \cdot (D + \Delta D) \cdot (U_{n-1} + \delta U_{n-1}) \cdot \ldots \cdot (U_1 + \Delta U_1) f, \tag{4.11}$$

which will lead to the desired bounds in (4.8) and in (4.7).

We start with (i) and (ii).

Lower triangular factors. We start with obtaining bounds for δL_k and ΔL in (4.10) and (4.11). The sparse nature of these matrices, see, e.g., Lemma 3.2, implies the following pleasing bound for the (i,j) entry of δL_k :

$$\frac{(1-u)^4}{(1+u)} \leq (\delta L_k)_{i,j} \leq \frac{(1+u)^4}{(1-u)}. \tag{4.12}$$

Moreover, even smaller bounds hold for the (k,k) entry :

$$(1-u)^2 \leq (\delta L_k)_{k,k} \leq (1+u)^2. \tag{4.13}$$

Since the L_k all have exactly the same sparsity pattern as their inverses, (4.12) and (4.13) imply that

$$|\Delta L_k| \leq (\frac{(1+u)^4}{(1-u)^3} - 1)|L_k|. \tag{4.14}$$

Diagonal factor. The simple structure of D immediately implies that

$$|\Delta D| \leq (\frac{(1+u)}{(1-u)} - 1)|D|. \tag{4.15}$$

Upper triangular factors. The analysis shows that for $i \neq k$, (i.e. excluding the entries of the k-th row) we have

$$\frac{(1-u)^3}{(1+u)} \leq (\delta U_k)_{i,j} \leq \frac{(1+u)^4}{(1-u)}. \tag{4.16}$$

If the inner product corresponding to the k-th row of U_k is evaluated from the last to the k-th entry, then the error in the (k,j) entry is bounded by

$$(1-u)^{j-k+3} \leq (\delta U_k)_{k,j} \leq (1+u)^{j-k+3}. \tag{4.17}$$

In particular the error in the (k,k) entry is bounded by

$$(1-u)^3 \leq (\delta U_k)_{k,k} \leq (1+u)^3. \tag{4.18}$$

Again, since U_k have exactly the same sparsity pattern as their inverses, (4.16), (4.17) and (4.18) imply that

$$|\Delta U_k| \leq (\frac{(1+u)^{n+2}}{(1-u)^3} - 1)|U_k|. \tag{4.19}$$

We are now ready to turn to (iii). To prove (4.8) we shall use the following easily verified fact (see, e.g., [Hig96], p.80) : let $|\Delta X_k| \leq \delta |X_k|$ for $k = 1, 2, ..., m$, then

$$|\prod_{k=1}^{m}(X_k + \Delta X_k) - \prod_{k=1}^{m} X_k| \leq ((1+\delta)^m - 1)\prod_{k=1}^{m}|X_k|. \tag{4.20}$$

This and (4.14) imply that

$$|\Delta L| = |(L_1 + \Delta L_1)\cdot ... \cdot (L_{n-1} + \Delta L_{n-1}) - L_1 \cdot ... \cdot L_{n-1}| \leq (\frac{(1+u)^{4(n-1)}}{(1-u)^{3(n-1)}} - 1)|L_1|\cdot ... \cdot |L_{n-1}|. \tag{4.21}$$

The sparsity pattern of L_k ($k = 1, 2, ..., n-1$) allows us to remove the moduli in the product on the right-hand side of (4.21), implying the first bound in (4.8). The third bound in (4.8) is deduced from (4.19) and (4.20) analogously. The second bound in (4.8) follows from (4.15) easily. Finally, the bound in (4.7) is deduced from (4.8) (cf. (4.5)).

5 Pivoting

In the previous section we established that the backward stability of all the fast Cauchy solvers suggested in the present paper is related to that of Gaussian elimination. This analogy will allow us to carry over the stabilizing techniques of Gaussian elimination to the new Cauchy solvers. First however, we identify the case when no pivoting is necessary.

5.1 Totally positive matrices

If we assume that

$$y_n < ... < y_1 < x_1 < ... < x_n \ , \tag{5.1}$$

i.e. the matrix $C(x_{1:n}, y_{1:n})$ is totally positive, so that all the entries of the exact factors L and U are positive [GK50]. In this case theorems 4.1 and 4.2 imply that the algorithms 2.5 and 3.3 produce a favorable small backward error.

Corollary 5.1 *Assume that condition (5.1) holds, i.e. that $C(x_{1:n}, y_{1:n})$ is totally positive, and assume that the algorithms 2.5 (Schur-type-direct-Cauchy algorithm) and 3.3 (quasi-Cauchy algorithm) are performed in the floating point arithmetic with unit roundoff u, and that no overflows were encountered during the computation.*

If the triangular factorization of the Schur-type-direct-Cauchy algorithm is used to solve the associated linear system, then the computed solution $\hat{a}$ solves a nearby system

$$(C(x_{1:n}, y_{1:n}) + \Delta C)\hat{a} = f,$$

with

$$|\Delta C| \leq ((10n - 3)u + O(u^2))C(x_{1:n}, y_{1:n}). \tag{5.2}$$

The analogous backward bound for the quasi-Cauchy algorithm is

$$|\Delta C| \leq ((n^2 + 11n - 10)u + O(u^2))C(x_{1:n}, y_{1:n}). \tag{5.3}$$

The above results show that the backward stability of the fast algorithms 2.5, 3.3 for totally positive Cauchy matrices is even more favorable than that of the slow Gaussian elimination procedure, see (1.16). Indeed the difference is that the bound (1.16) is valid only for the case when the entries of the *computed factors* $\hat{L}$ and $\hat{U}$ remain positive (which is usually not the case with ill-conditioned matrices), whereas the favorable bounds in the two above corollaries hold while there are no overflows. For example, for the Hilbert matrix $H = \left[\ \frac{1}{i+j-1} \ \right]$ the condition number $k_2(H)$ grows exponentially with the size, so already for small n we have $k_2(H) > q(n)\frac{1}{u}$. Here $q(n)$ is a polynomial of small degree in n. Then in accordance with [W68] the matrix H will likely lose during the elimination not only its total positivity, but also the weaker property of being positive definite. Correspondingly, the single precision LAPACK routine **SPOSV** for Cholesky factorization, when applied to the Hilbert matrix, exits with an error flag already for $n = 9$, warning that the entries of $\hat{L}$, $\hat{U}$ became negative, so the pleasing backward bound (1.16) is no longer valid for Gaussian elimination. In contrast, the favorable bounds (5.2), (5.3) are valid for higher sizes, as long as there are no overflows.

5.2 General case. Predictive pivoting techniques

Here we assume that the two sets of nodes $\{x_k\}$ and $\{y_k\}$ are not separated from each other. The similarity of the backward bounds (1.15) for Gaussian elimination and of (4.2), (4.7) for the new Cauchy solvers suggests to use the same pivoting techniques for preventing instability. More precisely, any row or column reordering that reduces the size of $|L||U|$ appearing in the bounds (4.2), (4.7) will stabilize the numerical performance of the algorithms 2.5, 3.3. Moreover, the normwise error analysis of [SB95] for the algorithm 2.3, reviewed at the beginning of section 4, also indicates that the pivoting will enhance the accuracy of the algorithm 2.3.

