Pivoting for Structured Matrices with Applications *

Vadim Olshevsky[†]

Contents

*This work was supported in part by NSF contract CCR 9732355.

†Department of Mathematics and Computer Science, Georgia State University, Atlanta, GA 30303,
E-mail: volshevsky@cs.gsu.edu Web: http://www-isl.stanford.edu/~olshevsk http://www.cs.gsu.edu/~matvro

Abstract

Gaussian elimination is a standard tool for computing triangular factorizations for general matrices, and thereby solving associated linear systems of equations. As is well-known, when this classical method is implemented in finite-precision-arithmetic, it often fails to compute the solution accurately because of the accumulation of small roundoffs accompanying each elementary floating point operation. This problem motivated a number of interesting and important studies in modern numerical linear algebra; for our purposes in this paper we only mention that starting with the breakthrough work of Wilkinson, several *pivoting techniques* have been proposed to stabilize the numerical behavior of Gaussian elimination.

Interestingly, matrix interpretations of many known and new algorithms for various applied problems can be seen as a way of computing triangular factorizations for the associated *structured matrices*, where different patterns of structure arise in the context of different physical problems. The special structure of such matrices [e.g., Toeplitz, Hankel, Cauchy, Vandermonde, etc.] often allows one to *speed-up* the computation of its triangular factorization, i.e., to efficiently obtain *fast implementations* of the Gaussian elimination procedure. There is a vast literature about such methods which are known under different

names, e.g., fast Cholesky, fast Gaussian elimination, generalized Schur, or Schur-type algorithms. However, without further improvements they are efficient fast implementations of a numerically inaccurate [for indefinite matrices] method.

In this paper we survey recent results on the fast implementation of various pivoting techniques which allowed us to improve numerical accuracy for a variety of fast algorithms. This approach led us to formulate new more accurate numerical methods for factorization of general and of J-unitary rational matrix functions, for solving various tangential interpolation problems, new Toeplitz-like and Toeplitz-plus-Hankel-like solvers, and new divided differences schemes. We beleive that similar methods can be used to design accurate fast algorithm for the other applied problems and a recent work of our colleagues supports this anticipation.

1 Motivation and a first example of structure: a Vandermonde matrix

The standard method for solving a $n \times n$ linear system of simultaneous equations, Gaussian elimination [GE], requires $O(n^3)$ arithmetic operations per system. However, this computational burden is certainly too expensive if the size n of the coefficent matrix is large, so often the use of GE can be impractical. Fortunately, many linear equations appearing in applications often have a certain *structure*, introduced by a particular physical phenomenon, and this structure can be exploited to speed-up the computation. For example, a Vandermonde linear system of equations,

$$V(x) \cdot a = f, \qquad \text{where} \qquad V(x) = \left[\ x_k^{j-1} \ \right]_{1 \le k,j \le n} \tag{1.1}$$

can be rapidly solved in only $O(n^2)$ operations via the *fast* Björck-Pereyra [BP] algorithm [BP70], [GVL96]. Of course, gain in speed is a very important factor from the practical point of view, but there are several other features that we may wish the algorithm to satisfy, such as amenability to parallel implementations, memory requirements and traffic, etc. In floating point arithmetic implementations, where the roundoff errors are present, the crucial factor that makes an algorithm practical is its *numerical accuracy.*

The accuracy of general purpose, i.e., *structure-ignoring* algorithms, such as GE, has been extensively studied over the last few decades, and for such algorithms we now have various so-called *stabilizing techniques* that improve the numerical accuracy of these algorithms. As a matter of fact, such techniques are based on the fact that a certain algorithm usually can be implemented in a variety of mathematically equivalent ways. So we often can use a certain heuristic to choose an implementation providing the *computed result* with a better accuracy.

In contrast, the analysis of accuracy for *fast algorithms*, viz., those that *exploit the structure* to speed-up the computation, has received much less attention. Moreover, we know many fast algorithms that are less accurate as compared to their general purpose counterparts. Of course, it would be too hasty to conclude that there is a certain *"original sin"* associated with all fast algorithms, and the explanation can be much simpler: at the moment the interest in the numerical behavior of fast algorithms is quite recent, so fewer stabilizing techniques are available. In this paper we survey recent progress in the design of fast implementations for one well-known technique of enhancing accuracy: *pivoting.*

To motivate the further discussions let us return to the special system of linear equations in (1.1), and observe that any reordering of the rows of $V(x)$ does not destroy the Vandermonde structure, and that it is equivalent to the reordering of the nodes $\{x_k\}$. Therefore, for each available algorithm, we can solve system (1.1) in a variety of mathematically equivalent, but *numerically different* ways, by preceding this algorithm with a certain permutation of $\{x_k\}$. Consider the following two particular orderings:

- Monotonic ordering,

$$x_1 < x_2 < \ldots < x_n,$$

- Leja ordering,

$$\begin{array}{c} |x_1| = \max_{1 \le j \le n} |x_j|, \\ \prod_{i=1}^{k-1} |x_k - x_i| = \max_{k \le j \le n} \prod_{i=1}^{k-1} |x_j - x_i| \quad \text{for} \quad 2 \le k \le n \ ; . \end{array} \tag{1.2}$$

Condition number of $V(x)$	Monotonic ordering GE	BP	Leja ordering GE	BP
2.36e+06	3.1e-08	9.7e-09	1.9e-10	2.3e-10

Table 1: *Residual error for nodes equidistant in (-1,1).*

Condition number of $V(x)$	Monotonic ordering GE	BP	Leja ordering GE	BP
1.53e+12	7.4e-05	2.7e-05	9.5e-05	6.5e-05

Table 2: *Residual error for nodes equidistant in (0,1).*

We used MATLAB©[1] to solve two 15×15 Vandermonde systems of the form (1.1), corresponding to the following two node configurations:

- equidistant in (-1,1),
- equidistant in (0,1),

and for the right-hand-side $f = \left[\begin{array}{ccccc} 1 & -1 & 1 & -1 & \dots \end{array} \right]^T$. For each of the two systems we tried two above orderings of $\{x_k\}$. For each of these four cases we applied GE and BP algorithms to compute the solution $\widehat{a}$; this computed solution is, of course, inaccurate, because of the roundoffs accompanying each elementary step of computation. The results of the measurement of the residual error $\|V(x)\widehat{a} - f\|_\infty$ are shown in Tables 1 and 2.

A closer look at the data in each table allows us to make the following two observations.

- **General matrices. Leja ordering.** The data in Table 1 and in numerous other examples suggest that for the nodes of both signs, $x_k \in (-1,1)$, the use of Leja ordering is a stabilizing technique for the both GE and BP algorithms, i.e., it provides a smaller size for the residual errors. This confirms our experience and the computational experience of many others that Leja ordering enhances the accuracy of various algorithms for Vandermonde matrices [Hig90], [R90b], [RO91], [GK93], [GO94a], [GO97].
- **Special matrices. Monotonic ordering.** Before analyzing the data in Table 2 observe that the coefficient matrix here differs from the one in Table 1 which implies different conditioning, different sizes of the solution, etc. Therefore we should just ignore any differences between the two above tables, and our analysis here should be aimed only at the data in Table 2. These data indicate that for the special class of Vandermonde matrices with positive nodes, $x_k \in (0,1)$, there is no need for Leja ordering, and monotonic ordering provides about the same accuracy.

The explanation for these two observations is simple enough to be briefly stated next.

- **General matrices. Partial pivoting.** The result in Table 2 is explained by the observation [perhaps first noted in [Hig90]] that (1.2) is nothing else but *partial pivoting*, *efficiently implemented* for a Vandermonde matrix.
- **Total positivity. Avoidance of pivoting.** The justification for the results in Table 2 is the fact that there are classes of matrices, in this case totally positive matrices, for which it is advantageous not to pivot [totally positive matrices are those for which the determinant of every submatrix is positive, see classic monographs [GK50], [K72] for a list applications].

[1]MATLAB is a trademark of The MathWorks Inc.

For the BP algorithms these two conclusions will be more transparently presented later in Sec. 9. For the GE procedure these two explanations are more explicit in the next section where we set the notation, and briefly survey several tutorial facts on pivoting for structure-ignoring general algorithms. Having this background, in the rest of the paper we shall present several similar stabilizing techniques for various other kinds of structure, and extend them to several other fast algorithms.

2 Ignoring structure: Pivoting to improve the accuracy of Gaussian elimination (GE)

2.1 Gaussian elimination

The first step of the GE procedure applied to a matrix R_1 can be described as a factorization of the following form. Let $R_1 := R$, then

$$R_1 \triangleq \begin{bmatrix} d_1 & u_1 \\ l_1 & R_{22}^{(1)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1}{d_1} l_1 & I \end{bmatrix} \cdot \begin{bmatrix} d_1 & u_1 \\ 0 & R_2 \end{bmatrix}, \tag{2.1}$$

where the matrix

$$R_2 = R_{22}^{(1)} - \frac{1}{d_1} l_l u_1 \tag{2.2}$$

is called the *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 the lower-triangular factor L and the first row $\begin{bmatrix} d_1 & u_1 \end{bmatrix}$ of the upper triangular factor U in the LU factorization of $R_1 = LU$. Recursively proceeding to factorize the Schur complement R_2, after $n-1$ steps we obtain the whole LU factorization.

Now, it is well-known, see, e.g., [SB80] or [Hig96], that in the standard model of floating point arithmetic,

$$fl\{x \ op \ y\} = (x \ op \ y)(1+\delta), \qquad |\delta| \le u, \tag{2.3}$$

where $op \in \{+,-,*,/\}$, $fl\{x \ op \ y\}$ denotes the *computed result*, and u is the *unit roundoff*,

$$u \approx \begin{cases} 10^{-8} & \text{in single precision} \\ 10^{-16} & \text{in double precision} \end{cases}$$

we have the following error bounds:

- **Factorization error.** The triangular factors $\widehat{L}$ and $\widehat{U}$ computed by Gaussian elimination satisfy

$$\widehat{L}\widehat{U} = R + \Delta R, \qquad |\Delta R| \le \gamma_n |\widehat{L}||\widehat{U}|, \qquad \text{where} \qquad \gamma_n = \frac{nu}{1-nu}, \tag{2.4}$$

  where the operations of taking the absolute value and of comparison are understood *componentwise*.

- **Backward error.** The use of the computed triangular factors $\widehat{L}$ and $\widehat{U}$ to solve the associated linear system $Ra = f$ via forward and backsubstitution produces a *computed solution* $\widehat{a}$ that is the exact solution of a nearby system $(R + \Delta R)\widehat{a} = f$, where

$$|\Delta R| \le 2\gamma_n |\widehat{L}||\widehat{U}|. \tag{2.5}$$

- **Residual error.** Finally,

$$|f - R\widehat{a}| \le 2\gamma_n |\widehat{L}|\,|\widehat{U}|\,|\widehat{a}|. \tag{2.6}$$

All three bounds involve the product $|\widehat{L}|\,|\widehat{U}|$, which can be quite large, indicating the potential numerical instability, in general, of the Gaussian elimination procedure. However, these bounds give a clue as how to numerically stabilize this classical algorithm. The standard way to deal with the often large size of

$$g_n = \frac{\|\,|\widehat{L}|\,|\widehat{U}|\,\|_\infty}{\|\,|R|\,\|_\infty} \tag{2.7}$$

is to exploit the freedom of row interchanges for R_k in (2.1) before the k-th step of elimination. Such techniques are called *pivoting* techniques, two of which are discussed next.

2.2 Partial or complete pivoting? Practical experience vs. error bounds

In this short subsection we briefly recall widely known heuristic arguments suggesting that while complete pivoting has better error bounds, nevertheless faster partial pivoting usually provides the same accuracy and is preferable. Though reasonings here concern with slow structure-ignoring algorithms only, but as we shall conjecture in Sec. 7.6 below the situation with fast transformation-and-pivoting algorithms exploiting the Toeplitz structure is analogous.

The *partial pivoting* technique performs the elimination on a permuted version of the coefficient matrix, $PR = LU$, where the permutation P is built recursively: $P = P_{n-1} \cdot \ldots \cdot P_1$. More specifically, an elementary permutation P_k is chosen to bring, at the k-th step, the maximal magnitude entry of the k-th column to the pivotal (k,k) position. This guarantees that the entries of the computed lower triangular factor $\widehat{L}$ are all less than 1, and loosely speaking, that the quantity g_n is usually of the order of unity, resulting in a small backward error (2.5) for GE *in practice.* The proviso *"in practice"* means that though partial pivoting has a good record of accurate performance, there are several examples for which it is not is not sufficient, and the backward error can be still large [see, e.g., [Hig96] and the references therein]. In these cases *complete pivoting* will do better: in this technique one recursively chooses a permutation matrix P in $PR = LU$ to bring, at the k-th step, the maximal magnitude entry in the *whole matrix* [not just in the first column] to the pivotal (k,k) position. However, complete pivoting requires significantly more comparisons, so partial pivoting is the stabilizing technique of the choice in most commercial and sharewhere packages, e.g., in MATLAB, LAPACK, etc. This choice is based on the fact that the size of (2.7) [so also of the backward error] is almost invariably small in practice for partial pivoting.

Here we might mention an interesting remark of W.M.Kahan [K80]: *"Intolerable pivot-growth [with partial pivoting] is a phenomenon that happens only to numerical analysts who are looking for that phenomenon."*

Moreover, in a recent monograph [Hig96] N.J.Higham states that *" although there are practically occurring matrices for which partial pivoting yields a moderately large, or even exponentially large, growth factor, the growth factor is almost invariably found to be small. Explaining this fact remains one of the major unsolved problems in numerical analysis."*

Interestingly, the importance of a proper balance between practical experience and error bounds [such *a-priori bounds are "often impractically large"* [W71]] was recognized already in the beginning of the era of error analysis. See, e.g., [W65] where in the introduction the author addressed this issue and noted that *" I felt that it is no longer practical to cover almost all known methods and to give an error analyses for them and decided to include mainly those methods of which I had extensive practical experience."*

2.3 Special classes of matrices and avoidance of pivoting

Nevertheless, there are classes of matrices for which it is advantageous not to pivot! For example, for *totally positive*[2] matrices the exact triangular factors have only positive entries. De Boor and Pinkus pointed out in [DBP77] that if the entries of the computed factors $\widehat{L}$ and $\widehat{U}$ remain nonnegative, then the quantity g_n in (2.7) is of the order of unity, so that the componentwise backward and residual errors (2.4) and (2.6) are pleasantly small, and moreover, $(R + \Delta R)\widehat{a} = f$ where

$$|\Delta R| \leq 3\gamma_n |R|. \tag{2.8}$$

2.4 Vandermonde matrices again

The above results give a theoretical justification for the numerical data in Tables 1 and 2. Indeed, the partial pivoting technique determines a permutation matrix P, such that at each elimination step the pivot elements d_k in

$$PR_1 = LU = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ * & & 1 \end{bmatrix} \begin{bmatrix} d_1 & & * \\ & \ddots & \\ 0 & & d_n \end{bmatrix}$$

[2]Totally positive matrices are defined as those for which the determinant of every submatrix is positive, see the classic monograph [GK50] for the definition and for several applications.

are as large as possible. Clearly, the determinant of the leading $k \times k$ submatrix of R_1 is equal to $d_1 \cdot \ldots \cdot d_k$, so we can say that the objective of partial pivoting is to successively maximize the determinants of the leading submatrices. This observation, and the well-known formula

$$\det V(x_{1:k}) = \prod_{\substack{1 \le j \le k \\ i < j}} (x_j - x_i)$$

for the determinant of the $k \times k$ Vandermonde matrix easily imply that the Leja ordering (1.2) mimics the partial pivoting ordering of the rows of $V(x)$, see [Hig90]. So it is not surprising that the accuracy of GE with Leja ordering will be higher than with monotonic ordering, thus explaining the occurrence in Table 1.

Furthermore, it is nice to note that that the condition $0 < x_1 < \ldots < x_n$ is well-known [GK50] to imply total positivity for the corresponding Vandermonde matrix, thus explaining the occurrence in Table 2.

In brief, the well-known techniques stabilizing the structure-ignoring GE procedure can be efficiently implemented for the special Vandermonde structure as a certain manipulation on the parameters defining such matrices. In the rest of the paper we show how this can be done for the other classes of structured matrices as well, and for fast algorithms exploiting their patterns of structure.

Before addressing structured matrices in detail we shall briefly describe next one more pivoting technique that we shall use below.

2.5 Hermitian matrices and diagonal pivoting

Matrices appearing in applications are often Hermitian, and exploiting this fact allows one to reduce the storage requirement by the factor 2, and since the symmetry is preserved during the elimination [cee, e.g., (2.10) below], we can perform twice as less computations. It is therefore desirable not to lose symmetry under pivoting operations, so to perform symmetric permutations of rows and columns: In this section we recall the Bunch-Kaufman diagonal pivoting technique

$$R \leftarrow P^T R P. \tag{2.9}$$

Clearly, the main diagonal of $P^T R P$ is a rearrangement of the main diagonal of R, so the reordering technique that brings the maximal magnitude element on the main diagonal to the pivotal position is called *symmetric pivoting.* Symmetric pivoting technique is sometimes used with positive definite matrices [GVL96], for which it is equivalent to complete pivoting. However, this method breaks down with indefinite matrices in the case when the whole main diagonal is zero. Moreover, even if the main diagonal will never become zero during elimination, symmetric pivoting in general cannot provide sufficiently large pivots, so this method is not backward stable for indefinite matrices.

The remedy [perhaps first suggested by W.M.Kahan] is to sometimes perform block elimination steps with appropriate 2×2 block pivots. Symmetric block Gaussian elimination is based on recursive applying the well-known block Schur complementation formula

$$R_1 = \begin{bmatrix} R_{11} & R_{21}^* \\ R_{21} & R_{22} \end{bmatrix} = \begin{bmatrix} I & 0 \\ R_{21} \cdot R_{11}^{-1} & I \end{bmatrix} \cdot \begin{bmatrix} R_{11} & 0 \\ 0 & R_{22} - R_{21} \cdot R_{11}^{-1} \cdot R_{21}^* \end{bmatrix} \cdot \begin{bmatrix} I & R_{11}^{-1} \cdot R_{21}^* \\ 0 & I \end{bmatrix}. \tag{2.10}$$

The output of the algorithm is the block triangular factorization,

$$R_1 = LDL^*, \tag{2.11}$$

with a lower triangular factor L storing the columns of the form $\begin{bmatrix} I \\ R_{21} R_{11}^{-1} \end{bmatrix}$, and a block diagonal factor D storing the blocks of the form R_{11}. The task is to specify a recipe to choose an appropriate symmetric permutation (2.9) [providing the *block pivot* R_{11}] to numerically stabilize the algorithm. For example, the LAPACK [LAPACK] package, renowned for the numerical reliability of the algorithms employed uses the *Bunch-Kaufman pivoting* technique. At each step the Bunch-Kaufman algorithm scans only two columns of the matrix R_1 to determine the size m (i.e., 1 or 2) of R_{11} and the permutation matrix P_1 in (2.10)). To understand the Bunch-Kaufman technique it helps to consider the matrix $R_1 = \begin{bmatrix} r_{11} & \cdots & r^*_{t1} & \cdots & \cdots & \cdots \\ \vdots & & \vdots & & & \\ r_{t1} & \cdots & r_{tt} & \cdots & \sigma^* & \cdots \\ \vdots & & \vdots & & & \\ \vdots & & \sigma & & & \\ \vdots & & \vdots & & & \end{bmatrix}$,

and to note that the pivot R_{11} is either r_{11} or r_{tt} or $\begin{bmatrix} r_{11} & r^*_{t1} \\ r_{t1} & r_{tt} \end{bmatrix}$.

<u>The Bunch-Kaufman algorithm [BK77]</u>

```
$\alpha = (1+\sqrt{17})/8$
$\lambda = |r_{t,1}| = \max\{|r_{2,1}|, ..., |r_{n,1}|\}$                    %This step determines $t$
if $\lambda \neq 0$
    if $|r_{1,1}| \geq \alpha\lambda$
            $m = 1$; $P = I$
    else
            $\sigma = |r_{p,t}| = \max\{|r_{1,t}|, ..., |r_{t-1,t}|, |r_{t+1,t}|, ..., |r_{n,t}|\}$
            if $\sigma|r_{1,1}| \geq \alpha\lambda^2$
                    $s = 1$; $P = I$
            else if $|r_{t,t}| \geq \alpha\sigma$
                    $m = 1$ and choose $P$ so $(P \cdot R_1 \cdot P^*)_{1,1} = r_{t,t}$
            else
                    $m = 2$ and choose $P$ so $(P \cdot R_1 \cdot P^*)_{2,1} = r_{t,p}$
            end
    end
end
```

The value of the constant $\alpha = (1+\sqrt{17})/8$ is determined to bound the element growth [BK77], [B71]; this method was recently shown to be backward stable in [Hig95].

3 Exploiting structure: Fast GE with fast partial pivoting (Fast GEPP)

3.1 Basic examples of structure

In many applications in mathematics and engineering the underlying physical properties of the problem introduce various patterns of structure into arising matrices, some of these structures are shown in Table 3. It turns out that many problems involving these and more general patterns of structure can be nicely approached by using the unifying concept of *displacement*. This concept was introduced in [FMKL79] [KKM79] first in connection with Toeplitz matrices, and then the range of application of this approach was significantly extended in [HR84] to attack the other patterns of structure including those in Table 3. We next briefly recall these now well-known observations and definitions.

3.2 Displacement structure

For each of the above patterns of structure it is possible to choose a pair of auxiliary matrices $\{F, A\}$ to define a *displacement operator* $\nabla_{\{F,A\}}(\cdot) : \mathbf{C}^{n\times n} \to \mathbf{C}^{n\times n}$ of the form

$$\nabla_{\{F,A\}}(R) = FR - RA. \tag{3.1}$$

The point is that for each class of matrices with a particular pattern of structure, such as Toeplitz, Vandermonde, etc., the pair $\{F, A\}$ can be chosen to map matrices from this class to low rank matrices. Moreover, the number

$$\alpha_{\{F,A\}}(R) = \text{rank}(FR - RA)$$

Toeplitz, $T = \left[\ t_{i-j}\ \right]$ $\begin{bmatrix} t_0 & t_{-1} & \cdots & \cdots & t_{-n+1} \\ t_1 & t_0 & t_{-1} & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & t_0 & t_{-1} \\ t_{n-1} & \cdots & \cdots & t_1 & t_0 \end{bmatrix}$	Hankel, $H = \left[\ h_{i+j-2}\ \right].$ $\begin{bmatrix} h_0 & h_1 & h_3 & \cdots & h_{n-1} \\ h_1 & h_2 & & \cdot^{\cdot^{\cdot}} & h_n \\ h_2 & & \cdot^{\cdot^{\cdot}} & \cdot^{\cdot^{\cdot}} & \vdots \\ \vdots & h_{n-1} & h_n & & h_{2n-3} \\ h_{n-1} & h_n & \cdots & h_{2n-3} & h_{2n-2} \end{bmatrix}$
Vandermonde, $V = \left[\ x_i^{j-1}\ \right]$ $\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{bmatrix}$	Cauchy, $C = \left[\ \frac{1}{x_i-y_j}\ \right]$ $\begin{bmatrix} \frac{1}{x_1-y_1} & \cdots & \frac{1}{x_1-y_n} \\ \vdots & & \vdots \\ \frac{1}{x_n-y_1} & \cdots & \frac{1}{x_n-y_n} \end{bmatrix}$
Polynomial Vandermonde, $V_P = \left[\ P_{j-1}(x_i)\ \right]$ $\begin{bmatrix} P_0(x_1) & P_1(x_1) & \cdots & P_{n-1}(x_1) \\ \vdots & \vdots & & \vdots \\ P_0(x_n) & P_1(x_n) & \cdots & P_{n-1}(x_n) \end{bmatrix}$ where $\deg P_k(x) = k$	Pick, $P = \left[\ \frac{\varphi_i^* \cdot J \cdot \varphi_j}{z_i+z_j^*}\ \right]$ $\begin{bmatrix} \frac{\varphi_1 \cdot J \cdot \varphi_1^*}{z_1+z_1^*} & \cdots & \frac{\varphi_1 \cdot J \cdot \varphi_n^*}{z_1+z_n^*} \\ \vdots & & \vdots \\ \frac{\varphi_n \cdot J \cdot \varphi_1^*}{z_n+z_1^*} & \cdots & \frac{\varphi_n \cdot J \cdot \varphi_n^*}{z_n+z_n^*} \end{bmatrix}$ where $\varphi_k \in \mathbf{C}^{1\times\alpha}$, and $J \in \mathbf{C}^{\alpha\times\alpha}$

Table 3: *Some examples of matrices with structure.*

is called a $\nabla_{\{F,A\}}$-displacement rank.

For example, we started the paper with an example involving a Vandermonde matrix, $V(x)$, for this pattern of structure we can choose $F = D_x = \operatorname{diag}(x_1, \ldots, x_n)$, $A = Z_1$, the cyclic lower shift matrix having ones on the first subdiagonal and in the (1,n) position, and zeros elsewhere. Then we have

$$\nabla_{\{D_x,Z_1\}}(V(x)) = D_x V(x) - V(x) Z_1 = \begin{bmatrix} x_1^n - 1 \\ \vdots \\ x_n^n - 1 \end{bmatrix} \left[\ 0 \quad \cdots \quad 0 \quad 1\ \right]. \tag{3.2}$$

Briefly, ordinary Vandermonde matrices have $\nabla_{\{D_x,Z_1\}}$-displacement rank one. If the $\nabla_{\{D_x,Z_1\}}$-displacement rank of R is bigger than one, but still much smaller than the size of the matrix, then it is natural to refer to such R as a *Vandermonde-like* matrix. Such Vandermonde-like matrices appear in several applications, e.g., in inverse spectral problems for matrix polynomials [GLR82].

Similar justifications can be given for the names of the other *"-like"* matrices, listed in Table 4. In this survey we will be especially interetsed in Cauchy-like matrices, defined via the displacement equation $\nabla_{\{D_x,D_y\}}(R) = D_x R - R D_y$, so let us first observe that Pick matrices, $P = \left[\ \frac{\varphi_i J \varphi_j^*}{z_i+z_j^*}\ \right]$, defined in Table 3 are Cauchy-like:

$$\nabla_{\{D_z,-D_z^*\}} = D_z P + P D_z^* = \begin{bmatrix} \varphi_1 \\ \vdots \\ \varphi_n \end{bmatrix} J \left[\ \varphi_1^* \quad \cdots \quad \varphi_n^*\ \right].$$

Indeed, their $\nabla_{\{D_z,-D_z^*\}}$-displacement rank is equal to α, the length of the involved row vectors φ_k. As we shall recall in Sec. 6.1, the displacement rank α of a Pick matrix is the size of the associated $\alpha \times \alpha$ rational matrix function $W(z)$, and it is usually much smaller than the size of the Pick matrix P itself, which will be equal to the number of tangential interpolation conditions we wish $W(z)$ to satisfy.

