Algorithms for Structured Linear Systems Solving and Their Implementation

Seung Gyu Hyun
Cheriton School of Computer Science
University of Waterloo
sghyun@edu.uwaterloo.ca

Romain Lebreton
LIRMM
Université de Montpellier
romain.lebreton@lirmm.fr

Éric Schost
Cheriton School of Computer Science
University of Waterloo
eschost@uwaterloo.ca

ABSTRACT

There exists a vast literature dedicated to algorithms for structured matrices, but relatively few descriptions of actual implementations and their practical performance in symbolic computation. In this paper, we consider the problem of solving Cauchy-like systems, and its application to mosaic Toeplitz systems, in two contexts: first in the unit cost model (which is a good model for computations over finite fields), then over $\mathbb{Q}$. We introduce new variants of previous algorithms and describe an implementation of these techniques and its practical behavior. We pay a special attention to particular cases such as the computation of algebraic approximants.

1 INTRODUCTION

Given input polynomials $(t_0, \ldots, t_{s-1})$ over a field $\mathbb{K}$, together with integers $(n_0, \ldots, n_{s-1})$ and σ, the *Hermite-Padé* approximation problem asks to compute polynomials $(p_0, \ldots, p_{s-1})$, not all zero, such that $\deg(p_i) < n_i$ holds for all i, and such that we have $p_0 t_0 + \cdots + p_{s-1} t_{s-1} = O(x^\sigma)$. There exist numerous applications to this type of question, very important particular cases being *algebraic approximants* (with $t_i = f^i$, for some given f) or *differential approximants* (with $t_i = d^i f / dx^i$, for some given f); see for instance [6, Chapitre 7].

Expressed in the canonical monomial bases, the matrix of a Hermite-Padé problem has size $\sigma \times (n_0 + \cdots + n_{s-1})$, and consists of s lower triangular Toeplitz blocks. More generally, our goal in this paper is to compute efficiently elements in the kernel of *mosaic Toeplitz* matrices [26]. An $m \times n$ Toeplitz matrix $\mathsf{T} = (t_{i-j})_{1 \le i \le m, 1 \le j \le n}$ can be succinctly represented by the polynomial $P_\mathsf{T} = t_{-n+1} + t_{-n+2}x + \cdots + t_{m-1}x^{m+n-2}$; multiplication of T by a vector $\mathsf{b} = [b_0 \ \cdots \ b_{n-1}]^t$ amounts to computing $P_\mathsf{T} P_\mathsf{b} \bmod x^{m+n-1}$ and keeping the coefficients of degrees $n-1, \ldots, m+n-2$. More generally, a mosaic Toeplitz $\mathsf{T} = (\mathsf{T}_{i,j})_{1 \le i \le p, 1 \le j \le q}$ of size $m \times n$ with a $p \times q$ block structure is made of pq blocks $\mathsf{T}_{i,j}$ being Toeplitz, and so can be described by a sequence of pq polynomials $\mathscr{P} = (P_{\mathsf{T}_{i,j}})_{1 \le i \le p, 1 \le j \le q}$. Then, our main problem is as follows.

Problem A. *Given $\mathscr{P}$ and integers I, J as above, defining a mosaic Toeplitz matrix T, find a non-zero vector in the kernel of T.*

We consider two situations, first over an arbitrary field $\mathbb{K}$, counting all operations in $\mathbb{K}$ at unit cost, then over $\mathbb{Q}$, taking bit-size into account. In all this paper, we closely follow the existing formalism of *structured matrix computations* developed in previous work by Morf [36], Bitmead-Anderson [3], Kailath and co-authors [30, 31], Pan [40, 41], Kaltofen [32], Cardinal [15], etc.

We complement the existing literature as follows. First, we define a class of Cauchy-like matrices (see definitions below) for which the matrix-vector product is faster by a constant factor than in previous designs. Next, we show how Pan's technique of "multiplicative transformation" of operators results in matrices that have generic rank profile (with high probability), so that further regularization is not needed in general. We then describe an improved iterative algorithm (of quadratic complexity with respect to the matrix size), that makes use of fast matrix multiplication; finally, for matrices defined over $\mathbb{Q}$, we introduce a divide-and-conquer algorithm as an alternative to Newton iteration, and we show how it can be improved in the case of algebraic approximation.

Another contribution of this paper is a discussion of the design and practical performance of a C++ implementation of these algorithms. To our knowledge, only a few papers address these methods from the practical viewpoint. An early reference is [47], which concludes that the divide-and-conquer (MBA) algorithm for solving Toeplitz matrices in quasi-linear time would require matrices of size 10^6 to break even with quadratic-time algorithms; a more recent article [27] estimates the crossover point to be around 8000.

Our experiments first consider computations over finite fields. We assess the practical impact of fast matrix multiplication for structured matrix algorithms (such as in the new algorithm mentioned above, or those in [8, 9]), and we estimate for what matrix size quasi-linear algorithms become effective (with much lower crossover points than above). In the context of computations over $\mathbb{Q}$, we show that our divide-and-conquer algorithm outperforms Newton iteration consistently, and we demonstrate significant speed-ups for the case of algebraic approximants.

We start in Section 2 with a review of basic results on structured matrices, including a discussion of transformation of operators and regularization. Section 3 describes algorithms applicable in a unit cost model, with in particular a new algorithm that uses fast matrix multiplication, and a discussion of the practical performance of these algorithms. Finally, Section 4 presents lifting algorithms for structured matrices, and the corresponding implementation.

2 BASIC RESULTS

Overview. Developed in [30], the *displacement operator* approach associates to a matrix A its displacement $\nabla(\mathsf{A})$, that is, the image of A under a *displacement operator* ∇. Then, we say that A is structured with respect to ∇ if $\nabla(\mathsf{A})$ has a small rank compared to its size;

ISSAC'17, July 25–28, 2017, Kaiserslautern, Germany

978-1-4503-5064-8/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3087604.3087659

the rank of $\nabla(\mathsf{A})$ is called the *displacement rank* of A with respect to ∇. A prominent example is the family of so-called *Toeplitz-like* matrices, which are structured for the Toeplitz displacement operator $\phi : \mathsf{A} \mapsto (\mathsf{Z}_m\mathsf{A} - \mathsf{A}\mathsf{Z}_n) = (\mathsf{A}\downarrow) - (\mathsf{A}\leftarrow)$ where the $n \times n$ lower shift matrix Z_n is the matrix with ones below the diagonal. The displacement rank of a Toeplitz matrix for this operator is at most two; the displacement rank of a mosaic Toeplitz with a $p \times q$ block structure is at most $p + q$.

The key idea of most algorithms for structured matrices is summarized by Pan's motto [42]: compress, operate, decompress. Indeed, for A of size $m \times n$ over a field $\mathbb{K}$, if $\nabla(\mathsf{A})$ 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 $\nabla(\mathsf{A}) = \mathsf{G}\mathsf{H}^t$; α is the *length* of the generators. The main idea behind algorithms for structured matrices is to use generators as a compact data structure, involving $\alpha(m + n)$ field elements instead of mn.

Cauchy-like matrices. Beyond the Toeplitz structure (and the directly related Hankel one), two other important cases are the so-called Vandermonde and Cauchy structures. While the case of Toeplitz-like matrices was the first one to be studied in detail, we will actually focus on Cauchy-like matrices, as we will see that this particular structure is quite convenient to work with.

For a sequence $\mathsf{u} = (u_1, \dots, u_m)$ in $\mathbb{K}^m$, let $\mathsf{D}_\mathsf{u} \in \mathbb{K}^{m\times m}$ be the diagonal matrix with entries $u_1, \dots, u_m$. Then, given u as above and v in $\mathbb{K}^n$, we will consider the operator $\nabla_{\mathsf{u},\mathsf{v}} : \mathsf{A} \in \mathbb{K}^{m\times n} \mapsto \mathsf{D}_\mathsf{u}\mathsf{A} - \mathsf{A}\mathsf{D}_\mathsf{v}$; *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 $\nabla_{\mathsf{u},\mathsf{v}}$-generators (G, H) of length α for A, we can reconstruct A as

$$\mathsf{A} = \sum_{i=1}^{\alpha} \mathsf{D}_{\mathsf{g}_i} \mathsf{C}_{\mathsf{u},\mathsf{v}} \mathsf{D}_{\mathsf{h}_i} = (\mathsf{G}\mathsf{H}^t) \odot \mathsf{C}_{\mathsf{u},\mathsf{v}}, \quad \mathsf{C}_{\mathsf{u},\mathsf{v}} = \left[\frac{1}{u_i - v_j}\right]_{\substack{1\le i\le m\\ 1\le j\le n}} \quad (1)$$

where g_i and h_i are the ith columns of respectively G and H, matrix $\mathsf{C}_{\mathsf{u},\mathsf{v}}$ is known as a *Cauchy matrix* and $\odot$ is the entry-wise product.

