Exact computations with quasiseparable matrices and polynomial matrices with a displacement structure

Hippolyte Signargout

THESE

en vue de l'obtention du grade de Docteur, délivré par

l'ECOLE NORMALE SUPERIEURE DE LYON

Ecole Doctorale N° 512
École doctorale en Informatique et Mathématiques de Lyon

Discipline : Informatique

Soutenue publiquement le 05/10/2023, par :

Hippolyte Signargout

Calcul exact avec des matrices quasiséparables et des matrices polynomiales à structure de déplacement

---

Devant le jury composé de :

BOITO Paula	Professeure	Università Di Pisa	Rapporteure
BOSTAN Alin	Directeur de recherche	Université Paris-Saclay	Rapporteur
LECERF Grégoire	Directeur de recherche	École Polytechnique	Examinateur
GUERRINI Eleonora	Maître de conférences	Université Montpellier	Examinatrice
PERNET Clément	Professeur des universités	Grenoble INP – UGA	Codirecteur
VILLARD Gilles	Directeur de recherche	ÉNS de Lyon	Directeur
THOMASSÉ Stéphan	Professeur des universités	ÉNS de Lyon	Examinateur

Exact computations with quasiseparable matrices and polynomial matrices with a displacement structure

Hippolyte Signargout

October 2020 - September 2023

Contents

1 Introduction 1
1.1 Exact structured linear algebra 1
1.1.1 Exact linear algebra 1
1.1.2 Models of computation 2
1.1.3 Structured linear algebra 4
1.2 Computing the determinant of polynomial matrices with a displacement structure 5
1.2.1 The displacement structures we consider 5
1.2.2 Computing with polynomial structured matrices 7
1.2.3 Techniques used for the computation of the determinant of polynomial structured matrices 8
1.2.4 Contributions in computing terms of the series expansion 11
1.2.5 Possible improvements for the computation of structured polynomial determinants 13
1.3 Comparative study of existing formats for quasiseparable matrices 14
1.3.1 Quasiseparable matrices and rank-revealing factorisations of submatrices 15
1.3.2 Efficient storage formats for numerical quasiseparable matrices . . . 16
1.3.3 Quasiseparable matrices in exact linear algebra 17
1.3.4 Contributions and comparisons 17
1.3.5 Further improvements for Bruhat and SSS 19

2 Computing the Characteristic Polynomial of Generic Toeplitz-like and Hankel-like Matrices 21
2.1 Introduction 21
2.2 Rank displacement structures 23
2.2.1 Product of Structured Matrices 24
2.2.2 Reconstruction of a Toeplitz+Hankel-like matrix from its generators 26
2.3 Matrix Polynomials 27
2.4 A randomised algorithm with the baby-steps/giant-steps method 28

2.4.1 Description of the algorithm . . . 28
2.4.2 Detailed cost analysis . . . 29
2.5 Using structured inversion . . . 31
2.5.1 Generic Toeplitz-like Matrices . . . 32
2.5.2 Generic Toeplitz+Hankel-like Matrices . . . 34
2.6 Special matrices for genericity . . . 35
2.6.1 A Toeplitz Point . . . 37
2.6.2 A Hankel Point . . . 38

3 High-order lifting for polynomial Sylvester matrices 39
3.1 Introduction . . . 39
3.1.1 Tools from previous works . . . 41
3.1.2 Overview of the contribution . . . 45
3.1.3 Related questions: resultants, characteristic polynomials and bivariate ideals . . . 47
3.1.4 Model of computation and notations . . . 48
3.2 Baby steps/giant steps for high-order lifting . . . 49
3.2.1 High-order lifting . . . 49
3.2.2 Baby steps/giant steps . . . 53
3.3 Matrices with a displacement structure . . . 55
3.4 Displacement structure of Sylvester matrices and its residues and high-order components . . . 58
3.4.1 Sylvester matrices over $\mathbb{K}$ . . . 59
3.4.2 Structure of high-order components . . . 61
3.4.3 Structure of residues . . . 63
3.5 Structured middle and truncated products . . . 67
3.5.1 Middle product: computation of the right generator . . . 68
3.5.2 High-order components . . . 71
3.6 Giant steps . . . 72
3.6.1 Concatenated middle and truncated products . . . 72
3.6.2 Cost bound for the giant steps . . . 75
3.7 Complete expansion algorithm . . . 75
3.8 Resultant algorithm . . . 79
3.8.1 Matrix fraction reconstruction . . . 79
3.8.2 Resultant algorithm . . . 81

4 Exact computations with quasiseparable matrices 85
4.1 Introduction . . . 85

4.1.1 Rank revealing factorisations . . . 86
4.1.2 Contributions . . . 86
4.2 Presentation of the formats . . . 87
4.2.1 SSS generators . . . 87
4.2.2 HSS generators . . . 89
4.2.3 Bruhat generators . . . 90
4.3 Construction of the generators . . . 91
4.3.1 SSS generator from a dense matrix . . . 91
4.3.2 HSS generator from a dense matrix . . . 92
4.3.3 Bruhat generator from a dense matrix . . . 94
4.3.4 Bruhat generator from a sparse matrix . . . 96
4.3.5 Experimental comparison . . . 98
4.4 Application to a block vector . . . 99
4.4.1 SSS × dense . . . 99
4.4.2 Bruhat × dense . . . 101
4.4.3 Experimental comparison . . . 101
4.5 Sum of quasiseparable matrices . . . 103
4.5.1 SSS sum . . . 103
4.5.2 Bruhat sum . . . 104
4.6 Product in SSS . . . 106

Bibliography 109

Chapter 1 | Introduction

Algorithms in linear algebra are tools that can be used in a large spectrum of applied fields, from engineering to computational mathematics. Techniques vary depending on the nature of the elements that are handled. This thesis focuses on exact computations, in which all digits of each number have the same importance and all need to be correct. More precisely, we study algorithms which handle matrices with a structure which allows for easier storage and faster operations. We present new fast algorithms in structured exact linear algebra, specifically algorithms for quasiseparable matrices in different formats, and algorithms for the computation of the determinant of polynomial matrices with a displacement structure.

1.1 Exact structured linear algebra

We detail in this section the framework to which this thesis contributes. We define exact linear algebra and compare the field to numerical analysis, explain our choices for cost analysis and present the structures of the matrices our algorithms handle.

1.1.1 Exact linear algebra

The study of algorithms can be seen as a goal in itself for a better understanding of mathematical structures, or as a tool for scientific computations. Depending on the domain for which computations will be done, algorithms in linear algebra can be separated into two types. The first one is algorithms designed for *numerical analysis*. They work over approximations of real and complex numbers, which can be represented in the machine with floating-point arithmetic. A strong focus needs to be maintained on avoiding numerical instability in order to recover approximations with precision close to the storage precision.

On the other hand, *computer algebra* is used when the numbers that are handled need to be exact and approximations would make no sense. This can be the case for example with integers, rationals and finite fields. Exact algorithms are usually slower than approx-

imated algorithms for use in physical and engineering applications. However, they have direct applications in other fields, namely cryptology and computational mathematics (*e.g.* number theory, algebraic topology or graph theory). Additionally, no special care needs to be taken to avoid numerical instability. This allows better partitioning of data and fast methods which are harder to use with approximations.

The framework of this thesis is that of exact linear algebra. Our work is based both on tools specifically designed for this framework and on algorithms designed in a numerical context, that we adapt and analyse under this perspective.

1.1.2 Models of computation

The theoretical cost bounds given in this thesis are for the number of arithmetic operations. The matrices and polynomials we handle are defined over a field $\mathbb{K}$ and we count the number of additions, subtractions, multiplications and divisions over $\mathbb{K}$. Note that this may not be suited for algorithms over integers or rationals, for which the time spent on an operation depends on the bitsize of the input. Computing exact values does not mean counting the exact number of operations and the cost analysis of algorithms can be done at different granularities, depending on how close the costs are to that of competitive algorithm and how precise we want to be.

Most of the theoretical cost bounds in algorithm analysis are asymptotic and given with the big O notation. Recall $f(n) = O(g(n))$ if $f(n)/g(n)$ is bounded when n tends to infinity. Sometimes, when the given cost bounds show an improvement in the exponent in n for a polynomial cost, the *soft-O* notation is used to avoid handling logarithmic factors which are negligible compared to the exponent improvement. We say $f(n) = \tilde{O}\left(g(n)\right)$ if there exists k for which $f(n) = O\Big(g(n)(\log n)^k\Big)$. A good asymptotic cost bound is often an indicator of practical efficiency: consider that if n operations are done in a second on a modern laptop (this is very roughly taking n equal to a billion), doing n^2 operations can take centuries. Compensating such a gain with a multiplicative constant hidden with the O notation is uncommon, yet it happens. Algorithms have been developed which are said to be *galactic* as even though their asymptotic cost is better than practical algorithms, they would be faster in practice only for inputs of size larger than the number of atoms in the universe (see for example [65]).

Moreover, when multiple algorithms share the same asymptotic cost, the next step in the comparison is to get the leading constants of the cost, that is the multiplicative constant of the term in the big O, hidden by this notation.

We do not give more precise theoretical cost bound estimates. However, we back up some of our analyses with practical experiments. While extrapolating from experimental timings may be less trustworthy than from theoretical cost bounds, they may reveal costs

that are not captured by the cost with leading constant, such as non-arithmetic operations, cache misses or even simply a non-negligible second term in the cost expression.

The costs of our algorithms depend on the cost of subroutines that we use as building blocks. One of these blocks is polynomial multiplication. The product of two polynomials of degree n on any field can be computed in $O(n \log n \log\log n)$ by [10]. As we only use this bound within *soft-O* complexity analysis, we simply consider the cost of polynomial multiplication to be $\tilde{O}(n)$. Algorithms achieving this soft bound are used in practice.

The case of square matrix multiplication is slightly more complex. In the analysis of our algorithms we consider that the product of two $n \times n$ matrices is computed in $O(n^\omega)$, where $2 < \omega \leq 3$ depends on the algorithm used for the computation. The lowest ω one can use as of today is that given by the algorithm of [18], which is lower than 2.38. Restricting to algorithms used in practice one can take $2.79 < \omega \leq 3$, where the lower bound is achieved by the algorithm of Strassen [19, 78]. Even when we consider that in practice matrix multiplications will be done on blocks of dimensions too small to use algorithms with subcubic costs, reducing our algorithms to matrix products is a step towards practical efficiency. First, the most efficient matrix multiplication algorithms can be used for each purpose, but also reducing to matrix blocks is more efficient in terms of memory access.

We choose to use square matrix multiplication even for matrices that are not square: the product of an $l \times m$ matrix by an $m \times n$ matrix can be computed in

$$O\left(lmn\gamma^{\omega-3}\right) \tag{1.1}$$

operations with $\gamma = \min(l, m, n)$ by dividing both matrices in blocks of size $\gamma \times \gamma$. There exist asymptotically faster algorithms for computing rectangular products [55] which could be used to lower further the asymptotic and practical costs of our algorithms. This would however be achieved only by revisiting all the techniques our work is based on and integrating the possibility of using the fastest rectangular product algorithms in the intermediate algorithms that we use as subroutines, which is out of the scope of this thesis.

We also use rank-revealing factorisations as subroutines. The algorithms which use this building block are correct for multiple factorisations, most of which can be achieved in the asymptotic time of matrix multiplication but with different leading constants. We give cost bounds depending on the leading constant of the factorisation that is used. An overview of existing constants is given in [71].

1.1.3 Structured linear algebra

Matrices can be represented in multiple ways in linear algebra. The most basic is the dense representation, where an $m \times n$ matrix is stored as its nm entries. When most of the entries of a matrix are zeros, a sparse representation can be used in which each nonzero entry is stored alongside its row and column indices.

Matrices that arise in applications may show a structure that is not seen by the amount of zeros. This thesis deals with structured matrices which can be uniquely generated by relations between a number of field elements which is lower than the number of entries. Although basic, the example of low-rank matrices is typical as the structures we deal with depend on low ranks. An $m \times n$ matrix A of rank r can be stored with $r(m+n)$ field elements as r independent columns and n linear relations between them to generate the remaining columns. As a generalisation we use a *rank-revealing factorisation*, that is a set of matrices of dimensions depending on m, n and r whose product is equal to A. A basic example is a pair of matrices $L \in \mathbb{K}^{m\times r}, R \in \mathbb{K}^{r\times n}$ such that $LR = A$. Multiple types of factorisations exist with different numbers of matrices, dimensions and constraints. For example an LU factorisation is a pair $L \in \mathbb{K}^{m\times r}, U \in \mathbb{K}^{r\times n}$ such that $LU = A$, L is lower-triangular and U is upper-triangular. It exists only if A has generic rank profile (all its principal minors are nonzero) and in general a PLUQ factorisation exists: the matrices P and Q are permutations of the rows of L and columns of U.

It is useful to consider classes of structured matrices that generalise a simple structure and are stable through multiple operations. This is the case of quasiseparable matrices, which are defined as matrices whose submatrices strictly over or under the main diagonal have low rank. The class includes band and low-rank matrices but also their inverses, their sums and products and inverses of these sums.

Often, storages for structured matrices use rank-revealing factorisations to store information specific to the matrix. For example, a Toeplitz matrix, whose entries are equal along each diagonal, can be transformed by a specific linear operator to get a low-rank matrix. It is one of the multiple classes of displacement structures, for which matrices are stored as a rank-revealing factorisation of their image through a linear operator.

Note that a field of linear algebra treats structured matrices as black-boxes. Considering the application of the matrix to a vector as the only given operation and that it can be done fast, black-box algorithms are designed to be used on any type of sparse or structured matrices. In an orthogonal direction, we focus on the specific structure of the matrices we handle and design algorithms which use other properties than fast matrix-vector products to their advantage.

1.2 Computing the determinant of polynomial matrices with a displacement structure

Displacement structures are characterised by rank constraints on transformations of matrices. When the dimension is n and the rank of the transformation is α, a structured matrix can be stored with $O(n\alpha)$ elements. When matrices are considered on a field, most operations can be done with a dependence in n for their cost within a logarithmic factor from the storage size. When dealing with matrices with polynomial entries of degree at most d, the techniques used for computing products can be directly adapted to get costs quasi-linear in n and d. However, directly using on polynomial entries the most efficient techniques used to compute determinants induce the computation of fractions whose degrees may become too high to guarantee such costs.

We explore the computation of the determinant for specific polynomial structured matrices. We first detail the displacement structures which we will consider and the behaviour of matrices benefitting from this structure over a field. We then cover the existing methods used for the computation of determinants of polynomial structured matrices. We finally present our contributions for the computation of the characteristic polynomial of Toeplitz-like and Hankel-like matrices and of the resultant of bivariate polynomials that we detail in Chapters 2 and 3.

1.2.1 The displacement structures we consider

We will introduce the general displacement rank theory with the example of Toeplitz matrices. An $m \times n$ Toeplitz matrix is a matrix whose entries are equal along each diagonal:

$$T = \begin{bmatrix} t_0 & t_{-1} & \cdots & t_{1-n} \\ t_1 & t_0 & \ddots & \\ \vdots & \ddots & \ddots & t_{-1} \\ t_{m-1} & & t_1 & t_0 \end{bmatrix}. \tag{1.2}$$

T can be stored with $m+n-1$ field elements, but it would not be clear how to use this storage for operations other than applying T to a vector or adding T to another Toeplitz matrix. Now for $Z_m = \begin{bmatrix} 0 & & & \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix} \in \mathbb{K}^{m\times m}$ and $^\mathsf{T}$ the transposition symbol, consider the

matrix

$$\tau = T - Z_m T {Z_n}^\mathsf{T} = \begin{bmatrix} t_0 & t_{-1} & \cdots & t_{1-n} \\ t_1 & & & \\ \vdots & & & \\ t_{m-1} & & & \end{bmatrix}. \tag{1.3}$$

The rank of τ is at most 2 and T can be stored as a rank-revealing factorisation of τ with $2(m+n)$ field elements. The class of Toeplitz-like matrices can be defined as the matrices whose images through the operator $\Delta_{Z_m,{Z_n}^\mathsf{T}} : T \mapsto T - Z_m T {Z_n}^\mathsf{T}$ has low rank [42] (note that the same class can be defined with other operators). This class includes the inverse, sums and products of Toeplitz matrices, which allows the use of this representation for a large panel of operations. The operator $\Delta_{Z_m,{Z_n}^\mathsf{T}}$ is called a *displacement operator*, the rank α of τ is the *displacement rank* of T and a pair of matrices $G \in \mathbb{K}^{m\times\alpha}, H \in \mathbb{K}^{n\times\alpha}$ such that $\tau = GH^\mathsf{T}$ is called a *generator* for T.

Taking $\Delta_{Z_m,{Z_n}^\mathsf{T}}(A) = GH^\mathsf{T}$ as an equation in A it can be shown [66, Example 4.4.1] that we get the unique solution

$$A = \sum_{i=1}^{\alpha} L_i U_i \tag{1.4}$$

where L_i is the Toeplitz lower triangular matrix with $G_{*,i}$ as its first column, and U_i is the Toeplitz upper triangular matrix with ${H_{*,i}}^\mathsf{T}$ as its first row. Applying a Toeplitz matrix to a vector is equivalent to computing a polynomial product, at a cost almost linear in the dimension of the vector (see Section 1.1.2). The link between Toeplitz matrices and polynomials can easily be seen for a lower triangular Toeplitz matrix: consider the application that multiplies a polynomial in the ring $\mathbb{K}[y]/(y^n)$ by $\sum_{i=0}^{n-1} l^{(i)} y^i$. The matrix $L = \begin{bmatrix} l^{(0)} & & \\ \vdots & \ddots & \\ l^{(n-1)} & \ldots & l^{(0)} \end{bmatrix}$ is the matrix of this application in the basis $(1, \ldots, y^{n-1})$. By permuting the entries of the vector, the same is true for an upper-triangular Toeplitz matrix and the product of a Toeplitz-like matrix represented by a generator of length α is then reduced to 2α polynomial products. We thus get that the cost of applying a Topelitz-like matrix to a vector is $\tilde{O}(n\alpha)$. The product of a Toeplitz-like matrix by an $n \times t$ dense matrix can thus be done in $\tilde{O}(n\alpha t)$; introducing fast matrix arithmetic [8, 9], a cost of $\tilde{O}(n\alpha t^{\omega-2})$ can be guaranteed when $t = O(\alpha)$.

Operations which take Toeplitz-like matrices as input and output thus generally work with generators, as the cost of these operations can be lower than the storage size of a dense matrix. Namely, computing a generator for the inverse of a Toeplitz-like matrix and for the product of two matrices of displacement rank α can be done in $\tilde{O}(n\alpha^{\omega-1})$ [8]. When needed, the dense form of a matrix can be recovered from a generator in $\tilde{O}(nm\alpha)$ using Eq. (1.4).

Specific examples of Toeplitz-like matrices include matrices of modular multiplication:

for any $f \in \mathbb{K}[y]$ of degree n and $a \in \mathbb{K}[y]/(f(y))$, the $n \times n$ matrix of the application that multiplies a polynomial by a modulo f in the basis $(1, \ldots, y^{n-1})$. We will also be particularly interested in Sylvester matrices which are the specific block Toeplitz matrices defined by two polynomials $p(y) = \sum_{i=0}^{n_p} p^{(i)} y^i$ and $q(y) = \sum_{i=0}^{n_q} q^{(i)} y^i$ as

$$S = \begin{bmatrix} p^{(n_p)} & & & q^{(n_q)} & & \\ p^{(n_p-1)} & \ddots & & q^{(n_q-1)} & \ddots & \\ \vdots & & p^{(n_p)} & \vdots & & q^{(n_q)} \\ p^{(0)} & & p^{(n_p-1)} & q^{(0)} & & q^{(n_q-1)} \\ & \ddots & \vdots & & \ddots & \vdots \\ & & p^{(0)} & & & q^{(0)} \end{bmatrix}. \tag{1.5}$$

It can be noticed that if $p^{(0)} \neq 0$, the Sylvester matrix defined by p and q is the matrix of multiplication by the mirror $\bar{p}(y) = y^{n_p} p(1/y)$ of p modulo the polynomial f such that $y^{n_q}\bar{p}(y) = \bar{q}(y) \mod f(y)$.

Toeplitz-like matrices are not the only class of displacement structures. Those are defined as the matrices which have low rank through a displacement operator $\Delta_{M,N} : A \mapsto A - MAN$ or $\nabla_{M,N} : A \mapsto MA - AN$. We will also focus on Hankel matrices which are matrices whose entries are equal along each anti-diagonal. The class is generalised to Hankel-like matrices by the operator $\nabla_{Z_m, Z_{n,1}}$, where $Z_{n,1} = Z_n + e_1 e_n{}^{\mathsf{T}}$ and e_i is the i-th canonical vector of appropriate dimension. The sum of a Hankel matrix and a Toeplitz matrix (for example the characteristic matrix of a Hankel matrix) is a Toeplitz+Hankel matrix [31] and the generalisation to Toeplitz+Hankel-like matrices is done by the operator ∇_{U_m, U_n}, where $U_n = Z_n + Z_n{}^{\mathsf{T}}$ [66].

While Hankel-like matrices behave mainly in the same way as Toeplitz-like matrices, Toeplitz+Hankel-like matrices show a particularity in the fact that the operator ∇_{U_m,U_n} is singular and a matrix can not be recovered directly from its generator. As detailed in Section 2.2.2, an irregularity set such as the first column of the matrix needs to be given for unique reconstruction, which introduces an additional difficulty to algorithms handling this class of matrices.

1.2.2 Computing with polynomial structured matrices

When the entries of a matrix with a displacement structure are polynomial, its structure can be linked to bivariate polynomials. For example the determinant of a polynomial Sylvester matrix defines the resultant of two bivariate polynomials, and a matrix of multiplication in a bivariate ideal is a univariate polynomial Toeplitz-like matrix. In this section we will consider matrices with constant displacement rank, as in the two examples

above. For an $n \times n$ matrix of degree d, its storage size as a generator is $O(nd)$ and the cost of multiplication, with a dense or structured matrix, remains within logarithmic factors of this cost using the simple technique presented in the previous section. On the other hand, the algorithms used for matrices over a field that need inversion of elements, such as those which compute the determinant, can not be directly applied to polynomial matrices for a quasi-linear cost. The difficulty lies in the fact that the inverse of a polynomial entry has to be seen in the field of fractions and unlike polynomials, the degrees of the fractions increase in sums and may become high.

Consider the determinant of such a matrix. It is a polynomial of degree and storage size nd but in most cases the best option to compute it is to use evaluation-interpolation, for a cost quasi-quadratic in the dimension of the matrix, $\tilde{O}(n^2 d)$. Two improvements over this cost have been obtained recently for specific determinants. In the case of a Sylvester matrix, the computation of the determinant, or equivalently of the resultant in y of two generic bivariate polynomials of degrees n in y and d in x, can be done in $\tilde{O}\left(n^{2-1/\omega} d\right)$ field operations by the algorithm of [83] on a generic input. By generic we mean here, and in the rest of this thesis, that the algorithm is correct for an input outside a hypersurface of the input space. Using a bit complexity model, this problem can generically be solved in quasi-linear time with the algorithms of [35,84] but the techniques are not directly adaptable to general fields. The second improvement is given in the case of the characteristic matrix of a matrix of modular multiplication. The computation of its determinant, or equivalently of the characteristic polynomial in a bivariate ideal, costs $\tilde{O}(n^{1.43})$ with the generic algorithm of [63] (the degree of the matrix in this case is $d = 1$).

The results we present in this thesis build upon the algorithms of [63,83] in two different directions. We first adapt both algorithms for computing the characteristic polynomial of larger classes of matrices in Chapter 2. We then dive deeper in the structure of polynomial Sylvester matrices to propose an adaptation of the second algorithm which improves the bound on the complexity of computing bivariate resultants in Chapter 3.

The following sections detail the techniques we use in our algorithms (Section 1.2.3) and how we combine and adapt them to our purposes (Section 1.2.4).

1.2.3 Techniques used for the computation of the determinant of polynomial structured matrices

Our work builds upon previous improvements in the computation of polynomials structured determinants, and related tools. The main framework is given by the block projections method of [47], inspired by Coppersmith's block-Wiedemann method [15]. It reduces the computation of a characteristic polynomial to the computation of the determinant of a smaller matrix of higher degree. This matrix is the denominator of a fraction that is

reconstructed from the first terms of its series expansion. The terms of this series can be computed either by inversion in a modular ring, the baby-steps/giant-steps method, or high-order lifting.

One of the main ideas behind the cost improvements of [47] to compute the determinant of $M(x) = x\mathrm{I}_n - T$ for $T \in \mathbb{K}^{n\times n}$ (the notation I_n is for the identity matrix of dimension n) is to project the inverse of the input matrix on the left and the right in order to compute the result as the determinant of a smaller matrix. It is shown in [47, Thm. 2.12] that there exist projections $U, V \in \mathbb{K}^{n\times m}$ and an irreducible fraction $N(x)D^{-1}(x) = U^{\mathsf{T}}M(x)^{-1}V \in \mathbb{K}^{m\times m}(x)$ such that the first m invariant factors of M are the same than that of D. The dense matrix $D(x)$ can be reconstructed from enough terms of the series expansion $U^{\mathsf{T}}M(x)^{-1}V = \sum_{i\geq 0} C_i x^i$. If $M(x)$ has m or fewer non-trivial invariant factors then the determinant of $D(x)$ is that of $M(x)$ up to a multiplicative constant. Another interpretation of this method for any polynomial matrix $M(x) \in \mathbb{K}[x]^{n\times n}$, given in [83], distinguishes three steps:

1. Compute enough terms of the series expansion of $U^{\mathsf{T}}M(x)^{-1}V \in \mathbb{K}^{m\times m}[[x]]$;
2. From these terms reconstruct the fraction $N(x)D(x)^{-1} = U^{\mathsf{T}}M(x)^{-1}V \in \mathbb{K}^{m\times m}(x)$;
3. Compute the determinant of $D(x)$ and the multiplicative constant $\det M(a)/\det D(a)$ for some $a \in \mathbb{K}$.

Item 2 can be done with Padé approximation, for example using σ-bases [26], and Item 3 is done with fast arithmetic on dense polynomial matrices [52]. Our contributions focus on computing the terms of the series in Item 1. We detail the existing methods for the computation of these terms in the following.

The method chosen in [83] for the computation of λ terms of the series is to simply see the truncated series $U^{\mathsf{T}}M(x)^{-1}V \mod x^\lambda$ as a polynomial matrix over the ring $\mathbb{K}[x]/(x^\lambda)$. The terms of the series to be computed are then the coefficients of the polynomial matrix, which are given by successively computing the inverse of $M(x) \in \mathbb{K}[x]/(x^\lambda)$ (under the condition $\det M(0) \neq 0$) and applying the projections U and V. The projections need to be chosen themselves structured instead of random to avoid a dominant cost for dense matrix products but it is shown that the following steps remain correct for generic input matrices.

When $M(x)$ is the characteristic matrix of a matrix T, the series $U^{\mathsf{T}}M(x)^{-1}V$ has the following special shapes for expansions at zero (if T is invertible) and infinity:

$$U^{\mathsf{T}}(x\mathrm{I}_n - T)^{-1}V = -\sum_{k\geq 0} U^{\mathsf{T}}T^{-k-1}Vx^k \tag{1.6}$$

$$= \sum_{k\geq 0} U^{\mathsf{T}}T^{k}Vx^{-k-1}. \tag{1.7}$$

Item 1 is then the computation of λ powers of T or T^{-1} projected on U^T and V. The baby-steps/giant-steps strategy used in [47] consists of four steps for r, s such that $rs \geq \lambda$ (the following steps detail the computation of projected powers of T):

1. Successively apply T on the right to U^T to compute the $U^\mathsf{T}T^i$ for $0 \leq i < r$;
2. Compute $R = T^r$, for example with fast exponentiation;
3. Successively apply R on the left to V to compute the $T^{rj}V$ for $0 \leq j < s$;
4. Compute all the products between the $U^\mathsf{T}T^i$ and the $T^{rj}V$ to get $U^\mathsf{T}T^{i+rj}V$ for $i < r$ and $j < s$.

This approach is used in [63] where its efficiency relies on the fast multiplication algorithms that are available for structured matrices in the four steps above and on the use of structured projections as in [83].

Another technique that is used to compute terms of a series and determinants in dense linear algebra is the high-order lifting method of [77]. It deals with the inverse of a general polynomial matrix $M(x) \in \mathbb{K}[x]^{n\times n}$ of degree d projected only to the right (similarly, take $U = \mathrm{I}_n$), with projection V polynomial of degree $< d$. The expansion of $M^{-1}V$ is computed by slices of degree d as a z-adic expansion with $z \in \mathbb{K}[x]$ of degree d such that $\gcd(\det M, z) = 1$.

The principle is the following. Consider the first k terms of the z-adic expansion of $M^{-1}V$ are already computed. Let R_k be the residue at order k defined by

$$M\left(M^{-1}V \mod z^k\right) = V - z^k R_k. \tag{1.8}$$

Notice that R_k has degree at most $d-1$ and can be computed from only the last term of the z-adic truncated expansion. Now by rearranging Eq. (1.8) as

$$M^{-1}V = \left(M^{-1}V \mod z^k\right) + M^{-1}R_k z^k \tag{1.9}$$

we see that the i-th term of $M^{-1}R_k$ is the $(k+i)$-th term of $M^{-1}V$. Taking $V = \mathrm{I}_n$, the $(k+i+1)$-th term of $M^{-1} = \sum_{k\geq 0} C_k z^k$ can be computed as

$$a + C_{k+i+1}z + bz^2 = (C_i + zC_{i+1})R_k \tag{1.10}$$

with a and b of degree $< d$. We hence have a way to compute from a term of the series a residue at the same order, and from these a term of the series at a higher order. In order to compute a full truncated series, residues at order that are powers of 2 are computed

first, before the rest is given by successively applying, for increasing i, the operation

$$\begin{bmatrix} C_0 & \cdots & C_{2^i-1} \end{bmatrix} R_{2^i} = \begin{bmatrix} C_0 & \cdots & C_{2^{i+1}-1} \end{bmatrix}. \tag{1.11}$$

One can already notice the similarity between the operation which computes a residue at order $k+i$ from a residue at order k and an expansion term at order i and the product of two powers of a matrix at orders k and i which results in the matrix at power $k+i$. The method described just above can be seen as fast exponentiation followed by Keller-Gehrig expansion [51]. We will use this similarity to replace power products in the baby-steps/giant-steps method when no matrix powers appear, *i.e.* when computing a determinant that is not a characteristic polynomial.

1.2.4 Contributions in computing terms of the series expansion

The techniques presented in the previous sections can be combined to make fast determinant algorithms, as those from which we extract these techniques. To sum up, we have seen that a baby-steps/giant-steps algorithm within the block-Wiedemann scheme was given in [47]; another interpretation of the scheme for the determinant of polynomial Sylvester matrices is given in [83] with the use of structured projections and the structured projections have then been included in the baby-steps/giant-steps strategy to compute a characteristic polynomial in [63].

The determinant algorithms we propose explore further the combinations of these building blocks. They remain in the three-step scheme presented above (computation of the terms of the series; reconstruction of an irreducible fraction; computation of the denominator's determinant) and our contributions focus on the first step, the computation of the terms of the series. We remark however that when using structured projections the computed terms are themselves structured and that the fraction reconstruction tools that we use do not exploit this structure (see Section 1.2.5).

We present three main algorithms, whose cost bounds are represented in Table 1.1.

- Algorithm ToeplitzLikeExpansion and Algorithm THLikeExpansion adapt the method of [83] to compute the terms of the truncated series using direct inversion of the input matrix modulo x^λ and structured projections in the case where it is the characteristic matrix of a Toeplitz-like or Hankel-like matrix, hence broadening the scope of the method. This results in lowering the cost bound for the computation of the characteristic polynomial of such matrices: for generic input matrices, the algorithms give a cost bound that matches that of [83]. If we consider an input matrix that satisfies the requirements of both algorithms (this forces a matrix of degree 1 and displacement rank 2), both achieved cost bounds are $\tilde{O}\left(n^{2-1/\omega}\right)$, and

the best exponent is below 1.58.

- The baby-steps/giant-steps strategy was used in a block-Wiedemann context for the computation of the characteristic polynomial of dense matrices in [47]. It was then combined with structured projections in [63] to achieve for the computation of the characteristic polynomial of generic matrices of modular multiplication the cost bound of $\tilde{O}(n^{1.43})$.

  Trying to apply the same method to general Toeplitz-like or Hankel-like matrices is not as effective. Powers of matrices of modular multiplication remain matrices of modular multiplication and keep a constant displacement rank. On the other hand, powers of general Toeplitz-like and Hankel-like matrices, while remaining Toeplitz-like or Hankel-like, progressively lose their structure and their displacement rank increases with the exponent. The structure is lost too soon in a baby-steps/giant-steps scheme to allow an improvement in the cost.

  However, using dense projections as in [47] allows for a randomised algorithm presented in Algorithm StructuredBSGS for the computation of Toeplitz-like and Hankel-like minimal polynomials with a cost sub-quadratic in the dimension of the matrix. The characteristic polynomial is obtained generically. The improvement compared to [47] is the use of fast algorithms to handle structured matrices.

- We saw that a Sylvester matrix was a matrix of multiplication and hence the characteristic polynomial algorithms of [63, Sec. 10.1] can be used to compute the resultant of two bivariate polynomials if their associated Sylvester matrix is the characteristic matrix of a constant Sylvester matrix. In this special case we get cost bounds of $\tilde{O}(n^{1.43})$ for the computation of the bivariate resultant, or $\tilde{O}(n^{1.46})$ when using the bound of Eq. (1.1) for the cost of rectangular matrix multiplication instead of using the fast algorithms of [55]. With a more general degree 1 matrix the best bound we have is that of [83], $\tilde{O}(n^{1.58})$.

  In Chapter 3 Algorithm StructuredResultant fills this complexity gap (except for the slight improvement given by the use of fast rectangular matrix multiplication) by extending the scope of the baby-steps/giant-steps and structured projections of [63] when the terms of the series are not powers of matrices by using high-order lifting and the special structure of Sylvester matrices. As mentioned in Section 1.2.3 we replace the computation of matrix (or polynomial) powers in the baby-steps/giant-steps with the computation of high-order components and residues, the structure of which needed to be precisely examined to guarantee fast operations.

Table 1.1: Comparison of the asymptotic cost of various methods for computing the determinant of polynomial matrices with displacement structures. All algorithms have genericity conditions on the input. The new algorithms allowing the costs in bold are presented in this thesis, and citations are given for the existing cost bounds. The last column gives the costs when fast rectangular matrix multiplication (FRMM) is used.

	Direct Inversion	Baby Steps Giant Steps	
Modular composition [63] Multiplication matrix ($d = 1$)	-	$\tilde{O}(n^{1.46})$	FRMM: $\tilde{O}(n^{1.43})$
Bivariate resultant Sylvester matrix	$\tilde{O}(n^{1.58}d)$ [83]	$\boldsymbol{\tilde{O}(n^{1.46}d)}$	$\rightarrow$ for $d = O\left(n^{1/3}\right)$
Characteristic polynomial ($d = 1$) Toeplitz/Hankel-like (dsp. rk. α)	$\boldsymbol{\tilde{O}(n^{1.58}\alpha^{1.16})}$	$\boldsymbol{\tilde{O}(n^{1.86}\alpha^{0.52})}$	

1.2.5 Possible improvements for the computation of structured polynomial determinants

Our algorithms compute the expected determinant only for matrices that are generic, *i.e.* matrices whose entries are outside a variety of the input space. This means that they give a correct output when the input parameters (either the entries of the matrix or that of the generator) are not a root of a given polynomial. This polynomial depends on the genericity conditions given for our algorithms.

One genericity condition is shared by all algorithms as it is inherent to the block-Wiedemann method. This is the condition on the number of non-trivial invariant factors. The other conditions are specific to the computation of the terms of the series.

We show that the genericity varieties of our algorithms are not the entire input spaces by showing the genericity polynomial is nonzero. We also give a randomised algorithm using Algorithm StructuredBSGS for the computation of the characteristic polynomial when the input meets the condition on invariant factors. Future improvements could be to try to give probabilistic versions of Algorithm ToeplitzLikeExpansion, Algorithm THLikeExpansion and Algorithm StructuredResultant.

What is needed is a randomised transformation of the input such that the transformed matrix is outside of the genericity hypersurface with nonzero probability and the fetched result can be recovered from the output of the algorithm on the transformed matrix. The first condition is equivalent to the existence of a nonzero polynomial such that the algorithm is correct when the random elements of the transformation are not one of its roots.

Take the schoolbook example of a characteristic polynomial algorithm $\mathcal{A}$ which is

correct only for non-singular matrices. The *genericity* polynomial is the determinant of the matrix, and its variables are the entries of the matrix. Consider the random transformation $f : A \mapsto A + c\mathrm{I}_n$ for c chosen at random in a subset of the input field. For a given A, applying $\mathcal{A}$ on $A + c$ will give the correct result if c is not a root of $P(x) = \det(A + x)$. The characteristic polynomial of χ_A of A can then be recovered from that of $f(A)$, $\chi_{f(A)}$, as $\chi_A(x) = \chi_{f(A)}(x - c)$.

We were not able to find such a transformation, which would at least lower our genericity conditions, despite looking into various transformations, including multiplying by random Toeplitz matrices or adding low-rank Toeplitz-like matrices. However, the algorithm of [63] has been made probabilistic with highly non-trivial techniques and a similar improvement could be possible at least for the resultant if not for the characteristic polynomial of more general Toeplitz-like matrices.

In terms of improving further the cost bound, one can notice that when using structured projections, the terms of the series that are obtained are themselves structured. It intuitively seems inefficient to reconstruct the dense forms of these terms before computing the associated fraction. Improving the cost of reconstructing the fraction from structured terms would improve the overall cost of the computation of the determinant. The resulting fraction denominator could then itself be also structured.

Also, as mentioned in Section 1.1.2, we do not take advantage of fast rectangular matrix multiplication as this would need to revisit the basic routines of the displacement rank theory. It could however lead to an improvement in the cost of the computation of the resultant, and make the cost match that of [63] for $d = 1$.

1.3 Comparative study of existing formats for quasiseparable matrices

From a unique structure definition arise multiple storage methods. Quasiseparable matrices are matrices with rank constraints on their submatrices, which can be stored in space linear in the largest dimension. At least four different storage formats achieve this space bound and allow for operations such as applying a matrix to a vector with cost linear in the storage space. These formats were developed independently, most of them for numerical applications.

We propose in Chapter 4 a comparative study of three storage formats for quasiseparable matrices in exact linear algebra: SSS, HSS and Bruhat. The comparisons are made with detailed asymptotic cost (with leading constants) and practical experiments. In order to pursue this comparison we defined the HSS and SSS formats in an exact linear algebra framework, designed a new algorithm for the computation of a Bruhat generator

and implemented the main kernel routines of the SSS format in the linear algebra library `fflas-ffpack` [27].

1.3.1 Quasiseparable matrices and rank-revealing factorisations of submatrices

The order of quasiseparability s of an $n \times n$ matrix is the maximal rank of its submatrices that are strictly over or under its main diagonal. We call a matrix quasiseparable when $s << n$. Note that definitions of quasiseparable matrices are sometimes more flexible and use more parameters (see the definition in [23] for an example). We chose not to take into account more general cases as it would make the framework harder to understand while we do not expect adaptations to be difficult to arrange in practice.

This class is a generalisation of many structured matrices. They include band matrices, whose quasiseparability order is the width of the band, their inverses which have the same quasiseparability order, low-rank matrices and semiseparable matrices which are defined as the sum of the lower triangular part of a low-rank matrix and the upper triangular part of another low-rank matrix.

These structures were first studied independently in numerical analysis and multiple storage formats thus emerged. Perhaps the most intuitive one is the $\mathcal{H}$ format (for $\mathcal{H}ierarchical$), in which a quasiseparable matrix is recursively divided into four blocks. The off-diagonal ones have low rank and are stored in rank-revealing factorisations, while the diagonal ones are quasiseparable and are subdivided recursively until their off-diagonal blocks have full rank. For a matrix of order s and dimension n the storage size in this format is $O(ns \log n)$.

This is not entirely satisfactory as none of the structures that quasiseparability generalises need storage space higher than $O(ns)$. However, it gives the main idea for efficient storage of quasiseparable matrices: the structure resides in the rank of matrix blocks, which can be stored as rank-revealing factorisations.

For most of our algorithms the factorisation that is used makes no notable difference (it is possible that choosing a specific factorisation could lead to a slight improvement in our algorithms, see Section 1.3.5). An exception is the Bruhat format which depends on factorisations revealing information on the rank profiles (see Section 1.3.3), for example the `CRE` factorisation [20], where a matrix A is factorised as $A = CRE$ with C and E in echelon form and R is a permutation.

We here need to make a distinction between the rank-revealing factorisations that are used in an exact context and the ones used for numerical linear algebra. The exact rank of an approximation of a real or complex matrix will often be full despite a clear structure in the matrix that it approximates. The notion of numerical rank is then used, which

can be defined in multiple ways. The first one uses a threshold for when negligible values can be treated as zeros. It is sometimes defined as the number of singular values above a certain threshold in the singular value decomposition (SVD) of a matrix , which indicates that the notion of numerical rank and the chosen factorisation are interdependent.

For quasiseparable matrices, the factorisation that is most often used is the compact SVD: $A = U\Sigma V^*$ where Σ is diagonal, U and V are semi-unitary ($UU^* = \mathrm{I}_r$) and U^* is the Hermitian transpose of U. The numerical rank is in this context a fixed parameter that is chosen to approximate a matrix with a factorisation [29]. In the case of the SVD, only the r largest singular values (elements of the diagonal matrix Σ) will be kept and the rest set to 0, whatever the exact rank. In order to adapt algorithms using such a notion of rank, a main contribution is to show that the algorithms remain correct when the rank of a matrix is no longer a fixed parameter but a value given by the problem itself.

1.3.2 Efficient storage formats for numerical quasiseparable matrices

Redundancy of information can be observed in the $\mathcal{H}$ format, which explains the difference in storage costs with band and low-rank matrices. Consider the block matrix

$$A = \begin{bmatrix} \cdot & A_{12} & A_{13} & A_{14} \\ & \cdot & A_{23} & A_{24} \\ & & \cdot & A_{34} \\ & & & \cdot \end{bmatrix} \tag{1.12}$$

of order s whose blocks are stored in factorisations as $A_{12} = L_{12}R_{12}$ and $\begin{bmatrix} A_{13} & A_{14} \\ A_{23} & A_{24} \end{bmatrix} = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} \begin{bmatrix} R_1 & R_2 \end{bmatrix}$. While $\begin{bmatrix} A_{12} & A_{13} & A_{14} \end{bmatrix}$ has rank at most s, it is stored in two independent factorisations. It should intuitively be possible to make L_1 and L_{12} interdependent.

This is what the $\mathcal{H}^2$ format achieves for a storage cost linear in n and s. It is also called HSS for Hierarchically Semi-Separable. The idea, presented in more details in Section 4.2.2, is to represent low-rank blocks recursively with rank revealing factorisation as in the $\mathcal{H}$ format and to also store concatenations of the matrices forming the factorisations as rank-revealing factorisations. In the example of Eq. (1.12), this means using a rank-revealing factorisation for $\begin{bmatrix} L_1 & L_{12} \end{bmatrix}$ and for all factorisation factors which are linearly dependent.

