# Matrix-vector Product for Confluent Cauchy-like Matrices with Application to Confluent Rational Interpolation

V.Olshevsky* and Amin Shokrollahi†

**Abstract**

Many important problems in pure and applied mathematics and engineering can be reduced to linear algebra on dense structured matrices. The *structure* of these dense matrices is understood in the sense that their $n^2$ entries can be "compressed" to a smaller number $O(n)$ of parameters. Operating directly on these parameters allows one to design efficient *fast algorithms* for these matrices. One of the most prominent matrix problems is that of multiplying a (structured) matrix with a vector. Many fundamental algorithms like convolution, Fast Fourier Transform, Fast Cosine/Sine Transform, and polynomial and rational multipoint evaluation and interpolation can be interpreted as superfast multiplication of a vector by structured matrices (like Toeplitz, DFT, Vandermonde, Cauchy). In this paper, we introduce a novel and fairly general class of structured matrices, which we call *confluent Cauchy-like matrices*, that contains all the above classes as a special case, and we will design new superfast algorithms for multiplication of matrices from our class with vectors. Our algorithm can be regarded as a generalization of all the above mentioned fast algorithms. Though this result is of interest by itself, its study was motivated by the following application. In a recent paper [18] the authors derived a superfast algorithm for solving the classical tangential Nevanlinna-Pick problem (rational matrix interpolation with norm constrains). Interpolation problems of Nevanlinna-Pick type appear in several important applications (see, e.g., [4]), and it is desirable to derive efficient algorithms for several similar problems. Though the method of [18] can be applied to compute solutions for certain other important interpolation problems (e.g., of Caratheodory-Fejer), the solution for the most general *confluent tangential* interpolation problems cannot be easily derived from [18]. Deriving new algorithms requires substantial further efforts; in particular, one needs to design a special fast algorithm to multiply a confluent Cauchy-like matrix by a vector. This is precisely what has been done in this paper.

## 1 Introduction

**1.1. Several examples of matrices with structure.** Structured matrices are encountered in a surprising variety of areas (e.g., signal and image processing, linear prediction, coding theory, oil exploration, to mention just a few), and algorithms (e.g., for Pade approximations, continuous fractions, classical algorithms of Euclid, Schur, Nevanlinna, Lanzcos, Levinson). There is an extensive literature on structured matrices, we mention here only several large surveys [13], [10], [3], [17] and two recent representative papers on rational interpolation [7, 18] and on list decoding of algebraic codes [19].

Many fundamental algorithms for polynomial and rational computations can be seen as algorithms for matrices with structure. Examples include Toeplitz $[t_{i-j}]$, Hankel $[h_{i+j-2}]$, Vandermonde $[x_i^{j-1}]$, and Cauchy matrices $[1/(x_i - y_j)]$.

Multiplication of these matrices with vectors often has an analytic interpretation. For instance, the problem of evaluation of a rational function $\sum_{k=1}^{n} \frac{a_k}{x-y_k}$ at the points $x_1, \ldots, x_n$ is clearly equivalent to computing the product of a Cauchy matrix $[1/(x_i - y_j)]$ by the vector $\begin{bmatrix} a_k \end{bmatrix}$. Table 1 lists some further interpretations for various matrices with structure. Its last column lists running times of the corresponding algorithms. Here, we denote by

$$M(n) = \begin{cases} n \log n & \text{if the field } K \text{ supports FFT's of length } n \\ n \log n \log\log n & \text{otherwise} \end{cases} \tag{1.1}$$

the running time of basic polynomial manipulation algorithms such as multiplication and division with remainder of polynomials of degree $< n$, cf. [1, Th. (2.8), Th. (2.13), Cor. (2.26)]. The running times in Table 1 are well

*Department of Mathematics, Georgia State University, Atlanta, GA 30303; volshevsky@gsu.edu; www.cs.gsu.edu/~matvro
†Fundamental Mathematics Research, Bell Laboratories, Murray Hill, NJ 07974, amin@research.bell-labs.com

| | | |
|---|---|---|
| Toeplitz matrices | convolution | $M(n)$ |
| Hankel matrices | convolution | $M(n)$ |
| Vandermonde matrices | multipoint polynomial evaluation | $M(n)\log n$ |
| DFT matrices (i.e., Vandermonde matrices with special nodes) | discrete Fourier transform | $M(n)$ |
| inverse Vandermonde matrices | polynomial interpolation | $M(n)\log n$ |
| Cauchy matrices | multipoint rational evaluation | $M(n)\log n$ |
| inverse Cauchy matrices | rational interpolation | $M(n)\log n$ |

Table 1: Algorithms

| | | |
|---|---|---|
| Toeplitz-like matrices $R$ | rank $(ZR - RZ)$ | $<< n$ |
| Hankel-like matrices $R$ | rank $(ZR - RZ^T)$ | $<< n$ |
| Vandermonde-like matrices $R$ | rank $(D_x^{-1}R - RZ^T)$ | $<< n$ |
| Cauchy-like matrices $R$ | rank $(D_xR - RD_y)$ | $<< n$ |

Table 2: Definitions of basic classes of structured matrices

known. We only mention that the problem of fast multiplying a Cauchy matrix by a vector is known in the numerical analysis community as the *Trummer problem*. It was posed by G. Golub and solved by Gerasoulis in [6]. We also mention the celebrated *fast multipole method* (FMM) of [20] that computes the approximate product of a Cauchy matrix by a vector (the FFM method is important in computaional potential theory).

