Using structured linear algebra to compute Hermite-Padé approximants

bla blu
bli
blo@ble

ABSTRACT

bla bla

In all this paper, we let $\mathscr{M}$ be such that over any ring, polynomials of degree at most d can be multiplied in $\mathscr{M}(d)$ base ring operations; we also assume that the super-linearity assumptions of [8, Chapter 8]. Using the Cantor-Kaltofen algorithm [4], we can take $\mathscr{M}(d) \in O(d\log(d)\log\log(d))$. We let ω be a feasible exponent for linear algebra, in the sense that matrices of size n can be multiplied in $O(n^\omega)$ base ring operations over any ring; the best bound to date is $\omega < 2.38$ [6, 15]. The notation $O\tilde{}(\,)$ indicates that we omit polylogarithmic terms.

1. STRUCTURED LINEAR ALGEBRA

This section reviews background material on structured matrices. Initially developed by [12], the *displacement operator* approach represents a matrix A by its displacement $\phi(\mathsf{A})$, that is, the image of A under a *displacement operator* ϕ. Then, we say that A is structured with respect to ϕ if $\phi(\mathsf{A})$ has a small rank compared to its size; the rank of $\phi(\mathsf{A})$ is called the *displacement rank* of A with respect to ϕ.

The key idea of most algorithms for structured matrices is summarized by Pan's motto: compress, operate, decompress. Indeed, for A of size $m \times n$, if $\phi(\mathsf{A})$ has rank α, it can be represented using few elements through *generators*, that is, two matrices (G, H) in $\mathbb{K}^{m\times\alpha} \times \mathbb{K}^{n\times\alpha}$, with $\phi(\mathsf{A}) = \mathsf{G}\mathsf{H}^t$ (here, $\mathbb{K}$ is our base field). The main idea behind algorithms for structured matrices is to use such generators as a compact data structure, involving $\alpha(m+n)$ field elements instead of mn.

The most famous structures that support such an approach are the Toeplitz, Hankel, Vandermonde and Cauchy structures. While the case of Toeplitz-like matrices was the first one to be studied in detail, we will focus on Cauchy-like matrices, as they are arguably more convenient to work with.

For a sequence $\mathsf{u} = (u_1, \ldots, u_m)$ in $\mathbb{K}^m$, let $\mathsf{D}_u \in \mathbb{K}^{m\times m}$ be the diagonal matrix with diagonal entries $u_1, \ldots, u_m$. Then, given u as above and v in $\mathbb{K}^n$, we will consider the operator $\nabla_{\mathsf{u},\mathsf{v}} : \mathsf{A} \in \mathbb{K}^{m\times n} \mapsto \mathsf{D}_u\mathsf{A} - \mathsf{A}\mathsf{D}_v$; *Cauchy-like* matrices (with respect to the choice of u and v) are those matrices A for which $\nabla_{\mathsf{u},\mathsf{v}}(\mathsf{A})$ has small rank.

Let u, v be given and suppose that $u_i \neq v_j$ holds for all i, j. Then, the operator $\nabla_{\mathsf{u},\mathsf{v}}$ is invertible: given generators (G, H) of length α for A, with respect to the operator $\nabla_{\mathsf{u},\mathsf{v}}$, we can reconstruct A as

$$\mathsf{A} = \sum_{1\leq i\leq\alpha} \mathsf{D}_{\mathsf{g}_i}\mathsf{C}_{\mathsf{u},\mathsf{v}}\mathsf{D}_{\mathsf{h}_i}, \text{ with } \mathsf{C}_{\mathsf{u},\mathsf{v}} = \begin{bmatrix} \frac{1}{u_1-v_1} & \cdots & \frac{1}{u_1-v_n} \\ \vdots & & \vdots \\ \frac{1}{u_m-v_1} & \cdots & \frac{1}{u_m-v_n} \end{bmatrix}, \tag{1}$$

where g_i and h_i are the ith columns of respectively G and H. The matrix $\mathsf{C}_{\mathsf{u},\mathsf{v}}$ is known as a *Cauchy matrix*. Using fast polynomial evaluation and interpolation, we can multiply $\mathsf{C}_{\mathsf{u},\mathsf{v}}$ by a vector in time $O(\mathscr{M}(m)\log(m) + \mathscr{M}(n)\log(n))$, so that we can multiply A by a vector in quasi-linear time $O(\alpha(\mathscr{M}(m)\log(m) + \mathscr{M}(n)\log(n))) \subset O\tilde{}(\alpha(m+n))$. Remark that we can equivalently rewrite A as

$$\mathsf{A} = (\mathsf{G}\mathsf{H}^t) \odot \mathsf{C}_{\mathsf{u},\mathsf{v}}, \tag{2}$$

where $\odot$ denotes the entrywise product.

2. SOLVING CAUCHY-LIKE SYSTEMS

If a matrix $\mathsf{A} \in \mathbb{K}^{n\times n}$ is invertible, and is structured with respect to an operator $\nabla_{\mathsf{u},\mathsf{v}}$, it is known that its inverse is structured with respect to $\nabla_{v,u}$. Indeed, if $\mathsf{D}_u\mathsf{A} - \mathsf{A}\mathsf{D}_v = \mathsf{G}\mathsf{H}^t$, one easily deduces that $\mathsf{D}_v\mathsf{A}^{-1} - \mathsf{A}^{-1}\mathsf{D}_u = -(\mathsf{A}^{-1}\mathsf{G})(\mathsf{A}^{-t}\mathsf{H})^t$.