The SSS format, for Sequentially Semi-Separable, was developed independently but uses the same techniques. The block divisions differ, as matrices are seen as an iterative grid of blocks instead of a recursive block division [23]. It is however interesting to notice that in its earliest versions the SSS format [22] was very different from $\mathcal{H}$ as each entry was defined separately and there was no block by block vision with rank revealing factorisations.

1.3.3 Quasiseparable matrices in exact linear algebra

The interest in quasiseparable matrices is recent in exact linear algebra. Quasiseparable matrices arise for example in problems related to linearisation of polynomial linear algebra. A simple example is the companion matrix of a polynomial which has quasiseparability order 1.

In [72] the RRR format is introduced as an adaptation of $\mathcal{H}$ and the Bruhat format is defined explicitly for exact computations. It uses the rank profile matrix (RPM), an invariant based on rank profiles. The column rank profile of a matrix of rank r is the lexicographically minimal set of r independent columns. The row rank profile of a matrix is defined similarly. Knowledge of the rank profile of a matrix A, or any set $\mathcal{C}$ of r independent columns (or rows), is useful as a rank-revealing factorisation of A can easily be computed from that of $A_{*,\mathcal{C}}$ (the columns of A with indices in $\mathcal{C}$).

A tool introduced in [20] to combine and complete the information given in the rank profiles is the rank profile matrix. The unique rank profile matrix R of A has zeros and r ones placed so that all its leading submatrices have the same ranks as the ones of A. The information is basically where a set of pivots of A can be found during an elimination: it pairs each row and column of the rank profiles.

The rank profile matrix induces a new rank-revealing factorisation known as `CRE` factorisation, which associates each pivot of the RPM to a rank 1 outer product. We have $A = CRE$ where C and E are in echelon form and R is the nonzero rows and columns of the RPM.

This `CRE` factorisation can be used to store a quasiseparable matrix A. Take the upper-triangular or lower-triangular part of A and switch the order of the rows or columns to get a left triangular matrix L (its entries under the anti-diagonal are zero). The quasiseparable constraints on the ranks on the submatrices of A are transferred to the leading submatrices of the RPM of L. This means that the number of pivots of its left triangular part constrains how close they must be to the diagonal: a pivot near the diagonal counts in the rank of only few submatrices of the left-triangular part while a pivot in the top-left corner counts for all submatrices. Consider also that the parts of the C and E matrices of the `CRE` factorisation of L that impact only the right-triangular part do not need to be stored. Hence for a specific pivot, the closer it is to the diagonal, the smaller the parts are of C and E that need to be stored.

1.3.4 Contributions and comparisons

From this succinct presentation of the previous work on quasiseparable matrices from the point of view of exact computations a main question arises: when quasiseparable

matrices need to be handled to solve a problem, what storage format is the best suited? As the existing asymptotic cost bounds are the same for most operations, a more detailed comparison needs to be done taking in consideration the theoretical leading constants and practical experiments. It can also be useful to understand what are the specificities of each formats in terms of flexibility for example, and what use cases they are most adapted to.

In order to handle this comparison, the formats coming from the numerical context need to be adapted to exact computations (as mentioned earlier, the use of numerical rank revealing factorisation needs adaptation). Cost comparisons need to be made on problems which are solved by existing algorithms in each format, or new algorithms need to be developed.

In Chapter 4 we chose to compare three formats: HSS, SSS and Bruhat. We do not work with RRR and $\mathcal{H}$ as we estimate the logarithmic factor in the storage and operation costs dominates a possible improvement on the leading constant too fast when the dimensions increase. Indeed, taking a large margin by saying the leading constants of the costs could be 10 times better for RRR than for HSS, the logarithmic factor would compensate for this improvement as soon as n gets close to 1000, which is a dimension for which operations are already extremely fast in any case. We also avoid the format based on Givens weights of [16] as its reliance on orthogonal transformations with complex numbers made it unclear to us how to adapt it to general fields (in particular those with positive characteristic), although the storage cost is linear.

The comparisons we chose to make are on the storage cost and the cost of multiple operations: construction of generators from dense and sparse matrices (Section 4.3), application of a quasiseparable matrix to a block of vectors (Section 4.4) and addition and multiplication of two quasiseparable matrices (Sections 4.5 an 4.6).

It soon became clear to us that the HSS format was not adapted to our framework as the first comparisons we made showed a factor at least 2 in the cost with no more flexibility than the SSS format, hence we chose to focus on Bruhat and SSS after the first basic operations. We also did not manage to produce fast algorithms for the product of two matrices in Bruhat format and construction of an SSS generator from a general sparse matrix at a reasonable cost, so no results are presented for these operations. The efforts made in the direction of a sub-quadratic algorithm for the product in Bruhat are presented in the following section.

For the other operations, we give algorithms with asymptotic cost bounds including the leading constants. We also make practical experiments for the cost of computing SSS and Bruhat generators from a dense matrix and applying a matrix given in these formats to a block of vectors. For these practical experiments, we use the implementation that was

already available in `fflas-ffpack` for the Bruhat format and implemented the necessary algorithms and benchmark routines for SSS.

A table summarising the theoretical cost bounds is given in Table 4.1 while the results of the practical experiments are shown in Sections 4.3.5 and 4.4.3.

1.3.5 Further improvements for Bruhat and SSS

The logarithmic factor in the cost of addition of two quasiseparable matrices in Bruhat form is not inherent to the operation, as it can be done faster with matrices in SSS form. It is *a priori* not inherent to the format either, since it has a storage cost lower than that of SSS. We believe it is possible to avoid this cost with a better use of the quasiseparable structure in our algorithm.

The logarithmic factor appears at multiple steps. We managed to avoid it in all steps which make eliminations and products by designing divide-and-conquer algorithms which take into account the echelon forms of intermediary matrices. Our algorithm also requires the computation of the rank profile of a concatenation of two echelon forms, which we did not manage to do at a cost which does not induce a logarithmic factor. A similar issue arises for the product and seems even more complex as the matrices for which we want to compute rank profiles show less structure. The best cost bound today is quadratic. It is our belief that once a linear or quasi-linear algorithm exists for the product, it should be possible to develop linear or quasi-linear algorithms based on divide-and-conquer and/or Krylov iterations for solving quasiseparable linear systems and computing the determinant and inverse of a matrix given in Bruhat form.

On the SSS side, the implementation in `fflas-ffpack` is made using PLUQ as the rank-revealing factorisation. The factorisation is made *in place*, which means that the result is stored in the memory slots of the input. The triangular parts of this result are then extracted to be stored in the SSS generator. It became clear to us while implementing the compression algorithm that a significant portion of the square matrices in the output generator were actually triangular, although this property is not used. Deeper study of the shape of the SSS generator when using PLUQ factorisation could lead to a new compression algorithm using specific storage for triangular matrices and with a lower leading constant for the storage cost. This improvement would also be seen in the cost of computing products.

Chapter 2 | Computing the Characteristic Polynomial of Generic Toeplitz-like and Hankel-like Matrices

This chapter is derived from a joint work with Clément Pernet, Pierre Karpman and Gilles Villard presented at the 2021 edition of the International Symposium on Symbolic and Algebraic Computation [49].

2.1 Introduction

We consider the problem of computing the minimal or the characteristic polynomial of Toeplitz-like and Hankel-like matrices, which include Toeplitz and Hankel ones. The necessary definitions about those structures are given in Section 2.2.

Throughout the chapter $T \in \mathbb{K}^{n\times n}$ is non-singular and either Toeplitz-like or Hankel-like, where $\mathbb{K}$ is a commutative field. The structure is parameterised by the displacement rank $1 \leq \alpha \leq n$ of T [42,66]. In particular a Toeplitz or a Hankel matrix has displacement rank $\alpha = 2$.

The determinant of T can be computed in $\tilde{O}(\alpha^{\omega-1}n)$ operations in $\mathbb{K}$, where $\omega \leq 3$ is a feasible exponent for square $n \times n$ matrix multiplication. For the best known value of ω one can take $\omega \approx 2.373$ [1, 53]. When T has generic rank profile (the leading principal submatrices are non singular) a complexity bound $\tilde{O}(\alpha^2 n)$ for the determinant is derived from [66, Cor. 5.3.3]. In the general case, for ensuring the rank profile one uses rank-regularisation techniques initially developed in [43,46] that lead to randomised Las Vegas algorithms assuming that the cardinality of $\mathbb{K}$ is large enough; see [66, Sec. 5] and [8] for detailed studies in our context. Taking advantage of fast matrix multiplication is possible using the results in [8,9], where fundamental matrix operations, including the determinant, are performed in time $\tilde{O}(\alpha^{\omega-1}n)$ for a wide spectrum of displacement structures. In this approach the determinant is revealed by the recursive factorisation of the inverse.

The characteristic polynomial $\det(x\mathrm{I}_n - T)$ of T is a polynomial of degree n. Using

an evaluation-interpolation scheme it follows that it can be computed in $\tilde{O}(\alpha^{\omega-1}n^2)$ operations in $\mathbb{K}$. We also refer to [66, Ch. 7] for a Newton-Structured iteration scheme in time $\tilde{O}(\alpha^2 n^2)$.

For a Toeplitz or Hankel matrix these complexity bounds for computing the characteristic polynomial were quadratic; our contribution establishes an improved bound $\tilde{O}\left(n^{2-1/\omega}\right)$ for generic matrices (given in compressed form), which is sub-quadratic including when using $\omega = 3$. We build on the results of [83] where especially the case of a Sylvester matrix was treated, and show that the approach can be generalised to larger displacement rank families. In particular, the Hankel-(like) case requires the use of sophisticated techniques in order to handle the Toeplitz+Hankel structure [31, 33] and its generalisations [66].

The algorithms we propose fit into the broad family of Coppersmith's block Wiedemann algorithms; we refer to [47] for the necessary material and detailed considerations on the approach. Another interpretation in terms of structured lifting and matrix fraction reconstruction is given in [83].

From $T \in \mathbb{K}^{n\times n}$, the problem is to compute the determinant (or a divisor) of the characteristic matrix $M(x) = x\mathrm{I}_n - T$. For $1 \le m \le n$ and well chosen projection matrices V and W in $\mathbb{K}^{n\times m}$, the principle is to reconstruct an irreducible fraction description $P(x)Q^{-1}(x)$ of $V^{\mathsf{T}}M(x)^{-1}W \in \mathbb{K}(x)^{m\times m}$, where $P, Q \in \mathbb{K}[x]^{m\times m}$, from a truncated series expansion of the fraction. The denominator matrix Q carries information on the Smith normal form of $M(x)$ [47, Thm. 2.12]. Using random V and W allows to recover the minimal polynomial of T from the largest invariant factor of $M(x)$, and for a generic matrix T the characteristic polynomial is obtained [47, 83].

The matrix Q is computed from a truncation $S^{(m)} \in \mathbb{K}[x]^{m\times m}$ of the series expansion of $V^{\mathsf{T}}M(x)^{-1}W$,

$$S^{(m)}(x) = -\sum_{k\ge 0}^{2\lceil n/m\rceil} V^{\mathsf{T}}(T^{-k-1})Wx^k, \tag{2.1}$$

using for example matrix fraction reconstruction [3, 26]. We will not detail these latter aspects in this chapter since they can be found elsewhere in the literature: see [47, 83] for the general techniques involved; [82, Cor. 6.4] for the power series truncation and [48] for alternative fraction reconstruction possibilities. The results we need on matrix polynomials are recalled in Section 2.3.

We focus on the computation of the power series terms $H_k = V^{\mathsf{T}}(T^{-1})^k W$ of Eq. (2.1). The idea for improving the complexity bounds is to use structured projections V and W in order to speed up the computation of the expansion, as has been done in [21, 83]. A typical choice is such that the matrix product by V and W is reduced. The central difficulty is to show that the algorithm remains correct; special choices for V and W could

prevent a fraction reconstruction with appropriate cost, or give a denominator matrix Q with too little information on the invariant structure of T.

For a generic input matrix and our best exponent, in Section 2.5 we follow the choice of [83] and work with $V = W = X$ where $X = \begin{pmatrix} \mathrm{I}_m & 0 \end{pmatrix}^{\mathsf{T}} \in \mathbb{K}^{n \times m}$. An $n \times n$ Toeplitz or a Hankel matrix is defined by $2n-1$ elements of $\mathbb{K}$, and our algorithm is correct except on a certain hypersurface of $\mathbb{K}^{2n-1}$. The same way, a Toeplitz-like or Hankel-like matrix of displacement rank α is defined by the $2n\alpha$ coefficients of its generators, and our algorithm is correct for all values of $\mathbb{K}^{2n\alpha}$ except for a hypersurface. If T is Hankel, the matrix $M(x) = x\mathrm{I}_n - T$ is Toeplitz+Hankel and the algorithm involves a compressed form that generalizes the use of generators associated to displacement operators [33, 66]. The algorithm computes a compressed representation of $M(x)^{-1}$ modulo $x^{2\lceil n/m \rceil + 1}$, and exploits its structure to truncate it into a compressed representation of $S^{(m)}(x) = X^{\mathsf{T}} M(x)^{-1} X$ mod $x^{2\lceil n/m \rceil + 1}$ at no cost. The parameter m can be optimised to get an algorithm using $\tilde{O}\left(n^{2-1/\omega}\right)$ operations when the displacement rank is considered constant.

Before considering the fast algorithm for the generic case, in Section 2.4 we consider the baby steps/giant steps algorithm of [47]. Indeed, thanks to the incorporation of fast matrix multiplication in basis structured matrix operations [8, 9], the overall approach with dense projections V and W already allows a slight exponent improvement. Taking into account that the input matrix T is structured, a direct cost analysis of the algorithm of [47] improves on the quadratic cost for Toeplitz and Hankel matrices as soon as one takes $\omega < 3$. However it is unclear to us how to compute the characteristic polynomial in this case (see the related Open Problem 3 in [44]). The algorithm we propose is randomised Monte Carlo and we compute the minimal polynomial in $\tilde{O}\left(n^{\omega - c(\omega)}\right)$ operations with $c(\omega) = \frac{\omega - 1}{5 - \omega}$. For Toeplitz-like and Hankel-like matrices with displacement rank α, the cost is multiplied by $\tilde{O}\left(\alpha^{c(\omega)}\right)$.

Notation Indices of matrix and vectors start from zero. The vectors of the n-dimensional canonical basis are denoted by $e_0^n, \ldots, e_{n-1}^n$. For a matrix M, $M_{i,j}$ denotes the coefficient (i, j) of this matrix, $M_{i,*}$, its row of index i and $M_{*,j}$ its column of index j.

2.2 Rank displacement structures

A wide range of structured matrices are efficiently described by the action of a displacement operator [42]. There are two types of such operators: the Sylvester operators of the form

$$\nabla_{M,N} : A \mapsto MA - AN,$$

and the Stein operators of the form

$$\Delta_{M,N} : A \mapsto A - MAN,$$

where M and N are fixed matrices. A Toeplitz matrix T is defined by $2n-1$ coefficients $t_{-n+1}, \ldots, t_{n-1} \in \mathbb{K}$ such that $T = (t_{i-j})_{i,j}$. Its image through $\Delta_{Z_n, {Z_n}^\intercal}$, where $Z_n = (\delta_{i,j+1})_{0 \leq i,j \leq n-1}$ has rank at most 2. Similarly, a Hankel matrix H is defined by $2n-1$ coefficients $h_0, \ldots, h_{2n-2}$ such that $H = (h_{i+j})_{i,j}$ and its image through $\nabla_{Z_n, {Z_{n,1}}^\intercal}$, where $Z_{n,1} = Z_n + e_0^n {e_{n-1}^n}^\intercal$ has rank at most 2.

As a generalisation, the class of Toeplitz-like (resp. Hankel-like) matrices is defined as those matrices whose image through $\Delta_{Z_n, {Z_n}^\intercal}$ (resp. $\nabla_{Z_n, {Z_{n,1}}^\intercal}$) has a bounded rank α [32,66], called the *displacement rank*, and can be represented by a product $GH^\intercal$, where $G, H \in \mathbb{K}^{n \times \alpha}$ are called *generators*. These operators are non-singular and a matrix can be uniquely recovered from its generators.

Lastly, any sum of a Toeplitz and a Hankel matrix, (forming the class of Toeplitz+Hankel matrices) has an image of rank at most 4 through the displacement operator ∇_{U_n, U_n} where $U_n = Z_n + {Z_n}^\intercal$ [31]. This operator is singular and the low rank image does not suffice to uniquely reconstruct the initial matrix: additional data (usually a first or a last column) is required for a unique reconstruction [33,66]. The class of Toeplitz+Hankel-like matrices is formed by those matrices whose image through ∇_{U_n, U_n} has a bounded rank.

2.2.1 Product of Structured Matrices

Proposition 2.2.1 ([8, Theorem 1.2]). *Let $A \in \mathbb{K}^{n \times n}$ be a Toeplitz-like or Hankel-like matrix with displacement rank α given by its generators and $B \in \mathbb{K}^{n \times m}$ be a dense matrix. The multiplication of A by B can be computed in $\tilde{O}\left(n \max(\alpha, m) \min(\alpha, m)^{\omega-2}\right)$ operations in $\mathbb{K}$.*

Proposition 2.2.2. *Let $A, B \in \mathbb{K}^{n \times n}$ be two Toeplitz-like matrices of displacement rank α and β respectively, then their product AB is a Toeplitz-like matrix of displacement rank at most $\alpha + \beta + 1$. Furthermore, given generators for A and B w.r.t. $\Delta_{Z_n, {Z_n}^\intercal}$, one can compute generators for AB w.r.t. the same operator in $\tilde{O}\left(n(\alpha+\beta)^{\omega-1}\right)$ field operations.*

Proof. Let G_A, H_A and G_B, H_B be the generators of A and B respectively. They satisfy A-$Z_n A {Z_n}^\intercal = G_A {H_A}^\intercal$ and $B - Z_n B {Z_n}^\intercal = G_B {H_B}^\intercal$. Consequently

$$\begin{aligned} AB = & \quad (Z_n A {Z_n}^\intercal + G_A {H_A}^\intercal)(Z_n B {Z_n}^\intercal + G_B {H_B}^\intercal) \\ = & \quad Z_n A B Z_n^\intercal - Z_n A_{*,n-1} B_{n-1,*} {Z_n}^\intercal + (Z_n A {Z_n}^\intercal G_B) {H_B}^\intercal \\ & + G_A({H_A}^\intercal Z_n B {Z_n}^\intercal + {H_A}^\intercal G_B {H_B}^\intercal), \end{aligned}$$

and therefore $AB - Z_n AB {Z_n}^\mathsf{T} = G_{AB} {H_{AB}}^\mathsf{T}$ for

$$\begin{aligned} G_{AB} &= \left(\; G_A \;\middle|\; Z_n A {Z_n}^\mathsf{T} G_B \;\middle|\; -Z_n A_{*,n-1} \; \right) \\ H_{AB} &= \left(\; Z_n B^\mathsf{T} {Z_n}^\mathsf{T} H_A + H_B {G_B}^\mathsf{T} H_A \;\middle|\; H_B \;\middle|\; Z_n {B_{n-1,*}}^\mathsf{T} \; \right), \end{aligned}$$

thus showing that AB has displacement rank at most $\alpha + \beta + 1$.

Computing these generators involves applying A on a dense $n \times \beta$ matrix and B on a dense $\alpha \times n$ matrix, and computing the product of an $\alpha \times n$ by an $n \times \beta$ matrix and the product of an $\alpha \times \beta$ by a $\beta \times n$ matrix. Using [8, Thm 1.2], these cost $\tilde{O}\left(n(\alpha+\beta)^{\omega-1}\right)$ field operations. □

Proposition 2.2.3. *Let $A, B \in \mathbb{K}^{n \times n}$ be two Hankel-like matrices of displacement rank α and β respectively, then their product AB is a Toeplitz-like matrix of displacement rank at most $\alpha + \beta + 2$. Furthermore, given generators for A and B w.r.t. $\nabla_{Z_n, {Z_{n,1}}^\mathsf{T}}$, generators for AB w.r.t. $\Delta_{Z_n, {Z_n}^\mathsf{T}}$ can be computed in $\tilde{O}\left(n(\alpha+\beta)^{\omega-1}\right)$.*

Proof. Let G_A, H_A and G_B, H_B be the generators of A and B respectively, satisfying $Z_n A - A {Z_{n,1}}^\mathsf{T} = G_A {H_A}^\mathsf{T}$ and $Z_n B - B {Z_{n,1}}^\mathsf{T} = G_B {H_B}^\mathsf{T}$. Using a similar reasoning as for Proposition 2.2.2 we can deduce that

$AB - Z_n AB {Z_n}^\mathsf{T} = G_{AB} {H_{AB}}^\mathsf{T}$ for

$$\begin{aligned} G_{AB} &= \left(\; G_A \;\middle|\; A {Z_{n,1}}^\mathsf{T} G_B \;\middle|\; A_{*,n-1} \;\middle|\; A {Z_{n,1}}^\mathsf{T} B_{*,n-1} \; \right) \\ H_{AB} &= \left(\; \left(H_B {G_B}^\mathsf{T} - B^\mathsf{T} {Z_n}^\mathsf{T} + e_0^n {B_{*,n-1}}^\mathsf{T} \right) H_A \;\middle|\; H_B \;\middle|\; {B_{n-1,*}}^\mathsf{T} \;\middle|\; e_0^n \; \right), \end{aligned}$$

thus showing that AB has displacement rank at most $\alpha + \beta + 2$. Computing these generators again costs $\tilde{O}\left(n(\alpha+\beta)^{\omega-1}\right)$ field operations. □

Proposition 2.2.4. *Let $A \in \mathbb{K}^{n \times n}$ be a Toeplitz-like (resp. Hankel-like) matrix of displacement rank α, then for an arbitrary (resp. even)r, A^r is a Toeplitz-like matrix of displacement rank at most $(\alpha+1)r$ and its generators can be computed from the generators of A in $\tilde{O}\left(n(\alpha r)^{\omega-1}\right)$ field operations.*

Proof. Using fast exponentiation one computes A^r as:

$$A^r = \prod_{k=0}^{\lfloor \log r \rfloor} \left(A^{2^k}\right)^{l_k} \text{ where the } l_k \text{ satisfy } \sum_{k=0}^{\log r} l_k 2^k = r,$$

which only requires squarings and products between matrices of the form A^{2^k}. When A is Toeplitz-like the result is a straightforward consequence of Proposition 2.2.2; when it is Hankel-like the product A^2 is computed using Proposition 2.2.3, the remaining products are between Toeplitz-like matrices, and the result again follows from Proposition 2.2.2. □

2.2.2 Reconstruction of a Toeplitz+Hankel-like matrix from its generators

The operator ∇_{U_n,U_n} is defined in [66, Section 4.5] as partly regular, which means that a Toeplitz+Hankel-like matrix is completely defined by its generators *and* its *irregularity set* that may be all the entries in its first column.

A formula to recover a dense representation of the matrix from its generators and its first column is given in [66].

Theorem 2.2.5 ([66, Thm. 4.5.1])**.** *Let $M \in \mathbb{K}^{n\times n}$ be a Toeplitz+Hankel-like matrix, $G, H \in \mathbb{K}^{n\times\alpha}$ its generators and $c_0 = Me_0^n$ its first column, then*

$$M = \tau_{U_n}(c_0) - \sum_{j=0}^{\alpha-1} \tau_{U_n}(G_{*,j})\tau_{Z_n}(Z_n H_{*,j})^{\mathsf{T}} \tag{2.2}$$

where for an $n \times n$ matrix A and a vector v of length n $\tau_A(v)$ denotes the matrix of the algebra generated by A which has v as its first column.

We show that one can derive a fast reconstruction algorithm for a Toeplitz+Hankel-like matrix from Eq. (2.2) and first detail the structure of the various $\tau_A(v)$ matrices.

Lemma 2.2.6. *$\tau_{Z_n}(v)^{\mathsf{T}}$ is the Toeplitz upper-triangular matrix with v^{T} as its first row.*

Lemma 2.2.7. *$\tau_{U_n}(v) = \sum_{i=0}^{n-1} v_i Q_i(U_n)$ where $Q_0(x) = 1$, $Q_1(x) = x$ and $Q_{i+1}(x) = xQ_i(x) - Q_{i-1}(x)$.*

Proof. The first column of $Q_i(U_n)$ is e_i^n. □

Corollary 2.2.8. *Column j of $\tau_{U_n}(v)$ is $Q_j(U_n)v$.*

Proof. With Lemma 2.2.7 and after checking the property for $j \in \{0, 1\}$, it suffices to prove $Q_i(U_n)_{*,j+1} = U_n Q_i(U_n)_{*,j} - Q_i(U_n)_{*,j-1}$. This is true for $i \in \{0, 1\}$ and if it is for i and $i - 1$, then

$$\begin{aligned} Q_{i+1}(U_n)_{*,j+1} = {} & U_n^2 Q_i(U_n)_{*,j} - U_n Q_i(U_n)_{*,j-1} \\ & - U_n Q_{i-1}(U_n)_{*,j} + Q_{i-1}(U_n)_{*,i-1} \end{aligned}$$

□

From these we can write the following proposition, inspired by [31, Prop. 4.2]. It enables fast recursive reconstruction of the columns of a Toeplitz+Hankel-like matrix from the first one.

Proposition 2.2.9. *Let $M \in \mathbb{K}^{n\times n}$ be a Toeplitz+Hankel-like matrix, $G, H \in \mathbb{K}^{n\times\alpha}$ its generators for ∇_{U_n,U_n} and $c_0 = Me_0^n$ its first column. With the notation $c_{-1} = 0$, the columns $(c_k)_{0\leq k\leq n-1}$ of M follow the recursion:*

$$c_{k+1} = U_n c_k - c_{k-1} - \sum_{j=0}^{\alpha-1} H_{k,j} G_{*,j}. \tag{2.3}$$

Proof. Let C be the matrix defined by the recursion formula and initial conditions of Proposition 2.2.9, we will prove $C = M$.

By definition c_0 is the first column of M; assume now that for $i \leq k, c_i$ is column i of M. Using Lemma 2.2.6 and Corollary 2.2.8 on Eq. (2.2) that is

$$c_i = Q_i(U_n)c_0 - \sum_{j=0}^{\alpha-1}\sum_{l=0}^{i-1} H_{i-1-l,j} Q_l(U_n) G_{*,j} \tag{2.4}$$

and Eq. (2.3) can be detailed as

$$\begin{aligned} c_{k+1} &= U_n \left(Q_k(U_n)c_0 - \sum_{j=0}^{\alpha-1}\sum_{i=0}^{k-1} H_{k-1-i,j} Q_i(U_n) G_{*,j} \right) \\ &\quad - \left(Q_{k-1}(U_n)c_0 - \sum_{j=0}^{\alpha-1}\sum_{i=0}^{k-2} H_{k-2-i,j} Q_i(U_n) G_{*,j} \right) - \sum_{j=0}^{\alpha-1} H_{k,j} G_{*,j} \\ &= Q_{k+1}(U_n)c_0 - \sum_{j=0}^{\alpha-1}\sum_{i=0}^{k} H_{k-i,j} Q_i(U_n) G_{*,j} \end{aligned}$$

□

2.3 Matrix Polynomials

We rely on the material from [47, 83]. For matrix polynomials and fractions the reader may refer to [41]. The rational matrix $H(x) = V^{\mathsf{T}} M(x)^{-1} W$ over $\mathbb{K}(x)$ can be written as a fraction of two polynomial matrices. A right fraction description is given by square polynomial matrices $P(x)$ and $Q(x)$ such that $H(x) = P(x)Q(x)^{-1} \in \mathbb{K}(x)^{m\times m}$, and a left description by $P_l(x)$ and $Q_l(x)$ such that $H(x) = Q_l(x)^{-1}P_l(x) \in \mathbb{K}(x)^{m\times m}$. Degrees of denominator matrices are minimised using column-reduced forms. A non-singular polynomial matrix is said to be column-reduced if its leading column coefficient matrix is non-singular [41, Sec. 6.3]. We also have the notion of irreducible and minimal fraction descriptions. If P and Q (resp. P_l and Q_l) have unimodular right (resp. left) matrix gcd's [41, Sec. 6.3] then the description is called irreducible. If Q (resp. Q_l) is

column-reduced then the description is called minimal.

For a given m, define $1 \leq \nu \leq n$ to be the sum of the degrees of the first m largest invariant factors of $M(x) = x\mathrm{I}_n - T$ (equivalently, the first m diagonal elements of its Smith normal form). The following will ensure that the minimal polynomial of T, which is the largest invariant factor of $M(x)$, can be computed from the Smith normal form of an appropriate denominator $Q(x)$.

Theorem 2.3.1. ([47, Thm. 2.12] and [82].) *Let V and W be block vectors over a sufficiently large field $\mathbb{K}$ whose entries are sampled uniformly and independently from a finite subset $S \subseteq \mathbb{K}$. Then with probability at least $1 - 2n/|S|$, $H(x) = V^{\mathsf{T}} M(x)^{-1} W$ has left and right irreducible descriptions with denominators of degree $\lceil \nu/m \rceil$, of determinantal degree ν, and whose i^{th} invariant factor (starting from the largest degree) is the i^{th} invariant factor of $M(x)$.*

The next result we need is concerned with the computation of an appropriate denominator Q as soon as the truncated power series in Eq. (2.1) is known. We notice that $H(x) = V^{\mathsf{T}} M(x)^{-1} W$ is strictly proper in that it tends to zero when x tends to infinity. For fraction reconstruction we use the computation of minimal approximant bases (or σ-bases) [3, 79], and the algorithm with complexity bound $\tilde{O}\left(m^{\omega-1} n\right)$ in [26, 39].

Theorem 2.3.2. ([26, Lem. 3.7].) *Let $H \in \mathbb{K}(x)^{m \times m}$ be a strictly proper power series, with left and right matrix fraction descriptions of degree at most d. A denominator Q of a right irreducible description $H(x) = P(x)Q(x)^{-1}$ can be computed in $\tilde{O}\left(m^{\omega} d\right)$ arithmetic operations from the first $2d+1$ terms of the expansion of H.*

In our case, from Theorem 2.3.1 we will obtain the existence of appropriate fractions of degree less than $\lceil n/m \rceil$, and use Theorem 2.3.2 for bounding the cost of the computation of Q.

2.4 A randomised algorithm with the baby-steps/giant-steps method

In this section, we propose a direct adaptation of the baby steps/giant steps variant of Coppersmith's block-Wiedemann algorithm developed in [47, Sec. 4] to the case of structured matrices. In order to compute the terms of the series (2.1), we will assume that the input matrix T has been inverted, using [8, Theorem 6.6]. In this section we will therefore denote by T this inverse and compute the projections of its powers.

2.4.1 Description of the algorithm

Let $U, V \in \mathbb{K}^{n \times m}$ be the block vectors used for the projections. Algorithm StructuredBSGS performs r baby steps and s giant steps to compute the first terms of the

sequence $H_k = U^\mathsf{T} T^{k+1} V = U^\mathsf{T} (T^r)^j T^{i+1} V$ for $0 \leq k \leq 2\lceil n/m \rceil$, $0 \leq i < r$, $0 \leq j < s$ and $rs \geq 2\lceil n/m \rceil + 1$.

Algorithm 2.4.1 StructuredBSGS

Input: A generator for $T \in \mathbb{K}^{n \times n}$, Toeplitz-like or Hankel-like
Input: $m, r, s \in \mathbb{N}$ s.t. $rs \geq 2\lceil n/m \rceil + 1$, r even if T is Hankel-like
Input: $U, V \in \mathbb{K}^{n \times m}$
Output: $H = (H_{rj+i})_{j<s, i<r}$ where $H_k = U^\mathsf{T} T^{k+1} V$

1: $V_0 \leftarrow TV$
2: **for** $1 \leq i \leq r-1$ **do**
3: $\quad V_i \leftarrow TV_{i-1}$
4: $R \leftarrow T^r$
5: $U_0 \leftarrow U$
6: **for** $1 \leq j \leq s-1$ **do**
7: $\quad {U_j}^\mathsf{T} \leftarrow {U_{j-1}}^\mathsf{T} R$
8: $H \leftarrow \begin{pmatrix} U_0 & \dots & U_{s-1} \end{pmatrix}^\mathsf{T} \begin{pmatrix} V_0 & \dots & V_{r-1} \end{pmatrix}$

This algorithm relies on three main matrix operations:

1. The product of a structured matrix by a dense rectangular matrix, supported by Theorem 3.3.2 for Steps 1, 3 and 7;
2. The exponentiation of a structured matrix, supported by Proposition 2.2.4 for Step 4;
3. The product of two dense rectangular matrices for Step 8.

2.4.2 Detailed cost analysis

Proposition 2.4.1. *Algorithm StructuredBSGS runs in* $\tilde{O}\left(n^{\omega - \frac{\omega-1}{5-\omega}} \alpha^{\frac{\omega-1}{5-\omega}}\right)$ *operations in* $\mathbb{K}$ *for well chosen* m, r *and* s.

Proof. Using Theorem 3.3.2, applying an $n \times m$ block to T uses $\tilde{O}\left(n \max(m, \alpha) \min(m, \alpha)^{\omega-2}\right)$ field operations. Hence the r baby steps, Step 3, computing the $(T^i W)_{0 \leq i < r}$ cost overall

$$\tilde{O}\left(nr \max(m, \alpha) \min(m, \alpha)^{\omega-2}\right) \tag{2.5}$$

field operations.

By Proposition 2.2.4, the initialisation of the giant steps at Step 4 is the computation of a structured representation for T^r, which can be done in

$$\tilde{O}\left(nr^{\omega-1} \alpha^{\omega-1}\right) \tag{2.6}$$

operations in $\mathbb{K}$.

Then each of the giant steps, at Step 7, is a product of an $m \times n$ dense matrix by an $n \times n$ matrix of displacement rank αr. From Theorem 3.3.2, these s steps cost

$$\tilde{O}\left(ns \max(m, \alpha r) \min(m, \alpha r)^{\omega-2}\right) \tag{2.7}$$

Lastly, the number of operations needed for computing the product resulting in H at Step 8 is $\tilde{O}\left(n \max(mr, ms) \min(mr, ms)^{\omega-2}\right)$, or equivalently

$$\tilde{O}\left(nm^{\omega-1} \max(r, s) \min(r, s)^{\omega-2}\right). \tag{2.8}$$

Let $m = \left\lceil n^{\frac{\omega-3}{\omega-5}} \alpha^{\frac{2}{5-\omega}} \right\rceil$ and set $r = s = \left\lceil \sqrt{2n/m} \right\rceil$. Note that $\alpha \leq m \leq \alpha r$, therefore the bound of Eq. (2.5) is dominated by the one of Eq. (4.13). Moreover the bound of Eq. (2.7) can be rewritten as $\tilde{O}\left(n^2 m^{\omega-3} \alpha\right)$, and from Eq. (4.13) we have $\tilde{O}\left(n^{\frac{\omega+1}{2}} m^{\frac{\omega-1}{2}}\right)$, and these two quantities are

$$\tilde{O}\left(n^{\omega-\frac{\omega-1}{5-\omega}} \alpha^{\frac{\omega-1}{5-\omega}}\right).$$

Finally, the bound of Eq. (2.6) can be rewritten as $\tilde{O}\left(n^{\frac{\omega+1}{2}} (\frac{\alpha^2}{m})^{\frac{\omega-1}{2}}\right)$, which is dominated by the one of Eq. (4.13). □

When the displacement rank α is constant, and with the best known estimate $\omega = 2.373$ [1,53] the cost bound given in Proposition 2.4.1 becomes $\tilde{O}\left(n^{1.851}\right)$, while it is $\tilde{O}\left(n^2\right)$ for $\omega = 3$.

Let us now suppose that the entries of V and W are sampled uniformly and independently from a finite subset $S \subseteq \mathbb{K}$, we then have the following.

Theorem 2.4.2. *The minimal polynomial of an $n \times n$ Toeplitz-like or Hankel-like matrix with displacement rank α can be computed by a randomised Monte Carlo algorithm using*

$$\tilde{O}\left(n^{\omega-\frac{\omega-1}{5-\omega}} \alpha^{\frac{\omega-1}{5-\omega}}\right)$$

field operations, with probability of success at least $1 - (n^2 + 3n + 2n^{5/3}\alpha^2)/|S|$.

Proof. The first step is to compute the inverse of T, using [8, Theorem 6.6] in $\tilde{O}\left(n\alpha^{\omega-1}\right)$ operations in $\mathbb{K}$. Then running Algorithm StructuredBSGS on T^{-1} costs $\tilde{O}\left(n^{\omega-\frac{\omega-1}{5-\omega}} \alpha^{\frac{\omega-1}{5-\omega}}\right)$ which dominates $\tilde{O}\left(n\alpha^{\omega-1}\right)$ since $\alpha \leq n$. From the sequence of matrices $(H_k)_{0 \leq k \leq 2n/m}$, one can compute a minimal denominator Q for $H(x) = V^{\mathsf{T}}(x\mathrm{I}_n - T)^{-1}W \in \mathbb{K}[x]^{m \times m}$ in $\tilde{O}\left(nm^{\omega-1}\right)$ field operations, by Theorem 2.3.2.

Using Theorem 2.3.1, the minimal polynomial is then obtained as the first invariant factor in the Smith form of Q, computed by [77, Proposition 41]. This step also costs

$\tilde{O}(nm^{\omega-1})$ field operations and since $m \leq n$ we have

$$nm^{\omega-1} \leq n^{\frac{\omega+1}{2}} m^{\frac{\omega-1}{2}}$$

which shows that the cost of these last two computations will always be dominated by the cost of the product in Eq. (4.13). The probability of failure for the computation of T^{-1} is at most $n(n+1)/|S|$ by [8, Lemma 6.2]. For the computation of the minimal polynomial it is at most $2m^2n \leq 2n^{5/3}\alpha^2$, from [77, Concl.] and [45, Thm 3.3]. A union bound combining this probability and the failure probability of Theorem 2.3.1 yields a probability of failure of $(n^2 + 3n + 2n^{5/3}\alpha^2)/|S|$. □

Note that this result carries over to the computation of the characteristic polynomial of any Toeplitz-like or Hankel-like matrix T having fewer than m invariant factors in its Frobenius normal form.

2.5 Using structured inversion

In this section we develop a new approach for computing the characteristic polynomial of generic structured polynomial matrices $T \in \mathbb{K}^{n\times n}$ with displacement rank α. Following [83, Sec. 7], in the Toeplitz-like case the idea is to exploit the structure of the ΣLU representation [42]. For Hankel-like matrices (see the discussion after Theorem 2.5.4), we generalize the approach using both generators and irregularity sets that has been introduced in Section 2.2.2 [66].

Principle of the approach Here, rather than using successive applications and powering of T^{-1} as in Section 2.4, the first terms of the sequence $\{H_k\}_k = \{V^\mathsf{T} T^{-k-1} W\}_k$ are obtained as the matrix coefficients of the series expansion of $V^\mathsf{T} M(x)^{-1} W$. Since $2\lceil n/m \rceil + 1$ terms are required, and with the special choice $V = W = X = \begin{pmatrix} \mathrm{I}_m & 0 \end{pmatrix}^\mathsf{T} \in \mathbb{K}^{n\times m}$, this boils down to computing a dense representation of the $m \times m$ leading principal submatrix of $M(x)^{-1} \bmod x^{2\lceil n/m \rceil+1}$. The outline of the algorithm is as follows.

1. Compute the inverse $M(x)^{-1} \bmod x^{2\lceil n/m \rceil+1}$ in a compressed representation;
2. Crop this representation to form a representation of the $m \times m$ leading principal submatrix;
3. Extract $S^{(m)}(x) = X^\mathsf{T} M(x)^{-1} X \bmod x^{2\lceil n/m \rceil+1}$ in dense form.

Below we specialize the approach for the two classes of interest. Our algorithms in Theorems 2.5.2 and 2.5.4 are correct for generic matrices T (in the Zariski sense),

see Assumptions (A1) and (A2) in Section 2.6 to which the discussion on genericity is deferred.

2.5.1 Generic Toeplitz-like Matrices

If T is Toeplitz-like, so is $M(x) = x\mathrm{I}_n - T$ which can be represented in ΣLU form by generators $G, H \in \mathbb{K}[x]^{n\times\alpha}$ such that $M(x) = \sum_{i=0}^{\alpha-1} L(G_{*,i})L(H_{*,i})^T$, where $L(v)$ is the lower triangular Toeplitz matrix with v as its first column [42, 43]. The $m \times m$ leading principal submatrix of any product $L(v)L(w)^\mathsf{T}$ is the product of the $m \times m$ leading principal submatrix of these factors, which in turn is $L(v_{0..m-1})L(w_{0..m-1})^\mathsf{T}$. Algorithm ToeplitzLikeExpansion relies on this property to produce $S^{(m)}$ from the m first rows of the generators of M^{-1}.

Algorithm 2.5.1 ToeplitzLikeExpansion

Input: G, H a generator for $M \in \mathbb{K}[x]^{n\times n}$, a Toeplitz-like matrix of displacement rank α
Output: $S^{(m)} = X^\mathsf{T}M^{-1}X \bmod x^{2\lceil n/m\rceil+1}$ in dense form
1: $(E, F) \leftarrow$ a generator for $M^{-1} \bmod x^{2\lceil n/m\rceil+1}$
2: $E' \leftarrow X^\mathsf{T}E$; $F' \leftarrow FX$
3: $S^{(m)} \leftarrow \sum_{i=0}^{\alpha-1} L(E'_{*,i})L(F'_{*,i})^\mathsf{T} \mod x^{2\lceil n/m\rceil+1}$

Proposition 2.5.1. *Algorithm ToeplitzLikeExpansion is correct for $M(x) = x\mathrm{I}_n - T$; if T has generic rank profile it uses $\tilde{O}(\alpha^{\omega-1}n^2/m + \alpha nm)$ operations in $\mathbb{K}$.*

Proof. From the discussion at the beginning of the section, $E' = E_{0..m-1,*}$ and $F' = F_{0..m-1,*}$ are generators for $S^{(m)} = X^\mathsf{T}M^{-1}X$. We use the algorithm of [9, Prop. 5] for computing the generators of the inverse. Note that no division by x in the ring $\mathbb{K}[x]/\langle x^{2\lceil n/m\rceil+1}\rangle$ will occur in Step 1 since $M(0) = T$ has generic rank profile, and consequently all leading principal minors of $M(x)$ are not divisible by x which shows the correctness.

By [9, Prop. 5], computing the generators of M^{-1} at Step 1 can be done in $\tilde{O}(n\alpha^{\omega-1})$ operations over $\mathbb{K}[x]/\langle x^{2\lceil n/m\rceil+1}\rangle$ which in turn is

$$\tilde{O}\left(\frac{n^2}{m}\alpha^{\omega-1}\right) \tag{2.9}$$

operations in $\mathbb{K}$.

The dense reconstruction of $S^{(m)}$ in Step 3 is achieved by α products of an $m \times m$ Toeplitz matrix $L(E'_{*,i})$ by an $m \times m$ dense matrix $L(F'_{*,i})^\mathsf{T}$ for a total cost of

$$\tilde{O}(nm\alpha) \tag{2.10}$$

operations in $\mathbb{K}$. □

From the efficient computation of the first terms of the expansion of $X^{\mathsf{T}}M(x)^{-1}X$ and using fraction reconstruction, the characteristic polynomial of T is obtained.