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 $\mathsf{A}_{I,J}$.

Another useful property relates to inverses of Cauchy-like matrices. If a matrix $\mathsf{A} \in \mathbb{K}^{n\times n}$ is invertible, and is structured with respect to an operator $\nabla_{\mathsf{u},\mathsf{v}}$, its inverse is structured with respect to $\nabla_{\mathsf{v},\mathsf{u}}$: if $\mathsf{D}_\mathsf{u}\mathsf{A} - \mathsf{A}\mathsf{D}_\mathsf{v} = \mathsf{G}\mathsf{H}^t$, one easily deduces that $\mathsf{D}_\mathsf{v}\mathsf{A}^{-1} - \mathsf{A}^{-1}\mathsf{D}_\mathsf{u} = -(\mathsf{A}^{-1}\mathsf{G})(\mathsf{A}^{-t}\mathsf{H})^t$, where A^{-t} is a shorthand for $(\mathsf{A}^{-1})^t$.

Algorithms. In most of the paper, we give costs in an algebraic model, counting base field operations at unit cost (in Section 4, we work over $\mathbb{Q}$ and use a boolean model). 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 [23, Chapter 8] hold. Using the Cantor-Kaltofen algorithm [14], 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 $\omega < 2.38$ [17, 33]. We will have to compute rank and rank profile of dense matrices; the cost reduces to that of matrix multiplication [28]. The notation $\tilde{O}(\,)$ indicates that we omit polylogarithmic terms.

Matrix-vector multiplication with $\mathsf{C}_{\mathsf{u},\mathsf{v}}$ reduces to degree n polynomial interpolation at the points v and evaluation at the points u. Using fast polynomial evaluation and interpolation, this can be done in time $O(\mathscr{M}(\nu)\log(\nu))$, with $\nu = \max(m, n)$; thus, we can multiply matrix A of (1) by a vector in time $O(\alpha\mathscr{M}(\nu)\log(\nu)) \subset \tilde{O}(\alpha\nu)$. In [42, 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 $\mathsf{C}_{\mathsf{u},\mathsf{v}}$ to $O(\mathscr{M}(\nu))$, since polynomial evaluation or interpolation at n points in geometric progression can be done in time $O(\mathscr{M}(n))$ [4, 11]; then, multiplication by A as above takes time $O(\alpha\mathscr{M}(\nu))$.

Remark 2.1. We propose here a refinement of this idea, that allows us to save a constant factor in runtime: we require that u and v be geometric progressions with *the same ratio* τ. Then, the Cauchy matrix $\mathsf{C}_{\mathsf{u},\mathsf{v}}$ has entries $1/(u_i - v_j) = 1/(u_1\tau^{i-1} - v_1\tau^{j-1})$, so it can be factored as

$$\mathsf{C}_{\mathsf{u},\mathsf{v}} = \mathsf{D}_{\tau^{-1}} \left[\frac{1}{u_1 - v_1\tau^{j-i}}\right]_{\substack{1\le i\le m\\ 1\le j\le n}},$$

where $\mathsf{D}_{\tau^{-1}}$ is diagonal with entries $(1, \tau^{-1}, \tau^{-2}, \dots, \tau^{1-m})$, and where the right-hand matrix is Toeplitz. In the reconstruction formula (1), the diagonal matrix $\mathsf{D}_{\tau^{-1}}$ commutes with all matrices $\mathsf{D}_{\mathsf{g}_i}$, 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. For $m = n$, using middle product techniques [10, 25], the cost goes down from $3\alpha\mathscr{M}(\nu) + O(\alpha\nu)$ to $\alpha\mathscr{M}(\nu) + O(\alpha\nu)$.

If one needs to multiply A by *several* vectors, further improvements are possible: we mention without giving details an algorithm from [8], that itself follows [9], and which makes it possible to multiply A by α vectors in time $O(\alpha^{\omega-1}\mathscr{M}(\nu))$ instead of $O(\alpha^2\mathscr{M}(\nu))$, by reduction to a sequence of polynomial matrix multiplications.

Reduction from mosaic Toeplitz to Cauchy, and regularization. Our primary interest lies in mosaic Toeplitz matrices. An important insight of Pan [40] shows that one can reduce questions about Toeplitz-like matrices to ones about Cauchy-like matrices (and conversely, if one wishes to), for a moderate cost overhead. To a vector u in $\mathbb{K}^m$, let us associate the corresponding Vandermonde matrices $\mathsf{V}_\mathsf{u} = [\,u_i^{j-1}\,]_{1\le i,j\le m}, \mathsf{W}_\mathsf{u} = [\,u_j^{m-i}\,]_{1\le i,j\le m}$. For u as above, v in $\mathbb{K}^n$ and T in $\mathbb{K}^{m\times n}$, we let $\mathsf{A} = \mathsf{V}_\mathsf{u}\,\mathsf{T}\,\mathsf{W}_\mathsf{v}$. If T is Toeplitz-like, then A is Cauchy-like : A has $\nabla_{\mathsf{u},\mathsf{v}}$-generators of length $p + q + 2$ (whereas T has ϕ-generators of length $p + q$, see [42, Chapter 4.8]). The bottleneck in the computation of $\nabla_{\mathsf{u},\mathsf{v}}$-generators from ϕ-generators are products of structured matrices by vectors. If all entries of u and v are in geometric progression, it takes time $O(p\mathscr{M}(n) + q\mathscr{M}(m))$.

In our algorithms for Cauchy-like matrices, we will assume that the input matrix has *generic rank profile*, that is, that its leading principal minors of size up to its rank are invertible. Regularization for structured matrices was introduced for this purpose by Kaltofen [32], for the Toeplitz structure. In our Cauchy context, one could apply to A as defined above the regularization procedure from [42, Section 5.6], which consists in replacing A by $\mathsf{A}' = \mathsf{D_x C_{a,u} A C_{v,b} D_y}$, for some new vectors $\mathsf{x}, \mathsf{a} \in \mathbb{K}^m$ and $\mathsf{y}, \mathsf{b} \in \mathbb{K}^n$. Theorem 5.6.2 in [42] shows that if a, u consist of $2m$ distinct scalars, and b, v consist of $2n$ distinct scalars, then there exists a non-zero polynomial Δ in the entries of x and y, of degree at most $\mu = \min(m, n)$ in each block of variables, such that the non-vanishing of Δ implies that A' has generic rank profile (that theorem is stated for square matrices, but the result holds in the rectangular case as well). The downside of this construction is that it requires to compute a pair of generators of length $p + q + 4$ for A', involving the multiplication of G' and H' by $\mathsf{C_{a,u}}$ and $\mathsf{C_{v,b}}$.

We now prove a new property, involving only $\mathsf{A} = \mathsf{V_u}\,\mathsf{T}\,\mathsf{W_v}$ as above. For the rest of this paragraph, we assume that the entries of u and v are indeterminates over $\mathbb{K}$, and we show that A has generic rank profile; this will imply the same property for a generic choice of u, v with entries in $\mathbb{K}$. This construction is clearly favorable over the one above, since it involves no extra computation on A. Following the proof of [42, Theorem 5.6.2], we can use the Cauchy-Binet formula to express the minors of A in terms of those of $\mathsf{V_u}$, T and $\mathsf{W_v}$. Let $i \le \mathrm{rank}(\mathsf{T})$ and let $I = \{1, \dots, i\}$. The determinant δ_i of the ith leading principal minor of A is the sum over all $J = \{j_1, \dots, j_i\} \subset \{1, \dots, m\}$ and $K = \{k_1, \dots, k_i\} \subset \{1, \dots, n\}$ of $\alpha_{I,J}\,\beta_{J,K}\,\gamma_{K,I}$, where $\alpha_{I,J}$ is the determinant of $(\mathsf{V_u})_{I,J}$, $\beta_{J,K}$ is the determinant of $\mathsf{T}_{J,K}$, and $\gamma_{K,I}$ is the determinant of $(\mathsf{W_v})_{K,I}$.