For the main question in this section, we will not assume that A is invertible. We will however suppose that is has *generic rank profile*, that is, that its leading principal minors of size up to rank(A) are invertible. As we will see below, this is an assumption which can be ensured by a random precondionning. We can then state the main question of this section; remark that the assumptions on the vectors u, v are slightly stronger than the one required for $\nabla_{\mathsf{u},\mathsf{v}}$ to be invertible.

PROBLEM A. *Consider vectors $\mathsf{u} = (u_1, \ldots, u_m)$ and $\mathsf{v} = (v_1, \ldots, v_n)$, with $u_i \neq v_j$, $u_i \neq u_{i'}$, $v_j \neq v_{j'}$ for all i, i', j, j'. Given $\nabla_{\mathsf{u},\mathsf{v}}$-generators of length α for a matrix A in $\mathbb{K}^{m\times n}$, with $\alpha \leq \min(m, n)$, do the following:*

- *if A does not have generic rank profile, raise an error,*
- *else, return $\nabla_{\mathsf{v}',\mathsf{u}'}$-generators for the inverse of the leading principal minor of A, with $\mathsf{v}' = (v_1, \ldots, v_r)$ and $\mathsf{u}' = (u_1, \ldots, u_r)$, where $r = \mathrm{rank}(\mathsf{A})$.*

There exist two classes of algorithms for handling such problems: iterative ones, of cost that grows like mn, and algorithms using divide-and-conquer techniques, of quasi-linear cost in $m+n$. In the following subsections, we review these algorithms and detail several improvements; we then present our implementation and its performance.

We will have to handle submatrices of A through their generators. The fact that D_u and D_v are diagonal matrices makes this easy (this is one of the aspects in which the Cauchy structure behaves more simply than the Toeplitz one). Suppose that (G,H) are generators for A, with respect to the operator $\nabla_{\mathsf{u},\mathsf{v}}$, and let $\mathsf{u}_I = (u_i)_{i\in I}$ and $\mathsf{v}_J = (v_j)_{j\in J}$ be subsequences of respectively u and v, corresponding to entries of indices I and J. Let $\mathsf{A}_{I,J}$ be the submatrix of A obtained by keeping rows and columns of indices respectively in I and J, and let $(\mathsf{G}_I,\mathsf{H}_J)$ be the matrices obtained from (G,H) by respectively keeping rows of G of indices in I, and rows of H of indices in J. Then, $(\mathsf{G}_I,\mathsf{H}_J)$ is a $\nabla_{\mathsf{u}_I,\mathsf{v}_J}$-generator for $\mathsf{A}_{I,J}$.

2.1 A faster quadratic algorithm

Iterative algorithms that solve a size n Toeplitz system in time $O(n^2)$ have been known for decades [16, 7, 20]; several extensions to more general structured matrices were later given, as for instance in [11]; see [19] for more references. For the particular form of our Problem A, we give here an algorithm inspired by [18, Algorithme 4]. In this reference, Mouilleron gives (without a proof) an algorithm that solves Problem A in time $O(\alpha n^2)$, in the case where $m=n$ and A is invertible (but without the rank profile assumption). He credits the origin of this algorithm to Kailath [13, §1.10], who dealt with symmetric matrices.

Even though we will discuss quasi-linear algorithms in the next section, we stress that having a fast quadratic-time algorithm is actually crucial in practice: as is the case for the HGCD, fast linear algebra algorithms, etc, the quasi-linear algorithm will fall back on the quadratic one for input sizes under a certain threshold, and the performance of the latter will be an important factor in the overall runtime. With this in mind, we introduce in this subsection an algorithm inspired by [18] that solves Problem A with the slightly better runtime $O(\alpha^{\omega-2}mn)$.

Let $(\mathsf{G},\mathsf{H}) \in \mathbb{K}^{m\times\alpha}\times\mathbb{K}^{n\times\alpha}$ be $\nabla_{\mathsf{u},\mathsf{v}}$-generators of a matrix A with respect to the operator $\nabla_{\mathsf{u},\mathsf{v}}$, with $\mathsf{u}=(u_1,\ldots,u_m)$ and $\mathsf{v}=(v_1,\ldots,v_n)$. Let further r be the rank of A. Our goal is to decide if A has generic rank profile, and if so, to return generators $(\mathsf{Y},\mathsf{Z}) \in \mathbb{K}^{r\times\alpha}\times\mathbb{K}^{r\times\alpha}$ of the inverse of the leading principal minor of A.

Let $p=\min(m,n)$. For i in $\{0,\ldots,p\}$, write A as

$$\mathsf{A} = \begin{bmatrix} \mathsf{A}_{0,0}^{(i)} & \mathsf{A}_{0,1}^{(i)} \\ \mathsf{A}_{1,0}^{(i)} & \mathsf{A}_{1,1}^{(i)} \end{bmatrix}.$$

If the principal minor $\mathsf{A}_{0,0}^{(i)}$ is invertible, we define as in [5]

$$\mathsf{S}^{(i)} = \begin{bmatrix} \mathsf{A}_{0,0}^{(i)\,-1} & -\mathsf{A}_{0,0}^{(i)\,-1}\mathsf{A}_{0,1}^{(i)} \\ \mathsf{A}_{1,0}^{(i)}\mathsf{A}_{0,0}^{(i)\,-1} & \mathsf{A}_{1,1}^{(i)} - \mathsf{A}_{1,0}^{(i)}\mathsf{A}_{0,0}^{(i)\,-1}\mathsf{A}_{0,1}^{(i)} \end{bmatrix}.$$