Here we should note that the partial pivoting technique can be directly incorporated into the Schur-type algorithms 2.3 and 2.5, see, e.g., [GKO95]. However, the corresponding ordering of $\{x_k\}$ can also be computed in advance in $O(n^2)$ flops. Indeed, the partial pivoting technique determines a permutation matrix P, such that at each elimination step the pivot elements d_k in

$$PR = L \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{bmatrix} U$$

are as large as possible. Clearly, the determinant of the $k \times k$ leading submatrix of R is equal to $d_1 \cdot ... \cdot d_k$, so the objective of partial pivoting is the successive maximization of the determinants of leading submatrices. This

observation, and the well-known formula [C41] for the determinant of a Cauchy matrix, imply that partial pivoting on $C(x_{1:n}, y_{1:n})$ is equivalent to the successive maximization of the quantities

$$|d_i| = |\frac{\prod_{j=1}^{i-1}(x_i - x_j)\ \prod_{j=1}^{i-1}(y_i - y_j)}{(x_i - y_i)\ \prod_{j=1}^{i-1}(x_i - y_j)\ \prod_{j=1}^{i-1}(x_j - y_i)}|, \qquad i = 1, \ldots, n. \tag{5.4}$$

We shall call this procedure *predictive partial pivoting*, because it can be rapidly computed *in advance* by the following algorithm.

Algorithm 5.2 *Predictive Partial Pivoting*[3] *Complexity:* $2n^2 - 4$ *flops.*

```
function x = partial(x,y)
         n = max(size(x));
         dist = 0; m = 1; aux = zeros(1,n);
         for i = 1:n
             aux(i) = abs(1 / (x(i) - y(1)));
             if dist<aux(i) m = i; dist = aux(i); end
         end
         x = swap(x,1,m); aux(m) = aux(1);
         if n<=2 return; end
         for i = 2:(n-1)
             dist = 0; m = i;
             for j = i:n
                 aux(j) = aux(j) * abs((x(j) - x(i-1)) / (x(j) - y(i)));
                 if dist<aux(j) m = j; dist = aux(j); end
             end
             x = swap(x,i,m); aux(m) = aux(i);
         end
return
```

A similar row reordering technique for Vandermonde matrices (and a fast $O(n^2)$ algorithm for achieving it) was proposed in [Hig90], and in [R90] it was called *Leja ordering*. Therefore, PPP may also be called *rational Leja ordering*, by analogy with (polynomial) Leja ordering of [Hig90], [R90].

In a recent paper [Gu95] a variation of complete pivoting was suggested for the more general Cauchy-like matrices. In the context of [Gu95] the corresponding displacement rank is 2 or higher. For the ordinary Cauchy matrices $C(x_{1:n}, y_{1:n})$ (displacement rank = 1), Gu's pivoting can be described as follows. At each elimination step one chooses the column of R_k with the maximal magnitude entry $b_m^{(k)}$ in its generator B_k (here we use the notations of Lemma 2.4). Then one interchanges this column with the 1-st one, and performs the partial pivoting step. The explicit expression (2.10) for the entries of the successive generators B_k readily suggests a modification of the algorithm 5.2 to perform the Gu's variant of pivoting in advance, leading to what can be called *predictive Gu pivoting*.

6 Numerical illustrations

We performed numerous numerical tests for the three algorithms suggested and analyzed in this paper. The results confirm theoretical results (as perhaps should be expected). In this section we illustrate with just a few examples the influence of different orderings on the numerical performance of the following algorithms :

(a) Schur-type-Cauchy. (Fast $O(n)$ algorithm 2.3 requiring $O(n^2)$ storage.)

(b) Schur-type-direct-Cauchy. (Fast $O(n)$ algorithm 2.5 requiring $O(n^2)$ storage.)

(c) quasi-Cauchy. (Fast $O(n)$ algorithm 3.3 requiring $O(n)$ storage.)

(d) BKO. (Fast $O(n^2)$ algorithm of ([BKO99]) requiring $O(n)$ storage.)

[3] The subroutine " `swap(x,i,m)` " in Algorithm 5.2 swaps the i-th and m-th elements of the vector x.

(e) INV. The use of the explicit inversion formula

$$C(x_{1:n},\ y_{1:n})^{-1} = \left[\frac{\prod_{k=1}^{n}(x_k - y_i)}{\prod_{\substack{k=1 \\ k\neq i}}^{n}(y_k - y_i)} \cdot \frac{1}{x_j - y_i} \cdot \frac{\prod_{k=1}^{n}(x_j - y_k)}{\prod_{\substack{k=1 \\ k\neq j}}^{n}(x_j - x_k)} \right]_{1\le i,j\le n}, \tag{6.1}$$

(Fast $O(n)$ algorithm requiring $O(n)$ storage.)

(f) GEPP. Gaussian elimination with partial pivoting. (Slow $O(n^3)$ algorithm, requiring $O(n^2)$ storage.)

We refer to [BKO99] for the discussion and computed examples related to the important case of totally positive Cauchy matrices, and restrict ourselves here to the generic case in which the two sets $\{x_k\}$ and $\{y_k\}$ cannot be separated from each other, so that they cannot be reordered to achieve (1.8). We solved various Cauchy linear systems

$$Ca = f \tag{6.2}$$

(including interlaced $x_1 < y_1 < x_2 < y_2 < ... < x_n < y_n$ equidistant, clustered or randomly distributed nodes, and with many others configurations) with different right-hand sides (RHS) f. We also solved the so-called Cauchy-Toeplitz linear systems with coefficient matrices of the form

$$C = \left[\frac{1}{a+(i-j)b} \right] \tag{6.3}$$

with different choices for the parameters a and b. All the experiments were performed on a DEC 5000/133 RISC workstation in single precision (unit roundoff $\approx 1.19 \cdot 10^{-7}$). For GEPP we used the LAPACK routine **SGESV**, and all the other algorithms were implemented in C. In order to check the accuracy we implemented all the above algorithms in double precision (unit roundoff $\approx 2.22 \cdot 10^{-16}$), and in each example we determined two particular algorithms providing solutions that were the closest to each other. In all cases these two solutions agreed in more than the 8 significant digits needed to check the accuracy for a solution obtained in single precision, so we regarded one of these double precision solutions $\hat{a}_d$ as being exact, and used it to compute the 2-norm relative error

$$e_i = \frac{\|\hat{a}_i - \hat{a}_d\|_2}{\|\hat{a}_d\|_2}$$

for the solutions $\hat{a}_i$ computed by each of the above algorithms. In addition we computed the residual errors

$$r_i = \frac{\|f - C\cdot\hat{a}_i\|_2}{\|f\|_2}$$

and the backward errors $b_i = \min\left\{ \frac{\|\Delta A\|_2}{\|A\|_2} \ : \ (A+\Delta A)\hat{a}_i = f \right\}$) using the formula

$$b_i = \frac{\|f - C\cdot\hat{a}_i\|_2}{\|C\|_2 \cdot \|\hat{a}_i\|_2},$$

a result probably first shown by Wilkinson, see, e.g., [Hig96]. The tables below display also the condition number $\kappa_2(C)$ of the coefficient matrix, norms for the solution $\|\hat{a}_d\|_2$ and the right-hand side $\|f\|_2$ as well as some other useful information.