In this extended abstract we continue the work of our colleagues and propose a new superfast algorithm to multiply by a vector a confuent Cauchy-like matrix (a far reaching generalization of a Cauchy matrix). To introduce confluent Cauchy-like matrices we need a concept of displacement recalled next.

**1.2. More general matrices, displacement structure.** Many applications give rise to the more general classes of structured matrices defined here. We start with a simple clarifying example. Let us define two auxiliary diagonal matrices $A_\zeta = \mathrm{diag}\{x_1, \ldots, x_n\}$, and $A_\pi = \mathrm{diag}\{y_1, \ldots, y_n\}$. It is immediate to see that for a Cauchy matrix $[1/(x_i - y_j)]$, the matrix $A_\zeta C - CA_\pi$ is the all-one matrix, and hence $\mathrm{rank}(A_\zeta C - CA_\pi) = 1$.

This observation is used to define a more general class of matrices which have appeared in many applications, e.g., [4, 18] and which have attracted much attention recently (see, e.g., [17] and the references therein). For these matrices the parameter

$$\alpha = \mathrm{rank}(A_\zeta C - CA_\pi), \tag{1.2}$$

is larger than 1, but it is still much less than the size of $C$. Such matrices are referred to as *Cauchy-like* matrices.

Similar observations can be made for all other patterns of structure discussed above. Simply for different kinds of structured matrices we need to use different auxiliary matrices $\{A_\pi, A_\zeta\}$. Table 2 contains definitions for basic classes of structured matrices. Here $Z^T = J(0)$ is one Jordan block with the eigenvalue 0, and $\{D_x, D_y\}$ are diagonal matrices.

Matrices in Table 2 are called *matrices with displacement structure*, the number $\alpha$ in (1.2) is called the *displacement rank*. The name *"displacement"* originates in signal processing literature [14, 5, 15] where Toeplitz and Hankel matrices are of special interest. For these matrices the auxiliray matrices $\{A_\zeta, A_\pi\}$ are shift (or displacement matrices) $Z$. The name *displacement structure* is now used also in connection to other classes of structured matrices in Table 2, though this terminology is not uniform (For example, in interpolation literature [2] they are called *null-pole coupling* matrices, in [3] they are referred to as matrices with low scaling rank).

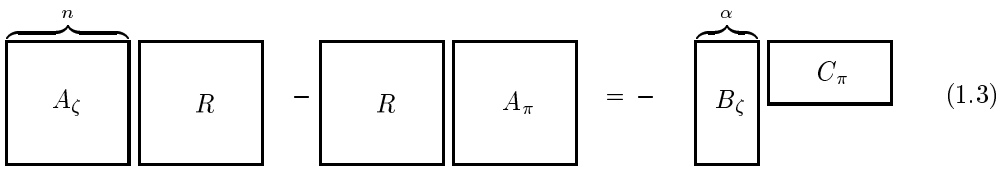
**1.3 A generator and superfast multiplication algorithms.** It is now well-understood [10], [3], [13], [17] that a useful approach to design fast matrix algorithms is in avoiding operations on $n^2$ entries, and in operating instead on what is called a *generator* of a structured matrix.

If the displacement rank (1.2) of a structured matrix $R$ is $\alpha$, one can factor (non-uniquely)

| | |
|---|---|
| Toeplitz-like | $\alpha M(n)$ |
| Hankel-like | $\alpha M(n)$ |
| Vandermonde-like | $\alpha M(n) \log n$ |
| Cauchy-like | $\alpha M(n) \log n$ |

| | |
|---|---|
| inverses of Toeplitz-like | $\alpha M(n) \log n$ |
| inverses of Hankel-like | $\alpha M(n) \log n$ |
| inverses of Vandermonde-like | $\alpha M(n) \log^2 n$ |
| inverses of Cauchy-like | $\alpha M(n) \log^2 n$ |

Table 3: Complexities of multiplication by a vector for matrices with displacement structure




(1.3)

where the two *rectangular* $\alpha \times n$ and $n \times \alpha$ matrices $\{C_\pi, B_\zeta\}$ are called a *generator* of $R$.

If the displacement equation (1.3) (with given $\{C_\pi, A_\pi, A_\zeta, B_\zeta\}$) has the unique solution $R$, then the entire information on $n^2$ entries of $R$ is conveniently compressed into only $2\alpha n$ entries of its generator $\{C_\pi, B_\zeta\}$.[1]

Avoiding operations on matrix entries and operating directly on a generator allows us to design fast and superfast algorithms. In particular, superfast algorithms to multiply by vectors matrices in Table 2 can be found in [9]. Table 3 lists the corresponding complexity bounds.

Notice that the problems of multiplying with the inverse is equivalent to solving the corresponding linear system of equations.