Let $<$ be the monomial ordering on the variables $u_1, \dots, u_i$, $v_1, \dots, v_i$ that first sorts monomials using the lexicographic order on $u_1, \dots, u_i$ with $u_1 < \cdots < u_i$, and then breaks the ties using the lexicographic order on $v_1, \dots, v_i$ with $v_1 > \cdots > v_i$. If $J = \{j_1, \dots, j_i\}$ with $j_1 < \cdots < j_i$ then the leading monomial (denoted $\mathrm{lm}(\cdot)$) of $\alpha_{I,J}$ is $\mathsf{u}^J := u_1^{j_1 - 1} \cdots u_i^{j_i - 1}$. Similarly if $K = \{k_1, \dots, k_i\}$ with $k_1 < \cdots < k_i$, then $\mathsf{v}^K := v_1^{n-k_1} \cdots v_i^{n-k_i} = \mathrm{lm}(\gamma_{K,I})$. Since T is of rank greater or equal to i, at least one of its ith minor $\beta_{J,K}$ does not vanish. Let $J_{\max}, K_{\max}$ be the pair of subsets that maximizes $\mathsf{u}^J \mathsf{v}^K$ among those for which $\beta_{J,K} \neq 0$. Then we must have $\mathrm{lm}(\det(\mathsf{A}_{I,I})) = \mathsf{u}^{J_{\max}} \mathsf{v}^{K_{\max}}$, which shows that $\det(\mathsf{A}_{I,I})$ is non-zero. The partial degree of $\det(\mathsf{A}_{I,I})$ in any variable is at most $\max(n, m)$.

3 OVER AN ABSTRACT FIELD

We work here over a field $\mathbb{K}$, and explain how to solve Problem A using Pan's reduction to the Cauchy-like Problem B below. Our goal is to solve a linear system, but the algorithms do slightly more: they compute the inverse of a given matrix (or of a maximal minor thereof); this is similar to what happens for dense matrices, where it is not known how to solve linear systems with an exponent better than ω. Then, the main question of this section is the following

Problem B. *Consider* $\mathsf{u} = (u_1, \dots, u_m)$ *and* $\mathsf{v} = (v_1, \dots, v_n)$, *such that* (u, v) *has* $m + n$ *distinct entries. Given* $\nabla_{\mathsf{u},\mathsf{v}}$*-generators of length* α *for* A *in* $\mathbb{K}^{m \times n}$, *with* $\alpha \le \min(m, n)$, *do the following.*

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

To solve an instance of Problem A, with input matrix T, we apply the transformation of the previous section, to obtain a Cauchy-like matrix $\mathsf{A} = \mathsf{V_u T W_v}$. We compute generators of A_r^{-1}, where A_r is the maximal leading minor of A, and we return a vector b with $\mathsf{b} = \mathsf{W_v} \begin{bmatrix} -\mathsf{A}_r^{-1}\mathsf{Bc} \\ \mathsf{c} \end{bmatrix}$, where $\mathsf{A}_{r,*} = \begin{bmatrix} \mathsf{A}_r & \mathsf{B} \end{bmatrix}$ and $\mathsf{c} \in \mathbb{K}^{n-r}$ random.

There exist two major classes of algorithms for handling Problem B, iterative ones, of cost $\Theta(mn)$ (for fixed α), and divide-and-conquer algorithms, of quasi-linear cost in $m + n$. We stress that having a fast quadratic-time algorithm is actually crucial in practice: as is the case for the Half-GCD, fast linear algebra algorithms, etc, the divide-and-conquer algorithm will fall back on the iterative one for input sizes under a certain threshold, and the performance of the latter will be an important factor in the overall runtime.

Iterative algorithms that solve a size n Toeplitz system in time $O(n^2)$ have been known since [20, 35, 50]; extensions to structured matrices were later given, as e.g. in [24]. After a few preliminary results, we give in 3.2 an algorithm inspired by [37, Algorithme 4], for the specific form of our Problem B. In this reference, Mouilleron sketches an algorithm that solves Problem B 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 [31, §1.10], who dealt with symmetric matrices. Our algorithm follows the same pattern, but reduces the cost to $O(\alpha^{\omega-2} mn)$.

In 3.3, we review divide-and-conquer techniques. Kaltofen [32] gave a divide-and-conquer algorithm that solves the analogue of Problem B for Toeplitz-like matrices, lifting assumptions of (strong) non-singularity needed in the original Morf and Bitmead-Anderson algorithm [3, 36]; a generalization to the Cauchy case is in [15, 45], and [44] discusses a unified version. A further improvement due to Jeannerod and Mouilleron [29], following [15], allows one to bypass costly compression stages that are needed in Kaltofen's algorithm and its extensions, by predicting the shape of the generators we have to compute. For the case of square Cauchy-like matrices of size n, this results in an algorithm of cost $O(\alpha^2 \mathscr{M}(n) \log(n)^2)$, but we will point out that better estimates are available by choosing u, v suitably.

3.1 Cauchy generators of the inverse

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 $\mathsf{u} = (u_1, \dots, u_m)$ and $\mathsf{v} = (v_1, \dots, v_n)$. 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. Below, we write $\mu = \min(m, n)$.

For $0 \le i \le \mu$, write $\mathsf{A} = \begin{bmatrix} \mathsf{A}_{0,0}^{(i)} & \mathsf{A}_{0,1}^{(i)} \\ \mathsf{A}_{1,0}^{(i)} & \mathsf{A}_{1,1}^{(i)} \end{bmatrix}$ with $\mathsf{A}_{0,0}^{(i)} \in \mathbb{K}^{i \times i}$, and

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

if $\mathsf{A}_{0,0}^{(i)}$ is invertible (this is similar to [15], where the order of block rows and columns was however different; see also Lemma 3.1 in [39] and references therein). The sequence of matrices $(\mathsf{S}^{(i)})_{i=0 \dots \mu}$ starts from $\mathsf{A} = \mathsf{S}^{(0)}$ and ends at $\mathsf{A}^{-1} = \mathsf{S}^{(\mu)}$, at least when A is square and invertible.

Given integers a, b, $\mathsf{u}_{a:b}$ denotes the sequence $(u_a, \dots, u_b)$, and similarly for $\mathsf{v}_{a:b}$; we then define $\mathsf{u}^{(i)} = (\mathsf{u}_{i+1:m}, \mathsf{v}_{1:i})$ and $\mathsf{v}^{(i)} = (\mathsf{v}_{i+1:n}, \mathsf{u}_{1:i})$. A key result for the sequel is Lemma 3.1, which is [15, Proposition 1]; it gives $\nabla_{\mathsf{u}^{(i)},\mathsf{v}^{(i)}}$-generators of $\mathsf{S}^{(i)}$ (remark that this operator is invertible, in view of our assumption on u and v).

For i as above, decompose G,H as $\mathrm{G} = \begin{bmatrix} \mathrm{G}_0^{(i)} \\ \mathrm{G}_1^{(i)} \end{bmatrix}$, $\mathrm{H} = \begin{bmatrix} \mathrm{H}_0^{(i)} \\ \mathrm{H}_1^{(i)} \end{bmatrix}$ with $\mathrm{G}_0^{(i)}, \mathrm{H}_0^{(i)} \in \mathbb{K}^{i\times\alpha}$ and define $(\mathrm{Y}^{(0)}, \mathrm{Z}^{(0)}) = (\mathrm{G},\mathrm{H})$ and

$$\mathrm{Y}^{(i)} = \begin{bmatrix} -\mathrm{A}_{1,0}^{(i)}\mathrm{A}_{0,0}^{(i)\,-1}\mathrm{G}_0^{(i)} + \mathrm{G}_1^{(i)} \\ -\mathrm{A}_{0,0}^{(i)\,-1}\mathrm{G}_0^{(i)} \end{bmatrix}, \mathrm{Z}^{(i)} = \begin{bmatrix} -\mathrm{A}_{0,1}^{(i)\,t}\mathrm{A}_{0,0}^{(i)\,-t}\mathrm{H}_0^{(i)} + \mathrm{H}_1^{(i)} \\ \mathrm{A}_{0,0}^{(i)\,-t}\mathrm{H}_0^{(i)} \end{bmatrix}.$$

Lemma 3.1. $(\mathrm{Y}^{(i)}, \mathrm{Z}^{(i)})$ *are* $\nabla_{\mathrm{u}^{(i)},\mathrm{v}^{(i)}}$*-generators for* $\mathrm{S}^{(i)}$.