We next write the decompositions $\mathsf{G} = \begin{bmatrix} \mathsf{G}_0^{(i)} \\ \mathsf{G}_1^{(i)} \end{bmatrix}$, $\mathsf{H} = \begin{bmatrix} \mathsf{H}_0^{(i)} \\ \mathsf{H}_1^{(i)} \end{bmatrix}$, with $\mathsf{G}_0^{(i)}$ and $\mathsf{H}_0^{(i)}$ of size $i\times\alpha$ and we define

$$\mathsf{Y}^{(i)} = \begin{bmatrix} \mathsf{Y}_0^{(i)} \\ \mathsf{Y}_1^{(i)} \end{bmatrix}, \quad \mathsf{Z}^{(i)} = \begin{bmatrix} \mathsf{Z}_0^{(i)} \\ \mathsf{Z}_1^{(i)} \end{bmatrix}, \tag{3}$$

with

$$\mathsf{Y}_0^{(i)} = -\mathsf{A}_{0,0}^{(i)\,-1}\mathsf{G}_0^{(i)}, \quad \mathsf{Y}_1^{(i)} = -\mathsf{A}_{1,0}^{(i)}\mathsf{A}_{0,0}^{(i)\,-1}\mathsf{G}_0^{(i)} + \mathsf{G}_0^{(i)}, \tag{4}$$

$$\mathsf{Z}_0^{(i)} = \mathsf{A}_{0,0}^{(i)\,-t}\mathsf{H}_0^{(i)}, \quad \mathsf{Z}_1^{(i)} = -\mathsf{A}_{0,1}^{(i)\,t}\mathsf{A}_{0,0}^{(i)\,-t}\mathsf{H}_0^{(i)} + \mathsf{H}_1^{(i)}. \tag{5}$$

Given integers a, b, $\mathsf{u}_{a:b}$ denotes the sequence $(u_a,\ldots,u_b)$ (and similarly for $\mathsf{v}_{a:b}$); we then define $\mathsf{e}^{(i)} = (\mathsf{v}_{1:i}, \mathsf{u}_{[i+1:m)}$ and $\mathsf{f}^{(i)} = (\mathsf{u}_{1:i}, \mathsf{v}_{i+1:n})$. Then, a key result for the sequel is the following, which is [5, Proposition 1]; remark that the operator $\nabla_{\mathsf{e}^{(i)},\mathsf{f}^{(i)}}$ is invertible, in view of our assumption on u and v.

LEMMA 1. *$(\mathsf{Y}^{(i)},\mathsf{Z}^{(i)})$ are $\nabla_{\mathsf{e}^{(i)},\mathsf{f}^{(i)}}$-generators for $\mathsf{S}^{(i)}$.*

For $i=0$, we simply have $\mathsf{S}^{(0)}=\mathsf{A}$ and $(\mathsf{Y}^{(0)},\mathsf{Z}^{(0)})=(\mathsf{G},\mathsf{H})$. If A has generic rank profile, then the above lemma shows that for $i=r$, $(\mathsf{Y}_0^{(r)},\mathsf{Z}_0^{(r)})$ are $\nabla_{\mathsf{v}',\mathsf{u}'}$-generators for $\mathsf{A}_{0,0}^{(r)\,-1}$, for u',v' as in Problem A, so they solve our problem. These facts will allow us to devise an iterative algorithm that starts from $(\mathsf{Y}^{(0)},\mathsf{Z}^{(0)})$ and computes $(\mathsf{Y}^{(i_1)},\mathsf{Z}^{(i_1)})$, $(\mathsf{Y}^{(i_2)},\mathsf{Z}^{(i_2)})$, ... for some sequence of indices $0<i_1<i_2<\ldots$, until we finally reach $(\mathsf{Y}^{(r)},\mathsf{Z}^{(r)})$, on which we read off our output; if A does not have generic rank profile, we will detect it. The basis of the algorithm is Lemma 2 below, which uses the following notation.

Let i,j be non-negative integers with $0\le i+j\le p$, and $\mathsf{A}_{0,0}^{(i)}$ invertible. Write a block decomposition of $\mathsf{S}^{(i)}$ as

$$\mathsf{S}^{(i)} = \begin{bmatrix} \mathsf{S}_{0,0}^{(i,j)} & \mathsf{S}_{0,1}^{(i,j)} & \mathsf{S}_{0,2}^{(i,j)} \\ \mathsf{S}_{1,0}^{(i,j)} & \mathsf{S}_{1,1}^{(i,j)} & \mathsf{S}_{1,2}^{(i,j)} \\ \mathsf{S}_{2,0}^{(i,j)} & \mathsf{S}_{2,1}^{(i,j)} & \mathsf{S}_{2,2}^{(i,j)} \end{bmatrix}, \tag{6}$$

with $\mathsf{S}_{0,0}^{(i,j)} = \mathsf{A}_{0,0}^{(i)\,-1}$ of size $i\times i$ and $\mathsf{S}_{1,1}^{(i,j)}$ of size $j\times j$ (from this, the sizes of all blocks can be deduced). Similarly, we can refine our decompositions of $\mathsf{Y}^{(i)}$ and $\mathsf{Z}^{(i)}$ as

$$\mathsf{Y}^{(i)} = \begin{bmatrix} \mathsf{Y}_0^{(i)} \\ \mathsf{Y}_1^{(i,j)} \\ \mathsf{Y}_2^{(i,j)} \end{bmatrix}, \quad \mathsf{Z}^{(i)} = \begin{bmatrix} \mathsf{Z}_0^{(i)} \\ \mathsf{Z}_1^{(i,j)} \\ \mathsf{Z}_2^{(i,j)} \end{bmatrix},$$

with $\mathsf{Y}_1^{(i,j)}$ and $\mathsf{Z}_2^{(i,j)}$ of size $j\times\alpha$.

