On the complexity of the Lickteig-Roy subresultant algorithm

Grégoire Lecerf

On the complexity of the Lickteig–Roy subresultant algorithm

Grégoire Lecerf

Laboratoire d'informatique de l'École polytechnique, UMR 7161 CNRS
École polytechnique
91128 Palaiseau Cedex, France

Email: lecerf@lix.polytechnique.fr

Preprint version of January 31, 2017

In their 1996 article, Lickteig and Roy introduced a fast "divide and conquer" variant of the subresultant algorithm which avoids coefficient growth in defective cases. The present article concerns the complexity analysis of their algorithm over effective rings endowed with the partially defined division routine. This leads to new convenient complexity bounds for gcds, especially when coefficients are in abstract polynomial rings where evaluation/interpolation schemes are not supposed to be available.

1. Introduction

Euclidean polynomial remainder sequences are a cornerstone of computer algebra for gcd, lcm, modular inversion, Berlekamp–Massey algorithm, Padé approximant, etc. This article brings new complexity results for computing subresultant polynomials over commutative rings endowed with the partially defined division routine. We design a "divide and conquer" algorithm with a bounded coefficient growth even in defective cases. In particular, this leads to new deterministic complexity bounds for bivariate gcds.

1.1. Notations and definitions

Until the end of the article, $\mathbb{A}$ is a commutative ring with unity, and $\mathrm{Frac}(\mathbb{A})$ is its *total ring of fractions* (also called the *total quotient ring*), namely $S^{-1}\,\mathbb{A}$ where S is the set of the nonzero divisors in $\mathbb{A}$. In other words, elements of $\mathrm{Frac}(\mathbb{A})$ are of the form $a\,/\,s$, with $a \in \mathbb{A}$ and $s \in S$. When $\mathbb{A}$ is integral, that is an integral domain, then $\mathrm{Frac}(\mathbb{A})$ is its *field of fractions*.

Divisions. For algorithmic purposes, $\mathbb{A}$ is assumed to be endowed with the partially defined division routine: precisely, if a and b are two elements of $\mathbb{A}$ such that b divides a, then this routine returns $a\,/\,b$. Operations in $\mathbb{A}$ at our disposal are: addition, subtraction, multiplication, and this partially defined division. Let A and B be two polynomials in $\mathbb{A}[x]$. If $B \neq 0$, we say that the division of A by B is *well defined* when there exist Q and R in $\mathbb{A}[x]$ such that $A = Q\,B + R$ and $\deg R < \deg G$. These polynomials Q and R are respectively written $\mathrm{quo}(A, B)$ and $\mathrm{rem}(A, B)$.

The *leading coefficient* of a polynomial A is written $\mathrm{lc}(A)$. The *pseudo-division* of A by $B \neq 0$ in $\mathbb{A}[x]$ is the division of $\mathrm{lc}(B)^{\deg A - \deg B + 1}\,A$ by B: the remainder (resp. quotient), written $\mathrm{prem}(A,\, B)$ (resp. $\mathrm{pquo}(A,\, B)$), is called the *pseudo-remainder* (resp. *pseudo-quotient*). Pseudo-divisions have the advantage to be well defined and easily computable without divisions in $\mathbb{A}$.

Subresultant and Euclidean sequences. Our two input polynomials are written $F = \sum_{i=0}^{n_0} f_i\, x^i$ and $G = \sum_{i=0}^{n_1} g_i\, x^i$, and are of respective degrees n_0 and n_1. Throughout the article, we assume that $n_0 \geqslant n_1$. For $0 \leqslant k < n_1$, the kth *subresultant coefficient* of F and G is written $s_k \in \mathbb{A}$, and the associated kth *subresultant polynomial* $S_k \in \mathbb{A}[x]$ (usual definitions are recalled in Section 2.1). The subresultant polynomial S_k is said to be *defective* when its degree is strictly less than k. Of course, when $n_0 < n_1$, without loss of generality, we may swap the two input polynomials since the subresultant sequences coincide up to signs.

The *extended Euclidean algorithm* consists in computing the remainder sequence, recursively defined by $R_0 = F$, $C_0 = 1$, $D_0 = 0$, $R_1 = G$, $C_1 = 0$, $D_1 = 1$, and $R_{i+1} = R_{i-1} - E_i\, R_i$, $C_{i+1} = C_{i-1} - E_i\, C_i$, $D_{i+1} = D_{i-1} - E_i\, D_i$, where $E_i = \mathrm{quo}(R_{i-1}, R_i)$. Consequently we have $R_{i+1} = \mathrm{rem}(R_{i-1}, R_i)$, and $R_i = C_i\, F + D_i\, G$ for all $i \geqslant 0$. The *extended Euclidean sequence* $(R_i, C_i, D_i)_i$ is well defined over $\mathrm{Frac}(\mathbb{A})$ whenever the leading coefficients of the nonzero R_i are invertible in $\mathrm{Frac}(\mathbb{A})$. In this case, the sequence ends after $w - 1$ division steps with $R_i \neq 0$ for all $0 \leqslant i \leqslant w$, and $R_{w+1} = 0$ — the last nonzero polynomial R_w is $\gcd(F, G)$ whenever $\mathbb{A}$ is an integral domain.

For all $0 \leqslant i \leqslant w + 1$, we let $n_i = \deg R_i$, and call $(n_i)_i$ the *Euclidean degree sequence*. This sequence is said to be *normal* when $n_{i+1} = n_i - 1$ for all $1 \leqslant i < w$. When $n_{i+1} < n_i - 1$ for some i we say that a degree *gap* occurs at step i. In addition, it is easy to verify that $\deg C_i < n_1 - n_i$ and $\deg D_i < n_0 - n_i$ by induction on $1 < i \leqslant w + 1$.

Complexity model. For complexities, we shall use *computation trees* over $\mathbb{A}$ with the *total complexity* point of view. This means that complexity estimates charge a constant cost for each arithmetic operation in $\mathbb{A}$ (addition, subtraction, multiplication, and division in our framework) and the equality test. All constants in $\mathbb{A}$ are though to be freely at our disposal. See definitions in [16, Chapter 4].

A univariate polynomial of degree n is represented by the vector of its $n+1$ coefficients. We write $\mathsf{M}\colon \mathbb{N} \to \mathbb{Z}$ for a function that bounds the cost of a polynomial product algorithm in terms of the number of ring operations performed independently of the coefficient ring, assuming a unity is available. In other words, two polynomials of degrees at most n over such a ring $\mathbb{A}$ may be multiplied with $\mathsf{M}(n)$ arithmetic operations in $\mathbb{A}$. The fastest known algorithm, due to Cantor and Kaltofen [17], provides us with $\mathsf{M}(n) = O(n \log n \log\log n) = \tilde{O}(n)$. Here, the *soft-Oh* notation $f(n) \in \tilde{O}(g(n))$ means that $f(n) = g(n) \log^{O(1)} g(n)$ (we refer the reader to [24, Chapter 25, Section 7] for technical details). In order to simplify cost analyses, we make the customary assumption that $n \mapsto \mathsf{M}(n)\,/\,n$ is non decreasing, which implies the *super-additivity* of M, namely $\mathsf{M}(n_1) + \mathsf{M}(n_2) \leqslant \mathsf{M}(n_1 + n_2)$ for all $n_1 \geqslant 0$ and $n_2 \geqslant 0$.

Over concrete rings or fields, explicitly presented over a finite field, or $\mathbb{Z}$, we shall use Turing machines endowed with sufficiently many tapes, in order to benefit from standard algorithms. Integers are though to be represented by their binary expansion. Additions take linear time, and we write $\mathsf{I}(n)$ for a function that bounds the cost of an integer product in size n, with the same customary assumptions as for M.

1.2. Related work

The Euclidean algorithm has a long history in computational mathematics, which goes back to Euclid. Nowadays it is widely used in computer algebra systems for gcds of integers and polynomials, with softly linear time in most common situations. The *naive algorithm* has quadratic cost in the input size (in the Turing model for integers, and in the computation tree model for univariate polynomials over abstract fields). The key ideas of the fast "divide and conquer" algorithm are due to Lehmer, for integers [35]: Euclidean quotients only depend on higher bits, and their total bit size does not exceed the input size. The first softly linear cost has been achieved by Knuth [33], namely $O(\mathsf{I}(n) \log^4 n)$ for input size n.