**1.4. First problem: matrix-vector product for confluent Cauchy-like matrices.** Notice that the auxiliary matrices $\{A_\zeta, A_\pi\}$ in Table 2 are all either shift or diagonal matrices, i.e., they all are special cases of the Jordan canonical form. Therefore it is natural to consider the more general class of structured matrices $R$ defined by using the displacement equation $\mathrm{rank}(A_\zeta R - R A_\pi) = B_\zeta C_\pi$ of the form (1.3), where

$$A_\zeta = \mathrm{diag}\{J_{m_1}(x_1) \oplus \ldots \oplus J_{m_s}(x_s)\}^T, \qquad A_\pi = \mathrm{diag}\{J_{k_1}(y_1) \oplus \ldots \oplus J_{k_t}(y_t)\}, \tag{1.4}$$

are general Jordan form matrices. We suggest to call such matrices $R$ *confluent Cauchy-like* matrices. Notice that the class of confluent Cauchy-like matrices is the most general class of structured matrices, containing all other classes listed in Table 2 as special cases. Therefore it is of interest to design a uniform superfast algorithm to multiply a confluent Cauchy-like matrix by a vector: such an algorithm would then contain all algorithms in Tables 1 and 3 (e.g., convolution, FFT, rational and polynomial multipoint evaluation and interpolation) as special cases.

Though such a generalized problem would be of interest by itself, we were motivated to study it by a rather general tangential interpolation problem formulated in the next section.

**1.5. Second problem: Confluent rational matrix interpolation problem.** Rational functions appear as transfer functions of linear time-invariant systems, and in the MIMO (Multi-Input Multi-Output) case the corresponding function is a rational *matrix* function (i.e., an $N \times M$ matrix $F(z)$ whose entries are rational functions $\frac{p_{ij}(z)}{q_{ij}(z)}$). It is often important to identify the transfer function $F(x)$ via certain interpolation conditions, one such rather general problem is formulated below. There is an extensive mathematical and electrical engineering literature on such problems, some of the pointers can be found in [18].

Tangential confluent rational interpolation problem.

**Given** : $r$ distinct points $\{z_k\}$ in the open right-half-plane $\Pi^+$, with their multiplicities $\{m_k\}$.
$r$ nonzero chains of $N \times 1$ vectors $\{x_{k,1}, \ldots, x_{k,m_k}\}$,
$r$ nonzero chains of $M \times 1$ vectors $\{y_{k,1}, \ldots, y_{k,m_k}\}$.

**Construct:** a rational $N \times M$ matrix function $F(x)$ such that

1 ). $F(z)$ is *analytic* inside the right half plane (i.e., all the poles are in the left half plane).

[1] Because of page limitations we do not discuss what happens if there are multiple solutions $R$ to the displacement equation, but all our complexity bounds fully apply to this more involved case. We only would like to mention an interesting connection of this case to the rational interpolation problem discussed in Sec. 1.5 below. Specifically, this degenerate case corresponds to the case of *boundary* interpolation, e.g., treated in [12] for the non-confluent case, and where just bounds of the form $O(n^2)$ were obtained.

2 ). $F(z)$ is *passive*, which by definition means that

$$\sup_{z \in \Pi^+ \cup i\mathbf{R}} \|F(z)\| \le 1. \tag{1.5}$$

3 ). $F(z)$ meets the tangential *confluent* interpolation conditions ($k = 1, 2, \ldots, r$):

$$\begin{bmatrix} x_{k1} & \ldots & x_{k,m_k} \end{bmatrix} \times \begin{bmatrix} F(z_k) & F'(z_k) & \cdots & \cdots & \frac{F^{(m_k-1)}(z_k)}{(m_k-1)!} \\ 0 & F(z_k) & \ddots & & \vdots \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & F'(z_k) \\ 0 & \cdots & \cdots & 0 & F(z_k) \end{bmatrix} = \begin{bmatrix} y_{k,1} & \cdots & y_{k,m_k} \end{bmatrix}. \tag{1.6}$$

The passivity condition (1.5) is naturally imposed by the conservation of energy (i.e., the energy of the output $y_{k,1}$ should not exceed the energy of input $x_{k,1}$). We next consider several clarifying special cases to show the fundamental nature of the above interpolation problem.

**Example 1.1 The tangential Nevanlinna-Pick problem.** *In the case of simple multiplicities $m_k = 1$ the interpolation condition (1.6) reduces to the usual tangential (i.e., x's and y's are vectors) interpolation condition*

$$x_k \cdot F(x_k) = y_k$$

*which in the case of the scalar $F(z)$ further reduces to the familiar interpolation condition of the form*

$$F(x_k) = \frac{y_k}{x_k}.$$

**Example 1.2 The Caratheodory-Fejer problem.** *Let the vectors $\{x_{k,j}\}$ are just the following scalars:*

$$\begin{bmatrix} x_{k,1} & \cdots & x_{k,m_k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}.$$

*Clearly, in this case (1.6) this is just a Hermite-type rational passive interpolation problem*

$$F(z_k) = y_{k,1}, \qquad F'(z_k) = y_{k,2}, \qquad \cdots \qquad \frac{F^{(m_k-1)}(z_k)}{(m_k-1)!} = y_{k,m_k}.$$