LEMMA 2. *$\mathsf{A}_{0,0}^{(i+j)}$ has generic rank profile if and only if $\mathsf{S}_{1,1}^{(i,j)}$ does, and $\operatorname{rank}(\mathsf{A}_{0,0}^{(i+j)}) = i + \operatorname{rank}(\mathsf{S}_{1,1}^{(i,j)})$. If these matrices are invertible, we have the equalities*

$$\mathsf{Y}^{(i+j)} = \begin{bmatrix} \mathsf{Y}_0^{(i)} - \mathsf{S}_{0,1}^{(i,j)}\mathsf{S}_{1,1}^{(i,j)\,-1}\mathsf{Y}_1^{(i,j)} \\ -\mathsf{S}_{1,1}^{(i,j)\,-1}\mathsf{Y}_1^{(i,j)} \\ \mathsf{Y}_2^{(i,j)} - \mathsf{S}_{2,1}^{(i,j)}\mathsf{S}_{1,1}^{(i,j)\,-1}\mathsf{Y}_1^{(i,j)} \end{bmatrix}$$

and

$$\mathsf{Z}^{(i+j)} = \begin{bmatrix} \mathsf{Z}_0^{(i)} - \mathsf{S}_{1,0}^{(i,j)\,t}\mathsf{S}_{1,1}^{(i,j)\,-t}\mathsf{Z}_1^{(i,j)} \\ \mathsf{S}_{1,1}^{(i,j)\,-t}\mathsf{Z}_1^{(i,j)} \\ \mathsf{Z}_2^{(i,j)} - \mathsf{S}_{1,2}^{(i,j)\,t}\mathsf{S}_{1,1}^{(i,j)\,-t}\mathsf{Z}_1^{(i,j)} \end{bmatrix}.$$

PROOF. By construction, $\mathsf{S}_{1,1}^{(i,j)}$ is the Schur complement of $\mathsf{A}_{0,0}^{(i)}$, seen as a submatrix of $\mathsf{A}_{0,0}^{(i+j)}$; this proves our first

claim. From the definition of $\mathsf{S}^{(i)}$ in (6), a direct calculation shows that S_{i+j} is given by

$$\mathsf{S}_{i+j} = \begin{bmatrix} \mathsf{J}_{0,0}^{(i,j)} & \mathsf{S}_{0,1}^{(i,j)}\mathsf{S}_{1,1}^{(i,j)-1} & \mathsf{J}_{0,2}^{(i,j)} \\ -\mathsf{S}_{1,1}^{(i,j)-1}\mathsf{S}_{1,0}^{(i,j)} & \mathsf{S}_{1,1}^{(i,j)-1} & -\mathsf{S}_{1,1}^{(i,j)-1}\mathsf{S}_{1,2}^{(i,j)} \\ \mathsf{J}_{2,0}^{(i,j)} & \mathsf{S}_{1,1}^{(i,j)-1}\mathsf{S}_{2,1}^{(i,j)} & \mathsf{J}_{2,0}^{(i,j)} \end{bmatrix},$$

with $\mathsf{J}_{a,b}^{(i,j)} = \mathsf{S}_{a,b}^{(i,j)} - \mathsf{S}_{a,1}^{(i,j)}\mathsf{S}_{1,1}^{(i,j)-1}\mathsf{S}_{1,b}^{(i,j)}$ for all a, b. In other words, up to permuting the first two blocks of rows, and the first two blocks of columns, $\mathsf{S}^{(i+j)}$ is derived from $\mathsf{S}^{(i)}$ in the same manner that $\mathsf{S}^{(i)}$ is derived from A. We may thus apply the rule that we used in Lemma 1 to derive $(\mathsf{Y}^{(i)}, \mathsf{Z}^{(i)})$ from (G, H) in order to derive $(\mathsf{Y}^{(i+j)}, \mathsf{Z}^{(i+j)})$ from $(\mathsf{Y}^{(i)}, \mathsf{Z}^{(i)})$. Taking into account the permutations we applied, we obtain the formulas given above. □

We can then describe the basic iterative step of our algorithm; we will use a step size $\beta \in \{1, \dots, \alpha\}$, given as a parameter. Suppose that we have found that $\mathsf{A}_{0,0}^{(i)}$ has rank i, with generic rank profile, and that we have computed $(\mathsf{Y}^{(i)}, \mathsf{Z}^{(i)})$, for some index i in $\{0, \dots, p\}$.