At present time, the best known asymptotic complexity bound is owned by Schönhage [42], namely $O(\mathsf{I}(n)\log n)$. For polynomials over fields, these algorithms have been adapted first to normal sequences by Moenck [40], and Aho, Hopcroft, Ullman [3], and then completed by Brent, Gustavson, and Yun [13], who reached the analogue cost $O(\mathsf{M}(n)\log n)$ for general sequences and input degree n. We shall refer to these fast variants of the Euclidean algorithms as the *half-gcd algorithm*. For modern pleasant presentations of this algorithm we refer the reader to classical books [16, 18, 24, 43].

In the polynomial case, the half-gcd algorithm works fine when coefficient sizes do not grow during the computation (typically over finite fields). Over integers or polynomial rings, it is well known that intermediate sizes grow very quickly, and straightforward implementations are not practical even in medium sizes. The subresultant theory provides a nice solution to this issue. First, it gives a simple condition to decide if a given degree occurs in the Euclidean degree sequence. Second, it offers polynomial expressions for the coefficients of these remainders in terms of the coefficients of the input polynomials (up to a suitable renormalisation). These polynomial expressions even turn out to be irreducible over $\mathbb{Z}$ [23, Lemma 4.2]. Informally speaking, this means that subresultant polynomials have generically optimal sizes, and are thus convenient representative of Euclidean remainders.

Subresultant coefficients were introduced by Sylvester in 1840 as determinants of matrices nowadays called *Sylvester matrices*. The terminology "subresultant" was coined latter by Bôcher in 1907 [12] for the subresultant coefficients. Straightforward naive computations of the subresultant polynomials by means of their defining determinants lead essentially to a cubic cost, which is far from efficient in general. The first algorithm with quadratic cost goes back to Habicht [27]: he showed that only the subresultant polynomials S_{n_i} and S_{n_i-1} are nonzero, and that they may be computed recursively as follows, by means of pseudo-divisions:

$$S_{n_{i+1}} = (s_{n_i-1,n_i}^{n_i-n_{i+1}-1} S_{n_i-1}) / s_{n_i}^{n_i-n_{i+1}-1}, \quad (1)$$

$$S_{n_{i+1}-1} = \operatorname{prem}(S_{n_i}, S_{n_i-1}) / s_{n_i}^{n_i-n_{i+1}}, \quad (2)$$

where s_{n_i-1,n_i} represents the coefficient of degree n_i in S_{n_i-1}. In particular, if all the nonzero subresultant coefficients s_k are invertible in $\operatorname{Frac}(\mathbb{A})$, then each nonzero subresultant polynomial is proportional over $\operatorname{Frac}(\mathbb{A})$ to the Euclidean remainder of the same degree (in particular the Euclidean sequence is well defined).

The first use of subresultants in computer algebra is due to Collins [19], who coined the "subresultant polynomial" terminology, and reported on the practical impact. He also showed lower bounds for the expression swell in the Euclidean algorithm. Then, Brown and Traub [15], followed by Loos [39], extended Collins' algorithm over unique factorization domains, with easier proofs for the proportionality to the Euclidean remainder sequence. Overall, this led to major practical algorithms. First, over a field, one may compute whatever polynomial remainder sequence (including using the half-gcd algorithm) in order to deduce subresultants by suitable *a posteriori* renormalisations. Second, over rings for which the coefficient size is an issue, it is often possible to use the specialization property of the subresultant in order to reduce the bulk of computations over finite fields, thanks to the multi-modular approach which was initiated by Collins [20] and Brown [14].

For non normal sequences, and when multi-modular techniques are not available, the coefficient growth becomes an issue for large gaps in the degree sequence (see our Example 18). In order to make the discussion precise on coefficient growth, we consider that $\mathbb{A}$ is a polynomial ring $\mathbb{K}[t]$, where $\mathbb{K}$ is an effective field, and that the degrees in t of F and G are $\leqslant d$. Then the degrees in t of the subresultant polynomials are $\leqslant (n_0+n_1)\,d$. The first part of the solution for formula (1) is due to Lazard: in a manu-

script remained unpublished, he showed that $s_{n_i-1,n_i}^e / s_{n_i}^{e-1}$ is in $\mathbb{A}$ for $1 \leqslant e \leqslant n_i - n_{i+1}$, and that its degree in t remains $\leqslant (n_0+n_1)\,d$ (see our Lemma 7). Ducos [21] found the second part of the solution, for the coefficient growth involved in (2): he designed the first algorithm with a quadratic number of operations in the ground ring, and with a degree in t not exceeding twice the degree bound in t of the subresultants, namely $2\,(n_0+n_1)\,d$. At present time, it is not known how to apply the "divide and conquer" paradigm to his algorithm.

In their 1996 article [36], Lickteig and Roy designed an alternative solution for the coefficient growth involved by degree gaps, by limiting this growth to a factor of 3. Their main improvement is an exact polynomial division scheme in $\mathbb{A}[x]$ in replacement of pseudo-divisions. They appealed to Lazard's optimization, and replaced formula (2) by

$$(-1)^{n_i-n_{i+1}-1}\, s_{n_i-1,n_{i+1}}\, s_{n_{i+1}}\, S_{n_i} = Q_{n_{i+1}}\, S_{n_i-1} + s_{n_i}^2\, S_{n_{i+1}-1}, \tag{3}$$

with $Q_{n_{i+1}} \in \mathbb{A}[x]$. This means that the division of $s_{n_i-1,n_{i+1}}\, s_{n_{i+1}}\, S_{n_i}$ by S_{n_i-1} is well defined in $\mathbb{A}[x]$, and this allows to obtain $S_{n_{i+1}-1}$ as $\mathrm{rem}((-1)^{n_i-n_{i+1}-1}\, s_{n_i-1,n_{i+1}}\, s_{n_{i+1}}\, S_{n_i}, S_{n_i-1}) / s_{n_i}^2$. We shall refer to formula (3) as the *Lickteig–Roy division*. Roughly speaking, the coefficients may grow by a factor of 3, but Lickteig and Roy showed how this division process may benefit from the half-gcd strategy. However they did not explain how to perform this division efficiently over an abstract ring $\mathbb{A}$. They contented themselves to rely on multi-modular techniques for the concrete ring $\mathbb{A} = \mathbb{Z}$ (see [37, p. 335]).

In his 2001 article [22], Ducos proposed an algorithm for performing the Lickteig–Roy division with a coefficient growth of only 2, but with a total quadratic cost. Lombardi, Roy, and Safey El Din [38], achieved the same growth and quadratic cost with an alternative algorithm.

When $\mathbb{A}$ is a multivariate polynomial ring $\mathbb{D}[t_1, ..., t_r]$, the most efficient techniques for subresultants rely on fast multi-point evaluation and interpolation algorithms (see [8, 29] for instance for recent advances). These algorithms require $\mathbb{D}$ to have sufficiently many elements, which is not very restrictive in practice. In fact, if necessary, we may often perform computations over an algebraic extension of $\mathbb{D}$. Nevertheless for an abstract domain $\mathbb{D}$, the complexity of subresultant computations is of theoretical interest. If d now represents a bound on the partial degrees in $t_1, ..., t_r$ of F and G, then the coefficient size of the k-th subresultant polynomial is $O((n_0+n_1-2\,k)^r\, d^r)$. A growth of a factor of α in the partial degrees implies a growth of a factor of α^r in the coefficient sizes. Consequently the algorithms presented so far behave well only when n_0 is sufficiently larger than d^r. Otherwise one may appeal to suitable linear algebra techniques as those designed in [1, 2, 4, 5, 6, 7, 11, 32]. We shall not investigate this situation in the present article.

For a modern use of Habicht's original ideas, the reader might consult Reischert's article [41], which also contains a "divide and conquer" variant of the subresultant algorithm based on formulas (1) and (2) (thus discarding the coefficient growth in defective cases). In an other more recent article in this vein, El Kahoui [23] proved the Lickteig–Roy division formula over any commutative ring. Finally, let us mention the 2003 article by von zur Gathen and Lücking [25], which contains a comprehensive history of the Euclidean algorithm, lower bounds for the coefficient swell, and also comparisons between performances of usual implementations with quadratic costs.