Theorem 2.5.2. *The characteristic polynomial of a generic $n \times n$ Toeplitz-like matrix with displacement rank α (assumptions (A1) and (A2) in Section 2.6) can be computed in $\tilde{O}\left(n^{2-\frac{1}{\omega}}\alpha^{\frac{(\omega-1)^2}{\omega}}\right)$ operations in $\mathbb{K}$ when $\alpha = O\left(n^{\frac{\omega-2}{-\omega^2+4\omega-2}}\right)$, and $\tilde{O}\left(n^{\frac{3}{2}}\alpha^{\frac{\omega}{2}}\right)$ otherwise.*

Proof. From Lemma 2.6.1 (genericity assumption (A2)), irreducible left and right fraction descriptions of $X^{\mathsf{T}}M^{-1}X$ have degree at most $\lceil n/m \rceil$. Thus Theorem 2.3.2 ensures that a denominator Q of a right description can be computed from $S^{(m)}(x) = X^{\mathsf{T}}M(x)^{-1}X \bmod x^{2\lceil n/m \rceil+1}$. By Lemma 2.6.1 again, the determinant of Q gives the characteristic polynomial of T.

Besides the computation of $S^{(m)}$ by Proposition 2.5.1 (genericity assumption (A1)), the computation of the denominator Q of its irreducible right fraction description costs

$$\tilde{O}\left(nm^{\omega-1}\right) \tag{2.11}$$

operations by Theorem 2.3.2. Computing the determinant of Q has same cost using the algorithm in [52]. The total cost depends on α.

Case 1: $\alpha = O\left(n^{\frac{\omega-2}{-\omega^2+4\omega-2}}\right)$. We set $m = n^{\frac{1}{\omega}}\alpha^{\frac{\omega-1}{\omega}}$ so that $\alpha = O(m^{\omega-2})$ and the term in Eq. (2.10) is dominated by the one in Eq. (2.11). For the chosen value of m the terms in Eq. (2.9) (decreasing in m) and Eq. (2.11) (increasing in m) are equal, leading to a full cost of $\tilde{O}\left(n^{2-\frac{1}{\omega}}\alpha^{\frac{(\omega-1)^2}{\omega}}\right)$ operations in $\mathbb{K}$.

Case 2: $\alpha = \Omega\left(n^{\frac{\omega-2}{-\omega^2+4\omega-2}}\right)$. We set $m = n^{\frac{1}{2}}\alpha^{\frac{\omega-2}{2}}$ so that $\alpha = \Omega(m^{\omega-2})$. In this case the term in Eq. (2.11) is dominated by the one in Eq. (2.10) and for this value of m we have equality between the terms in Eq. (2.9) and Eq. (2.10), leading to a full cost of $\tilde{O}\left(n^{\frac{3}{2}}\alpha^{\frac{\omega}{2}}\right)$ operations in $\mathbb{K}$. □

The exponent in Theorem 2.5.2 is $O(n^{1.579})$ (resp. $O(n^{1.667})$) for α constant and $\omega = 2.373$ (resp. $\omega = 3$). When $\alpha = \Theta\left(n^{\frac{\omega-2}{-\omega^2+4\omega-2}}\right)$ and taking $\omega = 2.373$ (resp. $\omega = 3$), both expressions become $\tilde{O}\left(n^{1.74}\right)$ (resp. $\tilde{O}\left(n^3\right)$). The complexity bound when α is small can also be written as

$$\tilde{O}\left(n^{\omega-f(\omega)}\alpha^{f(\omega)}\right),$$

similarly as in Proposition 2.4.1, which can be interpreted as a transfer of part of the exponent from n to α by using the structure of the matrix.

2.5.2 Generic Toeplitz+Hankel-like Matrices

We now adapt the previous approach to more general structures. If T is Hankel-like then $M(x) = x\mathrm{I}_n - T$ is Toeplitz+Hankel-like. In this section we consider generic matrices with such a structure.

Compared to the Toeplitz case in Section 2.5.1, only the computation of the truncated expansion of $X^\mathsf{T} M(x)^{-1} X$ is modified. Computing the characteristic polynomial from there does not depend on the structure of M or T (dense matrix polynomial operations).

In addition to the generators one has to consider an irregularity set for M^{-1}. This data is computed by Algorithm THLikeExpansion at Step 1 using the recursive matrix decomposition in [66, Ch. 5]. The irregularity set we consider is the first column. The dense form of $S^{(m)}(x) = X^\mathsf{T} M(x)^{-1} X \bmod x^{2\lceil n/m \rceil + 1}$ is then recovered from its compressed representation using Proposition 2.2.9.

Algorithm 2.5.2 THLikeExpansion

Input: (G, H, v) a generator and irregularity set of $M \in \mathbb{K}[x]^{n \times n}$, a Toeplitz+Hankel-like matrix of displacement rank α.
Output: $S^{(m)} = X^\mathsf{T} M^{-1} X \mod x^{2\lceil n/m \rceil + 1}$ in dense form

1: $(E, F, c) \leftarrow$ a generator and irregularity set for M^{-1}, the irregularity set is the first column ($Mc = e_0^n$)
2: $c_0 \leftarrow \begin{pmatrix} \mathrm{I}_{2m-1} & 0 \end{pmatrix} c$
3: $c_1 \leftarrow U_{2m-1} c_0 - \sum_{j=0}^{\alpha-1} E_{0,j} F_{0\ldots 2m-2,j}$
4: **for** $1 \le k \le m-2$ **do**
5: $\quad c_{k+1} \leftarrow U_{2m-1} c_k - c_{k-1} - \sum_{j=0}^{\alpha-1} F_{k,j} E_{0\ldots 2m-2,j}$
6: $S^{(m)}(x) = \begin{pmatrix} \mathrm{I}_m & 0 \end{pmatrix} (c_0 \cdots c_{m-1})$

Proposition 2.5.3. *Algorithm THLikeExpansion is correct for $M(x) = x\mathrm{I}_n - T$ and if T has generic rank profile it uses $\tilde{O}(\alpha^2 n^2/m + \alpha n m)$ operations in $\mathbb{K}$.*

Proof. As discussed in the proof of Proposition 2.5.1, no division by x occur in the ring $\mathbb{K}[x]/\langle x^{2\lceil n/m \rceil + 1} \rangle$ since since $M(0) = T$ has generic rank profile. Step 1 can be performed in $\tilde{O}(\alpha^2 n)$ operations on truncated power series, so $\tilde{O}\left(\frac{n^2}{m}\alpha^2\right)$ operations in $\mathbb{K}$ [66, Corollary 5.3.3]. Each step of the for loop consists of a number of polynomial operations modulo $x^{2\lceil n/m \rceil + 1}$ linear in $m\alpha$ as U_{2m-1} has only two nonzero entries on each row. Lines 2 to 5 can be performed in $\tilde{O}(m^2\alpha)$ power series operations, so $\tilde{O}(nm\alpha)$ operations in $\mathbb{K}$. By Proposition 2.2.9, if the first $2m-k$ coefficients of c_{k-1} are equal to the ones of column $k-1$ of M^{-1}, then the first $2m-k-1$ coefficients of c_k are equal to the ones of column k of M^{-1}. Since c_0 gives the $2m-1$ first coefficients of column 0 of M^{-1}, Step 6 outputs $S^{(m)}(x)$. □

The characteristic polynomial is then obtained following our general strategy.

Theorem 2.5.4. *The characteristic polynomial of a generic $n \times n$ Toeplitz+Hankel-like matrix with displacement rank α (assumptions (A1) and (A2) in Section 2.6) can be computed in $\tilde{O}\left(n^{2-\frac{1}{\omega}}\alpha^{\frac{2(\omega-1)}{\omega}}\right)$ field operations when $\alpha = O\left(n^{\frac{\omega-2}{4-\omega}}\right)$, and $\tilde{O}\left(n^{\frac{3}{2}}\alpha^{\frac{3}{2}}\right)$ otherwise.*

Proof. The arguments are similar to those of the proof of Theorem 2.5.2, we do not repeat them here. We have only have to discuss the slightly different cost bound. The overall cost is that for computing the matrix denominator Q and its determinant in $\tilde{O}\left(nm^{\omega-1}\right)$ operations in $\mathbb{K}$, plus the cost of computing the sequence $\{H_k\}_k$. We distinguish two cases:

If $\alpha = O\left(n^{\frac{\omega-2}{4-\omega}}\right)$: we take $m = n^{\frac{1}{\omega}}\alpha^{\frac{2}{\omega}}$ so that $\alpha = O(m^{\omega-2})$, with overall cost bound $\tilde{O}\left(n^{2-\frac{1}{\omega}}\alpha^{\frac{2(\omega-1)}{\omega}}\right)$.

If $\alpha = \Omega\left(n^{\frac{\omega-2}{4-\omega}}\right)$: we take $m = n^{\frac{1}{2}}\alpha^{\frac{1}{2}}$ so that $\alpha = \Omega(m^{\omega-2})$, with overall cost bound $\tilde{O}\left(n^{\frac{3}{2}}\alpha^{\frac{3}{2}}\right)$. □

Given $\nabla_{Z_n,Z_{n,1}{}^\mathsf{T}}$-generators of length α for a Hankel-like matrix T, ∇_{U_n,U_n}-generators of length $O(\alpha)$ can be computed in time $\tilde{O}\left(n\alpha\right)$. T can be written as a sum of α terms of the form $LU\mathrm{J}_n$, where L and U are Toeplitz and $\mathrm{J}_n = (\delta_{i,n-1-j})_{0\le i,j\le n-1}$ [66, Example 4.4.4]. Constant-length $\nabla_{Z_n,Z_n{}^\mathsf{T}}$- and $\nabla_{Z_n{}^\mathsf{T},Z_n}$-generators for each of the α terms can then be derived from ∇_{Z_n,Z_n}- and $\nabla_{Z_n{}^\mathsf{T},Z_n{}^\mathsf{T}}$-generators for the products LU using [66, Theorem 1.5.4] and the fact that J_n is in the kernel of $\nabla_{Z_n,Z_n{}^\mathsf{T}}$ and $\nabla_{Z_n{}^\mathsf{T},Z_n}$. Concatenation of the obtained generators yields the result.

Note that the complexity bound in n in Theorem 2.5.4 is the same as in the Toeplitz-like case (Theorem 2.5.2), we obtain however a stronger dependence in α. Indeed, we have used a Toeplitz+Hankel-like inversion in $O(n\alpha^2)$ [66], a better cost bound in $O(n\alpha^{\omega-1})$ would require to generalize the results of [8, 9] to partly regular operators.

2.6 Special matrices for genericity

In order to identify the matrices T for which the algorithms of Section 2.5 output the characteristic polynomial (Theorems 2.5.2 and 2.5.4), we use the rank of the block Hankel matrix [47]

$$\mathrm{Hk}_{m,\lceil n/m\rceil} = \left(X^\mathsf{T}T^{i+j}X\right)_{0\le i,j\le\lceil n/m\rceil-1}.$$

We indeed have the following.

Lemma 2.6.1. *Let $T \in \mathbb{K}^{n\times n}$. If rank $\mathrm{Hk}_{m,\lceil n/m\rceil} = n$ then the irreducible left and right fraction descriptions of $X^{\mathsf{T}}(x\mathrm{I}_n - T)^{-1}X$ have degree at most $\lceil n/m\rceil$. Furthermore, the determinant (made monic) of the denominator $Q \in \mathbb{K}[x]^{m\times m}$ of such a right irreducible description is the characteristic polynomial of T.*

Proof. The determinant of a denominator Q of an irreducible right fraction description of $X^{\mathsf{T}}(x\mathrm{I}_n - T)^{-1}X$ is a divisor of the characteristic polynomial of T [47, Thm 2.12], hence has degree at most n. The claims then follow from [83, Lem. 2.4] since $\mathrm{Hk}_{m,\lceil n/m\rceil}$ has maximal rank n. □

Genericity Assumptions. To apply Theorems 2.5.2 and 2.5.4, a matrix T is "sufficiently" generic if it satisfies the following assumptions:

(A1) T has generic rank profile, so that the truncated generators of $M(x)^{-1}$ can be computed fast [9, 66];

(A2) there exists an $n \times n$ submatrix $\mathrm{Hk}^{(n)}$ of $\mathrm{Hk}_{m,\lceil n/m\rceil}$ whose determinant is nonzero, so that Lemma 2.6.1 can be applied.

The genericity in the Zariski sense can be expressed either based on the coefficients of T or on its generators. Indeed, the determinant of an $n\times n$ submatrix $\mathrm{Hk}^{(n)}$ of $\mathrm{Hk}_{m,\lceil n/m\rceil}$ is a polynomial in the coefficients of T. Toeplitz and Hankel matrices have $2n-1$ independant coefficients. With non-singular displacement operators, the coefficients of a Toeplitz-like or Hankel-like matrix of displacement rank α are themselves polynomials in the coefficients of its generators, so $\det \mathrm{Hk}^{(n)}$ is by composition a polynomial on the $2n\alpha$ coefficients of the $n \times \alpha$ generators of T.

In Sections 2.6.1 and 2.6.2, we show that we can construct an $n \times n$ submatrix $\mathrm{Hk}^{(n)}$ of $\mathrm{Hk}_{m,\lceil n/m\rceil}$ such that $\det \mathrm{Hk}^{(n)}$ is not uniformly zero on the space of Toeplitz (resp. Hankel) matrices, by finding one Toeplitz (resp. Hankel) matrix for which $\mathrm{Hk}_{m,\lceil n/m\rceil}$ has rank n. This establishes that assumption (A2) is satisfied for all matrices of each class except for those with coefficients in a certain hypersurface of $\mathbb{K}^{2n-1}$. As the displacement rank of the matrices we show is at most 2, they are Toeplitz-like (resp. Hankel-like) and can be represented with larger generators (padded with zeros). (A2) is thus also satisfied for matrices with displacement rank $\alpha \geq 2$ whose generators' coefficients are not in a certain hypersurface of $\mathbb{K}^{2n\alpha}$. The special matrices we construct are also Toeplitz+Hankel and Toeplitz+Hankel-like so the same reasoning shows that (A2) is satisfied for all Toeplitz+Hankel matrices except for those with coefficients in a certain hypersurface of $\mathbb{K}^{4n-2}$ and all Toeplitz+Hankel-like matrices with displacement rank $\alpha \geq 4$ except for those on a certain hypersurface of $\mathbb{K}^{2n\alpha}$. Using the fact that in the Toeplitz+Hankel-like case the operator is partly regular [66, Sec. 4.5], the hypersurface

can also be defined by considering the coefficients of the generators together with the irregularity set.

The generic rank profile condition (A1) can be handled similarly by considering the product Δ of the principal minors of T, though we omit details. This polynomial in the coefficients of T is nonzero for $T = \mathrm{I}_n$ in the Toeplitz case. For the Hankel case, the determinant of an $n \times n$ Hankel matrix H defined by $h_0, \ldots, h_{2n-2}$ such that $H = (h_{i+j})_{i,j}$ has a unique term in h_{n-1}^n, hence is a nonzero polynomial in the h_i's; the same holds for Δ.

From the polynomial $(\det \mathrm{Hk}^{(n)}) \cdot \Delta$ in the entries of T, one can then define the general hypersurfaces outside of which our algorithms are correct.

2.6.1 A Toeplitz Point

Let

$$\mathcal{T} = \begin{pmatrix} 0 & \mathrm{I}_m \\ \mathrm{I}_{n-m} & 0 \end{pmatrix}$$

and $M(x) = x\mathrm{I}_n - \mathcal{T}$. Let $P(x) \in \mathbb{K}[x]^{n \times m}$ defined by:

$$\begin{aligned} &P_{n-m+k,k} = 1, && \text{for } 0 \le k < m; \\ &P_{i,k} = xP_{i+m,k}, && \text{for } 0 \le k \le m, 0 \le i \le n-m-1. \end{aligned}$$

With

$$D(x) = \begin{pmatrix} 0 & x^{\lfloor n/m \rfloor} \mathrm{I}_{n \bmod m} \\ x^{\lfloor n/m \rfloor - 1} \mathrm{I}_{-n \bmod m} & 0 \end{pmatrix}$$

we can write $P(x) = \begin{pmatrix} D(x)^\mathsf{T} & R(x) & \mathrm{I}_m \end{pmatrix}^\mathsf{T}$ for some polynomial matrix R. From there we have $M(x)P(x) = \begin{pmatrix} xD(x)^\mathsf{T} - \mathrm{I}_m & 0 \end{pmatrix}^\mathsf{T}$ and thus

$$X^\mathsf{T} M^{-1}(x) X = X^\mathsf{T} P(x) \left(xD(x) - \mathrm{I}_m\right)^{-1}.$$

That is $X^\mathsf{T} M^{-1}(x) X = D(x) Q^{-1}(x)$ (we have used the form of P) with $Q(x) = xD(x) - \mathrm{I}_m$. As $(x\mathrm{I}_m) \cdot D(x) - \mathrm{I}_m \cdot Q(x) = \mathrm{I}_m$, the fraction DQ^{-1} is irreducible and

$$\det Q(x) = \pm\, x^{(\lfloor n/m \rfloor + 1)(n \bmod m) + \lfloor n/m \rfloor (-n \bmod m)} - 1$$

from which we get $\deg \det Q = n$. By [83, Lemma 2.4], $\mathrm{Hk}_{m, \lceil n/m \rceil}$ has rank n.

2.6.2 A Hankel Point

Consider the $n \times n$ Hankel matrix $\mathcal{H} = (\mathrm{I}_n + {Z_n}^m)\mathrm{J}_n$ (with $\mathrm{J}_n = (\delta_{i,n-1-j})_{0 \leq i,j \leq n-1}$). For j such that $2j \leq \lceil n/m \rceil - 1$, rows jm to $(j+1)m - 1$ of $\mathcal{H}^{2j}X$ are I_m and the following rows are 0. This can be seen by recursively applying the band matrix $\mathcal{H}^2 = {Z_n}^m + \mathrm{I}_n + {Z_n}^m {Z_n}^{m\mathsf{T}} + {Z_n}^{m\mathsf{T}}$ to X. By applying $\mathcal{H}$ to $\mathcal{H}^{2j}X$ we get that the rows $n - (j+1)m$ to $n - jm - 1$ of $\mathcal{H}^{2j+1}X$ are J_m, and the preceding rows are 0.

Let K_r be the first n columns of $\left(X|\mathcal{H}X| \ \ldots |\mathcal{H}^{\lceil n/m \rceil - 1}X\right)$. This matrix K_r is non-singular, as its columns can be permuted to get a matrix of the form

$$\begin{pmatrix} {L_1}^{\mathsf{T}} & 0 \\ 0 & L_2 \end{pmatrix}$$

where L_1 and L_2 are lower triangular with ones on the diagonal. Since $\mathcal{H}$ is symmetric, the $n \times n$ principal submatrix of $\mathrm{Hk}_{m,\lceil n/m \rceil}$ is ${K_r}^{\mathsf{T}} K_r$, hence $\mathrm{Hk}_{m,\lceil n/m \rceil}$ has rank n.

Chapter 3 | High-order lifting for polynomial Sylvester matrices

This chapter is derived from a joint work with Clément Pernet and Gilles Villard submitted for publication in the Journal of Complexity [69].

3.1 Introduction

In this chapter, we propose a new algorithm for computing the resultant of two generic bivariate polynomials over a commutative field $\mathbb{K}$. For two polynomials $p, q \in \mathbb{K}[x, y]$, the resultant $\mathrm{Res}_y(p, q)$ with respect to y is the determinant of the Sylvester matrix associated to p and q over $\mathbb{K}[x]$ (see Eq. (3.2)). The reader may refer to the books [2,24], and to [25,56] and references therein on the whole subject.

As well as for many fundamental operations on univariate polynomials over $\mathbb{K}$ (multiplication, division with remainder, multipoint evaluation, greatest common divisor, etc.), the resultant of two univariate polynomials of degree at most n can be computed using $\tilde{O}(n)$ arithmetic operations in $\mathbb{K}$ [24, Chap. 11]. (The soft-O notation is used to omit logarithmic factors: $c' = \tilde{O}(c)$ if there exists $k \in \mathbb{N}$ for which $c' = O(c \log^k c)$.) In the bivariate case, and since the early 1970's, the best known complexity bound for the computation of the resultant in the general case is $\tilde{O}(n^2 d)$ for p, q of degree bounded by d in x and n in y [24, Chap. 11]. The resultant with respect to y is a polynomial of degree at most $2nd$ in $\mathbb{K}[x]$, we therefore see that the latter bound is within a factor of the order of n from the input/output size.

Usual solutions for the resultant of two polynomials are most often based on the extended Euclidean algorithm and polynomial remainder sequences [25, 56, 73].

By going beyond this path, complementary reductions of the complexity gap with respect to the input/output size were recently obtained [37, 83, 84]. These approaches exploit the properties of appropriate families of polynomials in the ideal $\mathcal{I} = \langle p, q \rangle$ in $\mathbb{K}[x, y]$ or structured matrix operations (see Section 3.1.3), and rely on genericity assumptions on p and q in the Zariski sense. Throughout the chapter, a property is generic if it holds

except on a hypersurface of the corresponding parameter space.

Given bivariate polynomials p, q of degree d in x and n in y, it is shown in [83] that the resultant $\mathrm{Res}_y(p, q)$ can be computed generically with respect to p and q using $\tilde{O}(n^{2-1/\omega}d)$ arithmetic operations, where $\omega \leq 3$ is a feasible exponent for the cost of square matrix multiplication (two $n \times n$ matrices over a ring can be multiplied using $O(n^\omega)$ arithmetic operations). For example in the case $d = n$ this gives the first subcubic complexity estimate for the problem. The algorithm is mainly based on polynomial matrix operations and our work builds upon it (see Section 3.1.1.1).

On the other hand, using a bit complexity model and in the specific case of a finite field $\mathbb{F}$, consider p and q of respective total degrees $n_1 \geq n_2$. If p and q are sufficiently generic then $\mathrm{Res}_y(p, q)$ can be computed in expected time $O((n_1 n_2 \log |\mathbb{F}|)^{1+\epsilon}) + \tilde{O}(n_1^2 \log |\mathbb{F}|)$ using a randomised Las Vegas algorithm [37] (a few more details are given in Section 3.1.3). Even if limited to certain fields, the latter bound is a major milestone since it is quasi-linear in the input/output size. It has been extended in [84] to the case of degree conditions on x and y individually (other situations than the one with the total degree). However, it is unclear to us whether these approaches, which use a bit complexity model, could be exploited for general fields.

Despite these recent advances, we see that the algebraic complexity question of lowering the exponent of the complexity estimate for the resultant over a general field $\mathbb{K}$ remains a long-standing open problem.

The starting point of our progress is to notice that, at least for $d = 1$ and p and q with a particular shape, the approach of [83] can be improved, and a better complexity bound can be obtained. Indeed if $p = x - a$ and $q = f$ for a, f univariate in $\mathbb{K}[y]$, then the resultant can be obtained from the characteristic polynomial χ_a of the multiplication by a modulo f. For such a and f of degree n, let $c = (-1)^n e^n$ where e is the leading coefficient of f, then we have (see for instance [2, Thms 4.26 (resultant from the roots of p and q) & 4.69 (Stickelberger)]:

$$\mathrm{Res}_y(p, q) = c\, \chi_a. \tag{3.1}$$

It follows that the resultant can be computed generically with respect to a using $\tilde{O}(n^{(\omega+2)/3})$ arithmetic operations from the characteristic polynomial algorithm of [63, Sec. 10.1]. In this special case, the latter algorithm is the first one that allows to break the barrier 3/2 in the exponent of n. The cost bound can be slightly improved using rectangular matrix multiplication [63].

A key ingredient for obtaining the better estimate $\tilde{O}(n^{(\omega+2)/3})$ compared to $\tilde{O}(n^{2-1/\omega})$ in [83] for $d = 1$, is the possibility of setting up a baby steps/giant steps strategy from the powers of a modulo f [67]. One of the main difficulties that we overcome is the

development of such a strategy for polynomials p and q having degree d and no special shape.

When d is not too large in relation to n our new algorithm also allows to cross the 3/2 barrier in the exponent of n for the general resultant. As long as $d = O(n^{1/3})$, what we get is reconciled with the case $d = 1$, indeed we establish that the resultant of two sufficiently generic polynomials can be computed using $\tilde{O}(n^{(\omega+2)/3}d)$ arithmetic operations. One might also expect a slight improvement by using fast rectangular matrix multiplication [54, 55, 85]. This would require technical developments for adapting the core arithmetic on structured matrices on which we rely [8].

More precisely, we prove the following (Section 3.8):

Theorem 3.1.1. *Let $p, q \in \mathbb{K}[x, y]$ be of degree d in x and n in y. Except if the coefficients of p and q are on a certain hypersurface of $\mathbb{K}^{2(n+1)(d+1)}$, Algorithm* StructuredResultant *computes the resultant of p and q with respect to y using:*

- $\tilde{O}(n^{(\omega+2)/3}d)$ *arithmetic operations in $\mathbb{K}$ if $d = O(n^{1/3})$;*
- $\tilde{O}(n^{\theta}d^{\tau})$ *arithmetic operations with $\theta = \frac{\omega^2-2}{3\omega-4}$ and $\tau = \frac{5\omega-6}{3\omega-4}$, otherwise.*

With the known bound $\omega < 2.372$ [1, 18, 85] and $d = O(n^{1/3})$, the cost of the algorithm is $O(n^{1.458}d)$. In particular, as long as $d = O(n^{0.47})$ our complexity estimate improves on the best previous one for generic polynomials over an arbitrary field, namely $\tilde{O}(n^{2-1/\omega}d)$ [83] (see Fig. 3.1 in Section 3.8).

3.1.1 Tools from previous works

We elaborate our algorithm from three complementary points of view. In this section we present the algorithmic ideas they each bring and that we combine.

3.1.1.1 Minor of the inverse and matrix fraction reconstruction

From this point on, we will rather use n to denote the dimension of the Sylvester matrix, which corresponds, in the context of Theorem 3.1.1, to polynomials p and q of y-degree $n/2$. In a more general way, let p, $q \in \mathbb{K}[x, y]$ be polynomials of x-degree bounded by d, and respective y-degrees n_p and n_q, with $n = n_p + n_q$. Given a polynomial $t \in \mathbb{K}[y]$, we denote by $t^{(j)}$ its coefficient in y^j (and by t_j the coefficient in x^j of a polynomial $t \in \mathbb{K}[x]$).

The Sylvester matrix

$$S = \begin{bmatrix} p^{(n_p)} & & & q^{(n_q)} & & \\ p^{(n_p-1)} & \ddots & & q^{(n_q-1)} & \ddots & \\ \vdots & & p^{(n_p)} & \vdots & & q^{(n_q)} \\ p^{(0)} & & p^{(n_p-1)} & q^{(0)} & & q^{(n_q-1)} \\ & \ddots & \vdots & & \ddots & \vdots \\ & & p^{(0)} & & & q^{(0)} \end{bmatrix} \in \mathbb{K}[x]^{n\times n} \tag{3.2}$$

associated to p and q is formed by two adjacent Toeplitz matrices such that with $i = 1,\dots,n$, $S_{i,j} = p^{(n_p+j-i)}$ for $j = 1,\dots,n_q$, $S_{i,j+n_q} = q^{(n_q+j-i)}$ for $j = 1,\dots,n_p$, and the remaining entries are zero. The resultant $\mathrm{Res}_y(p,q) \in \mathbb{K}[x]$ of p and q is the determinant of S. In the same vein as [83] our algorithm reduces the computation of this determinant to the computation of the determinant of a smaller matrix, while controlling the x-degree of the latter. The Sylvester matrix is assumed to be invertible for $x = 0$.

Given a parameter $m \ll n$, chosen at the end for optimizing the overall cost, consider the projections $X = [\mathrm{I}_m\ 0]^\mathsf{T}$ and $Y = [0\ \mathrm{I}_m]^\mathsf{T}$ in $\mathbb{K}^{n\times m}$ where $\mathrm{I}_m \in \mathbb{K}^{m\times m}$ is the identity matrix (its dimension will be omitted when clear from context). The first step of the algorithm of [83] consists in computing sufficiently many terms of the power series expansion of $X^\mathsf{T}S^{-1}Y$. Coprime matrices $N, D \in \mathbb{K}[x]^{m\times m}$, with D nonsingular, such that

$$X^\mathsf{T}S^{-1}Y = ND^{-1} \tag{3.3}$$

are then deduced in a second step using matrix fraction reconstruction [3, 26]. For $n_p = n_q = n/2$, let $\lambda = 4\lceil n/(2m)\rceil$. Generically in p and q, λd terms of the expansion of $X^\mathsf{T}S^{-1}Y$ are sufficient for the computation of an $m \times m$ denominator matrix D of degree $\lambda d/2$ (Section 3.8.1). The total size of D is therefore $O(mnd)$, which for $m \ll n$ is below the previously known complexity bound $\tilde{O}(n^2 d)$ for the resultant. The computation of the truncated expansion of $X^\mathsf{T}S^{-1}Y$ can be obtained from four solutions of Sylvester linear systems modulo $x^{\lambda d}$, using $\tilde{O}(n \times \lambda d) = \tilde{O}(n^2 d/m)$ arithmetic operations [83]. By looking at the determinants of the denominator matrices in Eq. (3.3), exploiting genericity, and by involving appropriate properties of irreducible fractions [41, Lem. 6.5-9, p. 446], the desired resultant is then obtained from

$$\mathrm{Res}_y(p,q) = \det S = c \det D$$

for some $c \in \mathbb{K}$ (a scalar obtained separately at a negligible cost). The determinant of D is computed using a fast algorithm on polynomial matrices [52, 77]. This general strategy

is suitable for handling more general structures than that of the Sylvester matrix, such as that of Toeplitz-like and Hankel-like matrices [49].

3.1.1.2 Characteristic polynomials and baby steps/giant steps approach

Equations of the type of Eq. (3.3) are at the heart of block Krylov-Wiedemann schemes for the computation of minimal or characteristic polynomials of scalar matrices, see [47] and references therein. For $A \in \mathbb{K}^{n\times n}$ and projections $U, V \in \mathbb{K}^{n\times m}$ $(1 \leq m \leq n)$, the central core of these schemes is the computation of coprime matrices $N, D \in \mathbb{K}[x]^{m\times m}$, with D nonsingular, such that

$$U^{\mathsf{T}}(x\mathrm{I} - A)^{-1}V = \sum_{k\geq 0} U^{\mathsf{T}}A^k V x^{-k-1} = ND^{-1}. \tag{3.4}$$

The matrix fraction ND^{-1} is obtained from a truncated expansion using fraction reconstruction as mentioned previously, or by computing minimal polynomials of matrix sequences [47, 48]. The invariant factors of the denominator matrix D then provide information on those of $x\mathrm{I} - A$ [47, Thm 2.12]. In particular, for generic U and V, the characteristic polynomial $\det(x\mathrm{I} - A)$ can be recovered from the determinant of D as soon as it coincides with the minimal polynomial.

An advantage here is given by the shape of $x\mathrm{I} - A$ compared to the situation of Eq. (3.3) where the entries of S are general polynomials. Using the explicit form of the expansion of $(x\mathrm{I} - A)^{-1}$, a baby steps/giant steps strategy can be used from the powers of A to compute sufficiently many $U^{\mathsf{T}}A^kV$ terms. For $\lambda = 2\lceil n/m \rceil$ terms, consider $r = \lceil \lambda^{1/2} \rceil$, $s = \lceil \lambda/r \rceil$ and the precomputation of A^r. The $U^{\mathsf{T}}A^{i+rj}V$ terms for $0 \leq i < r$ and $0 \leq j < s$ can be computed by [47]:

- getting $U^{\mathsf{T}}A^i$ for $i = 0, 1, \ldots, r-1$ by repeated multiplications by A^{T} (baby steps); (3.5a)
- getting $A^{jr}V$ for $j = 0, 1, \ldots, s-1$ by repeated multiplications by A^r (giant steps); (3.5b)
- multiplying $(U^{\mathsf{T}}A^i)(A^{jr}V)$ for $i = 0, 1, \ldots, r-1$ and $j = 0, 1, \ldots, s-1$. (3.5c)

In relation with the special resultant case with $d = 1$ seen at Eq. (3.1), let $a, f \in \mathbb{K}[y]$ with $\deg f = n$, and let $A \in \mathbb{K}^{n\times n}$ be the matrix of multiplication by a modulo f in the basis $(1, y, \ldots, y^{n-1})$. The above baby steps/giants steps approach can be made efficient using modular polynomial operations. Generically in a (A is invertible, and its minimal polynomial and characteristic polynomial coincide), and using for technical reasons an

expansion

$$U^{\mathsf{T}}(x\mathrm{I} - A)^{-1}V = \sum_{k\geq 0} -U^{\mathsf{T}}A^{-k-1}Vx^k = ND^{-1} \tag{3.6}$$

at zero rather than at infinity, this leads to a fast algorithm for computing the characteristic polynomial of a modulo f [63, Sec. 10.1]. It can be shown that λ terms of the expansion in Eq. (3.6) are sufficient for the reconstruction of a suitable description ND^{-1}. Further, these terms can be computed using $\tilde{O}(m^{(1-\omega)/2}n^{(\omega+1)/2})$ arithmetic operations, which is less than the estimate $\tilde{O}(n^2/m)$ of Section 3.1.1.1 for $d = 1$ as soon as $\omega < 3$.

3.1.1.3 Series solutions of polynomial linear systems

Consider $M \in \mathbb{K}[x]^{n\times n}$, $V \in \mathbb{K}[x]^{n\times m}$ $(1 \leq m \leq n)$ and $z \in \mathbb{K}[x]$ such that $\gcd(\det M, z) = 1$. For any given integer $\lambda \geq 0$, the high-order lifting method of [77] allows to compute the truncated z-adic expansion of $M^{-1}V$ modulo z^λ. For an integer $k \geq 0$, the z-adic expansion of $M^{-1}V$ is written in the form

$$M^{-1}V = \text{lower order terms} + z^k M^{-1}R_k \tag{3.7}$$

where R_k is a polynomial matrix called *residue* of V at order k. Noticing that computing the z-adic expansion of $M^{-1}R_k$ is a problem of the same type as computing that of $M^{-1}V$, the idea is to compute the expansion of $M^{-1}V$ recursively using residues of V at various orders.

We denote by $\mathbb{K}[x]_{<d}$ the set of polynomials in $\mathbb{K}[x]$ and degree less than d. If $\deg M \leq \deg z = d$ and $V \in \mathbb{K}[x]^{n\times m}_{<d}$, then the $\mathbb{K}$-linear map

$$\rho : \mathbb{K}[x]^{n\times m}_{<d} \to \mathbb{K}[x]^{n\times m}_{<d}$$

that sends V to $\rho(V) = R_1$ is well defined, and R_k is obtained from the functional power ρ^k as $\rho^k(V)$ (Lemmas 3.2.2 and 3.2.3).

A central point of the high-order lifting method is that the order of a residue can be efficiently increased from k to $k+i$, for an integer $i \geq 0$, using only two consecutive terms of the z-adic expansion of M^{-1}. These two terms, whose orders depend on i only, form a matrix $E^{(i)} \in \mathbb{K}[x]^{n\times n}_{<2d}$ called *high-order component* of M^{-1} that can be computed using $\tilde{O}(n^\omega d)$ arithmetic operations if $i \in O(n)$ [77, Prop. 12]. For two polynomials $f, g \in \mathbb{K}[x]$ with $\deg f \leq 2d$, $\deg g \leq d$ and $fg = \sum_{k=0}^{3d} h_k x^k$ (recall we write indices for coefficients in x of polynomials, as a distinction with superscripts for coefficients in y), let us denote by

$$f \odot g = h_d + h_{d+1}x + \ldots + h_{2d-1}x^{d-1} \tag{3.8}$$

the *middle product* operation. For univariate polynomial matrices F and G with appropriate dimensions and degrees at most $2d$ and d respectively, the middle product $F \odot G$ is defined to be the matrix obtained by extracting the middle coefficients of the entries of FG. Then we have (Lemma 3.2.5):

$$\rho^i(\rho^k(V)) = \rho^{k+i}(V) \equiv M(E^{(i)} \odot \rho^k(V)) \bmod z. \tag{3.9}$$

The ability to increase the residue orders thanks to high-order components leads to the following iteration [77, Sec. 8], in the style of the one of Keller-Gehrig for Krylov subspaces [51, Sec. 3]:

$$\rho^{2^i}([V, \rho(V), \rho^2(V), \ldots, \rho^{2^i-1}(V)]) = [\rho^{2^i}(V), \rho^{2^i+1}(V), \rho^{2^i+2}(V), \ldots, \rho^{2\cdot 2^i-1}(V)],\ i = 0, 1, \ldots \tag{3.10}$$

By considering $i = 0, 1, \ldots \lceil \log \lambda \rceil - 1$, this iteration allows to compute the residues $R_k = \rho^k(V)$ of V at all orders up to $\lambda - 1$. According to Eq. (3.9) only $\lceil \log \lambda \rceil$ high-order components of M^{-1} are required. Finally, we see from Eq. (3.7) that the coefficients of the z-adic expansion of $M^{-1}V$ modulo z^λ are obtained from the residues as $M^{-1}R_k \bmod z$ for $k = 0, 1, \ldots, \lambda - 1$. All the ingredients of the lifting needed for our algorithm are recalled in Section 3.2.

3.1.2 Overview of the contribution

Our resultant algorithm follows the line of Section 3.1.1.1 and is completely described and analyzed in Section 3.8. For technical reasons (simplification of the giant steps) we slightly modify the projections and rather consider the fraction $Y^\mathsf{T} S^{-1} X$. The different aspects of our contribution are about the reduction of the cost of the first step that computes $\lambda d = 4\lceil n/(2m) \rceil d$ terms of the x-adic expansion of $Y^\mathsf{T} S^{-1} X$. The other steps are treated in the same way as in [83] and are not discussed in this section.

Assuming that $\det S(0) \neq 0$ (see Section 3.8), we use the high-order lifting tools with $z = x^d$ and compute the z-adic expansion of $Y^\mathsf{T} S^{-1} X$ modulo z^λ. For $M = S$ and $V = X$, the high-order lifting method focuses on an expansion of $S^{-1}X$. If we consider that each coefficient of a power of x in the expansion has size $\Omega(n)$ elements in $\mathbb{K}$, then the output has size $\Omega(n^2 d/m)$ and this lower bound will also apply to the time complexity of this approach. This lower complexity bound is reached by the algorithm of [83]., up to logarithmic factors. In order to get a better complexity estimate, we design a baby steps/giant steps version of the lifting which takes into consideration the left projection by Y. Based on the role played by A in Section 3.1.1.2, we introduce the high-order component $E^{(r)}$ of S^{-1} for $r = \lceil \lambda^{1/2} \rceil$. Rather than handling all the residues up to order

$\lambda - 1$ as in Section 3.1.1.3, we consider $s = \lceil \lambda / r \rceil$ of them. We first implement *giant steps* and compute the residues

$$\rho^r(X), \rho^{2r}(X), \ldots, \rho^{(s-1)r}(X) \tag{3.11}$$

using a Keller-Gehrig iteration of the type of the one of Eq. (3.10) (Lemma 3.2.7). From Eq. (3.7) and for $0 \leq j \leq s-1$, these residues satisfy

$$Y^{\mathsf{T}} S^{-1} X = \text{lower order terms} + z^{jr} Y^{\mathsf{T}} S^{-1} \rho^{jr}(X), \tag{3.12}$$

where we have taken $\rho^0(X) = X$. Our algorithm then obtain the target z-adic expansion of $Y^{\mathsf{T}} S^{-1} X$ in the form of s successive pieces of length r. Following Eq. (3.12), the piece that gives the coefficients of $z^{jr}, z^{jr+1}, \ldots, z^{(j+1)r-1}$ in the expansion of $Y^{\mathsf{T}} S^{-1} X$ is indeed the result of the multiplication modulo z^r, of the projection $Y^{\mathsf{T}} S^{-1}$ by the residue $\rho^{jr}(X)$. For taking advantage of fast polynomial matrix multiplication, these multiplications for $0 \leq j \leq s-1$ are performed by cutting $Y^{\mathsf{T}} S^{-1} \bmod z^r$ itself into r pieces. Obtaining these pieces corresponds to the *baby steps*.

This approach is detailed in Section 3.2, by highlighting the relations between block-Krylov and high-order lifting points of view. We give a template of the final expansion algorithm which at this stage does not take into account the structure of the matrices that are manipulated (Algorithm ProjectedExpansion).

Next, a main contribution is to exploit the fact that S is a polynomial Sylvester matrix and its consequences on the other matrices involved. The class of structure that we are facing is the one of *Toeplitz-like polynomial matrices*, we recall all necessary tools around this class in Section 3.3. Toeplitz-like matrices over a field are commonly handled using the notion of *displacement rank* [42]. The notion allows to have a concise matrix representation through which matrix arithmetic can be implemented efficiently [66]. Typically, by extending the ΣLU form defined over fields [43], a polynomial Toeplitz-like matrix $T \in \mathbb{K}[x]^{n \times n}$ can be represented as

$$T(x) = \sum_{i=1}^{\alpha} L_i(x) U_i(x) \tag{3.13}$$

for a parameter α "small" compared to n, and where the L_i's and the U_i's are respectively lower and upper triangular polynomial Toeplitz matrices (Section 3.3). If α is minimal then α is precisely what is called displacement rank of T, and corresponds to the displacement rank over the field of rational fractions. We show in Section 3.4 that under genericity conditions on p and q (assumptions (A1) to (A3) in Section 3.8.2), the polynomial matrices involved in the high-order lifting with S are Toeplitz-like (the residues and

the high-order components). They further have displacement rank at most $d+2$ with a ΣLU representation as in Eq. (3.13) of degree at most d. Since these matrices have dimension n, using the structure is cost-effective when $d = o(n)$. This leads us to consider that $d < n$ for what follows.

The ΣLU form with factored terms in Eq. (3.13) is however not fully appropriate for the reduction modulo x^d (remember that $z = x^d$) and middle product operations as in Eq. (3.9). For example, deriving the ΣLU representation of T modulo x^d from the one of T may necessitate explicit matrix products to reconstitute the summands in Eq. (3.13). We show that a stronger representation can actually be used in our case, especially for the high-order components of S^{-1} (Section 3.4.2) and the residues (Section 3.4.3). Truncations and middle products are facilitated by the fact that for every i in Eq. (3.13), either L_i ot U_i can be chosen as a scalar matrix. Under the genericity conditions we have mentioned above, this representation is defined uniquely and is qualified as *canonical*. Moreover, a Toeplitz-like by Toeplitz-like matrix product generally leads to an increase in displacement rank (see Lemma 3.3.1). In contrast, the matrices involved in the lifting maintain the same structure and displacement rank. Canonical representations can thus be either computed directly or recovered by compressing results of products, thereby allowing to keep the costs contained. We describe in Section 3.5 how we can rely on canonical representations for all matrix operations involved.

Once efficient matrix arithmetic is available, we implement the giant steps for the computation of the residues of Eq. (3.11) in Section 3.6, then the expansion algorithm in Section 3.7.