[4]The displacement structure approach has been used, e.g., in [HJR88], [GK89], [SLAK93] to study the ordinary Toeplitz-plus-Hankel matrices. In [GKO95] algorithms for general Toeplitz-plus-Hankel-like matrices have been designed.

[4]The fact that Vandermonde matrices has displacement rank one was first observed in [HR84], but here we follow [GO94c]

Name	Choice of F	Choice of A
Toeplitz–like [FMKL79] [KKM79], [AG90], [GO92], [GO94c]	$F = Z_\gamma$	$A = Z_\delta$
Hankel–like	$F = Z_\gamma$	$A = Z_\delta^T$
Toeplitz–plus–Hankel–like [GKO95][3]	$F = Y_{\gamma_1,\delta_1}$	$A = Y_{\gamma_2,\delta_2}$
Cauchy–like [HR84]	$F = \text{diag}(c_1, ..., c_n)$,	$A = \text{diag}(d_1, ..., d_n)$
Vandermonde–like [HR84], [GO94c][4]	$F = \text{diag}(\frac{1}{x_1}, ..., \frac{1}{x_n})$	$A = Z_\gamma$
Chebyshev–Vandermonde–like [KO95]	$F = \text{diag}(x_1, ..., x_n)$	$A = Y_{\gamma,\delta}$
three-term–Vandermonde–like [KO97a]	$F = \text{diag}(x_1, ..., x_n)$	Comrade matrix
polynomial–Vandermonde–like [KO97a]	$F = \text{diag}(x_1, ..., x_n)$	Confederate matrix
Chebyshev–Hankel–like [KO97b]	Comrade matrix	Comrade matrix
Polynomial–Hankel–like [KO97b]	Confederate matrix	Confederate matrix

Table 4: *Basic examples of displacement structure with choices for F and A in (3.1). Here Z_γ is the lower shift γ–circulant matrix, $Y_{\gamma,\delta} = Z_0 + Z_0^T + \gamma e_1 e_1^T + \delta e_n e_n^T$, and the comrade and confederate matrices are defined as in [MB79] [we recall these definitions in Sec. 7.7.4 below].*

3.3 Generators and design of fast algorithms

Returning to the fact that all matrices in Table 3 are defined by a small number $O(n)$ of parameters, one could comment that the dependence of the entries upon these parameters is quite different for each particular pattern. The importance of the displacement rank concept is that it unifies such a dependence, as described next. Let $\alpha \triangleq \text{rank}\nabla_{\{F,A\}}(R)$, then one can factor [nonuniquely]

$$\nabla_{\{F,A\}}(R) = FR - RA = GB^T, \tag{3.3}$$

where both rectangular matrices on the right-hand side of (3.3) have only α columns each: $G, B \in \mathbf{C}^{n\times\alpha}$. The pair $\{G, B\}$ is called a $\nabla_{\{F,A\}}$*-generator* of R. Thus we see that 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 $\{G, B\}$. This gives us a clue on how to exploit the structure to devise a certain fast algorithm we may need:

$$\boxed{\textit{translate operations on entries of } R \textit{ to the language of operations on its generator } \{G, B\}.}$$

This approach can be used to efficently obtain fast implementations for variuos algorithms, and in the next subsection it is used to speed-up the Gaussian elimination procedure.

3.4 Exploiting the displacement structure to speed-up the block GE procedure

As is well-known, the classical Schur algorithm [S17] for checking if the analytic function is bounded by unity in the interior of unit disk can be seen as a *fast* $O(n^2)$ method to compute Cholesky factorization $T = LL^*$ for a certain positive definite Toeplitz matrix T. See, e.g., [K86], [K87] and Ch.1 for details. Two years after [S17], Nevanlinna [N19] applied a clever modification of this algorithm to solve a closely related, now classical, scalar Nevanlinna-Pick interpolation problem, see, e.g., [A65] and the references therein. It is widely known as well that the classical Nevanlinna algorithm can be seen as a *fast* $O(n^2)$ method to compute the Cholesky factorization for the associated Pick matrix, $P = \left[\frac{1-f_i f_j^*}{z_i + z_j^*}\right]$ [recall that the Nevanlinna-Pick problem is solvable if and only if the Pick matrix is positive definite, which ensures the existence of the Cholesky factorization $P = LL^*$].

and use a slightly different form (3.2) for displacement equation [i.e., with Z_1 instead of Z_0, because this form is more convenient for transformations of Vandermonde-like matrices to Toeplitz-like or Cauchy-like matrices defined next]. See [GO94c] or Sec. 7.1 below for details.

Now note that the usual Toeplitz or Pick matrices are just two particular examples of matrices with displacement structure [in both cases the corresponding displacement rank is ≤ 2], and recall that the triangular factorization can be done in $O(n^2)$ operations not only for these two particular patterns of structure, but for all kinds of displacement structure listed in Table 4. The crucial fact [explicitly first noted perhaps by M.Morf, e.g., in [M80]] that makes the speed-up possible can be vaguely formulated as follows:

$$\boxed{\textit{the displacement structure of a matrix is inherited under Schur complementation.}}$$

Hence the successive Schur complementation (2.1) of the Gaussian elimination procedure

$$\begin{tikzcd} R_1 \arrow[r] \arrow[d] & R_2 \arrow[r] \arrow[d] & \ldots \arrow[r] & R_n \arrow[d] \\ \{l_1, u_1\} & \{l_2, u_2\} & \ldots & \{l_n, u_n\} \end{tikzcd} \tag{3.4}$$

can be translated to the language of operations on generators,

$$\begin{tikzcd} \{G_1, B_1\} \arrow[r] \arrow[d] & \{G_2, B_2\} \arrow[r] \arrow[d] & \ldots \arrow[r] & \{G_n, B_n\} \arrow[d] \\ \{l_1, u_1\} & \{l_2, u_2\} & \ldots & \{l_n, u_n\} \end{tikzcd} \tag{3.5}$$

Algorithms implementing in $O(n^2)$ operations [or faster] the generator recursion (3.5) have been called *fast Cholesky* [or, in the non-symmetric case, *fast GE*] algorithms , but now they are more often referred to as *Schur-type* and *generalized Schur algorithms.* It is perhaps impossible to list all relevant connections, the analysis of algorithmic issues (first for Toeplitz-like matrices) starts with a breakthrough work of Martin Morf in [M74], [M80], [D82] followed by, e.g., [LAK84], [LAK86], [GKKL87], [CKLA87], [C89], [KS92], [S92], [GO94b], [GKO95], [KO95], [BKO95b], [KS95], [LA96], [KO97a], [KO97b] among others. Any of these forms as well as of other possible extensions can be used for our purposes, it will be convenient to use here the following variant of a generator recursion admitting a simple one-line-proof [see Sec. 4.4, 4.5 for an interpolation meaning of this result].

Lemma 3.1 *([GO93], [GO94b], [GKO95]) Let matrix R_1 be partitioned $R_1 = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix}$, where we assume the upper left $m \times m$ block R_{11} to be nonsingular. If R_1 satisfies the displacement equation*

$$\nabla_{\{F_1,A_1\}}(R_1) = \begin{bmatrix} F_{11} & 0 \\ * & F_2 \end{bmatrix} \cdot R_1 - R_1 \cdot \begin{bmatrix} A_{11} & * \\ 0 & A_2 \end{bmatrix} = G_1 \cdot B_1, \qquad (G_1 \in \mathbf{C}^{n\times\alpha}, B_1 \in \mathbf{C}^{\alpha\times n}). \tag{3.6}$$

Then the Schur complement $R_2 = R_{22} - R_{21}R_{11}^{-1}R_{12}$ satisfies the displacement equation

$$F_2 \cdot R_2 - R_2 \cdot A_2 = G_2 \cdot B_2, \tag{3.7}$$

with

$$\begin{bmatrix} 0 \\ G_2 \end{bmatrix} = G_1 - \begin{bmatrix} I_m \\ R_{21}R_{11}^{-1} \end{bmatrix} \cdot G_{11}, \qquad \begin{bmatrix} 0 & B_2 \end{bmatrix} = B_1 - B_{11} \cdot \begin{bmatrix} I_m & R_{11}^{-1}R_{21} \end{bmatrix}, \tag{3.8}$$

where G_{11} and B_{11} are the first m rows of G_1 and the first m columns of B_1, respectively.

Proof. For your pleasure: from (3.6) and the standard Schur complementation formula,

$$R_1 = \begin{bmatrix} I_m & 0 \\ R_{21}R_{11}^{-1} & I_{n-m} \end{bmatrix} \cdot \begin{bmatrix} R_{11} & 0 \\ 0 & R_2 \end{bmatrix} \cdot \begin{bmatrix} I_m & R_{11}^{-1}R_{12} \\ 0 & I_{n-m} \end{bmatrix},$$

it follows that

$$\begin{bmatrix} F_{11} & 0 \\ * & F_2 \end{bmatrix} \cdot \begin{bmatrix} R_{11} & 0 \\ 0 & R_2 \end{bmatrix} - \begin{bmatrix} R_{11} & 0 \\ 0 & R_2 \end{bmatrix} \cdot \begin{bmatrix} A_{11} & * \\ 0 & A_2 \end{bmatrix} =$$

$$\begin{bmatrix} I_m & 0 \\ -R_{21}R_{11}^{-1} & I_{n-m} \end{bmatrix} \cdot B_1 \cdot G_1 \cdot \begin{bmatrix} I_m & -R_{11}^{-1}R_{12} \\ 0 & I_{n-m} \end{bmatrix}.$$

Equating the (2,2) block entries, we obtain (3.8).

◊

3.5 Fast implementation of partial pivoting.

Formulas (3.8) is one possible implementation of the generator recursion (3.5) allowing one to speed-up the Gaussian elimination procedure. Since pivoting is not incorporated at the moment, this would be a fast implementation of a numerically inaccurate algorithm, cf. with Sec. 2.1, 2.2. Applying partial pivoting requires replacing of the maximum magnitude entry in the first column of R_1 to the (1,1) position using row interchange, or equivalently, by multiplication with the corresponding permutation matrix P_1, and then performing elimination :

$$P_1 \cdot R_1 = \begin{bmatrix} 1 & 0 \\ \frac{1}{d_1} l_1 & I \end{bmatrix} \cdot \begin{bmatrix} d_1 & u_1 \\ 0 & R_2 \end{bmatrix}. \tag{3.9}$$

Since we assume that R_1 satisfies (3.6), its permuted version, $P_1 R_1$, will satisfy

$$(P_1 F_1 P_1^T)(P_1 R_1) - (P_1 R_1) A_1 = (P_1 G_1) B_1^T. \tag{3.10}$$

Since we can run the generator recursion (3.8) only for lower triangular $(P_1 F_1 P_1^T)$, and since we do not have any a-priori restrictions on the permutation P_1, we have to assume F_1 to be a diagonal matrix. Summarizing, a row interchange does not destroy the Cauchy–like, Vandermonde–like, Chebyshev–Vandermonde–like and polynomial-Vandermonde-like displacement structures. In this case pivoting can be incorporated, and it is equivalent to the update

$$F_1 \leftarrow P_1 \cdot F_1 \cdot P_1^T, \qquad G_1 \leftarrow P_1 \cdot G_1,$$

leading to the computational procedure, formulated next.

<u>Fast GEPP for a structured matrix with diagonal F_1</u>

INPUT: A $\nabla_{\{F_1,A_1\}}$-generator $\{G_1, B_1\}$ in

$$F_1 R_1 - R_1 A_1 = G_1 B_1 \tag{3.11}$$

with diagonal F_1.

OUTPUT: The permuted triangular factorization $PR = LU$.

1. First recover from the generator the first column of R_1. The computations for doing this depend on the form of the matrices F_1 and A_1 in displacement equation (3.11), but for each kind of structure it is immediate to write down the corresponding formula, see, e.g., [GKO95], [KO95], [KO97a]. For example, for Cauchy-like matrices in $D_x R_1 - R_1 D_y = G_1 B_1$, we have $r_{ij} = \frac{g_i \cdot b_j}{x_i - y_j}$, where g_i is the i-th row of G_1 and b_j is the j-th column of B_1.

2. Next determine the position, say (k,1), of the entry with maximal magnitude in the recovered first column. Let P_1 be a permutation of the 1-st and the k-th entries. Interchange the 1-st and the k-th diagonal entries of F_1 in (1.2); interchange the 1-st and k-th rows in the matrix G_1 in (1.2).

3. Then recover from the generator the first row of $P_1 \cdot R_1 = \begin{bmatrix} d_1 & u_1 \\ l_1 & R_{22}^{(1)} \end{bmatrix}$. Now store $\begin{bmatrix} 1 \\ \frac{1}{d_1} l_1 \end{bmatrix}$ as the first column of L and $\begin{bmatrix} d_1 & u_1 \end{bmatrix}$ as the first row of U in the LU factorization of the permuted matrix, $P_1 \cdot R_1$ in (3.9).

4. Next compute by (3.8) a generator of the Schur complement $R_2 = R_{22}^{(1)} - \frac{1}{d_1} l_1 u_1$ of $P_1 \cdot R_1$ in (3.9).

5. Proceed recursively with R_2 which is now represented its generator $\{G_2, B_2\}$ to finally obtain the factorization $R_1 = P \cdot L \cdot U$, where $P = P_1 \cdot P_2 \cdot ... \cdot P_{n-1}$ with P_k being the permutation used at the k-th step of the recursion.

We refer to [GO93], [GO94b], [GKO95], [KO97a], [KO97a] for many computed examples, some of these examples will be reproduced below.

3.6 A problem: non-optimality for Hermitian matrices

It has been just shown that GEPP can be efficiently implemented to rapidly factorize in $O(n^2)$ operations Vandermonde-like, polynomial-Vandermonde-like and Cauchy-like matrices. In many applications however, Cauchy-like matrices are usually *Hermitian* [e.g., Pick matrices in Table 3], satisfying

$$\nabla_A(R) = A^*R + RA = GJG^*, \tag{3.12}$$

or

$$\nabla_F(R) = R - FRF^* = GJG^*. \tag{3.13}$$

Of course, partial pivoting destroys the symmetry of such matrices, so the above fast GEPP algorithm is not optimal for the Hermitian case.

To fully exploit the Hermitian structure of non-structured matrices one usually uses diagonal pivoting, see, e.g., [GVL96]; this allows to obtain twice as efficient algorithm in terms of both speed and memory. In Sec. 5 we will design its fast implementation for Hermitian Cauchy-like matrices. However, before doing so we describe next an important application of partial pivoting to rational interpolation problems.

4 Partial pivoting and accurate solution of tangential interpolation problems

4.1 State-space method

In the early 1960's R.E.Kalman introduced a state-space method to study linear time-invariant dynamical systems. This method is based on a representation of the transfer function for such systems, i.e., a rational $m \times m$ matrix function $W(z) = \left[\frac{p_{ij}(z)}{q_{ij}(z)} \right]$, in the form called a *realization*,

$$W(z) = D + C(zI - A)^{-1}B, \tag{4.1}$$

with the matrices $D \in \mathbf{C}^{\alpha\times\alpha}$, $C \in \mathbf{C}^{\alpha\times n}$, $A \in \mathbf{C}^{n\times n}$, $B \in \mathbf{C}^{n\times\alpha}$ of appropriate sizes. The realization is called *minimal*, if the size of A is the smallest possible.

Although the original interest in the state-space method was for specific engineering applications [e.g., the linear quadratic regulator problem and the Kalman filter], the state-space method turned out to be fundamental, leading to new insights in many other directions. Various engineering and mathematical problems have been addressed in this way; however, in this section we restrict ourselves only to several rational matrix interpolation problems, such as those of Lagrange-Sylvester, Schur, Nevanlinna-Pick, Caratheodory-Fejer, Nehari and Nehari-Takagi. We refer to a monograph [BGR90] for a systematic treatment of all these interpolation problems, as well as for their use to solve various engineering problems [sensitivity minimization, model reduction, robust stabilization], and for a wide list of references and historical remarks. Given such a vast literature on interpolation, we have to explain the need for the further analysis, especially in the simplest frameworks taken in this section. The point is that although there are a number of explicit solutions for a variety of such problems, the numerical properties of the corresponding computational schemes were totally ignored. As a result, there is a clear gap in practically reliable software tools for solving even simplest rational interpolation problems. In this section we show how the state-space method can provide not only mathematical descriptions, but also accurate and fast practical algorithms to compute solutions.

The attractive property of the representation (4.1) is that it can usually reduce a certain analytic problem involving $W(z)$ to just a linear algebra problem, i.e. to just standard manipulations on finite matrices, such as inversion, triangular factorization, solving the associated linear system, etc. Such a reduction seems to be beneficial for practical purposes, because it allows us to compute solutions for a variety of applied problems by using available standard linear algebra software tools. However, there are two difficulties. First, the size of matrices can be quite large, so computing the solution via general purpose algorithms may require substantional storage and time. As we shall see in a moment, particular interpolation problems often introduce a displacement structure into the corresponding matrices. This structure, of course, can be exploited to speed-up the computation, however, without any stabilizing technique such algorithms would be impractical, leading to the rapid computing of a numerically inaccurate solution. As we shall show the fast GEPP algorithm of Sec. 3.5, and is applicable, and it provides not only fast, but also more accurate, solutions.

To sum up, fast implementations of various pivoting techniques have a variety of important applications improving the accuracy of many algorithms for rational matrix interpolation and for solving several engineering problems, listed, e.g., in [BGR90].

4.2 Simplest example: homogeneous tangential interpolation problem and Cauchy-like matrices

In fact, in a variety of interpolation problems a rational matrix function $W(z)$ is often not given by the four matrices $\{A, B, C, D\}$ in (4.1), but by some other set of matrices, similar to the situation in the next example.

The simplest homogeneous tangential interpolation problem

Given :

n distinct points $\{x_1, \dots, x_n\}$ and n nonzero $1 \times \alpha$ row vectors $\{\varphi_1, \dots, \varphi_n\}$,
n distinct points $\{y_1, \dots, y_n\}$ and n nonzero $\alpha \times 1$ column vectors $\{\psi_1, \dots, \psi_n\}$.

[Let us assume $\{x_k, y_k\}$ to be $2n$ distinct complex numbers.]

Construct: A rational $\alpha \times \alpha$ matrix function $W(z)$ having the identity value at infinity, and such that

- *Left tangential interpolation condition*: det $W(z)$ has a simple zero at x_k, and
$$\varphi_k W(x_k) = 0. \tag{4.2}$$
- *Right tangential interpolation condition*: det $W(z)^{-1}$ has a simple zero at y_k, and
$$W(y_k)^{-1}\psi_k = 0. \tag{4.3}$$

Solvability condition: The unique [in this simples case] solution R of the displacement equation

$$\boxed{RA_\pi - A_\zeta R = B_\zeta C_\pi} \tag{4.4}$$

should be invertible, where we define

$$C_\pi = \begin{bmatrix} \psi_1 & \dots & \psi_n \end{bmatrix}, \qquad A_\pi = \text{ diag } \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix},$$
$$A_\zeta = \text{ diag } \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}, \qquad B_\zeta = \begin{bmatrix} \varphi_1^T & \dots & \varphi_n^T \end{bmatrix}^T,$$

Solution , as given in [BGR90]:

$$\boxed{W(z) = I + C_\pi (zI - A_\pi)^{-1} R^{-1} B_\zeta.} \tag{4.5}$$

So, the eigenvalues of A_π are the poles, and the eigenvalues of A_ζ are the zeros of $W(z)$ [explaining the choice for the subscripts to denote these matrices]. We see that in order to *couple* the *null data* $\{A_\zeta, B_\zeta\}$ and the *pole data* $\{C_\pi, A_\pi\}$ we need to bring into the consideration the displacement equation (4.4), whose solution R has been called a *null-pole coupling matrix* in [BGR90]. Now notice that the entries of R are not a part of the given data, and in this particular problem R is defined *implicitly*, by its generator $\{B_\zeta, C_\pi\}$.

4.3 Homogenuous interpolation and other classes of matrices with displacement structure

The above example with *diagonal* A_π and A_ζ [thus giving rise to a Cauchy-like matrix R in (4.5)] is simplest but by no means separate. In fact, the monograph [BGR90] is a manifestation of the application of explicit formula (4.5) to solve a wide variety of interpolation problems, including those listed at the beginning of Sec. 4.1. In all these problems a structured matrix R is also defined implicitly, by the two pairs of matrices, the "pole pair" $\{C_\pi, A_\pi\}$ for a minimal realization

$$W(z) = I + C_\pi (zI - A_\pi)^{-1} B_\pi, \tag{4.6}$$

[with B_π unknown], and another, "null pair" $\{A_\zeta, B_\zeta\}$ for a minimal realization

$$W(z)^{-1} = I + C_\zeta (zI - A_\zeta)^{-1} B_\zeta,$$

[now with C_ζ unknown]. As above, in order to recover a rational matrix function $W(z)$ from the null data $\{A_\zeta, B_\zeta\}$ and the pole data $\{C_\pi, A_\pi\}$, we need a *null-pole coupling matrix* R defined as a solution of the displacement equation

$$RA_\pi - A_\zeta R = B_\zeta C_\pi \tag{4.7}$$

with arbitrary $\{A_\pi, A_\zeta\}$, so all special structures of Table 4 are captured. The now well-known result of [BGR90] states that for a given $W(z)$ [i.e., given by the two pairs $\{C_\pi, A_\pi\}$ and $\{A_\zeta, B_\zeta\}$] there is a unique invertible solution to (4.7) satisfying

$$W(z) = I + C_\pi (zI - A_\pi)^{-1} R^{-1} B_\zeta \tag{4.8}$$

[if there are many solutions R to (4.7), then each of them defines a different $W(z)$ via (4.8)]. The above collection of matrices

$$\tau = \{C_\pi, A_\pi, A_\zeta, B_\zeta, R\} \tag{4.9}$$

has been called called a *global left null-pole triple*[5] for $W(z)$.

However, this seems to be a typical example of a situation when there is a complete mathematical description of a solution for an important class of problem, but there are no numerical algorithms to compute these solutions with good accuracy and rapidly. Indeed, the representation (4.7) is not particularly convenient for practical purposes [if we need to further evaluate $W(z)$], because (4.8) explicitly involves the inverse of R.

We next follow [GO93] [GO94c] to present two fast and accurate algorithms to compute more convenient representations for $W(z)$. Basically, there are two methods to replace (4.8) by more appropriate forms. First, we can solve α linear systems of equations,

$$RB_\pi = B_\zeta, \tag{4.10}$$

to replace (4.8) by (4.1). This approach does not require any further explanations, and we only refer to many computed examples in [GO94c], where R had a Hermitian partially reconstructible Cauchy-like structure [imposed by the Nehari problem, to be described later in Sec. 6.5]. In that context (4.10) was solved via the slow GECP [complete pivoting], and via the fast algorithms for solving Cauchy-like linear equations, including those described in Sec. 3.5 [based on partial pivoting] and those to be described later in Sec. 5.3, 6.3 [based on diagonal pivoting]. The conclusions were that without pivoting such fast methods are inaccurate, and the slow GECP provides much better results. However there is no need to "pay by the speed for accuracy", and a fast implementation of partial and diagonal pivoting techniques was shown to enhance the numerical behavior of these fast methods, so they provide the same accuracy as slow GECP, while computing the solution much faster.

We next describe how to obtain one more convenient representation for $W(z)$.

[5]The number of matrices in the triple shows a tendency to grow over the years. Indeed, in [GLR82] triple still consisted of three matrices.

4.4 Factorization of rational matrix functions

Another more convenient than (4.8) representation of $W(z)$ would be its cascade decomposition,

$$W(z) = \Theta_1(z) \cdot \ldots \cdot \Theta_n(z) \qquad \text{with} \qquad \Theta_k(z) = I + c_k \frac{1}{z - x_1} \frac{1}{d_k} b_k, \tag{4.11}$$

i.e., $\Theta_k(z)$ are the firs-order factors, having just one pole and one zero. Since c_k is just a $\alpha \times 1$ column, d_k is a scalar, and b_k is just a $1 \times \alpha$ row, the representation (4.11) is very attractive, because it allows us to evaluate $W(z)$ in only $O(\alpha n)$ operations. This motivates us to look for a numerically reliable fast algorithm to compute a factorization for rational matrix functions. As we shall see in a moment, such a factorization of $W(z)$ is just a Gaussian elimination on its null-pole coupling matrix R [this fact is known, see, e.g., [S86]], and the point is that fast implementations of several pivoting techniques are to be employed to achieve accuracy.

To present an algorithm, we need to recall several widely known results. There is a large number of theorems, e.g., in [P60], [S86], [BGKV80], [GVKDM83], [AD86], [AG88], [LAK92], [GO93], [SKLAC94], [GO94b] on factorization of rational matrix functions. In particular, the following statement is well-known.

Theorem 4.1 [[BGKV80][6], [BGK79], Theorem 4.8] *Let*

$$W_1(z) = I_p + C \cdot (zI_N - A)^{-1} \cdot B \tag{4.12}$$

be a minimal realization. If matrices $A \in \mathbf{C}^{N\times N}, B \in \mathbf{C}^{N\times p}, C \in \mathbf{C}^{p\times N}$ can be partitioned such that matrix A is upper triangular :

$$C = \begin{bmatrix} C_1 & C_2 \end{bmatrix}, \qquad A = \begin{bmatrix} A_1 & * \\ 0 & A_2 \end{bmatrix}, \qquad B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix},$$

and matrix $A^\times = A - B \cdot C$ is lower triangular :

$$A^\times = \begin{bmatrix} A_1^\times & 0 \\ * & A_2^\times \end{bmatrix},$$

then the rational matrix function $W_1(z)$ admits a minimal factorization $W_1(z) = \Theta_1(z) \cdot W_2(z)$, where

$$\Theta_1(z) = I_p + C_1 \cdot (zI_{N_1} - A_1)^{-1} \cdot B_1, \qquad W_2(z) = I_p + C_2 \cdot (zI_{N_2} - A_2)^{-1} \cdot B_2. \tag{4.13}$$

Moreover, each minimal factorization of $W_1(z)$ can be obtained in this way.