1.3. Our contributions

Our first contribution concerns the cost analysis of the Lickteig–Roy algorithm. In fact we propose to perform the Lickteig–Roy divisions by a "divide and conquer" algorithm over $\mathbb{A}$. We show that the quotient $Q_{n_{i+1}}$ in formula (3) may be obtained with $O(\mathsf{M}(n_i - n_{i+1}) \log(n_i - n_{i+1}))$ operations in $\mathbb{A}$. This is a logarithmic factor higher than by using

Newton's iteration when $\mathbb{A}$ is a field. At first sight, one might thus fear a total cost of $O(\mathsf{M}(n_0)\log^2 n_0)$ for the complete subresultant algorithm, but it is not so, because the total contribution of polynomial divisions is $O(\sum_i \mathsf{M}(n_i - n_{i+1})\log(n_i - n_{i+1})) = O(\mathsf{M}(n_0)\log n_0)$. Our Section 3.1 concerns fast divisions in $\mathbb{A}[x]$: the presented algorithm is certainly elementary, but we need to recall it for properly analyzing coefficient growths.

In Section 2 we prove the Lickteig–Roy formula from scratch. Our reasons for repeating this proof are twice. First, in their article [37], Lickteig and Roy consider the Habicht remainder sequence over an integral domain (this sequence coincides to the subresultant sequence up to signs). Second, they assume $n_1 = n_0 - 1$, and therefore we have to detail the two first division steps in the general case. We could not rely on [23] neither, mostly because of the coefficient growth in the second division step (see part 5 of Theorem 9). Algorithm 4 and Theorem 15 constitute our main results for any $\mathbb{A}$, whence our second contribution.

Our third contribution concerns refined complexity results when $\mathbb{A}$ is a univariate polynomial ring of the form $\mathbb{B}[t]$, where $\mathbb{B}$ is a commutative ring with unity endowed with its partially defined division routine. Complexity results are stated in Section 4.4. Then Section 4.5 contains corollaries for the deterministic complexity of bivariate gcds. Our new complexity bounds improve on previously known ones from the asymptotic point of view. Unfortunately, these new bounds do not turn out to be relevant to practice: for computations that last several minutes, Ducos' algorithm or evaluation/interpolation strategies are faster. Nevertheless, for testing purposes, we included an open source implementation of our main algorithm in MATHEMAGIX [28, 30].

2. SUBRESULTANTS

This section recalls formulas needed for the fast subresultant algorithm presented in the next section. Proofs are established from scratch, on the top of basic linear algebra statements. Recall that our input polynomials are $F = \sum_{i=0}^{n_0} f_i\, x^i$, with degree n_0, and $G = \sum_{i=0}^{n_1} g_i\, x^i$, with degree $n_1 \leqslant n_0$. The $\mathbb{A}$-module of polynomials of degrees $<n$ is written $\mathbb{A}[x]_{<n}$.

2.1. Definitions and main properties

For all $0 \leqslant k < n_1$, the k-th *Sylvester map* of F and G is defined as:

$$\begin{array}{rcl} \mathbb{A}[x]_{<n_1-k} \times \mathbb{A}[x]_{<n_0-k} & \to & \mathbb{A}[x]_{<n_0+n_1-k} \\ (U, V) & \mapsto & UF + VG. \end{array}$$

In the canonical monomial basis $1, x, x^2, \ldots$, its matrix is

$$\mathrm{Syl}_k = \begin{pmatrix} f_0 & & & g_0 & & & \\ & \ddots & & & \ddots & & \\ & & f_0 & \vdots & & \ddots & \\ \vdots & & & & & & \\ & & & g_{n_1} & & & g_0 \\ & & \vdots & & & & \\ f_{n_0} & & & & & \ddots & \vdots \\ & \ddots & & & & & \\ & & f_{n_0} & & & & g_{n_1} \end{pmatrix}. \tag{4}$$

The coefficients of F occupy the $n_1 - k$ first columns, and those of G the $n_0 - k$ last ones. The determinant of the submatrix of Syl_k obtained by discarding the k-th first rows, is called the k-th *subresultant coefficient* of F and G, and is written s_k. In other words, s_k

is the determinant of the map

$$\begin{array}{rcl} \mathbb{A}[x]_{<n_1-k} \times \mathbb{A}[x]_{<n_0-k} & \to & \mathbb{A}[x]_{<n_0+n_1-2k} \\ (U,V) & \mapsto & (UF+VG) \operatorname{quo} x^k. \end{array} \tag{5}$$

The determinant s_0 is the classical resultant of F and G, written $\operatorname{Res}(F,G)$. We define the k-th *subresultant polynomial*, written S_k, by the following determinant over $\mathbb{A}[x]$:

$$S_k = \begin{vmatrix} F & \cdots & x^{n_1-k-1}F & G & \cdots & x^{n_0-k-1}G \\ f_{k+1} & & f_{2k-n_1+2} & g_{k+1} & & g_{2k-n_0+2} \\ & & & \vdots & & \\ \vdots & & & g_{n_1} & & \\ & & \vdots & & & \vdots \\ f_{n_0} & & & & \ddots & \\ & \ddots & & & & \\ & & f_{n_0} & & & g_{n_1} \end{vmatrix}, \tag{6}$$

where the coefficients of F occupy the n_1-k first columns, and those of G the n_0-k last ones. The coefficient of degree l in S_k, written $s_{k,l}$, may be obtained as the following determinant over $\mathbb{A}$:

$$s_{k,l} = \begin{vmatrix} f_l & & f_{l+k-n_1+1} & g_l & & g_{l+k-n_0+1} \\ f_{k+1} & & f_{2k-n_1+2} & g_{k+1} & & g_{2k-n_0+2} \\ & & & \vdots & & \\ \vdots & & & g_{n_1} & & \\ & & \vdots & & & \vdots \\ f_{n_0} & & & & \ddots & \\ & \ddots & & & & \\ & & f_{n_0} & & & g_{n_1} \end{vmatrix}.$$

In this way, we see that S_k has degree $\leqslant k$, and that s_k is the coefficient $s_{k,k}$ of degree k in S_k. When s_k is zero, the subresultant polynomial S_k is said to be *defective*. Then we introduce the two polynomials U_k and V_k, called the *cofactors* of S_k:

$$U_k = \begin{vmatrix} 1 & \cdots & x^{n_1-k-1} & 0 & \cdots & 0 \\ f_{k+1} & & f_{2k-n_1+2} & g_{k+1} & & g_{2k-n_0+2} \\ & & & \vdots & & \\ \vdots & & & g_{n_1} & & \\ & & \vdots & & & \vdots \\ f_{n_0} & & & & \ddots & \\ & \ddots & & & & \\ & & f_{n_0} & & & g_{n_1} \end{vmatrix}, \tag{7}$$

$$V_k = \begin{vmatrix} 0 & \cdots & 0 & 1 & \cdots & x^{n_0-k-1} \\ f_{k+1} & & f_{2k-n_1+2} & g_{k+1} & & g_{2k-n_0+2} \\ & & & \vdots & & \\ \vdots & & & g_{n_1} & & \\ & & \vdots & & & \vdots \\ f_{n_0} & & & & \ddots & \\ & \ddots & & & & \\ & & f_{n_0} & & & g_{n_1} \end{vmatrix}. \tag{8}$$

The coefficient of degree l in U_k (resp. in V_k) is written $u_{k,l}$ (resp. $v_{k,l}$). From these definitions we straightforwardly obtain the *Bézout relation*

$$S_k = U_k F + V_k G,$$

which turns out to be unique whenever s_k is invertible in $\mathrm{Frac}(\mathbb{A})$, as detailed in the following proposition.

PROPOSITION 1. *If s_k is invertible in $\mathrm{Frac}(\mathbb{A})$, then S_k, U_k and V_k are the unique polynomials S, U, V in $\mathrm{Frac}(\mathbb{A})[x]$ satisfying the following conditions: $S = UF + VG$, $\deg S = k$, $\deg U < n_1 - k$, $\deg V < n_0 - k$, and $s_k = \mathrm{lc}(S)$.*

Proof. The uniqueness follows from the injectivity of the map defined in (5), for which U_k, V_k is the unique preimage of $(s_k, 0, ..., 0)$. □

We finally introduce the *transition matrix* $T_k = \begin{pmatrix} U_k & V_k \\ U_{k-1} & V_{k-1} \end{pmatrix}$, for $k < n_1$, so we have $\begin{pmatrix} S_k \\ S_{k-1} \end{pmatrix} = T_k \begin{pmatrix} F \\ G \end{pmatrix}$.