From there, the complexity estimate for the whole resultant algorithm is established in Section 3.8, where we also specify the genericity hypotheses. The displacement rank that quantifies the structure on which we rely and from which we can benefit, is a function of the degree bound d. This is what determines the range $d = O(n^{0.471})$ for which our complexity estimate improves over previous ones.

3.1.3 Related questions: resultants, characteristic polynomials and bivariate ideals

As seen in a special case with Eq. (3.1), the resultant problem is related to the problem of computing characteristic polynomials in quotient algebras. This relation has been used in particular for the diamond product in [7], and for the resultant algorithms in [37, 84] which have already been quoted in the introduction.

The algorithm in [37] resulted in the first quasi-linear complexity bound over finite fields $\mathbb{F}$ for sufficiently generic p and q, with respect to the total degree. It relies on the *concise representation* of a Gröbner basis of the bivariate ideal $\mathcal{I} = \langle p, q \rangle$ [36]. Such a

representation has size $\tilde{O}(n^2)$ elements in $\mathbb{F}$, and allows multiplication in $\mathbb{K}[x,y]/\mathcal{I}$ in quasi-linear time [36]. This fast multiplication then leads to an efficient reduction of the resultant problem to a bivariate modular composition problem, in turn reduced to a multivariate multipoint evaluation problem [37]. Thanks to efficient multipoint evaluation algorithms over finite fields [4, 50], a quasi-linear bit complexity bound is established for the resultant. This work has been extended to the case of generic polynomials of degree d in x and n in y in [84], by developing a method of multiplication in $\mathbb{K}[x,y]/\mathcal{I}$ which uses polynomial matrix division.

Our improvement for a general field $\mathbb{K}$ is based on a structured polynomial matrix formalism. We note however that the general approach we follow, as well as the characteristic polynomial algorithm of the case $d = 1$, can be interpreted in terms of operations on bivariate polynomials, see [83, Sec. 7] and [63, Sec. 1.6.2]. The denominator matrices D in Eqs. (3.3), (3.4) and (3.6) indeed give sets of m small degree polynomials in the corresponding ideals $\mathcal{I}$.

3.1.4 Model of computation and notations

Throughout this chapter, $\mathbb{K}$ is an effective field. We analyze our algorithms by bounding the number of arithmetic operations from $\mathbb{K}$ required for large enough inputs. Addition, subtraction, multiplication and nonzero division are considered as unit cost operations. Our complexity bounds are often given as a function of ω, which is a feasible exponent for square matrix multiplication [1, 18, 85].

Given a bivariate polynomial $t \in \mathbb{K}[x,y]$, we can either view it as a polynomial in y, $t = \sum_j t^{(j)} y^j$ with $t^{(j)} \in \mathbb{K}[x]$ or as a polynomial in x, $t = \sum_i t_i x^i$ where $t_i \in \mathbb{K}[y]$. In what follows, $t_i^{(j)}$ denotes the constant coefficient for monomial $x^i y^j$. As we will sometimes use simultaneously x-adic and x^d-adic representations, we introduce in such situations a dot over the coefficient variable when it refers to the x-adic representation (Sections 3.4.2 and 3.4.3). If t is a power series in $\mathbb{K}[[x]]$ then the notation $t \bmod x^k$ for $k \geq 0$ indicates that the terms of degree k or higher are ignored.

The m-th canonical vector will be denoted by e_m, as its dimension is always clear from the context. For a matrix $M \in \mathbb{K}^{m \times n}$, and two sets of row and column indices $I \subseteq \{1 \ldots m\}, J \subseteq \{1 \ldots n\}$, we denote by $M_{I,J}$ the submatrix of M formed by the rows of indices in I and columns of indices in J. Similarly, $M_{i,k..l}$ will denote the submatrix of M formed by row i and columns comprised between k and l.

3.2 Baby steps/giant steps for high-order lifting

A key ingredient of our resultant algorithm is the high-order lifting method of Storjohann [77]. Throughout this section, M is a nonsingular matrix in $\mathbb{K}[x]^{n\times n}$, and $z \in \mathbb{K}[x]$ is of degree $d > 0$ such that $\gcd(z, \det M) = 1$ (the resultant algorithm uses $z = x^d$).

For a given $V \in \mathbb{K}[x]^{n\times m}$, the aim of the lifting is the efficient computation of parts of the z-adic expansion

$$M^{-1}V = B_0 + B_1 z + B_2 z^2 + \ldots.$$

Two types of matrices play a central role for this computation: the residue terms (Definition 3.2.1) and the high-order components of the inverse (Eq. (3.20)). These notions, as well as all the elements which are the basis of the approach and that we need later on, are detailed in Section 3.2.1 where we essentially follow [77]. We however propose an adaptation of the definition of the residues, based on a linear map (Lemma 3.2.3), which allows us to highlight the relations between high-order lifting and Krylov iteration points of view.

For efficiency reasons we then also consider a left projection $U \in \mathbb{K}^{n\times m}$, and focus on computing parts of the expansion

$$U^{\mathsf{T}}M^{-1}V = H_0 + H_1 z + H_2 z^2 + \ldots.$$

In section Section 3.2.2 we introduce Algorithm ProjectedExpansion for computing this expansion up to an arbitrary order, by using a baby steps/giant steps strategy. Our approach somewhat interpolates between the power series expansion algorithms of [77], and the block Krylov-Wiedemann approaches mentioned in Section 3.1.1.2.

At this stage we do not take into account the structure of the matrices that are manipulated. Algorithm ProjectedExpansion should be seen as a template of which our structured expansion algorithm of Section 3.7 is a specialisation. We therefore delay the complexity analyses of the algorithms presented here to Sections 3.5, 3.6 and 3.7 where the structure of the matrices is detailed.

3.2.1 High-order lifting

Given any $h \in \mathbb{K}(x)$ whose denominator is coprime with z, we consider its z-adic expansion $h = h_0 + h_1 z + h_2 z^2 + \ldots$ and define two operations. For an integer $k \geq 0$,

$$\lceil h \rceil_k = h_0 + h_1 z + \ldots + h_{k-1} z^{k-1}$$

corresponds to the truncation operation, and

$$\lfloor h \rfloor_k = h_k + h_{k+1}z + h_{k+2}z^2 \dots .$$

denotes the quotient of the division of h by z^k. These notations are extended to matrices entry-wise. The core of high-order lifting is to compute the expansion of $M^{-1}V$ recursively from intermediate terms called *residues* (the residues play a role analogous to the one of residual terms in e.g. numerical iterative refinement [34, Chap. 12]).

Definition 3.2.1. *([77, Dfn. 5]) For $V \in \mathbb{K}[x]^{n\times m}$ and an integer $k \geq 0$, the matrix $R_k \in \mathbb{K}[x]^{n\times m}$ such that*

$$M^{-1}V = \lceil M^{-1}V \rceil_k + z^k M^{-1} R_k \tag{3.14}$$

is called the residue of V at order k.

With appropriate degree conditions, obtaining the residue at order $k = 1$ may be viewed as the application of a linear map ρ.

Lemma 3.2.2. *If $d = \deg z$ and $\deg M \leq d$ then the residue at order 1 induces the $\mathbb{K}$-linear map*

$$\begin{array}{rccc} \rho: & \mathbb{K}[x]^{n\times m}_{<d} & \to & \mathbb{K}[x]^{n\times m}_{<d} \\ & V & \mapsto & \lfloor V - M\lceil M^{-1}V \rceil_1 \rfloor_1. \end{array} \tag{3.15}$$

Proof. From Definition 3.2.1, $\rho(V)$ is the residue of V at order 1. The map is well defined since from [77, Cor. 10] we know that with the assumptions the residue has degree less than d; ρ is a $\mathbb{K}$-linear map by linearity of the operations $\lceil \cdot \rceil_1$ and $\lfloor \cdot \rfloor_1$. □

Then, the residue at order k is obtained from the functional power ρ^k.

Lemma 3.2.3. *Assume $\deg M \leq \deg z = d$. If $k \geq 0$ and $V \in \mathbb{K}[x]^{n\times m}_{<d}$ then $\rho^k(V)$ is the residue of V at order k.*

Proof. We have $\rho^0(V) = V$ and $\rho(V)$ is the residue at order 1. We proceed by induction and assume that ρ_k is the residue at order k. For $k+1$ we get $M^{-1}\rho^{k+1}(V) = M^{-1}\rho(\rho^k(V)) = \lfloor M^{-1}\rho^k(V) \rfloor_1$, where the second equality is from Eq. (3.14) (which can be written $\lfloor M^{-1}V \rfloor_k = M^{-1}R_k$) with $k = 1$. The induction hypothesis and Eq. (3.14) at order k then leads to $M^{-1}\rho^{k+1}(V) = \lfloor \lfloor M^{-1}V \rfloor_k \rfloor_1 = \lfloor M^{-1}V \rfloor_{k+1}$, which proves the assertion. □

If $M(x) = x\mathrm{I}n - A$ with A nonsingular in $\mathbb{K}^{n\times n}$, $z = x$, and $V \in \mathbb{K}^{n\times m}$, then the residue of V is $\rho(V) = A^{-1}V$ and $\rho^k(V) = A^{-k}V$. The expansion (see Eq. (3.6) in Section 3.1.1.2)

$$(x\mathrm{I}n - A)^{-1}V = \sum_{k\geq 0} -A^{-k-1}Vx^k$$

is generalised as follows. Taking $C_0 = \lceil M^{-1} \rceil_1$, for $V \in \mathbb{K}[x]^{n \times m}_{<d}$ the z-adic expansion of $M^{-1}V$ is

$$M^{-1}V = \sum_{k \geq 0} \lceil C_0 \rho^k(V) \rceil_1 z^k. \tag{3.16}$$

Since $\rho^k(V)$ is the residue at order k, we indeed know from Eq. (3.14) that $\lceil C_0 \rho^k(V) \rceil_1 = \lceil M^{-1} \rho^k(V) \rceil_1$ is the coefficient of z^k in the z-adic expansion of $M^{-1}V$.

We see from Eq. (3.16) that the role of the matrix powers in Krylov type methods can now be assigned to the powers of ρ, hence we now focus on how to increase the order of a given residue. Using the notation

$$M^{-1}V = \sum_{k \geq 0} B_k z^k$$

for the z-adic expansion of $M^{-1}V$, from Eq. (3.16) we obtain

$$\rho^k(V) = \lceil M B_k \rceil_1, \tag{3.17}$$

which reduces the computation of $\rho^{k+i}(V)$ for a given $i \geq 0$, to the computation of B_{k+i}. Note that another formulation of Eq. (3.17) could be [77, Thm. 9]:

$$\rho^k(V) = \lfloor -M B_{k-1} \rfloor_1. \tag{3.18}$$

Using Eq. (3.14) for writing the z-adic expansion of $M^{-1}\rho^k(V)$ at order $k+i$ we further have that B_{k+i} satisfies

$$M^{-1}\rho^k(V) = \lceil M^{-1}\rho^k(V) \rceil_{k+i} + z^{k+i} B_{k+i} + \ldots. \tag{3.19}$$

Then the ingredient given by Lemma 3.2.4 below is that, knowing $\rho^k(V)$, only a few terms of the expansion of M^{-1} are sufficient in Eq. (3.19) for computing B_{k+i}, and therefore then $\rho^{k+i}(V)$. These few terms form what is called a *high-order component* of M^{-1} [77, Sec. 6]. A high-order component is a piece of length 2 of the z-adic expansion of M^{-1} defined as follows. Writing $M^{-1} = \sum_{i \geq 0} C_i z^i$, we let $E^{(0)} = zC_0$ and for $i \geq 1$ we take

$$E^{(i)} = C_{i-1} + C_i z \in \mathbb{K}[x]^{n \times n}_{<2d}. \tag{3.20}$$

For two polynomial matrices F and G with appropriate dimensions, remember also the definition of the middle product operation $F \odot G = \lceil \lfloor FG \rfloor_1 \rceil_1$ (see Eq. (3.8)). Then, using $E^{(i)}$, B_{k+i} is computed as follows from $\rho^k(V)$.

Lemma 3.2.4. *Assume* $\deg M \leq \deg z = d$, *and* $\deg V < d$. *Let* $M^{-1}V = \sum_{k \geq 0} B_k z^k$. *For* $k, i \geq 0$ *we have* $E^{(i)} \odot \rho^k(V) = B_{k+i}$.

Proof. We first claim that $E^{(i)} \odot \rho^k(V)$ is the coefficient of z^i in the z-adic expansion of $M^{-1}\rho^k(V)$. For $i = 0$, $E^{(0)} \odot \rho^k(V) = \lceil C_0\rho^k(V)\rceil_1$ is the coefficient of z^0. For $i \geq 1$, the claim follows from [77, Thm. 8]. Then we conclude using Eq. (3.14) since indeed, the coefficient of z^i in the expansion of $M^{-1}\rho^k(V)$ is the coefficient of z^{k+i} in the expansion of $M^{-1}V$. □

In the way we have anticipated we can now compute $\rho^{k+i}(V)$ from $\rho^k(V)$, this operation is a main brick in [77, Algo. 3] for computing selected parts of the expansion of $M^{-1}V$. Note that from Lemma 3.2.4 we could write $B_{k+i} = E^{(0)} \odot \rho^{k+i}(V)$, hence the effect of the multiplication by M at Step 2 of Algorithm 3.2.1 is to "discard" $E^{(0)}$ from B_{k+i}.

Algorithm 3.2.1 FurtherResidue

Input: The high-order component $E^{(i)}$ of M^{-1} and the residue $\rho^k(V)$ for some $V \in \mathbb{K}[x]^{n\times m}_{<d}$, with $k, i \geq 0$
Output: The residue $\rho^{k+i}(V)$
1: $B \leftarrow E^{(i)} \odot \rho^k(V)$
2: **return** $\lceil MB\rceil_1$

Lemma 3.2.5. *Assume* $\deg M \leq \deg z = d$, *and* $\deg V < d$. *Given the residue* $\rho^k(V)$ *of order* $k \geq 0$, $i \geq 0$, *and the high-order component* $E^{(i)}$ *of* M^{-1}, *Algorithm* FurtherResidue *computes the residue* $\rho^{k+i}(V)$ *of order* $k + i$.

Proof. From Lemma 3.2.4 we know that B is the coefficient of z^{k+i} in the z-adic expansion of $M^{-1}V$, and we conclude using Eq. (3.17). □

For computing the expansion of $M^{-1}V$ efficiently as in [77, Algo. 3] as well as in our baby steps/giant steps approach, the application of Lemma 3.2.5 relies on the availability of high-order components of few selected orders (an amount logarithmic in the length of the expansion). The high-order component at some order can be computed by a sort of binary powering from components at lower orders [77, Algo. 1]. Consider indeed two high-order components $E^{(i)}$ and $E^{(j)}$. The residue ρ^i of the identity matrix involved in $E^{(i)} = (E^{(0)} \odot \rho^{i-1}(\mathrm{I})) + (E^{(0)} \odot \rho^i(\mathrm{I}))$ (Lemma 3.2.4), combined with ρ^{j-1}, ρ^j that are found in $E^{(j)}$ (after having discarded $E^{(0)}$), allows to construct the high-order component at order $i + j$.

Lemma 3.2.6. *Assume* $\deg M \leq \deg z = d$. *Given the high-order components* $E^{(i)}$ *and* $E^{(j)}$ *of* M^{-1}, *with* $i \geq 0$ *and* $j \geq 1$, *Algorithm* ComponentProduct *computes the high-order component* $E^{(i+j)}$ *of* M^{-1}.

Proof. From Eq. (3.17) we have $R_{j-1} = \rho^{j-1}(\mathrm{I})$ and $R_j = \rho^j(\mathrm{I})$, then Lemma 3.2.4 allows to conclude. □

Algorithm 3.2.2 ComponentProduct

Input: Two high-order components $E^{(i)}$ and $E^{(j)} = C_{j-1} + C_j z$ of M^{-1}, with $i \geq 0$ and $j \geq 1$
Output: The high-order component $E^{(i+j)}$ of M^{-1}
1: $R_{j-1} \leftarrow \lceil MC_{j-1} \rceil_1$
2: $R_j \leftarrow \lceil MC_j \rceil_1$
3: **return** $(E^{(i)} \odot R_{j-1}) + (E^{(i)} \odot R_j)z$

It can be noticed that Algorithm 3.2.2 is slightly different from the procedure of Storjohann in [77, Algo. 1]. The application of Eq. (3.18) rather than Eq. (3.17) at Step 1 could be used in order to compute $E^{(i+j+1)}$ from $E^{(i)}$ and $E^{(j)}$.

3.2.2 Baby steps/giant steps

We now apply the tools of Section 3.2.1 for computing parts of the z-adic expansion $U^\mathsf{T} M^{-1} V = H_0 + H_1 z + H_2 z^2 + \ldots$, where the left projection U is in $\mathbb{K}^{n \times m}$. Algorithm ProjectedExpansion is designed as an extension to the lifting context of the three phases (3.5a)-(3.5c) of the block Krylov approach.

First we focus on the giant steps, that is on the extension of step (3.5b). For some given $r, s \geq 0$, the purpose is to compute the residues $\rho^{jr}(V)$ for $j = 0, 1, \ldots s-1$. Following Storjohann [77, Sec. 8], the combination of Algorithm FurtherResidue and Algorithm ComponentProduct allows to compute such a sequence of residues *à la* Keller-Gehrig for the computation of Krylov subspaces [51, Sec. 3]. This is what Algorithm FurtherResidues does, computing s residues in $\lceil \log_2 s \rceil$ recursive steps. Taking $\varrho = \rho^r$, we proceed with an iteration of the type:

$$\varrho^{2^i}([V, \varrho(V), \varrho^2(V), \ldots, \varrho^{2^i-1}(V)]) = [\varrho^{2^i}(V), \varrho^{2^i+1}(V), \varrho^{2^i+2}(V), \ldots, \varrho^{2 \cdot 2^i-1}(V)] \quad (3.21)$$

for $i = 0, \ldots, l-1$, which generalizes the Krylov iteration

$$A^{2^i}[v, Av, A^2 v, \ldots, A^{2^i-1} v] = [A^{2^i} v, A^{2^i+1} v, A^{2^i+2} v, \ldots, A^{2 \cdot 2^i-1} v]$$

for $A \in \mathbb{K}^{n \times n}$ and $v \in \mathbb{K}^n$.

Lemma 3.2.7. *Assume* $\deg M \leq \deg z = d$. *Algorithm* FurtherResidues *is correct.*

Proof. From Lemma 3.2.6, for any given $0 \leq i \leq \lceil \log_2 s \rceil - 1$, at Step 4 we have that E is the high-order component computed at Step 6 for $i-1$, hence $E = E^{(2^{(i-1)}r + 2^{(i-1)}r)} = E^{(kr)}$. Then from Lemma 3.2.5, considering $B_{kr+jr} = E \odot \rho^{jr}(V)$ for $0 \leq j \leq k-1$, we know that $\lceil M B_{kr+jr} \rceil_1$ is $\rho^{kr+jr}(V)$, which shows that the residues of orders $kr, kr+r, \ldots, kr+(k-1)r$

Algorithm 3.2.3 FurtherResidues

Input: The high-order component $E^{(r)}$ of order r, $V \in \mathbb{K}[x]^{n\times m}_{<d}$, and $s \in \mathbb{N}_{>0}$
Output: $\mathcal{R} = [V\ \rho^r(V)\ \rho^{2r}(V)\ \ldots \rho^{(s-1)r}(V)] \in \mathbb{K}[x]^{n\times(sm)}_{<d}$

1: $E \leftarrow E^{(r)}$
2: **for** $i = 0, \ldots, \lceil \log_2 s \rceil - 1$ **do**
3: $\quad k \leftarrow 2^i$
4: $\quad \triangleright$ *New coefficients of the z-adic expansion of* $M^{-1}V$*, Lemma 3.2.4*
$\quad \mathcal{B} \leftarrow E \odot [V\ \rho^r(V)\ \rho^{2r}(V)\ \ldots, \rho^{(k-1)r}(V)]$
5: $\quad \triangleright$ *New residues, Eq. (3.17)*
$\quad [\rho^{kr}(V)\ \rho^{(k+1)r}(V)\ \rho^{(k+2)r}(V)\ \ldots\ \rho^{(2k-1)r}(V)] \leftarrow \lceil M\mathcal{B} \rceil_1$
6: $\quad \triangleright$ *Obtaining* $E^{(2^{i+1}r)}$
$\quad$ **if** $2k < s$ **then** $E \leftarrow$ ComponentProduct(E, E)
7: **return** $[V\ \rho^r(V)\ \rho^{2r}(V)\ \ldots \rho^{(s-1)r}(V)]$

are computed from those of orders $0, r, \ldots, (k-1)r$ at Step 5. This gives the residues $\rho^{jr}(V)$ for every $0 \le j \le (s-1)$ as soon as $k \ge s/2$. □

With $r = 1$, the iteration of Eq. (3.21) is dual to the one used by Storjohann for computing a truncated expansion of $M^{-1}V$ [77, Sec. 8]. As soon as $\rho^j(V)$ is known for every $0 \le j \le (s-1)$, then the truncated expansion $M^{-1}V \bmod z^s$ can be deduced using Eq. (3.16).

By relying on Algorithm FurtherResidues for the giant steps, we now have all the necessary ingredients for combining the approach of [47] for Krylov sequences with lifting techniques.

Algorithm 3.2.4 ProjectedExpansion

Input: $M \in \mathbb{K}[x]^{n\times n}$, $z \in \mathbb{K}[x]$ with $\deg M \le \deg z = d$ and $\gcd(z, \det M) = 1$,
$\quad U \in \mathbb{K}^{n\times m}$, $V \in \mathbb{K}[x]^{n\times m}_{<d}$, $r, s \in \mathbb{N}_{>0}$
Output: $\lceil U^{\mathsf{T}} M^{-1}(x) V \rceil_{rs}$

1: **for** $i = 0, \ldots, r-1$ **do** $\quad \triangleright$ *Baby steps, compare to (3.5a)*
2: $\quad D^{(i)} \leftarrow U^{\mathsf{T}} E^{(i)}$
3: Compute $E^{(r)}$
4: $[P^{(0)}, \ldots, P^{(s-1)}] \leftarrow$ FurtherResidues$(E^{(r)}, V, s)$ $\quad \triangleright$ *Giant steps, compare to (3.5b)*
5: **for** $i = 0, \ldots, r-1$ **do** $\quad \triangleright$ *Final products, compare to (3.5c)*
6: $\quad$ **for** $j = 0, \ldots, s-1$ **do**
7: $\quad\quad H_{i+rj} \leftarrow D^{(i)} \odot P^{(j)}$
8: **return** $H_0 + zH_1 + z^2H_2 + \ldots + z^{rs-1}H_{rs-1}$

Proposition 3.2.8. *Let M be a nonsingular matrix in $\mathbb{K}[x]^{n\times n}$, and $z \in \mathbb{K}[x]$ be of degree d such that $\deg M \le d$ and $\gcd(z, \det M) = 1$. Given block projections $U \in \mathbb{K}^{n\times m}, V \in \mathbb{K}[x]^{n\times m}_{<d}$, and positive integers r, s, Algorithm ProjectedExpansion computes the expansion of $U^{\mathsf{T}} M^{-1}(x) V$ truncated at order rs.*

Proof. From $P^{(0)} = V = \rho^0(V)$ and using Lemma 3.2.5, arriving at Step 5 the algorithm has computed $D^{(i)} = U^\mathsf{T} E^{(i)}$ for $0 \le i \le r-1$, and $P^{(j)} = \rho^{rj}(V)$ for $0 \le j \le s-1$. Since U has scalar coefficients we have $(U^\mathsf{T} E^{(i)}) \odot \rho^{rj}(V) = U^\mathsf{T}(E^{(i)} \odot \rho^{rj}(V))$, hence from Lemma 3.2.4 we know that $H_{i+rj} = U^\mathsf{T} B_{i+rj}$. With $0 \le i \le r-1$ and $0 \le j \le s-1$ the output gives therefore the rs appropriate terms of the expansion of $U^\mathsf{T} M^{-1} V$. □

Algorithm 3.2.4 mimics for a general $M \in \mathbb{K}[x]^{n\times n}$ what has been seen in Section 3.1.1.2 for $M = x\mathrm{I} - A$. Every step will be detailed in next sections and optimised by taking into considerations the specific structure and properties of Sylvester matrices.

3.3 Matrices with a displacement structure

We characterize the structure of the matrices which we handle using the customary notion of displacement rank [42]; the reader may refer e.g. to [32] and [66] for comprehensive overviews of the domain, and detailed introductions to the tools we use. We especially rely on [8] for the integration of asymptotically fast matrix multiplication in the algorithms.

In this chapter we define the displacement rank of a matrix from the specific Stein operator

$$\Delta_{m,n} : A \in \mathbb{K}^{m\times n} \mapsto A - Z_m A Z_n^\mathsf{T} \tag{3.22}$$

where $Z_m \in \mathbb{K}^{m\times m}$ is the lower shift matrix $(\delta_{i,j+1})_{1\le i,j\le m}$ ($\delta_{a,b}$ is 1 if $a = b$ and 0 otherwise), and Z_n^T is the transpose of $Z_n \in \mathbb{K}^{n\times n}$ (we will simply write Δ and Z when the dimensions are clear from the context). The *displacement rank* of A is defined as the rank of $\Delta(A)$, and A is called *Toeplitz-like* if its displacement rank is "small" compared to m and n. In particular, Toeplitz and Sylvester matrices are Toeplitz-like matrices of displacement rank at most 2 (see Section 3.4 for the Sylvester case).

If A has displacement rank α (or bounded by α), then a pair of matrices $G \in \mathbb{K}^{m\times\alpha}$ and $H \in \mathbb{K}^{n\times\alpha}$ such that

$$\Delta_{m,n}(A) = GH^\mathsf{T} \tag{3.23}$$

is called a *generator of length* α for A, and G and H are respectively called *left and right generator*. As soon as α is small enough, such generators (G, H) with size $(m+n)\,\alpha \ll mn$ are used as concise representations of Toeplitz-like matrices. Our central procedures for matrix operations take generators as input and return generators, in place of the corresponding matrices (Sections 3.5 and 3.6). When needed, matrices can be explicitly and uniquely recovered from their generator based representation. Indeed, the displacement operator Δ is invertible [66, Thm. 4.3.2], which means that for given G and H, Eq. (3.23)

viewed as an equation in A has a unique solution

$$A = \sum_{i=1}^{\alpha} \mathcal{L}(G_{*,i})\mathcal{U}(H_{*,i}) \tag{3.24}$$

where for $u \in \mathbb{K}^m$ and $v \in \mathbb{K}^n$, $\mathcal{L}(u) \in \mathbb{K}^{m\times m}$ is the lower triangular Toeplitz matrix whose first column is u, and $\mathcal{U}(v) \in \mathbb{K}^{m\times n}$ is the upper triangular Toeplitz matrix whose first row is v^T. The expression in Eq. (3.24) is called a ΣLU representation of A [43].

The multiplication of a Toeplitz matrix in $\mathbb{K}^{n\times n}$ by a vector in $\mathbb{K}^n$ reduces to univariate polynomial multiplication [66, Sec. 2.4], and polynomial multiplication is computed in softly linear time over any algebra [10]. From the ΣLU representation in Eq. (3.24) we therefore deduce that the multiplication of A or A^T by a scalar vector may be computed from 2α products of Toeplitz matrices by vectors. The resulting cost is $\tilde{O}(\max(m,n)\alpha)$.

An important property of Toeplitz-like matrices that we use for our algorithm is the fact that their product remains Toeplitz-like. As shown by the following, if A and B have respective displacement ranks bounded by α and β then AB has displacement rank at most $\alpha+\beta+1$.

Lemma 3.3.1 ([66, Thm 1.5.4]). *For $A \in \mathbb{K}^{l\times m}$ and $B \in \mathbb{K}^{m\times n}$ the product AB satisfies*

$$\Delta(AB) = \Delta(A)B + Z_l A Z_m^\mathsf{T}\Delta(B) - Z_l A e_m {e_m}^\mathsf{T} B Z_n^\mathsf{T}.$$

Proof. From the second item in [66, Thm 1.5.4] we have $\Delta_{l,n}(AB) = \Delta_{l,m}(A)B + Z_l A\nabla(B)$ where $\nabla(B) = Z_m^\mathsf{T}B - BZ_n^\mathsf{T}$. The assertion of the lemma follows by noticing that $\nabla(B) = Z_m^\mathsf{T}\Delta(B) - e_m e_m^\mathsf{T} B Z_n^\mathsf{T}$. □

Lemma 3.3.1 may not lead to a generator of minimal length for AB. When AB is known to have displacement rank less than $\alpha+\beta+1$, a shorter generator can be recovered by a compression mechanism [66, Sec. 4.6]. This will be the case in Section 3.5 where the compression method we use on polynomial generators also guarantees specific properties for the resulting shorter generator.

We rely on the following complexity bound for the cost of applying a Toeplitz-like matrix to a dense matrix.

Theorem 3.3.2 ([8, Theorem 1.2]). *Let $A \in \mathbb{K}^{l\times m}$ be Toeplitz-like given by a generator of length $\alpha \le \min(l,m)$ and $B \in \mathbb{K}^{m\times\beta}$. The product $AB \in \mathbb{K}^{m\times\beta}$ can be computed using $\tilde{O}(\max(l,m)\max(\alpha,\beta)\min(\alpha,\beta)^{\omega-2})$ arithmetic operations in $\mathbb{K}$.*

The following corollary expands the scope of Theorem 3.3.2 to the cases of structured matrix by dense matrix products when the structured matrix is applied on the right

and/or when the generator representation is larger than the size of the matrix (which may happen in our algorithms for extreme choices of parameters).

Corollary 3.3.3. *Let $A \in \mathbb{K}^{l\times m}$ be Toeplitz-like given by a generator of length $\alpha = O(\max(l,m))$, $B \in \mathbb{K}^{m\times\beta}$ and $C \in \mathbb{K}^{\beta\times l}$. The products $AB \in \mathbb{K}^{l\times\beta}$ and $CA \in \mathbb{K}^{\beta\times m}$ can be computed using*

$$\tilde{O}\left(\max(l,m)\max(\alpha,\beta)\min(\alpha,\beta)^{\omega-2}\right)$$

arithmetic operations in $\mathbb{K}$.

Proof. The conditions of Theorem 3.3.2 are recovered by padding the generators to appropriate dimensions. Let A be represented by (G,H) such that $GH^{\mathsf{T}} = \Delta(A)$ as in Eq. (3.23), and $\gamma = \max(l,m,\alpha)$. Construct $G' = [G^{\mathsf{T}}\ 0]^{\mathsf{T}} \in \mathbb{K}^{\gamma\times\alpha}$, $H' = [H^{\mathsf{T}}\ 0]^{\mathsf{T}} \in \mathbb{K}^{\gamma\times\alpha}$, $B' = [B^{\mathsf{T}}\ 0]^{\mathsf{T}} \in \mathbb{K}^{\gamma\times\beta}$ and consider $A' \in \mathbb{K}^{\gamma\times\gamma}$ such that $\Delta(A') = G'(H')^{\mathsf{T}}$. Then since $\gamma = O(\max(l,m))$, $AB = (A'B')_{1..l,*}$ can be computed from G', H' and B' in $\tilde{O}(\max(l,m)\max(\alpha,\beta)\min\alpha,\beta)^{\omega-2})$ by Theorem 3.3.2. Note that G', H', and B' are free to construct, and that there is no need to explicitly have A'.

If A is Toeplitz-like with generator (G,H), then A^{T} has a similar structure with generator (H,G), hence the product CA can be performed in a similar way. □

Note that by taking $\beta = 1$ in Corollary 3.3.3 we find the cost given earlier for the matrix-vector multiplication.

The previous definitions and properties are valid for any commutative field and can thus be applied to polynomial matrices, if seen over the field $\mathbb{K}(x)$. Note that in this case, generators of minimal length can always be taken as polynomial matrices themselves. Indeed, if $T \in \mathbb{K}[x]^{m\times n}$ of degree has displacement rank α (over the field of rational functions), then there exists a unimodular matrix $U \in \mathbb{K}[x]^{n\times n}$ such that $\Delta(T)U = [G\ 0]$ with $G \in \mathbb{K}[x]^{m\times\alpha}$ (consider for example the Hermite normal form of $\Delta(T)$, see e.g. [64, Ch. II, Sec. 6]). Hence a polynomial generator of minimal length for T is given by $\Delta(T) = G(U^{-1})_{1..\alpha,*}$.

Throughout the chapter, the Toeplitz-like polynomial matrices we handle are represented using polynomial generators. When the degree of the matrix is less than d, the generator we use will also be of degree less than d (see Sections 3.4.2 and 3.4.3). By extending Eqs. (3.23) and (3.24) we thus consider matrices $T \in \mathbb{K}[x]^{m\times n}_{<d}$ such that

$$\Delta_{m,n}(T) = G(x)H(x)^{\mathsf{T}},$$

where $G \in \mathbb{K}[x]^{m\times\alpha}_{<d}$ and $H \in \mathbb{K}[x]^{n\times\alpha}_{<d}$, whose ΣLU representation is

$$T = \sum_{i=1}^{\alpha} \mathcal{L}(G_{*,i}(x))\mathcal{U}(H_{*,i}(x)).$$

The products involving such matrices are treated from Theorem 3.3.2 and Corollary 3.3.3 as follows.

Theorem 3.3.4. *Let $T \in \mathbb{K}[x]_{<d}^{l\times m}$ be Toeplitz-like represented by a generator of degree less than d and length $\alpha = O(n)$ with $n = \max(l, m)$. For matrices $V \in \mathbb{K}[x]_{<d}^{m\times\beta}$ and $W \in \mathbb{K}[x]_{<d}^{\beta\times l}$, the products TV and WT can be computed using $\tilde{O}(\mathsf{MM}_{n,d}(\alpha, \beta))$ arithmetic operation in $\mathbb{K}$ with*

$$\mathsf{MM}_{n,d}(\alpha, \beta) = nd \max(\alpha, \beta) \min(\alpha, \beta)^{\omega-2}. \tag{3.25}$$

Proof. The matrix products can be computed by running the algorithm which underpins Corollary 3.3.3 with coefficients truncated modulo x^{2d-1}. In our case, [8, Thm. 1.2] relies on the algorithm `mul` of [8, Section 5.2], followed by the polynomial products in [8, Thm. 3.1, Cor. 3.2]. The only arithmetic operations performed in the case of the displacement operator $\Delta_{l,m}$ from Eq. (3.22) are additions, subtractions, multiplications, and possibly tests to zero, which can be handled in $\mathbb{K}[x]/(x^{2d-1})$ using $\tilde{O}(d)$ operations in $\mathbb{K}$ [10]. □

In particular, taking $\beta = 1$ in *Eq.* (3.25), we consider that the product of T or T^{T} by a vector with entries in $\mathbb{K}[x]_{<d}$ has cost $\tilde{O}(\mathsf{MM}_{n,d}(\alpha, 1)) = \tilde{O}(n\alpha d)$. We will keep using the notation MM in the complexity analyses when computing structured matrix products. Beyond the product, the class of nonsingular Toeplitz-like matrices is closed under inversion [42], [66, Thm. 1.5.3]. Our approach exploits this property in the special case of Sylvester matrices that are studied in detail in Section 3.4.

3.4 Displacement structure of Sylvester matrices and its residues and high-order components

For $p, q \in \mathbb{K}[x, y]$ the resultant is $\mathrm{Res}_y(p, q) = \det(S)$ where S is the associated Sylvester matrix with entries in $\mathbb{K}[x]$. By taking $z = x^d$, the first step of our resultant algorithm proceeds to do high-order lifting by implementing Algorithm ProjectedExpansion of Section 3.2 with $M = S$ and z-adic expansions. The algorithms in Section 3.2 are given for general polynomial matrices. In this section we show that in the case of a polynomial Sylvester matrix of degree at most d, the residues (Definition 3.2.1) and high-order components (Eq. (3.20)) involved are Toeplitz-like with displacement rank at most $d + 2$. This allows later in the chapter to represent these matrices by their generators, as we explained in Section 3.3. Since matrices have dimension n, the bound on the

displacement rank is useful when d is relatively smaller than n; the content of this section remains however correct for arbitrary degrees.

The polynomial case relies on properties of scalar Sylvester matrices and their inverses that are given in Section 3.4.1. However, operations such as truncation and middle product need a special attention. Especially consider the question of truncating a matrix given by its generator. Let typically $A \in \mathbb{K}[x]^{n\times n}$ be of degree $2d$ and represented by a generator (G, H), with G and H of degree d in $\mathbb{K}[x]^{n\times\alpha}$ such that $\Delta(A) = GH^{\mathsf{T}}$. Without additional assumptions, we are not aware of a way to compute a generator for the truncation $\lceil A\rceil_1$ of A modulo x^d which does not involve the reconstruction of a dense $n \times n$ matrix, and/or an expensive compression mechanism.

Our solution is obtained thanks to the fact that, in the Sylvester case, high-order components and residues can be represented by generators with a specific shape that makes the operations much easier. Indeed, it turns out that we can use a variation of the simple fact that if e.g. G (resp. H) is scalar, then a generator for $\lceil A\rceil_1$ is $(G, \lceil H\rceil_1)$ (resp. $(\lceil G\rceil_1, H)$). These specific generators which we call *canonical* are introduced in Section 3.4.2 for the coefficients of the z-adic expansion of S^{-1} and the high-order components, and in Section 3.4.3 for the residues.

3.4.1 Sylvester matrices over $\mathbb{K}$

Here we detail the structure properties we need for Sylvester matrices and their inverses over $\mathbb{K}$, the polynomial matrix case is treated using this in next sections. We consider polynomials p, $q \in \mathbb{K}[y]$ of respective degrees n_p and n_q, with $n = n_p + n_q$. The entries of the Sylvester matrix $S \in \mathbb{K}^{n\times n}$ associated to p and q are, in row $1 \le i \le n$: $S_{i,j} = p^{(n_p+j-i)}$ for $j = 1, \ldots, n_q$, $S_{i,j+n_q} = q^{(n_q+j-i)}$ for $j = 1, \ldots, n_p$, and zero otherwise (see Eq. (3.2)).

We study the structure of S by noticing that it can be viewed as a matrix of multiplication in a quotient algebra (we do not know whether this remark has been used previously). We omit the elementary proof of the following.

Lemma 3.4.1. *Consider the reverse polynomials $a = y^{n_p}p(1/y)$ and $b = y^{n_q}q(1/y)$, and define $f = b - y^{n_q}a$. If $p^{(0)} \neq 0$ then $\deg f = n$ and S is the matrix of multiplication by a modulo f in the basis $(1, y, \ldots, y^{n-1})$.*

This characterisation of Sylvester matrices allows us to highlight what makes these matrices special in the class of Toeplitz-like matrices. The fact that their displacement rank does not increase by raising to power makes the connection with the algorithm in [63] for $d = 1$ (see Eq. (3.1)), and gives an intuition about the displacement structure of high-order components and residues in the next sections. The characterisation also allows us to

directly deduce the structure of the inverse matrix, and more easily write the generators of Propositions 3.4.4 and 3.4.7 in terms of polynomial coefficients.

Matrices of modular multiplication are in particular Toeplitz-like matrices, the following recalls recurrence relations on its rows and columns, and the form of their generators (see Eq. (3.23) in Section 3.3). For $t \in \mathbb{K}[y]$, we let $v(t) = [t^{(0)}\ t^{(1)}\ \dots\ t^{(n-1)}]^\mathsf{T} \in \mathbb{K}^n$ be the vector of the coefficients of t mod y^n in the basis $(1, y, \dots, y^{n-1})$.

If T is the matrix of multiplication by t modulo f then the j-th column of T is $v(y^{j-1}t \operatorname{rem} f)$ (the rem operation returns the remainder of the Euclidean division). Further, if $f^{(0)} \neq 0$ then for any integer i one has

$$(t \operatorname{rem} f)^{(i)} = (yt \operatorname{rem} f)^{(i+1)} - c(f^{(i+1)}/f^{(0)}) \tag{3.26}$$

where c is the coefficient of degree 0 of $yt \operatorname{rem} f$. We can first deduce by replacing t with $y^j t$ for $j = 0, \dots, n-2$ in Eq. (3.26) that the rows of T follow the recursion

$$e_i^\mathsf{T} T - e_n^\mathsf{T} T_{i,n} = e_{i+1}^\mathsf{T} TZ - \frac{f^{(i)}}{f^{(0)}} e_1^\mathsf{T} TZ. \tag{3.27}$$

This recursion is used in Section 3.7 for reconstructing a whole submatrix of a multiplication matrix from only two of its rows. On other hand, we have a similar relation between the columns of T:

$$Te_{j+1} = ZTe_j - \frac{T_{n,j}}{f^{(n)}} v(f)\,, j = 1, \dots, n-1. \tag{3.28}$$

From there we deduce a generator expression for T [66, Sec. 2.7], which can therefore also be applied to S with a and f as in Lemma 3.4.1.

Lemma 3.4.2. *Given $f \in \mathbb{K}[y]$ of degree n and $t \in \mathbb{K}[y]_{<n}$, the matrix $T \in \mathbb{K}^{n\times n}$ of multiplication by t modulo f satisfies*

$$\Delta(T) = v(t)\, e_1^\mathsf{T} - v(f)\, w_t^\mathsf{T}, \tag{3.29}$$

where $w_t = ZT^\mathsf{T} e_n / f^{(n)}$.

Proof. The first column of $\Delta(T)$ is equal to the first column of T, that is $v(t)$. By definition, w_t^T is the last row of T, shifted to the right and divided by $f^{(n)}$; its first entry is thus 0, which ensures that the first column of $v(t)\, e_1^\mathsf{T} - v(f)\, w_t^\mathsf{T}$ is $v(t)$. For $j = 1, \dots, n-1$, the $(j+1)$-th column of $\Delta(T)$ is such that $\Delta(T)e_{j+1} = Te_{j+1} - ZTe_j$, hence from Eq. (3.28) we have $\Delta(T)e_{j+1} = -T_{n,j} v(f) / f^{(n)}$ and $T_{n,j}/f^{(n)}$ is precisely the $(j+1)$-th entry of w_t. □

Since S is a multiplication matrix, we know that it is invertible if and only if a is invertible modulo f, and in that case S^{-1} is the matrix of multiplication by $g \in \mathbb{K}[y]_{<n}$

such that $ag \equiv 1 \bmod f$. The following then gives useful relations between the rows 1, $n_q + 1$ and n of S^{-1}.

Lemma 3.4.3. *Assume that S is invertible with $p^{(0)} \neq 0$, and note that $q^{(n_q)} \neq 0$ by degree hypothesis. Viewing S as the matrix of multiplication by a modulo f, let $g \in \mathbb{K}[y]_{<n}$ be such that $ag \equiv 1 \bmod f$. We have*

$$e_1^\mathsf{T} S^{-1} = g^{(0)} e_1^\mathsf{T} - f^{(0)} w^\mathsf{T} \tag{3.30}$$

$$e_{n_q+1}^\mathsf{T} S^{-1} = \frac{1 - a^{(0)} g^{(0)}}{f^{(0)}} e_1^\mathsf{T} + a^{(0)} w^\mathsf{T} \tag{3.31}$$

$$e_n^\mathsf{T} S^{-1} Z^\mathsf{T} = f^{(n)} w^\mathsf{T}, \tag{3.32}$$

where $w = ZS^{-\mathsf{T}} e_n / f^{(n)}$ with the notation $S^{-\mathsf{T}}$ for the transpose of S^{-1}.