If A has generic rank profile, this shows that for $r = \mathrm{rank}(\mathrm{A})$, $(\mathrm{Y}_0^{(r)}, \mathrm{Z}_0^{(r)})$ are $\nabla_{\mathrm{v}',\mathrm{u}'}$-generators for $\mathrm{A}_{0,0}^{(r)-1}$, for u',v' as in Problem B, so they solve our problem; it remains to explain how to compute these matrices. Let i,j be such that $0 \le i+j \le \mu$, and $\mathrm{A}_{0,0}^{(i)}$ invertible. Decompose $\mathrm{S}^{(i)}$, $\mathrm{Y}^{(i)}$ and $\mathrm{Z}^{(i)}$ into blocks

$$\mathrm{S}^{(i)} = \begin{bmatrix} \mathrm{S}_{0,0}^{(i,j)} & \mathrm{S}_{0,1}^{(i,j)} \\ \mathrm{S}_{1,0}^{(i,j)} & \mathrm{S}_{1,1}^{(i,j)} \end{bmatrix}, \ \mathrm{Y}^{(i)} = \begin{bmatrix} \mathrm{Y}_0^{(i,j)} \\ \mathrm{Y}_1^{(i,j)} \end{bmatrix}, \ \mathrm{Z}^{(i)} = \begin{bmatrix} \mathrm{Z}_0^{(i,j)} \\ \mathrm{Z}_1^{(i,j)} \end{bmatrix}$$

with $\mathrm{S}_{0,0}^{(i,j)} \in \mathbb{K}^{j\times j}$ and $\mathrm{Y}_0^{(i,j)}, \mathrm{Z}_0^{(i,j)} \in \mathbb{K}^{j\times\alpha}$, write $\mathrm{u}^{(i)} = [\, \mathrm{u}_0^{(i,j)} \ \mathrm{u}_1^{(i,j)} \,]$ with $\mathrm{u}_0^{(i,j)} \in \mathbb{K}^j$ and the same for $\mathrm{v}^{(i)}$. A direct calculation shows

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

This implies the following formulae on the generators, which generalizes the previous formulae for $(\mathrm{Y}^{(i)}, \mathrm{Z}^{(i)})$:

$$\begin{aligned} \mathrm{Y}_1^{(i+j,m-j)} &= -\mathrm{S}_{0,0}^{(i,j)-1}\,\mathrm{Y}_0^{(i,j)}, \quad \mathrm{Z}_1^{(i+j,n-j)} = \mathrm{S}_{0,0}^{(i,j)^{-t}}\,\mathrm{Z}_0^{(i,j)}, \\ \mathrm{Y}_0^{(i+j,m-j)} &= -\mathrm{S}_{1,0}^{(i,j)}\,\mathrm{Y}_1^{(i+j,m-j)} + \mathrm{Y}_1^{(i,j)}, \qquad (3) \\ \mathrm{Z}_0^{(i+j,n-j)} &= -\mathrm{S}_{0,1}^{(i,j)\,t}\,\mathrm{Z}_1^{(i+j,n-j)} + \mathrm{Z}_1^{(i,j)}. \end{aligned}$$

3.2 A faster iterative algorithm

Following this discussion, we describe in Algorithm 1 a quadratic iterative algorithm for Problem B, using a step size $\beta \in \{1,\dots,\alpha\}$ given as a parameter. Computing the rank profile (line 5) costs $O(\beta^\omega)$. Using block matrix multiplication in Eq. (1), we recover $\mathrm{S}_{0,1}^{(i,j)}$ from its $\nabla_{\mathrm{u}_0^{(i,j)},\mathrm{v}_1^{(i,j)}}$-generators $(\mathrm{Y}_0^{(i,j)}, \mathrm{Z}_1^{(i,j)})$, and similarly for $\mathrm{S}_{1,0}^{(i,j)}$, in time $O(\beta^{\omega-2}\alpha(m+n))$, since $j \le \beta \le \alpha$ (line 8). To compute $\mathrm{Y}^{(i+j)}$ at line 9, we first compute $\mathrm{S}_{0,0}^{(i,j)-1}$ in time $O(\beta^\omega)$, $\mathrm{S}_{0,0}^{(i,j)-1}\mathrm{Y}_1^{(i,j)}$ in time $O(\beta^{\omega-1}\alpha)$; then all other operations take time $O(\beta^{\omega-2}\alpha m)$. Similarly, computing $\mathrm{Z}^{(i+j)}$ takes time $O(\beta^{\omega-2}\alpha n)$. We recover $\mathrm{S}^{(\bar r)}$ at line 11 from its $\nabla_{\mathrm{u}^{(\bar r)},\mathrm{v}^{(\bar r)}}$-generators $(\mathrm{Y}^{(\bar r)}, \mathrm{Z}^{(\bar r)})$ using (1) in time $O(\alpha^{\omega-2}mn)$, by block matrix multiplication.

One iteration of the while loop costs $O(\beta^{\omega-2}\alpha(m+n))$; we iterate $O(\mu/\beta)$ times, for a total of $O(\beta^{\omega-3}\alpha\mu(m+n))$, which is $O(\beta^{\omega-3}\alpha mn)$. This dominates the cost of line 11, so the whole runtime is $O(\beta^{\omega-3}\alpha mn)$. The algorithm of [37] 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 $O(\alpha^{\omega-2}mn)$. This theoretical improvement has also practical implications : $\omega < 3$ is achieved in practice [13, 19], and linear algebra implementations are highly optimized and gain a significant constant factor even for $\omega = 3$.

Algorithm 1: Iterative algorithm Iter for Problem B

```
   Input  : Generators $(\mathrm{G},\mathrm{H})$ of A and step size $\beta$
   Output : $r = \mathrm{rank}(\mathrm{A})$, generators of $\mathrm{A}_{0,0}^{(r)-1}$
1  $i = 0, \mathrm{Y}^{(0)} = \mathrm{G}, \mathrm{Z}^{(0)} = \mathrm{H}, \bar r = \mu$
2  while $i \ne \bar r$ do
3      $j = \min(\beta, \bar r - i)$
4      Recover $\mathrm{S}_{0,0}^{(i,j)}$ from its generators $(\mathrm{Y}_0^{(i,j)}, \mathrm{Z}_0^{(i,j)})$
5      Compute the rank $\rho$, rank profile, inverse of $\mathrm{S}_{0,0}^{(i,j)}$
6      if $\mathrm{S}_{0,0}^{(i,j)}$ has non generic rank profile then signal error
7      if $\rho < j$ then $j = \rho$, $\bar r = i + \rho$
8      Recover $\mathrm{S}_{0,1}^{(i,j)}$, $\mathrm{S}_{1,0}^{(i,j)}$ from their generators
9      Compute the generators $(\mathrm{Y}^{(i+j)}, \mathrm{Z}^{(i+j)})$ of $\mathrm{S}^{(i+j)}$
10     $i = i + j$
11 Recover $\mathrm{S}^{(\bar r)}$ from its generators $(\mathrm{Y}^{(\bar r)}, \mathrm{Z}^{(\bar r)})$
12 Read in $\mathrm{S}^{(\bar r)}$ the Schur complement of $\mathrm{A}_{0,0}^{(\bar r)}$ in A
13 if the Schur complement is non zero then signal error
14 else return $(\bar r, \mathrm{Y}_1^{(\bar r, m-\bar r)}, \mathrm{Z}_1^{(\bar r, n-\bar r)})$
```

3.3 The divide-and-conquer algorithm

We finally review the divide-and-conquer approach to solving Problem B, as a basis for the discussion of the next subsection. Algorithm 2 follows [29], recast in our framework, with the minor difference that we do not assume A invertible, and that we explicitly check if A satisfies the generic rank profile assumption.

Step 4, 9 are similar: using Eq. (2) and (3), we recover generators $(\mathrm{Y}^{(i+j)}, \mathrm{Z}^{(i+j)})$ of $\mathrm{S}^{(i+j)}$ from $(\mathrm{Y}_1^{(i+j,m-j)}, \mathrm{Z}_1^{(i+j,n-j)})$ of $\mathrm{S}_{0,0}^{(i,j)-1}$ and previous full generators $(\mathrm{Y}^{(i)}, \mathrm{Z}^{(i)})$ of $\mathrm{S}^{(i)}$. Rather than reconstructing the submatrices of $\mathrm{S}^{(i,j)}$ as in the previous subsection, we use them through their generators: the formulae of (3) do $O(1)$ multiplications of Cauchy-like matrices (for which we have generators of length α) by α vectors. In general, these multiplications cost $O(\alpha^2\mathscr{M}(\nu)\log(\nu))$ operations, with $\nu = \max(m,n)$. As pointed out in [42, Theorem 5.3.1] (see also [16]), if u and v are geometric progressions, the cost for the matrix-vectors multiplication drops to $O(\alpha^2\mathscr{M}(\nu))$. If u and v have the same ratio (if this is the case at the top-level, this will remain the case for all recursive calls), by Remark 2.1, we can save a further constant factor. For large values of α, we can also apply the algorithm of [8], which uses fast polynomial matrix multiplication to reduce the cost to $O(\alpha^{\omega-1}\mathscr{M}(\nu))$.