2.2. Highest subresultants and conventions

When $n_0 > n_1$, it makes sense to consider the case $k = n_1$, for which the Sylvester map is

$$\begin{aligned} \mathbb{A}[x]_{<0} \times \mathbb{A}[x]_{<n_0-n_1} &\to \mathbb{A}[x]_{<n_0} \\ (U, V) &\mapsto UF + VG, \end{aligned}$$

and to use the following conventions: $s_{n_1} = g_{n_1}^{n_0-n_1}$, $S_{n_1} = g_{n_1}^{n_0-n_1-1} G$, $U_{n_1} = 0$ and $V_{n_1} = g_{n_1}^{n_0-n_1-1}$. When $n_0 = n_1$, it is sometimes convenient to use the same values for s_{n_1}, S_{n_1}, U_{n_1} and V_{n_1} notwithstanding that S_{n_1} and V_{n_1} are not defined over $\mathbb{A}$ but over $\mathrm{Frac}(\mathbb{A})$, as long as g_{n_1} is not a zero divisor in $\mathbb{A}$.

When $n_1 \geqslant 1$ and $k = n_1 - 1$, the Sylvester map,

$$\begin{aligned} \mathbb{A}[x]_{<1} \times \mathbb{A}[x]_{<n_0-n_1+1} &\to \mathbb{A}[x]_{<n_0+1} \\ (U, V) &\mapsto UF + VG, \end{aligned}$$

has for matrix

$$\begin{pmatrix} f_0 & g_0 & & \\ & \ddots & & \\ & \vdots & & g_0 \\ \vdots & & & \\ & g_{n_1} & & \vdots \\ & & \ddots & \\ f_{n_0} & & & g_{n_1} \end{pmatrix}.$$

Elementary calculations then yield

$$S_{n_1-1} = \mathrm{prem}(F, G). \tag{9}$$

We introduce $Q_{n_1} = \mathrm{pquo}(F, G)$, so we have $n_2 = \deg S_{n_1-1} < n_1$. We shall define the other quotients Q_{n_i} latter on. Notice that we may also consider the first transition matrix T_{n_1}

$$T_{n_1} = \begin{pmatrix} 0 & s_{n_1} / g_{n_1} \\ g_{n_1}^{n_0-n_1+1} & -Q_{n_1} \end{pmatrix},$$

but it is not defined over $\mathbb{A}$ in general, although $\det T_{n_1} = -s_{n_1}^2$ actually belongs to $\mathbb{A}$.

2.3. Proportionality to the Euclidean sequence

If the Euclidean sequence is well defined over $\mathrm{Frac}(\mathbb{A})$, if R_i has degree $n_i \geqslant 0$ for some $i \geqslant 2$, and if s_{n_i} is invertible in $\mathrm{Frac}(\mathbb{A})$, then R_i is proportional to S_{n_i} over $\mathrm{Frac}(\mathbb{A})$, by Proposition 1. The following lemma is the key step to establish the converse: every subresultant polynomial is proportional to a remainder in the Euclidean sequence.

LEMMA 2. *If $n_1 \geqslant 0$ and if g_{n_1} is invertible in $\mathrm{Frac}(\mathbb{A})$, then R_2 is well defined over $\mathrm{Frac}(\mathbb{A})$, and the following properties hold:*

1. *When $n_1 \geqslant 1$, we have $S_{n_1-1} = \mathrm{prem}(F,G) = g_{n_1}^{n_0-n_1+1} R_2$;*
2. *When $n_2 < k < n_1 - 1$, we have $S_k = 0$;*
3. *When $n_2 \geqslant 0$, we have $S_{n_2} = \mathrm{lc}(R_2)^{n_1-n_2-1} g_{n_1}^{n_0-n_2} R_2$;*
4. *When $0 \leqslant k < n_2$, we have $S_k = (-1)^{(n_1-k)(n_2-k)} g_{n_1}^{n_0-n_2} \hat{S}_k$, where $\hat{S}_k$ is the k-th polynomial subresultant of G and R_2, over $\mathrm{Frac}(\mathbb{A})$.*

Proof. Let us write $r_{2,j}$ for the coefficient of degree j in R_2. When $n_2 \leqslant k < n_1$, interpreting the division of F by G in terms of column operations over $\mathrm{Frac}(\mathbb{A})$, we observe that

$$S_k = \begin{vmatrix}
R_2 & x R_2 & \cdots & x^{k-n_2} R_2 & x^{k-n_2+1} R_2 & \cdots & x^{n_1-k-1} R_2 & G & \cdots & x^{n_0-k-1} G \\
 & 0 & \cdots & 0 & r_{2,n_2} & & r_{2,2k-n_1+2} & g_{k+1} & & g_{2k-n_0+2} \\
 & & \ddots & \vdots & 0 & \ddots & \vdots & & & \\
 & & & 0 & \vdots & \ddots & r_{2,n_2} & & & \\
 & & & & 0 & & 0 & \vdots & & \\
 & & & & & \ddots & \vdots & & & \\
 & & & & & & 0 & & \vdots & \\
 & & & & & & & g_{n_1} & & \\
 & & & & & & & & \ddots & \\
 & & & & & & & & & g_{n_1}
\end{vmatrix},$$

For $k = n_1 - 1$, we recover the formula $S_{n_1-1} = g_{n_1}^{n_0-n_1+1} R_2$, already seen in equation (9). Then we obtain $S_k = 0$ when $n_2 < k < n_1 - 1$, and $S_{n_2} = r_{2,n_2}^{n_1-n_2-1} g_{n_1}^{n_0-n_2} R_2$. For $k < n_2$, the matrix Syl_k is column equivalent to

$$\begin{pmatrix}
r_{2,0} & & & & g_0 & & \leftarrow n_0 - n_2 \rightarrow \\
 & \ddots & & & & \ddots & \\
\vdots & & & & & g_0 & \\
 & & & & \vdots & & g_0 \\
r_{2,n_2} & & r_{2,0} & & & & \ddots \\
 & & & & & \vdots & g_0 \\
 & \ddots & \vdots & & g_{n_1} & & \vdots \\
 & & & & & \ddots & \\
 & & r_{2,n_2} & & & g_{n_1} & \vdots \\
 & & & & & & g_{n_1} \\
 & & & & & & \ddots \\
\longleftarrow n_1 - k \longrightarrow & & \longleftarrow n_2 - k \longrightarrow & & & & g_{n_1}
\end{pmatrix},$$

which implies that $S_k = (-1)^{(n_1-k)(n_2-k)} g_{n_1}^{n_0-n_2} \hat{S}_k$. □

COROLLARY 3. *If g_{n_1} and all the nonzero s_k are invertible in $\mathrm{Frac}(\mathbb{A})$, then the Euclidean remainder sequence is well defined over $\mathrm{Frac}(\mathbb{A})$, and the following properties hold:*

1. *S_{n_i-1} and $S_{n_{i+1}}$ are proportional to R_{i+1} over $\mathrm{Frac}(\mathbb{A})$, when $1 \leqslant i < w$;*

2. $S_k = 0$, *when* $n_{i+1} < k < n_i - 1$ *and* $1 \leqslant i < w$;

3. $S_k = 0$, *when* $k < n_w$.

Proof. The proof is done by induction on w. The case $w = 1$ means that $R_2 = 0$, and the previous lemma implies that all the subresultant polynomials are zero. Now assume $w \geqslant 2$. From the previous lemma, we know that: R_2 is well defined over $\mathrm{Frac}(\mathbb{A})$, R_2 is proportional to S_{n_1-1} and S_{n_2} over $\mathrm{Frac}(\mathbb{A})$, the leading coefficient of R_2 is invertible in $\mathrm{Frac}(\mathbb{A})$, $S_k = 0$ for all $n_2 < k < n_1 - 1$, and the other S_k are proportional to the subresultant polynomials of G and R_2. Since the length of the Euclidean sequence of G and R_2 is $w - 1$, we may use the corollary by induction. □

Corollary 4. *If* g_{n_1} *and all the nonzero* s_k *are invertible in* $\mathrm{Frac}(\mathbb{A})$, *then, for* $2 \leqslant i < w$ *the transition matrix* T_{n_i} *is the unique matrix* $T = \begin{pmatrix} T_{1,1} & T_{1,2} \\ T_{2,1} & T_{2,2} \end{pmatrix}$ *over* $\mathrm{Frac}(\mathbb{A})[x]$ *satisfying the following properties:* $\begin{pmatrix} S_{n_i} \\ S_{n_i-1} \end{pmatrix} = T \begin{pmatrix} F \\ G \end{pmatrix}$, $\deg T_{1,1} < n_1 - n_i$, $\deg T_{1,2} < n_0 - n_i$, $\deg T_{2,1} < n_1 - n_{i+1}$, $\deg T_{1,2} < n_0 - n_{i+1}$.