Here we have only to explain the name "*minimal factorization*" for $W_1(z) = \Theta_1(z) \cdot W_2(z)$. This simply means that there is no "zero-pole cancelation", i.e., the size of A in (4.12) is equal to the sum of the sizes of A_1 and A_2 in (4.13).

The above theorem factorizes $W_1(z)$ given in the form of realization (4.12). This result is not appropriate for our purpose, because in our applications $W_1(z)$ will be given in the form (4.8), i.e., by its global left null-pole triple (4.9). The following theorem describes a variant of the above theorem in the desired form.

Theorem 4.2 [[GO93] [GO94c], Theorem 2.3] *Let $\tau_1 = (C_\pi, A_\pi, A_\zeta, B_\zeta, R_1)$ be a global left null-pole triple for rational matrix function*

$$W_1(z) = I + C_\pi(zI - A_\pi)^{-1} R_1^{-1} B_\zeta,$$

with identity value at infinity. Let matrices in τ_1 be partitioned as

$$C_\pi = \begin{bmatrix} C_{\pi,1}, & C_{\pi,2} \end{bmatrix}, \qquad A_\pi = \begin{bmatrix} A_{\pi,1} & * \\ 0 & A_{\pi,2} \end{bmatrix},$$

[6]Interestingly, the paper [BGKV80] followed a curious situation, in which Bart-Gohberg-Kaashoek and P.Van Dooren reported this [independently obtained] result in two consecutive talks.

$$A_\zeta = \begin{bmatrix} A_{\zeta,1} & 0 \\ * & A_{\zeta,2} \end{bmatrix}, \qquad B_\zeta = \begin{bmatrix} B_{\zeta,1} \\ B_{\zeta,2} \end{bmatrix}, \tag{4.14}$$
$$R_1 = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix}.$$

If matrix R_{11} is invertible then $W_1(z)$ admits a minimal factorization $W_1(z) = \Theta_1(z) \cdot W_2(z)$, where

$$\tau_\Theta = (C_{\pi,1}, A_{\pi,1}, A_{\zeta,1}, B_{\zeta,1}, R_{11}) \tag{4.15}$$

is a global left null-pole triple for

$$\Theta_1(z) = I_p + C_{\pi,1} \cdot (zI_{N_1} - A_{\pi,1})^{-1} \cdot R_{11}^{-1} \cdot B_{\zeta,1}, \tag{4.16}$$

and

$$\tau_2 = (C_{\pi,2} - C_{\pi,1} \cdot R_{11}^{-1} \cdot R_{12}, \quad A_{\pi,2}, \quad A_{\zeta,2}, \quad B_{\zeta,2} - R_{21} \cdot R_{11}^{-1} \cdot B_{\zeta,1}, \quad R_{22} - R_{21} \cdot R_{11}^{-1} \cdot R_{12}) \tag{4.17}$$

is a global left null-pole triple for

$$W_2(z) = I_p + (C_{\pi,2} - C_{\pi,1} \cdot R_{11}^{-1} \cdot R_{12})(zI_{N_2} - A_{\pi,2})^{-1} \cdot (R_{22} - R_{21} \cdot R_{11}^{-1} \cdot R_{12})^{-1} \cdot (B_{\zeta,2} - R_{21} \cdot R_{11}^{-1} \cdot B_{\zeta,1}) \tag{4.18}$$

4.5 Computational aspects of theorems 4.1-4.2

Here we address computational implications of theorem 4.2, and observe that it leads to a recursive algorithm for factorization of rational matrix functions, and that this algorithm in fact coincides with the simple recursion of Lemma 3.1. To clarify this, let us make the following three comments.

- For a given $W_1(z)$, its cascade decomposition (4.11) can be computed recursively: $W_1(z) \to \{\Theta_1(z), W_2(z)\}$, then $W_2(z) \to \{\Theta_2(z), W_3(z)\}$, etc. Formulas in (4.17) reduce this function recursion to an array algorithm,

$$\begin{tikzcd} \tau_1 \arrow[r] \arrow[d] & \tau_2 \arrow[r] \arrow[d] & \ldots \arrow[r] & \tau_{n-1} \arrow[r] \arrow[d] & \tau_n \arrow[d] \\ \Theta_1(z) & \Theta_2(z) & \ldots & \Theta_{n-1}(z) & \Theta_n(z) \end{tikzcd}, \tag{4.19}$$

  where one manipulates on just matrices $\{B_\zeta, C_\pi\}$ in gobal left null-pole triples τ_k.

- The factorization steps $W_k(z) = \Theta_k(z) \cdot W_{k+1}(z)$ correspond to the step of Gaussian elimination on the null-pole coupling matrix R_1, e.g.:

$$\begin{tikzcd} W_1(z) \arrow[r] \arrow[d] & W_2(z) \arrow[d] \\ R_1 & R_2 := R_{22} - R_{21}R_{11}^{-1}R_{12} \end{tikzcd}$$

  To see this just note that the null-pole coupling matrix in (4.17) is the Schur complement R_2.

  In fact, the this second comment is known, see [S86] [and earlier in Russian], however the author of [S86] was not interested in computational aspects, so his variant of a factorization theorem lead to a "superslow" $O(n^4)$ algorithm.

Now we are ready to explain how theorem 4.2 leads to a fast $O(n^2)$ factorization algorithm, and the explanation is based on its connection to Lemma 3.1. By exploring the definition (4.7) for the null-pole coupling matrices of the initial function $W_1(z)$, as well as of the quotient $W_2(z)$, specified by theorem 4.2, we have:

$$\boxed{R_1 A_\pi - A_\zeta R_1 = B_\zeta C_\pi},$$

and

$$\boxed{R_2 A_{\pi,2} - A_{\zeta,2} R_2 = (B_{\zeta,2} - R_{21} \cdot R_{11}^{-1} \cdot B_{\zeta,1})(C_{\pi,2} - C_{\pi,1} \cdot R_{11}^{-1} \cdot R_{12})}$$

where $R_2 \triangleq (R_{22} - R_{21} \cdot R_{11}^{-1} \cdot R_{12})$ is the Schur complement. By comparing these formulas to the generator recursion (3.7) (3.8) we now see that Lemma 3.1 is just an "array part" [or matrix interpretation] of Theorem 4.2. Equivalently, the two schemes (4.19) and (3.5) are the same, so that the fast GEPP algorithm of Sec. 3.5 is the algorithm we are looking for, i.e., a fast algorithm for factorization of rational matrix functions. As we shall see in Sec.6.5, 6.6, 7.5 pivoting is crucial in achieving high relative accuracy of this factorization, so we next provide an interesting interpolation interpretation for pivoting.

4.6 Interpolation and fast implementation of pivoting

Now let us turn to the simplest case of $W_1(z) = I + C_\pi(I - A_\pi)^{-1}R_1^{-1}B_\zeta$ with simple poles and zeros, i.e., the matrices $A_\zeta = \text{diag}\left[\begin{array}{ccc} x_1 & \cdots & x_n \end{array}\right]$ and $A_\pi = \text{diag}\left[\begin{array}{ccc} y_1 & \cdots & y_n \end{array}\right]$ are diagonal, so that R_1 in $R_1A_\pi - A_\zeta R_1 = B_\zeta C_\pi$ is a Cauchy-like matrix. Theorem 4.2 can be used to factorize $W_1(z) = \Theta_1(z)W_2(z)$, where the first factor has just one zero x_1 and one pole y_1:

$$\Theta_1(z) = I + C_{\pi,1}\frac{1}{z - y_1}\frac{1}{r_{11}}B_{\zeta,1}, \qquad \text{where} \qquad r_{11}y_1 - x_1r_{11} = B_{\zeta,1}C_{\pi,1},$$

i.e., with $C_{\pi,1}$ be the 1-st column of C_π, and $B_{\zeta,1}$ be the 1-st row of B_ζ. We could, of course just proceed with $W_2(z)$ recursively to compute in $O(n^2)$ operations the cascade decomposition (4.11). However, in [GO93], [GO94b] this algorithm was found to be, in general, inaccurate [it even breaks down if R_1 encounters singular leading submatrices].

As one of the methods to overcome these difficulties it was suggested in [GO94b] to apply a factorization step to the same $W_1(z)$, but now rewritten in the form

$$W_1(z) = I + C_\pi(I - A_\pi)^{-1}(PR)^{-1}(PB_\zeta),$$

which now produces a different first factor,

$$\Theta_1(z) = I + C_{\pi,1}\frac{1}{z - y_1}\frac{1}{r_{k1}}B_{\zeta,k}, \qquad \text{where} \qquad r_{k1}y_1 - x_kr_{k1} = B_{\zeta,k}C_{\pi,1},$$

with the same pole y_1 but now with a different zero x_k. The conclusion is that at each factorization step we have freedom in choosing the zero for the next factor in factorization, but the task is to suggest a heuristic to exploit this freedom to improve accuracy.

Now, the association in theorem 4.2 of factorization of $W(z)$ and GE procedure was used in [GO94b] to suggest to mimic partial pivoting on R_1 to improve the accuracy of factorization of rational matrix functions, and of many associated algorithms for rational matrix interpolation problems.

Finally note that we essentially used here the diagonal structure only for the "null" matrix A_ζ, and not for the "pole" matrix A_π. Hence partial pivoting can be efficiently implemented as well for the case when the "pole" matrix A_π has a more general form, say the Jordan form. This case corresponds to the multiplicity of the poles of $W_1(z)$ [or, equivalently to the confluency in the null-pole coupling matrix R_1], but for brevity we omit these details here.

4.7 A problem: loss of symmetry for J-unitary rational matrix functions

In many applications, however [see, e.g., Sec. 6], rational matrix functions and their null-pole coupling matrices enjoy certain symmetry properties. For example we often study *J-unitary on the imaginary line* functions of the form

$$W_1(z) = I - C_\pi(zI - A_\pi)^{-1}R_1^{-1}C_\pi^*J, \tag{4.20}$$

where R_1 is a Hermitian matrix satisfying

$$A_\pi^*R_1 + R_1A_\pi = -C_\pi^*JC_\pi, \tag{4.21}$$

see Ch. 6 in [BGR90], or[GO94b]. This is a *continuous-time* version, and its *discrete-time* counterpart encounters *J-unitarity on the unit circle* rational functions of the form

$$W_1(z) = [I + C_\pi(zI - A_\pi)^{-1}R_1^{-1}(A_\pi^{-1})^*C_\pi^*J] \cdot D_\beta,$$

where R is a Hermitian matrix satisfying

$$R_1 - A_\pi^*R_1A_\pi = -C_\pi^*JC_\pi,$$

and

$$D_\beta = I + C_\pi R_1^{-1}(I - \beta A_\pi^*)^{-1}C_\pi^*J,$$

see, e.g., [BGR90], Ch. 7. We shall be interested in accurate and fast algorithms to factorize such $W_1(z)$, and of course, general algorithms of Sec. 4.4, 4.5 are immediately applicable. However, they have the following disadvantage. Since symmetry in R_1 is preserved under Schur complementation (2.10), it is easy to see that the symmetry of $W_1(z)$ shown, e.g., in (4.20) and (4.21) is also preserved under the factorization $W_1(z) = \Theta_1(z) \cdot W_2(z)$ described in theorem 4.2, see, e.g., (4.17). This means that not only the given function $W_1(z)$ but also both its factors $\Theta_1(z), W_2(z)$ are J-unitary as well. This observation leads to an immediate computational advantage: since $B_\zeta = JC_\pi^*$ we can run the recursion only on one matrix C_π in (6.6), i.e., to save half of the arithmetic operations and half of memory locations. Unfortunately, without pivoting this scheme suffers of losses of accuracy. Partial pivoting improves accuracy, but unfortunately it destroys symmetry, so it does not lead to savings in speed and memory. This difficulty will be resolved in Sec 6.1 after we shall show next how to implement symmetry-preserving diagonal pivoting techniques.

5 Diagonal diagonal pivoting for partially reconstructible Hermitian matrices

As we just noted in Sec. 4.7 a symmetry-preserving accurate algorithms for factorizing a Hermitian structured matrix R can be obtained by replacing partial pivoting by a fast implementation of diagonal pivoting technique of Sec. 3.5. Although such a replacement seems to cause no technical difficulties, as we shall see in a moment, the symmetric case requires a more scrupulous treatment, because of the following reason.

In the literature on generalized Schur algorithms the displacement operator $\nabla_A(\cdot)$ is usually assumed to be invertible. However, in applications and for many purposes one may need triangularization algorithms for the more delicate case when displacement operator has a non-trivial kernel [such as, e.g., in in [CK91] considering ordinary Hankel matrices]. In this paper we shall meet this necessity in Sec. 7 [see also [KO97b]] when designing more accurate Hermitian Toeplitz and Toeplitz-plus-Hankel solvers, This degeneracy condition for $\nabla_A(\cdot)$ naturally appears in the context of *boundary* tangential interpolation problems [KO97b], as well as of matrix Nehari and Nehari-Takagi interpolation problems [GO93], [GO94b] and Sec. 6.5, 6.6.

We follow Georg Heinig's suggestion and associate the name *"partially reconstructible"* with the structured matrices associated with degenerate displacement operators.

5.1 Generators for partially reconstructible matrices

If the displacement operator

$$\nabla_A(R) = A^*R + RA = GJG^*, \tag{5.1}$$

is invertible, then the ∇_A-generator $\{G, J\}$ on the right-hand-side of (5.1) contains all the information on R, justifying its name as a generator of R. In this section we follow [KO97b] to discuss a more peculiar case, when $\mathcal{K} = \mathrm{Ker}\nabla_A(\cdot)$ is assumed to be nontrivial. In the latter situation a matrix R with displacement structure (5.1) will be called a *partially reconstructible* matrix, because only part of the information on R is now contained in its ∇_A-generator $\{G, J\}$. To extend the definition of a ∇_A-generator to partially reconstructible matrices, let us represent

$$R = R_{\mathcal{K}} + R_{\mathcal{K}^\perp} \tag{5.2}$$

with respect to the orthogonal decomposition $\mathbf{C}^{n\times n} = \mathcal{K} \oplus \mathcal{K}^\perp$. A partially reconstructible matrix R is uniquely determined by the three matrices $\{G, J, R_{\mathcal{K}}\}$ in (5.1), (5.2), so we shall call them a ∇_A-*generator* of R. To complete the above definition, one should specify for $\mathbf{C}^{n\times n}$ the inner product in which the decomposition (5.2) is orthogonal. Here we choose

$$< A, B >= \mathrm{tr}(B^* \cdot A), \qquad A, B \in \mathbf{C}^{n\times n}, \tag{5.3}$$

where $\mathrm{tr}(A)$ denotes the sum of all diagonal entries of A. Note that the latter inner product induces the Frobenius norm in $\mathbf{C}^{n\times n}$ $< A, A >= \sum_{i,j=1}^{n} |a_{ij}|^2 = \|A\|_F^2$.

5.2 Generators for partially reconstructible Cauchy-like matrices

When $\mathcal{K} = \mathrm{Ker}\nabla_A = \{0\}$, all the information about the n^2 entries of R is efficiently stored in only $\alpha n + 2$ entries of $\{G, J\}$. However if $\mathcal{K} = \mathrm{Ker}\nabla_A$ is nontrivial, then $R_{\mathcal{K}}$ is a full $n \times n$ matrix, and at first glance the representation of a partially reconstructible matrix R by its generator $\{G, J, R_{\mathcal{K}}\}$ is no longer efficient. As we shall see in a moment, for the main structured classes, $\mathcal{K} = \mathrm{Ker}\nabla_A$ has a low dimension, and moreover, the matrix $R_{\mathcal{K}}$ has a simple form [say, circulant for Toeplitz-like structure, or diagonal matrix for Cauchy-like structure], which in turn can be described by a smaller number of parameters. For example, for Cauchy-like matrices R we use $A = \mathrm{diag}(a_1, \ldots, a_n)$ so that

$$\mathcal{K} = \mathrm{Ker}\nabla_A = \{\mathrm{diag}(d_1, \ldots, d_n)\} \qquad \text{where} \qquad \begin{cases} d_k \text{ is arbitrary} & \text{if} \quad a_k + a_k^* = 0 \\ d_k = 0 & \text{if} \quad a_k + a_k^* \neq 0 \end{cases}.$$

So the last matrix in the ∇_A-generator $\{G, J, R_{\mathcal{K}}\}$ is a diagonal matrix

$$R_{\mathcal{K}} = \mathrm{diag}(d_1, \ldots, d_n) \qquad \text{where} \qquad \begin{cases} d_k = r_{kk} & \text{if} \quad a_k + a_k^* = 0 \\ d_k = 0 & \text{if} \quad a_k + a_k^* \neq 0 \end{cases} \tag{5.4}$$

i.e., it is formed from the diagonal entries $\{r_{kk}\}$ of R.

Similarly, a generator for partially reconstructible Hermitian Toeplitz-like and Toeplitz-plus-Hankel-like matrices will be specified later when they will be needed in Sec. 7.4 and Sec. 7.7.2.

5.3 Symmetric GE for partially reconstructible Cauchy-like matrices

Similarly to the non-Hermitian case (3.4), (3.5), here we need a fast implementation for the block symmetric Schur complementation

$$\begin{tikzcd} R_1 \arrow[r] \arrow[d] & R_2 \arrow[r] \arrow[d] & \cdots \arrow[d] \\ \{R_{12}^{(1)}(R_{11}^{(1)})^{-1}, R_{11}^{(1)}\} & \{R_{12}^{(2)}(R_{11}^{(2)})^{-1}, R_{11}^{(2)}\} & \cdots \end{tikzcd} \tag{5.5}$$

for the case when R_1 is partially reconstructible. Modifying the designation in (2.10) here we use the uper script $^{(k)}$ to designate the block partitioning for the successive Schur complements R_k. Instead of the expensive $O(n^3)$ computation in (5.5), the desired fast algorithm should implement a *block* generator recursion

$$\begin{tikzcd} \{G_1, J, R_{\mathcal{K}_1}\} \arrow[r] \arrow[d] & \{G_2, J, R_{\mathcal{K}_2}\} \arrow[r] \arrow[d] & \cdots \arrow[d] \\ \{R_{12}^{(1)}(R_{11}^{(1)})^{-1}, R_{11}^{(1)}\} & \{R_{12}^{(2)}(R_{11}^{(2)})^{-1}, R_{11}^{(2)}\} & \cdots \end{tikzcd} \tag{5.6}$$

Here we denote by $R_{\mathcal{K}_k}$ the the third matrix in a generator $\{G_k, J, R_{\mathcal{K}_k}\}$ of the *partially reconstructible* matrix R_k. Recall that $R_{\mathcal{K}_k}$ is a diagonal matrix in the Cauchy-like case, see (5.4).

In fact, the desired fast algorithm for (5.6) has already been specified in the same Lemma 3.1. Indeed, its generator recursion (3.8) has two properties we need:

- It was formulated for the block Schur complementation step;
- it has been prove without any assumption on the invertibility of the corresponding displacement operator.

[we shall see later in Sec. 6.4 an interesting interpolation interpretation [GO93], [GO94b] for these two conditions].

Thus, Lemma 3.1 is applicable and it suggests a recursion for G_k. Therefore the only point to clarify here is how to implement the recursion for the diagonal matrices $R_{\mathcal{K}}^{(k)}$ in $\{G_k, J, R_{\mathcal{K}}^{(k)}\}$ for $k = 2, 3, \ldots, n$. Since $R_{\mathcal{K}_k}$ describes an "unstructured" part of R_k there seems to be no other choice but to directly apply the structure-ignoring formula (2.10), however using it to compute *only* the first diagonal entries of R_k which are also the entries $R_{\mathcal{K}_k}$, see (5.4) [in the context of interpolation such diagonal entries were called *coupling numbers* by Andre Ran, see [GO94b] and references therein]. Fortunately this structure-ignoring computation will not slow down our algorithm, because at each recursion step we need to compute not more than n such diagonal entries. This leads us to the following algorithm, in which we do not specify a particular diagonal pivoting technique, because this algorithm can incorporate symmetric or Bunch-Kaufman pivoting [cf. Sec. 2.5], or their combination, see [KO97b]. Other choices are also possible.

Fast symmetric GE with diagonal pivoting for partially reconstructible Cauchy-like matrices

```
Input :  A $\nabla_A$-generator $\{G_1, J, R_{\mathcal{K}_1}\}$ of a Cauchy-like matrix $R_1 \in \mathbf{C}^{n\times n}$ partitioned as in (2.10), where $G_1 \in \mathbf{C}^{n\times\alpha}$, and
         $R_{\mathcal{K}_1} = \mathrm{dag}(d_1^{(1)}, \ldots, d_s^{(1)}, 0, \ldots, 0)$.
Output : The block $LDL^*$ decomposition of $R_1$.
Steps :  1. Using a particular pivoting technique, perform symmetric row and column interchanges, and determine the size $m$
            for the pivot $R_{11} \in \mathbf{C}^{m\times m}$.
            For this, recover the necessary entries of $R_1$ from $\{G_1, J, R_{\mathcal{K}_1}\}$ $r_{ij} = \frac{g_i J g_j^*}{a_i^* + a_j}$ where $g_i$ denotes the $i$-th row of $G_1$.
            The algorithm remains fast as long as we need to recover $O(n)$ entries, e.g., in symmetric and Bunch-Kaufman
            pivoting.
         2. Recover from $\{G_1, J, R_{\mathcal{K}_1}\}$ the entries of the first $m$ columns $\begin{bmatrix} R_{11} \\ R_{21} \end{bmatrix}$ of $R_1$. [Note that some of the entries $r_{ij}$ can
            be already available after the step 1.]
         3. Store the first $m\times m$ block diagonal entry $R_{11}$ of the matrix $D$ and the first $m$ columns $\begin{bmatrix} I \\ R_{21}\cdot R_{11}^{-1} \end{bmatrix}$ of the matrix
            $L$ in the $LDL^*$ decomposition of $R_1$.
         4. Compute a generator $\{G_2, J, R_{\mathcal{K}_2}\}$ of the Schur complement $R_2$ :
            4.1. Compute $G_2 \in \mathbf{C}^{(n-m)\times\alpha}$ via
                 $\begin{bmatrix} 0 \\ G_2 \end{bmatrix} = G_1 - \begin{bmatrix} I_m \\ R_{21}R_{11}^{-1} \end{bmatrix} G_{11}.$
            4.2. Compute the diagonal matrix $R_{\mathcal{K}_2}$ by applying the standard Schur complementation formula $R_2 = R_{22} -$
                 $R_{21}R_{11}^{-1}R_{21}^*$ to compute only nonzero diagonal entries of $R_2$ [these entries $d_k^{(2)}$ are those for which we have
                 $a_k + a_k^* = 0$, e.g., (5.4). Namely
                 • if $m = 1$ then $d_k^{(2)} = d_k^{(1)} - r_{k1}\frac{1}{r_{11}}r_{k1}^*$.
                 • If $m = 2$ then $d_k^{(2)} = d_k^{(1)} - \begin{bmatrix} r_{k1} & r_{k2} \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{21}^* \\ r_{21} & r_{22} \end{bmatrix}^{-1} \cdot \begin{bmatrix} r_{k1}^* \\ r_{k2}^* \end{bmatrix}$.
         5. Repeat recursively for the Schur complement $R_2$, given by its generator $\{G_2, J, R_{\mathcal{K}_2}\}$.
```

5.4 Transformations between continuous-time and discrete-time displacement operators

Each pattern of structure in Table 4 can be equivalently defined via different choices of the displacement operator. For example, along with the *continuous-time* displacement operator

$$\nabla_A(R) = A^* \cdot R + R \cdot A = G_{\nabla_A} \cdot J \cdot G_{\nabla_A}^*, \tag{5.7}$$

mainly discussed above, its *discrete-time* counterpart

$$\nabla_F(R) = R - F \cdot R \cdot F^* = G_{\nabla_F} \cdot J \cdot G_{\nabla_F}^*. \tag{5.8}$$

also often appears in applications. Therefore we need versions of fast and accurate algorithms for this important case as well. The next two lemmas indicate how to transform the equations (5.7) and (5.8) to each other without changing the solution R. These formulas will show how to easily translate a certain algorithm from one form to another. We shall also need this technique later in Sec. 7.5.

Lemma 5.1 *Let $F \in \mathbf{C}^{n\times n}$ be arbitrary, and set*

$$A = (\beta I + F) \cdot (\beta I - F)^{-1}, \tag{5.9}$$

where β is any number such that $|\beta| = 1$ chosen to guarantee the invertibility of the second factor in (5.9). *Let the displacement operators ∇_F and ∇_A be defined by* (5.8) *and* (5.7), *respectively. Then*

(i) $\text{Ker}\nabla_F = \text{Ker}\nabla_A$.

(ii) *For any $R \in \mathbf{C}^{n\times n}$, specified by the ∇_F-generator $\{G_{\nabla_F}, J, R_{\mathcal{K}}\}$, its ∇_A-generator is $\{G_{\nabla_A}, J, R_{\mathcal{K}}\}$, where*

$$G_{\nabla_A} = \sqrt{2}\cdot(\beta I - F)^{-1} G_{\nabla_F}. \tag{5.10}$$

In particular, the ∇_F-displacement rank and the ∇_A-displacement rank of R are the same.

Proof. Both assertions follow from the identity

$$2\cdot(\beta I - F)^{-1}\cdot(R - F\cdot R\cdot F^*)\cdot(\beta^* I - F^*)^{-1} =$$
$$= (\beta I - F)^{-1}\cdot(\beta I + F)\cdot R + R\cdot(\beta^* I + F^*)\cdot(\beta^* I - F^*)^{-1}.$$

$\diamondsuit$

The next lemma is the converse to Lemma 5.1, and it is deduced by exactly the same arguments.

Lemma 5.2 *Let $A \in \mathbf{C}^{n\times n}$ be arbitrary, and set*

$$F = (tI + A)^{-1}\cdot(tI - A), \tag{5.11}$$

where $t > 0$ is any number that will guarantee the invertibility of the second factor in (5.11). Let the displacement operators ∇_F and ∇_A be defined by (5.8) and (5.7), resp. Then

(i) $\text{Ker}\nabla_F = \text{Ker}\nabla_A$.

(ii) *Any $R \in \mathbf{C}^{n\times n}$ given by the ∇_A-generator $\{G_{\nabla_A}, J, R_{\mathcal{K}}\}$ has a ∇_F-generator $\{G_{\nabla_F}, J, R_{\mathcal{K}}\}$, where*

$$G_{\nabla_F} = \sqrt{2t}\cdot(tI + A)^{-1}\cdot G_{\nabla_A} \tag{5.12}$$

In particular, the ∇_F-displacement rank and the ∇_A-displacement rank of R are the same.