On line 6, the Schur complement is zero iff $\mathrm{Y}'\,\mathrm{Z}' = 0$. This is tested by finding a minimal set of independent rows in Z' (this takes time $O(\alpha^{\omega-1}\nu)$) and multiplying their transposes by Y'. There are at most α such rows, so this takes time $O(\alpha^{\omega-1}\nu)$ as well. Taking all recursive steps into account, the total cost of DAC is a factor $\log(\nu)$ times that of the Cauchy-like matrix products.

3.4 Experimental results

We implemented the algorithms described so far, together with all subroutines they rely on, in a C++ library available at https://github.com/romainlebreton/structured_linear_system_solving. Our implementation is based on Shoup's NTL [48, 49] version 10.3.0, and is dedicated to word-size primes (NTL's `lzz_p` class); the divide-and-conquer algorithm of Subsection 3.3 actually requires FFT primes. In all that follows, timings are measured on an Intel i7-4790 CPU with 32 GB RAM; only one thread is used throughout.

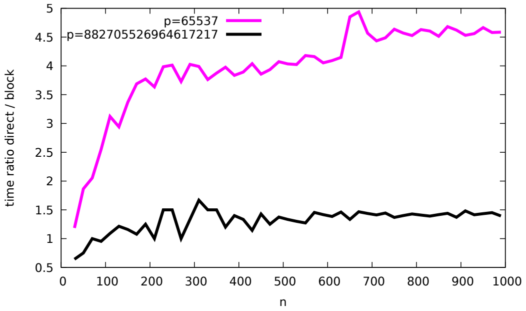
Algorithm 2: Divide-And-Conquer algorithm DAC

```
Input  : Generators $(\mathsf{G},\mathsf{H})$ of $\mathsf{A}$, threshold $\nu_0$
Output : $r = \mathrm{rank}(\mathsf{A})$, generators of $\mathsf{A}_{0,0}^{(r)-1}$
$\mu = \min(m,n)$, $i = \lceil \mu/2 \rceil$, $(\mathsf{Y}^{(0)},\mathsf{Z}^{(0)}) = (\mathsf{G},\mathsf{H})$
if $\max(m,n) \le \nu_0$ then return $\mathsf{Iter}(\mathsf{G},\mathsf{H},\alpha)$
$(r_0, \mathsf{Y}_1^{(r_0,m-r_0)}, \mathsf{Z}_1^{(r_0,n-r_0)}) = \mathsf{DAC}(\mathsf{Y}_0^{(0,i)}, \mathsf{Z}_0^{(0,i)}, \nu_0)$
Recover Schur compl. gen. $(\mathsf{Y}_0^{(r_0,m-r_0)}, \mathsf{Z}_0^{(r_0,n-r_0)}) = (\mathsf{Y}',\mathsf{Z}')$
  from $(\mathsf{Y}_1^{(r_0,m-r_0)}, \mathsf{Z}_1^{(r_0,n-r_0)})$ and $(\mathsf{Y}^{(0)},\mathsf{Z}^{(0)})$
if $r_0 < i$ then
    if Schur complement is non zero then signal error
    else return $(r_0, \mathsf{Y}_0^{(r_0)}, \mathsf{Z}_0^{(r_0)})$
$(r_1, \mathsf{Y}_1^{(r_2,m-r_1)}, \mathsf{Z}_1^{(r_2,n-r_1)}) = \mathsf{DAC}(\mathsf{Y}',\mathsf{Z}',\nu_0)$, $r_2 = r_0 + r_1$
return $r_2$ and $(\mathsf{Y}_1^{(r_2,m-r_2)}, \mathsf{Z}_1^{(r_2,n-r_2)})$ computed from
  $(\mathsf{Y}_1^{(r_2,m-r_1)}, \mathsf{Z}_1^{(r_2,n-r_1)})$ and $(\mathsf{Y}^{(r_0)},\mathsf{Z}^{(r_0)})$
```

Our goals are to assess if matrix multiplication can bring practical improvements that reflect the theoretical ones, find crossover points between iterative and divide-and-conquer algorithms, and what is the range of applicability of these structured methods compared to dense linear algebra. Thus, we focused on comparisons between our own implementations of the various techniques seen so far.

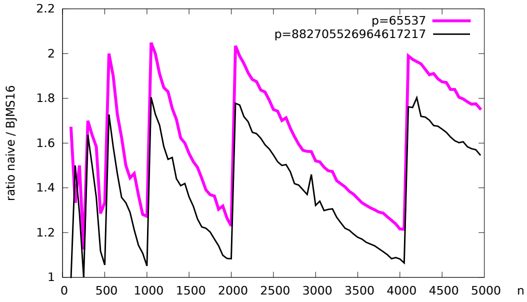
NTL already offers efficient FFT-based polynomial arithmetic; matrix multiplication over small fields $\mathbb{Z}/p\mathbb{Z}$, for $p < 2^{23}$, is now extremely efficient, comparable to reference implementations such as FFLAS-FFPACK [21]. On top of this, we implemented a fast polynomial matrix multiplication, using the cyclotomic TFT of [1].

We first discuss Problem B. Our first tests compare our new $O(\alpha^{\omega-2}n^2)$ algorithm to that of [37], with runtime $O(\alpha n^2)$. Except for very small values of n, say $n < 50$, for which the behavior fluctuates rapidly, we found that the new algorithm becomes more efficient for rather small values of α: our crossover points are $\alpha = 8$ or 9 for primes less than 2^{23}, and $\alpha = 13$ or 14 for larger word-size primes. The following graph shows the time ratio between the algorithm of [37] and our algorithm, for $\alpha = 30$; the larger value of p used here is the FFT prime $p = 82705526964617217 = 7^2 2^{54} + 1$. We see that for small primes, for which matrix multiplication is very efficient, our algorithm brings a substantial improvement.



We consider next the divide-and-conquer algorithm. The key factor for the efficiency of this algorithm is the cost of multiplying an $n \times n$ Cauchy-like matrix of displacement rank α by α vectors. We compare the approach of cost $O(\alpha^2 \mathscr{M}(n))$ of Remark 2.1 to the algorithm of [8], with cost $O(\alpha^{\omega-1}\mathscr{M}(n))$, with a view of determining for what values of α (if any) the latter becomes useful.

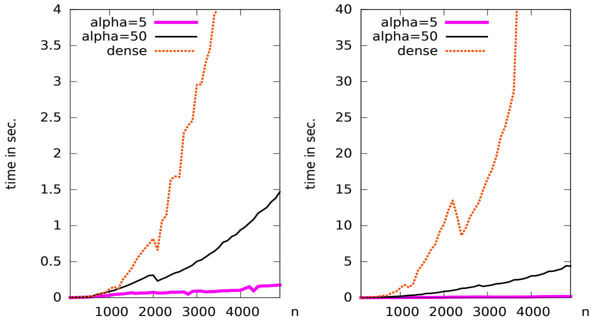
For the former algorithm, because we are able to cache several FFTs, we found it slightly more advantageous to use NTL's FFT rather than TFTs. The runtime of the first algorithm then displays the typical FFT staircase behavior, so that as n grows, the crossover value for α fluctuates, roughly between 30 and 55. The following graph shows the time ratio between the algorithm of [8] and the direct one, for $\alpha = 60$; at best, the new algorithm wins by a factor of 2. The results do not depend much on the nature of the prime (since polynomial arithmetic is a significant part of the runtime, and behaves essentially in the same manner, independently of p).



Using these results, we determined empirical crossover values $\nu_0(\alpha)$ to end the recursion in the divide-and-conquer algorithm and switch to the iterative algorithm. We expect $\nu_0(\alpha)$ to grow with α, since it behaves like the solution of $\alpha n^2 = \alpha^2 \mathscr{M}(n)\log(n)$ (assuming here we do not use fast linear algebra, for simplicity). The following table reports some values for ν_0 (obtained by searching for ν_0 in increments of 100). The threshold is higher for small primes, since such primes benefit more from our new iterative algorithm.