Proof. Proposition 1 already implies $T_{1,1} = U_{n_i}$ and $T_{1,2} = V_{n_i}$. By the previous corollary, S_{n_i-1} is proportional to $S_{n_{i+1}}$, and the conclusion follows again from Proposition 1 since $\deg U_{n_i-1} < n_1 - (n_i - 1) \leqslant n_1 - n_{i+1}$ and $\deg V_{n_i-1} < n_0 - (n_i - 1) \leqslant n_0 - n_{i+1}$. □

About the uniqueness of the cofactors U_{n_w-1} and V_{n_w-1} of $S_{n_w-1} = 0$, we begin with the following simple lemma:

Lemma 5. *For all* $1 \leqslant k \leqslant n_1$, *the following identities hold:*

$$u_{k-1,n_1-k} = (-1)^{n_1-k} g_{n_1} s_k, \quad v_{k-1,n_0-k} = (-1)^{n_1-k-1} f_{n_0} s_k.$$

Proof. These formulas are straightforward, from expanding determinants of equations (7) and (8) along their first rows. □

Corollary 6. *Assume* $w \geqslant 2$, *and that* f_{n_0}, g_{n_1} *and all the nonzero* s_k *are invertible in* $\mathrm{Frac}(\mathbb{A})$. *Then, the cofactors* U_{n_w-1} *and* V_{n_w-1} *are the unique polynomials* A *and* B *of* $\mathrm{Frac}(\mathbb{A})[x]$ *satisfying the following properties:* $A\,F + B\,G = 0$, $\deg A = n_1 - n_k$, $\deg B = n_0 - n_k$, *and* $\mathrm{lc}(A) = (-1)^{n_1-n_w} g_{n_1} s_{n_w}$.

Proof. Let a_j (resp. b_j) represent the coefficient of degree j in A (resp. in B). We consider the relation

$$(A - U_{n_w-1})\,F + (B - V_{n_w-1})\,G = 0.$$

By Lemma 5 we have $\deg(A - U_{n_w-1}) < n_1 - n_k$. On the other hand, using $a_{n_1-n_k} f_{n_0} + b_{n_0-n_k} g_{n_1} = 0$, we observe that

$$(-1)^{n_1-n_w-1} b_{n_0-n_w} V_{n_w-1} / (f_{n_0} s_{n_w}) = (-1)^{n_1-n_w} a_{n_1-n_w} V_{n_w-1} / (g_{n_1} s_{n_w}) = V_{n_w-1}.$$

Using Lemma 5 again, we obtain that $\deg(B - V_{n_w-1}) < n_0 - n_k$. Since the map

$$\begin{aligned} \mathbb{A}[x]_{<n_1-n_w} \times \mathbb{A}[x]_{<n_0-n_w} &\to \mathbb{A}[x]_{<n_0+n_1-2n_w} \\ (U, V) &\mapsto (UF + VG) \operatorname{quo} x^{n_w} \end{aligned}$$

is injective, we deduce that $A = U_{n_w-1}$ and $B = V_{n_w-1}$. □

2.4. Structure theorem

We are now ready to prove Lazard's lemma and the Lickteig–Roy division (equation (3)).

LEMMA 7. *Assume that f_{n_0}, g_{n_1}, and all the nonzero s_k are invertible in* $\mathrm{Frac}(\mathbb{A})$*, and let* $1 \leqslant i < w$*. Then, the following properties hold:*

1. *When $n_{i+1} \leqslant k < n_i$, the ratio $s_{n_i-1,k}^{n_i-k} / s_{n_i}^{n_i-k-1}$ equals a minor of size $n_0 + n_1 - 2\,k$ of* Syl_k*, hence it belongs to* $\mathbb{A}$ *(Lazard's lemma);*
2. $S_{n_{i+1}} = (s_{n_i-1,n_{i+1}}^{n_i-n_{i+1}-1}\, S_{n_i-1}) / s_{n_i}^{n_i-n_{i+1}-1}$.

Proof. From Corollary 3, we know that S_{n_i-1} has degree n_{i+1}. Thanks to Lemma 5, over $\mathrm{Frac}(\mathbb{A})$ we may write

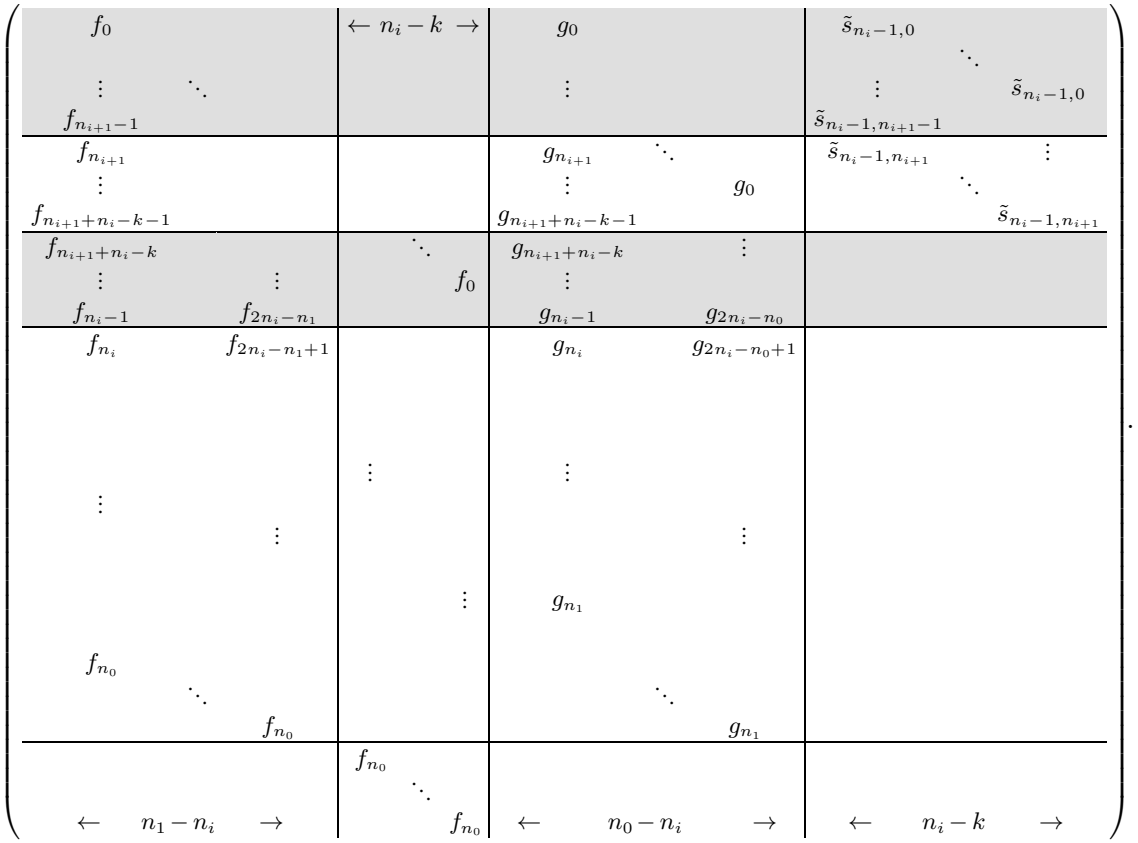
$$\tilde{S}_{n_i-1} := (-1)^{n_1-n_i-1}\,\frac{S_{n_i-1}}{f_{n_0}\, s_{n_i}} = \tilde{U}_{n_i-1}\,F + \tilde{V}_{n_i-1}\,G, \tag{10}$$

where $\tilde{U}_{n_i-1} = (-1)^{n_1-n_i-1}\, U_{n_i-1} / (f_{n_0}\, s_{n_i})$, and $\tilde{V}_{n_i-1} = (-1)^{n_1-n_i-1}\, V_{n_i-1} / (f_{n_0}\, s_{n_i})$, so $\tilde{V}_{n_i-1}$ is monic of degree $n_0 - n_i$. Assume $n_{i+1} \leqslant k \leqslant n_i$. The Bézout relation (10) rephrases in terms of the following column operations on the matrix (4) defining Syl_k: for l from $n_i - k - 1$ down to 0, the column $x^{n_0-n_i+l}\,G(x)$ may be replaced by $x^l\,\tilde{S}_{n_i-1}(x)$ by adding to it

$$\sum_{j=0}^{n_1-n_i} \tilde{u}_{n_i-1,j}\, x^{j+l}\, F(x) + \sum_{j=0}^{n_0-n_i-1} \tilde{v}_{n_i-1,j}\, x^{j+l}\, G(x),$$

where $\tilde{s}_{n_i-1,j}$, $\tilde{u}_{n_i-1,j}$, $\tilde{v}_{n_i-1,j}$ are the coefficients of degree j in $\tilde{S}_{n_i-1}$, $\tilde{U}_{n_i-1}$, $\tilde{V}_{n_i-1}$ respectively.