**Example 1.3 Linear matrix pencils.** *Let $F(z) = A - zI$, then $F'(z) = -I$, $F''(z) = 0$. If the condition (1.6) has the form*

$$\begin{bmatrix} u_{k1} & u_{k2} & u_{k3} \end{bmatrix} \cdot \begin{bmatrix} (A - z_k I) & -I & 0 \\ 0 & (A - z_k I) & -I \\ 0 & 0 & (A - z_k I) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

*then $u_{k1}$ is the (left)eigenvector: $u_{k1}(A - z_k I) = 0$, whereas $u_{k2}$ is the first generalized eigenvector: $u_{k2}(A - z_k I) = u_{k1}$, etc. In this simplest case the confluent interpolation problem reduces to recovering a matrix from its eigenvalues and eigenvectors.*

To sum up, the confluent tangential rational interpolation problem is a rather general inverse eigenvalue problem that captures several classical interpolation problems as its special cases.

**1.6. Main result.** In this extended abstract we describe a novel superfast algorithm to solve the confluent tangential rational interpolation problem. The running time of our algorithm is

$$Compl(n) = O\left( M(n) \log n \cdot \left[ 1 + \sum_{k=1}^{r} \frac{m_k}{n} \log \frac{n}{m_k} \right] \right) \tag{1.7}$$

where the multipliciteies $\{m_k\}$ are defined in Sec 1.4, $n = \sum m_k$, and $M(n)$ is defined in (1.1).

To understand the bound (1.7) it helps to consider two extreme cases. First, if all $m_k = 1$ (The Nevanlinna-Pick case: $n$ points with simple multiplicities) then $Compl(n) = M(n)\log^2(n)$ (or $n\log^3(n)$ if $K$ supports FFT's). Secondly, if $m_1 = n$ (The Caratheodory-Fejer case: one point with full multiplicity) then $Compl(n) = M(n)\log n$ (or $n\log^2 n$ if $K$ supports FFT).

The algorithm is based on a reduction of the above analytical problem to a structured linear algebra problem, namely to the problem of multiplication by a vector of a confluent Cauchy-like matrix. The corresponding running time is shown to be $O(M(n)\cdot[1+\sum_{k=1}^{r}\frac{m_k}{n}\log\frac{n}{m_k}+\sum_{k=1}^{s}\frac{t_k}{n}\log\frac{n}{t_k}])$, where $\{m_k\}$ are the sizes of the Jordan blocks of $A_\zeta$ and $\{t_k\}$ are the sizes of the Jordan blocks of $A_\pi$, see, e.g., (1.4).

**1.7. Derivation of the algorithm and the structure of the extended abstract.** The overall algorithm is derived in several steps, using quite different techniques. Interestingly, sometimes purely matrix methods are advantageous, and other times analytic arguments help.

1 ). First, the confluent tangential rational interpolation problem is reduced to the problem of multiplying by a vector a confluent Cauchy matrix $R$ whose generator is composed from the interpolation data. Namely, $R$ is defined via $A_\zeta R + RA_\zeta^* = B_\zeta J B_\zeta^*$,

$$A_\zeta = \begin{bmatrix} \boxed{J_{m_1}(z_1)^T} & & & \\ & \boxed{J_{m_2}(z_2)^T} & & \\ & & \ddots & \\ & & & \boxed{J_{m_r}(z_r)^T} \end{bmatrix}, \quad B_\zeta = \left[\begin{array}{cc} x_{11} & -y_{11} \\ & \vdots \\ x_{1,m_1} & -y_{1,m_1} \\ \hline & \vdots \\ x_{r1} & -y_{r1} \\ & \vdots \\ x_{r,m_n} & -y_{r,m_n} \end{array}\right],$$

$$J = \begin{bmatrix} I_M & 0 \\ 0 & -I_N \end{bmatrix},$$

2 ). Secondly, the problem for the confluent Cauchy-like $R$ above is reduced to the analogous problem for the *scaled confluent Cauchy* matrix (i.e., not *-like*) defined in section 2.

3 ). Then the problem for the scaled confluent Cauchy matrix is further reduced in Sec. 3 to the following two problems. One is to multiply a confluent Vandermonde matrix by a vector, and the second is to multiply the inverse of a confluent Vandermonde matrix by a vector. The solution for the second problem is available in the literature (Hermite-type polynomial interpolation).

4 ). The solution for the remaining problems (multiplication of a confluent Vandermonde matrix by a vector) is equivalent to the problem of multipoint Hermite-type evaluation, and the algorithm for it is described in the last section 4.