Proof. Both assertions follow from the easily checked identity

$$2\cdot t\cdot(tI + A)^{-1}\cdot(A\cdot R + R\cdot A^*)\cdot(tI + A^*)^{-1} =$$
$$= R - (tI + A)^{-1}\cdot(tI - A)\cdot R\cdot(tI - A^*)\cdot(tI + A^*)^{-1}.$$

$\diamondsuit$

5.5 Algorithm for the discrete-time case

The above two lemmas show how to translate certain formulas from continuous-time to discrete-time, obtaining the following counterpart of lemma 3.1.

Lemma 5.3 *([KO97b] [see also Sec. 8 in [GO94b]]; in the non-partially reconstructible case see [KS95b]) Let $F \in \mathbf{C}^{n\times n}$ be a block lower triangular matrix, $R_1 \in \mathbf{C}^{n\times n}$ be a Hermitian matrix, satisfying a discrete-time Lyapunov displacement equation of the form*

$$\nabla_F(R) = R_1 - F\cdot R_1\cdot F^* = G_1\cdot J\cdot G_1^*, \tag{5.13}$$

for some $G_1 \in \mathbf{C}^{n\times\alpha}$, and some Hermitian matrix $J \in \mathbf{C}^{\alpha\times\alpha}$. Let the matrices in (5.13) be partitioned as

$$F = \begin{bmatrix} F_{11} & 0 \\ F_{21} & F_{22} \end{bmatrix}, \qquad R_1 = \begin{bmatrix} R_{11} & R_{21}^* \\ R_{21} & R_{22} \end{bmatrix}, \qquad G_1 = \begin{bmatrix} G_{11} \\ G_{21} \end{bmatrix}.$$

If the upper left block $R_{11} \in \mathbf{C}^{m\times m}$ of R_1 is invertible, then the Schur complement $R_2 = R_{22} - R_{21}\cdot R_{11}^{-1}\cdot R_{21}^* \in \mathbf{C}^{(n-m)\times(n-m)}$ satisfies

$$R_2 - F_{22}\cdot R_2\cdot F_{22}^* = G_2\cdot J\cdot G_2^*, \tag{5.14}$$

with

$$G_2 = G_{21} - ((\beta I - F_{22})\cdot R_{21}\cdot R_{11}^{-1} - F_{21})\cdot(\beta I - F_{11})^{-1}\cdot G_{11}, \tag{5.15}$$

where β is any number on the unit circle, which is not an eigenvalue of F_1.

This lemma is easily deduced by applying Lemma 5.2 to change the form of displacement equation to the continuous-time one, then use the Lemma 3.1 to obtain the formula for the generator recursion, and finally to again change the form of displacement equation to the discrete-time one. See, e.g., [KO97b] for details.

We now have several fast displacement-based implementations of the GEPP and of symmetric GE with diagonal pivoting for Cauchy-like matrices [for usual and for partially reconstructible]. Thanks to pivoting these algorithms have a good record of numerically accurate performance in certain applications, see [GO93], [GO94b], and therefore it is attractive to use these algorithms for solving linear equations with coefficient matrices having other [i.e., not only Vandermonde-like or Cauchy-like] patterns of structure listed in Table 4. In Sec. 7 it will be achieved by using the transformation approach. However, before doing so we return to the problem noted in Sec. 4.7 and to clarify how fast algorithms of Sec. 5 with diagonal pivoting factorize J-unitary rational matrix functions.

6 Diagonal pivoting and factorization of J-unitary rational matrix functions

To begin with, let us indicate a class of applications giving rise to J-unitary functions.

6.1 Tangential interpolation with norm constraints

Often one is looking for a rational rectangular $M \times N$ matrix interpolant $F(z)$, satisfying non-homogeneous [as opposed to the example in Sec. 4.2] tangential interpolation conditions

$$u_k \cdot F(z_k) = v_k, \tag{6.1}$$

where $\{u_k, v_k\}$ are given vectors of appropriate sizes. It is trivial to reduce this problem to the homogeneous one. Indeed, if we have

$$\begin{bmatrix} u_k & -v_k \end{bmatrix} \cdot \begin{bmatrix} W_{11}(z_k) & W_{12}(z_k) \\ W_{21}(z_k) & W_{22}(z_k) \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}, \tag{6.2}$$

then, of course, one $F(z)$ for (6.1) would be given by

$$F(z) = W_{12}(z) W_{22}(z)^{-1}. \tag{6.3}$$

For physical reasons [i.e., conservation of energy] we are often looking for a *passive* interpolant, imposing the norm constraint

$$\|F(z)\| \leq 1 \tag{6.4}$$

for a certain set of point, say on the imaginary line $i\mathbf{R}$ or on the unit circle. For example, if all the nodes $\{x_k\}$ are in the interior of the unit disk, and we are looking for a scalar rational passive interpolant $f(x_k) = v_k$, then this is the celebrated Nevanlinna-Pick problem [see [A65] and references therein], and the Hermitian null-pole coupling matrix is just a Pick matrix $R = \begin{bmatrix} \frac{1 - v_j v_j^*}{1 - x_i^* x_j} \end{bmatrix}$.

Fortunately, such non-homogenuous problems can be reduced to homogenuous ones, so methods of Sec.4.2, 4.3 and 4.4 are applicable. Indeed, the condition (6.4) for the non-homogeneous interpolant $F(z)$ can be guaranteed by requiring a certain symmetry for the homogeneous interpolant $W(z) = \begin{bmatrix} W_{11}(z_k) & W_{12}(z_k) \\ W_{21}(z_k) & W_{22}(z_k) \end{bmatrix}$. Specifically, we are done if $W(z)$ is a J-unitary rational matrix function [formally defined below], whose null-pole coupling matrix R is Hermitian [R is even positive definite if we require passivity not only on the boundary, but for the whole region, say, in the half-plane, or in the unit disk].

6.2 J-unitary rational matrix functions

Let J be a $\alpha \times \alpha$ signature matrix. A rational $\alpha \times \alpha$ matrix function $W(z)$ is called *J-unitary on the imaginary line $i\mathbf{R}$* if it satisfies

$$(W(z))^* \cdot J \cdot W(z) = J \qquad \text{on} \qquad z \in i\mathbf{R}. \tag{6.5}$$

The point is that in this case the symmetry in $W(z)$ allows us to represent it using only half of the data [as compared to the general case]. More precisely, if we know only two "pole" matrices $\{C_\pi, A_\pi\}$ in (4.6), with B_π unknown, then we are done. Specifically [see Ch. 6 in [BGR90], or[GO94b]], the global left null-pole triple defined in (4.9) has a lot of redundant information:

$$\tau = \{C_\pi, A_\pi, -A_\pi^*, -C_\pi^* J, R\}, \tag{6.6}$$

so that

$$W(z) = I - C_\pi(zI - A_\pi)^{-1} R^{-1} C_\pi^* J, \tag{6.7}$$

where R is a Hermitian matrix satisfying

$$A_\pi^* R + R A_\pi = -C_\pi^* J C_\pi. \tag{6.8}$$

6.3 Diagonal pivoting and factorization of J-unitary rational matrix functions

Now one can easily apply the arguments of Sec. 4.5 to see that the fast symmetric GE with diagonal pivoting algorithm of Sec. 5.3 for triangularization of R in (6.8) also computes the cascade decomposition (4.11) for the corresponding J-unitary on the imaginary line rational matrix function $W(z)$ in (6.7). Moreover, all factors $\Theta_k(z)$ in this decomposition are J-unitary as well, and this algorithm is twice as fast as compared to the fast GEPP of Sec. 3.5, 4.4, 4.5.

We conclude this section by giving an interesting interpolation interpretation to the two comments made just before formulating the algorithm in Sec. 5.3.

- **Block elimination steps.** It is well-known that it is not always possible to decompose a J-unitary rational matrix function into a product of the first-order J-unitary factors [i.e., ones having just one pole and zero]. It was shown in [AD86] that fortunately, it is always possible to write down a cascade decomposition for a J-unitary rational function with at most second-order factors [i.e., having two poles and two zeros]. In fact, in the Bunch-Kaufman pivoting technique we apply either scalar or block 2×2 block elimination steps, see Sec. 2.5. A single elimination step on R_1 corresponds to writing down a first-order factor in the cascade decomposition (4.11), whereas a 2×2 block elimination step corresponds to writing down a second-order factor. Clearly, if the (1,1) entry of R_1 iz zero, than we cannot perform a GE step, and if the whole diagonal of R_1 zero, than the situation cannot be fixed with symmetric row/column interchanges, and we have to perform a block elimination step. In fact, the Bunch-Kaufman pivoting technique performs such 2×2 block elimination steps not only in the case of exactly zero diagonal, but for sufficiently small pivots as well.

- **Displacement operator with nontrivial kernels.** A rational *scalar* function cannot have a pole and a zero at the same point; this is obviously possible with rational *matrix* functions. A closer look at (4.7) [where the eigenvalues of A_π are the poles, and the eigenvalues of A_ζ are the zeros of $W(z)$] shows that this situation gives rise to a displacement operator with a non-trivial kernel, and to what we have called partially reconstructible matrices in Sec. 5. In fact, rational matrix functions with the zeros and poles at the same points appear in many problems. For example, in *boundary tangential interpolation problems* we may be given the interpolation data $\{x_k, u_k, v_k\}$ [i.e., corresponding to an unknown function $F(z)$ in (6.1)], but now with the points not only inside the right-half plane, but also on its boundary $i\mathbf{R}$. In this case we shall also reduce the problem to the homogeneous one as in (6.2), and again the nodes x_k will be the zeros of $W(z)$. The point is that since $W(z)$ will be J-unitary, its zeros on iR should also be its poles, see, e.g., (6.5). To sum up, the boundary nodes $x_k \in i\mathbf{R}$ give rise to a displacement operator with nontrivial kernel, as can be seen directly from (6.8). We refer to [KO97b] for the details.

  Another example of such a problem is the rational matrix Nehari interpolation problem, discussed in Sec. 6.5, 6.6 below.

6.4 Discrete-time case

We have discussed the case of a J-unitary on the line rational matrix function, which is the *continuous-time* version of the problem. The *discrete-time* version corresponds to the *J-unitarity on the unit circle*

$\partial\mathcal{D}$ rational $\alpha \times \alpha$ matrix functions $W(z)$, i.e., those satisfying $W(z) \cdot J \cdot (W(\frac{1}{z^*}))^* = J$ on $|z| = 1$. In the continuous-time case above we assumed $W(z)$ to have the identity value at infinity. Here, in the discrete-time case, we shall assume instead that $W(\beta) = I$, where β is a certain point on the unit circle, $|\beta| = 1$. Again, the left global left null-pole triple has a lot of redundant information. As above, if we are given only two "pole matrices," $\{C_\pi, A_\pi\}$, they already define the whole triple $\tau = \{C_\pi, A_\pi, (A_\pi^{-1})^*, (A_\pi^{-1})^* C_\pi^* J, R\}$, so that

$$W(z) = [I + C_\pi (zI - A_\pi)^{-1} R^{-1} (A_\pi^{-1})^* C_\pi^* J] \cdot D_\beta,$$

where $D_\beta = I + C_\pi R^{-1} (I - \beta A_\pi^*)^{-1} C_\pi^* J$, and R is a Hermitian matrix satisfying

$$R - A_\pi^* R A_\pi = -C_\pi^* J C_\pi. \tag{6.9}$$

see, e.g., [BGR90], Ch. 7.

Now recall that Lemma 5.3 describes the displacement structure of the Schur complement for the case when this displacement structure is defined via the discrete-time displacement operator, such as in (6.9). Therefore this Lemma 5.3 is just an "array part" of the corresponding version of theorem on factorization of rational J-unitary on the unit circle matrix functions.

We conclude this subsection with noting that these two versions of a factorization theorem can be easily deduced from each other, by providing an interpolation interpretation for the proof of Lemma 5.3. Recall that it was based on the results of Sec. 5.4, and consisted of three steps. First transform the discrete-time displacement equation (6.9) to a continuous-time one (6.8), then perform one step of Schur complementation using Lemma 3.1, and finally transform the obtained formulas back to the discrete-time form. In the function domain these three steps are even simpler: we are given a J-unitary on the circle rational matrix function $W_1(z)$, which we would like to factorize. In the first step we choose an appropriate Mobius transformation $\varphi(z) = \beta\frac{z-1}{z+1}$ of the unit circle onto the right-half-plane, so that $W_1(\beta\frac{z-1}{z+1})$ is now a J-unitary on the $i\mathbf{R}$ rational matrix function. Therefore, in the second step we can use theorem 4.2 to factorize it,

$$W_1(\beta\frac{z-1}{z+1}) = \Theta_1(\beta\frac{z-1}{z+1}) \cdot W_2(\beta\frac{z-1}{z+1}).$$

Finally, in the third step we map the right-half plane back to the unit circle, obtaining formulas in Lemma 5.3, see, e.g., [KO97b] for details. This reinforces the point made in Sec. 5.4 that there are no essential differences between different forms for the displacement operator, but it may be important to provide a user with a particular form of a theorem or of an algorithm he may need.

6.5 Numerical example: Nehari problem

There are many interpolation problems, e.g., listed in Sec. 4.1, for which we are able to specify the above results of Sec. 6. We present only one such such example, for which it was shown numerically in [GO94b] that partial and diagonal pivoting techniques indeed lead to the higher accuracy. Loosely speaking, in this problem we are given a rational matrix function $K(z)$

- with several poles in the left-half plane,
- bounded on the imaginary line: $\sup_{z\in i\mathbf{R}} \|K(z)\| < \infty$,

and our task is to fix it, i.e, to find a new function $K(z) - R(z)$

- with the same pole structure in the left-half-plane [i.e., the fixing term $R(z)$ is analytic there],
- passive on the imaginary line: $\sup_{z\in i\mathbf{R}} \|K(z) - R(z)\| < 1$.

Here is the precise formulation of this problem.

<u>Suboptimal rational matrix Nehari problem. Simple poles.</u>

Given :

n complex points	$\{z_1, \ldots, z_n\}$	in the left-half-plane Π^-,
n $1 \times N$ row vectors	$\{w_1, \ldots, w_n\}$,	
n $M \times 1$ column vectors	$\{\gamma_1, \ldots, \gamma_n\}$,	

and a rational $M \times N$ matrix function $K(z)$ having only n simple poles $\{z_k\}$ in the left-half-plane, with a local Laurent series in a neighborhood of each z_k:

$$K(z) = (z - z_k)^{-1} \gamma_k \cdot w_k + \text{ [analytic at } z_k] \ .$$

such that

$$\sup_{z \in i\mathbf{R}} \|K(z)\| < \infty.$$

Find : A rational $M \times N$ matrix function $R(z)$ with no poles in $\Pi^- \cup i\mathbf{R}$, such that

$$\sup_{z \in i\mathbf{R}} \|K(z) - R(z)\| < 1.$$

Solvability condition : The $2n \times 2n$ matrix

$$R = \begin{bmatrix} Q & I \\ I & P \end{bmatrix} \tag{6.10}$$

has exactly n positive and n negative eigenvalues, [equivalently, $\lambda_1(PQ) < 1$], where

$$P = \begin{bmatrix} -\frac{w_i \cdot w_j^*}{z_i + z_j^*} \end{bmatrix}, \qquad Q = \begin{bmatrix} -\frac{\gamma_i^* \cdot \gamma_j}{z_i^* + z_j} \end{bmatrix}$$

Solution , as given, e.g., in [BGR90]:

$$K(z) - R(z) = [W_{11}(z) G(Z) + W_{12}(z)] \cdot [W_{21}(z) G(z) + W_{22}(z)]^{-1},$$

with an arbitrary parameter $G(z)$ satisfying

- $G(z)$ has no poles in $\Pi^- \cup i\mathbf{R}$,
- $\sup_{z \in i\mathbf{R}} \|G(z)\| \le 1$,

and

$$\boxed{W(z) = \begin{bmatrix} W_{11}(z) & W_{12}(z) \\ W_{21}(z) & W_{22}(z) \end{bmatrix} = I_{M+N} - C_\pi (z I_{2n} - A_\pi)^{-1} R^{-1} C_\pi^* J}, \tag{6.11}$$

where R is given by (6.10), and

$$A_\pi = \text{ diag } (z_1, \ldots, z_n, -z_1^*, \ldots, -z_n^*), \qquad C_\pi = \begin{bmatrix} \gamma_1 & \ldots & \gamma_n & 0 & \ldots & 0 \\ 0 & \ldots & 0 & w_1^* & \ldots & w_n^* \end{bmatrix}, \qquad J = \begin{bmatrix} I_N & 0 \\ 0 & -I_M \end{bmatrix},$$

The solution (6.11) for the Nehari problem is yet another example of using the same BGR formula (4.8) which is not convenient because it involves the inverse of R. Therefore we might reinforce the comment made in Sec. 4.2. Specifically, we have a complete description for the solution of the Nehari problem, but we still have to design efficient accurate algorithm to compute this solution. Of course, R in (6.10) is a partially reconstructible Cauchy-like matrix satisfying

$$A_\pi^* R + R A_\pi = -C_\pi^* J C_\pi, \tag{6.12}$$

The given data does not specify the entries of R, and it is given by its generator $\{C_\pi, -J, R_{\mathcal{K}}\}$ [see Sec. 5.1 for the definition of a generator of partially reconstructible matrices]. The first two matrices of this generator

Fast Cascade decomposition			Fast Solving (4.10)			Slow Structure-ignoring GECP
No pivoting	Partial pivoting	Diagonal pivoting	No pivoting	Partial pivoting	Diagonal pivoting	
4.4e+00	3.2e-05	9.2e-05	2.4e+00	1.3e-05	6.1e-05	2.4e-05

Table 5: *Forward relative error for a matrix Nehari problem.*

are provided by the right-hand-side of (6.12), and the third matrix has the form $R_{\mathcal{K}} = \begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix}$, because the kernel of ∇_A is clearly the set of matrices of the form $\begin{bmatrix} 0 & D_1 \\ D_2 & 0 \end{bmatrix}$ with arbitrary diagonal D_1, D_2.

Given this generator $\{C_\pi, -J, R_{\mathcal{K}}\}$, we are able to run the fast symmetric GE with diagonal pivoting algorithm of Sec. 5.3 to rapidly compute the block LDL^* factorization of R in $O(n^2)$ operations. The rational matrix function $W(z)$ given by (6.11) is J-unitary on the imaginary line. As was explained in Sec. 6.3, the same algorithm of Sec. 5.3 rapidly computes the cascade decomposition (4.11) for $W(z)$. Having this cascade decomposition we are able to evaluate $W(z)$ [and therefore $K(z) - R(z)$ as well] in linear time.

In a wide variety of computed examples in [GO94b] it was found that indefinite matrix R [recall that it has exactly n positive and n negative eigenvalues] is usually moderately conditioned. However, its blocks P and Q are positive definite matrices that were always extremely ill-conditioned. Therefore it is not surprising that numerical examples led us to the conclusion that partial and diagonal pivoting techniques are needed to compute not only fast, but also accurate, factorization. We refer to [GO94b] for many numerical experiments and for practical recommendations. Here we only include one illustration with a 2×2 rational matrix function given by the interpolation data at 60 interpolation points [so R is 120×120 matrix]. The solvability condition [which is $\lambda_1(PQ) < 1$] is almost violated: $\lambda_1(PQ) = 0.99$. Table 6.5 shows the relative forward error $\frac{\|F(z_0) - \widehat{F}(z_0)\|_\infty}{\|F(z_0)\|_\infty}$ for the evaluation of the solution $F(z) = K(z) - R(z)$ at a certain point z_0. As usual, the hatted quantity denotes the actually *computed* result.

In this case all the fast methods produce the same accuracy as the much slower GECP [we found that partial pivoting is often slightly better than diagonal pivoting]. There are classes of problems for which fast algorithms are even more accurate than the slow GECP, as illustrated in the next section.

6.6 Numerical Example: Nehari-Takagi problem

We again recall the solvability condition for the matrix Nehari problem: $\lambda_1(PQ) < 1$, or equivalently, the null-pole coupling matrix $R = \begin{bmatrix} Q & I \\ I & P \end{bmatrix}$ has the inertia $(n, n, 0)$ [i.e., exactly n positive and exactly n negative eigenvalues]. The more general Nehari-Takagi problem deals with the situation where this solvability condition is violated, and the inertia of R is, say $(n + s, n - s, 0)$. In this case we cannot fix our given $K(z)$, i.e. to achieve the desired inequality

$$\|K(z) - R(z)\| \le 1 \qquad\qquad z \in i\mathbf{R},$$

with the fixing function $R(z)$ analytic in Π^- [as it was in the Nehari problem], but the point is that we can now fix $K(z)$ with $R(z)$ having only s poles in Π^-. Moreover, all the formulas in Sec. 6.5, and therefore all the algorithms remain valid. Interestingly, such slightly different eigenvalue profile causes different numerical properties for the algorithms discussed. Specifically, it was found that diagonal pivoting now performs better than partial pivoting, and that fast algorithms provide higher accuracy than the method based on the structure-ignoring GECP. Again we illustrate these conclusions only with one example, and refer to [GO94b] for more information. The following table shows the relative forward error for a 2×2 matrix function, given by the interpolation data at 60 interpolation points [so R is 120×120 matrix], and we allow $R(z)$ to have 15 simple poles.

Fast Cascade decomposition			Fast Solving (4.10)			Slow Structure-ignoring GECP
No pivoting	Partial pivoting	Diagonal pivoting	No pivoting	Partial pivoting	Diagonal pivoting	
1.5e+00	2.2e-02	2.5e-05	6.7e+00	2.1e-02	2.6e-03	7.5e-02

Table 6: *Forward relative error for a matrix Nehari-Takagi problem.*

Given the evidence that fast algorithms with pivoting for Cauchy-like matrices are accurate, it is attractive to extend this approach to solve linear systems with other patterns of structure. One way of doing this is described next.

7 Toeplitz and Toeplitz-plus-Hankel solvers: transformations and pivoting

7.1 Transformation-and-pivoting

As we noted in (3.10), fast implementation of partial pivoting are possible for all those structures in Table 4 that are defined using diagonal matrices F for the corresponding displacement operators

$$\nabla_{\{F,A\}}(R) = FR - RA = GB \tag{7.1}$$

[cf. with the comment made after (3.10)]. The question is: how to devise accurate fast algorithms for the other important kinds of displacement structure in Table 4, including Toeplitz-like, Hankel-like, Toeplitz-plus-Hankel-like, Chebyshev-Hankel-like, polynomial-Hankel-like matrices for which F not diagonal. One way of doing this is to exploit a useful observation that these matrices F can be diagonalized, $F = T_F^{-1} D_F T_F$, $A = T_A^{-1} D_A T_A$, so that the matrix $T_F R T_A^{-1}$ is a Cauchy-like matrix satisfying

$$D_F(T_F R T_A^{-1}) - (T_F R T_A^{-1}) D_A = (T_F G)(B T_A^{-1}).$$

So its permuted triangular factorization, $T_F R T_A^{-1} = PLU$, can be computed via the fast GEPP algorithm of Sec. 3.5. Of course, all the computation can be done as a manipulation on generators, as seen from the following scheme of the overall algorithm for solving $Ra = f$.

A scheme for transformation-and-pivoting algorithms for solving $Ra = f$

INPUT: A $\nabla_{\{F,G\}}$-generator $\{G, B\}$ in (7.1) and f.

OUTPUT: The solution of $Ra = f$.

1. Compute the $\nabla_{\{D_F,D_A\}}$-generator
$$\{T_F G, B T_A^{-1}\} \tag{7.2}$$
for the Cauchy-like matrix $T_F R T_A^{-1}$.
2. Use this generator as the input for the fast GEPP algorithm in Sec. 3.5 to compute the permuted triangular factorization $T_F R T_A^{-1} = PLU$.
3. Solve
$$(PLU)b = T_F f \tag{7.3}$$
via the forward and back-substitution.
4. Finally, compute the solution,
$$a = T_A^{-1} b. \tag{7.4}$$

As we shall see in a moment, in all cases interesting to us, matrices T_F, and T_A will be certain discrete transform matrices, for which accurate superfast $O(n \log n)$ algorithms are available for multiplication and solving linear system. Therefore the algorithm just specified will have the same complexity $O(n^2)$ operations as the fast GEPP algorithm of Sec. 3.5 used in the step 2 of the above scheme. Indeed, the only difference will be $O(n \log n)$ operations to compute $2(\alpha+1)$ discrete transforms needed in (7.2), (7.3) and (7.4). Here α is the displacement rank, i.e., the number of columns of each of matrices $\{G, B^T\}$.

We next specify this scheme for Toeplitz-like and Toeplitz-plus-Hankel-like matrices.

7.2 Toeplitz-like matrices. FFT-based transformation

Toeplitz-like matrices have low displacement rank with respect to the displacement operator

$$\nabla_{\{Z_1, Z_{-1}\}}(R) = Z_1 R - R Z_{-1} = GB, \qquad \text{where} \qquad Z_\phi = \begin{bmatrix} 0 & 0 & \cdots & 0 & \phi \\ 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix}. \tag{7.5}$$

Now, for arbitrary Toeplitz-like matrix the general scheme of Sec. 7.1 can be nicely implemented, because Z_ϕ in (7.5) is a ϕ-circulant matrix, so it is diagonalized by the scaled discrete Fourier matrix $\mathcal{F} = \frac{1}{\sqrt{n}}(\omega^{ij})_{i,j=0}^{n-1}$ [with ω denoting the primitive n-th root of unity]:

$$Z_\phi = D_\phi^{-1} \cdot \mathcal{F}^* \cdot D_{Z_\phi} \cdot \mathcal{F} \cdot D_\phi, \tag{7.6}$$

with $D_{Z_\phi} = \text{ diag } ((\xi\omega^i)_{i=0}^{n-1})$, and $D_\phi = \text{ diag } ((\xi^i)_{i=0}^{n-1})$, where ξ is an arbitrary complex number satisfying $\xi^n = \phi$. Summarizing, for Toeplitz-like matrices the transformation-and-pivoting algorithm of Sec. 7.1 is based on Fast Fourier transform (FFT), because in this case we have:

$$T_{Z_1} = \mathcal{F}, \qquad T_{Z_{-1}} = \mathcal{F} D_{-1}.$$

Note that the both above similarity matrices are unitary, and that in this case the transformed Cauchy-like matrix $\mathcal{F} R D_{-1}^* \mathcal{F}^*$ in

$$D_1(\mathcal{F} R D_{-1}^* \mathcal{F}^*) - (\mathcal{F} R D_{-1}^* \mathcal{F}^*) D_{-1} = (\mathcal{F} G)(B D_{-1}^* \mathcal{F}^*)$$

is very special, with the nodes [i.e., diagonal entries of D_1, D_{-1}] being primitive n-th roots of 1 and -1, resp.