The Sylvester matrix Syl_k is thus column equivalent over $\mathrm{Frac}(\mathbb{A})$ to



When $n_{i+1} \leqslant k < n_i$, we discard the n_{i+1} first grayed rows and the other $k - n_{i+1}$ grayed rows corresponding to the monomials $x^{n_{i+1}+n_i-k}, ..., x^{n_i-1}$, and obtain the determinant

$$f_{n_0}^{n_i-k}\, s_{n_i}\, \tilde{s}_{n_i-1,n_{i+1}}^{n_i-k} = \pm f_{n_0}^{n_i-k}\, s_{n_i} \left(\frac{s_{n_i-1,n_{i+1}}}{f_{n_0}\, s_{n_i}} \right)^{n_i-k} = \pm \frac{s_{n_i-1,n_{i+1}}^{n_i-k}}{s_{n_i}^{n_i-k-1}},$$

which equals the minor of Syl_k obtained by deleting the same rows. This concludes part 1. When $k = n_{i+1}$ this minor coincides to $s_{n_{i+1}} = s_{n_i-1,n_{i+1}}^{n_i-n_{i+1}} / s_{n_i}^{n_i-n_{i+1}-1}$, whence part 2, since $S_{n_{i+1}}$ is proportional to S_{n_i-1} by Corollary 3. □

PROPOSITION 8. *For all $2 \leqslant i \leqslant w$, we have*

$$\begin{aligned} \det T_{n_i} &= \begin{vmatrix} U_{n_i} & V_{n_i} \\ U_{n_i-1} & V_{n_i-1} \end{vmatrix} = (-1)^{n_1-n_i-1}\, s_{n_i}^2, \\ (-1)^{n_1-n_i-1}\, s_{n_i}^2\, T_{n_i}^{-1} &= \begin{pmatrix} V_{n_i-1} & -V_{n_i} \\ -U_{n_i-1} & U_{n_i} \end{pmatrix}. \end{aligned}$$

Proof. By left multiplying booth sides of

$$\begin{pmatrix} S_{n_i} \\ S_{n_i-1} \end{pmatrix} = \begin{pmatrix} U_{n_i} & V_{n_i} \\ U_{n_i-1} & V_{n_i-1} \end{pmatrix} \begin{pmatrix} F \\ G \end{pmatrix}$$

by $\begin{pmatrix} V_{n_i-1} & -V_{n_i} \end{pmatrix}$, we obtain $V_{n_i-1}\, S_{n_i} - V_{n_i}\, S_{n_i-1} = F \det T_{n_i}$. By Proposition 1, we have $\deg(V_{n_i}\, S_{n_i-1}) \leqslant n_0 - 2$, and $\deg(V_{n_i-1}\, S_{n_i}) \leqslant n_0$. Consequently $\det T_{n_i}$ is in $\mathbb{A}$, and we have $v_{n_i-1,n_0-n_i}\, s_{n_i} = f_{n_0} \det T_{n_i}$. The conclusion follows from Lemma 5. □

THEOREM 9. *Let $\mathbb{A}$ be a commutative ring, and let F, G be polynomials of respective degrees $n_0 \geqslant n_1 \geqslant 0$. We assume that f_{n_0}, g_{n_1}, and all the nonzero subresultant coefficients are invertible in* $\mathrm{Frac}(\mathbb{A})$. *Then we have:*

1. *$S_{n_1-1} = \mathrm{prem}(F, G)$, $Q_{n_1} = \mathrm{pquo}(F, G)$, when $n_1 \geqslant 1$;*
2. *$S_k = 0$ when $n_{i+1} < k < n_i - 1$ and $1 \leqslant i \leqslant w$;*
3. *$s_{n_i-1,n_{i+1}}^{n_i-k-1} / s_{n_i}^{n_i-k-2}$ equals a minor of size $n_0 + n_1 - 2\,k$ of Syl_k, when $n_{i+1} \leqslant k < n_i - 1$;*
4. *$s_{n_i}\, S_{n_{i+1}} = \rho_{n_i}\, S_{n_i-1}$, where $\rho_{n_i} = s_{n_i-1,n_{i+1}}^{n_i-n_{i+1}-1} / s_{n_i}^{n_i-n_{i+1}-2} \in \mathbb{A}$, and when $n_{i+1} < n_i - 1$;*
5. *The division of $s_{n_1-1,n_2}\, s_{n_2}\, G$ by S_{n_1-1} is well defined over $\mathbb{A}$, and we have*

$$(-1)^{n_1-n_2-1}\, s_{n_1-1,n_2}\, s_{n_2}\, G = Q_{n_2}\, S_{n_1-1} + g_{n_1}\, s_{n_1}\, S_{n_2-1}, \quad \text{with} \quad Q_{n_2} \in \mathbb{A}[x];$$

6. *When $i \geqslant 2$, the division of $s_{n_i-1,n_{i+1}}\, s_{n_{i+1}}\, S_{n_i}$ by S_{n_i-1} is well defined over $\mathbb{A}$, and we have*

$$(-1)^{n_i-n_{i+1}-1}\, s_{n_i-1,n_{i+1}}\, s_{n_{i+1}}\, S_{n_i} = Q_{n_{i+1}}\, S_{n_i-1} + s_{n_i}^2\, S_{n_{i+1}-1}, \quad \text{with} \quad Q_{n_{i+1}} \in \mathbb{A}[x].$$

Proof. Parts 1 to 4 have already been proved. As for part 5, from Corollary 3 and part 2 of Lemma 7, there exist φ invertible in $\mathrm{Frac}(\mathbb{A})$, and $\Phi \in \mathrm{Frac}(\mathbb{A})[x]$ of degree $n_1 - n_2$, such that

$$\begin{pmatrix} S_{n_2} \\ S_{n_2-1} \end{pmatrix} = \begin{pmatrix} 0 & \rho_{n_1} / s_{n_1} \\ \varphi & (-1)^{n_1-n_2}\, \Phi \end{pmatrix} \begin{pmatrix} G \\ S_{n_1-1} \end{pmatrix},$$

which implies

$$\begin{aligned} \begin{pmatrix} S_{n_2} \\ S_{n_2-1} \end{pmatrix} &= \begin{pmatrix} 0 & \rho_{n_1} / s_{n_1} \\ \varphi & (-1)^{n_1-n_2}\, \Phi \end{pmatrix} \begin{pmatrix} 0 & 1 \\ g_{n_1}\, s_{n_1} & -Q_{n_1} \end{pmatrix} \begin{pmatrix} F \\ G \end{pmatrix} \\ &= \begin{pmatrix} g_{n_1}\, \rho_{n_1} & -\rho_{n_1}\, Q_{n_1} / s_{n_1} \\ (-1)^{n_1-n_2}\, g_{n_1}\, s_{n_1}\, \Phi & \varphi - (-1)^{n_1-n_2}\, Q_{n_1}\, \Phi \end{pmatrix} \begin{pmatrix} F \\ G \end{pmatrix} \end{aligned}$$

by using $S_{n_1-1} = \operatorname{prem}(F, G) = g_{n_1}\, s_{n_1}\, F - Q_{n_1}\, G$. Assume $w > 2$. Since $\deg(g_{n_1}\, \rho_{n_1}) = 0 < n_1 - n_2$, $\deg Q_{n_1} = n_0 - n_1 < n_0 - n_2$, $\deg \Phi < n_1 - (n_2 - 1)$, $\deg (Q_{n_1}\, \Phi) = n_0 - n_2 < n_0 - (n_2 - 1)$, we may apply Corollary 4 to obtain

$$T_{n_2} = \begin{pmatrix} g_{n_1}\, \rho_{n_1} & -\rho_{n_1}\, Q_{n_1} / s_{n_1} \\ (-1)^{n_1-n_2}\, g_{n_1}\, s_{n_1}\, \Phi & \varphi - (-1)^{n_1-n_2}\, Q_{n_1}\, \Phi \end{pmatrix}. \tag{11}$$