α	1	5	10	15	20	25	30
$p = 65537$	400	400	500	1000	2000	3300	4200
$p \simeq 2^{59}$	200	400	400	700	1300	1600	2000

These values being set, we show runtimes for solving Problem B modulo $p = 65537$ and $p = 882705526964617217$, for increasing values of n, with $\alpha = 5$ and $\alpha = 50$; we also show the runtime of a dense matrix inversion in the same size. For a small displacement rank such as $\alpha = 5$, the runtime is essentially the same for these two primes; with $\alpha = 50$, we observe a difference, by a factor of up to 3. In any case, there is a clear gain over dense methods.



We also determined, for a given value of n, the crossover value α_0 above which dense linear algebra becomes faster than structured methods. The value $\alpha_0(n)$ grows quite regularly with n, good approximations being $\alpha_0(n) \simeq 0.2n$ (for $p < 2^{23}$) and $\alpha_0(n) \simeq 0.25n$ (for larger values of p). This means that there is a wide range of inputs for which structured methods can be of use.

Our solution of Problem A is by a direct reduction to the Cauchy-like case. Putting the problem into Cauchy form accounts for a small fraction of the total runtime: between 5% and 10% for small α (say $\alpha < 10$), and less than 5% for larger values of α, in all instances we considered. In the thousands of experiments we made, for matrix sizes such as $n \geq 5000$ and small primes $p \simeq 2^{16}$, we observed some instances where the Cauchy matrix did not have generic rank profile. This never happened for p having more than 50 bits.

4 MODULAR TECHNIQUES

We now address Problem A for $\mathbb{K} = \mathbb{Q}$. Several *modular algorithms* are available to solve dense linear systems over $\mathbb{Q}$. A first option relies on the Chinese Remainder Theorem, solving the system modulo several primes $p_1, p_2, \ldots$ before reconstructing the solution, making sure to ensure consistency of the modular solutions when $\ker(\mathsf{T})$ has dimension more than 1. Other approaches such as Newton iteration or divide-and-conquer algorithms use one prime p, and lift the solution modulo powers of p. We first briefly present these techniques and analyze their complexity; we will discuss their practical performance. We highlight the case of algebraic approximants, for which we are able to obtain significant improvements.

We denote by $\mathscr{I} : \mathbb{N} \to \mathbb{R}$ a function such that integers of bit size at most d can be multiplied in $\mathscr{I}(d)$ bit operations; we can take $\mathscr{I}(d) = O(d \log(d) \log\log(d))$ or $\mathscr{I}(d) = d \log(d) 2^{O(\log^*(d))}$ [22, 46]. As in [23], we assume that $d \mapsto \mathscr{I}(d)/d$ is non-decreasing.

Newton iteration for structured matrices goes back to [41]; see details in [42, Chapter 7]; the divide-and-conquer approach is discussed in [34]. Pan *et al.* discuss in [43] alorithms inspired by Dixon's lifting algorithm [18], for non-singular and singular matrices (in our experiments, Dixon's algorithm was not competitive with the other approaches, so we do not discuss it here).

4.1 Solving square systems by lifting

We are given a prime power p^t, where t is a power of 2, an $n \times n$ matrix A and a vector b, both with entries modulo p^t, such that A is invertible modulo p^t, with inverse B; we discuss algorithms that solve the equation $\mathsf{Ax} = \mathsf{b}$. To simplify the cost analysis, we assume here that $p = O(1)$. We take $\mathscr{C}_\mathsf{A}$ such that, for any t, we can compute $\mathsf{Ax} \bmod p^t$ in $O(\mathscr{C}_\mathsf{A} \mathscr{I}(t))$ bit operations, for a vector x with entries defined modulo p^t. Below, we will assume that A is Cauchy-like as in Remark 2.1, and given by generators of length α, so we can take $\mathscr{C}_\mathsf{A} \in O(\alpha \mathscr{M}(n))$. We will see however that in the case of algebraic approximants, better estimates are available.

We consider two approaches: a divide-and-conquer algorithm and Newton iteration, which both feature a running time linear in the target precision t. We start with the divide-and-conquer approach. A version of it is in [2] for dense matrices; the PhD thesis [34] describes this algorithm for Toeplitz-like matrices.

Algorithm 3: Divide-And-Conquer algorithm $\mathsf{DAC}_\mathbb{Q}$

```
**Input**  : A, b, B mod $p$, $p$, $t$ as above
**Output** : a solution of $\mathsf{Ax} = \mathsf{b} \bmod p^t$
**if** $t = 1$ **then return** $\mathsf{B}\,\mathsf{b} \bmod p$
Compute $\mathsf{x}_0 = \mathsf{DAC}_\mathbb{Q}(\mathsf{A}, \mathsf{b}, \mathsf{B}, p, t/2)$
Compute $\mathsf{r}_0 = (\mathsf{Ax}_0 - \mathsf{b}) \bmod p^t$ and $\mathsf{r}_1 = r_0/p^{t/2}$
Compute $\mathsf{x}_1 = \mathsf{DAC}_\mathbb{Q}(\mathsf{A}, \mathsf{r}_1, \mathsf{B}, p, t/2)$
**return** $\mathsf{x}_0 - p^{t/2}\mathsf{x}_1 \bmod p^t$
```

Assume that we have know generators of length α for B. Thus, at the leaves of the recursion, each product $\mathsf{Bb} \bmod p$ can be computed using $O(\alpha \mathscr{M}(n))$ bit operations; with our assumption on $\mathscr{I}$, the total runtime is $O(\mathscr{C}_\mathsf{A} \mathscr{I}(t) \log(t) + \alpha \mathscr{M}(n) t)$ bit operations. With our upper bounds on $\mathscr{C}_\mathsf{A}$, this simplifies as $O(\alpha \mathscr{M}(n) \mathscr{I}(t) \log(t))$.

We turn next to Newton iteration. The matrix form of Newton iteration computes the inverses $\mathsf{B}_k = \mathsf{A}^{-1} \bmod p^{2^k}$ by $\mathsf{B}_{k+1} = 2\mathsf{B}_k - \mathsf{B}_k \mathsf{A} \mathsf{B}_k \bmod p^{2^{k+1}}$; once they are known, we deduce the solution to our system by means of a matrix-vector product.

Let (G, H) be generators for A. Pan's insight was to use the relation above to compute the generators $(\mathsf{X}, \mathsf{Y}) = (-\mathsf{BG}, \mathsf{B}^t\mathsf{H})$ for B. Given $(\mathsf{X}_k, \mathsf{Y}_k) = (\mathsf{X}, \mathsf{Y}) \bmod p^{2^k}$, we can reconstruct B_k, but to deduce $(\mathsf{X}_{k+1}, \mathsf{Y}_{k+1})$, we have to multiply G and H by B_{k+1} or B_{k+1}^t. This is done using the expression for B_{k+1} above, which gives

$$\begin{aligned} \mathsf{X}_{k+1} &= -(2\mathsf{B}_k - \mathsf{B}_k \mathsf{A} \mathsf{B}_k) \;\; \mathsf{G} \bmod p^{2^{k+1}} \\ \mathsf{Y}_{k+1} &= \;\;\;\, (2\mathsf{B}_k - \mathsf{B}_k \mathsf{A} \mathsf{B}_k)^t \; \mathsf{H} \bmod p^{2^{k+1}}. \end{aligned}$$

This time, we are multiplying A and B_k (and their transposes) by matrices of size $n \times \alpha$, all computations being done modulo $p^{2^{k+1}}$. Each multiplication by *one* vector takes $O(\alpha \mathscr{M}(n) \mathscr{I}(2^k))$ operations. Since we multiply these matrices by α vectors, taking all steps into account, we arrive at a total of $O(\alpha^2 \mathscr{M}(n) \mathscr{I}(t))$ bit operations. Using the algorithm of [8], this can be further reduced to $O(\alpha^{\omega-1} \mathscr{M}(n) \mathscr{I}(t))$; this improvement was not implemented.

Altogether, because it computes (generators of) a whole inverse, Newton iteration is slower by a factor α (or slightly less, if we use [8]); on the other hand, it saves a factor of $\log(t)$. This is similar to what one observed when comparing these techniques for e.g. the solution of differential equations [5, 7].