0. If $i = p$, return $(\mathsf{Y}_0^{(p)}, \mathsf{Z}_0^{(p)})$.

1. Let $j = \min(\beta, p - i) \geq 1$, and compute the matrices $\mathsf{S}_{1,0}^{(i,j)}$, $\mathsf{S}_{1,1}^{(i,j)}$, $\mathsf{S}_{1,2}^{(i,j)}$, $\mathsf{S}_{0,1}^{(i,j)}$ and $\mathsf{S}_{0,2}^{(i,j)}$ of (6). Since $(\mathsf{Y}^{(i)}, \mathsf{Z}^{(i)})$ are $\nabla_{\mathsf{e}^{(i)},\mathsf{f}^{(i)}}$-generators of $\mathsf{S}^{(i)}$, $(\mathsf{Y}_1^{(i,j)}, \mathsf{Z}^{(i)})$ and $(\mathsf{Y}^{(i)}, \mathsf{Z}_1^{(i,j)})$ are respectively $\nabla_{\mathsf{u}_{i+1:i+j},\mathsf{f}^{(i)}}$-generators and $\nabla_{\mathsf{e}^{(i)},\mathsf{v}_{i+1:i+j}}$-generators of its submatrices

$$\begin{bmatrix} \mathsf{S}_{1,0}^{(i,j)} & \mathsf{S}_{1,1}^{(i,j)} & \mathsf{S}_{1,2}^{(i,j)} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \mathsf{S}_{0,1}^{(i,j)} \\ \mathsf{S}_{1,1}^{(i,j)} \\ \mathsf{S}_{2,1}^{(i,j)} \end{bmatrix}.$$

Using block matrix multiplication in Eq. (2), since $j \leq \beta \leq \alpha$, we see that we can recover all these matrices in time $O(\beta^{\omega-2}\alpha(m+n))$.

2. If $\mathsf{S}_{1,1}^{(i,j)}$ does not have generic rank profile, raise an error. Else, compute its rank ρ. This costs $O(\beta^\omega)$.

3. If $\rho = j$, $\mathsf{A}_{0,0}^{(i+j)}$ has generic rank profile and rank $i + j$. Compute $(\mathsf{Y}^{(i+j)}, \mathsf{Z}^{(i+j)})$ by means of Lemma 2 and re-enter Step **0** with index $i+j$. To compute $\mathsf{Y}^{(i+j)}$, we first compute $\mathsf{S}_{1,1}^{(i,j)-1}\mathsf{Y}_1^{(i,j)}$ in time $O(\beta^{\omega-1}\alpha)$; then all other operations take time $O(\beta^{\omega-2}\alpha m)$. Similarly, computing $\mathsf{Z}^{(i+j)}$ takes time $O(\beta^{\omega-2}\alpha n)$.

4. If $\rho < j$, $\mathsf{A}_{0,0}^{(i+\rho)}$ has generic rank profile and rank $i + \rho$, but $\mathsf{A}_{0,0}^{(i+\rho+1)}$ still has rank $i + \rho$. Thus, either A has rank $i + \rho$, in which case we are done, or it has rank greater than $i + \rho$, in which case it does not have generic rank profile.

As a result, we go over Step **1** again, with index ρ instead of j, and we now deduce the generators $(\mathsf{Y}^{(i+\rho)}, \mathsf{Z}^{(i+\rho)})$, as we did in the previous item (the cost remains of the same order). This allows us to write down $\mathsf{S}^{(i+\rho)}$, on which we can read off the Schur complement $\mathsf{A}_{1,1}^{(i+\rho)} - \mathsf{A}_{1,0}^{(i+\rho)}\mathsf{A}_{0,0}^{(i+\rho)-1}\mathsf{A}_{0,1}^{(i+\rho)}$. If it vanishes, we return $(\mathsf{Y}_0^{(i+\rho)}, \mathsf{Z}_0^{(i+\rho)})$; else, we raise an error. Computing $\mathsf{S}^{(i+\rho)}$ from its $\nabla_{\mathsf{e}^{(i+\rho)},\mathsf{f}^{(i+\rho)}}$-generators $(\mathsf{Y}^{(i+\rho)}, \mathsf{Z}^{(i+\rho)})$ is done by means of (2), using block matrix multiplication, with a cost $O(\alpha^{\omega-2}mn)$.

The cost of Steps **0-3** is $O(\beta^{\omega-2}\alpha(m+n))$; we enter these steps $O(p/\beta)$ times, for a total cost of $O(\beta^{\omega-3}\alpha p(m+n))$, which is $O(\beta^{\omega-3}\alpha mn)$. This dominates the cost of Step **4** (which we enter once, at most), so that the whole running time is $O(\beta^{\omega-3}\alpha mn)$. The algorithm of [18] uses $\beta = 1$, for which the cost is $O(\alpha mn)$; choosing $\beta = \alpha$, we benefit from fast matrix multiplication, as the cost drops to $O(\alpha^{\omega-2}mn)$, as claimed previously.

2.2 The divide-and-conquer algorithm

We now review the divide-and-conquer approach to solving Problem A, and discuss a constant factor improvement for the case of certain Cauchy-like matrices.

Kaltofen [14] gave a divide-and-conquer algorithm that solves the analogue of Problem A for Toeplitz-like matrices, lifting assumptions needed in the original Morf and Bitmead-Anderson algorithm [17, 1]; a generalization of his algorithm to most usual structures is in [19]. We based our implementation on a further improvement due to Jeannerod and Mouilleron [10], which follows Cardinal's work [5]: in a nutshell, it allows us to bypass costly "compression" stages that were needed in Kaltofen's algorithm, by predicting the shape of the generators we have to compute.