6.1 Example 1. Well-conditioned Cauchy-Toeplitz matrices

In this example we solved the linear system (6.2) with Cauchy-Toeplitz coefficient matrix in (6.3), with $a = 1$, $b = 2$, and with the right-hand side $f = \left[\ 1 \quad 1 \quad \cdots \quad 1\ \right]^T$. We used two orderings

- The nodes were ordered using the predictive partial pivoting (PPP) technique (1.8).
- The nodes $\{x_k\}$ sorted in an increasing order, and the nodes $\{y_k\}$ sorted in a decreasing order; the difference with (1.8) is in that now two sets of nodes $\{x_k\}$, $\{y_k\}$ are not separated from each other .

Forward error.

Table 1. Forward error. Partial pivoting ordering.

n	$\kappa_2(c)$	2-norms $\|C\|_2$	$\|\hat{a}_d\|_2$	$\|f\|_2$	INV e_1	BKO e_2	quasi-Cauchy e_3	Schur-type-Cauchy e_4	Schur-type-direct-Cauchy e_5	GEPP e_6
10	2e+00	1.6e+00	6.1e+00	4.5e+00	7e-08	1e-07	1e-07	6e-08	6e-08	8e-08
50	3e+00	1.6e+00	1.6e+01	1.0e+01	1e-07	2e-01	9e-08	2e-07	4e-07	1e-07
100	3e+00	1.6e+00	2.4e+01	1.4e+01	4e-07	1e+10	1e-07	1e-07	5e-07	1e-07

Table 2. Forward error. Monotonic ordering.

n	$\kappa_2(c)$	2-norms $\|C\|_2$	$\|\hat{a}_d\|_2$	$\|f\|_2$	INV e_1	BKO e_2	quasi-Cauchy e_3	Schur-type-Cauchy e_4	Schur-type-direct-Cauchy e_5
5	1e+00	1.6e+00	4.0e+00	3.2e+00	8e-08	2e-07	3e-06	1e-06	2e-06
10	2e+00	1.6e+00	6.1e+00	4.5e+00	1e-07	2e-05	1e-03	7e-03	6e-03
20	3e+00	1.6e+00	9.3e+00	6.3e+00	1e-07	6e-02	9e+04	3e+04	5e+04
30	3e+00	1.6e+00	1.2e+01	7.7e+00	1e-07	7e+03	3e+16	6e+17	2e+16
50	3e+00	1.6e+00	1.6e+01	1.0e+01	2e-07	1e+20	NaN	NaN	NaN
60	3e+00	1.6e+00	1.8e+01	1.1e+01	NaN	4e+26	NaN	NaN	NaN

Backward error.

Table 3. Backward error. Partial pivoting ordering.

n	$\max_{ij}(\|L\|\|DU\|)$	$\max_{ij}(C)$	INV $\hat{b}_1$	BKO $\hat{b}_2$	quasi-Cauchy $\hat{b}_3$	Schur-type-Cauchy $\hat{b}_4$	Schur-type-direct-Cauchy $\hat{b}_5$	GEPP $\hat{b}_6$
10	2e+00	1.0e+00	6e-08	1e-07	6e-08	6e-08	4e-08	5e-08
50	2e+00	1.0e+00	1e-07	2e-01	8e-08	1e-07	2e-07	1e-07
100	2e+00	1.0e+00	3e-07	1e+00	1e-07	1e-07	2e-07	1e-07

Table 4. Backward error. Monotonic ordering.

n	$\max_{ij}(\|L\|\|DU\|)$	$\max_{ij}(C)$	INV $\hat{b}_1$	BKO $\hat{b}_2$	quasi-Cauchy $\hat{b}_3$	Schur-type-Cauchy $\hat{b}_4$	Schur-type-direct-Cauchy $\hat{b}_5$
5	6e+02	1.0e+00	6e-08	2e-07	3e-06	1e-06	2e-06
10	3e+06	1.0e+00	7e-08	2e-05	1e-03	7e-03	6e-03
20	1e+14	1.0e+00	1e-07	6e-02	1e+00	1e+00	1e+00
30	4e+21	1.0e+00	1e-07	1e+00	1e+00	1e+00	1e+00
50	6e+36	1.0e+00	2e-07	1e+00	NaN	NaN	NaN
60	2e+44	1.0e+00	NaN	1e+00	NaN	NaN	NaN

Residual error.

Table 5. Residual error. Partial pivoting ordering.

n	$\max_i(\|L\|\|DU\|\|\hat{a}_d\|)$	$\max_{ij}(C)$	INV r_1	BKO r_2	quasi-Cauchy r_3	Schur-type-Cauchy r_4	Schur-type-direct-Cauchy r_5	GEPP r_6
10	9e+00	1e+00	1e-07	3e-07	1e-07	1e-07	9e-08	1e-07
50	2e+01	1e+00	4e-07	5e-01	2e-07	3e-07	4e-07	3e-07
100	3e+01	1e+00	8e-07	3e+10	3e-07	3e-07	5e-07	3e-07

Table 6. Residual error. Monotonic ordering.

n	$\max_i(\|L\|\|DU\|\|\hat{a}_d\|)$	$\max_{ij}(C)$	INV r_1	BKO r_2	quasi-Cauchy r_3	Schur-type-Cauchy r_4	Schur-type-direct-Cauchy r_5
5	9e+02	1e+00	1e-07	3e-07	6e-06	3e-06	4e-06
10	1e+07	1e+00	2e-07	4e-05	3e-03	2e-02	1e-02
20	7e+14	1e+00	3e-07	1e-01	2e+05	8e+04	1e+05
30	3e+22	1e+00	3e-07	2e+04	7e+16	1e+18	4e+16
50	7e+37	1e+00	4e-07	3e+20	NaN	NaN	NaN
60	3e+45	1e+00	NaN	9e+26	NaN	NaN	NaN

Comparing the data in Tables 1-6 indicates that the ordering of the nodes has a profound influence on the accuracy of all algorithms designed in the present paper. Specifically, let us recall that the quantity $|L|\,|DU|$ appear in the backward error bounds (4.3) and (4.7) for the algorithms Schur-type-direct-Cauchy and quasi-Cauchy, respectively. The second columns of Tables 3 and 4 show that the latter quantity is huge with monotonic ordering and moderate with PPP ordering. Correspondingly, the backward errors shown in the tables are large with monotonic ordering, and pleasantly small with PPP ordering.

Analogously, a comparison of the data in the second columns of Tables 5 and 6, shows that PPP technique reduces the quantity $(|L|\,|DU||\hat{a}|$ appearing in the residual bounds for the algorithms Schur-type-direct-Cauchy and quasi-Cauchy, resulting in a favorable small residual error for these algorithms.

Further, it is well-known that

$$\frac{\|a-\hat{a}\|}{\|a\|} \leq \kappa_2(C)\frac{\|\Delta C\|}{\|C\|}.$$

Since the coefficient matrix C in this example is quite well-conditioned (see, e.g., the data in the second column of Table 1), the PPP technique yields a pleasant forward accuracy for all algorithms Schur-type-Cauchy, Schur-type-direct-Cauchy and quasi-Cauchy.