# 2 Scaled confluent Cauchy matrices

**2.1. Definition.** Suppose we have two sets of $n$ nodes each

$$\{\overbrace{\underbrace{x_1,\ldots,x_1}_{m_1},\ldots,\underbrace{x_s,\ldots,x_s}_{m_s}}^{n}\} \quad \text{and} \quad \{\overbrace{\underbrace{y_1,\ldots,y_1}_{k_1},\ldots,\underbrace{y_t,\ldots,y_t}_{k_t}}^{n}\},$$

so that $n = m_1 + m_2 + \ldots + m_s$, and $n = k_1 + k_2 + \ldots + k_t$. We do not assume that the $s + t$ nodes $x_1, \ldots, x_s, y_1, \ldots, y_t$ are pairwise distinct. For a scalar bivariate function

$$B(x,y) = \frac{b(x)-b(y)}{x-y}, \qquad \text{where} \qquad b(x) = (x-y_1)^{k_1}\cdot(x-y_2)^{k_2}\cdot\ldots\cdot(x-y_s)^{k_t} \tag{2.8}$$

we define the block matrix

$$C = [\ C_{i,j}\ ]_{1\le i\le s, 1\le j\le t}, \tag{2.9}$$

where the $m_i \times k_j$ block $B_{i,j}$ has the form

$$C_{i,j} = \begin{bmatrix} B(x_i,y_j) & \partial_y^1 B(x_i,y_j) & \partial_y^2 B(x_i,y_j) & \cdots & \partial_y^{k_j-1} B(x_i,y_j) \\ \partial_x^1 B(x_i,y_j) & \partial_x^1\partial_y^1 B(x_i,y_j) & \partial_x^1\partial_y^2 B(x_i,y_j) & \cdots & \partial_x^1\partial_y^{k_j-1} B(x_i,y_j) \\ \partial_x^2 B(x_i,y_j) & \partial_x^2\partial_y^1 B(x_i,y_j) & \partial_x^2\partial_y^2 B(x_i,y_j) & \cdots & \partial_x^2\partial_y^{k_j-1} B(x_i,y_j) \\ \vdots & \vdots & \vdots & & \vdots \\ \partial_x^{m_i-1} B(x_i,y_j) & \partial_x^{m_i-1}\partial_y^1 B(x_i,y_j) & \partial_x^{m_i-1}\partial_y^2 B(x_i,y_j) & \cdots & \partial_x^{m_i-1}\partial_y^{k_j-1} B(x_i,y_j) \end{bmatrix},$$

and we denote

$$\partial_z^k := \frac{1}{k!}\frac{\partial^k}{\partial z^k}. \tag{2.10}$$

Before giving a special name to $C$ let us notice that we do not assume that $\{x_1,\ldots,x_s,y_1,\ldots,y_n\}$ are pairwise distinct, so in the case $x_i = y_j$ the denominator in (2.8) is zero, hence we need to clarify what we mean by $B(x_i,x_i)$ and its partial derivatives. Using the following combinatorial identity

$$\partial_x^p\ \partial_y^q B(z,z) = \partial_z^{p+q+1} b(z).$$

we see that in the case $x_i = y_j$ the definition (2.8) implies that the block $C_{ij}$ is a Hankel matrix of the following form

$$\begin{bmatrix} 0 & \cdots & 0 & \times \\ \vdots & \iddots & \iddots & \vdots \\ 0 & \iddots & & \vdots \\ \times & & & \vdots \\ \vdots & & & \vdots \\ \times & \cdots & \cdots & \times \end{bmatrix} \quad \text{if } m_i > k_j \quad , \qquad \text{or} \qquad \begin{bmatrix} 0 & \cdots & \cdots & \cdots & 0 & \times \\ \vdots & & & \iddots & \iddots & \vdots \\ 0 & \cdots & 0 & \times & \cdots & \times \end{bmatrix} \quad \text{if } m_i < k_j \quad .$$

For example, if $m_i > k_j$ then $C_{i,j}$ has the following Hankel structure:

$$C_{ij} = \begin{bmatrix} 0 & \cdots & 0 & \partial^{k_j} b(x_i) \\ \vdots & \iddots & \iddots & \partial_x^{k_j+1} b(x_i) \\ 0 & \partial_x^{k_j} b(x_i) & \iddots & \vdots \\ \partial_x^{k_j} b(x_i) & \partial_x^{k_j+1} b(x_i) & \cdots & \partial_x^{2k_j-1} b(x_i) \\ \partial_x^{k_j+1} b(x_i) & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \partial_x^{m_i} b(x_i) & \partial_x^{m_i+1} b(x_i) & \cdots & \partial_x^{m_i+k_j-1} b(x_i) \end{bmatrix}.$$

We shall refer to $C$ in (2.9) as a *scaled confluent Cauchy matrix*, and to explain the name we consider next a simple example.

**Example 2.1** *Let $m_i = k_j = 1$ (so that $s = t = n$), and $\{x_1,\ldots,x_n,y_1,\ldots,y_n\}$ are pairwise distinct. Since $b(y_k) = 0$ by (2.8), we have:* $C = \operatorname{diag}\{b(x_1), b(x_2), \ldots, b(x_n)\} \cdot \begin{bmatrix} \frac{1}{x_1-y_1} & \cdots & \frac{1}{x_1-y_n} \\ \vdots & & \vdots \\ \frac{1}{x_n-y_1} & \cdots & \frac{1}{x_n-y_n} \end{bmatrix}$.

The above example presents the case of simple multiplicities; in the general situation of higher multiplicities $\{m_i\}$ and $\{k_j\}$ we call $C$ in (2.9) a *confluent* scaled Cauchy matrix.