4.2 Solving Problem A

In order to solve instances of Problem A, that is, find nullspace elements for a mosaic Toeplitz matrix T, we rely on the transformation to the Cauchy structure / regularization technique of Section 2. Over $\mathbb{Q}$, this approach has a certain shortcoming: the output is a vector $\mathsf{b} = \mathsf{W}_\mathsf{v} \begin{bmatrix} -\mathsf{A}_{0,0}^{(r)-1} \mathsf{A}_{0,1}^{(r)} \mathsf{c} \\ \mathsf{c} \end{bmatrix}$, with $\mathsf{A} = \mathsf{V}_\mathsf{u} \mathsf{T} \mathsf{W}_\mathsf{v}$, $\mathsf{A}_{0,0}^{(r)}$ a maximal minor of it and c a random vector in $\mathbb{K}^{n-r}$, and due to the preconditioning, the entries of b are expected to be of large height (larger than what we may expect for a solution of T). When $\ker(\mathsf{T})$ has dimension 1, we are not affected by this issue: in this case, all solutions are of the form $\lambda \mathsf{b}_0$, for some vector $\mathsf{b}_0 \in \mathbb{Z}^n$ whose bit-size can be bounded only in terms of T. Hence, it suffices to compute the solution b of the regularized system modulo a large enough integer N and normalize it by setting one of its entries to 1. In general, we reduce the nullspace dimension by adding a new block of equations; when $\ker(\mathsf{T})$ has moderate dimension, this barely affects the overall runtime (for particular applications to algebraic approximants, another solution is described below).

For simplicity, we give here a solution that may run forever in unlucky cases. Choose a prime p and compute $\nabla_{\mathsf{u},\mathsf{v}}$-generators for $\mathsf{A} = \mathsf{V}_\mathsf{u} \mathsf{T} \mathsf{W}_\mathsf{v} \pmod{p}$, for u, v as in Remark 2.1. Call the algorithm of the previous section, to determine the rank of A; if A does not have generic rank profile, choose another u, v. After an expected $O(1)$ attempts, we obtain the rank of A over $\mathbb{F}_p$, and thus the dimension s of its nullspace. If s is greater than one, we add a block of Toeplitz matrices having $s - 1$ rows to T, with small random entries, and update A accordingly. Heuristically the new matrix T has nullspace dimension 1 (otherwise, add another block of equations).

Let $d = \mathrm{rank}(\mathsf{T})$, A_d be the $d \times d$ top-left submatrix of A, and b_d be the vector of the first d entries of the last column of A. We assume that A has generic rank profile, so that A_d is invertible. We compute $\mathsf{x} = \mathsf{A}_d^{-1}\mathsf{b}_d \pmod p$ and $\mathsf{y} = \mathsf{W}_{\mathsf{v}}[x_1, \cdots, x_d, -1]^t \pmod p$, we normalize y by dividing it by its first non-zero entry, and we set $t = 1$. While we either cannot apply rational reconstruction to the entries of y, or after applying rational reconstruction to y we have $\mathsf{Ty} \neq 0$, we do the following: set $t = 2t$, update modulo p^t the quantities A_d, b_d, x (use one of the algorithms of Section 4.1) and y, and divide y by its first non-zero entry.

A complete analysis of this algorithm would quantify the primes of bad reduction, give bounds that allow us to stop lifting the solutions if we reduced modulo such a bad prime and study the previous reduction to nullity one of T; we leave this to future work. In any case, if the lifting stops, we have obtained a solution to our system. Due to the doubling nature of this procedure, the runtime is proportional to that of the algorithm for solving square systems used at line 3, plus the cost of rational reconstruction. The former is $O(\mathscr{C}_{\mathsf{A}}\mathscr{I}(t)\log(t) + \alpha\mathscr{M}(n)t)$ bit operations using divide-and-conquer methods, and $O(\alpha^2\mathscr{M}(n)\mathscr{I}(t))$ or $O(\alpha^{\omega-1}\mathscr{M}(n)\mathscr{I}(t))$ using Newton iteration; the latter is $O(n\mathscr{I}(t)\log(t))$.

Finally, we mention the cost of Chinese Remaindering techniques: instead of computing the solution modulo the t-th power of a single prime p, we might want to solve the system modulo t primes of the same magnitude. If we assume that A remains invertible with generic rank profile modulo all these primes, and all these primes are $O(1)$, the runtime for solving the systems is now $O(\alpha^2\mathscr{M}(n)\log(n)t)$ or $O(\alpha^{\omega-1}\mathscr{M}(n)\log(n)t)$, by the results of Section 3 (to this, we have to add the cost $O(n\mathscr{I}(t)\log(t))$ of Chinese Remaindering and rational reconstruction).

We conclude this subsection with an important particular case, the computation of algebraic approximants. We are given a power series f in $\mathbb{Q}[[x]]$, together with degree bounds d, e; our goal is to compute a polynomial $P \in \mathbb{Q}[x, y]$, with $\deg(P, x) \leq d$, $\deg(P, y) \leq e$, such that $P(x, f) = 0$. For any $\sigma \geq 0$, finding P with the above degree bounds and such that $P(x, f) = 0 \bmod x^\sigma$ is a Hermite-Padé approximation problem of a very special kind (all input series are powers of f). To our knowledge, no algorithm in the framework of Section 3 can exploit this extra structure; in the case of computations over $\mathbb{Q}$, we will now see that improvements are possible.

First, we show how to simplify the reduction to nullity one. Théorème 7.15 in [6] shows that if such a P exists and is irreducible, then any $Q \in \mathbb{Q}[x, y]$ with the same degree bounds as above and such that $Q(x, f) = 0 \bmod x^{2de+1}$ is a multiple of P. From this, one deduces easily that if we compute *two* such polynomials Q_1, Q_2, their GCD will generically be P itself. For all primes p except a finite number, the rank of our matrix T does not change modulo p, and the GCD of two random basis elements commutes with reduction modulo p (note that $P \bmod p$ may not be irreducible anymore). Hence, we can (probabilistically) find $P \bmod p$ by computing two solutions Q_1, Q_2 to the above Hermite-Padé problem modulo p and taking their GCD. This reveals the support of P; we can then refine the degree bounds in our Hermite-Padé problem, which in turn reduces the nullity of matrix T to 1.

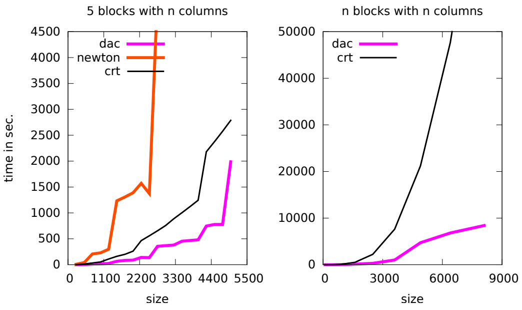
Next, we show how to speed-up algorithm $\mathsf{DAC}_{\mathbb{Q}}$ in this case. The block-Toeplitz matrix T of our Hermite-Padé problem has displacement rank $\alpha = O(e)$, with $e + 1$ Sylvester blocks having $O(d)$ columns and $O(de)$ rows; as a result, the naive estimate on the cost of the matrix-vector product by matrix $\mathsf{A} = \mathsf{V}_{\mathsf{u}}\mathsf{T}\mathsf{W}_{\mathsf{v}}$ is $\mathscr{C}_{\mathsf{A}} = O(e\mathscr{M}(de))$. However, we can reduce this cost using baby-steps / giant steps techniques: the bivariate modular composition algorithm of [38] shows that we can do matrix-vector product by T using $O(e^{1/2}\mathscr{M}(de) + e^{(\omega+1)/2}\mathscr{M}(d))$ operations; since multiplications by V_{u} and W_{v} take $O(\mathscr{M}(de))$, we obtain the improved estimate $\mathscr{C}_{\mathsf{A}} = O(e^{1/2}\mathscr{M}(de) + e^{(\omega+1)/2}\mathscr{M}(d))$ in this case. Algorithm $\mathsf{DAC}_{\mathbb{Q}}$ is the only algorithm we know of that takes into account the extra structure of algebraic approximants; we expect that similar improvements are possible for *differential approximants*, using the evaluation algorithm of [12].

4.3 Experimental Results

Our experiments are dedicated to solving instances of Hermite-Padé approximation, which is a useful particular case of Problem A. We discuss two families of problems, first the approximation of general power series, then algebraic approximants. Timings are measured on the same machine as Subsection 3.4.

In both cases, we show two graphs: on the left, we have five blocks, each with n columns; on the right, we have n blocks, each with n columns. With the notation of the introduction, this means we are looking for approximants $(p_0, \ldots, p_4)$, resp. $(p_0, \ldots, p_{n-1})$, with degree bounds (n, n, n, n, n), resp. $(n, \ldots, n)$. The displacement rank α of these matrices if 6, resp. $n + 1$.

We first examine the results for general power series. Our experiments showed that the runtime grows predictably with respect to the input bit-size; as a result, we fix the input coefficients to be 10 bit integers, so the number of lifting steps depends just on n.