The PPP technique also improves the numerical performance of the BKOBP-type algorithm, however for this algorithm the results are not as favorable as for other algorithms (see, e.g., introduction for the explanation of this phenomena, and [BKO99] for the discussion on extremely high accuracy of this algorithm for totally positive Cauchy matrices).

The use of explicit inversion formula also yields high accuracy, predicted by the analysis of [GK93], and apparently, this is the only algorithm whose accuracy does not depend upon the ordering of the nodes. At the same time,

comparison of the data in Tables 1 and 2 as well as in other examples indicates that the use of the PPP technique prevents the INV algorithm from overflows, allowing to solve larger linear systems.

Since in this example the coefficient matrix is well-conditioned, the $O(n^3)$ GEPP algorithm, while slow, also provides good forward and backward accuracy.

The results of many other computed examples are quite similar to those in Tables 1-6, and the algorithms Schur-type-Cauchy, Schur-type-direct-Cauchy and quasi-Cauchy always yield a favorable small backward and residual errors, which are often better than those of GEPP and of the use of inversion formula. As for the forward stability, there seem to be no clear winner, however the use of of inversion formula often provides provides smaller forward error, especially when using the unit vectors for the right-hand side (which means that one has to find a column of $C(x_{1:n}, y_{1:n})^{-1}$). At the same time, new algorithms can provide smaller forward error in other cases, as illustrated by the next examples.

6.2 Example 2. Ill-conditioned Cauchy-Toeplitz matrices.

The condition number of Cauchy-Toeplitz matrices depends upon the choice of parameters a and b in (6.3). In this example we chose $a = 1$, $b = -0.3$ to obtain a Cauchy-Toeplitz matrix, whose condition number is several order if magnitude bigger than the reciprocal to the machine precision $u \approx 1.19 \cdot 10^{-7}$. The next tables 7-12 present the data on the forward and backward errors for the corresponding linear system with the RHS $f = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T$. Along with the PPP ordering we also consider the original ordering (no ordering), and the Gu's pivoting ordering of $\{x_k\}$, $\{y_k\}$.

Forward error.

Table 7. Forward error. Partial pivoting ordering.

n	$\kappa_2(c)$	2-norms $\|C\|_2$	$\|\hat{a}_d\|_2$	$\|f\|_2$	INV e_1	BKO e_2	quasi-Cauchy e_3	Schur-type-Cauchy e_4	Schur-type-direct-Cauchy e_5	GEPP e_6
60	5e+10	1.2e+01	2.7e+05	7.7e+00	5e-03	4e+23	3e-03	2e-06	2e-04	1e+00
80	3e+11	1.2e+01	7.1e+05	8.9e+00	2e-02	NaN	5e-03	1e-05	3e-04	1e+00
100	9e+11	1.2e+01	1.5e+06	1.0e+01	2e-02	NaN	8e-03	3e-06	6e-04	1e+00

Table 8. Forward error. Gu's pivoting.

n	INV e_1	BKO e_2	quasi-Cauchy e_3	Schur-type-Cauchy e_4	Schur-type-direct-Cauchy e_5
60	2e-03	7e+01	8e-04	1e-04	3e-04
80	6e-03	4e+05	4e-03	3e-04	2e-03
100	1e-02	2e+09	5e-03	5e-04	4e-03

Table 9. Forward error. No pivoting.

n	INV e_1	BKO e_2	quasi-Cauchy e_3	Schur-type-Cauchy e_4	Schur-type-direct-Cauchy e_5
60	5e-03	1e+02	9e-04	2e-04	7e-05
80	3e-02	2e+05	1e-03	5e-04	1e-04
100	2e-02	1e+09	7e-04	6e-04	2e-04

Backward error.

Table 10. Backward error. Partial pivoting ordering.

n	$\max_{ij}(\|L\|\|DU\|)$	$\max_{ij}(C)$	INV $\hat{b}_1$	BKO $\hat{b}_2$	quasi-Cauchy $\hat{b}_3$	Schur-type-Cauchy $\hat{b}_4$	Schur-type-direct-Cauchy $\hat{b}_5$	GEPP $\hat{b}_6$
60	1e+03	1.0e+01	4e-04	1e+00	2e-07	3e-07	3e-07	4e-07
80	1e+03	1.0e+01	1e-03	NaN	4e-07	4e-07	4e-07	7e-07
100	2e+03	1.0e+01	6e-03	NaN	6e-07	6e-07	7e-07	1e-06

Table 11. Backward error. Gu's pivoting.

n	$\max_{ij}(\|L\|\|DU\|)$	INV $\hat{b}_1$	BKO $\hat{b}_2$	quasi-Cauchy $\hat{b}_3$	Schur-type-Cauchy $\hat{b}_4$	Schur-type-direct-Cauchy $\hat{b}_5$
60	1e+04	2e-04	1e+00	9e-05	1e-04	5e-05
80	3e+04	2e-03	1e+00	2e-04	3e-04	1e-04
100	4e+04	2e-03	1e+00	5e-04	5e-04	2e-04

Table 12. Backward error. No pivoting.

n	$\max_{ij}(\|L\|\|DU\|)$	INV $\hat{b}_1$	BKO $\hat{b}_2$	quasi-Cauchy $\hat{b}_3$	Schur-type-Cauchy $\hat{b}_4$	Schur-type-direct-Cauchy $\hat{b}_5$
60	1e+04	1e-03	1e+00	1e-04	2e-04	7e-05
80	3e+04	4e-03	1e+00	4e-04	5e-04	1e-04
100	5e+04	2e-03	1e+00	4e-04	6e-04	2e-04

Again, comparison of the data in Tables 10-12 confirms the analytical results of sections 4,5, indicating that an appropriate pivoting technique can reduce the size of backward errors for the new algorithms, making them as favorable as those of GEPP. The coefficient matrix in examples 3 and 4 is quite ill-conditioned, so the forward accuracy of GEPP is less favorable. However, the algorithms Schur-type-Cauchy, Schur-type-direct-Cauchy and quasi-Cauchy combined with partial pivoting provide smaller forward errors than GEPP (and the use of inversion

formula), showing that the use of the structure often allows us not only to speed-up the computation, but to also achieve more accuracy, as compared to general structure-ignoring methods.

It may seem to be quite unexpected that for Cauchy-Toeplitz matrices the Gu's pivoting technique (combining row and column permutations) can lead to less accurate solutions as compared to the PPP technique (based on row permutations only). To understand this occurrence it is useful to observe that the entries of the diagonals of Cauchy-Toeplitz matrices $\left[\frac{1}{a+b(i-j)} \right]$ depend hyperbolically on the difference $(i-j)$, thus giving a pick for the diagonal with $(i-j) \approx -a/b$. We next display the MATLAB graphs for the several permuted versions of the matrix in Example 2 for $n = 10$.