We refer to [GKO95] for a large number of computed examples showing that this transformation-and-pivoting method [called the GKO algorithm] in practice provides a high relative forward and backward accuracy.

7.3 A problem: loss of symmetry. Partially reconstructible matrices

The weakness of the approach in Sec. 7.2 is that it ignores the possible symmetry in R. Indeed, we used (7.5) to define the class of Toeplitz-like matrices, so even for Hermitian R we have two different generator matrices $\{G, B\}$. As a result, a transformed Cauchy-like matrix, $\mathcal{F} R D_{-1}^* \mathcal{F}^*$, is no longer Hermitian.

To describe a symmetry-preserving transformation-and-pivoting algorithm we exploit the fact that there are many equivalent forms of displacement equations for Toeplitz-like matrices. For example, for Hermitian Toeplitz-like matrices R we can equivalently use the *ϕ-cyclic* displacement equation

$$\nabla_{Z_\phi}(R) = R - Z_\phi \cdot R \cdot Z_\phi^* = G \cdot J \cdot G^*, \tag{7.7}$$

where J is a $\alpha \times \alpha$ signature matrix, so the structure of R is now captured by the αn entries of only one generator matrix G. For any $|\phi| \neq 1$ the identity (7.6) can be used to transform a *Hermitian* Toeplitz-like matrix R into a *Hermitian* Cauchy-like matrix $\mathcal{F} D_\phi R D_\phi^* \mathcal{F}^*$, satisfying

$$\nabla_{\Lambda_\phi}(\mathcal{F} D_\phi R D_\phi^* \mathcal{F}^*) = (\mathcal{F} D_\phi R D_\phi^* \mathcal{F}^*) - \Lambda_\phi \cdot (\mathcal{F} D_\phi R D_\phi^* \mathcal{F}^*) \cdot \Lambda_\phi^* = (\mathcal{F} D_\phi G) \cdot J \cdot (\mathcal{F} D_\phi G)^*. \tag{7.8}$$

Then instead of fast GEPP of Sec. 3.5 [which would destroy the Hermitian character of the Cauchy-like matrix $\mathcal{F}D_\phi R D_\phi^* \mathcal{F}^*$] we can use the fast symmetric GE with diagonal pivoting of Sec. 5.5 or of Sec. 5.3 [see [KO97b] for details]. For *any* choice of ϕ in (7.7) the resulting algorithm is about twice as fast as the original GKO algorithm.

The problem is, that while the different choices of ϕ are all theoretically equivalent, the corresponding computational schemes have different numerical properties. Our numerical experiments indicate that choices with $|\phi| \neq 1$ are impractical, often leading to overflows, and to losses of accuracy for reasonably well-conditioned Toeplitz systems. By extensive testing [KO97b] it was found that fast algorithms corresponding to the choice $\phi = 1$ allow good numerical properties.

However the choice $|\phi| = 1$ means that the displacement operator on (7.7) has a non-trivial kernel [consisting of all ϕ-circulant matrices], so R is a partially reconstructible Toeplitz-like matrix.

In Sec. 5.3 we specified fast GE with diagonal pivoting algorithms for partially reconstructible Cauchy-like matrices. So the only point remaining to clarify is how to efficiently organize the transformation to a Cauchy-like matrix for partially reconstructible Toeplitz-like matrices.

7.4 Transformation of partially reconstructible Toeplitz-like matrices to partially reconstructible Cauchy-like matrices

Here we shall specify the definition of Sec. 5.1 of a generator for a partially reconstructible Toeplitz-like matrix R in

$$\nabla_{Z_1}(R) = R - Z_1 \cdot R \cdot Z_1^*. \tag{7.9}$$

As is well known [see, e.g., [GO92]], the kernel of $\nabla_{Z_1}(\cdot)$ in (7.9) is the subspace of all circulants, i.e.,

$$\mathcal{C} = \{\mathrm{Circ}(c) : c \in \mathbf{C}^n\} \subset \mathbf{C}^{n\times n} \qquad \text{where} \qquad \mathrm{Circ}(c) = \begin{bmatrix} c_0 & c_{n-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & \cdots & c_2 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ c_{n-2} & & c_1 & c_0 & c_{n-1} \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{bmatrix}. \tag{7.10}$$

Given arbitrary R, the next lemma and corollary show how to compute in $O(n^2)$ arithmetic operations the decomposition of R with respect to $\mathbf{C}^{n\times n} = \mathcal{C} \oplus \mathcal{C}^\perp$.

Lemma 7.1 *Let $\mathcal{C} \subset \mathbf{C}^{n\times n}$ denote the subspace of all circulant matrices, use an inner product in $\mathbf{C}^{n\times n}$ defined by* (5.3)*, and let $R = [\ r_{ij}\ ] \in \mathbf{C}^{n\times n}$ be arbitrary. Then $\quad R \perp \mathcal{C} \quad$ if and only if for $k = 1, 2, ..., n$ we have $\quad \sum_{i=1}^n r_{i+k-1,i} = 0 \quad$ [Here the indices are integers modulo n, i.e., if $k > n$, then $k := k - n$].*

Briefly, a matrix R is orthogonal [in the sense of (5.3)] to all circulants if and only if for each of its circulant diagonals, the sum of the entries is zero.

Proof. It is easy to check that $< Z_1^k, R >= \sum_{i=1}^n r_{i+k-1,i}$. Since $\{I, Z_1, Z_1^2, ..., Z_1^{n-1}\}$ form a basis in $\mathcal{C} \subset \mathbf{C}^{n\times n}$, the assertions follow.

$\diamondsuit$

Corollary 7.2 *With $\mathcal{C} \subset \mathbf{C}^{n\times n}$ as above, a matrix $R = [\ r_{ij}\ ] \in \mathbf{C}^{n\times n}$ is decomposed with respect to $\mathbf{C}^{n\times n} = \mathcal{C} \oplus \mathcal{C}^\perp$ as*

$$R = R_{\mathcal{C}} + R_{\mathcal{C}^\perp}, \tag{7.11}$$

with

$$R_{\mathcal{C}} = \mathrm{Circ}(c_0, c_1, ..., c_{n-1}), \qquad \text{where} \qquad c_{i-1} = \frac{1}{n}\sum_{k=1}^n r_{i+k-1,k}, \tag{7.12}$$

and the indices are again integers modulo n.

For example, the ordinary Hermitian Toeplitz matrix has a ∇_{Z_1}-generator

$$\{ \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} \\ t_1 - t_{n-1}^* & t_1 - t_{n-1}^* \\ \vdots & \vdots \\ t_{n-1} - t_1^* & t_{n-1} - t_1^* \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \mathrm{Circ}(c_0, c_1, ..., c_{n-1})\},$$

where $c_i = \frac{(n-i)\cdot t_i + i\cdot t_{i-n}}{n}$. To transform R to a partially reconstructible Cauchy-like matrix we specify (7.8),

$$Z_1 = \mathcal{F}^* \cdot D_1 \cdot \mathcal{F}, \qquad \text{with} \qquad D_1 = \operatorname{diag}(1, w, w^2, ..., w^{n-1}), \tag{7.13}$$

where $\omega = e^{\frac{2\pi i}{n}}$ is the n-th primitive root of unity. It is a trivial exercise to check that the kernel of $\nabla_{D_1}(R) = R - D_1 R D_1^*$ is the subspace

$$\mathcal{D} = \{\operatorname{diag}(f) : f \in \mathbf{C}^n\} \subset \mathbf{C}^{n\times n}$$

of all diagonal matrices. Now we are ready to state the transformation result.

Lemma 7.3 *(Discrete-time transformation) Let Z_1, D_1 be as in* (7.13)*, and let $R = \left[\ r_{ij}\ \right]$ be a partially reconstructible Toeplitz-like matrix, given by a ∇_{Z_1}-generator $\{G_{\nabla_{Z_1}}, J, \operatorname{Circ}(c)\}$, i.e.,*

$$\nabla_{Z_1}(R) = R - Z_1 \cdot R \cdot Z_1^* = G_{\nabla_{Z_1}} \cdot J \cdot G_{\nabla_{Z_1}}^*,$$

and $c_i = \frac{1}{n}\sum_{k=1}^n r_{i+k-1}$. Then $\mathcal{F}\cdot R\cdot \mathcal{F}^*$ is a partially reconstructible Cauchy-like matrix, satisfying

$$\nabla_{D_1}(\mathcal{F}\cdot R\cdot\mathcal{F}^*) = (\mathcal{F}\cdot R\cdot\mathcal{F}^*) - D_1\cdot(\mathcal{F}\cdot R\cdot\mathcal{F}^*)\cdot D_1^* = G_{\nabla_{D_1}}\cdot J\cdot G_{\nabla_{D_1}}^*,$$

with a ∇_{D_1}-generator $\{G_{\nabla_{D_1}}, J, \operatorname{diag}(d)\}$, where

$$G_{\nabla_{D_1}} = \mathcal{F}\cdot G_{\nabla_{Z_1}}, \qquad \text{and} \qquad d = \sqrt{n}\cdot\mathcal{F}\cdot c. \tag{7.14}$$

$\diamondsuit$

Lemma 7.3 naturally leads to the symmetry-preserving transformation-and-pivoting algorithm based on the discrete-time algorithm for Cauchy-like matrices formulated in Sec. 5.5. This algorithm is about twice as fast as the GKO algorithm [based on partial pivoting] formulated in Sec. 3.5.

In the next lemma we specify a continuous-time analog of this transformation which is immediately deduced using the simple technique of Sec. 5.4.

Lemma 7.4 *(Continuous-time transformation) Let R be a partially reconstructible Toeplitz matrix, satisfying*

$$\nabla_{Z_1}(R) = R - Z_1\cdot R\cdot Z_1^* = G_{\nabla_{Z_1}}\cdot J\cdot G_{\nabla_{Z_1}}^*, \tag{7.15}$$

so that it is given by a ∇_{Z_1}-generator $\{G_{\nabla_{Z_1}}, J, \operatorname{Circ}(c)\}$. Define

$$A_1 = \operatorname{diag}(\frac{\tau+1}{\tau-1}, \frac{\tau+w}{\tau-w}, \frac{\tau+w^2}{\tau-w^2}, ..., \frac{\tau+w^{n-1}}{\tau-w^{n-1}}), \qquad \text{where} \qquad w = e^{\frac{2\pi i}{n}},$$

with τ be any number with $|\tau| = 1$ so that $\tau^n \neq 1$. Then $\mathcal{F}\cdot R\cdot\mathcal{F}^*$ is a Cauchy-like matrix, satisfying, , satisfying

$$\nabla_{A_1}(\mathcal{F}\cdot R\cdot\mathcal{F}^*) = A_1\cdot(\mathcal{F}\cdot R\cdot\mathcal{F}^*) + (\mathcal{F}\cdot R\cdot\mathcal{F}^*)\cdot A_1^* = G_{\nabla_{A_1}}\cdot J\cdot G_{\nabla_{A_1}}^*, \tag{7.16}$$

with a ∇_{A_1}-generator $\{G_{\nabla_{A_1}}, J, \operatorname{diag}(d)\}$, where

$$G_{\nabla_{A_1}} = \sqrt{2}\cdot(\tau I - D_1)\cdot\mathcal{F}\cdot G_{\nabla_{Z_1}}, \qquad \text{and} \qquad d = \sqrt{n}\cdot\mathcal{F}\cdot c. \tag{7.17}$$

This lemma naturally leads to one more symmetry-preserving transformation-and-pivoting algorithm based on the continuous-time algorithm for Cauchy-like matrices formulated in Sec. 5.3. This algorithm is also about twice as fast as GKO algorithm formulated in Sec. 3.5.

slow	slow	fast	fast		fast	
GEPP	SGEBKP	GKO	cont-TP		disc-TP	
		[partial]	without	diagonal	without	diagonal
		[pivoting]	pivoting	pivoting	pivoting	pivoting
7.8e-06	1.6e-0.5	2.5e-0.5	1.3e+00	2.2e-06	6.1e-01	4.5e-06

Table 7: *Chebyshev Toeplitz matrix with* $a = 0.2$.

slow	slow	fast	fast	fast	fast		fast	
GEPP	SGEBKP	Levinson	Schur	GKO	cont-TP		disc-TP	
				[partial]	without	diagonal	without	diagonal
				[pivoting]	pivoting	pivoting	pivoting	pivoting
2.0e-07	1.4e-0.6	2.8e-04	1.2e-07	7.9e-0.7	1.9e-04	1.6e-07	1.5e-03	2.8e-07

Table 8: *Gaussian Toeplitz matrix with* $a = 0.9$.

7.5 Numerical example

We refer to [KO97b] for a thorough comparison of the numerical performance of both discrete-time and continuous-time versions of fast GE with diagonal pivoting with several other available algorithms, including the usual slow GEPP, the fast $O(n^2)$ classical Schur and Levinson algorithms, etc. Here we just present two illustrations suggesting that the approach indeed works numerically. In Table 7 we include the forward relative error for a linear system with a 70×70 Chebyshev-Toeplitz coefficient matrix, i.e., with an indefinite symmetric Toeplitz matrix with the first row $\left[\begin{array}{cccccc} T_0(a) & \dots & T_{35}(a) & 0 & \dots & 0 \end{array} \right]$ [tabulation of Chebyshev polynomials at a certain point a]. We note that the three-term recurrence relations $xT_k(x) = \frac{T_{k-1}(x)+T_{k+1}(x)}{2}$ for Chebyshev polynomials imply that all $k \times k$ leading minors are singular for $k = 3, 4, \dots, 35$. This means that the classical fast Schur and Levinson Toeplitz solvers will just break down. The algorithms for the slow GEPP and SGEBKP [symmetric Gaussian elimination with Bunch-Kaufman pivoting] are taken from LAPACK, the GKO is the fast algorithm of Sec. 3.5, 4.5 [partial pivoting], cont-TP and disc-TP are the continuous-time and discrete-time transformation+pivoting algorithms, resp. We used the following variant of diagonal pivoting: first perform the symmetric pivoting step to maximize the (1,1) entry over the main diagonal, then perform the usual Bunch-Kaufman step. Table 7 illustrates the accuracy of fast pivoting techniques.

It is usually commented that symmetric pivoting with positive definite matrices is not necessary [because of the diagonal dominance], if one employs slow GE. However, many generally valid guidelines cannot be automatically carried over to problems where structured matrices and structure-exploiting algorithms are involved. To illustrate this point we included an example with [positive definite] ill-conditioned Gaussian Toeplitz matrix, $T = \left[\begin{array}{c} a^{(i-j)^2} \end{array} \right]$, arising, e.g., in image restoration problems. The next table shows the residual error for a 70×70 Gaussian Toeplitz system. In this example we used symmetric pivoting [for diagonal pivoting], because for positive definite matrices it is equivalent to complete pivoting.

We refer to [GKO95] and [KO97b] for many other computed examples, and for practical recommendations, and here only briefly note that

- pivoting is needed to improve the accuracy of fast algorithms cont-TP and disc-TP even with positive definite matrices,

- even with positive definite matrices the numerical properties of the Levinson algorithm are less favorable as compared to the Schur and other fast algorithms.

- Recall that for indefinite matrices the classical Schur and Levinson algorithms are inaccurate. Sometimes such inaccuracy can be overcomed via one step of iterative refinement, but there are examples, e.g., in Table 7, where there is nothing to refine, i.e., the classical Schur and Levinson algorithms just break down.

7.6 Some remarks

Toeplitz linear equations appear in numerous applications, and along with general algorithms, several fast $O(n^2)$ solvers are available, e.g., the classical fast Schur and Levinson algorithms. These fast algorithms perform quite well for positive definite Toeplitz matrices [see, e.g. [GKO95] [KO97b] for numerical illustrations and recommendations, and [C80], [BBHS95] for the error analysis]. Nevertheless, their poor numerical properties for indefinite Toeplitz linear equations are well-known, the problem arises from the fact that the classical versions of the Levinson and the Schur algorithms recursively process all leading submatrices, they break down numerically if some of these submatrices are ill-conditioned. This fact motivated a number of the authors to design *look-ahead* versions for these fast algorithms; the idea behind the look-ahead approach is to jump over ill-conditioned submatrices which in practice has been shown to often improve accuracy [CH92], [FZ93], [GH94]. The language used in these and other look-ahead papers is rather different, varying from matrix arguments, through the use of the Padé table to applying a theory of formally biorthogonal polynomials. However, such look-ahead algorithms have their own limitations, for example, they are slow if the coefficient matrix has a large number of ill-conditioned submatrices [for example, for $k = 3, 4, \ldots, n/2$ the leading minors of the coefficient matrix in Table 7 are zero!]. So, the transformation-and-pivoting approach pursued in this section seems to be an attractive alternative, because it is simpler and it is always fast. We, however, have to point out that transformations of structured matrices is the known tool which has been used earlier by many authors for different purposes. For example, in [F84] it was shown how to transform a Hankel matrix into a Loewner matrix of the form $\left[\frac{a_i - b_j}{x_i - y_j} \right]$ [which is, of course, a Cauchy-like matrix], and a similar approach can be found in [L74]. For Toeplitz matrices transformations based on the discrete cosine/sine transforms were studied in [O93], [O95], however, in these papers the structure of the transformed matrix was not identified as a Cauchy-like one.

These results concern with the ordinary Toeplitz and Hankel matrices, and in [DGK81] it was suggested to replace the classical Schur-Cohn stability test by another method based on the classical Nevanlinna algorithm. A matrix interpretation of this method is the transformation of the Schur-Cohn matrix [which is Toeplitz-like, with displ. rank = 2] to a Pick matrix [which is Cauchy-like, with displ. rank = 2]. Interestingly, the motivation of [DGK81] was numerical: to reduce the stability test to a possibly better conditioned problem. However, since the transformation is based on unitary discrete transform matrices, such a reduction cannot lead to a better conditioned matrix. Furthermore, for Toeplitz-like, Vandermonde-like and Cauchy-like matrices with arbitrary displacement ranks the transformation approach was suggested in [P90] [see also [BP94]] as an interesting general tool to design new algorithms [i.e., to use algorithm designed for one class of structure to solve problems for another]. This idea was also used in [GO94c] to design new algorithms for matrix-vector multiplication for Vandermonde-like and Cauchy-like matrices [via their transformation to Toeplitz-like matrices].

The judicious idea to combine the transformation technique with the possibility to pivot on Cauchy-like matrices was suggested in [H95], however the fast algorithm of [H95] is of a hybrid Levinson-Schur-type, and this algorithm, as it stated, does not provide a satisfactory accuracy. In [GKO95] a variant of the fast transformation-and-pivoting algorithm was suggested as a combination of the results of [GO94c] [transformation to a Cauchy-like matrix], and of the fast GEPP of [GO93], [GO94b] [used to used to compute the triangular factorization for the transformed Cauchy-like matrix]. A large number of numerical experiments in [GKO95] and [KO97b] indicated that the approach is indeed practical, and that the algorithms of this type can be recommended as accurate in practice Toeplitz solvers. Further applications and results in this direction can be found in [KO95], [GTVDV96], [KVB96], [KO97a], [HB97], [KL96], [K97].

An a-priori rounding error analysis for the GKO algorithm of [GKO95] has been recently performed in [SB95]. Along with the usual *element-growth-factor* [appearing also in the error bounds of the usual structure-ignoring GE], the corresponding error bound for the GKO algorithm also involves one more term, called a *generator-growth-factor*. This bound suggests that the accuracy of the usual slow GEPP and its fast implementation can be different. More precisely, if the element-growth-factor is small, but the generator-growth-factor is large, then the accuracy of the fast GKO algorithm can be less than the one of the slow GEPP. This bound motivated further modifications of the GKO algorithm, based on further fast implementations of several improved pivoting techniques [G95], [S96] aimed at the reduction of the generator-growth-factor.

All these results are quite recent, and, of course, only preliminary conclusions can be made at the moment. At the same time our current numerical experience [and the private communications with R.Brent, D.Sweet, M.Gu and M.Stewart] suggests that although there constructed few examples of Toeplitz matrices with larger generator-growth-factor [SB95], [S96], nevertheless the size of this factor is almost invariable comparable with the size of the element-growth factor, and the GKO algorithm [i.e., with partial pivoting] seems to be quite reliable in practice. To sum up, we think that the comments made and cited in Sec. 2.2 for GE and unstructured matrices can be applied to structured matrices and fast algorithms as well.

7.7 Transformations for polynomial-Hankel-like matrices

7.7.1 Polynomial Hankel-like matrices

To motivate introducing a general class of polynomial-Hankel-like matrices let us look first more closely at its important special case of Toeplitz-plus-Hankel-like matrices. Consider the displacement equation

$$\nabla_{H_Q}(R) = H_Q^T \cdot R - R \cdot H_Q = GJG^T, \tag{7.18}$$

with a tridiagonal matrix

$$H_Q = \frac{1}{2}\,\text{tridiag}\begin{bmatrix} & * & & 1 & & 1 & \cdots & & 1 & & * & \\ * & & 0 & & 0 & & \cdots & 0 & & 0 & & * \\ & * & & 1 & & 1 & \cdots & & 1 & & * & \end{bmatrix} \tag{7.19}$$

where we do not specify the entries denoted by *, to exploit this freedom later. It is well-known that the ∇_{H_Q}-displacement rank of a sum $R = T + H$ of any Toeplitz and Hankel matrices does not exceed four, see, e.g., table 4 for references [reader can easily verify this fact by exploiting shift-invariances in $R = T + H$ along with the observation that H_Q is essentially a combination of lower and upper shifts]. A general transformation-to-Cauchy-and-pivoting scheme of Sec. 7.1 can be applied to Toeplitz-plus-Hankel-like matrices as well [GKO95], and it is based on the use of real discrete cosine/sine transforms [as opposed to the complex FFT used in Sec. 7.2].

We shall return to Toeplitz-plus-Hankel-like matrices in Sec. 7.7.4 after clarifying such transformation for the more general *polynomial-Hankel-like matrices* [KO96]. These matrices are defined via (7.18), but with a general upper *Hessenberg* matrix

$$H_Q = \begin{bmatrix} a_{01} & a_{02} & \cdots & & \cdots & a_{0,n} \\ a_{11} & a_{12} & \cdots & & \cdots & a_{1,n} \\ 0 & a_{22} & \cdots & & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & & \ddots & \vdots \\ 0 & \cdots & 0 & a_{n-1,n-1} & & a_{n-1,n} \end{bmatrix} \tag{7.20}$$

[of course, tridiagonal matrix (7.19) appears as a special case]. In [KO96] matrices having low ∇_{H_Q}-displacement rank have been *polynomial Hankel-like matrices*, because they can be seen as Hankel-like matrices related to polynomials $Q = \{Q_0(x), Q_1(x), ..., Q_n(x)\}$ defined by

$$x \cdot Q_{k-1}(x) = a_{k,k} \cdot Q_k(x) + a_{k-1,k} \cdot Q_{k-1}(x) + ... + a_{0,k} \cdot Q_0(x) \tag{7.21}$$

[Many useful results on connection of Hessenberg matrices H_Q to polynomials Q can be found in [MB79] where the name *confederate matrix* was associated with H_Q in (7.20)].

7.7.2 Generators for polynomial Hankel-like matrices

Since (7.18) has a nontrivial kernel, such polynomail-Hankel-like matrices R are partially reconstructible. Therefore we need a description for the kernel of ∇_{H_Q}, and it is provided by the next theorem.

Theorem 7.5 *Let us choose and fix an arbitrary invertible diagonal matrix W_Q, and denote by*

$$V_Q = \begin{bmatrix} Q_0(x_1) & Q_1(x_1) & \cdots & Q_{n-1}(x_1) \\ Q_0(x_2) & Q_1(x_2) & \cdots & Q_{n-1}(x_2) \\ \vdots & \vdots & & \vdots \\ Q_0(x_n) & Q_1(x_n) & \cdots & Q_{n-1}(x_n) \end{bmatrix} \tag{7.22}$$

a polynomial-Vandermonde matrix whose nodes $\{x_k\}$ are the zeros of the n-th polynomial $Q_n(x)$ in (7.21). Then the kernel of $\nabla_{H_Q}(\cdot)$ in (7.18) has the form

$$\mathcal{K} = \text{span}\{(H_Q^T)^k \cdot (V_Q^* W_Q^2 V_Q), \qquad k = 0, 1, \ldots n-1\}. \tag{7.23}$$

Furthermore, any $R_\mathcal{K} \in \mathcal{K} =$ Ker ∇_{H_Q} can be represented as

$$R_\mathcal{K} = \sum_{k=0}^{n-1} r_k \cdot Q_k(H_Q^T) \cdot (V_Q^T W_Q^2 V_Q). \tag{7.24}$$

Thus, a partially reconstructible polynomial Hankel-like matrix R in (7.18) can be represented by its generator $\{G, J, R_\mathcal{K}\}$ where the last matrix $R_\mathcal{R}$ is described by only n parametrs $\{r_k\}$ in (7.24). One special case will be considered in Sec. 7.7.4, and here we shall indicate another important one. Consider the simplest case $Q = \{\frac{1}{\sqrt{n}}, \frac{x}{\sqrt{n}}, \frac{x^2}{\sqrt{n}}, \ldots, \frac{x^{n-1}}{\sqrt{n}}, \frac{x^n - 1}{\sqrt{n}}\}$, then it is easy to see that $H_Q = Z_1$, the lower shift circulant matrix defined in (7.5). So, it is clear that the kernel of (7.18) are all circulants [cf. Sec. 7.4]. Moreover, in this case V_Q is just a DFT matrix $\mathcal{F} = \frac{1}{\sqrt{n}}(\omega^{ij})_{i,j=0}^{n-1}$ [with ω denoting the primitive n-th root of unity]. Chosing $W_Q = I$, we note that since $\mathcal{F}$ is unitary and $Q_k(H_Q) = \frac{1}{\sqrt{n}} Z_1^k$, the equation (7.24) reduces to the obviuous description of a circulant by the entries of its first column $\{\frac{r_k}{\sqrt{n}}\}$.