**2.2. Analytic interpretation.** We have mentioned that multiplication of structured matrices by a vector often has an analytic interpretation, see, e.g., Table 1. Confluent Cauchy matrices are not an exception, and

multiplying $C$ in (2.9) by a vector $\left[\begin{array}{ccc|c|ccc} a_{0,1} & \dots & a_{k_1-1,1} & \dots & a_{0,t} & \dots & a_{k_t-1,t} \end{array}\right]^T$ is equivalent to multipoint evaluation of a rational function (with fixed poles $\{y_j\}$ of the orders $\{t_j\}$)

$$r(x) = \sum_{j=1}^{t} \sum_{l=0}^{k_j-1} a_{l,j} \partial_y^l B(x, y_j)$$

at points $\{x_i\}$ as well as of its derivatives up to the orders $\{m_i\}$.

**2.3. Displacement structure.** Let the auxiliary matrices $\{A_\zeta, A_\pi\}$ be defined as in (1.4), and define the class of *confluent Cauchy-like matrices* as those having a low displacement rank (1.2) with these $\{A_\zeta, A_\pi\}$. The following example justifies the latter name.

**Example 2.2** *Our aim here is to show that for the scaled confluent Cauchy matrix $R$ in (2.9) we have*

$$\operatorname{rank}(A_\zeta R - R A_\pi) = 1 \tag{2.11}$$

*where $\{A_\zeta, A_\pi\}$ are as in (1.4).*

*Indeed, applying $\partial_x^p \partial_y^q$ (recursively for $p, q = 0, 1, 2, \dots$) to the both sides of*

$$xB(x,y) - yB(x,y) = b(x) - b(y)$$

*we obtain*

$$x\partial_x^p \partial_y^q R(x,y) + \partial_x^{p-1} \partial_y^q R(x,y) - y\partial_x^p \partial_y^q R(x,y) - \partial_x^p \partial_y^{q-1} R(x,y) = \begin{cases} 0 & \text{if } p > 0 \text{ and } q > 0 \\ b^i(x) & \text{if } q = 0 \\ b^j(y) & \text{if } p = 0 \end{cases}$$

*Now arranging the latter equation in a matrix form we obtain*

$$J_{m_i}(x_i)^T C_{ij} - C_{ij} J_{k_j}(y_j) = \begin{bmatrix} b(x_i) & 0 & \cdots & 0 \\ \partial_x^1 b(x_i) & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ \partial_x^{m_i-1} b(x_i) & 0 & \cdots & 0 \end{bmatrix}$$

*Combining such eqautions for each block $R_{ij}$ together we finally obtain*

$$A_\zeta C - C A_\pi = \left[\begin{array}{cccc|c|cccc} b(x_1) & 0 & \cdots & 0 & & b(x_1) & 0 & \cdots & 0 \\ \partial_x^1 b(x_1) & 0 & \cdots & 0 & \cdots \quad \cdots & \partial_x^1 b(x_1) & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots & & \vdots & \vdots & & \vdots \\ \partial_x^{m_1-1} b(x_1) & 0 & \cdots & 0 & & \partial_x^{m_1-1} b(x_1) & 0 & \cdots & 0 \\ \hline & & \vdots & & & & & \vdots & \\ & & \vdots & & & & & \vdots & \\ \hline b(x_s) & 0 & \cdots & 0 & & b(x_s) & 0 & \cdots & 0 \\ \partial_x^1 b(x_s) & 0 & \cdots & 0 & \cdots \quad \cdots & \partial_x^1 b(x_s) & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots & & \vdots & \vdots & & \vdots \\ \partial_x^{m_s-1} b(x_s) & 0 & \cdots & 0 & & \partial_x^{m_s-1} b(x_s) & 0 & \cdots & 0 \end{array}\right]$$

*which yields (2.11).*

Thus, the displacement rank of scaled confluent Cauchy matrices is 1, so we coin the name *confluent Cauchy-like* with the matrices whose displacement rank, though higher than 1, still much smaller than the size $n$.

# 3 Factorization of confluent Cauchy matrices. Confluent Vandermonde matrices

We shall need a superfast algorithm to multiply a confluent Cauchy matrix by a vector, this algorithm will be based on the formula (3.12) derived next.

**Theorem 3.1** *Let $C$ be the confluent Cauchy-like matrix defined in (2.9), $b(x)$ is defined in (2.8) and $\partial_x$ is defined in (2.10). Then*

$$C = V_P(x)V_P(y)^{-1}diag\{B_1, B_2, \dots B_n\}, \tag{3.12}$$

*where*

$$V_P(x) = col\begin{pmatrix} V_P(x_1) \\ \vdots \\ V_P(x_k) \end{pmatrix} \quad with \quad V_P(x_i) = \begin{bmatrix} P_0(x_i) & P_1(x_i) & \cdots & P_{n-1}(x_i) \\ \partial_x P_0(x_i) & \partial_x P_1) & \cdots & \partial_x P_{n-1}(x_i) \\ \partial_x^2 P_0(x_i) & \partial_x^2 P_1(x_i) & \cdots & \partial_x^2 P_{n-1}(x_i) \\ \vdots & \vdots & & \vdots \\ \partial_x^{m_i-1} P_0(x_i) & \partial_x^{m_i-1} P_1(x_i) & \cdots & \partial_x^{m_i-1} P_{n-1}(x_i) \end{bmatrix},$$