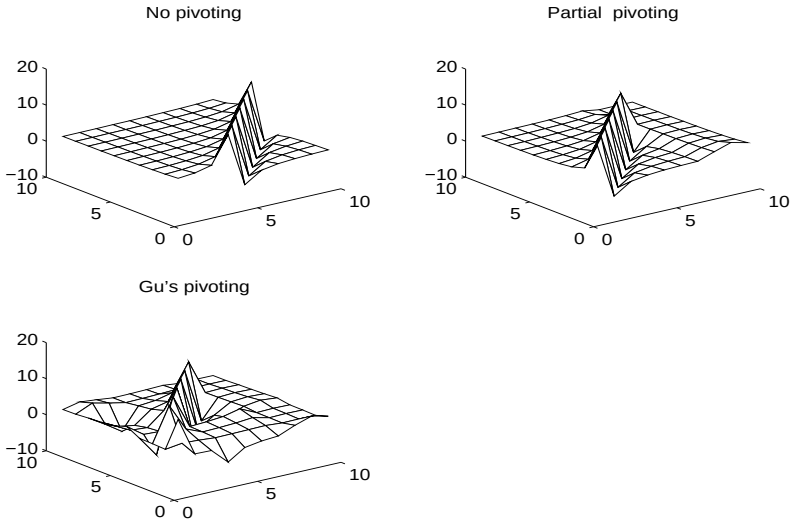

Figure 1: Permuted versions of a Cauchy-Toeplitz matrix, corresponding to different pivoting techniques

One sees that in the original matrix $C(x_{1:10}, y_{1:10})$ the maximal magnitude entries (all = 10) occupy the 4-th subdiagonal (i.e., in the lower triangular part of the matrix). Applying partial pivoting technique means moving each of the rows 4-10 three postions up, so that the maximal magnitude entries are now all located on the main diagonal. In the next table we list the condition numbers for the $k \times k$ leading submatrices corresponding to the three pivoting techniques.

Table 13. Conditioning of leading submatrices.

k	No pivoting	Gu's pivoting	Partial pivoting
1	1.0	1.0	1.0
2	$4.6 \cdot 10^1$	9.8	1.2
3	$7.2 \cdot 10^2$	$2.0 \cdot 10^1$	1.3
4	$1.0 \cdot 10^4$	$3.2 \cdot 10^1$	1.3
5	$3.8 \cdot 10^4$	$2.6 \cdot 10^1$	1.3
6	$1.0 \cdot 10^5$	$2.3 \cdot 10^1$	1.3
7	$2.5 \cdot 10^5$	$2.3 \cdot 10^1$	1.4
8	$5.5 \cdot 10^5$	$1.0 \cdot 10^2$	$1.3 \cdot 10^2$
9	$1.0 \cdot 10^6$	$6.8 \cdot 10^3$	$1.2 \cdot 10^4$
10	$1.9 \cdot 10^6$	$1.9 \cdot 10^6$	$1.9 \cdot 10^6$

We note, however, that the motivation for introducing the Gu's pivoting technique was given in [Gu95], where an application of [GKO95] with displacement rank 2 or higher was discussed.

6.3 Example 3. A transposed system

However, an immediate question is what will happen for a transposed to the matrix in Example 2 (clearly the transposed Cauchy matrix is a Cauchy matrix itself). Therefore we consider here a Cauchy-Toeplitz matrix with

the parameters $a = 1$ and $b = 0.3$. For such a matrix the maximal magnitude entries will now be located above the main diagonal. Therefore it is reasonable to apply a partial *column* pivoting technique. As in the above example, we next display the permuted versions of a matrix, corresponding to different pivoting techniques.

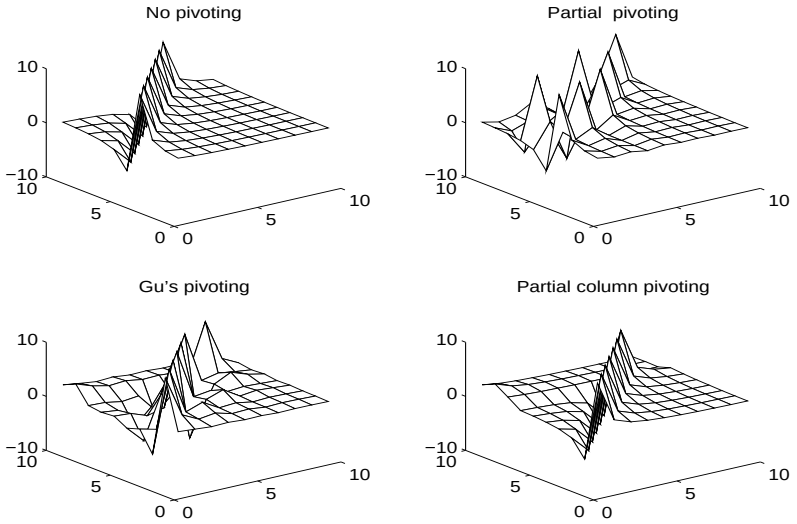

Figure 2: Permuted versions of a Cauchy-Toeplitz matrix, corresponding to different pivoting techniques

In Table 14 we list the corresponding condition numbers for all successive leading submatrices.

Table 14. Conditioning of leading submatrices.

k	No pivoting	Gu's pivoting	Partial column pivoting
1	1.0	1.0	1.0
2	$4.6 \cdot 10^1$	$2.0 \cdot 10^1$	1.2
3	$7.2 \cdot 10^2$	$3.6 \cdot 10^1$	1.3
4	$1.0 \cdot 10^4$	$2.9 \cdot 10^1$	1.3
5	$3.8 \cdot 10^4$	$6.3 \cdot 10^1$	1.3
6	$1.0 \cdot 10^5$	$5.4 \cdot 10^1$	1.3
7	$2.5 \cdot 10^5$	$9.2 \cdot 10^1$	1.4
8	$5.5 \cdot 10^5$	$1.1 \cdot 10^2$	$1.3 \cdot 10^2$
9	$1.0 \cdot 10^6$	$1.2 \cdot 10^4$	$1.2 \cdot 10^4$
10	$1.9 \cdot 10^6$	$1.9 \cdot 10^6$	$1.9 \cdot 10^6$

We now turn to the numerical results comparing the performance of the algorithms designed in the present paper. We again used the vector $f = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T$ for the right-hand side.

Forward error.

Table 15. Forward error. Partial pivoting ordering.

n	$\kappa_2(c)$	2-norms $\|C\|_2$	2-norms $\|\hat{a}_d\|_2$	2-norms $\|f\|_2$	INV e_1	BKO e_2	quasi-Cauchy e_3	Schur-type-Cauchy e_4	Schur-type-direct-Cauchy e_5	GEPP e_6
40	5e+09	1.2e+01	7.1e+04	6.3e+00	2e-03	3e-05	3e-05	5e-06	5e-05	2e+00
60	5e+10	1.2e+01	2.7e+05	7.7e+00	4e-03	1e-04	4e-04	7e-06	6e-04	1e+00
80	3e+11	1.2e+01	7.1e+05	8.9e+00	9e-03	6e+02	3e-03	6e-06	3e-04	1e+00
100	9e+11	1.2e+01	1.5e+06	1.0e+01	3e-02	3e+04	2e-03	6e-06	2e-03	1e+00

Table 16. Forward error. Partial column pivoting.