7.7.3 Transformations to Cauchy-like matrices

Now, our goal is to transform R in (7.18) to a Cauchy-like matrix, so we need to diagonalize H_Q. For this, an easily verified formula involving the *polynomial Vandermonde matrix* $V_Q(x)$ defined in (7.22) is readily available:

$$H_Q = V_Q^{-1} D_x V_Q, \qquad \text{with} \qquad D_x = \text{diag}(x_1, x_2, ..., x_n), \tag{7.25}$$

where $\{x_k\}$ are the zeros of $Q_n(x)$. This identity (7.25) can be rewritten as $H_Q = V_Q^{-1} W_Q^{-1} D_x W_Q V_Q$ [as above, W_Q is an arbitary diagonal matrix] which leads to the following transformation result.

Theorem 7.6 *Let R be a polynomial Hankel-like matrix in (7.18), given by its ∇_{H_Q}-generator $\{G, J, R_\mathcal{K}\}$, and W_Q denotes an arbitrary invertible diagonal matrix. Then $W_Q^{-T} V_Q^{-T} R V_Q^{-1} W_Q^{-1}$ is a Cauchy-like matrix with respect to*

$$\nabla_{D_x}(R) = D_x R - R D_x \qquad [\text{ with } D_x \text{ is given by (7.25)}]$$

with a ∇_{D_x}-generator

$$\{W_Q^{-T} V_Q^{-T} G, \quad J, \quad W_Q^{-T} V_Q^{-T} R_\mathcal{K} V_Q^{-1} W_Q^{-1}\}. \tag{7.26}$$

where

$$W_Q^{-T} V_Q^{-T} R_\mathcal{K} V_Q^{-1} W_Q^{-1} = \begin{bmatrix} r(x_1) & & \\ & \ddots & \\ & & r(x_n) \end{bmatrix}$$

where the diagonal entries are computed via a polynomial Vandermonde transform

$$\begin{bmatrix} r(x_1) \\ \vdots \\ r(x_n) \end{bmatrix} = V_Q \begin{bmatrix} r_0 \\ \vdots \\ r_{n-1} \end{bmatrix}.$$

with $\{r_k\}$ from (7.24).

Returning to our comment made after theorem 7.5 we may note that this theorem generalizes the well-known result on diagonalization of circulant matrices by the discrete Fourier transform. Another important special case is considered next.

7.7.4 Toeplitz-plus-Hankel-like matrices. Discrete Cosine and Sine transforms

Here we return to the special case (7.19) for the displacement operator (7.18) associated in Sec. 7.7.1 with Toeplitz-plus-Hankel matrices. Now, using definitions and notations of Sec. 7.7.1 we notice that the matrix H_Q in a (7.19) is a confederate matrix whose associated polynomial system (7.21) satisfies the three-term recurrence relations,

$$xQ_k(x) = \frac{Q_{k-1}(x) + Q_{k+1}(x)}{2}. \tag{7.27}$$

Of course, these are the well-known three-term recurrence relations for Chebyshev polynomials which are essentially cosines and sines:

$$T_k(x) = \cos(k \arccos x), \qquad U_k = \frac{\sin((k+1)\arccos x)}{\sin(\arccos x)}. \tag{7.28}$$

But here we have a complete freedom to fill in asterisks in (7.19) which means the freedom in choosing the first and the last polynomials of Q. For our purposes here it is convenient to make the following 8 choices listed in the second column of Table 9. The point in making these choices is that for these tridiagonal matrices H_Q we are able to write down the explicit expressions for the corresponding scaled polynomial Vandermonde matrices, $W_Q V_Q$, so that these matrices are just 8 versions of discrete cosine and sine transforms listed in Table 10.

There are well-known superfast $O(n \log n)$ and accurate real arithmetic algorithms for all 8 matrices $W_Q V_Q$ listed in the second column of Table 10. Therefore we are able to specify the results of Sec. 7.7 and to suggest a variety of real-arithmetic DCT/DST-based alternatives to the complex-arithmetic FFT-based transformation-and-pivoting methods of Sec. 7.2, 7.4. More details can be found in [KO96], where this transformation-to-Cauchy-like approach was used to derive in a uniform manner 8 DCT/DST-based analogs of T.Chan circulant preconditioner for Toeplitz linear equations [the usual T.Chan preconditioner is a circulant matrix, i.e. it is associated with the complex-arithmetic DFT].

8 Ordinary Cauchy matrices. Specific algorithms and predictive pivoting

The algorithms in Sec. 3.5, 5.3 were designed for the general Cauchy-like matrices, some applications for this general case were considered in Sec. 4 and 6. However, a special case of ordinary Cauchy matrices $C(x,y) = \left[\frac{1}{x_i - y_j} \right]$ is also of interest in several areas, see, e.g., [B37], [C41], [Ro85], [T86], [OS86], [R90a], [HLM95], [RS94], [MS77], [BGR90]. Recall that the ordinary Cauchy matrices have displacement rank one,

$$\nabla_{\{D_x, D_y\}}(C(x,y)) = D_x C(x,y) - C(x,y) D_y = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}. \tag{8.1}$$

and this fact can be exploited to obtain more specific fast algorithms and more elaborated stabilizing techniques often yielding a remarkably high numerical accuracy. These results and their implications [e.g., to divided differences] are described in the rest of the paper.

8.1 Direct generator recursion. GS-direct algorithm

To begin with, it is possible to implement the generator recursion (3.5) for $C(x,y)$ as a direct manipulation on the nodes $\{x_k\}$, $\{y_k\}$, as shown in the next statement.

Lemma 8.1 *(GS-direct algorithm)* *Let R_k in*

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

DCT-I	$H_Q = \text{ tridiag} \left[\begin{array}{ccccccccccc} & \frac{1}{\sqrt{2}} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \frac{1}{\sqrt{2}} & \\ 0 & & 0 & & 0 & \cdots & 0 & & 0 & & 0 \\ & \frac{1}{\sqrt{2}} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \frac{1}{\sqrt{2}} & \end{array} \right]$
DCT-II	$H_Q = \text{ tridiag} \left[\begin{array}{ccccccccccc} & \frac{1}{2} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \frac{1}{2} & \\ \frac{1}{2} & & 0 & & 0 & \cdots & 0 & & 0 & & \frac{1}{2} \\ & \frac{1}{2} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \frac{1}{2} & \end{array} \right]$
DCT-III	$H_Q = \text{ tridiag} \left[\begin{array}{ccccccccccc} & \frac{1}{\sqrt{2}} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \frac{1}{2} & \\ 0 & & 0 & & 0 & \cdots & 0 & & 0 & & 0 \\ & \frac{1}{\sqrt{2}} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \frac{1}{2} & \end{array} \right]$
DCT-IV	$H_Q = \text{ tridiag} \left[\begin{array}{ccccccccccc} & \frac{1}{2} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \frac{1}{2} & \\ \frac{1}{2} & & 0 & & 0 & \cdots & 0 & & 0 & & -\frac{1}{2} \\ & \frac{1}{2} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \frac{1}{2} & \end{array} \right]$
DST-I	$H_Q = \text{ tridiag} \left[\begin{array}{ccccccccccc} & \frac{1}{2} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \frac{1}{2} & \\ 0 & & 0 & & 0 & \cdots & 0 & & 0 & & 0 \\ & \frac{1}{2} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \frac{1}{2} & \end{array} \right]$
DST-II	$H_Q = \text{ tridiag} \left[\begin{array}{ccccccccccc} & \frac{1}{2} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \frac{1}{2} & \\ -\frac{1}{2} & & 0 & & 0 & \cdots & 0 & & 0 & & -\frac{1}{2} \\ & \frac{1}{2} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \frac{1}{2} & \end{array} \right]$
DST-III	$H_Q = \text{ tridiag} \left[\begin{array}{ccccccccccc} & \frac{1}{2} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \sqrt{\frac{1}{2}} & \\ 0 & & 0 & & 0 & \cdots & 0 & & 0 & & 0 \\ & \frac{1}{2} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \sqrt{\frac{1}{2}} & \end{array} \right]$
DST-IV	$H_Q = \text{ tridiag} \left[\begin{array}{ccccccccccc} & \frac{1}{2} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \frac{1}{2} & \\ -\frac{1}{2} & & 0 & & 0 & \cdots & 0 & & 0 & & \frac{1}{2} \\ & \frac{1}{2} & & \frac{1}{2} & & \cdots & & \frac{1}{2} & & \frac{1}{2} & \end{array} \right]$

Table 9: *Choices of H_Q in (7.19).*

	Discrete transform	Inverse transform
DCT-I	$C_N^I = \sqrt{\frac{2}{N-1}} \left[\eta_k \eta_{N-1-k} \eta_j \eta_{N-1-j} \cos \frac{kj\pi}{N-1} \right]_{k,j=0}^{N-1}$	$[C_N^I]^{-1} = [C_N^I]^T = C_N^I$
DCT-II	$C_N^{II} = \sqrt{\frac{2}{N}} \left[\eta_k \cos \frac{k(2j+1)\pi}{2N} \right]_{k,j=0}^{N-1}$	$[C_N^{II}]^{-1} = [C_N^{II}]^T = C_N^{III}$
DCT-III	$C_N^{III} = \sqrt{\frac{2}{N}} \left[\eta_j \cos \frac{(2k+1)j\pi}{2N} \right]_{k,j=0}^{N-1}$	$[C_N^{III}]^{-1} = [C_N^{III}]^T = C_N^{II}$
DCT-IV	$C_N^{IV} = \sqrt{\frac{2}{N}} \left[\cos \frac{(2k+1)(2j+1)\pi}{4N} \right]_{k,j=0}^{N-1}$	$[C_N^{IV}]^{-1} = [C_N^{IV}]^T = C_N^{IV}$
DST-I	$S_N^I = \sqrt{\frac{2}{N+1}} \left[\sin \frac{kj}{N+1}\pi \right]_{k,j=1}^{N}$	$[S_N^I]^{-1} = [S_N^I]^T = S_N^I$
DST-II	$S_N^{II} = \sqrt{\frac{2}{N}} \left[\eta_k \sin \frac{k(2j-1)}{2N}\pi \right]_{k,j=1}^{N}$	$[S_N^{II}]^{-1} = [S_N^{II}]^T = S_N^{III}$
DST-III	$S_N^{III} = \sqrt{\frac{2}{N}} \left[\eta_j \sin \frac{(2k-1)j}{2N}\pi \right]_{k,j=1}^{N}$	$[S_N^{III}]^{-1} = [S_N^{III}]^T = S_N^{II}$
DST-IV	$S_N^{IV} = \sqrt{\frac{2}{N}} \left[\sin \frac{(2k-1)(2j-1)}{4N}\pi \right]_{k,j=1}^{N}$	$[S_N^{IV}]^{-1} = [S_N^{IV}]^T = S_N^{IV}$

Table 10: *Discrete trigonometric transforms.*

be the successive Schur complements of the Cauchy matrix $C(x,y) = \left[\frac{1}{x_i - y_j} \right]$. *Then the generator recursion (3.8) can be specialized to*

$$\begin{bmatrix} g_{k+1}^{(k+1)} \\ \vdots \\ g_n^{(k+1)} \end{bmatrix} = \begin{bmatrix} \frac{x_{k+1}-x_k}{x_{k+1}-y_k} g_{k+1}^{(k)} \\ \vdots \\ \frac{x_n - x_k}{x_n - y_k} g_n^{(k)} \end{bmatrix}, \qquad \begin{bmatrix} b_{k+1}^{(k+1)} & \cdots & b_n^{(k+1)} \end{bmatrix} = \begin{bmatrix} \frac{y_{k+1}-y_k}{y_{k+1}-x_k} b_n^{(k)} & \cdots & \frac{y_n - y_k}{y_n - x_k} b_n^{(k)} \end{bmatrix}. \tag{8.2}$$

The nonzero entries of the factors in $C(x,y) = LDU$ *are given by* $d_k = x_k - y_k$ *and*

$$\begin{bmatrix} l_{k,k} \\ \vdots \\ l_{n,k} \end{bmatrix} = \begin{bmatrix} \frac{1}{x_k - y_k} g_k^{(k)} \\ \vdots \\ \frac{1}{x_n - y_k} g_n^{(k)} \end{bmatrix}, \qquad \begin{bmatrix} u_{k,k} & \cdots & u_{k,n} \end{bmatrix} = \begin{bmatrix} \frac{1}{x_k - y_k} b_k^{(k)} & \cdots \frac{1}{x_k - y_n} b_n^{(k)} \end{bmatrix} \tag{8.3}$$

A fast Cauchy solver based on recursion (8.2) (8.3) was called *GS-direct algorithm* in [BKO95b], and it requires only $6n^2$ flops, but it needs $O(n^2)$ locations of memory [to store the triangular factors of $C(x,y)$].

8.2 Quasi-Cauchy algorithm

A more efficient fast algorithm requiring just $O(n)$ memory locations is based on the next statement.

Lemma 8.2 *(quasi-Cauchy algorithm)* *The Cauchy matrix and its inverse can be factored as*

$$C(x,\, y) = L_1 \ldots L_{n-1} \, D \, U_{n-1} \ldots U_1, \tag{8.4}$$

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

$$L_k = \left[\begin{array}{c|ccc} I_{k-1} & & & \\ \hline & \frac{1}{x_k - y_k} & & \\ & & \ddots & \\ & & & \frac{1}{x_n - x_k} \end{array}\right] \left[\begin{array}{c|cccc} I_{k-1} & & & & \\ \hline & 1 & & & \\ & 1 & 1 & & \\ & \vdots & & \ddots & \\ & 1 & & & 1 \end{array}\right] \left[\begin{array}{c|cccc} I_{k-1} & & & & \\ \hline & 1 & & & \\ & & x_{k+1} - x_k & & \\ & & & \ddots & \\ & & & & x_n - x_k \end{array}\right],$$

$$U_k = \left[\begin{array}{c|cccc} I_{k-1} & & & & \\ \hline & 1 & & & \\ & & y_k - y_{k+1} & & \\ & & & \ddots & \\ & & & & y_k - y_n \end{array}\right] \left[\begin{array}{c|cccc} I_{k-1} & & & & \\ \hline & 1 & 1 & \dots & 1 \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{array}\right] \left[\begin{array}{c|ccc} I_{k-1} & & & \\ \hline & \frac{1}{x_k - y_{k+1}} & & \\ & & \ddots & \\ & & & \frac{1}{x_k - y_n} \end{array}\right].$$

The factorization (8.4) is easily deduced from Lemma 8.1 by factoring the successive Schur complements as

$$R_k = L_k C(x,y) U_k$$

at each step of the recursion [Here we use the same designation $C(x,y)$ for all $(n+1-k) \times (n+1-k)$ Cauchy matrices defined by the nodes $\{x_k, \dots, x_n\}$ and $\{y_k, \dots, y_n\}$, $(k = 1, \dots, n)$].

Each of the factors L_k, U_k cab be easily inverted by just changing the sign for the off-diagonal entries, thus leading to a new Cauchy solver. This fast Cauchy solver requires only $6n^2$ operations per system and only $O(n)$ memory locations. Because of a certain analogy with quasi-Toeplitz matrices, it was called *quasi-Cauchy* algorithm in [BKO95b].

8.3 Error analysis

An experience shows that without stabilizing techniques the GS-direct algorithm [based on Lemmas 8.1] and of quasi-Cauchy algorithm [based on Lemma 8.2] are often less accurate than the more expensive structure-ignoring GEPP. A remedy stabilizing the numerical performance of these fast algorithms will be suggested in Sec. 8.4 and 8.5. To motivate and justify this remedy we next present the results the error analysis. Here we assume the standard model of the floating point arithmetic (2.3) with unit roundoff u.

Theorem 8.3 *(Accuracy of the GS-direct algorithm)* *If the triangular factorization computed via the GS-direct algorithm [based on Lemma 8.1] is used to solve the associated linear system $C(x,y)a = f$ then the computed solution $\widehat{a}$ solves a nearby system $(C(x,y) + \Delta C)\widehat{a} = f$ with a backward error*

$$|\Delta C| \le ((10n-2)u + O(u^2))|L||DU|, \tag{8.5}$$

and a residual error

$$|C(x,y)\widehat{a} - f| \le ((10n-2)u + O(u^2))|L|\,|DU|\,|\widehat{a}|, \tag{8.6}$$

Theorem 8.4 *(Accuracy of quasi-Cauchy algorithm)* *The solution $\widehat{a}$ of a Cauchy linear system $C(x,y)a = f$ computed via the quasi-Cauchy algorithm [based on Lemma 8.2] solves a nearby system $(C(x,y) + \Delta C)\widehat{a} = f$ with a backward error*

$$|\Delta C| \le ((n^2 + 11n - 10)u + O(u^2))|L||DU|. \tag{8.7}$$

and a residual error

$$|C(x,y)\widehat{a} - f| \le ((n^2 + 11n - 10)u + O(u^2))|L|\,|DU|\,|\hat{a}|, \tag{8.8}$$

We refer to [BKO95b] for more error bounds and for complete proofs, and next we turn to the implications of these results to the enhancement of the numerical performance of these algorithms.

8.4 Numerical example: pivoting and GS-direct-Cauchy and quasi-Cauchy algorithms

The backward error bounds (8.5), (8.7) and the residual error bounds (8.6), (8.8) of the two new fast algorithms are similar to the analogous bounds (2.5) and (2.6) of the GE procedure. This observation indicates that various pivoting techniques, so successful for GE, will also stabilize the numerical performance of the GS-direct-Cauchy and quasi-Cauchy algorithms.

Recall that the entries of the first rows and columns are immediately available during the recursive GS-direct-Cauchy algorithm, see, e.g., (8.3). Therefore it is straightforward to incorporate pivoting into this algorithm, thus achieving its numerical accuracy. In contrast, the quasi-Cauchy algorithm is not recursive, so we exploit next a different approach to incorporate pivoting.

slow	fast	fast		fast	
GEPP	inversion	GS-direct		quasi-Cauchy	
		partial pivoting	no pivoting	partial pivoting	no pivoting
1e-06	6e-03	7e-07	2e-04	6e-07	4e-4

Table 11: *Cauchy-Toeplitz matrix* $[1/(1-0.3(i-j))]$.

8.5 Predictive pivoting.

We started the paper with an example of a Vandermonde matrix, $V(x)$, for which partial pivoting ordering of the rows can be achieved *in advance* by computing in $O(n^2)$ operations the *Leja ordering* (1.2) of the nodes $\{x_k\}$. This possibility is quite useful, because it allows us to incorporate pivoting into *any* fast $O(n^2)$ algorithm for $V(x)$, without slowing it down. Ordinary Vandermonde and ordinary Cauchy matrices have many similar properties, for example they both have displacement rank one. Therefore, a natural question is: can we design a similar fast $O(n^2)$ algorithm to compute in advance a partial pivoting ordering of the rows for an ordinary Cauchy matrix? This would allow us to incorporate partial pivoting into any fast $O(n^2)$ Cauchy solver, e.g., into quasi-Cauchy algorithm.

As was noted in Sec. 2.4, partial pivoting of the rows of R successively maximizes the determinants of its leading submatrices. Vandermonde and Cauchy matrices have two nice properties in common: first, their leading submatrices have the same pattern of structure; and secondly, there are well-known explicit formulas for their determinants in terms of the nodes, e.g.,

$$\det C(x,y) = \frac{\prod_{j>k}(x_j - x_k) \prod_{j<k}(y_j - y_k)}{\prod_{j,k}(x_j - y_k)} \qquad 1 \le j,k \le n. \tag{8.9}$$

This observation was used in Sec. 2.4 to to express partial pivoting of the rows for $V(x)$ as Leja ordering (1.2). Analogously, partial pivoting for $C(x,y)$ can be obtained as a successive maximization of the quantities

$$|d_i| = \left|\frac{\prod_{j=1}^{i-1}(x_i - x_j) \prod_{j=1}^{i-1}(y_i - y_j)}{(x_i - y_i) \prod_{j=1}^{i-1}(x_i - y_j) \prod_{j=1}^{i-1}(x_j - y_i)}\right|, \qquad (i = 1, \dots, n). \tag{8.10}$$

This procedure has been called *predictive partial pivoting* [BKO95a], because it can be rapidly computed *in advance* in only $O(n^2)$ flops.

Predictive partial pivoting stabilizes in practice the numerical behavior of the GS-direct and quasi-Cauchy algorithms, as illustrated in Table 8.5 containing the backward error for a 100×100 Cauchy-Toeplitz matrix $\left[\frac{1}{1-0.3(i-j)}\right]$.

Note that the approach allows us to incorporate other pivoting techniques, e.g., Gu's pivoting, see, e.g., [BKO95b] for comparison and references.

8.6 Totally positive matrices. Avoidance of pivoting

Let us now again return to the example of $V(x)$ given in section 1, and recall that for totally positive Vandermonde matrices,

$$0 < x_1 < x_2 < \dots < x_n, \tag{8.11}$$

it is advantageous not to pivot when using structure-ignoring Gaussian elimination, see, e.g., the discussion at the end of Sec. 1 and the error bound (2.8) in Sec. 2.3. A natural question would be: Is there an analogue of these results for our fast algorithms for Cauchy matrices $C(x,y)$? We next present an affirmative answer to this question.

It is known [GK50] that the condition

$$y_n < \dots y_1 < x_1 < \dots < x_n \tag{8.12}$$

is equivalent to the total positivity of $C(x,y)$. Under this condition the results of Theorems 8.3 and 8.4 imply that not only slow GE, but also the new fast GS-direct-Cauchy and quasi-Cauchy algorithms compute solutions with a favorable small backward error.

Theorem 8.5 *(Backward stability of GS-direct algorithm for totally positive matrices) Assume that condition (8.12) holds, i.e. that $C(x,y)$ is totally positive. If the triangular factorization of the GS-direct algorithm is used to solve the associated linear system, then the computed solution $\widehat{a}$ solves a nearby system*

$$(C(x,y) + \Delta C)\widehat{a} = f,$$

with

$$|\Delta C| \leq ((10n-3)u + O(u^2))C(x,y). \tag{8.13}$$

The analogous backward bound for the quasi-Cauchy algorithm is

$$|\Delta C| \leq ((n^2 + 11n - 10)u + O(u^2))C(x,y). \tag{8.14}$$

The above results show that the backward stability of the fast GS-direct and quasi-Cauchy algorithms for totally positive Cauchy matrices is even more favorable than that of the slow Gaussian elimination procedure. Indeed the difference is that for the GE procedure the bound (2.8) is valid only for the case when the entries of the *computed factors* $\widehat{L}$ and $\widehat{U}$ remain positive (which is usually not the case with ill-conditioned matrices), whereas the favorable bounds in the above theorem hold as long as 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 $\widehat{L}$, $\widehat{U}$ became negative, so the pleasing backward bound (2.8) is no longer valid for Gaussian elimination. In contrast, the favorable bounds (8.13), (8.14) are valid for higher sizes, as long as there are no overflows.

For totally positive Cauchy matrices the GS-direct and the quasi-Cauchy algorithms have a remarkably small backward error, but their *forward accuracy* is not as high as the one of another new Cauchy solver, described in the next section.

9 Totally positive structured matrices and avoidance of pivoting: Björck-Pereyra-type algorithms

9.1 The Björck-Pereyra (BP) algorithm for Vandermonde matrices

We again return to the example of a Vandermonde matrix of Sec. 1, and now discuss the Björck-Pereyra (BP) algorithm [BP70]. This algorithm is based on the explicit decomposition of the inverse of a Vandermonde matrix into the product

$$V(x)^{-1} = U_1 \cdot \ldots \cdot U_{n-1} \cdot L_{n-1} \cdot \ldots \cdot L_1, \tag{9.1}$$

of the upper and lower *bidiagonal* factors [the entries of L_k, U_k are specified in [BP70] explicitly, without computation]. The BP algorithm solves the associated linear system by multiplying the representation (9.1) by a right-hand-side, and the sparse bidiagonal structure of the factors L_k, U_k yields a favorable efficiency of $\frac{5}{2}n^2$ operations per system. This algorithm is now a well-known example where the exploitation of the structure allows one not only to speed-up computations, but also to achieve, for special right-hand sides, more accuracy in the computed solution than when using slow structure-ignoring methods. The first indication of this interesting phenomena can be found in [BP70], where it was observed that *"some problems, connected with Vandermonde systems, which traditionally have been considered to be too ill-conditioned to be attacked, actually can be solved with good precision"*.

This fact attracted much attention and motivated a number of papers appearing in the last decade. Specifically, many efforts were devoted to the extension of the Björck-Pereyra algorithm to more general classes of matrices. Thus, W.Tang and G.Golub [TG81] devised a closely related algorithm for block Vandermonde matrices, N.J.Higham extended in [Hig88] this algorithm to the so-called Vandermonde-like matrices involving orthogonal polynomials[7], and L.Reichel and G.Opfer [RO91] specified a progressive [i.e., allowing updating] Björck-Pereyra-type algorithm for Chebyshev-Vandermonde matrices.

Another challenging problem was to give a theoretical support for the favorable numerical properties of the Björck-Pereyra algorithm by performing an *a priori* rounding error analysis, the results in this direction are given next.

- **Forward error bound.** It was shown in [Hig87] that for totally positive Vandermonde systems $V(x)a = f$ the forward error of the BP algorithm is nicely bounded:

$$|a - \widehat{a}| \leq 5nu|V(x)^{-1}||f| + O(u^2). \tag{9.2}$$

- **Backward error bound.** In [BKO95a] it was shown that for totally positive Vandermonde matrices not only forward, but also backward, error is remarkable small:

$$|\Delta V| \leq 12n^2 u V(x)|\widehat{a}| + O(u^2). \tag{9.3}$$

Here we may recall Wilkinson's [W71] advocation not to give *" ... much importance to the precise error bounds obtained by a priori error analysis,"* even they are often impractically large, they can *"expose the potential instabilities, if any, of an algorithm, so that hopefully from the insight thus obtained one might be led to improved algorithms."* In contrast to many such impractical (though still useful) bounds, the bounds in (9.2), (9.3)are surprisingly favorable, predicting that the Björck-Pereyra algorithm can produce all 7 possible-in-single-precision digits, even in situations where Gaussian elimination with complete pivoting will fail to produce as much as one correct digit. For example, the total positivity implies that for the sign-oscillating right-hand-side we have $|V(x)^{-1}||f| = |V(x)^{-1}f|$, so (9.2) becomes

$$|a - \widehat{a}| \leq 5nu|a| + O(u^2), \tag{9.4}$$

see, e.g., [Hig87], implying the full relative accuracy.