Proof. Since $f^{(0)} = q^{(n_q)}$ and $f^{(n)} = -p^{(0)}$, all the quantities are well defined. As the first row of $\Delta(S^{-1})$ is the first row of S^{-1}, Eq. (3.30) is a consequence of Lemma 3.4.2, and Eq. (3.32) is the definition of w in the same lemma. For Eq. (3.31) we use the fact that S is a Sylvester matrix. Considering $e_1^\mathsf{T} S S^{-1} = e_1^\mathsf{T}$ we have $a^{(0)} e_1^\mathsf{T} S^{-1} + b^{(0)} e_{n_q+1}^\mathsf{T} S^{-1} = e_1^\mathsf{T}$, then we note that $b^{(0)} = f^{(0)}$ and conclude using Eq. (3.30). □

3.4.2 Structure of high-order components

We move to the structured polynomial matrix case. Let now $p, q \in \mathbb{K}[x, y]$ be of degree at most $d > 0$ in x and respective degrees n_q and n_p in y, with $n = n_q + n_p$. The associated Sylvester matrix S is in $\mathbb{K}[x]_{\leq d}^{n \times n}$. Taking $z = x^d$ and assuming that $\gcd(\det S, z) = 1$, we write $S^{-1} = \sum_{k \geq 0} C_k z^k$ with $C_k \in \mathbb{K}[x]_{<d}^{n \times n}$. We thereafter also use the name *slices* for the coefficients of the z-adic expansion. From Eq. (3.20), the high-order components are formed by two consecutive slices. This section is devoted to the description of the slices, and in doing so, the one of the high-order components.

Whenever an ambiguity between x-adic and z-adic expansions may appear, we distinguish them by denoting with a dot the x-adic coefficients of a series (hence of a polynomial). In particular we have $S^{-1} = \sum_{i \geq 0} \dot{C}_i x^i$ with $\dot{C}_i \in \mathbb{K}^{n \times n}$ and $C_k = \sum_{i=0}^{d-1} \dot{C}_{kd+i} x^i$. Further, we use a bar notation for the scalar matrices involved in the generators to mark their difference from general polynomial matrices.

Rather than polynomials of $\mathbb{K}[y]$ as in Lemma 3.4.1 we now have bivariate polynomials

$$a = y^{n_p} p(1/y),\ b = y^{n_q} q(1/y),\ f = b - y^{n_q} a, \tag{3.33}$$

and we write e.g. $f = \sum_{j=0}^{d} \dot{f}_j x^j$ with $\dot{f}_j \in \mathbb{K}[y]_{\leq n}$. Recall that for bivariate polynomials, indices name coefficients in x while superscripts are used for coefficients in y. Assuming

that $p^{(0)}$ is nonzero, S is the matrix of multiplication by a modulo f. Then the identities of Lemma 3.4.3 are considered on power series, assuming $f^{(0)}$ is invertible. Using that S is invertible we know there exists $g \in \mathbb{K}(x)[y]_{<n}$ such that

$$ag \equiv 1 \bmod f.$$

Since $\gcd(\det S, z) = 1$, the x-adic and z-adic expansions of g can be considered in the form $g = \sum_{k\geq 0} g_k z^k = \sum_{i\geq 0} \dot{g}_i x^i$ where the g_k's in $\mathbb{K}[x, y]$ have respective degrees less than d in x and n in y, and the $\dot{g}_i$ are in $\mathbb{K}[y]_{<n}$. We also assume that the constant terms $q_0^{(n_q)}$ and $p_0^{(0)}$, of $q^{(n_q)}$ and $p^{(0)}$ respectively, are nonzero. Hence the expansions of $f^{(0)}$ and $f^{(n)}$ are well defined and we write

$$w = ZS^{-\mathsf{T}} e_n / f^{(n)} = \sum_{k\geq 0} w_k z^k = \sum_{i\geq 0} \dot{w}_i x^i \in \mathbb{K}[[x]]^n \quad (\dot{w}_i = 0 \text{ for } i < 0). \tag{3.34}$$

The following proposition describes the displacement structure of S^{-1} and the specific representation we take for its z-adic slices. We keep the notation previously used by associating with a polynomial t in y over an arbitrary domain of coefficients, the vector $v(t)$ of the coefficients of t mod y^n in the basis $(1, y, \ldots, y^{n-1})$.

Proposition 3.4.4. *Assume that the constant terms of* $\det S$, $p^{(0)}$ *and* $q^{(n_q)}$ *in* $\mathbb{K}[x]$ *are nonzero. Let* $\bar{F} \in \mathbb{K}^{n\times(d+1)}$ *be the matrix whose* j*-th column is* $v(\dot{f}_j)$*. For any* $k \geq 0$*, the slice* C_k *of* S^{-1} *is Toeplitz-like (over the field* $\mathbb{K}(x)$*) with displacement rank at most* $d+2$ *and one of its generators is given by*

$$\Delta(C_k) = v(g_k)\, e_1^{\mathsf{T}} - \bar{F}\, W_k^{\mathsf{T}} \in \mathbb{K}[x]_{<d}^{n\times n}, \tag{3.35}$$

where the j*-th column of* $W_k \in \mathbb{K}[x]_{<d}^{n\times(d+1)}$ *is* $\sum_{i=0}^{d-1} \dot{w}_{kd-(j-1)+i} x^i$*. A generator in this form is called canonical. The matrix* W_k *can be fully constructed in* $O(nd^2)$ *operations from its first and last column, which are the coefficients* w_k *and* w_{k-1} *of the* z*-adic expansion of* w*.*

Proof. Since S^{-1} is the matrix of multiplication by g modulo f, Eq. (3.29) gives $\Delta(S^{-1}) = v(g)\, e_1^{\mathsf{T}} - v(f)\, w^{\mathsf{T}}$, hence for $i \geq 0$ we get $\Delta(\dot{C}_i) = v(\dot{g}_i) e_1^{\mathsf{T}} - \bar{F}[\dot{w}_i\ \dot{w}_{i-1}\ \cdots\ \dot{w}_{i-d}]^{\mathsf{T}}$. Combining the $\dot{C}_j$'s into slices of size d, we obtain the z-adic coefficient in Eq. (3.35). We can check that W_k is fully determined by its first and last column. Indeed, every column of W_k is a sum of d vectors among the $\dot{w}_{(k-1)d+i}$ for $0 \leq i \leq 2d-1$, each multiplied by a distinct power of x. All these vectors appear as coefficient vectors either in the first column of W_k which is the coefficient w_k of the z-adic expansion of w, or in the last column which is w_{k-1}; the cost bound is given by the size of W_k. □

High-order components have generators directly given by those of the slices. For $k \geq 0$, the high-order component $E^{(k)}$ of S^{-1} for $z = x^d$ is indeed a sum of two slices (Eq. (3.20)). From Proposition 3.4.4 we can write

$$\Delta(E^{(k)}) = v(g_{k-1} + g_k x^d)e_1^{\mathsf{T}} - \bar{F}\,(W_{k-1}^{\mathsf{T}} + W_k^{\mathsf{T}} x^d) \in \mathbb{K}[x]^{n\times n}_{<2d}, \tag{3.36}$$

which gives a generator in canonical form for E. Generators in canonical form are uniquely defined and have properties that will be useful for lowering the computational cost. We remark that the first entry of w is zero (see Eq. (3.34)) and the first column of a canonical generator is the first column of the matrix itself. This will be exploited by separating the computation of first columns from the computation of the remaining parts of the generators. The fact that generators are polynomials only on one side allows to directly represent a truncated matrix using truncations of parts of its generator. Further, we are going to take advantage of the structure of the W_k's by restricting computations to only two of their columns.

Remark 3.4.5. *Note that $\bar{F}$ and W_k in Eq. (3.35) do not necessarily have full rank, which means that the slice may have displacement rank less than $d+2$. Genericity assumptions on p and q ensure that $\bar{F}$ has rank $d+1$ (see Section 3.8), which is used in Section 3.5 for the efficient computation of canonical generators for matrix middle products.*

3.4.3 Structure of residues

We work under the same assumptions as in Section 3.4.2 and study the displacement structure of residues (Definition 3.2.1). Since we apply high-order lifting for an expansion of $Y^{\mathsf{T}}S^{-1}X$, where $Y = [0 \;\; \mathrm{I}_m]^{\mathsf{T}}$, $X = [\mathrm{I}_m \;\; 0]^{\mathsf{T}}$ and $1 \leq m \leq n$ (see Section 3.7), only residues of the type $\rho^k(\mathrm{I})$ as in Algorithm ComponentProduct and $\rho^k(V) = \rho^k(X)$ as in Algorithm FurtherResidues are involved for integers $k \geq 0$. From Definition 3.2.1 we also see that $\rho^k(X) = \rho^k(\mathrm{I})X$, therefore noticing that any $n \times n$ matrix M satisfies $\Delta(MX) = \Delta(M)X$ (Eq. (3.22)) we deduce that

$$\Delta(\rho^k(X)) = \Delta(\rho^k(\mathrm{I}))X.$$

This allows us to limit ourselves in this section, to the description of canonical generators for residues $\rho^k(\mathrm{I})$ of the identity matrix. From Eq. (3.17), these matrices are obtained as truncated products of S and slices of S^{-1}: $\rho^k(\mathrm{I}) = \lceil SC_k \rceil_1$. The following describes the displacement structure for the scalar summands of this product. We use the notation of Section 3.4.2 for $a, b \in \mathbb{K}[x, y]$ as in Eq. (3.33) and w as in Eq. (3.34), as well as the dot convention for the x-adic coefficients of expansions and polynomials.

Lemma 3.4.6. *Assume that the constant terms of* $\det S$, $p^{(0)}$ *and* $q^{(n_q)}$ *in* $\mathbb{K}[x]$ *are nonzero. For* $i, j \in \mathbb{N}$ *the product* $\dot{S}_j \dot{C}_i \in \mathbb{K}^{n\times n}$ *is Toeplitz-like with displacement rank at most* $d+1$. *One of its generators is given by*

$$\Delta(\dot{S}_j \dot{C}_i) = \left(\dot{S}_j \dot{C}_i e_1\right) e_1^{\mathsf{T}} + \sum_{\substack{l=0 \\ l\neq j}}^{d} v^{(l,j)} \dot{w}_{i-l}^{\mathsf{T}} \tag{3.37}$$

where $v^{(l,j)} = v(\dot{a}_l \dot{b}_j - \dot{a}_j \dot{b}_l) \in \mathbb{K}^n$ *for* $l \in \mathbb{N}$.

Proof. We describe the generator for the product using Lemma 3.3.1:

$$\Delta(\dot{S}_j \dot{C}_i) = \Delta(\dot{S}_j)\dot{C}_i + Z\dot{S}_j \left(Z^{\mathsf{T}} \Delta(\dot{C}_i) - e_n e_n^{\mathsf{T}} \dot{C}_i Z^{\mathsf{T}}\right). \tag{3.38}$$

From Lemma 3.4.2 we have that $\Delta(S) = v(a)\, e_1^{\mathsf{T}} + v(f)\, e_{n_q+1}^{\mathsf{T}}$ and hence $\Delta(\dot{S}_j) = v(\dot{a}_j) e_1^{\mathsf{T}} + v(\dot{f}_j) e_{n_q+1}^{\mathsf{T}}$. Since S^{-1} is the matrix of multiplication by g modulo f, one can derive the x-adic coefficients $e_1^{\mathsf{T}} \dot{C}_i$ and $e_{n_q+1}^{\mathsf{T}} \dot{C}_i$, of $e_1^{\mathsf{T}} S^{-1}$ and $e_{n_q+1}^{\mathsf{T}} S^{-1}$, by applying Lemma 3.4.2 on power series. Here we have used the assumptions for having nonzero coefficients at $x = 0$ hence the existence of the expansions. From Eqs. (3.30) and (3.31) and for some $\bar{v} \in \mathbb{K}^n$, the first term of the sum in Eq. (3.38) can be written as

$$\Delta(\dot{S}_j)\dot{C}_i = \bar{v} e_1^{\mathsf{T}} - v(\dot{a}_j) \sum_{l=0}^{d} \dot{f}_l^{(0)} \dot{w}_{i-l}^{\mathsf{T}} + v(\dot{f}_j) \sum_{l=0}^{d} \dot{a}_l^{(0)} \dot{w}_{i-l}^{\mathsf{T}}.$$

On other hand, Lemma 3.4.2 on power series gives $\Delta(\dot{C}_i) = v(\dot{g}_i)\, e_1^{\mathsf{T}} + \sum_{l=0}^{d} v(\dot{f}_l) \dot{w}_{i-l}^{\mathsf{T}}$. Therefore from Eq. (3.32), the term being multiplied by $Z\dot{S}_j$ in Eq. (3.38) is

$$Z^{\mathsf{T}} \Delta(\dot{C}_i) - e_n e_n^{\mathsf{T}} \dot{C}_i Z^{\mathsf{T}} = Z^{\mathsf{T}} v(\dot{g}_i)\, e_1^{\mathsf{T}} - \sum_{l=0}^{d} Z^{\mathsf{T}} v(\dot{f}_l) \dot{w}_{i-l}^{\mathsf{T}} - \sum_{l=0}^{d} \dot{f}_l^{(n)} e_n \dot{w}_{i-l}^{\mathsf{T}},$$

and Eq. (3.38) becomes

$$\Delta(\dot{S}_j \dot{C}_i) = \bar{G} \left[e_1 \;\; \dot{w}_i \;\; \dot{w}_{i-1} \;\; \ldots \;\; \dot{w}_{i-d}\right]^{\mathsf{T}}$$

with a matrix $\bar{G} \in \mathbb{K}^{n\times(d+2)}$ that we now study. The first column of $\bar{G}$ is the first column of $\Delta(\dot{S}_j \dot{C}_i)$, that is $\dot{S}_j \dot{C}_i e_1$. For $l = 0, \ldots d$, the remaining columns of $\bar{G}$ can be expressed as

$$\bar{G} e_{l+2} = -\dot{f}_l^{(0)} v(\dot{a}_j) + \dot{a}_l^{(0)} v(\dot{f}_j) - Z\dot{S}_j \left(Z^{\mathsf{T}} v(\dot{f}_l) + \dot{f}_l^{(n)} e_n\right).$$

From

$$Z\dot{S}_j Z^{\mathsf{T}} = \dot{S}_j - \left(v(\dot{a}_j) e_1^{\mathsf{T}} + v(\dot{f}_j) e_{n_q+1}^{\mathsf{T}}\right),$$

we get

$$\bar{G}e_{l+2} = -\dot{f}_l^{(0)} v(\dot{a}_j) + \dot{a}_l^{(0)} v(\dot{f}_j) - \dot{S}_j v(\dot{f}_l) + v(\dot{a}_j) f_l^{(0)} + v(\dot{f}_j) e_{n_q+1}^{\mathsf{T}} v(\dot{f}_l) - \dot{f}_l^{(n)} Z \dot{S}_j e_n,$$

and since $e_{n_q+1}^{\mathsf{T}} v(\dot{f}_l) = \dot{b}_l^{(n_q)} - \dot{a}_l{}^{(0)}$ (one has $f = b - y^{n_q} a$), we arrive at

$$\bar{G}e_{l+2} = \dot{b}_l^{(n_q)} v(\dot{f}_j) - \dot{S}_j v(\dot{f}_l) - \dot{f}_l^{(n)} Z \dot{S}_j e_n. \tag{3.39}$$

Then consider each of the three summand vectors in this equation from the corresponding polynomials modulo y^n:

$$\begin{aligned}
\dot{b}_l^{(n_q)} v(\dot{f}_j) &= v\Big(\dot{b}_l^{(n_q)} \left(\dot{b}_j - y^{n_q} (\dot{a}_j \mod y^{n_p})\right)\Big), \\
\dot{S}_j v(\dot{f}_l) &= v\Big(\dot{a}_j \left(\dot{b}_l \mod y^{n_q}\right) + \dot{b}_j \left(\dot{b}_l^{(n_q)} - (\dot{a}_l \mod y^{n_p})\right)\Big), \\
\dot{f}_l^{(n)} Z \dot{S}_j e_n &= v\Big(-\dot{a}_l^{(n_p)} y^{n_p} (\dot{b}_j \mod y^{n_q})\Big).
\end{aligned}$$

By viewing Eq. (3.39) on polynomials modulo y^n this allows to assert that $\bar{G}e_{l+2} = v(\dot{a}_l \dot{b}_j - \dot{a}_j \dot{b}_l)$. Hence $\bar{G} = [\dot{S}_j \dot{C}_i e_1 \; v^{(0,j)} \; v^{(1,j)} \; \cdots \; v^{(d,j)}]$, which by noticing that $v^{(j,j)} = 0$ concludes the proof. □

Lemma 3.4.6 reveals a structure similar to the one of the slices in Proposition 3.4.4: the first column can be separated from the remaining ones; the left remaining part of the generator does not depend on the considered order in the expansion of S^{-1}; the right part of the generator is given by the z-adic coefficient w_k of w. From there we define a canonical representation for the residues which retains these properties.

Proposition 3.4.7. *Assume that the constant terms of* $\det S$, $p^{(0)}$ *and* $q^{(n_q)}$ *in* $\mathbb{K}[x]$ *are nonzero, and for indices* $i, j \in \mathbb{N}$ *consider the* $v^{(i,j)}$*'s as in Lemma 3.4.6. For any* $k \geq 0$ *and* $z = x^d$*, the residue* $\rho^k(\mathrm{I}) = \lceil SC_k \rceil_1$ *(see Eq. (3.17)) is Toeplitz-like (over the field* $\mathbb{K}(x)$*) with displacement rank at most* $d + 1$*. One of its generators is given by*

$$\Delta(\varphi^k(\mathrm{I})) = \lceil SC_k \rceil_1 e_1 e_1^{\mathsf{T}} + L \bar{W}_{k-1}^{\mathsf{T}} \in \mathbb{K}[x]_{<d}^{n \times n}, \tag{3.40}$$

where the l*-th column of* $L \in \mathbb{K}[x]_{<d}^{n \times d}$ *is* $\sum_{i=0}^{d-1} x^i \sum_{j=0}^{i} v^{(i+l-j,j)}$ *and the* l*-th column of* $\bar{W}_{k-1} \in \mathbb{K}^{n \times d}$ *is* $\dot{w}_{kd-l}$*. A generator in this form is called canonical. The left part* L *of the generator does not depend on* k *and can be computed using* $\tilde{O}(nd^2)$ *operations.*

Proof. By looking at the x-adic coefficients of $\lceil SC_k \rceil_1$ we can write

$$\Delta(\rho^k(\mathrm{I})) = \sum_{i=0}^{d-1} x^i \sum_{j=0}^{i} \Delta\left(\dot{S}_j \dot{C}_{kd+i-j}\right).$$

From Lemma 3.4.6 and since w has its first entry zero we have

$$\Delta(\rho^k(\mathrm{I})) = \rho^k(\mathrm{I})e_1e_1^\mathsf{T} + \delta,$$

where $\delta = \Delta(\rho^k(\mathrm{I}))ZZ^\mathsf{T}$ has first column zero. Hence it remains to study the structure of δ which gives the last $n-1$ columns of $\Delta(\rho^k(\mathrm{I}))$. From Lemma 3.4.6 (omitting to write $l \neq j$ since $v^{(j,j)} = 0$) and by substituting l by $i-j-l$ we obtain

$$\delta = \sum_{i=0}^{d-1} x^i \sum_{j=0}^{i} \sum_{l=0}^{d} v^{(l,j)} \dot{w}_{kd+i-j-l}^\mathsf{T} = \sum_{i=0}^{d-1} x^i \sum_{j=0}^{i} \sum_{l=i-j-d}^{i-j} v^{(i-j-l,j)} \dot{w}_{kd+l}^\mathsf{T},$$

then by swapping the sums the contribution of w can be factored out:

$$\delta = \sum_{l=-d}^{d-2} \left(\sum_{i=0}^{d-1} x^i \sum_{\substack{j=0 \\ j \leq i-l, j \geq i-l-d}}^{i} v^{(i-j-l,j)} \right) \dot{w}_{kd+l}^\mathsf{T}. \tag{3.41}$$

This sum can be divided into two parts, for $l < 0$ and $l \geq 0$. We show that the latter sum is zero. It is indeed given by

$$\sum_{l=0}^{d-2} \left(\sum_{i=0}^{d-1} x^i \sum_{\substack{j=0 \\ j \leq i-l,\ j \geq i-l-d}}^{i} v^{(i-j-l,j)} \right) \dot{w}_{kd+l}^\mathsf{T} = \sum_{l=0}^{d-2} \left(\sum_{i=l}^{d-1} x^i \sum_{j=0}^{i-l} v^{(i-j-l,j)} \right) \dot{w}_{kd+l}^\mathsf{T}. \tag{3.42}$$

Now notice that $v^{(i,i)} = 0$ and $v^{(i,j)} + v^{(j,i)} = 0$. It follows that for all l and i the summands of $\sum_{j=0}^{i-l} v^{(i-j-l,j)}$ cancel each other out (one summand is zero for even values of $i-l$), and the sum in Eq. (3.42) is zero.

By substituting l by $-l$, the nonzero terms in Eq. (3.41) then give $\delta = L\bar{W}_{k-1}^\mathsf{T}$ where for $l = 1, \ldots, d$ the l-th column of $L \in \mathbb{K}[x]_{<d}^{n \times d}$ is

$$\sum_{i=0}^{d-1} x^i \sum_{j=0, j \geq i+l-d}^{i} v^{(i+l-j,j)},$$

and $\bar{W}_{k-1}$ is as asserted (the constraints in the sum over j have vanished as $v^{(i+l-j,j)} = 0$ for $i+l-j > d$). The matrix L can then be computed in time $\tilde{O}(nd^2)$ from $O(d^2)$ products of the $\dot{a}_i$'s by the $\dot{b}_j$'s modulo y^n. □

Remark 3.4.8. *The columns of $\bar{W}_{k-1}$ in Proposition 3.4.7 are the scalar coefficient vectors of the z-adic coefficient w_{k-1}. It follows from Proposition 3.4.4 that the last d columns (the first one is e_1) of the right generator for $\rho^k(\mathrm{I})$ are the linearisation of the last column of the right generator for the slice C_k. As L can be precomputed once and used for all*

residues (Section 3.6.2), the computation of the canonical generator for $\varphi^k(\mathrm{I})$ from the one for C_k is essentially the computation of its first column.

3.5 Structured middle and truncated products

Our specialisation of high-order lifting to the Sylvester case represents all high-order components of S^{-1} and residues by their canonical generators as in Sections 3.4.2 and 3.4.3. In this section we show that these representations allow to lower the cost of the two central matrix operations on which we rely: middle and truncated products. We work with $z = x^d$ under the assumptions of Propositions 3.4.4 and 3.4.7, and consider that $0 < d < n$ (the displacement rank structure does not directly enable faster operations for degrees that reach the dimension). We keep using a bar notation for the scalar matrices involved in the generators.

Typically, consider a high-order component E of S^{-1} at some arbitrary order. From Eq. (3.36) we know that E satisfies

$$\Delta(E) = v_E e_1^\mathsf{T} + \bar{F}\, W_E^\mathsf{T} \in \mathbb{K}[x]^{n\times n}_{<2d}, \tag{3.43}$$

with $v_E \in \mathbb{K}[x]^n$, $\bar{F} \in \mathbb{K}^{n\times(d+1)}$ and $W_E \in \mathbb{K}[x]^{n\times(d+1)}$. Consider also a residue $R = \rho^k(\mathrm{I})$ at some other arbitrary order k, from Eq. (3.40) we have

$$\Delta(R) = v_R e_1^\mathsf{T} + L\, \bar{W}_R^\mathsf{T} \in \mathbb{K}[x]^{n\times n}_{<d}, \tag{3.44}$$

where $v_R \in \mathbb{K}[x]^n$, $L \in \mathbb{K}[x]^{n\times d}$, and $\bar{W}_R \in \mathbb{K}^{n\times d}$. Then a central brick of the high-order lifting approach is the computation of the middle product

$$C = E \odot R \in \mathbb{K}[x]^{n\times n}_{<d},$$

where, according to Lemma 3.2.4, C is a coefficient of the z-adic expansion of S^{-1}. Hence using the canonical form Eq. (3.35) again, we know there exists a matrix $W_C \in \mathbb{K}[x]^{n\times(d+1)}_{<d}$ such that

$$\Delta(C) = v_C e_1^\mathsf{T} + \bar{F} W_C^\mathsf{T} \in \mathbb{K}[x]^{n\times n}_{<d}, \tag{3.45}$$

with $v_C \in \mathbb{K}[x]^n_{<d}$. For the computation of C, we exploit the fact that the generator parts $\bar{F}$ and $\bar{W}_R^\mathsf{T}$ are scalar matrices, which allows us to perform the middle product without resorting to a change of representation (Lemma 3.5.2). Further, several middle products will follow one another. We therefore ensure that the resulting C is itself represented by its canonical generator, which by the way also avoids the increase of the sizes of the representations (inherent, in general, to the product of structured matrices, see Section 3.3).

We note that the first column of a left canonical generator (e.g. v_E, v_R, v_C) is the first column of the matrix itself. In addition, the last $n-1$ columns of the displaced matrices of Eqs. (3.43) to (3.45) are associated with the "W" parts $W_E, \bar{W}_R, W_C$ of the generators (whose first rows are zero). This leads us to separate the computation of first columns of generators from the computation of their W parts. In Section 3.5.1 we start by focusing on the computation of the right generator part W_C of the middle product as in Eq. (3.45); Lemma 3.5.1 actually deals with a slightly more general situation that is required later for the concatenation of several products. An additional advantage is that, according to Proposition 3.4.4, the right generator for a slice is determined by its first and last column. This allows us to further decrease the exponent of d in the complexity bound, and compute these two columns of W_C using $\tilde{O}(\mathsf{MM}_{n,d}(d,1))$ operations with $\mathsf{MM}_{n,d}$ from Eq. (3.25). This is essentially the cost of multiplying a matrix of displacement rank d in $\mathbb{K}[x]^{n\times n}_{<d}$ by a vector in $\mathbb{K}[x]^{n}_{<d}$. Here the left part $\bar{F}$ of the generator for the slices will be assumed to have rank $d+1$, which corresponds to generic situations (see Remark 3.4.5).

The whole generator for C is deduced in Section 3.5.2, also in time $\tilde{O}(\mathsf{MM}_{n,d}(d,1))$ (Lemma 3.5.2). We will use for that (computation of the first column), and in some other places, the fact that for a vector b, the middle product $E \odot b$ does not cause any difficulty compared to the matrix middle product. Indeed, $E \odot b$ can simply be computed from the regular product Eb, by extracting the middle coefficients.

Besides the middle product, high-order lifting involves the truncated product operation as for instance at Step 5 of Algorithm FurtherResidues or Steps 1 and 2 of Algorithm ComponentProduct. These truncated products are used for the computation of residues. From Remark 3.4.8 the right generator for a residue $\lceil SC \rceil_1$ is directly known from that of C, and in the same way as for a slice of the inverse, the first column can be computed separately. Combining this with the middle product, in Section 3.5.2 we derive the cost bound $\tilde{O}(\mathsf{MM}_{n,d}(d,1))$ for high-order components handling (Lemma 3.5.3).

3.5.1 Middle product: computation of the right generator

Given a high-order component $E \in \mathbb{K}[x]^{n\times n}_{<2d}$ of S^{-1}, we consider the computation of a right generator for a middle product $E \odot \mathcal{R}$, for $\mathcal{R} \in \mathbb{K}[x]^{n\times c}_{<d}$ a residue or a concatenation or several residues. The latter case is addressed for performing the giant steps which we will discuss in Section 3.6.

We let $\mathcal{E}$ be a matrix in $\mathbb{K}^{c\times s}$ whose columns are s distinct canonical vectors $e_{i_1}, \ldots, e_{i_s}$ such that $1 \le i_1, \ldots i_s \le c$, and $\bar{\mathcal{W}}_{\mathcal{R}}$ be a matrix in $\mathbb{K}^{c\times d}$ whose submatrix $(\bar{\mathcal{W}}_{\mathcal{R}})_{I,*} \in \mathbb{K}^{s\times d}$ is zero for $I = \{i_1, \ldots, i_s\}$. Then generalizing Eq. (3.44), we consider $\mathcal{R}$ such that

$$\Delta(\mathcal{R}) = V_{\mathcal{R}}\mathcal{E}^{\mathsf{T}} + L\bar{\mathcal{W}}_{\mathcal{R}}^{\mathsf{T}} \in \mathbb{K}[x]^{n\times c}_{<d}, \tag{3.46}$$

with $V_{\mathcal{R}} \in \mathbb{K}[x]^{n\times s}_{<d}$. For $c = n$ and $s = 1$ we are in the case of a unique residue as in Eq. (3.44). We assume that the middle product $\mathcal{B} = E \odot \mathcal{R}$ satisfies

$$\Delta(E \odot \mathcal{R}) = V_{\mathcal{B}}\mathcal{E}^{\mathsf{T}} + \bar{F}\mathcal{W}_{\mathcal{B}}^{\mathsf{T}} \tag{3.47}$$

with $V_{\mathcal{B}} \in \mathbb{K}[x]^{n\times s}_{<d}$ and $\mathcal{W}_{\mathcal{B}} \in \mathbb{K}[x]^{c\times(d+1)}_{<d}$, and focus on the computation of the first and last column of $\mathcal{W}_{\mathcal{B}}$. Note that assuming the generator form as in Eq. (3.47) is appropriate for covering the case $s = 1$. This indeed generalizes the canonical form for a single slice as in Eq. (3.45). Focusing on the first and last column of the generator part is to be put in correspondence with the last assertion of Proposition 3.4.4.

From Lemma 3.3.1, for the product $E\mathcal{R}$ we have

$$\Delta(E\mathcal{R}) = \Delta(E)\mathcal{R} + ZEZ^{\mathsf{T}}\Delta(\mathcal{R}) - ZEe_ne_n^{\mathsf{T}}\mathcal{R}Z^{\mathsf{T}},$$

which can also be decomposed according to

$$\Delta(E\mathcal{R}) = v_E(e_1^{\mathsf{T}}\mathcal{R}) + \bar{F}(W_E^{\mathsf{T}}\mathcal{R}) + (ZEZ^{\mathsf{T}}V_{\mathcal{R}})\mathcal{E}^{\mathsf{T}} + (ZEZ^{\mathsf{T}}L)\bar{\mathcal{W}}_{\mathcal{R}}^{\mathsf{T}} - (ZEe_n)(e_n^{\mathsf{T}}\mathcal{R}Z^{\mathsf{T}}), \tag{3.48}$$

where we have kept the notation $\Delta(E) = v_E e_1^{\mathsf{T}} + \bar{F}\, W_E^{\mathsf{T}}$ introduced in Eq. (3.43).

We then focus on the $c - s$ columns of $\Delta(E \odot \mathcal{R})$ with indices in $\bar{I} = \{1, \ldots, c\} \setminus I$. Since $\bar{F}$ and $\bar{\mathcal{W}}_{\mathcal{R}}$ have entries in $\mathbb{K}$, and using that $\Delta(E \odot \mathcal{R}) = \lceil \lfloor \Delta(E\mathcal{R}) \rfloor_1 \rceil_1$, these columns can be deduced from Eq. (3.48) in the following form:

$$\Delta(E\odot\mathcal{R})_{*,\bar{I}} = v_E \odot (e_1^{\mathsf{T}}\mathcal{R})_{\bar{I}} + \bar{F}\cdot(W_E^{\mathsf{T}}\odot\mathcal{R}_{*,\bar{I}}) + (ZEZ^{\mathsf{T}}\odot L)\cdot(\bar{\mathcal{W}}_{\mathcal{R}}^{\mathsf{T}})_{*,\bar{I}} - (ZEe_n)\odot(e_n^{\mathsf{T}}\mathcal{R}Z^{\mathsf{T}})_{\bar{I}}. \tag{3.49}$$

The right hand side of Eq. (3.49) can be rewritten as a generator. For the second term we simply let $\bar{G}_1 = \bar{F} \in \mathbb{K}^{n\times(d+1)}$ and $H_1^{\mathsf{T}} = \lceil \lfloor W_E^{\mathsf{T}}\mathcal{R}_{*,\bar{I}} \rfloor_1 \rceil_1 \in \mathbb{K}[x]^{(d+1)\times(c-s)}$. Then the first and last terms in Eq. (3.49) are partially linearised. For a polynomial vector $r = \sum_{i=0}^{2d-1} r_i x^i \in \mathbb{K}[x]^n_{<2d}$, we denote by M_r the matrix in $\mathbb{K}[x]^{n\times d}_{<d}$ whose j-th column is $\sum_{i=0}^{d-1} r_{d+i-j+1}x^i$. Likewise, for $t = \sum_{i=0}^{d-1} t_i x^i \in \mathbb{K}[x]^{c-s}_{<d}$ we denote by $\bar{M}_t$ the matrix in $\mathbb{K}^{(c-s)\times d}$ whose j-th column is t_{j-1}. We these notations we have

$$r \odot t^{\mathsf{T}} = M_r \cdot \bar{M}_t^{\mathsf{T}}. \tag{3.50}$$

Applying Eq. (3.50) allows to write the last $c-s$ columns of $v_E \odot (e_1^{\mathsf{T}}\mathcal{R})$ and $-(ZEe_n) \odot (e_n^{\mathsf{T}}\mathcal{R}Z^{\mathsf{T}})$ as $G_2\bar{H}_2^{\mathsf{T}}$ and $G_3\bar{H}_3{}^{\mathsf{T}}$, with $G_2, G_3 \in \mathbb{K}[x]^{n\times d}_{<d}$ and $\bar{H}_2, \bar{H}_3 \in \mathbb{K}^{(c-s)\times d}$. Finally, we let $G_4 = \lceil \lfloor ZEZ^{\mathsf{T}}L \rfloor_1 \rceil_1 \in \mathbb{K}[x]^{n\times d}_{<d}$ and take $\bar{H}_4^{\mathsf{T}} = (\bar{\mathcal{W}}_{\mathcal{R}}^{\mathsf{T}})_{*,\bar{I}}$. The construction leads to

the generator expression:

$$\Delta(E \odot \mathcal{R})_{*,\bar{I}} = \left(\begin{array}{c|c|c|c} \bar{G}_1 & G_2 & G_3 & G_4 \end{array}\right) \left(\begin{array}{c} {H_1}^{\mathsf{T}} \\ \hline \bar{H}_2^{\mathsf{T}} \\ \hline \bar{H}_3^{\mathsf{T}} \\ \hline \bar{H}_4^{\mathsf{T}} \end{array}\right) \in \mathbb{K}[x]_{<d}^{n \times (c-s)}. \quad (3.51)$$

Now, $\mathcal{W}_{\mathcal{B}}$ as in Eq. (3.47) can be obtained by compression of above right-hand side matrices. Let $G = [G_2, G_3, G_4] \in \mathbb{K}[x]_{<d}^{n \times 3d}$ and $\bar{H} = [\bar{H}_2, \bar{H}_3, \bar{H}_4]^{\mathsf{T}} \in \mathbb{K}^{(c-s) \times 3d}$. From Eq. (3.51) we indeed have

$$\bar{F}\,(\mathcal{W}_{\mathcal{B}}^{\mathsf{T}})_{*,\bar{I}} = \bar{F}\,H_1^{\mathsf{T}} + G\,\bar{H}^{\mathsf{T}},$$

hence

$$(\mathcal{W}_{\mathcal{B}}^{\mathsf{T}})_{*,\bar{I}} = H_1^{\mathsf{T}} + \bar{U}^{-1} G_{J,*}\,\bar{H}^{\mathsf{T}}, \quad (3.52)$$

where we assume that $\bar{F}$ has rank $d+1$ and $\bar{U}$ is a $(d+1) \times (d+1)$ nonsingular submatrix of $\bar{F}$ constructed from row indices forming the set J. Since the rows of $\mathcal{W}_{\mathcal{B}}$ whose indices are in I are zero, the first and last columns of $\mathcal{W}_{\mathcal{B}}$ can be fully obtained from Eq. (3.52), the nonzero rows of these columns are given by:

$$e_i^{\mathsf{T}}(\mathcal{W}_{\mathcal{B}}^{\mathsf{T}})_{*,\bar{I}} = e_i^{\mathsf{T}} H_1^{\mathsf{T}} + e_i^{\mathsf{T}} \bar{U}^{-1} G_{J,*}\,\bar{H}^{\mathsf{T}} \in \mathbb{K}[x]_{<d}^{c-s}, \text{ for } i \in \{1, d+1\}. \quad (3.53)$$

Lemma 3.5.1. *For $0 < d < n$, assume that the inverse of a $(d+1) \times (d+1)$ submatrix of $\bar{F}$ as in Proposition 3.4.4 is given, with the corresponding set of row indices J. From a high-order component E of S^{-1} and $\mathcal{R} \in \mathbb{K}[x]_{<d}^{n \times c}$ with $c = O(n)$, respectively given by their generators as in Eq. (3.43) and Eq. (3.46), one can compute the first and the last column of $\mathcal{W}_{\mathcal{B}}$ for $E \odot \mathcal{R}$ as in Eq. (3.47) using $\tilde{O}(\mathsf{MM}_{n,d}(s+d,1)) = \tilde{O}\,(nd(s+d))$ operations.*

Proof. For $i = 1$, the first term $e_1^{\mathsf{T}} H_1^{\mathsf{T}}$ in Eq. (3.53) is computed from $e_1^{\mathsf{T}} W_E^{\mathsf{T}} \in \mathbb{K}[x]^n$ by multiplication by $\mathcal{R}$, then extraction of the middle coefficients. From Eq. (3.46), $\mathcal{R}$ is seen as a Toeplitz-like matrix of degree less than d and displacement rank bounded by $s+d$, which from Theorem 3.3.4 gives a cost $\tilde{O}(\mathsf{MM}_{n,d}(s+d,1))$ for those first computations.

The target cost bound is valid for obtaining the generators $G_2, \bar{H}_2, G_3$ and $\bar{H}_3$ from E and $\mathcal{R}$. This is indeed equivalent to having the first (v_E is part of the generators) and last column of E, and the first and last row of $\mathcal{R}$. The required products involving canonical vectors and E and $\mathcal{R}$ can be computed in time $\tilde{O}(\mathsf{MM}_{n,d}(s+d,1))$.

Then, let $u \in \mathbb{K}^{d+1}$ be the first row of $\bar{U}^{-1}$. From $u^{\mathsf{T}}(G_2)_{J,*}$ and $u^{\mathsf{T}}(G_3)_{J,*}$ in $\mathbb{K}[x]^d$, $u^{\mathsf{T}}(G_2)_{J,*}\bar{H}_2^{\mathsf{T}}$ and $u^{\mathsf{T}}(G_3)_{J,*}\bar{H}_3^{\mathsf{T}}$ are deduced in time $O(nd^2)$ by matrix times vector prod-

ucts using that $\bar{H}_2^\mathsf{T}$ and $\bar{H}_3^\mathsf{T}$ are matrices in $\mathbb{K}^{d\times(c-s)}$ and $c = O(n)$. Here, recall that $\mathsf{MM}_{n,d}(d,1) = nd^2$. It thus remains to verify the cost bound for the computation of $u^\mathsf{T}(G_4)_{J,*}\bar{H}_4^\mathsf{T}$. Since u has scalar entries we can first compute $u^\mathsf{T}(ZEZ^\mathsf{T})_{J,*} \in \mathbb{K}[x]^n$, then multiply the result by $L \in \mathbb{K}[x]^{n\times d}_{<d}$, and multiply the middle coefficients of the latter by $\bar{H}_4$ using a total of $\tilde{O}(nd^2)$ operations. What we have just said with e_1 is also valid with e_{d+1}, which concludes the proof. □

The inversion of an appropriate submatrix of $\bar{F}$ that is required for Lemma 3.5.1 will be done only once for all products in Section 3.6.2.

3.5.2 High-order components

By combining Lemma 3.5.1 in the case $s = 1$ and a direct computation of the first column we can perform the middle product of a high-order component by a residue as shown by next lemma.

Lemma 3.5.2. *Under the assumptions of Lemma 3.5.1 for $\bar{F}$, consider a high-order component E of S^{-1} and a residue $R = \rho^k(\mathrm{I})$ for some $k \geq 0$, both represented by their generators as in Eq. (3.43) and Eq. (3.44). A generator for $E \odot R$ as in Eq. (3.45) can be computed using $\tilde{O}(\mathsf{MM}_{n,d}(d,1)) = \tilde{O}(nd^2)$ operations.*

Proof. The first column $v_C = \lceil \lfloor Ev_R \rfloor_1 \rceil_1$ of C can be computed by applying E to v_R in $\tilde{O}(\mathsf{MM}_{n,d}(d,1))$ operations, and by extracting the middle coefficients. The assertion of the lemma then follows from Lemma 3.5.1 with $s = 1$ and $i_1 = 1$ for the computation of the first and last column of W_C, and from Proposition 3.4.4 for the whole generator. □

Given a slice C of S^{-1}, the right generator for the residue at the same order is obtained using Remark 3.4.8 and allows to manipulate high-order components in the following way.

Lemma 3.5.3. *Under the assumptions of Lemma 3.5.1 for $\bar{F}$, we consider further that L as in Proposition 3.4.7 is given. For two high-order components $E^{(i)}$ and $E^{(j)}$ with $i \geq 0$ and $j \geq 1$, both represented by their generators as in Eq. (3.43), Algorithm ComponentProduct computes generators having the same shape for the high-order component $E^{(i+j)}$ using $\tilde{O}(\mathsf{MM}_{n,d}(d,1)) = \tilde{O}(nd^2)$ operations.*

Proof. We keep the notation used for Algorithm ComponentProduct. Since $\bar{F}$ is a scalar matrix, the generators for $E^{(j)}$ directly give generators for C_{j-1} and C_j, which from Proposition 3.4.4 give in particular w_{j-2} and w_{j-1}. Therefore from Remark 3.4.8 we have right generators for R_{j-1} and R_j, which are residues of order $j-1$ and j, respectively. The first columns of C_{j-1} and C_j are available from the generators, by truncated multiplication by S these two columns give the first columns of R_{j-1} and R_j. Hence the whole generators for R_{j-1} and R_j are known and we finally apply Lemma 3.5.2 twice. □

3.6 Giant steps

In order to perform the giant steps (Step 4 in Algorithm ProjectedExpansion) we specialize Algorithm FurtherResidues to the Sylvester case. The successive products that involve high-order components and concatenated residues as in Eq. (3.21) are implemented thanks to middle and truncated products. As before, high-order components and residues are represented by their generators as those in Sections 3.4.2 and 3.4.3. Their respective orders do not matter in this section. We work under the assumptions of Propositions 3.4.4 and 3.4.7, and following Section 3.5 we take $0 < d < n$. The representation for concatenated matrices, which for technical reasons is a little bit different, is specified in Section 3.6.1.

Taking advantage of the special shape of the generators we can split up the middle products into regular matrix products for obtaining their left parts, and apply the strategy of Section 3.5.1 for computing their right parts. Then, truncated products are used for the computation of residues from slices of the inverse according to Eq. (3.17). Remark 3.4.8 actually implies that the right generator parts for residues are directly deduced from those of the slices; in the same way as for middle products we compute their left generator parts by regular matrix product.