n	$\kappa_2(c)$	2-norms $\|C\|_2$	2-norms $\|\hat{a}_d\|_2$	2-norms $\|f\|_2$	INV e_1	BKO e_2	quasi-Cauchy e_3	Schur-type-Cauchy e_4	Schur-type-direct-Cauchy e_5
40	5e+09	1.2e+01	7.1e+04	6.3e+00	2e-03	3e+14	2e-03	2e-06	2e-04
60	5e+10	1.2e+01	2.7e+05	7.7e+00	5e-03	1e+28	3e-03	2e-06	2e-04
80	3e+11	1.2e+01	7.1e+05	8.9e+00	6e-03	NaN	9e-03	8e-06	2e-03
100	9e+11	1.2e+01	1.5e+06	1.0e+01	1e-02	NaN	3e-03	8e-06	6e-04

Table 17. Forward error. Gu's pivoting.

n	$\kappa_2(c)$	2-norms $\|C\|_2$	2-norms $\|\hat{a}_d\|_2$	2-norms $\|f\|_2$	INV e_1	BKO e_2	quasi-Cauchy e_3	Schur-type-Cauchy e_4	Schur-type-direct-Cauchy e_5
40	3e+08	2.0e+01	3.3e+03	6.3e+00	1e-03	9e-07	3e-04	9e-06	5e-04
60	3e+08	1.2e+01	1.1e+03	7.7e+00	8e-03	3e-04	1e-03	1e-04	2e-04
80	2e+09	1.2e+01	9.9e+02	8.9e+00	4e-02	7e+02	8e-03	2e-05	8e-03
100	4e+11	1.2e+01	6.6e+05	1.0e+01	3e-02	8e+03	2e-03	6e-05	5e-03

Table 18. Forward error. No pivoting.

n	$\kappa_2(c)$	2-norms $\|C\|_2$	$\|\hat{a}_d\|_2$	$\|f\|_2$	INV e_1	BKO e_2	quasi-Cauchy e_3	Schur-type-Cauchy e_4	Schur-type-direct-Cauchy e_5
40	5e+09	1.2e+01	7.1e+04	6.3e+00	4e-04	1e-07	7e-04	2e-05	3e-04
60	5e+10	1.2e+01	2.7e+05	7.7e+00	9e-03	1e-04	6e-04	8e-05	1e-03
80	3e+11	1.2e+01	7.1e+05	8.9e+00	1e-02	2e+01	4e-03	1e-04	6e-05
100	9e+11	1.2e+01	1.5e+06	1.0e+01	2e-02	3e+03	1e-02	4e-04	2e-03

In this example the forward accuracy of the Schur-type-Cauchy algorithm is better than that of the Schur-type-direct-Cauchy and quasi-Cauchy algorithms. Note that there are many other examples, however, where these algorithms have roughly the same accuracy.

Backward error. It turns out that for many different orderings all the algorithms designed in this paper exhibit a favorable backward stability. Moreover, for n varying from 5 to 100, for partial row pivoting, partial column pivoting, the Gu's pivoting, and for no-pivoting the algorithms Schur-type-Cauchy, Schur-type-Cauchy-direct and quasi-Cauchy produced backward errors of the order of 10^{-8} which is comparable to that of GEPP. We however, found that monotonic ordering, defined in Sec. 6.1, and randomized ordering produce poor results.

This indicates that analytical error bounds obtained for the fast algorithms of this paper in fact may lead to a wide variety of different pivoting techniques, each aimed at the reduction of the quantity $|L| \cdot |U|$.

7 Transformation of a Polynomial-Vandermonde Matrix into a Cauchy Matrix

In this section we shall show that all the fast Cauchy solvers suggested in the present paper can be used to solve linear systems with *polynomial Vandermonde* matrices,

$$V_{\mathcal{P}}(x_{1:n}) \stackrel{\text{def}}{=} \begin{bmatrix} P_0(x_1) & P_1(x_1) & \dots & P_{n-1}(x_1) \\ P_0(x_2) & P_1(x_2) & \dots & P_{n-1}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ P_0(x_n) & P_1(x_n) & \dots & P_{n-1}(x_n) \end{bmatrix}. \tag{7.1}$$

where $\mathcal{P} = \{P_0(x), \dots, P_{n-1}(x)\}$ denotes a basis in the linear space $\mathbb{C}_{n-1}[x]$ of all complex polynomials whose degree does not exceed $n-1$. When $\mathcal{P}$ is the power basis, then $V_{\mathcal{P}}(x_{1:n})$ is the ordinary Vandermonde matrix. If P stands for the basis of Chebyshev polynomials (of the first or of the second kind), then $V_{\mathcal{P}}(x_{1:n})$ is called a *Chebyshev-Vandermonde matrix*. Fast $O(n^2)$ algorithms for solving Chebyshev-Vandermonde systems were suggested in [Hig88], [HHR89], [CR93], [GO94d]. Here we suggest an alternative, based on the next result.

Proposition 7.1 *Let $\{x_1, ..., x_n; y_1, ..., y_n\}$ be $2n$ pairwise distinct complex numbers, and let $u(x) = \prod_{i=1}^{n}(x - y_i)$. Then the following formula is valid.*

$$V_{\mathcal{P}}(x_{1:n}) = \mathit{diag}\{u(x_1), \dots, u(x_n)\}\ C(x_{1:n},\ y_{1:n})\ \mathit{diag}\{\tfrac{1}{u'(y_1)}, \dots, \tfrac{1}{u'(y_n)}\}\ V_{\mathcal{P}}(y_{1:n}). \tag{7.2}$$

We shall prove the above proposition at the end of this section.

Observe that formula (7.2) relates $V_{\mathcal{P}}(x_{1:n})$ and $V_{\mathcal{P}}(y_{1:n})$, or, in other words, it allows us to change the nodes from $\{x_k\}$ to $\{y_k\}$, while keeping the polynomial basis $\mathcal{P} = \{P_0, \dots, P_{n-1}\}$. Suitable choices of the new points $\{y_{1:n}\}$ can ensure that $V_{\mathcal{P}}(y_{1:n})$ has low complexity. In such cases Proposition 7.1 allows us to reduce the problem of solving a linear system with $V_{\mathcal{P}}(x_{1:n})$ to the analogous problem of solving a linear system with the Cauchy matrix $C(x_{1:n},\ y_{1:n})$.

In the next proposition we specify several sets of points $\{y_{1:n}\}$ for which ordinary Vandermonde matrices and Chebyshev-Vandermonde matrices have low complexities.

Proposition 7.2

1). Let $y_j = \cos\left(\ \frac{j\pi}{n}\ \right)$, $j \in \{0, \dots, n\}$ be the extrema of $T_n(x)$. Then

$$V_T(y_{0:n}) = \left[\ \cos\tfrac{jk\pi}{n}\ \right]_{j,k=0}^{n} \tag{7.3}$$

is the (scaled) Discrete Cosine Transform I matrix.

2). Let $y_j = \cos\left(\ \frac{2j-1}{n}\frac{\pi}{2}\ \right)$, $j \in \{1,\ldots,n\}$ *be the zeros of the* $T_n(x)$. *Then*

$$V_T(y_{1:n}) = \left[\ \cos\tfrac{(2j-1)k}{n}\tfrac{\pi}{2}\ \right]_{j,k=1}^{n} \tag{7.4}$$

is the Discrete Cosine Transform II matrix.