*and*

$$B_i = \begin{bmatrix} 0 & \cdots & 0 & \partial_x^{m_1} b(x_i) \\ \vdots & \iddots & \iddots & \partial_x^{m_1} b(x_i) \\ 0 & \partial_x^{m_1} b(x_i) & \iddots & \vdots \\ \partial_x^{m_1} b(x_i) & \partial_x^{(m_1+1)} b(x_i) & \cdots & \partial_x^{(2m_1-1)} b(x_i) \end{bmatrix}, \tag{3.13}$$

The proof is based on the following formula which is of interest by itself:

$$V_P(y)^{-1} = \tilde{I} V_{\widehat{P}}^T(y) \cdot \text{ diag } (B_1^{-1}, B_2^{-1}, \dots, B_k^{-1}),$$

where $\tilde{I}$ stands for the flip permutation matrix, and $\{\widehat{P}\}$ denotes the *associated* system of polynomials defined in [11] (see also [16] for a connection with inversion of discrete transmission lines) .

We now turn to the computational aspects of the formula (3.12). Though the formula is given for an arbitrary polynomial system $\{P\}$ we shall use it here only for the usual power basis $P_k(x) = x^k$ (other choices, e.g., the Chebyshev polynomials are useful from the numerical point of view). The formula reduces the problem of multiplication of $C$ by a vector to the same problem for the three matrices on the right hand side of (3.12). Each of the three problems is treated next (it is convenient to discuss these three problem in a reversed order, i.e., first step 3, then step 2, and finally step 1).

**Step 3. Multipoint Hermite-type evaluation** The problem of multiplying $V(x)$ by a vector $[a_k]$ is equivalent to the problem of multipoint Hermite-type evaluation. Indeed, let us use the entries $[a_k]$ of to define the polynomial $a(x) = \sum_{k=0}^{n-1} a_k x^k$. Then it is clear that we need to evaluate the value of $a(x)$ at each $x_i$ as well as the values of its first $m_i$ (scaled) derivatives $\frac{a^{(s)}(x_i)}{s!}$ (notice that the rows of $V(x_i)$ are obtained by differentiation of their predecessors). The algorithm for this problem is offered in the next section.

**Step 2. Hermite interpolation** The problem of multiplying $V_P(x)^{-1}$ by a vector is the problem that is inverse to the one discussed in the step 3 above. The superfast algorithm for this problem can be found in [1]. The complexity is the same as in the step 3.

**Step 1. Convolution** The problem of multiplication of $diag\{B_1, B_2, \dots B_n\}$ by a vector is clearly reduced to the convolution, since all diagonal blocks (3.13) have a Hankel structure (constant along anti-diagonals). We also need to compute the entries of all these blocks $B_k$, i.e., for each point $x_k$ we need to compute the first $2m_i$ Taylor coefficients $\partial_x^s b(x_k)$ $(s = 0, 1, \dots 2m_i - 1)$. This is exactly the problem we treat in the step 3 above.

# 4 Multipoint Hermite-type evaluation

**4.1. The problem.** The problem we discuss here is that of computing the Taylor expansion of a univariate polynomial at different points. More precisely, for $f \in K[x]$ and $\xi \in K$ there are uniquely determined elements $f_{0,\xi}, f_{1,\xi}, \ldots$ such that

$$f = f_{0,\xi} + f_{1,\xi}(x-\xi) + f_{2,\xi}(x-\xi)^2 + \cdots.$$

The sequence $T(f,\xi,n) := (f_{0,\xi}, \ldots, f_{n-1,\xi})$ is called the sequence of Taylor coefficients of $f$ up to order $n$. Let $\xi_1, \ldots, \xi_t$ be elements in $K$ and $d_1, \ldots, d_t$ be positive integers, and suppose that $f$ is a univariate polynomial of degree $< n := d_1 + \cdots + d_t$. We want to develop a fast algorithm for computing the sequences

$$T(f,\xi,d_1), \ldots, T(f,\xi,d_t).$$

Our algorithm will use as a subroutine a fast multiple evaluation algorithm. Its running time involves the entropy of the sequence $(d_1, \ldots, d_t)$ defined by

$$\mathcal{H}(d_1, \ldots, d_t) := -\sum_{i=1}^{t} \frac{d_i}{n} \log \frac{d_i}{n},$$

where $n = \sum_i d_i$ and log is the logarithm to the base 2.

**4.2. Our Algorithm.** The algorithm we present for the problem stated above consists of several steps.

**Algorithm 4.1** *On input $\xi \in K$ and $d \geq 1$ the algorithm computes the polynomials $\Pi_{\ell,\xi} := (x-\xi)^{2^\ell}$ for $\ell = 0, \ldots, \lceil \log d \rceil$, as well as the polynomial $(x-\xi)^d$.*

(1) *Put $\Pi_{0,\xi} := (x-\xi)$.*

(2) *For $\ell := 1, \ldots, \lceil \log d_i \rceil$ put $\Pi_{\ell,\xi} := \Pi_{\ell-1,\xi}^2$.*

(3) *Let $d = 2^{\ell_1} + 2^{\ell_2} + \ldots + 2^{\ell_s}$ with $\ell_1 < \cdots < \ell_s$. Put $P := \Pi_{\ell_1,\xi}$.*

(4) *For $i = 2, \ldots, s$ set $P := P \cdot \Pi_{\ell_i,\xi}$. The final value of $P$ equals $(x-\xi)^d$.*

**Lemma 4.2** *Algorithm 4.1 correctly computes its output with $O(M(d))$ operations.*

The proof of this and the following assertions are omitted and will be included in the final version of the paper.