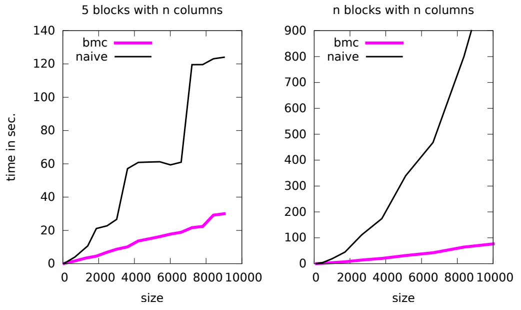
Matrices are generated so as to have 1 less row than they have columns; since the inputs are random, they have nullspace of dimension 1 (we also generated instances with nullity up to 10; using our heuristic to reduce nullity, runtimes were almost indistinguishable). The primes we use have 59 to 60 bits. For $\mathsf{DAC}_{\mathbb{Q}}$ and Newton iteration, the sharp increases indicate an additional lifting step.

Newton iteration is slower than $\mathsf{DAC}_{\mathbb{Q}}$, especially when the number of block grows (since its runtime is quadratic in the displacement rank α). Newton iteration should theoretically be competitive with $\mathsf{DAC}_{\mathbb{Q}}$ when the number of blocks is fixed, and the size (and thus the output bit size) grow, since its theoretical runtime is better by a $\log(t)$ factor, where t is essentially the output bit size. However, this is not noticeable on our experiments: in practice, the integer multiplication function $\mathscr{I}(d)$ grows like $d^{1+\varepsilon}$, for some $\varepsilon > 0$; in that case, the analysis of $\mathrm{DAC}_{\mathbb{Q}}$ can be refined to $O(\alpha\mathscr{M}(n)\mathscr{I}(t))$.

CRT, on the other hand, seems to be competitive with $\mathrm{DAC}_{\mathbb{Q}}$ when α is small but is significantly worse as α grows: the runtime CRT is not linear in α, while $\mathrm{DAC}_{\mathbb{Q}}$ is.

Next, we examine the computation of algebraic approximants; on the basis of the previous experiments, we consider algorithm $\mathrm{DAC}_{\mathbb{Q}}$ only. We start by generating a bivariate polynomial $P(x,y)$ and compute one of its power series solutions f. Since we expect the coefficients of P to be smaller than the coefficients of f, we can better control the behavior of the algorithms by choosing the bit size of P (here, it was fixed to be 1000 bit integers). We compare algorithm $\mathrm{DAC}_{\mathbb{Q}}$ as in the general case, with $\mathscr{C}_{\mathrm{A}} = O(\alpha\mathscr{M}(n))$, to the improved version using the bivariate modular composition (BMC) algorithm of [38] described above, featuring a lower value for $\mathscr{C}_{\mathrm{A}}$. The latter algorithm uses polynomial matrix multiplication, which we implemented using reduction modulo FFT primes and TFT polynomial multiplication.

As a result, we can see a significant difference between the two algorithms as the size of the matrix grows: the theoretical speed-up predicted in Subsection 4.2 is observed in practice.



The divide-and-conquer lifting algorithm turned out to be the most efficient method in all our experiments, especially as it can take extra structure into account, as in the case of algebraic approximants.

REFERENCES

[1] A. Arnold and É. Schost. A Truncated Fourier Transform middle product. *ACM Commun. Comput. Algebra*, 48(3/4):98–99, 2015.

[2] J. Berthomieu and R. Lebreton. Relaxed p-adic Hensel lifting for algebraic systems. In *ISSAC'12*, pages 59–66. ACM, 2012.

[3] 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.

[4] L. I. Bluestein. A linear filtering approach to the computation of the discrete Fourier transform. *IEEE Trans. Electroacoustics*, AU-18:451–455, 1970.

[5] A. Bostan, M. F. I. Chowdhury, R. Lebreton, B. Salvy, and É. Schost. Power series solutions of singular (q)-differential equations. In *ISSAC'12*, pages 107–114. ACM, 2012.

[6] A. Bostan, F. Chyzak, M. Giusti, R. Lebreton, G. Lecerf, B. Salvy, and É. Schost. Algorithmes efficaces en calcul formel. hal.archives-ouvertes.fr/AECF/, 2017.

[7] A. Bostan, F. Chyzak, F. Ollivier, B. Salvy, S. Sedoglavic, and É. Schost. Fast computation of power series solutions of systems of differential equations. In *Symposium on Discrete Algorithms, SODA'07*, pages 1012–1021. ACM-SIAM, 2007.

[8] A. Bostan, C.-P. Jeannerod, C. Mouilleron, and É. Schost. On matrices with displacement structure: generalized operators and faster algorithms. 2016.

[9] A. Bostan, C.-P. Jeannerod, and É. Schost. Solving structured linear systems with large displacement rank. *Theor. Comput. Sci.*, 407(1):155–181, 2008.

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

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

[12] A. Bostan and É Schost. Fast algorithms for differential equations in positive characteristic. In *ISSAC'09*, pages 47–54. ACM, 2009.

[13] B. Boyer and J.-G. Dumas. Matrix multiplication over word-size modular rings using approximate formulas. *ACM TOMS*, 42(3):20:1–20:12, 2016.

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

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

[16] Z. Chen and V. Y. Pan. An efficient solution for Cauchy-like systems of equations. *Computers Math. Applic.*, 48:529–537, 2004.

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

[18] J. Dixon. Exact solution of linear equations using p-adic expansions. *Numer. Math.*, 40:137–141, 1982.

[19] J.-G. Dumas, P. Giorgi, and C. Pernet. Dense linear algebra over word-size prime fields: The FFLAS and FFPACK packages. *ACM TOMS*, 35(3):19:1–19:42, 2008.

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

[21] FFLAS-FFPACK-Team. *FFLAS-FFPACK: Finite Field Linear Algebra Subroutines / Package*, v2.2.2 edition, 2016. http://github.com/linbox-team/fflas-ffpack.

[22] M. Fürer. Faster Integer Multiplication. In *STOC'07*, pages 57–66, 2007.

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

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

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

[26] G. Heinig and T. Amdeberhan. On the inverses of Hankel and Toeplitz mosaic matrices. In *Seminar Analysis (Berlin, 1987/1988)*, pages 53–65. 1988.

[27] T. Huckle. Implementation of a superfast algorithm for symmetric positive definite linear equations of displacement rank 2. volume 2296 of *Proceedings of the SPIE*, pages 494–503. 1994.

[28] O. H Ibarra, S. Moran, and R. Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *J. Algorithms*, 3(1):45–56, 1982.

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

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

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

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

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

[34] R. Lebreton. *Contributions to relaxed algorithms and polynomial system solving*. PhD thesis, École Polytechnique, Palaiseau, France, 2012.

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

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

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

[38] M. Nüsken and M. Ziegler. Fast multipoint evaluation of bivariate polynomials. In *ESA 2004*, number 3222 in LNCS, pages 544–555. Springer, 2004.

[39] V. Olshevsky and V. Y. Pan. A unified superfast algorithm for boundary rational tangential interpolation problems and for inversion and factorization of dense structured matrices. In *Proc. 39th IEEE FOCS*, pages 192–201, 1998.

[40] V. Y. Pan. On computations with dense structured matrices. *Math. Comp.*, 55(191):179–190, 1990.

[41] V. Y. Pan. Parametrization of Newton's iteration for computations with structured matrices and applications. *Computers Math. Applic.*, 24(3):61–75, 1992.

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

[43] V. Y. Pan, B. Murphy, and R. E. Rosholt. Nearly optimal symbolic-numerical algorithms for structured integer matrices and polynomials. In *SNC'09*, pages 105–114. ACM, 2009.

[44] V. Y. Pan, Brian J. Murphy, and R. E. Rosholt. Unified nearly optimal algorithms for structured integer matrices. In *Numerical Methods for Structured Matrices and Applications*, pages 359–375. Birkhäuser, 2010.

[45] V. Y. Pan and A. Zheng. Superfast algorithms for Cauchy-like matrix computations and extensions. *Linear Algebra Appl.*, 310:83–108, 2000.

[46] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.

[47] H. Sexton, M. Shensa, and J. Speiser. Remark on a displacement-rank inversion method for Toeplitz systems. *Linear Algebra Appl.*, 45:127–130, 1982.

[48] V. Shoup. NTL: A library for doing number theory. `http://www.shoup.net`.

[49] V. Shoup. A new polynomial factorization algorithm and its implementation. *J. Symb. Comp.*, 20(4):363–397, 1995.

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