3). Let $y_j = \cos\left(\ \frac{j\pi}{n}\ \right)$, $j \in \{1,\ldots,n-1\}$ *be the zeros of* $U_{n-1}(x)$. *Then*

$$V_U(y_{1:n-1}) = \left[\ \sin\tfrac{jk\pi}{n}\ \right]_{j,k=1}^{n-1} \tag{7.5}$$

is the (scaled) Discrete Sine Transform I matrix.

4). Let $y_j = \cos\left(\ \frac{2j-1}{n}\frac{\pi}{2}\ \right)$, $j \in \{1,\ldots,n\}$ *be the zeros of* $T_n(x)$. *Then*

$$V_U(y_{1:n}) = \left[\ \sin\tfrac{(2j-1)k}{n}\tfrac{\pi}{2}\ \right]_{j,k=1}^{n} \tag{7.6}$$

is the (scaled and transposed) Discrete Sine Transform II matrix.

5). Let $y_j = \exp\left(\ \imath\frac{2\pi j}{n}\ \right)$, $\imath = \sqrt{-1}$, $j \in \{0,1,...,n-1\}$ *denote the roots of unity. Then*

$$V(y_{1:n}) = \left[\ \exp\left(\ \imath\tfrac{2\pi j(k-1)}{n}\ \right)\ \right]_{j,k=0}^{n-1} \tag{7.7}$$

is the Discrete Fourier Transform matrix.

The latter proposition is easily deduced from the definitions of Chebyshev polynomials. Before proving Proposition 7.1, let us introduce the necessary notations. Let n denote the maximal degree of two polynomials $u(x)$ and $v(x)$. The bivariate function

$$\mathcal{B}_{u,v}(x,y) = \frac{u(x)\cdot v(y) - u(y)\cdot v(x)}{x-y}$$

is called the *Bezoutian* of $u(x)$ and $v(x)$. Now, let $\mathcal{Q} = \{Q_0(x),\ldots,Q_{n-1}(x)\}$ be another basis in the linear space $\mathbb{C}_{n-1}[x]$. The matrix

$$B_{\{\mathcal{P},\mathcal{Q},u,v\}} = \left[\ b_{ij}\ \right]_{i,j=0}^{n-1} \quad ,$$

whose entries are determined by

$$\begin{aligned}\mathcal{B}_{u,v}(x,y) &= \sum_{i,j=0}^{n-1} b_{ij}\cdot P_i(x)\cdot Q_j(y) \\ &= \left[\ P_0(x) \quad P_1(x) \quad \cdots \quad P_{n-1}(x)\ \right] B_{\{\mathcal{P},\mathcal{Q},u,v\}} \begin{bmatrix} Q_0(y) \\ Q_1(y) \\ \vdots \\ Q_{n-1}(y) \end{bmatrix}.\end{aligned} \tag{7.8}$$

is called the *Bezout matrix* of $u(x)$ and $v(x)$ with respect to the two sets of polynomials $\mathcal{P}$ and $\mathcal{Q}$.

Proof of Proposition 7.1. The proof is based on the following useful property of the Bezout matrix

$$V_{\mathcal{P}}(x_{1:n})\ B_{\{\mathcal{P},\mathcal{Q},u,v\}}\ V_{\mathcal{Q}}(y_{1:n})^T = \left[\ \mathcal{B}_{u,v}(x_i,y_j)\ \right]_{i,j=1}^{n} \quad . \tag{7.9}$$

which follows immediately from (7.8). It is easy to see that the matrix on the right-hand side of Eq. (7.9) is a quasi-Cauchy matrix :

$$\begin{aligned} V_{\mathcal{P}}(x_{1:n})\ B_{\{\mathcal{P},\mathcal{Q},u,v\}}\ V_{\mathcal{Q}}(y_{1:n})^T &= \left[\ \tfrac{u(x_i)\,v(y_j)}{x_i-y_j}\ \right]_{i,j=1}^{n} \\ = \operatorname{diag}\{u(x_1),\ldots,u(x_n)\}\ C(x_{1:n},y_{1:n})\ \operatorname{diag}\{v(y_1),\ldots,v(y_n)\}. \end{aligned} \tag{7.10}$$

On the other hand, by using (7.9) and the obvious relation

$$\mathcal{B}_{u,v}(y,y) = u'(y)\,v(y) - u(y)\,v'(y),$$

it is easy to check that

$$V_{\mathcal{P}}(y_{1:n})\ B_{\{\mathcal{P},\mathcal{Q},u,v\}}\ V_{\mathcal{Q}}(y_{1:n})^T = \operatorname{diag}\{u'(y_1)\,v(y_1)\,,\ \ldots\ ,\ u'(y_n)\,v(y_n)\}.$$

Now, substituting the $B_{\{\mathcal{P},\mathcal{Q},u,v\}}\ V_{\mathcal{Q}}(x_{1:n})^T$ obtained from the last equation back into (7.10) yields (7.2).

8 Conclusion

In [BKO99] we developed a fast $O(n^2)$ Björck-Pereyra-type Cauchy solver, and proved that for the important class of totally positive coefficient matrices it yields pleasantly small forward, backward and residual errors. However, experience shows that in the generic case the numerical performance of the BP-type algorithm can be less favorable. Since the use of explicit inversion formula for Cauchy matrices also can produce a large backward error, no fast and accurate methods methods were available for solving Cauchy linear equations. In this paper we designed several alternative fast $O(n^2)$ Cauchy solvers, and the rounding error analysis suggests that their backward stability is similar to that of Gaussian elimination (GE), so that various pivoting techniques (so successful for GE) will stabilize the numerical behavior also for these new algorithms. It is further shown that the row ordering of partial pivoting and of the Gu's pivoting [Gu95] can be achieved in advance, without actually performing elimination, and fast $O(n^2)$ algorithms for these purposes are suggested. We also identified a class of totally positive Cauchy matrices, for which it is advantageous not to pivot when using the new algorithms, which yields a remarkable backward stability. This matches the conclusion of de Boor and Pinkus, who suggested to avoid pivoting when performing standard Gaussian elimination on totally positive matrices. Analytical error bounds and results of numerical experiments indicate that the methods suggested in the present paper enjoy favorable backward stability.

References

[BKO94] T.Boros, T.Kailath and V.Olshevsky, *Fast algorithms for solving Cauchy linear systems*, Stanford Information Systems Laboratory report, 1994.

[BKO99] T.Boros, T.Kailath and V.Olshevsky, *Fast Björck-Pereyra-type algorithm for parallel solution of Cauchy linear equations*," Linear Algebra and Its Applications, 302-303 (1999), p.265-293.

[BP70] A.Björck and V.Pereyra, *Solution of Vandermonde Systems of Equations*, Math. Comp., **24** (1970), 893-903.

[C41] A.L. Cauchy, *Mémoires sur les fonctions alternées et sur les sommes alternées*, Exercices d'analyse et de phys. math., ii (1841), 151-159.

[CF88] T.Chan and D.Foulser, *Effectively well-conditioned linear systems*, SIAM J. Sci. Stat. Computation, 9 (1988), 963 – 969.