Let $\mathsf{A}, \mathsf{G}, \mathsf{H}, p, r$ be as in the previous subsection, and for i in $\{0, \dots, p\}$, define $\mathsf{A}_{0,0}^{(i)}, \mathsf{A}_{0,1}^{(i)}, \mathsf{A}_{1,0}^{(i)}, \mathsf{A}_{1,1}^{(i)}, \dots$ as before. The algorithm will compute the rank r of A, together with specific generators for the inverse of the leading principal minor $\mathsf{A}_{r,0,0}$, namely $\mathsf{Y}_0^{(r)} = -\mathsf{A}_{0,0}^{(r)-1}\mathsf{G}_0^{(r)}$ and $\mathsf{Z}_0^{(r)} = \mathsf{A}_{0,0}^{(r)-t}\mathsf{H}_0^{(r)}$, as did the algorithm in the previous subsection.

We choose i in $\{0, \dots, p\}$, and we proceed as follows, essentially following [10] (with the minor difference that do not assume A invertible, and that we explicitly check if A satisfies the generic rank profile assumption).

0. If i is less than a certain fixed threshold, return the output of the algorithm of the previous section.

1. Call the algorithm recursively for the submatrix $\mathsf{A}_{0,0}^{(i)}$, given by its $\nabla_{\mathsf{u}_{1:i},\mathsf{v}_{1:i}}$-generators $(\mathsf{G}_0^{(i)}, \mathsf{H}_0^{(i)})$. This will raise an error if $\mathsf{A}_{0,0}^{(i)}$ does not have generic rank profile, so we assume we are not in that case, and that we know $\rho = \operatorname{rank}(\mathsf{A}_{0,0}^{(i)})$, $\mathsf{Y}_0^{(\rho)} = -\mathsf{A}_{0,0}^{(\rho)-1}\mathsf{G}_0^{(\rho)}$ and $\mathsf{Z}_0^{(\rho)} = \mathsf{A}_{0,0}^{(\rho)-t}\mathsf{H}_0^{(\rho)}$.

2. Compute $(\mathsf{Y}_1^{(\rho)}, \mathsf{Z}_1^{(\rho)})$ using (4) and (5). The dominant cost is that of computing the products $\mathsf{A}_{1,0}^{(\rho)}\mathsf{Y}_0^{(\rho)}$ and $\mathsf{A}_{0,1}^{(\rho)t}\mathsf{Z}_0^{(\rho)}$, given the $\nabla_{\mathsf{u}_{\rho+1:m},\mathsf{v}_{1:\rho}}$-generators $(\mathsf{G}_1^{(\rho)}, \mathsf{H}_0^{(\rho)})$ of $\mathsf{A}_{1,0}^{(\rho)}$, and the $\nabla_{\mathsf{u}_{1:\rho},\mathsf{v}_{\rho+1:n}}$-generators $(\mathsf{G}_0^{(\rho)}, \mathsf{H}_1^{(\rho)})$ of $\mathsf{A}_{0,1}^{(\rho)}$. We examine this cost below.

3. If $\rho < i$, test whether the Schur complement $\mathsf{A}_{1,1}^{(\rho)} - \mathsf{A}_{1,0}^{(\rho)}\mathsf{A}_{0,0}^{(\rho)-1}\mathsf{A}_{0,1}^{(\rho)}$ vanishes, or equivalently whether $\mathsf{Y}_1^{(\rho)}\,\mathsf{Z}_1^{(\rho)t} = 0$: if so, return $(\mathsf{Y}_0^{(\rho)}, \mathsf{Z}_0^{(\rho)})$; else, raise an error. This is done by finding a minimal set of independent rows in $\mathsf{Z}_1^{(\rho)}$ (this takes time $O(\alpha^{\omega-1}n)$) and multiplying their transposes by $\mathsf{Y}_1^{(\rho)}$ (there are at most α such rows, so this takes time $O(\alpha^{\omega-1}n)$ as well).

4. If $\rho = i$, call the algorithm recursively for the Schur complement $\mathsf{A}_{1,1}^{(i)} - \mathsf{A}_{1,0}^{(i)}\mathsf{A}_{0,0}^{(i)-1}\mathsf{A}_{0,1}^{(i)}$, given by its $\nabla_{\mathsf{u}_{i+1:m},\mathsf{v}_{i+1:n}}$-generators $(\mathsf{Y}_1^{(i)}, \mathsf{Z}_1^{(i)})$. This will raise an error if A does not have generic rank profile. Else, we obtain the rank σ of the Schur complement, as well as $\nabla_{\mathsf{v}_{i+1:i+\sigma},\mathsf{u}_{i+1:i+\sigma}}$-generators for its leading principal minor; with our notation, they are $-\mathsf{S}_{1,1}^{(i,\sigma)-1}\mathsf{Y}_1^{(i,\sigma)}$ and $\mathsf{S}_{1,1}^{(i,\sigma)-t}\mathsf{Z}_1^{(i,\sigma)}$.

5. We compute and return $(\mathsf{Y}_0^{(i+\sigma)}, \mathsf{Z}_0^{(i+\sigma)})$ using the formu-

las of Lemma 2. This is done by multiplying the above matrices by respectively $\mathsf{S}_{0,1}^{(i,\sigma)}$, given through its $\nabla_{\mathsf{u}_{i+1:i+\sigma},\mathsf{u}_{1:i}}$-generator $(\mathsf{Y}_0^{(i)}, \mathsf{Z}_1^{(i,\sigma)})$, and $\mathsf{S}_{1,0}^{(i,\sigma)t}$, given by its $\nabla_{\mathsf{v}_{1:i},\mathsf{v}_{i+1:i+\sigma}}$-generator $(\mathsf{Y}_1^{(i,\sigma)}, \mathsf{Z}_0^{(i)})$.