We proceed to high-order lifting with the projection $V = X = [\mathrm{I}_m\ 0]^{\mathsf{T}}$, where $1 \leq m \leq n$. Considering a number s of giant steps, our purpose in this section is to bound the cost of a call to FurtherResidues with input the high-order component E of order r of S^{-1}, which computes the matrix

$$\mathcal{R} = \left[X\ \rho^r(X)\ \rho^{2r}(X)\ \ldots \rho^{(s-1)r}(X)\right] \in \mathbb{K}[x]_{<d}^{n\times(sm)}. \tag{3.54}$$

We first study in Section 3.6.1 the two central products at Steps 4 and 5 of Algorithm FurtherResidues, and then bound the overall cost in Section 3.6.2.

3.6.1 Concatenated middle and truncated products

Let us first specify the representation we use for the block residue matrices involved. Noting that the residue map satisfies $\rho^k(X) = \rho^k(\mathrm{I})X$, for $k = 1, 2, 3, \ldots, 2^{l-1}$ the right operand at Step 4 of Algorithm FurtherResidues is of the type

$$\mathcal{P} = \left[P^{(0)}X\ \ P^{(1)}X\ \ \cdots\ P^{(k-1)}X\right] \in \mathbb{K}[x]_{<d}^{n\times km}, \tag{3.55}$$

where for $j \geq 0$ each $P^{(j)} \in \mathbb{K}[x]^{n\times n}$ is some residue of the identity. The displacement operator applied to such a $\mathcal{P}$ gives

$$\Delta(\mathcal{P}) = \mathcal{P} - Z_n \mathcal{P} Z_{km}^{\mathsf{T}} = \mathcal{P} - \Big[(Z_n P^{(0)} X Z_m^{\mathsf{T}}) \quad \ldots \quad (Z_n P^{(k-1)} X Z_m^{\mathsf{T}})\Big] - \sum_{j=1}^{k-1} Z_n P^{(j-1)} e_m \epsilon_{1+jm}^{\mathsf{T}}, \tag{3.56}$$

here, in order to avoid confusion, we have $e_m \in \mathbb{K}^n$ and we use ϵ_{1+jm} for the canonical vectors in $\mathbb{K}^{km}$. On the other hand, from Eq. (3.40) we can write

$$\Delta(P^{(j)} X) = \Delta(P^{(j)}) X = p_j e_1^{\mathsf{T}} + L \bar{W}_{P^{(j)}}^{\mathsf{T}}, \tag{3.57}$$

where p_j is the first column of $P^{(j)}$, $L \in \mathbb{K}[x]_{<d}^{n\times d}$, and $\bar{W}_{P^{(j)}} \in \mathbb{K}^{m\times d}$. Equations (3.56) and (3.57) then give

$$\Delta(\mathcal{P}) = V_{\mathcal{P}} \mathcal{E}^{\mathsf{T}} + L \bar{\mathcal{W}}_{\mathcal{P}}^{\mathsf{T}}, \tag{3.58}$$

such that: $V_{\mathcal{P}} \in \mathbb{K}[x]^{n\times k}$ has first column p_0 and column j being $p_{j-1} - Z_n P^{(j-2)} e_m$ for $2 \leq j \leq k$; $\mathcal{E} \in \mathbb{K}^{km\times k}$ has column j being $\epsilon_{1+(j-1)m}$; $\bar{\mathcal{W}}_{\mathcal{P}} \in \mathbb{K}^{km\times d}$ has k blocks of rows with j-th block being $\bar{W}_{P^{(j)}}$.

For a high-order component E of S^{-1}, let $\mathcal{B} = E \odot \mathcal{P}$. Then the resulting matrix $\mathcal{Q} = \lceil S\mathcal{B} \rceil_1$ at Step 5 of Algorithm FurtherResidues involves residues Q^j at further orders for $j = 0, \ldots, k-1$ such that

$$\Delta(Q^{(j)} X) = q_j e_1^{\mathsf{T}} + L \bar{W}_{Q^{(j)}}^{\mathsf{T}}, \tag{3.59}$$

where q_j is the first column of $Q^{(j)}$ and $\bar{W}_{Q^{(j)}} \in \mathbb{K}^{m\times d}$. In accordance with Eq. (3.58) we have

$$\Delta(\mathcal{Q}) = V_{\mathcal{Q}} \mathcal{E}^{\mathsf{T}} + L \bar{\mathcal{W}}_{\mathcal{Q}}^{\mathsf{T}}, \tag{3.60}$$

with: $V_{\mathcal{Q}} \in \mathbb{K}[x]^{n\times k}$ has first column q_0 and column j being $q_{j-1} - Z_n Q^{(j-2)} e_m$ for $2 \leq j \leq k$; $\bar{\mathcal{W}}_{\mathcal{Q}} \in \mathbb{K}^{km\times d}$ has k blocks of rows with j-th block being $\bar{W}_{Q^{(j)}}$.

Equations (3.58) and (3.60) lead us to represent $\mathcal{P}$ and $\mathcal{Q}$ as follows. To ensure that a canonical representation is maintained throughout the algorithm, hence following our approach in Section 3.5, we separate out left and right generator parts. The right parts are given by $\bar{\mathcal{W}}_{\mathcal{P}}$ and $\bar{\mathcal{W}}_{\mathcal{Q}}$. Besides L and from the characterisations of $V_{\mathcal{P}}$ and $V_{\mathcal{Q}}$, we slightly modify the representation of the left generator parts. They are represented by the first and m-th columns of the $P^{(j)}$'s and $Q^{(j)}$'s. This is temporarily slightly different from the canonical representation with $V_{\mathcal{P}}$ and $V_{\mathcal{Q}}$ in order to simplify the following statement.

Lemma 3.6.1. *For $d < n$, assume that the inverse of a $(d+1) \times (d+1)$ submatrix of $\bar{F}$ is given, with the corresponding row indices set J, also assume that L and*

the canonical generator for E is given. Consider $\mathcal{P}$ represented by $\{P^{(j)}e_1\}_{j=0,\dots k-1}$, $\{P^{(j)}e_m\}_{j=0,\dots k-2}$ and $\bar{\mathcal{W}}_{\mathcal{P}}$ as in Eq. (3.58). If $\mathcal{P}$ has $km = O(n)$ columns, then Steps 4 and 5 of Algorithm FurtherResidues compute $\{Q^{(j)}e_1\}_{j=0,\dots k-1}$, $\{Q^{(j)}e_m\}_{j=0,\dots k-2}$ and $\bar{\mathcal{W}}_{\mathcal{Q}}$ using $\tilde{O}(\mathsf{MM}_{n,d}(d,k))$ operations.

Proof. The specification of the output is from Lemma 3.2.7, we have to prove the cost bound. The matrix $\mathcal{B} = E \odot \mathcal{P}$ has k blocks of columns $E \odot P^{(j)}X$, each of which is the projection of a slice $C^{(j)} = E \odot P^{(j)}$ of S^{-1} (Lemma 3.2.4). From Eq. (3.35) we can thus write:

$$\Delta(E \odot P^{(j)}X) = \Delta(C^{(j)}X) = c_j e_1^{\mathsf{T}} + \bar{F}W_{C^{(j)}}^{\mathsf{T}}, \tag{3.61}$$

where c_j is the first column of $C^{(j)}$, and $W_{C^{(j)}} \in \mathbb{K}[x]_{<d}^{m\times(d+1)}$. Hence by doing the same manipulation as for $\mathcal{P}$ and $\mathcal{Q}$, we arrive at

$$\Delta(\mathcal{B}) = V_{\mathcal{B}}\mathcal{E}^{\mathsf{T}} + \bar{F}\mathcal{W}_{\mathcal{B}}^{\mathsf{T}}, \tag{3.62}$$

involving matrices such that: $V_{\mathcal{B}} \in \mathbb{K}[x]^{n\times k}$ has first column c_0 and column j being $c_{j-1} - Z_n C^{(j-2)}e_m$ for $2 \le j \le m$; $\mathcal{W} \in \mathbb{K}^{km\times(d+1)}$ has k blocks of rows with j-th block being $W_{C^{(j)}}$.

Using that $\mathcal{Q} = \lceil S\mathcal{B}\rceil_1$, we first detail the whole computation of the first and m-th columns of the $Q^{(j)}$'s from the representations of $\mathcal{P}$ and E. We then conclude with the right generator part for $\mathcal{Q}$.

For $j = 0,\dots k-1$ and $i = 1, m$ we have $C^{(j)}e_i = E \odot (P^{(j)}e_i)$. All these middle products can be computed from the regular product $E \cdot [P^{(0)}e_i \ \dots \ P^{(k-1)}e_i]$ by extraction of the middle coefficients. From Theorem 3.3.4 this can be performed using $\tilde{O}(\mathsf{MM}_{n,d}(d,k))$ operations. For all j we then have $Q^{(j)}e_i = \lceil SC^{(j)}e_i\rceil_1$, products which can computed by extraction of the coefficients of $S \cdot [C^{(0)}e_i \ \dots \ C^{(k-1)}e_i]$ within the same cost bound.

We now deduce the right generator part $\bar{\mathcal{W}}_{\mathcal{Q}}$ as in Eq. (3.60) from the corresponding $\mathcal{W}_{\mathcal{B}}$ of Eq. (3.62). The last column of $\mathcal{W}_{\mathcal{B}}$ can indeed be computed using Lemma 3.5.1: the generator for $\mathcal{P}$ given by Eq. (3.58) has the shape of the one for $\mathcal{R}$ as in Eq. (3.46); the generator for $\mathcal{B}$ in Eq. (3.62) corresponds to the one for $E \odot \mathcal{R}$ in Eq. (3.47). Remark that the application of Lemma 3.5.1 explicitly requires the generator part $V_{\mathcal{P}}$ for $\mathcal{P}$ as in Eq. (3.58): $V_{\mathcal{P}}$ can be reconstructed from $\{P^{(j)}e_1\}_{j=0,\dots,k-1}$ and $\{P^{(j)}e_m\}_{j=0,\dots,k-2}$ in time $O(nkd)$. Taking $c = km$ in Lemma 3.5.1, the last column of $\mathcal{W}_{\mathcal{B}}$ is therefore obtained in time $\tilde{O}(\mathsf{MM}_{n,d}(k+d,1))$, which is $\tilde{O}(\mathsf{MM}_{n,d}(d,k))$.

By definition of $\mathcal{W}_{\mathcal{B}}$, we now know for $j = 0,\dots,k-1$ the last column of $W_{C^{(j)}}$ given by Eq. (3.61) for the projected slice $C^{(j)}X$ of S^{-1}. For any fixed j let κ be the z-adic order of $C^{(j)}X$, and remark that $W_{C^{(j)}}$ must be the projection of the full right generator part for $C^{(j)}$. It follows from Proposition 3.4.4 that the last column of $W_{C^{(j)}}$ provides us

with the z-adic coefficient $X^\mathsf{T} w_{\kappa-1}$ of the first m entries $X^\mathsf{T} w$ of w. We conclude using Remark 3.4.8. Indeed, from Eq. (3.17), $Q^{(j)} = \lceil SC^{(j)} \rceil_1$ is the residue $\rho^\kappa(\mathrm{I})$, hence its right generator part is directly deduced from $w_{\kappa-1}$. Equivalently, by projection using X, $\bar{W}_{Q^{(j)}}$ as in Eq. (3.59) is deduced from $X^\mathsf{T} w_{\kappa-1}$ in $O(md^2)$ operations. Finally, since we have been working for an arbitrary $0 \leq j \leq k-1$, all the blocks of rows for $\bar{\mathcal{W}}_\mathcal{Q}$ as in Eq. (3.60) are obtained in $O(kmd^2)$, which does not dominate the cost. □

3.6.2 Cost bound for the giant steps

We can now bound the cost of Algorithm FurtherResidues in the case of the Sylvester matrix, with input some high-order component E of the inverse and $V = X$.

Lemma 3.6.2. *Let E be the high-order component E of order r of S^{-1} represented by its canonical generator, and assume that $\bar{F}$ has rank $d+1 \leq n$. With input E and the projection $X = [\mathrm{I}_m \ \ 0]^\mathsf{T}$ for $1 \leq m \leq n$, Algorithm FurtherResidues computes a generator as in Eq. (3.46) for the matrix $\mathcal{R} = [X \ \ \rho^r(X) \ \ \rho^{2r}(X) \ \ \ldots \rho^{(s-1)r}(X)] \in \mathbb{K}[x]_{<d}^{n \times (sm)}$. If $sm = O(n)$ the cost of the computation is $\tilde{O}(\mathsf{MM}_{n,d}(d,s))$.*

Proof. From Lemma 3.2.7 we know that the algorithm correctly computes $\mathcal{R}$. The cost bound comes from $\log s$ applications of Lemma 3.6.1 for the representations of the new residues at Steps 4 and 5 with $k \leq s$ and $(\log s) - 1$ applications of Lemma 3.5.3 for the generators for the new high-order components at Step 6. For the first application of Lemma 3.6.1, $P^{(0)} = X$ is represented using $\Delta(X) = e_1 {\epsilon_1}^\mathsf{T}$ (here ϵ_1 is the canonical vector in $\mathbb{K}^m$). Both Lemma 3.6.1 and Lemma 3.5.3 require the preliminary computation of L whose cost is $\tilde{O}(nd^2)$ operations from Proposition 3.4.7. They also rely on the inverse of a $(d+1) \times (d+1)$ submatrix of $\bar{F}$; such an inverse can be computed in $O(nd^{\omega-1})$ operations [38, 40], which does not dominate since for $d < n$ and $s \geq 1$ we have $\mathsf{MM}_{n,d}(d,s) \in \Omega(nd^2)$.

Lemma 3.6.1 uses a representation of the matrices slightly different from here. The left generator part $V_\mathcal{R}$ of Eq. (3.46) still needs to be recovered from $\{\rho^{jr}(e_1)\}_{j=0,\ldots s-1}$, $\{\rho^{jr}(e_m)\}_{j=0,\ldots s-2}$ and the right generator. From the shape of the left generator given by Eq. (3.58), we deduce that the latter can be reconstructed from $O(s)$ polynomial vector additions with cost $O(nsd)$. □

3.7 Complete expansion algorithm

From the preceding sections we have all the ingredients for giving the cost of the specialisation of Algorithm ProjectedExpansion to the Sylvester case. For $X = [\mathrm{I}_m \ 0]^\mathsf{T}$ and

$Y = [0\ \mathrm{I}_m]^\mathsf{T}$ in $\mathbb{K}^{n\times m}$, Algorithm StructuredExpansion computes the truncated expansion of $Y^\mathsf{T}S^{-1}X$ at the z-adic order rs, following the three main phases of the general approach of Section 3.2.2.

First, the baby steps are performed based on two linear system solutions computed using the seminal lifting methods of [17, 61]. (The high-order lifting as in Section 3.2 is not required for this step.) The loop at Step 1 of Algorithm ProjectedExpansion can indeed be implemented by computing the expansion of $Y^\mathsf{T}S^{-1}$ at the order r, of which coefficients give the projections of the high-order components. We actually compute only truncations of $e_1^\mathsf{T}S^{-1}$ (at order $r+1$ as it will also be used to compute a high-order component at order r) and of $e_n^\mathsf{T}S^{-1}$, since by using the recursion of Eq. (3.27), these two rows are sufficient to recover the expansion of $Y^\mathsf{T}S^{-1}X$ at the end. These polynomial vectors of degree rd are linearised as r vectors of degree d in order to take advantage of fast structured matrix multiplication.

The special structures we have identified for the high-order components and the residues are then used in the giant steps as well as in the third phase which computes the final product. This product is applied on the linearised rows of the baby steps, which are combined together in the following step. In anticipation of the reconstruction of the expansion of $Y^\mathsf{T}S^{-1}X$, the two latter phases actually use a modified projection $X' = [\mathrm{I}_{2m-1}\ 0]^\mathsf{T}$ on the right, and give the first and last row of the expansion of $S^{-1}X'$ (this trick is taken from [63, Sec. 3.4.3]). Finally, the whole target expansion of $Y^\mathsf{T}S^{-1}X$ is reconstructed using Eq. (3.27) as mentioned above.

Proposition 3.7.1. *Let p and q in $\mathbb{K}[x,y]$ be of respective y-degrees n_p and n_q, and of x-degree at most $d < n = n_p + n_q$. Assume that the constant terms of $p^{(0)}$, $q^{(n_q)}$ and $\det S$ in $\mathbb{K}[x]$ are nonzero, where $S \in \mathbb{K}[x]^{n\times n}_{\le d}$ is the Sylvester matrix associated to p and q, and also that $\bar{F} \in \mathbb{K}^{n\times(d+1)}$ as defined in Proposition 3.4.4 has rank $d+1$. For $X = [\mathrm{I}_m\ 0]^\mathsf{T}$ and $Y = [0\ \mathrm{I}_m]^\mathsf{T}$ with $2m-1 \le n$, $z = x^d$, and positive integers r, s, Algorithm StructuredExpansion computes the expansion of $Y^\mathsf{T}S^{-1}X$ modulo z^{rs}. If $s = O(r)$ and $mr = O(n)$, then it uses $\tilde{O}(\mathsf{MM}_{n,d}(r+d,r) + m^2rsd)$ arithmetic operations in $\mathbb{K}$.*

Proof. Step 1 computes truncated expansions of $e_1^\mathsf{T}S^{-1}$ and $e_n^\mathsf{T}S^{-1}$ which are used for constructing the generators for $E^{(r)}$ and for the final products. This can be done by x-adic lifting using that $S(0)$ is nonsingular [17,61]. We follow the description of the method in [17], carried over to the case $\mathbb{K}[x]$. The cost is essentially that of $O(rd)$ multiplications of the transpose inverse $S(0)^{-\mathsf{T}}$ by a vector. Using a displacement rank-based representation of $S(0)^{-\mathsf{T}}$— see e.g. [83, Sec. 5] and references therein, such a multiplication costs $\tilde{O}(n)$; this gives a total of $\tilde{O}(nrd)$ operations for Step 1. The rows of A and B are the z-adic coefficients of the computed expansions.

Algorithm 3.7.1 StructuredExpansion

Input: $p, q \in \mathbb{K}[x, y]$ of respective y-degrees n_p and n_q and of x-degree at most $d < n = n_p + n_q$, $m \leq (n+1)/2, r, s \in \mathbb{N}^*$

Assumptions: the constant terms of $p^{(0)}$, $q^{(n_q)}$ and $\det S$ in $\mathbb{K}[x]$ are nonzero, where $S \in \mathbb{K}[x]^{n \times n}_{\leq d}$ is the Sylvester matrix associated to p and q, and $\bar{F} \in \mathbb{K}^{n \times (d+1)}$ as defined in Proposition 3.4.4 has rank $d+1$.

Output: $\lceil Y^{\mathsf{T}} S^{-1} X \rceil_{rs}$ where $Y = [0 \;\; \mathrm{I}_m]^{\mathsf{T}}$ and $X = [\mathrm{I}_m \;\; 0]^{\mathsf{T}}$

1: ▷ *Baby steps*
$z \leftarrow x^d$; $a \leftarrow e_1^{\mathsf{T}} S^{-1} \bmod z^{r+1}$; $b \leftarrow e_n^{\mathsf{T}} S^{-1} \bmod z^r$
$A \leftarrow [a_0^{\mathsf{T}} \; a_1^{\mathsf{T}} \; \ldots \; a_{r-1}^{\mathsf{T}}]^{\mathsf{T}}$; $B \leftarrow [b_0^{\mathsf{T}} \; b_1^{\mathsf{T}} \; \ldots \; b_{r-1}^{\mathsf{T}}]^{\mathsf{T}}$ ▷ *Both in* $\mathbb{K}[x]^{r \times n}_{<d}$

2: ▷ *Generator for* $E = E^{(r)} \in \mathbb{K}[x]^{r \times n}_{<2d}$ *of length* $d+2$*:* $\Delta(E) = v e_1^{\mathsf{T}} + \bar{F} W^{\mathsf{T}}$
$f \leftarrow y^{n_q} q(1/y) - y^{n_q} y^{n_p} p(1/y)$ ▷ $f = \sum_{j=0}^{d} \sum_{i=0}^{n} \dot{f}_j^{(i)} x^j y^i$
$\bar{F} \leftarrow [\dot{f}_{j-1}^{(i-1)}]_{1..n, 1..d+1}$ ▷ *In* $\mathbb{K}^{n \times (d+1)}$
$v \leftarrow S^{-1} e_1 \bmod z^{r+1}$; $v \leftarrow \lfloor v \rfloor_{r-1}$
$w \leftarrow -a / f^{(0)} \bmod z^{r+1}$; $w \leftarrow [0 \; w_{2..n}]$ ▷ *See Eq.* (3.30)
Construct W from $w_{r-2} + z w_{r-1}$ and $w_{r-1} + z w_r$ ▷ *See Proposition 3.4.4 and Eq.* (3.36)

3: ▷ *Giant steps*
$m' \leftarrow 2m - 1$; $X' \leftarrow [\mathrm{I}_{m'} \; 0]^{\mathsf{T}}$
$\mathcal{R} \leftarrow (\text{FurtherResidues}(E, X', s))_{*, 1..sm'}$ ▷ *In* $\mathbb{K}[x]^{n \times (sm')}_{<d}$*, see Lemma 3.6.2*

4: ▷ *Final products*
$A' \leftarrow A \cdot \mathcal{R}$; $A'_{r,*} \leftarrow A'_{r,*} \bmod z$ ▷ *Matrices in* $\mathbb{K}[x]^{r \times sm'}_{<2d}$
$B' \leftarrow B \cdot \mathcal{R}$; $B'_{r,*} \leftarrow B'_{r,*} \bmod z$

5: ▷ *Reconstruction of the first and last row of* $\lceil S^{-1} X' \rceil_{rs} \in \mathbb{K}[x]^{n \times m'}$
$H' \leftarrow 0 \in \mathbb{K}[x]^{m'}$; $H \leftarrow 0 \in \mathbb{K}[x]^{m \times m'}$
for $i = 0, \ldots, r-1$ ▷ *See Proposition 3.7.1 and proof thereof*
  for $j = 0, \ldots, s-1$
    $H'_{1..m'} \leftarrow H'_{1..m'} + z^{i+rj} A'_{i+1, (jm'+1)..(jm'+m')}$
    $H_{m, 1..m'} \leftarrow H_{m, 1..m'} + z^{i+rj} B'_{i+1, (jm'+1)..(jm'+m')}$ ▷ *Remaining rows left to* 0

6: ▷ *Obtaining* $\lceil Y^{\mathsf{T}} S^{-1} X \rceil_{rs} \in \mathbb{K}[x]^{m \times m}$ *using truncated power series operations*
for $i = m-1, \ldots, 1$ ▷ *See Proposition 3.7.1 and proof thereof*
  $c \leftarrow m + i - 1$
  $H_{i, 1..c} \leftarrow H_{i+1, 2..c+1} - f^{(i)} (H'_{2..c+1} / f^{(0)}) \bmod z^{rs}$ ▷ *See Eq.* (3.63)

7: **return** $H_{1..m, 1..m}$

Step 2 computes the canonical generator for $E^{(r)} \equiv \lfloor S^{-1} \rfloor_{r-1} \bmod z^2$. The first column v is computed in the same way as in Step 1. +(Note that the multiplication of $S(0)^{-1}$ by a vector could also be computed by using the +extended Euclidean algorithm in $\mathbb{K}[x]$ [24, Sec. 4.5].) Then according to Eq. (3.36), the left generator part $\bar{F}$ is given by f, and the right generator requires w_{r-2}, w_{r-1} and w_r. The first entry of w is zero and from Eq. (3.30) the remaining ones modulo z^{r+1} are deduced as $(e_1^{\mathsf{T}} S^{-1})_{2\ldots n}/f^{(0)}$ using the first row of S^{-1} obtained at previous step (the inverse of $f^{(0)}$ exists from $f(0,0) = q^{(n_q)}(0) \neq 0$). From Proposition 3.4.4, this leads to the wanted z-adic coefficients of w and the generator for $E^{(r)}$ in time $\tilde{O}(nrd) + O(nd^2)$.

We then deduce from Lemma 3.6.2 that with $X' = [\mathrm{I}_{m'}\ 0]^{\mathsf{T}}$, a generator for $\mathcal{R} = [X'\ \rho^r(X')\ \rho^{2r}(X')\ \ldots \rho^{(s-1)r}(X')]$ is correctly computed in allotted time at Step 3 as $s = O(r)$. This generator is as in Eq. (3.46), hence of length at most $s + d$. Using the bounds $s = O(r)$ and thus $sm' = O(n)$ in Theorem 3.3.4, the matrix products $A \cdot \mathcal{R}$ and $B \cdot \mathcal{R}$ are computed using $\tilde{O}(\mathsf{MM}_{n,d}(r + d, r))$ operations.

Then let $a'_{i,j}$ be the $(i+1)$-th row of $A'_{*,(jm'+1)..(jm'+m')}$ at Step 4. From the product with the block of columns corresponding to $\rho^{rj}(X')$ in $\mathcal{R}$ and Definition 3.2.1 we have

$$\begin{aligned}\sum_{i=0}^{r-1} a'_{i,j} z^{i+rj} &\equiv (\sum_{i=0}^{r-1} a_i z^i)\, \rho^{rj}(X') \bmod z^r \\ &\equiv (e_1^{\mathsf{T}} S^{-1} \bmod z^r)\, \rho^{rj}(X') \bmod z^r \\ &\equiv \lfloor e_1^{\mathsf{T}} S^{-1} X' \rfloor_{rj} \bmod z^r,\end{aligned}$$

hence the sums at Step 5 give $H'_{1..m'} \equiv e_1^{\mathsf{T}} S^{-1} X' \bmod z^{rs}$; in an equivalent manner, with B' we get that the last row of H is $H_{m,1..m'} \equiv e_n^{\mathsf{T}} S^{-1} X' \bmod z^{rs}$. The cost is bounded by the one of rs additions of polynomial vectors of dimension m' and degree d, which is dominated by the previous step.

We conclude at Step 6 by following the trick in [63, Sec. 3.4.3], which consists in using the recursion given by Eq. (3.27) for reconstructing the whole expansion of $Y^{\mathsf{T}} S^{-1} X$ from those of $e_1^{\mathsf{T}} S^{-1} X'$ and $e_n^{\mathsf{T}} S^{-1} X'$. For the inverse C of S which is a matrix of multiplication and $1 \leq i, c < n$, Eq. (3.27) indeed leads to:

$$C_{i,1..c} = C_{i+1,2..c+1} - f^{(i)}(C_{1,2..c+1}/f^{(0)}). \tag{3.63}$$

From $C_{1,2..2m-1}$ and $C_{m,1..2m-1}$, given by $e_1^{\mathsf{T}} S^{-1} X'$ and $e_n^{\mathsf{T}} S^{-1} X'$, the application of Eq. (3.63) for $i = m-1, m-2, \ldots 1$ on truncated power series modulo z^{rs} provides with the m^2 entries of $Y^{\mathsf{T}} S^{-1} X$ using $\tilde{O}(m^2 rsd)$ operations. □

3.8 Resultant algorithm

Following the previous works in Section 3.1.1.1 and Section 3.1.1.2, once sufficiently many terms of the expansion of $Y^\mathsf{T} S^{-1} X \in \mathbb{K}(x)^{m\times m}$ are known then a matrix fraction description ND^{-1} with coprime matrices $N, D \in \mathbb{K}[x]^{m\times m}$ is computed. For generic polynomials p and q of degree d in x and $n_p = n_q = n/2$ in y, we recall in Section 3.8.1 how such a fraction description can be reconstructed from only $O(n/m)$ terms of the x^d-adic expansion of $YS^{-1}X$. Furthermore, the denominator matrix D that is obtained is such that its determinant is the resultant of p and q up to a scalar factor. Together with Algorithm StructuredExpansion this leads us to the resultant algorithm given in Section 3.8.2, and to the proof of Theorem 3.1.1.

3.8.1 Matrix fraction reconstruction

The number of terms sufficient for reconstructing a matrix fraction depends on the degrees of its possible descriptions. First, let us recall a few notions on matrix fractions (the reader may refer to the comprehensive material in [41, Chap. 6] and its applications in [83], [63, Sec. 5]). For a matrix $F \in \mathbb{K}(x)^{m\times m}$, a description $F = ND^{-1}$ with $N, D \in \mathbb{K}[x]^{m\times m}$ is said to be minimal if N and D are right coprime (have unimodular right matrix gcd's), and D has minimal column degrees among all possible denominators. The fraction F is said to be describable in degree δ if it admits both a left description $F = D_L^{-1} N_L$ and a right description $F = ND^{-1}$ with denominators D_L and D of degree at most δ [63, Sec. 5.1.1]. Generically, we have the following for the fraction $Y^\mathsf{T} S^{-1} X$ we are interested in. This is an adaptation of [83, Prop. 4.1] which used slightly different projections. Indeed we chose to switch the role of the projections X and Y, in order to make the giant steps simpler.

Proposition 3.8.1. *For any even n and integers $d, m \in \{1, \ldots, n\}$ there exists a nonzero polynomial Φ in $2(n/2+1)(d+1)$ variables over $\mathbb{K}$ and of degree $O(n^3 d^2)$ such that for $p = \sum_{0\le i\le d, 0\le j\le n/2} p_i^{(j)} x^i y^j$ and $q = \sum_{0\le i\le d, 0\le j\le n/2} q_i^{(j)} x^i y^j$ of y-degree $n/2$ in $\mathbb{K}[x, y]$, if $\Phi(p_0^{(0)}, \ldots, p_d^{(n/2)}, q_0^{(0)}, \ldots, q_d^{(n/2)}) \neq 0$ then:*

i) S is invertible and S^{-1} is strictly proper (each entry has its numerator degree less than its denominator degree);

ii) $Y^\mathsf{T} S^{-1} X$ is describable in degree $\delta = 2\lceil n/(2m)\rceil d$;

iii) if $Y^\mathsf{T} S^{-1} X = ND^{-1}$ is a minimal description then $\det D = c\, Res_y(p, q)$ for some nonzero $c \in \mathbb{K}$.

Proof. Consider $\hat{q} = y^{n/2}q(1/y)$, $\hat{p} = y^{n/2}p(1/y)$, and the associated Sylvester matrix $\hat{S} \in \mathbb{K}[x]^{n\times n}$. From [83, Sec. 4], there exists a nonzero polynomial $\hat{\Phi}$ in $2(n/2+1)(d+1)$ variables and of degree $O(n^3d^2)$, such that if the coefficients of $\hat{q}$ and $\hat{p}$ do not form a zero of $\hat{\Phi}$, then: $\hat{q}$ and $\hat{p}$ have degree $n/2$; $\hat{S}$ is invertible and $\hat{S}^{-1}$ is strictly proper; $X^\mathsf{T}\hat{S}^{-1}Y$ is describable in degree δ; a minimal description

$$X^\mathsf{T}\hat{S}^{-1}Y = \hat{N}\hat{D}^{-1} \tag{3.64}$$

has a denominator that satisfies $\det \hat{D} = \hat{c}\,\mathrm{Res}_y(\hat{q},\hat{p})$ for some $\hat{c} \in \mathbb{K}^*$.

Let Φ be the polynomial obtained from $\hat{\Phi}$ by swapping variables so that evaluating Φ at the coefficients of p and q is evaluating $\hat{\Phi}$ at the coefficients of $\hat{q}$ and $\hat{p}$. We show that Φ is appropriate.

Assume that the coefficients of p and q do not form a zero of Φ. We have $\hat{S} = J_nSJ_n$, where J_n is the reversal matrix of dimension n. Since i) is satisfied with $\hat{S}$ as $\hat{q}$ and $\hat{p}$ do not form a zero of $\hat{\Phi}$ we have that i) is also satisfied with S.

We then show that if $X^\mathsf{T}\hat{S}^{-1}Y$ is describable in degree δ, then $Y^\mathsf{T}S^{-1}X$ is also describable in degree δ. From Eq. (3.64) and using $\hat{S}^{-1} = J_nS^{-1}J_n$ we get

$$Y^\mathsf{T}S^{-1}X = J_m\hat{N}(J_m\hat{D})^{-1}, \tag{3.65}$$

which shows the existence of an appropriate right description for $Y^\mathsf{T}S^{-1}X$. Indeed, $\deg \hat{D} \le \delta$ since $\hat{D}$ is a minimal denominator of $X^\mathsf{T}\hat{S}^{-1}Y$. In a similar way, a left denominator of degree at most δ for $Y^\mathsf{T}S^{-1}X$ is obtained from a left denominator of degree at most δ for $X^\mathsf{T}\hat{S}^{-1}Y$.

Item iii) is finally proved by noticing that $J_m\hat{D}$ is minimal in Eq. (3.65) if and only if $\hat{D}$ is minimal in Eq. (3.64). A minimal denominator D for $Y^\mathsf{T}S^{-1}X$ hence gives a minimal denominator $\hat{D} = J_mD$ for $X^\mathsf{T}\hat{S}^{-1}Y$, and $\det D = \pm\det\hat{D} = \pm\hat{c}\,\mathrm{Res}_y(\hat{q},\hat{p}) = c\mathrm{Res}_y(p,q)$. □

If $Y^\mathsf{T}S^{-1}X$ satisfies the first two items in Proposition 3.8.1, then a minimal description ND^{-1} can be computed from 2δ terms of its expansion. We follow [26,63], and perform the reconstruction of the fraction using minimal approximant bases [3,79]. We especially refer to [63, Sec. 5.2, 5.3] for a detailed treatment of the reconstruction, which we do not repeat here, Algorithm FractionReconstruction being exactly Step 2 of [63, Algorithm 5.1]. Note that the latter algorithm is applied to a fraction that is constructed in a different way than $Y^\mathsf{T}S^{-1}X$ but this does not intervene for the reconstruction itself.

Lemma 3.8.2. *Assume that $Y^\mathsf{T}S^{-1}X$ satisfies i) and ii) in Proposition 3.8.1. Given $H = \lceil Y^\mathsf{T}S^{-1}X\rceil_{2\delta/d}$ (x^d-adic notation here), Algorithm FractionReconstruction*

Algorithm 3.8.1 FractionReconstruction

Input: $\delta \in \mathbb{N}, H \in \mathbb{K}[x]^{m\times m}_{<2\delta}$
Output: $(N, D) \in \mathbb{K}[x]^{m\times m}_{\leq\delta}$ such that $ND^{-1} \equiv H \mod x^{2\delta}$
1: $F \leftarrow [H \quad -\mathrm{I}_m] \in \mathbb{K}[x]^{m\times 2m}$
2: ▷ *Computation of a minimal approximant basis* $P \in \mathbb{K}[x]^{2m\times 2m}_{\leq 2\delta}$*, see [26, Thm. 2.4], [39, Prop. 3.2]*
$P \leftarrow$ PM-Basis$(F^\mathsf{T}, 2\delta, 0)$, with P in weak Popov Form; $P \leftarrow P^\mathsf{T}$ ▷ *PM-Basis from [26]*
3: **return** $(P_{m+1..2m,1..m}, P_{1..m,1..m})$

computes a minimal description $Y^\mathsf{T}S^{-1}X = ND^{-1}$ *using* $\tilde{O}(m^\omega\delta)$ *arithmetic operations in* $\mathbb{K}$*.*

Proof. Item *(iii)* of [63, Proposition 5.4] proves the correctness as soon as a correct approximant basis is computed at Step 2. This basis is obtained using $\tilde{O}(m^\omega\delta)$ operations [26, Thm. 2.4], [39, Prop. 3.2]. Following [63], transposes are used at Step 2 because in [26, 39] approximant bases are considered row-wise rather than column-wise. □

3.8.2 Resultant algorithm

We now present our Algorithm StructuredResultant that computes the resultant of two generic polynomials p and q whose Sylvester matrix is S. Once sufficiently many terms of the expansion of $Y^\mathsf{T}S^{-1}X$ are computed with Algorithm StructuredExpansion, Algorithm FractionReconstruction is called to compute a fraction description $Y^\mathsf{T}S^{-1}X = ND^{-1}$ whose denominator's determinant is the resultant up to a constant factor. This determinant is obtained using dense polynomial linear algebra [52], and the multiplicative constant is retrieved by comparing the determinant with the resultant at $x = 0$.

Our improved complexity bound is proved for generic p and q of degree $d < n$ in x and $n_p = n_q = n/2$ in y. More precisely, the resultant algorithm is correct with the prescribed cost if the following assumptions are satisfied.

(A1) $\det S(0) \neq 0$. This allows to choose $x = 0$ as expansion point. (Note that a truncated resultant algorithm which avoids this hypothesis is studied in [62].)

(A2) $p^{(0)}(0) \neq 0$ and $q^{(n_q)}(0) \neq 0$. These conditions are introduced in Section 3.4 in order to identify the structure of the high-order components and residues.

(A3) $\bar{F} \in \mathbb{K}^{n\times(d+1)}$ as defined in Proposition 3.4.4 has rank $d + 1$. This is assumed in Lemma 3.5.1 in order to compress the results of middle products into canonical generators.

(A4) The coefficients of p and q do not form a zero of the polynomial Φ of Proposition 3.8.1. This assumption ensures that an appropriate description of $Y^{\mathsf{T}}S^{-1}X$ can be recovered from a small number of terms in the expansion of the matrix fraction.

Algorithm 3.8.2 StructuredResultant

Input: $p, q \in \mathbb{K}[x, y]$ of degree $d < n$ in x and degree $n/2$ in y (n even), and $S \in \mathbb{K}[x]^{n\times n}_{\leq d}$ their associated Sylvester matrix; $m \leq n/2, r, s \in \mathbb{N}^*$ such that $rs \geq 4\lceil n/(2m)\rceil$
Genericity assumptions: (A1) to (A4)
Output: $\mathrm{Res}_y(p, q) \in \mathbb{K}[x]_{\leq nd}$

1: $H \leftarrow$ StructuredExpansion(p, q, m, r, s) ▷ $H = \lceil Y^{\mathsf{T}}S^{-1}X\rceil_{rs}$
2: $(N, D) \leftarrow$ FractionReconstruction(rsd, H) ▷ $ND^{-1} = Y^{\mathsf{T}}S^{-1}X$
3: $t \leftarrow \det D \in K[x]_{\leq nd}$ ▷ *Determinant computation from [52, Thm. 1.1]*
$c \leftarrow \det(S(0))/t(0) \in \mathbb{K}^*$ ▷ *Nonzero scalar to obtain the resultant*
4: **return** ct

Lemma 3.8.3. *Let $p, q \in \mathbb{K}[x, y]$ of degree $d < n$ in x and degree $n/2$ in y (n even) and $m \leq n/2, r, s \in \mathbb{N}^*$. If the assumptions (A1) to (A4) hold and $rs \geq 4\lceil n/(2m)\rceil$ then Algorithm StructuredResultant computes the resultant of p and q with respect to y. With $s = O(r)$, $mr = O(n)$ the algorithm uses $\tilde{O}\left(\mathsf{MM}_{n,d}(r + d, r) + m^{\omega}rsd\right)$ arithmetic operations in $\mathbb{K}$.*

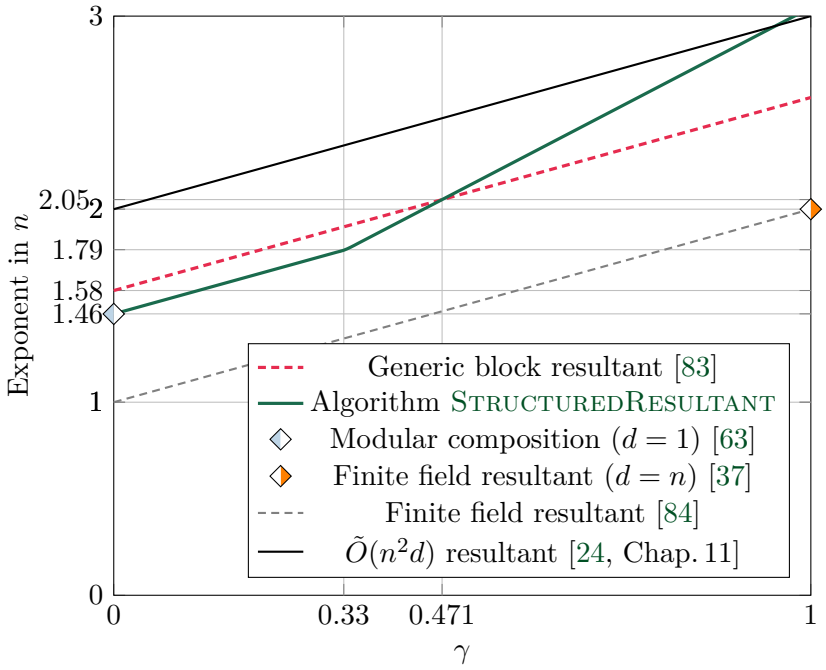
Proof. The truncated expansion $H = \lceil Y^{\mathsf{T}}S^{-1}X\rceil_{rs}$ is computed at Step 1 in $\tilde{O}(\mathsf{MM}_{n,d}(r + d, r) + m^2rsd)$ from Proposition 3.7.1, here we have used assumptions (A1) to (A3). Assumption (A4) ensures that Proposition 3.8.1 can be applied. Items *i*) and *ii*) of the latter proposition and Lemma 3.8.2 ensure that Step 12 is performed in time $\tilde{O}(m^{\omega}rsd)$ since $rs \geq 4\lceil n/(2m)\rceil = 2\delta/d$. Items *iii*) of the same proposition shows that $\mathrm{Res}_y(p, q) = ct$ for some nonzero $c \in \mathbb{K}$, which is computed by considering the constant terms of both polynomials at negligible cost. Note that by (A1) we know that $\det(S(0)) \neq 0$, hence $t(0) = \det(D(0)) \neq 0$. From *ii*) in Proposition 3.8.1, the degree of $D \in \mathbb{K}[x]^{m\times m}$ is bounded by δ, its determinant is also computed in time $\tilde{O}(m^{\omega}rsd)$ [52, Thm. 1.1]. □

The input parameters m, r and s of Algorithm StructuredResultant can be optimised with respect to n and d, which allows us to prove Theorem 3.1.1.

Proof of Theorem 3.1.1. Now taking p and q of y-degree n, for every $m \in \{1, \ldots, n\}$ we can associate to each of the assumptions (A1) to (A4) a non identically zero polynomial in $2(n+1)(d+1)$ variables over $\mathbb{K}$, whose zeros are the inputs which do not meet the conditions. An appropriate hypersurface is defined by the product of these polynomials for a well chosen value of m. Consider $s = r$ and m such that $mr^2 \sim 2n$. From Lemma 3.8.3 and

Eq. (3.25), the resultant can be computed generically using $\tilde{O}(nd(m^{\omega-1} + r^{\omega-1} + dr^{\omega-2}))$ operations. When $d \leq r$ the optimal choice for the parameters leads to $m = r$, and we arrive to the announced bound in the case $d = O(n^{1/3})$. For greater values of d, we consider $m = n^{\frac{\omega-2}{3\omega-4}} d^{\frac{2}{3\omega-4}}$ and keep the same relations between r, s and m. □

Figure 3.1: Comparison of known asymptotic cost bounds for the resultant of generic polynomials. Exponents in n with $d = n^\gamma$ and $\omega = 2.372$.