9.2 The BP-type algorithm for Cauchy matrices and numerical example

As we already several times exploited an analogy between ordinary Vandermonde and Cauchy matrices, we naturally arrive at the following question: is there a counterpart of the BP algorithm for Cauchy matrices, and if yes, does it have a similar remarkable high forward and backward numerical accuracy? In this section we present affirmative answers to these questions. First we formulate the analog of the decomposition (9.1) into the product of *bidiagonal* factors.

Lemma 9.1 *The inverse of the Cauchy matrix $C(x,\ y)$ can be decomposed as*

$$C^{-1}(x,\ y) = U_1\, U_2\, \ldots\, U_{n-1}\, D\, L_{n-1}\, \ldots\, L_2\, L_1, \tag{9.5}$$

where

$$L_k = \left[\begin{array}{c|cccc} I_k & & & & \\ \hline & \frac{1}{x_{k+1}-x_1} & & & \\ & & \frac{1}{x_{k+2}-x_2} & & \\ & & & \ddots & \\ & & & & \frac{1}{x_n - x_{n-k}} \end{array}\right] \left[\begin{array}{c|cccc} I_{k-1} & & & & \\ \hline & 1 & 0 & & \\ & -(x_1 - y_k) & (x_{k+1} - y_k) & & \\ & & \ddots & \ddots & \\ & & & -(x_{n-k} - y_k) & (x_n - y_k) \end{array}\right], \tag{9.6}$$

$$U_k = \left[\begin{array}{c|cccc} I_{k-1} & & & & \\ \hline & 1 & -(x_k - y_1) & & \\ & 0 & (x_k - y_{k+1}) & \ddots & \\ & & & \ddots & -(x_k - y_{n-k}) \\ & & & & (x_k - y_n) \end{array}\right] \left[\begin{array}{c|cccc} I_i & & & & \\ \hline & \frac{1}{y_1 - y_{k+1}} & & & \\ & & \frac{1}{y_2 - y_{k+2}} & & \\ & & & \ddots & \\ & & & & \frac{1}{y_{n-k} - y_n} \end{array}\right] \tag{9.7}$$

$$D = \operatorname{diag}\{(x_1 - y_1), (x_2 - y_2), ..., (x_n - y_n)\}. \tag{9.8}$$

[7]We call them three-term Vandermonde matrices, because we use the postfix *"like"* in a different meaning, which it has in the context of displacement structure theory.

Using this representation the associated linear system can be solved in $7n^2$ operations.

An error analysis of [BKO95a] produced the following counterparts of the bounds (9.2) and (9.3):

Theorem 9.2 *Suppose that the condition $y_n < \ldots < y_1 < x_1 < \ldots < x_n$ holds [so that $C(x,y)$ is totally positive, and the BP-type algorithm is carried out in floating-point arithmetic with unit round-off u, and that no overflows were encountered during computation. Then the computed solution $\widehat{a}$ to $C(x,y)a = f$ satisfies*

1. **Forward bound**
$$|a - \widehat{a}| \leq 5(2n+1) \cdot u \cdot |C(x,y)^{-1}||f| + \mathcal{O}(u^2) , \tag{9.9}$$

2. **Backward bound** *The computed solution $\widehat{a}$ is an exact solution of a nearby system $\hat{C}\widehat{a} = f$ with*
$$|C(x,y) - \widehat{C}| \leq c_n \cdot u \cdot C(x,y) + \mathcal{O}(u^2) , \tag{9.10}$$
where $c_n = 20n(2n-1)$.

3. **Residual bound**
$$|f - C(x,y)\widehat{a}| \leq c_n \cdot u \cdot C(x,y)|\widehat{a}| + \mathcal{O}(u^2) , \tag{9.11}$$

For example, similarly to (9.4) we have that for the sign-oscillating right-hand side we have
$$|a - \widehat{a}| \leq 5(2n+1)u|a| + O(u^2). \tag{9.12}$$

These results predict a remarkable small errors in the solution, computed via the BP-type algorithm, as illustrated in Figure 1 and Table 12 next example.

n	10	20	30	40	50	60
$\kappa_\infty(C(x,y))$	2e+08	4e+15	4e+21	4e+22	4e+22	7e+22

Table 12: *Conditioning of Cauchy matrix with $x_i = i^4/n^4$, $y_i = -i^4/n^4$.*

We see that the GE with complete pivoting fails to provide even one correct digit in the computed solution, which is expectable, given extremely large condition number. Nevertheless, the BP-type algorithm with monotonically ordered nodes (8.12) [thus preserving total positivity] gives us the full relative accuracy [i.e., about 7 digits out of about 7 possible in single precision].

9.3 Effective stability and numerical example

In [CF88] Chan and Foulser, motivated by the high accuracy of the BP algorithm for sign-oscillating right-hand side [see, e.g., (9.4)], and by some other examples, introduced the concept of *effective well-conditioning.* The latter reflects the fact that the sensitivity to perturbations of a solution depends upon the direction of the right-hand side vector, which suggests the potential existence of algorithms that can exploit the effective well-conditioning to produce higher accuracy for special right-hand sides. Note, however, that in general the numerical performance of a certain algorithm and the effective well-conditioning of a system to be solved have nothing to do with each other, and many algorithms are insensitive to the direction of the right-hand side vector. For example, for Gaussian elimination this point is illustrated by the numerical examples below. Here we use the approach of [CF88] to show that our new BP-type Cauchy solver is indeed an example of an algorithm that is guaranteed to accurately solve effectively well-conditioned Cauchy systems with totally positive coefficient matrices.

Before stating the result, let us introduce the necessary notations. Let
$$\kappa_2(A,f) = \lim_{\varepsilon \to 0} \sup_{\|\triangle f\|_2 \leq \varepsilon \|f\|_2} \frac{\|\triangle a\|_2}{\|a\|_2 \cdot \varepsilon}, \tag{9.13}$$
measures, in the 2-norm, the sensitivity of the solution of $Aa = f$ to small perturbations in the right hand-side. Chan and Foulser derived an upper bound for $\kappa_2(A,f)$ in terms of the direction of the right-hand side vector relative to the left singular vectors of A. To be more precise, let $A = U \cdot \Sigma \cdot V^T$

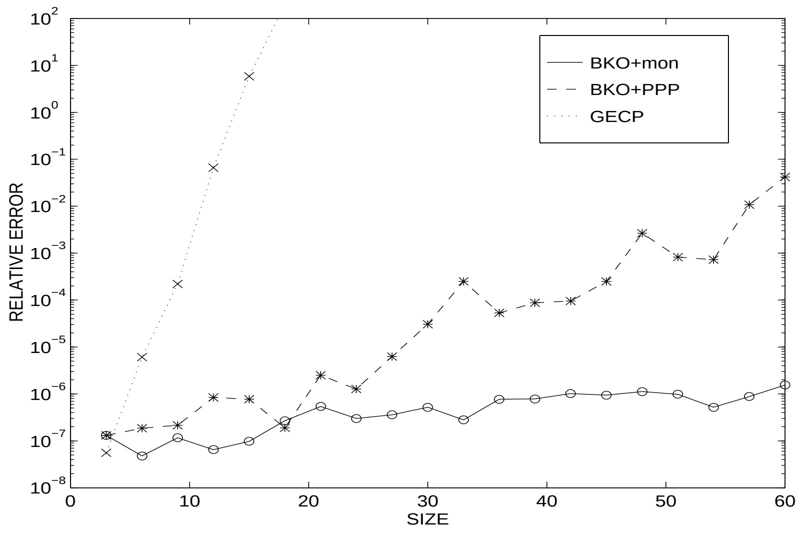

Figure 1: *Cauchy linear system with $x_i = i^4/n^4$, $y_i = -i^4/n^4$, and $f_i = (-1)^i$, $(i = 1, \ldots, n)$. The graphs display the relative error $\|a - \hat{a}\|_\infty / \|a\|_\infty$ as a function of n, for the following three algorithms :* **BKO+mon:** *BKO algorithm with monotonically ordered nodes,* **BKO+PPP:** *BKO algorithm with predictive partial pivoting,* **GECP:** *Gaussian elimination with complete pivoting.*

be the singular value decomposition, where the columns of $U = \begin{bmatrix} u_1 & u_2 & \cdots & u_n \end{bmatrix}$ are the left singular vectors, the columns of $V = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$ are the right singular vectors, and the diagonal entries of $\Sigma = \operatorname{diag}\{ \sigma_1 \quad \sigma_2 \quad \cdots \quad \sigma_n \}$ are the singular values of A in decreasing order. Denote by $P_k = \begin{bmatrix} u_{n+1-k} & \cdots & u_n \end{bmatrix} \cdot \begin{bmatrix} u_{n+1-k} & \cdots & u_n \end{bmatrix}^T$ the projection operator onto the linear span of the smallest k left singular vectors of A. Then, in accordance with [CF88],

$$\kappa_2(A, f) \leq \gamma(A, f) \leq \kappa_2(A), \tag{9.14}$$

where

$$\gamma(A, f) = \min_k \frac{\sigma_{n-k+1}}{\sigma_n} \cdot \frac{\|f\|_2}{\|P_k f\|_2}, \tag{9.15}$$

and $\kappa_2(A) = \|A^{-1}\|_2 \cdot \|A\|_2$ is the 2-norm condition number. Note that the second inequality in (9.14) can easily be deduced by inspecting (9.15) for $k = n$. At the same time, if a large part of f lies in the span of the small left singular vectors of A, then $\gamma(A, f)$ can be much smaller than $\kappa_2(A)$, thus providing a better bound for $\kappa_2(A, f)$ in (9.14). Linear systems for which the *Chan-Foulser number* in (9.15) is much smaller than $\kappa_2(A)$ are called *effectively well-conditioned.* We shall refer to an algorithm as *effectively (forward) stable*, if it is guaranteed to produce high relative accuracy for effectively well-conditioned systems.

It was shown in [BKO95a] imply that the relative error in the solution computed via the new BP-type Cauchy solver is bounded by a multiple of the Chan-Foulser number :

$$\frac{\|a - \hat{a}\|_\infty}{\|a\|_\infty} \leq (19n + 2)\sqrt{n} \cdot \gamma(C(x, y), f) \cdot u + \mathcal{O}(u^2). \tag{9.16}$$

This means that

$$\boxed{\textit{the new BP-type Cauchy solver m is effectively stable for totally positive Cauchy systems.}}$$

Table 13 below illustrates these results with a computed example, in which we solved sixteen linear systems with 16×16 Hilbert coefficient matrix using its seven left singular vectors u_k for the right-hand sides.

	Conditioning		Relative error	
i	$\kappa_2(C)$	$\sqrt{n}\cdot\gamma(C,f)$	BP-type	GEPP
1	5e+17	4e+16	1e+00	4e+04
2	5e+17	4e+16	1e+00	1e+04
3	5e+17	4e+16	1e+00	6e+03
4	5e+17	6e+15	1e+00	2e+04
5	5e+17	5e+14	1e+00	3e+04
6	5e+17	3e+13	1e+00	1e+05
7	5e+17	2e+12	1e+00	8e+03
8	5e+17	8e+10	1e+00	2e+04
9	5e+17	3e+09	6e-01	3e+03
10	5e+17	9e+07	7e-01	1e+05
11	5e+17	2e+06	2e-03	3e+08
12	5e+17	4e+04	4e-04	1e+09
13	5e+17	5e+02	2e-05	7e+09
14	5e+17	1e+01	3e-07	9e+12
15	5e+17	5e+00	1e-07	3e+12
16	5e+17	4e+00	4e-08	4e+12

Table 13: *Left singular vectors for the right-hand sides.*

The numerical data do not demonstrate any dependence of the accuracy of Gaussian elimination with complete pivoting (GECP) upon the direction of the right-hand side vector. Moreover, in all 16 cases the GECP algorithm does not produce even one correct digit in the solution from about 7 possible-in-single-precision, consistent with the large condition number of the coefficient matrix, displayed in the second column.

The same table 13 shows that the numerical behavior of the BP-type Cauchy solver indeed depends upon the direction of the right-hand side vector, thus confirming the analytical results in (9.16). Moreover, when the largest eight left singular vectors $u_1, ..., u_8$ are used for the right-hand side, then the Chan-Foulser number $\gamma(C(x,y),f)$, displayed in the 3-rd column of Table 13, is bigger than the reciprocal of the machine precision, $\frac{1}{u} \approx 10^8$. Correspondingly, the BP-type algorithm performs similar to GECP. At the same time, when we used three smallest left singular vectors u_{14}, u_{15} and u_{16} for the right-hand sides, the Chan-Foulser number $\sqrt{n}\cdot\gamma(C(x,y),f)$ is of the order of unity and much smaller that $\kappa_2(C(x,y))$. Correspondingly, the BP-type Cauchy solver performs much better than GECP, giving now about 7 correct digits from about 7 possible-in-single-precision.

[Here we may note that the solution $\widehat{a}$ of a Vandermonde system computed by the original BP algorithm satisfies

$$\frac{\|a-\hat{a}\|_\infty}{\|a\|_\infty} \leq 5n\sqrt{n}\cdot\gamma(V(x),f)\cdot u + \mathcal{O}(u^2), \tag{9.17}$$

see, e.g., [BKO95a]. The latter bound proves that *the BP algorithm is also effectively stable for totally positive Vandermonde systems.* This conclusion justifies the motivation of [CF88], and gives a theoretical support for the numerical examples in [V93].]

10 Pivoting and backward stability of polynomial and rational divided difference schemes. Updating and downdating schemes

10.1 Relation of the BP algorithm to divided differences

As was noted, the BP algorithm was based on the decomposition of the inverse of the Vandermonde matrix,

$$V(x)^{-1} = U_1\cdot\ldots\cdot U_{n-1}\cdot\tilde{L}_{n-1}\cdot\ldots\cdot\tilde{L}_1. \tag{10.1}$$

The particular form of the upper triangular factors U_k is not relevant for our purposes here, and we shall be interested only in the structure of the lower triangular factors:

$$\tilde{L}_k = \left[\begin{array}{c|cccc} I_k & & & \\ \hline & \frac{1}{x_{k+1}-x_k} & & \\ & & \ddots & \\ & & & \frac{1}{x_n-x_{n-k}} \end{array}\right] \cdot \left[\begin{array}{c|cccc} I_{k-1} & & & & \\ \hline & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{array}\right]. \tag{10.2}$$

In fact, Björck and Pereyra formulated the first part of their algorithm,

$$\begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = \tilde{L}_{n-1} \cdot \ldots \cdot \tilde{L}_1 \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \tag{10.3}$$

as a matrix interpretation of the well-known divided differences scheme, shown in Figure 2. We call this classical scheme *updating*, to distinguish it from an alternative *downdating* scheme to be introduces later in this section.

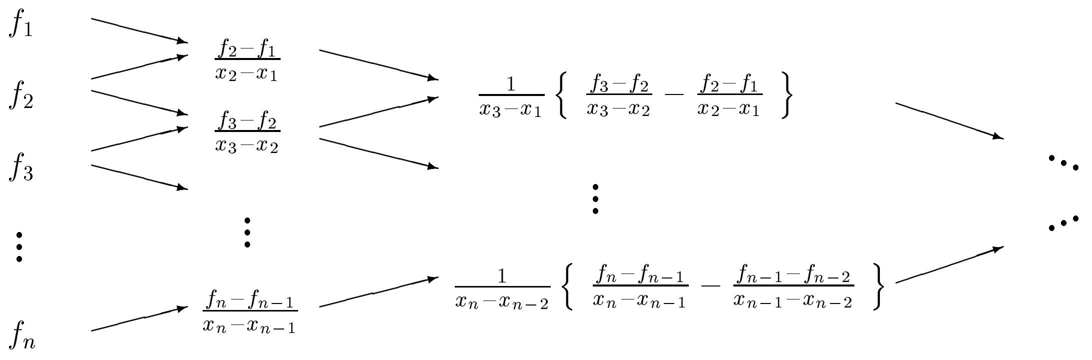
f_1
f_2
f_3
f_n
$\frac{f_2-f_1}{x_2-x_1}$
$\frac{f_3-f_2}{x_3-x_2}$
$\frac{f_n-f_{n-1}}{x_n-x_{n-1}}$
$\frac{1}{x_3-x_1}\left\{\frac{f_3-f_2}{x_3-x_2}-\frac{f_2-f_1}{x_2-x_1}\right\}$
$\frac{1}{x_n-x_{n-2}}\left\{\frac{f_n-f_{n-1}}{x_n-x_{n-1}}-\frac{f_{n-1}-f_{n-2}}{x_{n-1}-x_{n-2}}\right\}$

Figure 2: *Updating divided differences.*

The top row of Figure 2 gives us the coefficients $c_1 = f_1, c_2 = \frac{f_2-f_1}{x_2-x_1}, \ldots$, in the classical Newton representation

$$f(x) = \sum_{i=1}^{n} c_i \cdot \prod_{k=1}^{i-1} (x - x_k) \tag{10.4}$$

for the interpolating polynomial satisfying

$$f(x_k) = f_k, \qquad (k = 1, 2, \ldots, n).$$

10.2 Downdating divided differences

As we shall see in a moment, the error analysis for the above classical updating scheme will indicate to certain limitations in using pivoting with this algorithm, so it will be reasonable to look for an alternative. To obtain such an alternative we notice that the formula (10.1) can be obtained by using purely matrix arguments, as just an implementation of Gaussian elimination on $V(x)$, cf. with [TG81]. The bidiagonal structure of $\tilde{L}_k$ means that in this elimination scheme we use the k-th entry in the first column to eliminate the next $(k+1)$-st entry, this variant was called the Neville elimination in [GP92]. An alternative scheme can be obtained by using just standard pivotal scheme of Gaussian elimination on $V(x)$, where we use the same (1,1) entry to eliminate all other entries in the first column. As a result we obtain almost the same decomposition

$$V(x)^{-1} = U_1 \cdot \ldots \cdot U_{n-1} \cdot L_{n-1} \cdot \ldots \cdot L_1. \tag{10.5}$$

$$\begin{tikzcd}
f_1 \arrow[dr] \arrow[ddr] \arrow[ddddr] & & & \\
f_2 \arrow[r] & \frac{f_2-f_1}{x_2-x_1} \arrow[dr] \arrow[dddr] & & \\
f_3 \arrow[r] & \frac{f_3-f_1}{x_3-x_1} \arrow[r] & \frac{1}{x_3-x_2}\left\{ \frac{f_3-f_1}{x_3-x_1} - \frac{f_2-f_1}{x_2-x_1} \right\} \arrow[dr] & \\
\vdots & \vdots & \vdots & \ddots \\
f_n \arrow[r] & \frac{f_n-f_1}{x_n-x_1} \arrow[r] & \frac{1}{x_n-x_2}\left\{ \frac{f_n-f_1}{x_n-x_1} - \frac{f_2-f_1}{x_2-x_1} \right\} \arrow[r] & \cdots
\end{tikzcd}$$

Figure 3: *Downdating divided differences.*

but now with slightly different lower triangular factors

$$L_k = \begin{bmatrix} 1 & & & \\ & \frac{1}{x_2-x_1} & & \\ & & \ddots & \\ & & & \frac{1}{x_3-x_1} \end{bmatrix} \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ \vdots & & \ddots & \\ -1 & & & 1 \end{bmatrix}, \tag{10.6}$$

see, e.g., [KO96b]. Since the L factor the LU factorization for $V(x) = LU$ is unique [up to a diagonal factor], the computation

$$\begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = L_{n-1} \cdot \ldots \cdot L_1 \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix},$$

with L_k as in (10.6) produces the same result, i.e., the vector c of the coefficients in (10.4), thus giving rise to a new, *downdating* divided difference table:

The justification for the nomenclatures for these two divided differences Tables is given next.

10.3 Polynomial interpretation

Denote by $P_{i_1 i_2 \ldots i_k}(x)$ the k-th degree Newton polynomial interpolating at k points $\{x_{i_1}, x_{i_2}, ..., x_{i_k}\}$. Then the standard divided differences in Figure 2 have an *updating* nature, recursively computing $P_{i,i+1,\ldots,i+k}(x) \to P_{i,i+1,\ldots,i+k+1}(x)$, see, e.g., [SB80]. For example, the first step updates $P_i(x) = f_i \to P_{i,i+1}(x) = f_i + \frac{f_{i+1}-f_i}{x_{i+1}-x_i}(x - x_i)$ for $i = 1, 2, ..., n-1$. Note that the interpolation data remain unchanged.

In contrast, the non-standard divided differences in Figure 3 has a *downdating* nature, recomputing the interpolation data as follows. The desired Newton interpolating polynomial clearly has the form $f(x) = f_1 + (x - x_1) \cdot g(x)$, with some $g(x)$. The Figure 3 starts with the data $\{x_i, f_i\}_{i=1}^n$ for $f(x)$, and then it recomputes the interpolating data $\{x_i, \frac{f_i-f_1}{x_i-x_1}\}_{i=2}^n$ for $g(x)$, i.e., it removes one interpolation point. Then we proceed with $g(x)$ recursively.

Both divided differences schemes have the same computational complexity $O(n^2)$ operations, however the downdating scheme in Figure 3 allows us a suitable reordering of the points $\{x_k\}$ to achieve a better backward stability of the algorithm, as explained next.

10.4 Backward stability of downdating divided differences

In fact, the above interpolation interpretation is simpler than its matrix formulation, however the latter seems to be more useful for the analysis of numerical performance, and its provides an insight on how to numerically stabilize the algorithm. Recall that the standard updating divided differences are computed by $c = \tilde{L}_{n-1} \cdot \ldots \cdot \tilde{L}_1 \cdot f$, with $\tilde{L}_k$ as in (10.2), and that the rounding error analysis for it can be found, e.g., in

[Hig96]. It turns out that in the standard model of floating point arithmetic (2.3) the vector $\widehat{c}$ of computed coefficients of (10.4) satisfies

$$|f - L\cdot\widehat{c}| \le ((1-u)^{-3(n-1)} - 1)||\tilde{L}_1^{-1}|\cdot\ldots\cdot|\tilde{L}_{n-1}^{-1}|\cdot|\widehat{c}|. \tag{10.7}$$

Denoting by

$$L = \tilde{L}_1^{-1}\cdot\ldots\cdot\tilde{L}_{n-1}^{-1}, \tag{10.8}$$

a lower triangular factor of the Vandermonde matrix $V(x) = LU$, we notice that because of possible cancelation the entries this product can be much smaller then the entries of the product of the moduli of the same matrices, i.e.

$$|L| \le |\tilde{L}_1^{-1}|\cdot\ldots\cdot|\tilde{L}_{n-1}^{-1}|.$$

Now observe that for a given interpolation data $\{x_i, f_i\}_{i=1}^n$ there are many different Newton polynomials of the form (10.4), depending upon different orderings of interpolation points, and that these different forms may have different numerical properties. It was noted in [Hig96], p.111 that if interpolation points are monotonically ordered, $x_1 < x_2 < \ldots < x_n$, then the sign pattern of the factors (10.2) prevents any cancelation, allowing one to replace (10.7) by a better bound

$$|f - L\cdot\widehat{c}| \le ((1-u)^{-3(n-1)} - 1)|\cdot|L|\cdot|\widehat{c}|. \tag{10.9}$$

Denoting by $\widehat{f} = L\cdot\widehat{c}$ the vector of the values of the *actually computed* Newton polynomial

$$\widehat{f}(x) = \sum_{i=1}^{n}\widehat{c}_i\cdot\prod_{k=1}^{i-1}(x - x_k)$$

at interpolation points $\{x_k\}$, we can rewrite (10.9) in a more clear form

$$|f - \widehat{f}| \le 3nu|\cdot|L|\cdot|\widehat{c}| + O(u^2). \tag{10.10}$$

Note, however, that $|L|$ still can be large with monotonic ordering, and that the bidiagonal structure of $\tilde{L}_k$ do not allow one to formulate the attractive bound (10.10) for an arbitrary ordering.

Now turn to the new downdating divided difference table, for which we can derive a similar to (10.7) bound

$$|f - L\cdot\widehat{c}| \le ((1-u)^{-3(n-1)} - 1)||L_1^{-1}|\cdot\ldots\cdot|L_{n-1}^{-1}|\cdot|\widehat{c}|, \tag{10.11}$$

but now with the factors of the form (10.6). In contrast to the factors $\tilde{L}_k$, whose bidiagonal structure invalidates, in general, the nice bound (10.10), the particular sparsity pattern of L_k *always* implies (10.9) [because $|L| = |L_1^{-1}|\cdot\ldots\cdot|L_{n-1}^{-1}|$]. This means that with the downdating divided differences table we have a complete freedom of rearrangements of the $\{x_i\}$, and can use this freedom to reduce the size of $|L|$ in (10.10) to achieve a better numerical stability. Recalling that

$$L = \breve{L}\cdot\text{diag}(1,\ \ (x_2 - x_1),\ \ \ldots,\ \ \prod_{k=1}^{n-1}(x_n - x_k)).$$

where $\breve{L}$ is the *unit* [i.e., with ones on the main diagonal] lower triangular factor in the LU decomposition $V_P(x) = \breve{L}U$, and taking an advantage of the observation that rearrangement of $\{x_i\}$ is equivalent to a row permutation of $V_P(x)$, we arrive at the conclusion that the *partial pivoting* ordering guarantees that the entries of $\breve{L}$ is less than unity in magnitude. Therefore (10.11) yields a a better backward stability of the downdating divided differences :

$$|f_k - \widehat{f}_k| \le 3nu\sum_{i=1}^{k}|\widehat{c}_i|\cdot|\prod_{j<i}(x_i - x_j)| + O(u^2). \tag{10.12}$$

A comparison of the bounds (10.10) and (10.12) suggests that the downdating divided differences table with partial pivoting is more attractive then the [classical] updating one, as long as the backward stability is concerned. Moreover, partial pivoting for Vandermonde matrices is equivalent to Leja ordering (1.2) for which we have a fast $O(n^2)$ algorithm [Hig90], so its incorporation will not essentially slow down the overall algorithm.