The bottleneck in this algorithm, at Steps **2** and **4**, is the multiplication of a Cauchy-like matrix of size roughly $m \times n$, given by generators of lengh α, by α vectors, either on the left or on the right. We saw that *one* such matrix-vector product can be done in time $O(\alpha\mathscr{M}(p')\log(p'))$, with $p' = \max(m,n)$, so computing α such products directly takes time $O(\alpha^2\mathscr{M}(p')\log(p'))$. We conclude by discussing our strategies to improve this cost.

A first improvement lies in the choice of u and v. In [19, Theorem 4.7.3], Pan shows that if the entries of both u and v are in *geometric progression*, one can reduce the cost of the matrix-vector multiplication by any of the matrices used above to $O(\alpha\mathscr{M}(p'))$. Indeed, the underlying algorithm involves α interpolations of polynomials at the points v, and α evaluations of polynomials of degree less than m at the points u, which can be done in respective times $O(\mathscr{M}(n))$ and $O(\mathscr{M}(m))$ when both u and v represent points in geometric progression.

Our implementation relies on a further refinement of this idea, that allows us to save constant factors in runtime: we require that u and v be geometric progressions with *the same ratio* τ. Then, the cauchy matrix $\mathsf{C}_{\mathsf{u},\mathsf{v}}$ of (1) has entries $1/(u_i - v_j) = 1/(u_1\tau^{i-1} - v_1\tau^{j-1})$, so it can be factored as

$$\mathsf{C}_{\mathsf{u},\mathsf{v}} = \mathsf{D}_\tau \begin{bmatrix} \frac{1}{u_1-v_1} & \frac{1}{u_1-v_1\tau} & \cdots & \frac{1}{u_1-v_1\tau^{n-1}} \\ \frac{1}{u_1-v_1\tau^{-1}} & \frac{1}{u_1-v_1} & \cdots & \frac{1}{u_1-v_1\tau^{n-2}} \\ \vdots & & \vdots & \\ \frac{1}{u_1-v_1\tau^{1-m}} & \frac{1}{u_1-v_1\tau^{2-m}} & \cdots & \frac{1}{u_1-v_1\tau^{n-m}} \end{bmatrix},$$

where D_τ is diagonal with entries $(1, \tau, \tau^2, \ldots, \tau^{m-1})$, and where the left-hand matrix is Toeplitz. In the reconstruction formula (1), the diagonal matrix D_τ commutes with all matrices D_{g_i}, so we can take it out of the sum. Hence, we replaced α evaluations / interpolations at geometric progressions by α product by Toeplitz matrices, each of which can be done in a single polynomial multiplication. The cost for a matrix-vector product by A remains $O(\alpha\mathscr{M}(p'))$, but the constant in the big-O is lower: for $m = n$, using middle product techniques [9, 2, 3], the cost goes down from $3\alpha\mathscr{M}(p') + O(\alpha p')$ to $\alpha\mathscr{M}(p') + O(\alpha p')$. The improvement applies for the products at both Steps **2** and **4**.

If u and v are chosen as geometric progressions (with the same ratio) at the top-level, this will remain the case for all recursive calls. Choosing $i = \lceil p/2 \rceil$ balances the costs of the two recursive calls; as a result, the overall runtime of the algorithm is $O(\alpha^2\mathscr{M}(p')\log(p))$.

The second improvement reduces the cost for multiplication to $O(\alpha^{\omega-1}\mathscr{M}(p')\log(p'))$.

3. HERMITE-PADÉ APPROXIMANTS

PROBLEM B. *Let* $\boldsymbol{F} = (f_1, \ldots, f_s)$ *be in* $\mathbb{K}[x]$,

PROBLEM C. *Let* f *be in* $\mathbb{K}[x]$.

3.1 Reduction to Cauchy systems

4. SOLVING STRUCTURED SYSTEMS

Given a prime p, an invertible structured matrix $\mathsf{A} \in (\mathbb{Z}/p^t)^{n\times m}$, and a vector $b \in (\mathbb{Z}/p^t)^n$, we review three algorithms that solve the equation $\mathsf{A}x = b \pmod{p^t}$ where t is a power of 2 by lifting.

4.1 The divide and conquer algorithm

DAC(A, b, p, t):

0. If $t = 1$, return $\mathsf{A}^{-1} * b \pmod{p}$

1. Compute $x_0 := \text{DAC}(\mathsf{A}, b, p, \frac{t}{2})$

2. Compute $r_0 := \mathsf{A} * x_0 - b \pmod{p^t}$ and $r_1 := r_0 \,/\, p^{\frac{t}{2}}$

3. Compute $x_1 := \text{DAC}(\mathsf{A}, r_1, p, \frac{t}{2})$

4. Return $x_0 - p^{\frac{t}{2}} * x_1 \pmod{p^t}$

4.2 Dixon's algorithm

Dixon(A, b, p, t):

0. Compute $\mathsf{M} := \mathsf{A}^{-1} \pmod{p}$

1. Compute $x_0 := \mathsf{M} * b \pmod{p}$

2. for $i \in [1, t-1]$:

- Compute $b := (b - \mathsf{A} * x_{i-1}) \,/\, p$
- Compute $x_i := \mathsf{M} * b \pmod{p}$

3. Return $\sum_{i=0}^{t-1} p^i * x_i$

4.3 Newton iteration

Let G and H be the generators of A.

Newton(G, H, p, t):

0.