On Fig. 3.1 we compare the asymptotic exponent of Algorithm StructuredResultant to the exponents of existing algorithms for d varying in comparison to n. Note that the genericity conditions (in the Zariski sense) differ for each algorithm. When $d = O(n^{1/3})$, our algorithm has an exponent in n that coincides with the one of [63] (without fast rectangular matrix multiplication), and allows to be essentially linear in the degree. In that case, Algorithm StructuredResultant compares favorably to [83] as soon as $\omega < 3$: the cost is $\tilde{O}(n^{1.458} d)$ with $\omega < 2.372$ [1, 18, 85]. Our new estimate breaks $\tilde{O}(n^{1.5} d)$, which is the cost estimate for [83] in the best possible case where ω would be 2. The cost bound in the second item of Theorem 3.1.1 has a stronger dependence in d, hence our new algorithm is not better for large values of d compared to n. However, it remains faster as long as $d = O(n^{(\omega-1)(4-\omega)/(2\omega)})$, hence $d = O(n^{0.47})$ for $\omega = 2.372$.

Algorithm StructuredResultant may be viewed as a generalisation of [63, Sec. 10.1] for structured polynomial matrices with arbitrary degrees. On the other hand it also generalises the resultant algorithm of [83] (up to a minor technical change in projections),

which can be revealed by taking $s = 1$. With this parameter choice we have no more giant steps, neither application of high-order lifting. The computation of the expansion is essentially done in the baby steps at Step 1; this may be compared to the use of [83, Prop. 5.1] with truncated power series. For $s = 1$, the displacement rank of $\mathcal{R}$ at Step 3 is constant, and our bound $s + d$ for this rank (proof of Proposition 3.7.1) leads to an overestimation of the cost of the reconstruction from Step 4 to Step 6. Nevertheless, by taking into account the displacement rank simplification in this degenerate case, the resultant algorithm in [83] is recovered.

Chapter 4 | Exact computations with quasiseparable matrices

This chapter is derived from a joint work with Clément Pernet and Gilles Villard presented at the 2023 edition of the International Symposium on Symbolic and Algebraic Computation [70].

4.1 Introduction

Quasiseparable matrices arise frequently in various problems of numerical analysis and are becoming increasingly important in computer algebra, e.g. by their application to handle linearisations of polynomial matrices [6]. Structured representations for these matrices and their generalisations have been widely studied but to our knowledge they have not been compared in detail with each other. In this chapter we aim to adapt `SSS` [22] and `HSS` [13, 57], two of the most prominent formats of numerical analysis to exact computations and compare them theoretically and experimentally to the Bruhat format [72]. These formats all have linear storage size in both the dimension and the structure parameter. We do not investigate the Givens weight representation [16] as it strongly relies on orthogonal transformations in $\mathbb{C}$, which is more challenging to translate in the algebraic setting. See [29, 80, 81] for an extensive bibliography on computing with quasiseparable matrices.

Definition 4.1.1. *An $n \times n$ matrix A is s-quasiseparable if* $\operatorname{rank}(A_{1..k,k+1..n}) \leq s$ *and* $\operatorname{rank}(A_{k+1..n,1..k}) \leq s$ *for all* $k \in [\![1, n]\!]$.

Complexity bound notation. We consider matrices over an abstract commutative field $\mathbb{K}$, and count arithmetic operations in $\mathbb{K}$. Our comparison focuses on the leading term in the complexities, namely a function $\mathsf{T}_{\mathsf{XXX}}(n, s)$ such that running Algorithm XXX with parameters (n, s) costs $\mathsf{T}_{\mathsf{XXX}}(n, s) + o(\mathsf{T}_{\mathsf{XXX}}(n, s))$ asymptotically in n and s. We proceed similarly for the space cost bounds with the notation $\mathsf{S}_{\mathsf{XXX}}(n, s)$. We denote by ω a feasible exponent for square matrix multiplication, and C_ω the corresponding leading constant;

namely, using above notation, $\mathsf{T}_{\mathsf{MM}}(n) = C_\omega n^\omega$. The straightforward generalisation gives $\mathsf{T}_{\mathsf{MM}}(m,k,n) = C_\omega mnk \min(m,k,n)^{\omega-3}$ for the product of an $m \times k$ by a $k \times n$ matrix.

4.1.1 Rank revealing factorisations

Space efficient representations for quasiseparable matrices rely on rank revealing factorisations: a rank r matrix $A \in \mathbb{K}^{m\times n}$ is represented by two matrices $L \in \mathbb{K}^{m\times r}, R \in \mathbb{K}^{r\times n}$ such that $A = LR$. In exact linear algebra, such factorisations are usually computed using Gaussian elimination, such as PLUQ, CUP, PLE, CRE decompositions [20, 40, 76], which we will generically denote by `RF`.

Cost estimates of the above factorisation algorithms are either given as $O(mnr^{\omega-2})$ or with explicit leading constants $\mathsf{T}_{\mathsf{RF}}(m,n,r) = K_\omega n^\omega$ under genericity assumptions: $m = n = r$ and generic rank profile [20, 40]. We refer to [71] for an analysis in the non-generic case of the leading constants in the cost of the two main variants of divide and conquer Gaussian elimination algorithms. We may therefore assume that $\mathsf{T}_{\mathsf{RF}}(m,n,r) = C_{\mathtt{RF}} mnr^{\omega-2}$ for a constant $C_{\mathtt{RF}}$, for $\omega \geq 1 + \log_2 3$, which is the case for all practical matrix multiplication algorithm. Note that for $\omega = 3$, these costs are both equal to $2mnr$. Unfortunately, the non-predictable rank distribution among the blocks being processed leads to an over-estimation of some intermediate costs which forbids tighter constants (i.e. interpolating the known one $K_3 = 2/3$ in the generic case). The algorithms presented here are still valid for smaller values of ω, but for the sake of clarity, we will not state their more sophisticated constants.

Our algorithms for SSS and HSS can use any rank revealing factorisation. On the other hand, the Bruhat format requires one revealing the additional information of the rank profile matrix, e.g. the CRE decompositions used here (See [20]).

Theorem 4.1.2 ([20, 58])**.** *Any rank r matrix $A \in \mathbb{K}^{m\times n}$ has a CRE decomposition $A = CRE$ where $C \in \mathbb{K}^{m\times r}$ and $E \in \mathbb{K}^{r\times n}$ are in column and row echelon form, and $R \in \mathbb{K}^{r\times r}$ is a permutation matrix.*

The costs related to Bruhat generator therefore rely on constants $C_{\mathtt{RF}}$ from factorisations allowing to produce a CRE decomposition, like the ones in [71].

4.1.2 Contributions

In Section 4.2 we define the SSS, HSS and Bruhat formats. We then adapt algorithms operating with HSS and SSS generators from the literature to the exact context. The HSS generation algorithm is given in a new iterative version and the SSS product algorithm has an improved cost. We focus for SSS on basic bricks on which other operations can

be built. This opens the door to adaptation of fast algorithms for inversion and system solving [11, 12, 23] and format modeling operations such as merging, splitting and model reduction [12]. In Section 4.3.3 we give a generic Bruhat generation algorithm from which we derive new fast algorithms for the generation from a sparse matrix and from a sum of matrices in Bruhat form.

Table 4.1 displays the best cost estimates for different operations on an $n \times n$ s-quasiseparable matrix in the three formats presented in the chapter. The best and optimal storage size is reached by the Bruhat format which also has the fastest generator computation algorithm. However, this is not reflected in the following operation costs as applying a quasiseparable matrix to a dense matrix is least expensive with an SSS generator and addition and product of $n \times n$ matrices given in Bruhat form is super-linear in n. We notice in Proposition 4.2.5 that HSS is twice as expensive as SSS and gives no advantage in our context. We thus stop the comparison at the generator computation. We still give in Table 4.1 the cost of quasiseparable $\times$ dense product which is proportional to the generator size [57]. We complete this analysis with experiments showing that despite slightly worse asymptotic cost estimates, SSS performs better than Bruhat in practice for the construction in Section 4.3.5 and the product by a dense block vector in Section 4.4.3.

4.2 Presentation of the formats

4.2.1 SSS generators

Introduced in [22], SSS generators were later improved independently in [23] and [12] using block-versions, which we present here. In particular, the space was improved from $O(ns^2)$ to $O(ns)$.

An s-quasiseparable matrix is sliced following a grid of $s \times s$ blocks. Blocks on, over and under the diagonal are treated separately. On one side of the diagonal, each block is defined by a product depending on its row (left-most block of the product), its column (right-most block), and its distance to the diagonal (number of blocks in the product).

Definition 4.2.1. *Let* $A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,N} \\ \vdots & & \vdots \\ A_{N,1} & \cdots & A_{N,N} \end{bmatrix} \in \mathbb{K}^{n \times n}$ *with* $t \times t$ *blocks* $A_{i,j}$ *for* $i, j < N$ *and* $N = \lceil n/t \rceil$. *A is given in sequentially semi-separable format of order t (t-SSS) if it is given by the $t \times t$ matrices* $(P_i, V_i)_{i \in [\![2,N]\!]}$, $(Q_i, U_i)_{i \in [\![1,N-1]\!]}$, $(R_i, W_i)_{i \in [\![2,N-1]\!]}$, $(D_i)_{i \in [\![1,N]\!]}$ *s.t.*

$$A_{i,j} = \begin{cases} P_i R_{i-1} \dots R_{j+1} Q_j & \text{if } i > j \\ D_i & \text{if } i = j \\ U_i W_{i+1} \dots W_{j-1} V_j & \text{otherwise} \end{cases} \tag{4.1}$$

Table 4.1: Summary of operation and storage costs

	ω			$\omega = 3$		
	SSS	HSS	Bruhat	SSS	HSS	Bruhat
Storage	$7ns$	$18ns$	$4ns$	$7ns$	$18ns$	$4ns$
Gen. from Dense	$2C_{\text{RF}}n^2s^{\omega-2}$	$2^\omega C_{\text{RF}}n^2s^{\omega-2}$	$C_{\text{RF}}n^2s^{\omega-2}$	$4n^2s$	$16n^2s$	$2n^2s$
$\times$ Dense block vector$(n \times v)$	$7C_\omega nsv^{\omega-2}$	$18C_\omega nsv^{\omega-2}$	$8C_\omega nsv^{\omega-2}$	$14nsv$	$36nsv$	$16nsv$
Addition	$(10+2^\omega)C_\omega ns^{\omega-1}$		$\left(\frac{9\cdot 2^{\omega-2}-8}{2^{\omega-2}-1}C_\omega + 2C_{\text{RF}}\right)ns^{\omega-1}\log n/s$	$36ns^2$		$24ns^2\log n/s$
Product	$(31+2^\omega)C_\omega ns^{\omega-1}$			$78ns^2$		

Proposition 4.2.2. *Any $n \times n$ s-quasiseparable matrix has an s-SSS representation. It uses $\mathsf{S}_{\mathsf{SSS}}(n,s) = 7ns$ field elements.*

Proof. Direct consequence of Proposition 4.3.1. □

4.2.2 HSS generators

The HSS format was first introduced in [14], although the idea originated with the *uniform $\mathcal{H}$-matrices* of [28] and in more details with the *$\mathcal{H}^2$-matrices* of [30], with algorithms relying on [75]. The $\mathcal{H}^2$ format is slightly different from HSS, more details in [29].

The format is close to SSS (see Proposition 4.2.4) as the way of defining blocks is similar. Yet, the slicing grid is built recursively and the definition of blocks product depends on the path to follow in the recursion tree. Also, both sides of the diagonal are treated jointly and the format is therefore less compact, which as will be shown makes HSS less efficient.

The structure is complex and notations differ in the literature. We made the following choices: we avoid the recursive tree definition inherited from the Fast Multipole Method [14] and thus only consider constant-depth recursive block divisions. We made this choice to focus on linear algebra and quasiseparable matrices with no pre-requisites (no notion of where the rank is). For the same reason we focus on uniform subdivisions. Most literature on HSS uses non-uniform grids in order to adapt to matrices with a structure within the quasiseparable rank structure [14]. Despite being more general, this adds confusion which is not needed in our case.

We use a notation similar to [86] with transition matrices.

Definition 4.2.3. *Let $A \in \mathbb{K}^{n\times n}$ and the uniform block divisions*

$$A = \begin{bmatrix} A_{k;1,1} & \cdots & A_{k;1,2^k} \\ \vdots & & \vdots \\ A_{k;2^k,1} & \cdots & A_{k;2^k,2^k} \end{bmatrix}. \tag{4.2}$$

A is given in hierarchically semi-separable format of order t (t-HSS) if it is given by the $t \times t$ matrices $(U_{K;i}, V_{K;i}, D_i)_{i\in[\![1,N]\!]}$, $(R_{k;i}, W_{k;i})_{\substack{k\in[\![2,K]\!] \\ i\in[\![1,2^k]\!]}}$ and $(B_{k;i})_{\substack{k\in[\![1,K]\!] \\ i\in[\![1,2^k]\!]}}$ with $N = \lceil n/t \rceil$ and $K \geq \log N$ such that for $i \in [\![1,N]\!]$, $A_{K;i,i} = D_i$ and if we define recursively for k from $K-1$ to 1 and $i \in [\![1,2^k]\!]$, $U_{k;i} = \begin{bmatrix} U_{k+1;2i-1}R_{k+1;2i-1} \\ U_{k+1;2i}R_{k+1;2i} \end{bmatrix}$ and $V_{k;i} = \begin{bmatrix} W_{k+1;2i-1}V_{k+1;2i-1} & W_{k+1;2i}V_{k+1;2i} \end{bmatrix}$ then

$$\begin{aligned} A_{k;2i-1,2i} &= U_{k;2i-1}B_{k;2i-1}V_{k;2i} \\ A_{k;2i,2i-1} &= U_{k;2i}B_{k;2i}V_{k;2i-1} \end{aligned} \tag{4.3}$$

The HSS generator can be seen as a recursive SSS generator with two differences : the use of the B matrices, and the distribution of the translation matrices. The similarity is made clear in Proposition 4.2.4.

Proposition 4.2.4. *Let $U_{K;i}, V_{K;i}, D_i, R_{k;i}, W_{k;i}, B_{k;i}$ for appropriate $k \leq K, i \leq 2^k$ a t-HSS generator for A. Let $I, J \in [\![1, 2^K]\!]$ and k the highest level of recursion for which $A_{K;I,J}$ is not included in a diagonal block. For $i_1 = \lfloor I/2^{K-k-1} \rfloor, i_0 = \lfloor I/2^{K-k} \rfloor$ and $j_1 = \lfloor J/2^{K-k-1} \rfloor$ we have*

$$A_{K;I,J} = U_{K;I} R_{K;I} ... R_{k+1;i_1} B_{k;i_0} W_{k+1;j_1} ... W_{K;J} V_{K;J}. \tag{4.4}$$

Proof. By induction on Equation (4.3). □

Proposition 4.2.5. *Any $n \times n$ s-quasiseparable matrix has a $2s$-HSS representation. This is the optimal block parameter and the representation uses $\mathsf{S}_{\mathsf{HSS}}(n, s) = 18ns$ field elements.*

Proof. Consequence of Proposition 4.3.2. For optimality let A be s-quasiseparable given in t-HSS form. We use Proposition 4.2.4:

$$\begin{bmatrix} A_{K;3...4,1...2} & A_{K;3...4,5...6} \end{bmatrix} = \begin{bmatrix} U_{K;3} R_{K;3} \\ U_{K;4} R_{K;4} \end{bmatrix} H \tag{4.5}$$

where $H \in K^{t \times 4t}$. The quasiseparability of A bounds the rank of the left part of Eq. (4.5) by $2s$ while the one of the right side is bounded by t. When the first bound is tight we get $t \geq 2s$. □

4.2.3 Bruhat generators

The Bruhat generator was first defined in [68, 72]. Contrarily to SSS and HSS, it does not use a pre-defined grid but relies on the rank profile information contained in the rank profile matrix [20] of the lower and upper triangular parts of the quasiseparable matrix.

Recall from [72] that a matrix is t-overlapping if any subset of $t+1$ of its nonzero columns (resp. rows) contains at least one whose leading nonzero element is below (resp. before) the trailing nonzero element of another. We call J_n the anti-identity matrix of dimension n and define the *Left* operator $\triangledown : \mathbb{K}^{n \times n} \to \mathbb{K}^{n \times n}$ s.t.

$$\triangledown (A)_{i,j} = \begin{cases} A_{i,j} & \text{if } i + j \leq n \\ 0 & \text{otherwise} \end{cases}. \tag{4.6}$$

Definition 4.2.6. *An $n \times n$ matrix A is represented in t-Bruhat format if it is given by a diagonal matrix $D \in \mathbb{K}^{n \times n}$ and 6 matrices $C^{(L)}, R^{(L)}, E^{(L)}, C^{(U)}, R^{(U)}, E^{(U)}$ where $C^{(L)} \in$*

$\mathbb{K}^{n\times u}$ *and* $C^{(U)} \in \mathbb{K}^{n\times v}$ *are in column echelon form and* t*-overlapping,* $E^{(L)} \in \mathbb{K}^{u\times n}$ *and* $E^{(U)} \in \mathbb{K}^{v\times n}$ *are in column echelon form and* t*-overlapping and* $R^{(L)} \in \mathbb{K}^{u\times u}, R^{(U)} \in \mathbb{K}^{v\times v}$ *are permutation matrices and satisfy*

$$A = D + \mathrm{J}_n \triangledown \left(C^{(L)} R^{(L)} E^{(L)}\right) + \triangledown \left(C^{(U)} R^{(U)} E^{(U)}\right) \mathrm{J}_n$$

Proposition 4.2.7. *Any* $n \times n$ s*-quasiseparable matrix has an* s-Bruhat *representation. It uses* $\mathsf{S}_{\mathsf{Bruhat}}(n, s) = 4ns$ *field elements which is optimal.*

Proof. By [72, Theorem 20]. As $2ns$ coefficients are necessary to represent all rank s triangular matrices, $4ns$ is optimal. □

4.3 Construction of the generators

4.3.1 SSS generator from a dense matrix

We recall in Algorithm DenseToSSS the construction of an SSS generator from a dense s-quasiseparable matrix $A \in \mathbb{K}^{n\times n}$. It is adapted from [12, §6.1] and [23, Alg. 6.5] where the SVD based numerical rank revealing factorisations are replaced by RF.

The blocks D_i are directly extracted from the dense matrix in Step 3. Each block-triangular part is then compressed independently. Each step eliminates a chunk made of a block-row of A and a remainder from the previous step. The result is three blocks of the generator and a remainder to be eliminated at the subsequent step.

Algorithm 4.3.1 DenseToSSS

Input: A an $n \times n$ s-quasiseparable matrix with $s \le t$

Output: $P_i, Q_i, R_i, U_i, V_i, W_i, D_i$ for appropriate $i \in [\![1, N]\!]$ a t-SSS representation of A

1: $A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,N} \\ \vdots & & \vdots \\ A_{N,1} & \cdots & A_{N,N} \end{bmatrix}, H = \begin{bmatrix} H_{0,1} & \cdots & H_{0,N} \\ \vdots & & \vdots \\ H_{N,1} & \cdots & H_{N,N} \end{bmatrix} \leftarrow 0$

2: **for** $k = 1 \dots N-1$ **do**

3: $\quad D_k \leftarrow A_{k,k}$

4: $\quad \left(\begin{bmatrix} W_k \\ U_k \end{bmatrix}, \begin{bmatrix} V_{k+1} & H_{k,k+2\dots N} \end{bmatrix}\right) \leftarrow \mathtt{RF}\left(\begin{bmatrix} H_{k-1,k+1\dots N} \\ A_{k,k+1\dots N} \end{bmatrix}\right)$

5: $\quad \left(\begin{bmatrix} Q_{k+1} \\ H_{k+2\dots N,k} \end{bmatrix}, \begin{bmatrix} R_k & P_k \end{bmatrix}\right) \leftarrow \mathtt{RF}\left(\begin{bmatrix} H_{k+1\dots N,k-1} & A_{k+1\dots N,k} \end{bmatrix}\right)$

6: $D_N = A_{N,N}$

Proposition 4.3.1. *Algorithm* DenseToSSS *computes a* t-SSS *generator for an* s*-quasiseparable matrix* ($s \le t$) *in* $\mathsf{T}_{\mathsf{DenseToSSS}}(n, t) = 2C_{RF} n^2 s^{\omega-2}$ *field operations.*

Proof. For $k \in [\![1, N-1]\!]$, the dimensions of the output of Lines 4 and 5 are sufficient since the input of the factorisation is a concatenation of a block of A with a rank-revealing factor of another block of A on the same side of the diagonal, and is hence of rank at most s.

Let $i, j \in [\![1, N]\!]$. If $i = j$ Step 3 for $k = i$ gives $D_i = A_{i,i}$. If $i < j$, Step 4 gives

$$W_{j-1}V_j = H_{j-2,j} \tag{4.7}$$
$$W_k H_{k,j} = H_{k-1,j} \qquad (k \in [\![1, j-2]\!]) \tag{4.8}$$
$$U_i H_{i,j} = A_{i,j} \qquad (i < N) \tag{4.9}$$
$$U_i V_{i+1} = A_{i,i+1} \tag{4.10}$$

which combines to $U_i W_{i+1} \dots W_{j-1} V_j = A_{i,j}$. The same way, if $i > j$ then $A_{i,j} = P_i R_{i-1} \dots R_{j+1} Q_j$.

The cost is $\sum_{k=1}^{N-1} 2\mathsf{T}_{\mathtt{RF}}(t(N-k), 2t, s) = 2C_{\mathtt{RF}} n^2 s^{\omega-2}$. □

4.3.2 HSS generator from a dense matrix

The first construction algorithm for a general quasiseparable matrix is presented in [14]. We present in Algorithm DenseToHSS an iterative version of the faster and simpler algorithm of [86].

Each step of the loop on k passes block-row-wise and block-column-wise on the matrix inherited from the previous step, factorising block rows and block columns two by two. At each step each block is hence factorised twice, producing transition matrices R and W, the remainder being either passed to the following step or finally stored as a B matrix.

Proposition 4.3.2. *Algorithm DenseToHSS computes a t-HSS generator for an s-quasiseparable matrix if $2s \leq t$ in $C_{\mathtt{RF}} n^2 t^{\omega-2}$ field operations. Taking $t = 2s$, this is $\mathsf{T}_{\mathsf{DenseToHSS}}(n, s) = 2^{\omega} C_{\mathtt{RF}} n^2 s^{\omega-2}$.*

Proof. Let $k \in [\![1, K]\!]$, $i \neq j \in [\![1, 2^k]\!]$. The dimensions of the output in Lines 6 and 7 are sufficient since the matrices being factorised are each time a concatenation of two blocks of rank at most s and are hence of rank at most $2s \leq t$. If $|i - j| = 1$, the instructions give

$$H_{k;i,j} = \begin{bmatrix} R_{k+1;2i-1} \\ R_{k+1;2i} \end{bmatrix} B_{k;i} \begin{bmatrix} W_{k+1;2j-1} & W_{k+1;2j} \end{bmatrix}. \tag{4.11}$$

Otherwise,

$$H_{k;i,j} = \begin{bmatrix} R_{k+1;2i-1} \\ R_{k+1;2i} \end{bmatrix} H''_{k;i,j} \begin{bmatrix} W_{k+1;2j-1} & W_{k+1;2j} \end{bmatrix}. \tag{4.12}$$

Algorithm 4.3.2 DenseToHSS

Input: A an $n \times n$ quasiseparable matrix of order s

Output: $U_{K;i}, V_{K;i}, D_i, R_{k;i}, W_{k;i}, B_{k;i}$ for appropriate $k \leq K, i \leq 2^k$ a t-HSS representation of A with $t \geq 2s$

$H \leftarrow A$ ▷ *Use the block division of Eq. (4.2) with $k = K$*

for $i = 1 \dots 2^K$ **do**

  $D_i \leftarrow A_{K;i,i}$

for $k = K \dots 1$ **do**

  for $i = 1 \dots 2^k$ **do** ▷ *All operations are in this loop*

  ▷ *$R_{K+1;2i}$ (resp. $W_{K+1;2i}$) has row (resp. column) dimension 0*

$$\left(\begin{bmatrix} R_{k+1;2i-1} \\ R_{k+1;2i} \end{bmatrix}, \begin{bmatrix} H''_{k;i,1\dots i-1} & H'_{k;i,i+1\dots 2^k} \end{bmatrix} \right) \leftarrow \mathtt{RF}\left(\begin{bmatrix} H'_{k;i,1\dots i-1} & H_{k;i,i+1\dots 2^k} \end{bmatrix} \right)$$

$$\left(\begin{bmatrix} H''_{k;1\dots i-1,i} \\ H'_{k;i+1\dots 2^k,i} \end{bmatrix}, \begin{bmatrix} W_{k+1;2i-1} & W_{k+1;2i} \end{bmatrix} \right) \leftarrow \mathtt{RF}\left(\begin{bmatrix} H'_{k;1\dots i-1,i} \\ H_{k;i+1\dots 2^k,i} \end{bmatrix} \right)$$

  for $i = 1 \dots 2^{k-1}$ **do** ▷ *Only renaming from here*

    $B_{k;2i-1} \leftarrow H''_{k;2i-1,2i}$

    $B_{k;2i} \leftarrow H''_{k;2i,2i-1}$

    for $j = 1 \dots 2^{k-1}, j \neq i$ **do**

$$H_{k-1;i,j} \leftarrow \begin{bmatrix} H''_{k;2i-1,2j-1} & H''_{k;2i-1,2j} \\ H''_{k;2i,2j-1} & H''_{k;2i,2j} \end{bmatrix}$$

$$H_{k-1;j,i} \leftarrow \begin{bmatrix} H''_{k:2j-1,2i-1} & H''_{k:2j-1,2i} \\ H''_{k:2j,2i-1} & H''_{k:2j,2i} \end{bmatrix}$$

for $i = 1 \dots 2^K$ **do**

  $U_{K;i} \leftarrow R_{K+1;2i-1}$

  $V_{K;i} \leftarrow W_{K+1;2i-1}$

Let now $I, J \in [\![1, N]\!]$. If $I = J$, Step 3 gives $D_I = A_{K;I,J}$. Otherwise, let k be the highest level of recursion for which $A_{K;I,J}$ is not included in a diagonal block. From Step 1, $A_{K;I,J} = H_{K;I,J}$. Equation (4.12) can be used $K - k$ times, together with Step 12 to get

$$A_{K;I,J} = R_{K+1;2I-1} \dots R_{k+2;i_2} H''_{k+1;i_1,j_1} W_{k+2;j_2} \dots W_{K+1;2J-1} \tag{4.13}$$

where $i_2 = \lfloor I/2^{K-k-2} \rfloor, i_1 = \lfloor I/2^{K-k-1} \rfloor, j_1 = \lfloor J/2^{K-k-1} \rfloor$ and $j_2 = \lfloor J/2^{K-k-2} \rfloor$. The matrices $R_{K+1;2I-1}$, $W_{K+1;2J-1}$ and $H''_{k+1;i_1,j_1}$ can be replaced in Eq. (4.13) using Lines 15 and 16 and Eq. (4.11) (from the definition of k we have $|i_1 - j_1| = 1$) in order to get Eq. (4.4); this concludes the proof of correctness.

Step 6 at $k < K$ and i performs a rank revealing decompositions on an input formed by the $2t \times (i-1)t$ block $H'_{k;i,1\dots i-1}$ and the $2t \times 2t(2^k - i)$ block $H_{k;i,i+1\dots 2^k}$ at cost $\mathsf{T}_{\mathtt{RF}}(t(2^{k+1} - i), 2t, t)$. The cost is equal for Step 7. The overall cost is then $\sum_{k=1}^{\log_2 \frac{n}{t}} \sum_{i=1}^{2^k} 4C_{\mathtt{RF}}(2^{k+1} - i)t^\omega \leq 4C_{\mathtt{RF}} n^2 t^{\omega-2} \leq 2^\omega C_{\mathtt{RF}} n^2 s^{\omega-2}$. □

Because the blocks of each side of the diagonal are defined by the same matrices, Algorithm DenseToHSS and any HSS construction algorithm applies rank revealing factorisations on blocks with rank bounded by $2s$ for s-quasiseparable matrices instead of s in Algorithm DenseToSSS. The optimal HSS block size of s-quasiseparable matrices is thus $2s$, which makes HSS less efficient in terms of storage and operation cost.

As the costs are higher and HSS has the same drawbacks as SSS, namely needing a fixed slicing grid and a previously computed quasiseparability order, we do not detail more algorithms for HSS. For information in the numerical context we mainly refer to [57, 74]. Note that faster construction algorithms exist, probabilistic in [59] and with constraints on the input in [14].

4.3.3 Bruhat generator from a dense matrix

The construction of a Bruhat generator from a dense matrix is achieved by [72, Alg. 12] run twice, once for each of the upper and lower triangular parts of the input matrix, and the diagonal matrix D is directly extracted from the dense matrix.

We give in Algorithm LBruhatGen an updated version of [72, Alg. 12], where Schur complement computations are delayed until they are needed. This allows for faster computations when the input is not given as a dense matrix and will be used for computing the sum of two matrices in Bruhat form and generators from a sparse matrix.

Algorithm LBruhatGen can be as input a matrix under any structured format, provided we have a way to compute for any submatrix B of the input matrix

1. $\mathtt{CRE}(B, G, H)$ a CRE decomposition of $B - GH^{\mathsf{T}}$;

2. for $\mathcal{R}$ a set of indices, $B_{\mathcal{R},*}$ and $B_{*,\mathcal{R}}$ the rows and columns of B with indices in $\mathcal{R}$.

We use the notation TRSM for *TRiangular Solve Matrix*: $\texttt{TRSM}(L, A)$ outputs $L^{-1}A$ for L triangular.

Algorithm 4.3.3 LBRUHATGEN

Input: $A \in \mathbb{K}^{m\times m}$ left triangular s_A-quasiseparable
Input: $G, H \in \mathbb{K}^{m\times t}$ ▷ $t = 0$ *on the first call*
Output: C, R, E a left-Bruhat generator for $A - \triangledown\left(GH^{\mathsf{T}}\right)$

1: Split $A = \begin{bmatrix} A^{(11)} & A^{(12)} \\ A^{(21)} & \end{bmatrix}, G = \begin{bmatrix} G^{(1)} \\ G^{(2)} \end{bmatrix}, H = \begin{bmatrix} H^{(1)} \\ H^{(2)} \end{bmatrix}$ where $A^{(11)} \in \mathbb{K}^{\frac{m}{2}\times\frac{m}{2}}$
2: $C_0, R_0, E_0 \leftarrow \texttt{CRE}(A^{(11)}, G^{(1)}, H^{(1)})$
3: $\mathcal{R} \leftarrow \texttt{RRP}(C_0); \mathcal{C} \leftarrow \texttt{CRP}(E_0); r_0 \leftarrow \#\mathcal{R}$
4: $\begin{bmatrix} U & V \end{bmatrix} \leftarrow E_0 Q_{\mathcal{C}}$ where $U \in \mathbb{K}^{r_0\times r_0}$ is upper triangular.
5: $\begin{bmatrix} L \\ M \end{bmatrix} \leftarrow P_{\mathcal{R}} C_0$ where $L \in \mathbb{K}^{r_0\times r_0}$ is lower triangular.
6: $X \leftarrow A^{(12)}_{\mathcal{R},*} - G^{(1)}_{\mathcal{R},*} H^{(2)\mathsf{T}}$
7: $B^{(12)} \leftarrow \triangledown(\texttt{TRSM}(L, X))_{\mathcal{R},*}$ ▷ $A^{(12)}_{\mathcal{R},*} = \triangledown\left(LB^{(12)} + G^{(1)}_{\mathcal{R},*} H^{(2)\mathsf{T}}\right)_{\mathcal{R},*}$
8: $Y \leftarrow A^{(21)}_{*,\mathcal{C}} - G^{(2)} H^{(1)\mathsf{T}}_{\mathcal{C},*}$
9: $B^{(21)\mathsf{T}} \leftarrow \triangledown\left(\texttt{TRSM}(U^{\mathsf{T}}, Y^{\mathsf{T}})\right)_{*,\mathcal{C}}$
▷ $A^{(21)}_{*,\mathcal{C}} = \triangledown\left(B^{(21)}U + G^{(2)} H^{(1)\mathsf{T}}_{*,\mathcal{C}}\right)_{*,\mathcal{C}}$
10: $C_1, R_1, E_1 \leftarrow$
$\texttt{LBruhatGen}\left(A^{(21)}_{*,\overline{\mathcal{C}}} \mathrm{I}_{\overline{\mathcal{C}},*}, \begin{bmatrix} G^{(2)} & B^{(21)} \end{bmatrix}, \mathrm{I}_{*,\overline{\mathcal{C}}} \begin{bmatrix} H^{(1)}_{\overline{\mathcal{C}},*} & V^{\mathsf{T}} \end{bmatrix}\right)$
11: $C_2, R_2, E_2 \leftarrow$
$\texttt{LBruhatGen}\left(\mathrm{I}_{*,\overline{\mathcal{R}}} A^{(12)}_{\overline{\mathcal{R}},*}, \mathrm{I}_{*,\overline{\mathcal{R}}} \begin{bmatrix} G^{(1)}_{\overline{\mathcal{R}},*} & M \end{bmatrix}, \begin{bmatrix} H^{(2)} & B^{(12)\mathsf{T}} \end{bmatrix}\right)$
12: $P_{01} \leftarrow$ the permutation which sorts the rows of E_0 and E_1 by increasing column of pivot
13: $P_{02} \leftarrow$ the permutation which sorts the columns of C_0 and C_2 by increasing row of pivot
14: $C \leftarrow \begin{bmatrix} C_0 & C_2 \\ B^{(21)} R_0{}^{\mathsf{T}} & C_1 \end{bmatrix} \begin{bmatrix} P_{02} & \\ & I \end{bmatrix}$
15: $R \leftarrow \begin{bmatrix} P_{02}{}^{\mathsf{T}} & \\ & I \end{bmatrix} \begin{bmatrix} R_0 & & \\ & & R_2 \\ & R_1 & \end{bmatrix} \begin{bmatrix} P_{01}{}^{\mathsf{T}} & \\ & I \end{bmatrix}$
16: $E \leftarrow \begin{bmatrix} P_{01} & \\ & I \end{bmatrix} \begin{bmatrix} E_0 & R_0{}^{\mathsf{T}} B^{(12)} \\ E_1 & \\ & E_2 \end{bmatrix}$
17: **return** C, R, E

Proposition 4.3.3. *An s-*Bruhat *generator can be computed from an $n \times n$ dense s-quasiseparable matrix in $\mathsf{T}_{\mathsf{DenseToB}}(n, s) = C_{RF} n^2 s^{\omega-2}$.*

Proof. Algorithm LBRUHATGEN is adapted from [72, Alg. 12]; we therefore refer to the proof of [72, Theorem 24] for its correctness. Apart from the order in which they are made, the operations are the same in both algorithms when the input is dense and the cost is

hence the same. Computing a Bruhat generator from a dense matrix is two applications of Algorithm LBruhatGen. The cost satisfies:

$\mathsf{T}_{\mathsf{LBG}}(n,s) \leq C_{\mathtt{RF}}/4n^2s^{\omega-2} + 2\mathsf{T}_{\mathsf{LBG}}(n/2,s) \leq C_{\mathtt{RF}}/2n^2s^{\omega-2}$. □

4.3.4 Bruhat generator from a sparse matrix

In applications, matrices are often presented in a sparse structure. In order to detect and/or harness their quasiseparable structure, it is crucial to exploit the sparsity in the construction of the quasiseparable generators. For the construction of a Bruhat generator, the generic algorithm Algorithm LBruhatGen can be applied on a sparse matrix, provided two operations are specialised:

1. the extraction of a subset of $\leq s$ rows or columns into a dense format, which is straightforward for a sparse matrix;
2. the computation of a CRE decomposition, which is specialised in Algorithm SparseCRE which in turn uses Algorithm SparseRankProfiles

Algorithm 4.3.4 SparseCRE

Input: $A \in \mathbb{K}^{m\times m}$ a rank $\leq s$ sparse matrix
Input: $G, H \in \mathbb{K}^{m\times t}$
Output: C, R, E such that $A = CRE + GH^{\mathsf{T}}$

1: $\mathcal{R}, \mathcal{C} \leftarrow$ SparseRankProfiles(A, G, H)
2: $P = \begin{bmatrix} \mathrm{I}_{\mathcal{R},*} \\ \mathrm{I}_{\overline{\mathcal{R}},*} \end{bmatrix}$; $Q = \begin{bmatrix} \mathrm{I}_{*,\mathcal{C}} & \mathrm{I}_{*,\overline{\mathcal{C}}} \end{bmatrix}$

▷ *With* $\bar{A}^{(11)} \in \mathbb{K}^{|\mathcal{R}|\times|\mathcal{R}|}$ *write* $P\left(A - GH^{\mathsf{T}}\right)Q = \begin{bmatrix} \bar{A}^{(11)} & \bar{A}^{(12)} \\ \bar{A}^{(21)} & \bar{A}^{(22)} \end{bmatrix} - \begin{bmatrix} \bar{G}^{(1)} \\ \bar{G}^{(2)} \end{bmatrix} \begin{bmatrix} \bar{H}^{(1)} \\ \bar{H}^{(2)} \end{bmatrix}^{\mathsf{T}}$

3: $M^{(11)} \leftarrow \bar{A}^{(11)} - \bar{G}^{(1)}(\bar{H}^{(1)})^{\mathsf{T}}$
4: $M^{(12)} \leftarrow \bar{A}^{(12)} - \bar{G}^{(1)}(\bar{H}^{(2)})^{\mathsf{T}}$
5: $M^{(21)} \leftarrow \bar{A}^{(21)} - \bar{G}^{(2)}(\bar{H}^{(1)})^{\mathsf{T}}$
6: $(L, R, U) \leftarrow \mathtt{DenseCRE}\left(M^{(11)}\right)$
7: $C \leftarrow \mathtt{TRSM}(L, \bar{M}^{(12)})$ ▷ $C = L^{-1}(\bar{A}^{(12)} - G^{(1)}H^{(2)})$
8: $D \leftarrow \mathtt{TRSM}(\bar{A}^{(21)}, U^{\mathsf{T}})$ ▷ $D = (\bar{A}^{(21)} - G^{(2)}H^{(1)})U^{-1}$
9: $E \leftarrow \begin{bmatrix} U & R^{\mathsf{T}}C \end{bmatrix} Q^{\mathsf{T}}$
10: $C \leftarrow P^{\mathsf{T}} \begin{bmatrix} L \\ DR^{\mathsf{T}} \end{bmatrix}$
11: **return** (C, R, E)

Lemma 4.3.4. *Algorithm* SparseRankProfiles *is correct with probability at least* $1 - 2r/|S|$ *and runs in* $\mathsf{T}_{\mathsf{SparseRP}}(n,r) = 2(C_\omega + C_{RF})nr^{\omega-1} + 2r|A|$ *with* $r = t + s$.

Algorithm 4.3.5 SparseRankProfiles

Input: $A \in \mathbb{K}^{n\times n}$ a sparse matrix of rank $\leq s$.
Input: $G, H \in \mathbb{K}^{n\times t}$ dense matrices
Output: $\mathcal{R}_A, \mathcal{C}_A$ the row and column rank profiles of $A - GH^{\mathsf{T}}$
1: $T^{(1)} \leftarrow$ a unif. random $n \times (s+t)$ Toeplitz matrix from $S \subseteq \mathbb{K}$
2: $T^{(2)} \leftarrow$ a unif. random $(s+t) \times n$ Toeplitz matrix from $S \subseteq \mathbb{K}$
3: $K \leftarrow H^{\mathsf{T}}T^{(1)}$
4: $L \leftarrow T^{(2)}G$
5: $P \leftarrow AT^{(1)} - GK$
6: $Q \leftarrow T^{(2)}A - LH^{\mathsf{T}}$
7: **return** `RowRankProfile`(P), `ColRankProfile`(Q)

Proof. Applying the Toeplitz preconditioners in Steps 3 and 4 costs $\frac{nt}{r}\tilde{O}(r)$ which is dominated by $nr^{\omega-1}$. □

Proposition 4.3.5. *Algorithm* SparseCRE *computes a CRE decomposition of* $A - GH^{\mathsf{T}}$ *with probability at least* $1 - 2r/|S|$ *in* $\mathsf{T}_{\mathsf{SparseCRE}}(n,r) = \left(\frac{2^\omega-3}{2^{\omega-2}-1}C_\omega + 2C_{RF}\right) nr^{\omega-1} + 2r|A|$ *field operations for* $s + t \leq r$.

Proof. Let ρ be the rank of $A - GH^{\mathsf{T}}$.

$$\begin{aligned}\mathsf{T}_{\mathsf{SparseCRE}}(n,r) &= 2\mathsf{T}_{\mathsf{MM}}(n,r,t) + \mathsf{T}_{\mathsf{CRE}}(\rho,\rho,\rho) + 2\mathsf{T}_{\mathsf{TRSM}}(n-\rho,\rho)\\ &\quad + \mathsf{T}_{\mathsf{SparseRP}}(n,r)\\ &\leq nr^{\omega-1}\left(4C_\omega + \frac{2C_\omega}{2^{\omega-1}-2} + 2C_{\mathsf{RF}}\right) + 2r|A|.\end{aligned}$$

□

Proposition 4.3.6. *Algorithm* LBruhatGen *computes a Left-*Bruhat *generator from a sparse* s*-quasiseparable matrix* $A \in \mathbb{K}^{n\times n}$ *in*

$$\mathsf{T}_{\mathsf{SpGenB}}(n,s,|A|) = \left(\frac{2^{\omega+1}-9}{2^{\omega-1}-2}C_\omega + C_{\mathsf{RF}}\right) ns^{\omega-1}\log n/s + 2s|A|$$

field operations with probability at least $1 - 2n/|S|$.

Proof. First, remark that the G and H matrices correspond to delayed Schur complement updates for pivots processed in the previous calls. Hence, in every call to Algorithm LBruhatGen, these pivots are located to the left, to the top or in the left-top corner of the work matrix. The quasiseparable condition imposes that there are $t \leq 2s$ of them. Moreover, in the call to Algorithm SparseCRE, the ranks verify $r_A + r_B + t \leq s$. Hence we can bound t and write the cost of Algorithm LBruhatGen only in terms of n the dimension of the matrix, s the initial quasiseparability order, and $|\cdot|$ the amount of

nonzero coefficients of the submatrices we consider.

$$\begin{aligned} T(n,s,|A|) \leq & \quad T(n/2,s,|A_2|) + T(n/2,s,|A_3|) \\ & + \mathsf{T}_{\mathsf{SparseCRE}}(n/2,s,|A_1|) \\ & + 2\mathsf{T}_{\mathsf{MM}}(s,2s,n/2) + 2\mathsf{T}_{\mathsf{TRSM}}(s,n/2) \\ \leq & \quad T(n/2,s,|A_2|) + T(n/2,s,|A_3|) + 2s|A_1| \\ & + \left(\frac{2^{\omega+1}-9}{2^{\omega-1}-2}C_\omega + C_{\mathsf{RF}}\right) ns^{\omega-1}. \end{aligned}$$

The failure probability is obtained by a union bound on the failure probability of each of the n/s calls to Algorithm SparseCRE. □

We are not aware of any similar algorithm for computing an SSS or HSS generator using the sparsity of the input matrix and can hence only compare our result to the quadratic generation from a dense matrix.