10.5 Rational divided differences

The above discussion concerns two different algorithms to find the interpolating polynomial in the Newton form, i.e., in this problem we were looking for an interpolant the linear space with the basis

$$\{1,\ (x-x_1),\ (x-x_1)(x-x_2),\ \ldots,\ (x-x_1)\cdot\ldots\cdot(x-x_{n-1})\}. \tag{10.13}$$

Other bases of functions, not necessarily polynomials, are also of interest. Say, finding an interpolant with respect to the basis

$$\{\frac{1}{x-y_1},\ \ldots,\ \frac{1}{x-y_n}\} \tag{10.14}$$

[i.e., among rational functions with the fixed poles $\{y_k\}$] is equivalent to solving a Cauchy linear system of equations. Other bases in this space of rational functions can be better conditioned than (10.14), motivating us to look more closely at a rational analog of the Newton basis, i.e., at the following combination of (10.13) and (10.14):

$$\{\frac{1}{x-y_1},\ \frac{x-x_1}{(x-y_1)}\frac{}{(x-y_2)},\ \ldots,\ \frac{(x-x_1)\cdot\ldots\cdot(x-x_{n-1})}{(x-y_1)\cdot\ldots\cdot(x-y_{n-1})}\frac{}{(x-y_n)}\}.$$

It turns out that the results on Cauchy matrices in Sec. 8 and 9 readily suggest two immediate algorithms to compute a rational interpolant in the Newton-like form:

$$f(x)=\sum_{k=1}^{n} c_k\cdot\frac{(x-x_1)\cdot\ldots\cdot(x-x_{k-1})}{(x-y_1)\cdot\ldots\cdot(x-y_{k-1})}\frac{}{(x-y_k)}, \tag{10.15}$$

Indeed, it can be shown that the situation is totally analogous to the relation of (10.3) to polynomial divided differences, so computing the coefficients of a rational function of the form (10.15) satisfying $f(x_k)=f_k$ is equivalent to solving a linear system of equations

$$\begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = L^{-1}\cdot\begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix},$$

where L is a lower triangular factor of $C(x,y)=LU$. Now just recall that in Lemmas 9.1 and 8.2 we specified two different decompositions for the desired lower triangular factor, both of the form

$$L^{-1}=D\cdot L_{n-1}\cdot\ldots\cdot L_1. \tag{10.16}$$

As the reader already guessed, these two decompositions give rise to two schemes, *updating* and *downdating*, to compute the vector $\{c_k\}$ of *rational divided differences*. Specifically, the BP-type decomposition (9.5) [proven via the Neville elimination scheme] suggests a bidiagonal form for the L_k's in (10.16):

Factors for the updating rational divided differences

$$L_k=\left[\begin{array}{c|cccc} I_k & & & & \\ \hline & \frac{1}{x_{k+1}-x_1} & & & \\ & & \frac{1}{x_{k+2}-x_2} & & \\ & & & \ddots & \\ & & & & \frac{1}{x_n-x_{n-k}} \end{array}\right]\left[\begin{array}{c|cccc} I_{k-1} & & & & \\ \hline & 1 & 0 & & \\ & -(x_1-y_k) & (x_{k+1}-y_k) & & \\ & & \ddots & \ddots & \\ & & & -(x_{n-k}-y_k) & (x_n-y_k) \end{array}\right]. \tag{10.17}$$

Similarly, the decomposition (8.4) [obtained via the usual pivotal scheme of Gaussian elimination] suggests slightly different L_k's for (10.16):

Factors for the downdating rational divided differences

$$L_k = \left[\begin{array}{c|cccc} I_{k-1} & & & & \\ \hline & 1 & & & \\ & & \frac{1}{x_{k+1}-x_k} & & \\ & & & \ddots & \\ & & & & \frac{1}{x_n-x_k} \end{array}\right] \left[\begin{array}{c|cccc} I_{k-1} & & & & \\ \hline & 1 & & & \\ & -1 & 1 & & \\ & \vdots & & \ddots & \\ & -1 & & & 1 \end{array}\right] \left[\begin{array}{c|ccc} I_{k-1} & & & \\ \hline & (x_k - y_k) & & \\ & & \ddots & \\ & & & (x_n - x_k) \end{array}\right] \tag{10.18}$$

The computational complexity of the updating and downdating rational divided difference schemes is the same, but we next show that a fast implementation of partial pivoting [e.g., as a predictive partial pivoting of Sec. 8.5] again yields much smaller backward error bounds for the downdating scheme.

10.6 Backward stability of rational downdating divided differences

Essentially the same backward error bound (10.7) of polynomial divided differences can be obtained for their rational counterparts:

$$|f - L \cdot \widehat{c}| \le d_n u |L_1^{-1}| \cdot \ldots \cdot |L_{n-1}| \cdot |D^{-1}| \cdot |\widehat{c}|. \tag{10.19}$$

where $\widehat{c}$ denotes the vector of *computed* rational divided differences, L is a lower triangular factor of the Cauchy matrix $C(x,y) = LU$:

$$L^{-1} = D \cdot L_{n-1} \cdot \ldots \cdot L_1,$$

and d_n is a polynomial of small degree in n [whose exact form is not relevant for our analysis]. The error bound (10.19) is valid for the both updating and downdating rational divided differences, and it is quite pessimistic, because of the absolute values on the right-hand-side. Denoting by $\widehat{f} = L \cdot \widehat{c}$ the vector of the values at $\{x_k\}$ of the *actually computed* rational interpolant

$$f(x) = \sum_{k=1}^{n} \widehat{c}_k \cdot \frac{(x-x_1) \cdot \ldots \cdot (x - x_{k-1})}{(x-y_1) \cdot \ldots \cdot (x-y_{k-1})} \frac{}{(x-y_k)}$$

at $\{x_k\}$, we can use the same arguments as in Sec. 10.4 to motivate that the bound of the form

$$|f - \widehat{f}| \le d_n u |L| \cdot |\widehat{c}|, \tag{10.20}$$

would be much better than (10.19). In fact, the other arguments of Sec 9.4 carry over to the rational divided differences as well: for the updating scheme we are able to remove moduli in (10.19) and to deduce (10.20), but we have to assume

$$y_k < x_1 < x_2 < \ldots < x_n.$$

So, even if y_k's are all less than x_k's, the desired bound (10.20) was established for the updating scheme with only one particular ordering, i.e., only for one Newton-like representation of the rational interpolant $f(x)$. But unfortunately this particular ordering unfortunately does not guarantee a small $|L|$.

In contrast, the situation with downdating rational divided differences is favorable, and for any ordering we may wish we shall have the identity

$$|L| = |L_1^{-1}| \cdot \ldots \cdot |L_{n-1}^{-1}| \cdot |D^{-1}|,$$

always implying the desired bound (10.20). Of course, the first candidate to look at is the partial pivoting ordering, because it produces the unit lower triangular factor $\breve{L}$ [in the LU factorization of $C(x,y)$] with the entries less that unity. Our L and the unit $\breve{L}$ are related as

$$L = \breve{L} \cdot \text{ diag } \{\frac{1}{x_1 - y_1}, \ \frac{x_2 - x_1}{(x_2 - y_1)} \frac{}{(x_2 - y_2)}, \ \ldots, \ \frac{(x_n - x_1) \cdot \ldots \cdot (x_n - x_{n-1})}{(x_n - y_1) \cdot \ldots \cdot (x_n - y_{n-1})} \frac{}{(x_n - y_n)}\},$$

so that (10.20) implies a better backward stability:

$$|f_k - \widehat{f}_k| \le d_n u \cdot \sum_{k=1}^{n} |\widehat{c}_k| \cdot \frac{|(x_k - x_1) \cdot \ldots \cdot (x_k - x_{k-1})|}{|(x_k - y_1) \cdot \ldots \cdot (x_k - y_{k-1})} \frac{}{(x_k - y_k)|} + O(u^2).$$

for the downdating rational divided differences. To sum up, for the two rational divided difference schemes the conclusion is exactly the same as for the polynomial ones: the downdating scheme with partial pivoting can be preferable if the backward stability is concerned. Finally recall that we have a fast $O(n^2)$ predictive partial pivoting algorithm of Sec. 8.5 to use with the downdating rational divided difference scheme without slowing it down.

11 Concluding remarks

Analysis of the numerical behavior of array algorithms is one of the central themes in numerical linear algebra, perhaps the most important for the design of quality software for solving a wide variety of the associated applied problems. After about two decades of close attention, we now have a good understanding of the numerical properties of many standard algorithms for basic linear algebra problems, and we now have many stabilizing techniques, i.e., recipes to improve the accuracy. The most classical and important among such problems is, perhaps, the analysis of the performance of the Gaussian elimination procedure in the finite-precision-arithmetic. One of the early important results in this direction was the Wilkinson's error analysis leading to a recipe to use pivoting to numerically stabilize this classical algorithm.

In a different direction, reformulating a certain applied problem as a linear algebra problem, we quite often have to form a square array which is much larger than the actual amount of the input parameters. In other words, matrices appearing in applications are, as a matter of fact, structured, in the sense that they are defined by a small number of parameters. Different applications give rise to many different patterns of structure, including Toeplitz, Hankel, Vandermonde, Cauchy matrices, their confluent versions, Bezoutians, controllability, Schur-Cohn, Rouz-Hurwitz matrices, and many other patterns of structure. Exploiting the structure of a matrix to speedup the computation is another well-known topic in numerical linear algebra, but for the time being the analysis of accuracy of such structure-exploiting fast algorithms seems to be still behind the results obtained for their general purpose counterparts. As a result, much less stabilizing techniques have been proposed, and often a user has no other choice but to use slow structure-ignoring, but accurate, algorithms. Briefly, to sacrifice speed to achieve accuracy.

In this paper we reviewed recent progress made at the intersection of the above two areas. The point is that many well-known and new algorithms for important classes of applied problems admit matrix interpretations as a Gaussian elimination procedure, but efficiently implemented for a matrix with a certain pattern of structure. This association suggests to use pivoting, so successful for the usual Gaussian elimination procedure, to improve the accuracy of its fast structure-exploiting relatives. We described recently developed fast algorithms for factorization of Cauchy and Cauchy-like matrices, appearing in various tangential interpolation problems [we used an example of suboptimal rational matrix Nehari problem to show how to solve interpolation problems belonging to this class]; presented realistic algorithms to factorize general and J-unitary rational matrix functions, obtained various divided difference schemes [including polynomial and rational, updating and downdating]; developed Toeplitz-like and Toeplitz-like solvers through the transformation-to-Cauchy approach. In all these cases we suggest structure-exploiting stabilizing techniques, whose matrix interpretation is pivoting, efficiently implemented for a particular pattern of structure. We found, either numerically or through rounding error analysis, that such fast pivoting is crucial to computing not only fast, but also accurate, solutions. These results will contribute to the design of a comprehensive quality software package for solving structured matrix problems and applications.

12 Acknowledgement

The author is thankful to Gene Golub for valuable discussions and suggestions.

References

[A65] N.I.Akhiezer, *The classical moment problem and some related problems in analysis*, Hafner Publishing Co., New York, 1965.

[AD86] D.Alpay and H.Dym, *On applications of reproducing kernel spaces to the Schur algorithm and rational J-unitary factorizations*, in I.Schur Methods in Operator Theory and Signal Processing (I.Gohberg, ed.), OT18, Birkhäuser Verlag, Basel, 1986, 89 –160.

[AG88] D.Alpay and I.Gohberg, *Unitary rational matrix functions* in Topics in interpolation theory of rational matrix functions (I.Gohberg, ed.), OT33, Birkhäuser Verlag, Basel, 1988, 175 –222.

[AG90] G.Ammar and P.Gader, *New decompositions of the inverse of a Toeplitz matrix*, Signal processing, Scattering and Operator Theory, and Numerical Methods, Proc. Int. Symp. MTNS-89, vol. III, 421-428, Birkhauser, Boston, 1990.

[B37] J.P.M. Binet, *Observations sur des théoremes de Géométrie, énoncées page 160 de ce volume et page 222 du volume et page 222 du volume précédent*, Journ. (de Liouville) de Math, ii. (1837), 248-252.

[B71] J.R.Bunch, *Analysis of the diagonal pivoting method*, SIAM J. Num. Anal., **8** (1971), 656 -680.

[BBHS95] A.W.Bojanczyk, R.P.Brent, F.R. de Hoog and D.R.Sweet, *On the stability of the Bareiss and related Toeplitz factorization algorithms*, SIAM J. on Matrix Analysis Appl., **16**No.1 (1995).

[BKO95a] T.Boros, T.Kailath and V.Olshevsky. *The fast Björck-Pereyra-type algorithm for solving Cauchy linear equations*, submitted, 1995.

[BKO95b] T.Boros, T.Kailath and V.Olshevsky, *Predictive pivoting and backward stability of fast Cauchy solvers*, 1995, submitted.

[BGK79] H. Bart, I. Gohberg and M.A.Kaashoek, *Minimal factorization of matrix and operator functions*, OT1, Birkhäuser Verlag, Basel, 1979.

[BGKV80] H.Bart, I.Gohberg, M.A.Kaashoek and P.Van Dooren, *Factorizations of transfer functions*, SIAM J. Control and Optim., **18** (1980), 675-696.

[BGR90] J. Ball, I.Gohberg and L.Rodman, *Interpolation of rational matrix functions*, OT45, Birkhäuser Verlag, Basel, 1990.

[BK77] J.R.Bunch and L.Kaufman, *Some stable methods for calculating inertia and solving symmetric linear systems*, Math. Comp., **31**, 162 - 179.

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

[BP94] D.Bini and V.Pan, *Polynomial and Matrix Computations*, Volume 1, Birkhauser, Boston, 1994.

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

[C80] G.Cybenko, *The numerical stability of Levinson–Durbin algorithm for Toeplitz systems of equations*, SIAM J. Sci. Statist. Comput., **1** (1980), 303 – 319.

[C88] T.Chan, *An optimal circulant preconditioner for Toeplitz systems*, SIAM J. Sci. Stat. Comput. **9 (4)** (1988), 166 - 771.

[C89] J.Chun, *Fast array algorithms for structured matrices*, Ph.D.Thesis, 1989, Stanford University, Stanford, CA.

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

[CH92] T.Chan and P.Hansen, *A look-ahead Levinson algorithm for indefinite l Toeplitz systems*, SIAM J. on Matrix Anal. and Appl., 13(1992), 1079-1090.

[CKLA87] J.Chun, T.Kailath and H.Lev-Ari, *Fast parallel algorithms for QR and triangular factorizations*, J. Sci. Stat. Comput., **8**(1987), 899-913.

[CK91] J.Chun and T.Kailath, *Fast triangularization and orthogonalization of Hankel and Vandermonde matrices*, Linear Algebra Appl., **151** (1991), 199 – 228.

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

[D82] J.M.Delosme, *Fast algorithms for finite shift-rank processes*, Ph.D.Thesis, Stanford University, Stanford, CA, 1982.

[DGK81] Ph.Delsarte, Y.Genin and Y.Kamp, *On the role of the Nevanlinna-Pick problem in circuit and system theory*, Circuit Theory and Appl., **9** (1981), 177-187.

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

[F84] M.Fiedler, *Hankel and Loewner matrices*, Linear Algebra and Appl., **28**(1984), 75-95.

[FMKL79] B.Friedlander, M.Morf, T.Kailath and L.Ljung, *New inversion formulas for matrices classified in terms of their distance from Toeplitz matrices*, Linear Algebra and Appl., **27** (1979), 31-60.

[FZ93] R.Freund and H.Zha, *A look-ahead strategy for the solution of general Hankel systems*, Numerische Mathematik, **64**, (1993), 295-322.

[G95] Ming Gu, *Stable and efficient algorithms for structured systems of linear equations*, ti appear in SIAM J. on Matrix Analysis and Appl., 1997.

[GH94] M.Gutknecht and M.Hochbruck, *Look-ahead Levinson and Schur recurrences in the Pad'e table*, Electronic Transactions on Numerical Analysis, **2** (1994), 104-129.

[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.

[GK89] I.Gohberg and I.Koltracht, *Efficient algorithm for Toeplitz plus Hankel matrices*, Integral Equations and Operator Theory, **12** (1989), 136 – 142.

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

[GKKL87] I.Gohberg, T.Kailath, I.Koltracht and P.Lancaster, *Linear Complexity parallel algorithms for linear systems of equations with recursive structure*, Linear Algebra Appl., **88/89** (1987), 271–315.

[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.

[GLR82] I.Gohberg, P.Lancaster and L.Rodman, *Matrix Polynomials*, Academic Press, New York, 1982.

[GO92] I.Gohberg and V.Olshevsky, *Circulants, displacements and decompositions of matrices*, Integral Equations and Operator Theory, **15**, No. 5 (1992), 730 -743.

[GO93] I.Gohberg and V.Olshevsky, *Fast algorithm for matrix Nehari problem*, Proceedings of MTNS-93, Systems and Networks: Mathematical Theory and Applications, v.2, Invited and Contributed Papers,edited by U. Helmke, R. Mennicken and J. Sauers, Academy Verlag, 1994, p. 687-690.

[GO94a] I.Gohberg and V.Olshevsky, *Fast inversion of Chebyshev–Vandermonde matrices*, Numerische Mathematik, **67**, No. 1 (1994), 71 – 92.

[GO94b] 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.

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

[GO97] I.Gohberg and V.Olshevsky, *The fast generalized Parker-Traub algorithm for inversion of Vandermonde and related matrices*, J.of Complexity, 1997.

[GTVDV96] K.Gallivan, S.Thirumalai, P.Van Dooren, V.Vermaut, *High performance algorithms for Toeplitz and block Toeplitz matrices*, Linear Algebra Appl., **241-243**(1996), 343-388.

[GP92] M.Gasca, J.M.Pena, *Total positivity and Neville elimination*, Linear Algebra Appl. **165**(1992), 25-44.

[GVKDM83] Y.Genin, P. Van Dooren, T.Kailath, J.Delsome and M.Morf, *On Σ-lossless transfer functions and related questions*, Linear Algebra Appl., **50** (1983), 251 - 275.

[GVL96] G. Golub and C, Van Loan, *Matrix Computations*, second edition, John Hopkins U. P., Baltimore, 1996.

[H95] G. Heinig, *Inversion of generalized Cauchy matrices and other classes of structured matrices*, in Linear Algebra in Signal Processing, IMA volumes in Mathematics and its Applications, vol. **69** (1995), 95 - 114.

[HB97] G.Heinig and A.Bojanczyk, *Transformation techniques for Toeplitz and Toeplitz-plus-Hankel matrices, I. Transformations*, to appear in Linear Algebra Appl., 1997.

[HJR88] G.Heinig, P.Jankowski and K.Rost, *Fast inversion of Toeplitz-plus-Hankel matrices*, Numer.Math. **52** (1988), 665 – 682.

[HLM95] C.He, A.Laub and V.Mehrmann, *Placing plenty of poles is pretty preposterous*, preprint, 1995.

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

[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. Appl., **11**(1) (1990), 23–41.

[Hig95] N.J.Higham, *Stability of the diagonal pivoting method with partial pivoting*, Preprint, 1995.

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

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

[K80] W.Kahan, *Interval arithmetic options in the proposed IEEE floating point arithmetic standard*, in Interval Mathematics, 1980, (Karl L.E.Nickel, editor), Academic Press, New York, 1980, 99-128.

[K86] T.Kailath, *A theorem of I.Schur and its impact on modern signal processing*, in Operator Theory: Advances and Applications (I.Schur methods in Operator Theory and Signal Processing), **18**, 9-30, Birkhauser, 1986.

[K87] T.Kailath, *Signal processing applications of some moment problems*, Proc. of Symposia in Appl. Math., vol. **37**, 71-109. AMS annual meeting, short course reprinted in *Moments in mathematics*, ed. H.Landau, San Antonio, TX, January 1987.

[KL96] Misha E. Kilmer and Dianne P. O'Leary, *Pivoted Cauchy-Like Preconditioners for Regularized Solution of Ill-Posed Problems,* Computer Science Department Report CS-TR-3682, Institute for Advanced Computer Studies Report UMIACS-TR-96-63, University of Maryland, September 1996.

[K97] Misha E. Kilmer, *Cauchy-Like Preconditioners for 2-Dimensional Ill-Posed Problems,* Computer Science Department Report CS-TR-3776, University of Maryland, March 1997.

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

[KO95] T.Kailath and V.Olshevsky, *Displacement structure approach to Chebyshev-Vandermonde and related matrices*, Integral Equations and Operator Theory, 1995.

[KO96] T.Kailath and V.Olshevsky, *Displacement structure approach to discrete-trigonometric-transform based preconditioners of G.Strang and T.Chan types*, submitted, 1996; a short 15 pages version will appear in the Italian journal Calcolo, the issue devoted to the Proc. of the international workshop on Toeplitz matrices, Cortona, Italy, September 1996.

[KO96b] T.Kailath and V.Olshevsky, *Unitary Hessenberg matrices, and the generalized Parker-Forney-Traub and Björck-Pereyra algorithms for Szegö-Vandermonde matrices*, ISL technical report, 1996.

[KO97a] T.Kailath and V.Olshevsky, *Displacement structure approach to polynomial Vandermonde and related matrices*, preprint, 1994, to appear in Linear Algebra Appl., 1997.

[KO97b] T.Kailath and V.Olshevsky, *Bunch-Kaufman Pivoting for Partially Reconstructible Cauchy-like Matrices, with Applications to Toeplitz-like Linear Equations and to Boundary Rational Matrix Interpolation Problems*, to appear in Linear Algebra Appl., the issue devoted to Proc. of ILAS-95 symposium.

[KS92] T. Kailath and A.H.Sayed, *Fast algorithms for generalized displacement strcutures*, in Recent advances in mathematical theory of systems, control, networks and signal processing II, Proc. of the MTNS-91 (H.Kimura, S.Kodama, Eds.), Mita Press, Japan, 1992, 27 – 32.

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

[KS95b] T.Kailath and A.H.Sayed, *A look-ahead block Schur algorithm for Toeplitz-like matrices*, SIAM J. of Matrix Analysis Appl., **16** (1995), 388-414.

[KVB96] P.Kravanija and M. Van Barel, *A fast Hankel solver based on inversion formula for Loewner matrices*, preprint, 1996.

[L74] F. Lander, *The Bezoutian and the inversion of Hankel and Toeplitz matrices* (in Russian), Matem. Issled., Kishinev, **9** (No. 2), 69 - 87.

[LAPACK] E.Anderson, Z.Bai, C.Bishof, J.Demmel, J.Dongarra, J. Du Croz, A.Greenbaum, S.Hammarling, A.McKenney, S.Ostrouchov and D.Sorensen, *LAPACK User's Guide, Release 2.0*, SIAM, Philadelphia, PA, USA, second edition, 1995.

[LA83] H.Lev-Ari, *Nonstationary Lattice-Filter Modeling*, Ph.D. thesis, Stanford University, December 1983.

[LA96] H.Lev-Ari, *Displacement structure: two related perspectives*, pp. 233-242, in Communications, Computation, Control and Signal Processing, a tribute to Thomas Kailath (edited by A.Paulraj, V.Raychowdhury and C.Schaper), Kluwer Academic Publishers, Boston, 1996.

[LAK84] H.Lev-Ari and T.Kailath, *Lattice filter parameterization and modeling of nonstationary processes*, IEEE Trans on Information Theory, **30** (1984), 2-16.

[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.

[LAK92] H.Lev-Ari and T.Kailath *State-space approach to factorization of lossless transfer functions and structured matrices*, Linear Algebra Appl., **162 - 164** (1992), 273 - 295.

[M74] M.Morf, *Fast algorithms for Multivariable Systems*, Ph.D.Thesis, Stanford University, Stanford, CA, 1974.

[M80] M.Morf, *Doubling algorithms for Toeplitz and related equations*, Proc. IEEE Inter. Conf. on Acoustics, Speech and Signal Processing, Denver, 1980, 954-959.

[MS77] F.J. McWilliams and N.J.Sloane, *The Theory of Error-Correcting Codes*, North-Holland Publishing Company, Amsterdam, 1977.

[MB79] J.Maroulas and S.Barnett, Polynomials with respect to a general basis. I. Theory, *J. of Math. Analysis and Appl.*, **72** : 177 -194 (1979).

[N19] R.Nevanlinna, *Über beschränkte funktionen die in gegebenen punkten vorgeschreibene funktionswerte bewirkt werden*, Anal. Acad. Sci. Fenn., **13** (1919), 1 -71.

[O93] M. Ohsmann, *Fast cosine transform of Toeplitz matrices, algorithm and applications*, IEEE Transactions on Signal Processing, **41, No. 10** (1993), 3057-3061.

[O95] M.Ohsmann, *Fast transforms of Toeplitz matrices*, Linear Algebra Appl., **231** (1995), 181-192.

[OS86] A.Odlyzko and A.Schönage, *Fast algorithms for multiple evaluations of the Riemann zeta function*, Technical report, AT&T Bell Labs, Murray Hill, N.J., 1986.

[P60] Y.Potapov, *The multiplicative structure of J-contractive matrix functions*, Amer. Math.Translations, **15** (1960) 131-244.

[P90] V.Pan, *On computations with dense structured matrices*, Math. of Computation, **55**, No. 191 (1990), 179 – 190.

[R90a] L.Reichel, *A matrix problem with application to rapid solution of integral equations*, SIAM J. Sci. Stat. Comput, **11** (1990), 263-280.

[R90b] L.Reichel, *Newton interpolation at Leja points*, BIT, 30(1990), 23 – 41.

[Ro85] V. Rokhlin, *Rapid solution of integral equations of classic potential theory*, J. Comput Phys. **60** (1985), 187-207.

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

[RS94] *Reed-Solomon codes and their applications*, (S.Wicker and V.Bhargava, eds.), IEEE Press, New York, 1994.

[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.

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

[S92] A.H.Sayed, *Displacement structure in signal processing in engineering and mathematics*, Ph.D.Thesis, 1992, Stanford University, Stanford, CA.

[S96] M.Stewart, *Pivoting for Stability in the fast factorization of Cauchy-like matrices*, preprint, 1996.

[S97] S. Serra Cappizano, *A Korovkin-type theory, for finite Toeplitz operators via matrix algebras*, preprint, 1997.

[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.

[SKLAC94] A.Sayed,T.Kailath, H.Lev-Ari and T.Constantinescu, *Recursive solutions of rational interpolation problems via fast matrix factorization*, Integral Equations and Operator Theory, 1994.

[SLAK93] A.Sayed, H.Lev-Ari and T.Kailath, *Fast triangular factorization of the sum of quasi-Toeplitz and quasi-Hankel matrices*, Linear Algebra and Appl., **191** (1993), 77 – 106.

[T86] M.Trummer, *An efficient implementation of conformal mapping method using the Szegö kernel*, SIAM J. Numer. Anal., **23** (1986), 853-872.

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

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

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

[W65] J.H.Wilkinson, *The algebraic eigenvalue problem*, Clarendon Press, Oxfrord, 1965.

[W71] J.H.Wilkinson, *Modern error analysis*, SIAM Review, **14**(1971), 548-568.