5. SOLVING BLOCK TOEPLITZ SYSTEMS

Let $x \in \mathbb{Q}^m$, $\alpha = (\alpha_1, \cdots, \alpha_b) \in \mathbb{N}^b$, and let $\mathsf{M} \in \mathbb{Q}^{n\times m}$ be a block Toeplitz matrix with b blocks such that for $i \in [1, b]$, the i^{th} block has $n \times \alpha_i$ entries. We will outline an algorithm that finds a solution to $Mx = 0$.

5.1 Systems with one dimension kernel space

First, we look at a specific case where $m = \text{rank}(\mathsf{M}) - 1$.

0. Choose a prime p and compute $\mathsf{A} := \mathsf{XMY} \pmod{p}$ for choices of X and Y mentioned above such that A is a Cauchy matrix and has generic rank profile. Let b be the last column of A.

1. Let $d := \text{rank}(\mathsf{M})$, A_d be the $d \times d$ top-left submatrix of A, and b_d be a vector of the first d entries of b. By assumption, A_d is invertible.

2. Set $t := 1$. Compute $x := \mathsf{A}_d^{-1} * b_d \pmod{p}$ and $y := Y * (x_1, \cdots, x_d, -1) \pmod{p}$. Divide every entry of y by the first non-zero entry of y.

3. While we cannot apply rational reconstruction to the entries of y or after applying rational reconstruction to y, $M * y \neq 0 \pmod{p'}$ for some prime p':

- Set $t := t * 2$ and recompute $\mathsf{M}, \mathsf{X}, \mathsf{Y}, \mathsf{A}, \mathsf{A}_d, b_d \pmod{p^t}$
- Compute $r := A_d * x - b_d \pmod{p^t}$ and $r_1 := r \,/\, p^{\frac{t}{2}}$
- Call one of the three algorithms in section 4 to find x' such that $\mathsf{A}_d * x' = r_1 \pmod{p^{\frac{t}{2}}}$

- Set $x := x - p^{\frac{t}{2}} * x'$, $y := Y * (x_1, \cdots, x_d, -1) \pmod{p^t}$, and divide every entry of y by the first non-zero entry of y.

4. Apply rational reconstruction to every entry of y and return the result.

5.2 General block Toeplitz systems

For a general block Toeplitz system, we want to preprocess M to produce a matrix M' such that the $\text{rank}(\mathsf{M}') = m - 1$. We do this by creating a block Toeplitz matrix M'' of size $n' \times m$, where $n' = m - 1 - \text{rank}(\mathsf{M})$. Every block of M'' is lower triangluar and the i^{th} block

5.3 Solution to Hermite-Padé approximant

If M is algebraic, that is for $i \in [1, b]$ and $f \in \mathbb{Q}[x]$, the i^{th} block of M is composed of coefficients of f^{i-1}, then x is a Hermite-Padé approximant of f and type α.

6. REFERENCES

[1] R. R. Bitmead and B. D. O. Anderson. Asymptotically fast solution of Toeplitz and related systems of linear equations. *Linear Algebra Appl.*, 34:103–116, 1980.

[2] A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In *ISSAC'03*, pages 37–44. ACM, 2003.

[3] A. Bostan and É. Schost. Polynomial evaluation and interpolation on special sets of points. *J. Complexity*, 21(4):420–446, 2005.

[4] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991.

[5] J.-P. Cardinal. On a property of Cauchy-like matrices. *C. R. Acad. Sci. Paris Série I*, 388:1089–1093, 1999.

[6] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, March 1990.

[7] J. Durbin. The fitting of time series models. *Rev. Inst. Int. Stat.*, 28:233–243, 1960.

[8] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, Cambridge, third edition, 2013.

[9] G. Hanrot, M. Quercia, and P. Zimmermann. The Middle Product Algorithm, I. *Appl. Algebra Engrg. Comm. Comp.*, 14(6):415–438, 2004.

[10] C.-P. Jeannerod and C. Mouilleron. Computing specified generators of structured matrix inverses. In *ISSAC'10*, pages 281–288, 2010.

[11] T. Kailath, I. Gohberg, and V. Olshevsky. Fast Gaussian elimination with partial pivoting for matrices with displacement structure. *Math. Comp.*, 64(212):1557–1576, 1995.

[12] T. Kailath, S. Y. Kung, and M. Morf. Displacement ranks of matrices and linear equations. *J. Math. Anal. Appl.*, 68(2):395–407, 1979.

[13] T. Kailath and A. H. Sayed. *Fast Reliable Algorithms for Matrices with Structure*. SIAM, 1999.

[14] E. Kaltofen. Asymptotically fast solution of Toeplitz-like singular linear systems. In *ISSAC'94*, pages 297–304. ACM, 1994.

[15] F. Le Gall. Powers of tensors and fast matrix multiplication. In *ISSAC'14*, pages 296–303. ACM, 2014.

[16] N. Levinson. The Wiener RMS error criterion in filter design and prediction. *J. Math. Phys.*, 25:261–278, 1947.

[17] M. Morf. Doubling algorithms for toeplitz and related equations. In *IEEE Conference on Acoustics, Speech, and Signal Processing*, pages 954–959, 1980.

[18] C. Mouilleron. Algorithmes rapides pour la résolution de problèmes algébriques structurés. Master's thesis, ENS Lyon, 2008.

[19] V. Y. Pan. *Structured Matrices and Polynomials*. Birkhäuser Boston Inc., 2001.

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