[CR92] D.Calvetti and L.Reichel, *A Chebyshev-Vandermonde solver*, Lin. Alg. Appl., **172**(1992), 219-229.

[CR93] Calvetti, D. and Reichel, L. : *Fast inversion of Vandermonde-like matrices involving orthogonal polynomials*, BIT, 1993.

[D01] H.Dym, *Structured Matrices, Reproducing Kernels and Interpolation,* to appear in *Structured Matrices in Mathematics, Computer Science and Engineering*, (V.Olshevsky, Editor), Contemporary Mathematics Series, vol. 280, AMS Publicatons, 2001.

[DBP77] C. de Boor and A. Pinkus, *Backward error analysis for totally positive linear systems*, Numer. Math., **27** (1977), 485-490.

[GK50] F.R.Gantmacher and M.G.Krein, *Oscillatory matrices and kernels, and small vibrations of mechanical systems*, second edition, (in Russian), GITTL, Moscow, 1950. *German translation :* Oszillationsmatrizen, Oszillationskerne und kleine Schwingungen mechanischer Systeme, Berlin, Akademie Verlag, 1960.

[GK90] I.Gohberg and I.Koltracht, *On the inversion of Cauchy matrices*, In Signal processing, Scattering and Operator Theory, and Numerical methods. Proc of the MTNS-89 (M.A.Kaashoek, J.H.van Schuppen and A.C.M.Ran, eds), 381-392, Birkhäuser, Boston, MA, 1990.

[GK93] I.Gohberg and I.Koltracht, *Mixed, componentwise and structured condition numbers*, SIAM J. Matrix Anal., **14**(1993), 688–704.

[GKO95] I.Gohberg, T.Kailath and V.Olshevsky, *Fast Gaussian elimination with partial pivoting for matrices with displacement structure*, Math. of Comp., **64** (1995), 1557-1576.

[GO94a] I.Gohberg and V.Olshevsky, *Fast state space algorithms for matrix Nehari and Nehari-Takagi interpolation problems*, Integral Equations and Operator Theory, **20**, No. 1 (1994), 44 – 83.

[GO94b] I.Gohberg and V.Olshevsky, *Fast algorithms with preprocessing for matrix–vector multiplication problems*, Journal of Complexity, **10** (1994), 411-427.

[GO94c] I.Gohberg and V.Olshevsky, *Complexity of multiplication with vectors for structured matrices*, Linear Algebra Appl., **202**, 1994, 163 – 192.

[GO94d] I.Gohberg and V.Olshevsky, *Fast inversion of Chebyshev–Vandermonde matrices*, Numer. Math., **67**, No. 1 (1994), 71 – 92.

[GVL89] G. Golub and C, Van Loan, *Matrix Computations*, second edition, John Hopkins U. P., Baltimore, 1989.

[Gu95] Ming Gu, *Stable and efficient algorithms for structured linear systems*, preprint, 1995.

[Hig87] N.J.Higham, *Error analysis of the Björck–Pereyra algorithms for solving Vandermonde systems*, Numer. Math., 50 (1987), 613 – 632.

[Hig88] N.J.Higham, *Fast solution of Vandermonde-like systems, involving orthogonal polynomials*, IMA J. Numer. Anal., **8** (1988), 473-486.

[Hig90] N.J.Higham, *Stability analysis of algorithms for solving confluent Vandermonde-like systems*, SIAM J. Matrix Anal., **11**(1) (1990), 23–41.

[Hig96] N.J.Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 1996.

[HHR89] G.Heinig, W.Hoppe and K.Rost, *Structured matrices in interpolation and approximation problems*, Wissenschaftl. Zeitschrift der TU Karl-Marx-Stadt 31, 2 (1989), 196 – 202.

[HR84] G.Heinig and K.Rost *Algebraic methods for Toeplitz-like matrices and operators*, Operator Theory, vol. 13, 1984, Birkauser Verlag, Basel.

[K72] S.Karlin, *Total Positivity*, vol 1, Stanford University Press, Stanford, 1972.

[KKM79] T.Kailath, S.Kung and M.Morf, *Displacement ranks of matrices and linear equations*, J. Math. Anal. and Appl., **68** (1979), 395-407.

[KO95a] T.Kailath and V.Olshevsky, *Displacement structure approach to Chebyshev–Vandermonde and related matrices*, Integral Equations and Operator Theory, **22**, 1995, 65 – 92.

[KO95b] T.Kailath and V.Olshevsky, *Bunch-Kaufman pivoting for partially reconstructable Cauchy-like matrices with application to Toeplitz-like linear equations and to boundary rational matrix interpolation problems*, 1995, to appear in Linear Algebra and Appl.

[KS95] T.Kailath and A.H.Sayed, *Displacement structure : Theory and Applications*, SIAM Review, **37** No.3 (1995), 297-386.

[LAK86] H.Lev-Ari and T.Kailath, *Triangular factorization of structured Hermitian matrices*, in *Operator Theory : Advances and Applications* (I.Gohberg. ed.), vol. **18**, 301 - 324, Birkhäuser, Boston, 1986.

[M80] M.Morf, *Doubling Algorithms for Toeplitz and Related Equations,* Proc. IEEE Internat. Conf. on ASSP, pp. 954-959, IEEE Computer Society Press, 1980.

[O97] V. Olshevsky, *Pivoting for structured matrices, with applications*, 53 p.
http://www.cs.gsu.edu/˜matrvo

[R90] L.Reichel, *Newton interpolation at Leja points*, BIT, 30(1990), 23 - 41.

[RO91] L. Reichel and G. Opfer, *Chebyshev-Vandermonde Systems*, Math. of Comp., **57**(1991) , 703-721.

[S17] I.Schur, *Über potenzreihen die im Inneren des Einheitskreises beschränkt sind*, Journal für die Reine und Angewandte Mathematik, **147** (1917), 205 - 232. English translation : in *Operator Theory : Advances and Applications* (I.Gohberg. ed.), vol. **18**, 31 – 88, Birkhäuser, Boston, 1986.

[S76] L.A. Sakhnovich, *The factorization of the operator-valued transfer functions,* (in Russian) Soviet Math. Dokl., 17 (1976), 203–207.

[S86] L.Sakhnovich, *Factorization problems and operator identities*, Russian Mathematical Surveys, 41, 1 (1986), 1 – 64.

[SB80] J.Stoer and R.Bulirsch, *Introduction to Numerical Analysis*, Springer-Verlag, New York, 1980.

[SB95] D.Sweet and R.Brent, *Error analysis of a partial pivoting method for structured matrices*, Advanced Signal processing algorithms, Proc of SPIE-1995, vol. **2563**, 266-280.

[TG81] W.Tang and G.Golub, *The block decomposition of a Vandermonde matrix and its aplications*, BIT, **21** (1981), 505-517.

[Ty94] E.Tyrtyshnikov, *How bad are Hankel matrices*, Numer. Math., **67**, No. 2 (1994), 261 – 269.

[V93] J. M. Varah, *Errors and Perturbations in Vandermonde Systems*, IMA J. of Numer. Anal., **13** (1993), 1-12.

[W68] J.Wilkinson, *A priori error analysis of algebraic processes*, Proc. Intern. Congr. Math. (1966), pp. 629-639, Moskow, Mir 1968.