In particular, Proposition 8 gives $\det T_{n_2} = (-1)^{n_1-n_2-1}\, s_{n_2}^2 = \varphi\, g_{n_1}\, \rho_{n_1}$, which leads to $\varphi = (-1)^{n_1-n_2-1}\, s_{n_2}^2 / (g_{n_1}\, \rho_{n_1}) = (-1)^{n_1-n_2-1}\, s_{n_1-1,n_2}\, s_{n_2} / (g_{n_1}\, s_{n_1})$. On the other hand $g_{n_1}\, s_{n_1}\, \Phi$ belongs to $\mathbb{A}[x]$, which gives part 5, with $Q_{n_2} = (-1)^{n_1-n_2+1}\, g_{n_1}\, s_{n_1}\, \Phi$.

If $w = 2$, then $S_{n_2-1} = 0$, and we may divide φ and Φ by $\operatorname{lc}(\Phi)$ without loss of generality. Corollary 6 implies $(-1)^{n_1-n_2}\, g_{n_1}\, s_{n_1}\, \Phi = U_{n_2-1}$ and $\varphi - (-1)^{n_1-n_2}\, Q_{n_1}\, \Phi = V_{n_2-1}$, which again yields (11), and the conclusion follows as in the case $w > 2$.

Now for part 6, from Corollary 3 and part 2 of Lemma 7, again, there exist ψ invertible in $\operatorname{Frac}(\mathbb{A})$, and $\Psi \in \operatorname{Frac}(\mathbb{A})[x]$ of degree $n_i - n_{i+1}$, such that

$$\begin{pmatrix} S_{n_{i+1}} \\ S_{n_{i+1}-1} \end{pmatrix} = \begin{pmatrix} 0 & \rho_{n_i} / s_{n_i} \\ \psi & (-1)^{n_1-n_{i+1}}\, \Psi \end{pmatrix} \begin{pmatrix} S_{n_i} \\ S_{n_i-1} \end{pmatrix},$$

which implies

$$\begin{aligned} \begin{pmatrix} S_{n_{i+1}} \\ S_{n_{i+1}-1} \end{pmatrix} &= \begin{pmatrix} 0 & \rho_{n_i} / s_{n_i} \\ \psi & (-1)^{n_1-n_{i+1}}\, \Psi \end{pmatrix} T_{n_i} \begin{pmatrix} F \\ G \end{pmatrix} \\ &= \begin{pmatrix} \rho_{n_i}\, U_{n_i-1} / s_{n_i} & \rho_{n_i}\, V_{n_i-1} / s_{n_i} \\ \psi\, U_{n_i} + (-1)^{n_1-n_{i+1}}\, \Psi\, U_{n_i-1} & \psi\, V_{n_i} + (-1)^{n_1-n_{i+1}}\, \Psi\, V_{n_i-1} \end{pmatrix} \begin{pmatrix} F \\ G \end{pmatrix}. \end{aligned}$$

Assume $w > i + 2$. Since $\deg U_{n_i-1} < n_1 - (n_i - 1) \leqslant n_1 - n_{i+1}$, $\deg V_{n_i-1} < n_0 - (n_i - 1) \leqslant n_0 - n_{i+1}$,

$$\deg(\psi\, U_{n_i} + \Psi\, U_{n_i-1}) < \max(n_1 - n_i, n_1 - (n_{i+1} - 1)) \leqslant n_1 - (n_{i+1} - 1), \text{ and}$$

$$\deg(\psi\, V_{n_i} + \Psi\, V_{n_i-1}) < \max(n_0 - n_i, n_0 - (n_{i+1} - 1)) \leqslant n_0 - (n_{i+1} - 1),$$

we may apply Corollary 4 again to obtain

$$T_{n_{i+1}} = \begin{pmatrix} \rho_{n_i}\, U_{n_i-1} / s_{n_i} & \rho_{n_i}\, V_{n_i-1} / s_{n_i} \\ \psi\, U_{n_i} + (-1)^{n_1-n_{i+1}}\, \Psi\, U_{n_i-1} & \psi\, V_{n_i} + (-1)^{n_1-n_{i+1}}\, \Psi\, V_{n_i-1} \end{pmatrix}. \tag{12}$$

In particular Proposition 8 gives us

$$\det T_{n_{i+1}} = (-1)^{n_1-n_{i+1}-1}\, s_{n_{i+1}}^2 = -(\rho_{n_i}\, \psi / s_{n_i}) \det T_{n_i} = (-1)^{n_1-n_i}\, s_{n_i}^2\, s_{n_{i+1}}\, \psi / s_{n_i-1,n_{i+1}},$$

which leads to $\psi = (-1)^{n_i-n_{i+1}-1}\, s_{n_i-1,n_{i+1}}\, s_{n_{i+1}} / s_{n_i}^2$. From

$$\begin{pmatrix} \psi & \Psi \end{pmatrix} T_{n_i} = \begin{pmatrix} U_{n_{i+1}-1} & V_{n_{i+1}-1} \end{pmatrix}, \tag{13}$$

Proposition 8 implies that $s_{n_i}^2\, \Psi$ belongs to $\mathbb{A}[x]$. We thus obtain part 6 with $Q_{n_{i+1}} = -s_{n_i}^2\, \Psi$.

If $w = i + 2$, then $S_{n_{i+1}-1} = 0$, and Corollary 6 implies $\psi\, U_{n_i} + (-1)^{n_1-n_{i+1}}\, \Psi\, U_{n_i-1} = c\, U_{n_{i+1}-1}$ and $\psi\, V_{n_i} + (-1)^{n_1-n_{i+1}}\, \Psi\, V_{n_i-1} = c\, V_{n_{i+1}-1}$ for some element c invertible in $\operatorname{Frac}(\mathbb{A})$. Without loss of generality we may divide ψ and Ψ by c so that (12) still holds. The conclusion then follows as in the case $w > i + 2$. □

2.5. Atomic transition matrices

We introduce the following *atomic transition matrices* M_{n_i} over $\mathrm{Frac}(\mathbb{A})[x]$:

$$\begin{aligned} M_{n_1} &= \begin{pmatrix} 0 & s_{n_1}/g_{n_1} \\ g_{n_1}\,s_{n_1} & -Q_{n_1} \end{pmatrix}, \\ M_{n_{i+1}} &= \begin{pmatrix} 0 & \rho_{n_i}\,s_{n_i} \\ (-1)^{n_i-n_{i+1}-1}\,s_{n_i-1,n_{i+1}}\,s_{n_{i+1}} & -Q_{n_{i+1}} \end{pmatrix}/s_{n_i}^2, \quad \text{when } 1\leqslant i<w, \end{aligned}$$

so $\begin{pmatrix} S_{n_1} \\ S_{n_1-1} \end{pmatrix} = M_{n_1}\begin{pmatrix} F \\ G \end{pmatrix}$, and by Theorem 9 we have

$$\begin{pmatrix} S_{n_{i+1}} \\ S_{n_{i+1}-1} \end{pmatrix} = M_{n_{i+1}}\begin{pmatrix} S_{n_i} \\ S_{n_i-1} \end{pmatrix}, \quad \text{when } 1\leqslant i<w.$$

Notice that $s_{n_i}^2\,M_{n_{i+1}}$ has all its entries in $\mathbb{A}[x]$, while the entry s_{n_1}/g_{n_1} in M_{n_1} does not necessarily belong to $\mathbb{A}$. In addition, we have the following useful formulas:

$$\begin{aligned} \det M_{n_1} &= -s_{n_1}^2, \\ \det M_{n_{i+1}} &= (-1)^{n_i-n_{i+1}}\,s_{n_{i+1}}^2/s_{n_i}^2, \\ T_{n_i} &= M_{n_i}\cdots M_{n_1}, \\ T_{n_2} &= \begin{pmatrix} g_{n_1}\,\rho_{n_1} & -(\rho_{n_1}\,Q_{n_1})/s_{n_1} \\ -Q_{n_2} & ((-1)^{n_1-n_2+1}\,s_{n_1-1,n_2}\,s_{n_2}+Q_{n_1}\,Q_{n_2})/(g_{n_1}\,s_{n_1}) \end{pmatrix}. \end{aligned} \tag{14}$$

By construction, T_{n_2} has all its entries in $\mathbb{A}[x]$.