The next step of our presentation is the solution to our problem in case $t = 1$.

**Algorithm 4.3** *On input a positive integer $d$, and element $\xi \in K$, and a univariate polynomial $f \in K[x]$ of degree less than $d$, the algorithm computes $T(f,\xi,d)$. We assume that we have precomputed the polynomials $(x-\xi), (x-\xi)^2, \ldots, (x-\xi)^{2^\tau}$ where $\tau = \lceil \log d \rceil$.*

(1) *If $d = 1$, return $f$, else perform Step (2).*

(2) *Compute $f_0 := f \bmod (x-\xi)^{2^{\tau-1}}$ and $f_1 := (f - f_0)/(x-\xi)^{2^{\tau-1}}$ and run the algorithm recursively on $f_0$ and $f_1$. Output the concatenation of their outputs.*

**Lemma 4.4** *The above algorithm correcly computes its output in time $O(M(d))$.*

Now we are ready for our final algorithm.

**Algorithm 4.5** *On input positive integers $d_1, \ldots, d_t$, elements $\xi_1, \ldots, \xi_t \in K$, and a polynomial $f \in K[x]$ of degree less than $n := \sum_{i=1}^{t} d_i$, the algorithm computes the sequences $T(f,\xi_i,d_i)$, $i = 1, \ldots, t$.*

(1) *We use Algorithm 4.1 to the inputs $(\xi, d_i)$ for $i = 1, \ldots, t$. At this stage, we have in particular computed $(x-\xi_i)^{d_i}$ for $i = 1, \ldots, t$.*

(2) *Now we use the multiple evaluation algorithm given in the proof of [1, Th. (3.19)] to compute $f_i := f \bmod (x-\xi_i)^{d_i}$, $i = 1, \ldots, t$.*

(3) *Use Algorithm 4.3 on input $(f, \xi_i, d_i)$ to compute $T(f,\xi_i,d_i)$ for $i = 1, \ldots, t$.*

**Theorem 4.6** *The above algorithm correctly computes its output in time $O(M(n)\mathcal{H})$ where $\mathcal{H}$ is the entropy of the sequence $(d_1, \ldots, d_t)$.*

We remark that the algorithm can obviously be customized to run in parallel time $O(\mathcal{H})$ on $O(n)$ processors.

# References


[1] P. Bürgisser, M. Clausen and A. Shokrollahi, Algebraic Complexity Theory, series = Grundlehren der Mathematischen Wissenschaften, vol. 315, Springer Verlag, Heidelberg, 1996.

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

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

[4] 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.

[5] 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**, 31-60, 1979.

[6] A. Gerasoulis, A fast algorithm for the multiplication of generalized Hilbert matrices with vectors, *Math. of Computation*, **50** (No. 181), 1987, 179 – 188.

[7] 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, pp. 44-83, 1994.

[8] I. Gohberg and V. Olshevsky, *Fast inversion of Chebyshev-Vandermonde matrices*, *Numerische Mathematik*, **67 (1)** (1994), 71-92.

[9] I. Gohberg and V. Olshevsky, *Complexity of multiplication with vectors for structured matrices*, *Linear Algebra and Its Applications*, **202** (1994), 163-192.

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

[11] T. Kailath and V. Olshevsky, *Displacement Structure Approach to Polynomial Vandermonde and Related Matrices*, *Linear Algebra and Its Applications*, **261** (1997), 49-90.

[12] T. Kailath and V. Olshevsky, *Diagonal Pivoting for Partially Reconstructible Cauchy-like Matrices, With Applications to Toeplitz-like Linear Equations and to Boundary Rational Matrix Interpolation Problems*, *Linear Algebra and Its Applications*, **254** (1997), 251-302.

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

[14] M.Morf, "Fast algorithms for multivariable systems," Ph.D. thesis, Department of Electrical Engineering, Stanford University, 1974.

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

[16] V. Olshevsky, Eigenvector computation for almost unitary Hessenberg matrices and inversion of Szego-Vandermonde matrices via discrete transmission lines. Linear Algebra and Its Applications, (285)1-3 (1998) pp. 37-67

[17] V. Olshevsky, "Pivoting for structured matrices with applications," to appear in Linear Algebra and Its Applications, 2000;
available on http://www.cs.gsu.edu/~matvro

[18] V. Olshevsky and V. Pan, "A superfast state-space algorithm for tangential Nevanlinna-Pick interpolation problem," in *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pp. 192–201, 1998.

[19] V. Olshevsky and A. Shokrollahi, "A Displacement Approach to Efficient Decoding of Algebraic Codes," in *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, 235–244, 1999.

[20] V. Rokhlin, Rapid Solution of Integral Equations of Classical Potential Theory, *J. Comput. Physics*, **60**, 187–207, 1985.