4.3.5 Experimental comparison

To complement the asymptotic cost analysis, we present in Fig. 4.1 experiments comparing the computation time for the construction of SSS and Bruhat generators. The experiments are made on an implementation of algorithms handling SSS and Bruhat generators over a finite field in the `fflas-ffpack` library [27], at commit 33474b31aa. This library provides efficient dense basic linear algebra routines, such as matrix multiplication, TRSM and Gaussian elimination revealing the rank profile matrix. It was compiled with the GNU C++ compiler `g++` version 9.3.0 and linked with the OpenBLAS library version 0.3.8[1].The benchmarks are run on a single core of an Intel i5-i7300U@2.6GHz running a Linux Mint-20 system.

For all experiments, the matrices have a fixed dimension $n = 3000$, over the finite field $\mathbb{Z}/131071\mathbb{Z}$. We draw the computation times depending on the quasiseparability orders, on three type of instances, indexed by a rank parameter $r \in \{1000, 1500, 1750\}$. This parameter informally measures the rank of the lower and upper triangular parts. It is defined as the number of pivots in the left triangular part of the rank profile matrix of JL and UJ, where L and U are the lower and upper triangular part of the matrix. For a given quasiseparability order, a larger rank parameter forces pivots to be closer to the main diagonal.

Each point corresponds to the mean of the running times of 50 random instances with same parameters. Figure 4.1 compares the running times for the generation from a dense

[1] `https://www.openblas.net`

matrix. We did not consider in these experimental comparisons the straightforward divide and conquer formats such as RRR [72] or $\mathcal{H}$ [29], for they incur at least a logarithmic overhead in their cost estimates. However recent work by [60] suggests that these formats may be competitive for small dimensions, as this overhead may be compensated by a smaller leading constant.

The timings for Bruhat are sub-linear in s, as could be expected from Proposition 4.3.3 but also slightly depends on r which comes from neglected costs arising e.g. from the numerous permutations. The SSS cost is constant on our values for reasons we are unable to explain yet. It is almost always lower than the Bruhat cost. Yet remember that Algorithm DenseToSSS takes the quasiseparable order as input, so it has to be computed first (for example with Algorithm LBruhatGen).

4.4 Application to a block vector

We study here the application of an s-quasiseparable matrix $A \in \mathbb{K}^{n\times n}$ given by its generators (SSS or Bruhat) to a block of v vectors $B \in \mathbb{K}^{n\times v}$. We give the costs for $v \leq s$ (they can be otherwise deduced by slicing B in blocks of s columns).

4.4.1 SSS $\times$ dense

We here recall the algorithm of [12, §2] for computing the product of an SSS matrix with a dense matrix (independently published in [23, Alg. 7.1]). For simplicity, Algorithm LowSSSxDense only details the computations with a strictly lower-block-triangular SSS matrix, that is a matrix whose SSS representation is zero except for the P_i, Q_i and R_i. Extrapolating from there to the product with any SSS matrix can be done by transposing the algorithm for the upper-block-triangular part, and adding the product with the block-diagonal matrix made of the D_i.

Algorithm 4.4.1 LowSSSxDense

Input: P_i, Q_i, R_i for $i \in [\![1, N]\!]$ an s-SSS generator for a strictly lower-block-triangular matrix A; B and C dense $n \times v$ matrices
Output: $C+ = AB$

1: Split $B = \begin{bmatrix} B_1 \\ \vdots \\ B_N \end{bmatrix}$, $C = \begin{bmatrix} C_1 \\ \vdots \\ C_N \end{bmatrix}$ in $s \times s$ blocks
2: $H_1 \leftarrow Q_1 B_1$
3: **for** $i = 2 \dots N$ **do**
4: $\quad H_i \leftarrow Q_i B_i + R_i H_{i-1}$
5: $\quad C_i \leftarrow C_i + P_i H_{i-1}$

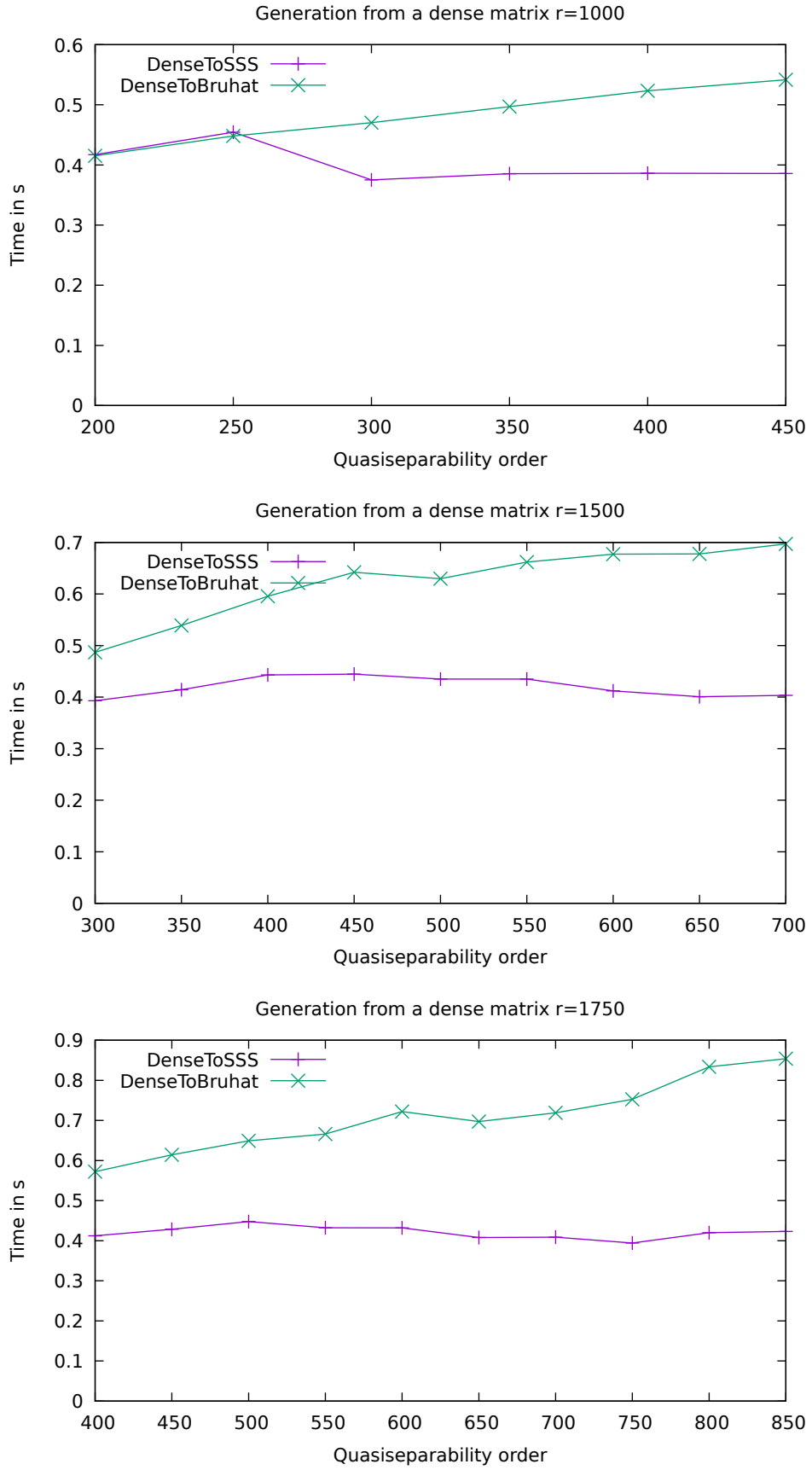


Figure 4.1: Experimental timings for the computation of SSS and Bruhat generators with $n = 3000$ over $\mathbb{Z}/131071\mathbb{Z}$

Proposition 4.4.1. *The product of an $n \times n$ matrix given by its s-SSS generator with an $n \times v$ dense matrix with $v \leq s$ can be computed in $\mathsf{T}_{\mathsf{SxDense}}(n, s, v) = 7C_\omega nsv^{\omega-2}$.*

Proof. In Algorithm LowSSSxDense we have by induction that

$$\text{for } i \in [\![1, N]\!], H_i = \sum_{j=1}^{i} R_i \dots R_{j+1} Q_j B_j. \tag{4.14}$$

As the blocks of the product follow

$$C_i = P_i \sum_{j=1}^{i-1} R_{i-1} \dots R_{j+1} Q_j B_j, \tag{4.15}$$

H_{i-1} can be multiplied once by P_i to compute C_i and once by R_i to compute the following blocks. The cost is $N \times C_\omega s^2 v^{\omega-2}$ for the diagonal blocks and two applications of Algorithm LowSSSxDense in which each step costs $3C_\omega s^2 v^{\omega-2}$. □

4.4.2 Bruhat × dense

Proposition 4.4.2. *The product of an $n \times n$ matrix given by its s-Bruhat generator by a dense $n \times v$ matrix with $v \leq s$ can be computed in $\mathsf{T}_{\mathsf{BxDense}}(n, s, v) = 8C_\omega nsv^{\omega-2}$.*

Proof. This is given by [72, Alg. 14] called twice on the lower and upper triangular part of the quasiseparable matrix. □

Note that in order to benefit from fast matrix multiplication, the Bruhat generator (using $4ns$ space) needs to be transferred into a Compact-Bruhat form, by storing each echelon from into two block diagonal matrices using twice as many field elements (additional ones being zeros). This compression can be done online, hence the space storage remains $4ns$, but the cost of the product by a dense matrix becomes $8C_\omega nst^{\omega-2}$ hence losing the advantage over the SSS format (with cost $7C_\omega nsv^{\omega-2}$ for the same operation).

4.4.3 Experimental comparison

Experimental results are given in Fig. 4.2, which compares the running times for the product by a random dense $n \times 500$ block vector, using the generators resulting from the experiments of Section 4.3.5, in the same experimental framework. As expected from Propositions 4.4.1 and 4.4.2 we obtain costs that are linear in s; we can also observe the same slight dependance in r of the Bruhat cost as in Section 4.3.5. On the parameters we chose, SSS is about four times faster than Bruhat. This can be explained by the compactification of the Bruhat generator needed for the product. This operation involves

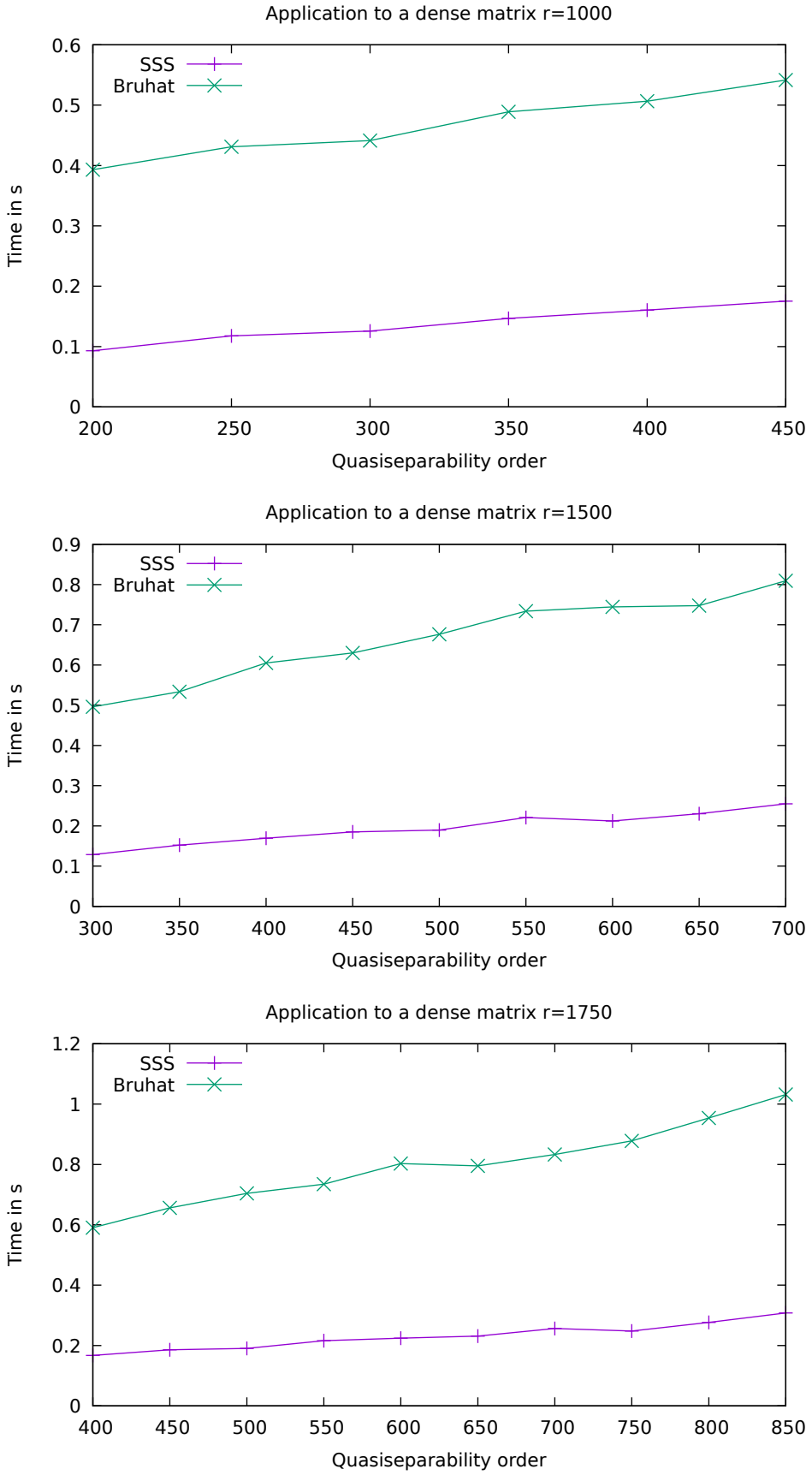


Figure 4.2: Experimental timings for the computation of SSS and Bruhat times a dense matrix with $n = 3000$ and $v = 500$ over $\mathbb{Z}/131071\mathbb{Z}$

numerous data transfers, nonnegligible in practice, although they they do not appear in the cost of Proposition 4.4.2.

4.5 Sum of quasiseparable matrices

The sum and product of two quasiseparable matrices of order s_B and s_C are quasiseparable matrices of order at most s_B+s_C. In this section we show how to compute SSS and Bruhat generators for the sum of two quasiseparable matrices.

The result we give in Proposition 4.5.1 for the sum of matrices given in SSS form can only be used on two generators defined on the same grid. This is a drawback of most operations in SSS which is avoided with the Bruhat format. As a consequence, in a large sequence of operations, the SSS grid size needs to be chosen according to the maximal quasiseparability order among all intermediate results, while the Bruhat always fits to the current quasiseparable order. This can impact the overall cost. The slower original SSS format of [22] avoids this issue, at the expense of multiplying space and time costs by the quasiseparability order, as in [5, 6].

4.5.1 SSS sum

Consider two matrices B and C with the same order s. We first note that the concatenation of the blocks of both input generators leads to matrices which satisfy Eq. (4.1) for $A = C + B$ [12, §10.2].

Let $P_i^{(K)}, V_i^{(K)}, Q_i^{(K)}, U_i^{(K)}, R_i^{(K)}, W_i^{(K)}, D_i^{(K)}$ for appropriate $i \in [\![1, N]\!]$ be an s-SSS representation of K for $K \in \{B, C\}$. The following matrices satisfy Eq. (4.1) with $A = B + C$:

$$P_i = \begin{bmatrix} P_i^{(B)} & P_i^{(C)} \end{bmatrix}, Q_i = \begin{bmatrix} Q_i^{(B)} \\ Q_i^{(C)} \end{bmatrix}, R_i = \begin{bmatrix} R_i^{(B)} & \\ & R_i^{(C)} \end{bmatrix} \tag{4.16}$$

$$U_i = \begin{bmatrix} U_i^{(B)} & U_i^{(C)} \end{bmatrix}, V_i = \begin{bmatrix} V_i^{(B)} \\ V_i^{(C)} \end{bmatrix}, W_i = \begin{bmatrix} W_i^{(B)} & \\ & W_i^{(C)} \end{bmatrix} \tag{4.17}$$

$$D_i = D_i^{(B)} + D_i^{(C)} \tag{4.18}$$

Such sets of matrices with these dimensions satisfying Eq. (4.1) will be called an $(s, 2s)$-SSS generator for A. The granularity of their description remains that of $s \times s$ blocks, but the dimension of the matrices in the representation is doubled and leads to a suboptimal storage size. A second step therefore uses Algorithm SssCompression to form a $2s$-SSS generator and reduce the storage size by $4s(n - 2s)$.

Algorithm 4.5.1 SssCompression

Input: $P_i, Q_i, R_i, U_i, V_i, W_i, D_i$ for appropriate $i \in [\![1, N]\!]$, an $(s, 2s)$-SSS generator for $A \in \mathbb{K}^{n\times n}$

Output: $P'_i, Q'_i, R'_i, U'_i, V'_i, W'_i, D'_i$ for appropriate $i \in [\![1, M]\!]$, a $2s$-SSS representation of A with $M = \lceil N/2 \rceil$

1: **for** $i \leftarrow 1 \dots M$ **do**

2: $\quad P'_i \leftarrow \begin{bmatrix} P_{2i-1} \\ P_{2i}R_{2i-1} \end{bmatrix}$

3: $\quad Q'_i \leftarrow \begin{bmatrix} R_{2i}Q_{2i-1} & Q_{2i} \end{bmatrix}$

4: $\quad R'_i \leftarrow R_{2i}R_{2i-1}$

5: $\quad U'_i \leftarrow \begin{bmatrix} U_{2i-1}W_{2i} \\ U_{2i} \end{bmatrix}$

6: $\quad V'_i \leftarrow \begin{bmatrix} V_{2i-1} & W_{2i-1}V_{2i} \end{bmatrix}$

7: $\quad W'_i \leftarrow W_{2i-1}W_{2i}$

8: $\quad D'_i \leftarrow \begin{bmatrix} D_{2i-1} & U_{2i-1}V_{2i} \\ P_{2i}Q_{2i-1} & D_{2i} \end{bmatrix}$

Proposition 4.5.1. *A $2s$-SSS representation of $B + C \in \mathbb{K}^{n\times n}$ can be computed from s-SSS representations of B and C in time $\mathsf{T}_{\mathsf{S+S}}(n, s) = (10 + 2^\omega)\, C_\omega n s^{\omega-1}$.*

Proof. For any $s \times s$ block $A_{i,j}$ of $A = B + C$, it can be checked that the representation in the output of Algorithm SssCompression called on the generator of Section 4.5.1 matches. The additions of Eq. (4.18) are dominated by the call to Algorithm SssCompression whose cost is of M steps with four $2s \times 2s$ by $2s \times s$ products, two $2s \times 2s$ square products, and two $s \times 2s$ by $2s \times s$ products. □

Note that the $(s, 2s)$-SSS generator is intermediate between the SSS form and the original definition of quasiseparable matrices given in [22], where the generators are $s \times s$ matrices but the granularity of the description is of dimension 1.

4.5.2 Bruhat sum

As with SSS, the sum of two matrices in Bruhat form can be computed by first concatenation of both generators, then by retrieving the Bruhat format in a second step.

Given two left triangular matrices A and B given by Bruhat generators $C^{(A)}, R^{(A)}, E^{(A)}$, $C^{(B)}, R^{(B)}, E^{(B)}$, their sum indeed writes

$$A + B = \triangledown \left(\begin{bmatrix} C^{(A)} & C^{(B)} \end{bmatrix} \begin{bmatrix} R^{(A)} & \\ & R^{(B)} \end{bmatrix} \begin{bmatrix} E^{(A)} \\ E^{(B)} \end{bmatrix} \right). \tag{4.19}$$

A Bruhat generator for the right side in Eq. (4.19) can be obtained from a call to Algorithm LBruhatGen, viewed here as a compression algorithm. This relies on a specific CRE

decomposition (Algorithm BruhatSumCRE), and on having $D_{\mathcal{R},*}$ for D a submatrix of a sum given as in Eq. (4.19) and $\mathcal{R}$ a set of row indices (Proposition 4.5.3).

Algorithm 4.5.2 BruhatSumCRE

Input: $A, B \in \mathbb{K}^{n\times n}$ of rank $\leq r_A$ and $\leq r_B$ given by generators $C^{(A)}, R^{(A)}, E^{(A)}, C^{(B)}, R^{(B)}, E^{(B)}$ s.t. $A = C^{(A)}R^{(A)}E^{(A)}$ and $B = C^{(B)}R^{(B)}E^{(B)}$ which are submatrices of Bruhat generators of matrices comprising A and B
Input: $G, H \in \mathbb{K}^{n\times t}$
Output: C, R, E such that $A + B = CRE + GH^{\mathsf{T}}$

1: $C^{(R)}, R^{(R)}, E^{(R)} \leftarrow \texttt{DenseCRE}\left(\begin{bmatrix} R^{(A)}E^{(A)} \\ R^{(B)}E^{(B)} \\ -H^{\mathsf{T}} \end{bmatrix}\right)$
2: $C^{(L)}, R^{(L)}, E^{(L)} \leftarrow \texttt{DenseCRE}\left(\begin{bmatrix} C^{(A)} & C^{(B)} & G \end{bmatrix}\right)$
3: $X \leftarrow R^{(L)}E^{(L)}C^{(R)}R^{(R)}$
4: $C^{(X)}, R^{(X)}, E^{(X)} \leftarrow \texttt{DenseCRE}(X)$
5: $C \leftarrow C^{(L)}C^{(X)}$
6: $R \leftarrow R^{(X)}$
7: $E \leftarrow E^{(X)}E^{(R)}$

Proposition 4.5.2. *Algorithm BruhatSumCRE computes a CRE decomposition of* $A+B-GH^{\mathsf{T}}$ *in* $\mathsf{T}_{\mathsf{BSumCRE}}(n, r) = (3C_\omega + 2C_{RF})\, nr^{\omega-1}$ *for* $r_A + r_B + t \leq r$.

Proof. The matrices C and E are in column and row echelon form respectively as they are products of two echelon forms. The cost is that of two dense CRE decompositions of size $n\times(r_A+r_B+t)$ and products of an $n\times(r_A+r_B+t)$ matrix by two $(r_A+r_B+t)\times(r_A+r_B+t)$ and one $(r_A + r_B + t) \times n$ matrices. □

Proposition 4.5.3. *For* $D \in \mathbb{K}^{n\times n}$ *a submatrix of a the left-triangular part of a sum as in Eq. (4.19) and* $\mathcal{R}$ *a set of* s *row indices,* $D_{\mathcal{R},*}$ *can be computed in* $\mathsf{T}_{\mathsf{SumExp}}(n, s) = C_\omega n s^{\omega-1}$.

Proof. There are at most s_A (resp. s_B) pivots of A (resp. B) impacting D. We can thus write $D = CRE$ with C made of n rows and $s_A + s_B$ columns of $\begin{bmatrix} C^{(A)} & C^{(B)} \end{bmatrix}$, R a permutation and E made of n columns and s_A rows of $E^{(A)}$ and s_B rows of $E^{(B)}$. □

Proposition 4.5.4. *The* Bruhat *form of the sum of two* $n \times n$ *matrices of respective quasiseparable orders* s_A *and* s_B *given in* Bruhat *form can be computed in* $\mathsf{T}_{\mathsf{B+B}}(n, s) = \left(\frac{9\cdot 2^{\omega-2}-8}{2^{\omega-2}-1}C_\omega + 2C_{\mathsf{RF}}\right) ns^{\omega-1} \log n/s$ *field operations for* $s = s_A + s_B$.

Proof. Each lower and upper triangular part is converted to a left triangular instance and computed independently. Algorithm LBruhatGen is then called twice with $t = 0$ on an input matrix in factorised form as in (4.19).

The proof is the same as for Proposition 4.3.5 except that in the cost, the $\mathsf{T}_{\mathsf{SparseCRE}}$ terms are replaced by $\mathsf{T}_{\mathsf{BruhatSumCRE}}$ terms and the rows and columns of the submatrices

are computed at a cost given by $\mathsf{T}_{\mathsf{SumExp}}$. Then we have

$$\begin{aligned} T(n,s) \leq & \quad 2T(n/2,s) + \mathsf{T}_{\mathsf{BSumCRE}}(n/2,s) + 2\mathsf{T}_{\mathsf{SumExp}}(n/2,s,s) \\ & + 2\mathsf{T}_{\mathsf{MM}}(s,2s,n/2) + 2\mathsf{T}_{\mathsf{TRSM}}(s,n/2) \\ \leq & \quad 2T(n/2,s) + \left(\frac{9\cdot 2^{\omega-3}-4}{2^{\omega-2}-1}C_\omega + C_{\mathsf{RF}}\right) ns^{\omega-1} \end{aligned}$$

for one call to Algorithm LBruhatGen. □

4.6 Product in SSS

The product of two matrices given in SSS form uses two tricks we have seen previously. The first one is to start by computing an $(s, 2s)$-SSS representation before compression, as in the sum. Unlike the sum, computations are needed in addition to concatenation to get this representation. The second trick is to speed up these computations by using a Horner-like accumulation as in Algorithm LowSSSxDense. This accumulation will be done on both sides for the computation of all necessary products $A_{i,k}B_{k,j}$ where $A_{i,k}$ is under (resp. over) the diagonal and $B_{k,j}$ is over (resp. under) it.

Algorithm SSSxSSS details these computations, using the G_i and H_i as accumulators. It presents an improvement over the algorithm of [12, §3] and [23, Alg. 7.2]: 4 products have been avoided at each step by keeping them in memory in the T_i and S_i. They can also be avoided in the numerical context.

Theorem 4.6.1. *Algorithm SSSxSSS computes a $2s$-SSS generator for the product of two $n \times n$ matrices given in s-SSS form in $\mathsf{T}_{\mathsf{SSSxSSS}}(n,s) = (31+2^\omega)\,C_\omega n s^{\omega-1}$.*

Proof. Using Lines 2 and 4 for G_i and Lines 10 and 14 for H_i, induction on i shows that

$$G_i = \sum_{k=1}^{i-1} R_{i-1}^{(A)} \dots R_{k+1}^{(A)} Q_k^{(A)} U_k^{(B)} W_{k+1}^{(B)} \dots W_{i-1}^{(B)} \tag{4.20}$$

$$H_i = \sum_{k=i+1}^{N} W_{i+1}^{(A)} \dots W_{k-1}^{(A)} V_k^{(A)} P_k^{(B)} R_{k-1}^{(B)} \dots R_{i+1}^{(B)} \tag{4.21}$$

Combining these results with Step 3 for S_i, Step 11 for T_i and finally Step 12, we get that $D_i^{(C)} = \sum_{k=1}^{N} A_{i,k}B_{k,i} = C_{i,i}$.

When $i < j$, the products $A_{i,k}B_{k,j}$ take five shapes: lower block of A × upper block of B, diagonal block × upper block, upper × upper, upper × diagonal and upper × lower.

Algorithm 4.6.1 SSSxSSS

Input: For both $M \in \{A, B\}$, $P_i^{(M)}, Q_i^{(M)}, R_i^{(M)}, U_i^{(M)}, V_i^{(M)}, W_i^{(M)}, D_i^{(M)}$ for appropriate $i \in [\![1, N]\!]$ an s-SSS generator for M

Output: A $2s$-SSS generator for $C = AB$

▷ *All values not given as input are initialised to 0*

for $i \leftarrow 1 \dots N$ **do**

$G_i \leftarrow Q_{i-1}^{(A)} U_{i-1}^{(B)} + T_{i-1} W_{i-1}^{(B)}$

$T_i \leftarrow R_i^{(A)} G_i$

$S_i \leftarrow P_i^{(A)} G_i$

$$Q_i \leftarrow \begin{bmatrix} Q_i^{(B)} \\ Q_i^{(A)} D_i^{(B)} + T_i V_i^{(B)} \end{bmatrix}$$

$$R_i \leftarrow \begin{bmatrix} R_i^{(B)} & 0 \\ Q_i^{(A)} P_i^{(B)} & R_i^{(A)} \end{bmatrix}$$

$$U_i \leftarrow \begin{bmatrix} U_i^{(A)} & D_i^{(A)} U_i^{(B)} + S_i W_i^{(B)} \end{bmatrix}$$

$$W_i \leftarrow \begin{bmatrix} W_i^{(A)} & V_i^{(A)} U_i^{(B)} \\ 0 & W_i^{(B)} \end{bmatrix}$$

for $i \leftarrow N \dots 1$ **do**

$H_i \leftarrow V_{i+1}^{(A)} P_{i+1}^{(B)} + T_{i+1} R_{i+1}^{(B)}$

$T_i \leftarrow U_i^{(A)} H_i$

$D_i \leftarrow D_i^{(A)} D_i^{(B)} + S_i V_i^{(B)} + T_i Q_i^{(B)}$

$P_i^C \leftarrow \begin{bmatrix} D_i^{(A)} P_i^{(B)} + T_i R_i^{(B)} & P_i^{(A)} \end{bmatrix}$

$T_i \leftarrow W_i^{(A)} H_i$

$$V_i \leftarrow \begin{bmatrix} V_i^{(A)} D_i^{(B)} + T_i Q_i^{(B)} \\ V_i^{(B)} \end{bmatrix}$$

return SssCompression$((P_i, Q_i, R_i, U_i, V_i, W_i, D_i)_{i \in [\![1,N]\!]})$

The equality

$$U_i^{(C)} W_{i+1}^{(C)} \dots W_{j-1}^{(C)} V_j^{(C)} = \sum_{k=1}^{N} A_{i,k} B_{k,j} \tag{4.22}$$

and its counterpart when $i > j$ can be checked with tedious but straightforward calculations.

The cost is that of 21 products and 8 sums of $s \times s$ matrices at each of the N steps and on call to Algorithm SssCompression. □

Again the result of Theorem 4.6.1 is limited to matrices defined on the same grid and the result always has the same storage size, whatever its quasiseparability order. This is also true for product with HSS generators in numerical analysis [74]. The Bruhat format can avoid these issues, but to our knowledge no sub-quadratic algorithm exists for the

product of two Bruhat generators. The method used for the sum in Section 4.5.2 opens the door towards a linear or quasi-linear product algorithm using Algorithm LBruhatGen.

Bibliography

[1] J. Alman and V. V. Williams. A Refined Laser Method and Faster Matrix Multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

[2] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Springer, 1rst edition, 2003.

[3] B. Beckermann and G. Labahn. A uniform approach for the fast computation of matrix-type Padé approximants. *SIAM J. Matrix Analysis and Applications*, 15(3):804–823, 1994.

[4] V. Bhargava, S. Ghosh, Z. Guo, M. Kumar, and C. Umans. Fast multivariate multipoint evaluation over all finite fields. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 221–232, 2022.

[5] P. Boito, Y. Eidelman, and L. Gemignani. Implicit QR for rank-structured matrix pencils. *BIT Numerical Mathematics*, 54(1):85–111, Mar. 2014.

[6] P. Boito, Y. Eidelman, and L. Gemignani. A real QZ algorithm for structured companion pencils. *Calcolo*, 54(4):1305–1338, Dec. 2017.

[7] A. Bostan, P. Flajolet, B. Salvy, and É. Schost. Fast computation of special resultants. *J. Symb. Comput.*, 41(1):1–29, 2006.

[8] A. Bostan, C.-P. Jeannerod, C. Mouilleron, and É. Schost. On matrices with displacement structure: generalized operators and faster algorithms. *SIAM J. on Matrix Analysis and Applications*, 38(3):733–775, 2017.

[9] A. Bostan, C.-P. Jeannerod, and É. Schost. Solving structured linear systems with large displacement rank. *Theoretical Computer Science*, 407(1-3):155–181, 2008.

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

[11] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A. J. van der Veen, and D. White. Some fast algorithms for sequentially semiseparable representations. *SIAM Journal on Matrix Analysis and Applications*, 27(2):341–364, 2005.

[12] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, and A. J. van der Veen. Fast stable solver for sequentially semi-separable linear systems of equations. In *High Performance Computing — HiPC 2002*, pages 545–554, 2002.

[13] S. Chandrasekaran and M. Gu. Fast and stable algorithms for banded plus semiseparable systems of linear equations. *SIAM J. Matrix Analysis Applications*, 25:373–384, 2003.

[14] S. Chandrasekaran, M. Gu, and T. Pals. A fast ULV decomposition solver for hierarchically semiseparable representations. *SIAM Journal on Matrix Analysis and Applications*, 28(3):603–622, 2006.

[15] D. Coppersmith. Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm. *Math. Comput.*, 62(205):333–350, 1994.

[16] S. Delvaux and M. Van Barel. A Givens-weight representation for rank structured matrices. *SIAM Journal on Matrix Analysis and Applications*, 29(4):1147–1170, 2008.

[17] J. Dixon. Exact solution of linear equations using p-adic expansions. *Numerische Mathematik*, 40(1):137–141, 1982.

[18] R. Duan, H. Wu, and R. Zhou. Faster Matrix Multiplication via Asymmetric Hashing. arXiv:2210.10173, 2022.

[19] J. G. Dumas, T. Gautier, and C. Pernet. Finite field linear algebra subroutines. In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, page 63–74, 2002.

[20] J.-G. Dumas, C. Pernet, and Z. Sultan. Fast computation of the rank profile matrix and the generalized Bruhat decomposition. *Journal of Symbolic Computation*, 83:187–210, 2017. Special issue on the conference ISSAC 2015: Symbolic computation and computer algebra.

[21] W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, and G. Villard. Faster inversion and other black box matrix computation using efficient block projections. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, pages 143–150. ACM Press, 2007.

[22] Y. Eidelman and I. Gohberg. On a new class of structured matrices. *Integral Equations and Operator Theory*, 34:293–324, 1999.

[23] Y. Eidelman and I. Gohberg. On generators of quasiseparable finite block matrices. *Calcolo*, 42:187–214, 12 2005.

[24] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999. Third edition 2013.

[25] J. von zur Gathen and T. Lücking. Subresultants revisited. *Theoretical Computer Science*, 297(1-3):199–239, 2003.

[26] P. Giorgi, C. Jeannerod, and G. Villard. On the complexity of polynomial matrix computations. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, pages 135–142. ACM Press, 2003.

[27] T. F.-F. group. *FFLAS-FFPACK: Finite Field Linear Algebra Subroutines / Package*, v2.5.0 edition, 2021. `http://github.com/linbox-team/fflas-ffpack`.

[28] W. Hackbusch. A sparse matrix arithmetic based on H-matrices. Part I: Introduction to H-matrices. *Computing*, 62:89–108, 1999.

[29] W. Hackbusch. *Hierarchical Matrices: Algorithms and Analysis*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[30] W. Hackbusch, B. Khoromskij, and S. A. Sauter. On H2-matrices. In H.-J. Bungartz, R. H. W. Hoppe, and C. Zenger, editors, *Lectures on Applied Mathematics*, pages 9–29, 2000.

[31] G. Heinig, P. Jankowski, and K. Rost. Fast inversion algorithms of toeplitz-plus-hankel matrices. *Numerische Mathematik*, 52(6):665–682, 1988.

[32] G. Heinig and K. Rost. *Algebraic Methods for Toeplitz-like Matrices and Operator*. Springer, Birkhäuser Basel, 1984.

[33] G. Heinig and K. Rost. New fast algorithms for Toeplitz-plus-Hankel matrices. *SIAM J. Matrix Analysis and Applications*, 25(3):842–857, 2004.

[34] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2002.

[35] J. v. d. Hoeven and G. Lecerf. Fast computation of generic bivariate resultants. *J. of Complexity*, 2020.

[36] J. van der Hoeven and R. Larrieu. Fast Gröbner basis computation and polynomial reduction for generic bivariate ideals. *Appl. Algebr. Eng. Comm.*, 30(6):509–539, 2019.

[37] J. van der Hoeven and G. Lecerf. Fast computation of generic bivariate resultants. *J. of Complexity*, 62, 2021.

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

[39] C.-P. Jeannerod, V. Neiger, and G. Villard. Fast computation of approximant bases in canonical form. *J. Symbolic Computation*, 98:192–224, 2020.

[40] C.-P. Jeannerod, C. Pernet, and A. Storjohann. Rank-profile revealing Gaussian elimination and the CUP matrix decomposition. *J. Symb. Comput.*, 56:46–68, 2013.

[41] T. Kailath. *Linear Systems*. Prentice-Hall, 1980.

[42] T. Kailath, S. Kung, and M. Morf. Displacement ranks of matrices and linear equations. *J. Mathematical Analysis and Applications*, 68(2):395–407, 1979.

[43] E. Kaltofen. Asymptotically fast solution of Toeplitz-like singular linear systems. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, pages 297–304. ACM Press, 1994.

[44] E. Kaltofen. Challenges of symbolic computation: my favorite open problems. *J. Symbolic Computation*, 29(6):891–919, 2000.

[45] E. Kaltofen, M. Krishnamoorthy, and B. Saunders. Parallel algorithms for matrix normal forms. *Linear Algebra and its Applications*, 136:189–208, 1990.

[46] E. Kaltofen and B. Saunders. On Wiedemann's method of solving sparse linear systems. In *Proc. AAECC-9*, LNCS 539, Springer Verlag, pages 29–38, 1991.

[47] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Comput. Complex.*, 13(3):91–130, 2005.

[48] E. Kaltofen and G. Yuhasz. On the matrix Berlekamp-Massey algorithm. *ACM Trans. Algorithms*, 9(4):33:1–33:24, 2013.

[49] P. Karpman, C. Pernet, H. Signargout, and G. Villard. Computing the characteristic polynomial of generic Toeplitz-like and Hankel-like matrices. In *Proceedings of the 2021 on International Symposium on Symbolic and Algebraic Computation*, pages 249–256, 2021.

[50] K. S. Kedlaya and C. Umans. Fast polynomial factorization and modular composition. *SIAM J. on Computing*, 40(6):1767–1802, 2011.

[51] W. Keller-Gehrig. Fast algorithms for the characteristic polynomial. *Theoretical computer science*, 36:309–317, 1985.

[52] G. Labahn, V. Neiger, and W. Zhou. Fast, deterministic computation of the Hermite normal form and determinant of a polynomial matrix. *J. Complexity*, 42:44–71, 2017.

[53] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, pages 296–303. ACM Press, 2014.

[54] F. Le Gall. Faster Rectangular Matrix Multiplication by Combination Loss Analysis. arXiv:2307.06535, 2023.

[55] F. Le Gall and F. Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proc. ACM-SIAM SODA*, pages 1029–1046, 2018.

[56] G. Lecerf. On the complexity of the Lickteig-Roy subresultant algorithm. *J. Symb. Comput.*, 92:243–268, 2019.

[57] W. Lyons. *Fast algorithms with applications to PDEs*. University of California, Santa Barbara, 2005.

[58] W. Manthey and U. Helmke. Bruhat canonical form for linear systems. *Linear Algebra and its Applications*, 425(2–3):261–282, Sept. 2007.

[59] P. Martinsson. A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix. *SIAM J. Matrix Analysis Applications*, 32:1251–1274, 2011.

[60] S. Massei, L. Robol, and D. Kressner. hm-toolbox: MATLAB software for HODLR and HSS matrices. *SIAM Journal on Scientific Computing*, 42(2):C43–C68, 2020.

[61] R. Moenck and J. Carter. Approximate algorithms to derive exact solutions to systems of linear equations. In *Proc. EUROSAM*, LNCS 72, pages 63–73, 1979.

[62] G. Moroz and É. Schost. A fast algorithm for computing the truncated resultant. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, pages 341–348. ACM Press, 2016.

[63] V. Neiger, B. Salvy, É. Schost, and G. Villard. Faster modular composition. *CoRR*, abs/2110.08354, 2021.

[64] M. Newman. *Integral Matrices*. Academic Press, 1972. First edition.

[65] V. Y. Pan. Strassen's algorithm is not optimal. Trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations. In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pages 166–176, 1978.

[66] V. Y. Pan. *Structured Matrices and Polynomials: Unified Superfast Algorithms*. Springer-Verlag, Berlin, Heidelberg, 2001.

[67] M. Paterson and L. J. Stockmeyer. On the Number of Nonscalar Multiplications Necessary to Evaluate Polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.

[68] C. Pernet. Computing with quasiseparable matrices. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '16, pages 389–396, New York, NY, USA, 2016. ACM.

[69] C. Pernet, H. Signargout, and G. Villard. High-order lifting for polynomial Sylvester matrices. working paper or preprint, Oct. 2022.

[70] C. Pernet, H. Signargout, and G. Villard. Exact computations with quasiseparable matrices. In *Proceedings of the 2023 on International Symposium on Symbolic and Algebraic Computation*, 2023.

[71] C. Pernet, H. Signargout, and G. Villard. Leading constants of rank deficient Gaussian elimination. Technical report, 2023. `hal:03976168`.

[72] C. Pernet and A. Storjohann. Time and space efficient generators for quasiseparable matrices. *Journal of Symbolic Computation*, 85:224 – 246, 2018. Special issue on the 41th International Symposium on Symbolic and Algebraic Computation (ISSAC'16).

[73] D. Reischert. Asymptotically fast computation of subresultants. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, pages 233–240. ACM Press, 1997.

[74] Z. Sheng, P. Dewilde, and S. Chandrasekaran. *Algorithms to Solve Hierarchically Semi-separable Systems*, volume 176, pages 255–294. 2007.

[75] H. P. Starr. *On the Numerical Solution of One-Dimensional Integral and Differential Equations*. PhD thesis, USA, 1992. UMI Order No. GAX92-35558.

[76] A. Storjohann. *Algorithms for Matrix Canonical Forms*. PhD thesis, Institut für Wissenschaftliches Rechnen, ETH-Zentrum, Zürich, Switzerland, Nov. 2000.

[77] A. Storjohann. High-order lifting and integrality certification. *J. Symbolic Computation*, 36(3-4):613–648, 2003.

[78] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13(4):354–356, aug 1969.

[79] M. Van Barel and A. Bultheel. A general module theoretic framework for vector M-Padé and matrix rational interpolation. *Numerical Algorithms*, 3:451–462, 1992.

[80] R. Vandebril, M. V. Barel, G. H. Golub, and N. Mastronardi. A bibliography on semiseparable matrices. *CALCOLO*, 42:249–270, 2005.

[81] R. Vandebril, M. Van Barel, G. H. Golub, and N. Mastronardi. *Matrix Computations and Semiseparable Matrices: Linear Systems*. Johns Hopkins University Press, 2008.

[82] G. Villard. A study of Coppersmith's block Wiedemann algorithm using matrix polynomials. RR 975 IM IMAG, 1997.

[83] G. Villard. On computing the resultant of generic bivariate polynomials. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, pages 391–398. ACM Press, 2018.

[84] G. Villard. Elimination ideal and bivariate resultant over finite fields. In *International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 526–534. ACM Press, 2023.

[85] V. V. Williams, Y. Xu, Z. Xu, and R. Zhou. New Bounds for Matrix Multiplication: from Alpha to Omega. arXiv:2307.07970, 2023.

[86] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li. Fast algorithms for hierarchically semiseparable matrices. *Numerical Linear Algebra with Applications*, 17(6):953–976, 2010.