Lemma 10. *When $1\leqslant i<j\leqslant w$, the product*

$$s_{n_i}^2\,M_{n_j}\cdots M_{n_{i+1}} = T_{n_j}\,(s_{n_i}^2\,T_{n_i}^{-1}) = (-1)^{n_1-n_i-1}\begin{pmatrix} U_{n_j} & V_{n_j} \\ U_{n_j-1} & V_{n_j-1} \end{pmatrix}\begin{pmatrix} V_{n_i-1} & -V_{n_i} \\ -U_{n_i-1} & U_{n_i} \end{pmatrix}$$

has all its entries in $\mathbb{A}[x]$, the matrix of its degrees is $\begin{pmatrix} n_{i+1}-n_{j-1} & n_i-n_{j-1} \\ n_{i+1}-n_j & n_i-n_j \end{pmatrix}$, with the convention that a negative degree (in the top left entry) means the zero polynomial.

Proof. Since $T_{n_j} = M_{n_j}\cdots M_{n_1} = M_{n_j}\cdots M_{n_{i+1}}\,T_{n_i}$ has all its entries in $\mathbb{A}[x]$, so has $s_{n_i}^2\,M_{n_j}\cdots M_{n_{i+1}}$, by Proposition 8. When $j=i+1$ the degree matrix of $M_{n_{i+1}}$ is $\begin{pmatrix} <0 & 0 \\ 0 & n_i-n_{i+1} \end{pmatrix}$. Then the degree matrix of $M_{n_{i+2}}\,M_{n_{i+1}}$ is $\begin{pmatrix} 0 & n_i-n_{i+1} \\ n_{i+1}-n_{i+2} & n_i-n_{i+2} \end{pmatrix}$, and the one of $M_{n_{i+3}}\,M_{n_{i+2}}\,M_{n_{i+1}}$ is $\begin{pmatrix} n_{i+1}-n_{i+2} & n_i-n_{i+2} \\ n_{i+1}-n_{i+3} & n_i-n_{i+3} \end{pmatrix}$. The conclusion follows easily by induction. □

3. Half subresultant algorithm

The key idea of the half-gcd algorithm is the computation of the atomic transition matrices by a "divide and conquer" approach. In our framework over a ring endowed with its partial division routine, we first need to describe how well defined divisions in $\mathbb{A}[x]$ may be performed fast. Let $A=\sum_{i\geqslant 0}a_i\,x^i$ be a polynomial in $\mathbb{A}[x]$, and let $n\geqslant 0$, we shall use the upper and lower *truncations* written $\lceil A\rceil^n := \sum_{i=0}^{n-1}a_i\,x^i$ and $\lfloor A\rfloor_n := \sum_{i\geqslant 0}a_{i+n}\,x^i$.

3.1. Fast polynomial division

Let A and B be two polynomials in $\mathbb{A}[x]$ such that the division of A by B is well defined. In the same vein as the Barrett or Sieveking–Kung division algorithms, we first compute quotients and then remainders. Quotients are obtained from jet computations. Precisely, if Φ and Γ are two jets in $\mathbb{A}[\![x]\!]/(x^n)$, the division Φ/Γ is said to be well defined whenever $\Gamma \neq 0$ and there exists Ψ in $\mathbb{A}[\![x]\!]/(x^n)$ such that $\Phi = \Gamma\,\Psi$. If $a \geqslant 0$ is a real number, then $\lfloor a \rfloor$ represents the largest integer $\leqslant a$, and $\lceil a \rceil$ is the smallest integer $\geqslant a$.

Algorithm 1

Input. $\Phi = \sum_{i=0}^{n-1} \varphi_i\, x^i$ and $\Gamma = \sum_{i=0}^{n-1} \gamma_i\, x^i$ in $\mathbb{A}[\![x]\!]/(x^n)$.

Output. Φ/Γ.

Assumption: the division of Φ by Γ is well-defined, and $\gamma_0 \neq 0$.

1. If $n = 1$ then return φ_0/γ_0.
2. Let $h = \lceil n/2 \rceil$.
3. Call recursively the algorithm with $\lceil \Phi \rceil^h$ and $\lceil \Gamma \rceil^h$ to obtain $\Psi_l = \lceil \Phi \rceil^h / \lceil \Gamma \rceil^h$ in $\mathbb{A}[\![x]\!]/(x^h)$.
4. Call recursively the algorithm with $\lfloor \Phi \rfloor_h - \lfloor \Psi_l \lceil \Gamma \rceil^h \rfloor_h - \Psi_l \lfloor \Gamma \rfloor_h$ and $\lceil \Gamma \rceil^h$ in $\mathbb{A}[\![x]\!]/(x^{n-h})$, and write Ψ_h the jet in return.
5. Return $\Psi_l + \Psi_h\, x^h$, seen in $\mathbb{A}[\![x]\!]/(x^n)$.

Proposition 11. *Algorithm 2 is correct and takes $O(\mathsf{M}(n)\log n)$ operations in $\mathbb{A}$.*

Proof. Let Ψ represent the quotient Φ/Γ in $\mathbb{A}[\![x]\!]/(x^n)$. First, it is clear that the division in step 3 is well-defined and that $\Psi_l = \lceil \Psi \rceil^h$. Then, using $2h \geqslant n$, we obtain

$$\Phi = \lceil \Psi \rceil^h \lceil \Gamma \rceil^h + (\lceil \Psi \rceil^h \lfloor \Gamma \rfloor_h + \lceil \Gamma \rceil^h \lfloor \Psi \rfloor_h)\, x^h + O(x^n)$$

which yields the correctness of step 4:

$$\Psi_h = \frac{\lfloor \Phi \rfloor_h - \lfloor \lceil \Psi \rceil^h \lceil \Gamma \rceil^h \rfloor_h - \lceil \Psi \rceil^h \lfloor \Gamma \rfloor_h}{\lceil \Gamma \rceil^h} + O(x^{n-h}) = \lfloor \Psi \rfloor_h + O(x^{n-h}).$$

The correctness of the algorithm follows by strong induction on n. Its cost function $\mathsf{C}(n)$ satisfies $\mathsf{C}(n) = \mathsf{C}(h) + \mathsf{C}(n-h) + O(\mathsf{M}(n))$, which classically leads to $\mathsf{C}(n) = O(\mathsf{M}(n)\log n)$. □

For a polynomial $A \in \mathbb{A}[x]$ of degree $\leqslant n$, we write $\mathrm{rev}(A, n)$ for the *reverse polynomial* $x^n\, A(1/x)$. The latter algorithm classically allows to compute polynomial quotients.

Algorithm 2

Input. $A = \sum_{i=0}^{n} a_i\, x^i$ and $B = \sum_{i=0}^{m} b_i\, x^i$ in $\mathbb{A}[x]$ of respective degrees n and $m \geqslant 0$.

Output. $\mathrm{quo}(A, B)$.

Assumption: the division of A by B is well defined.

1. Let $l = n - m$, $\tilde{A} = \mathrm{rev}(A, n)$ and $\tilde{B} = \mathrm{rev}(B, m)$.
2. Compute $\tilde{C} = \tilde{A}/\tilde{B} + O(x^{l+1})$ with Algorithm 1.
3. Return $\mathrm{rev}(\tilde{C}, l)$, where $\tilde{C}$ is seen as a polynomial of degree l.

Proposition 12. *Algorithm 2 is correct and takes $O(\mathsf{M}(n-m)\log(n-m))$ operations in $\mathbb{A}$.*

Proof. Let $R = \mathrm{rem}(A, B)$, $Q = \mathrm{quo}(A, B)$, $\tilde{Q} = \mathrm{rev}(Q, l)$, and $\tilde{R} = \mathrm{rev}(R, n)$. From $A = Q\,B + R$ we deduce $\tilde{A} = \tilde{Q}\,\tilde{B} + \tilde{R}$, and then $\tilde{Q} = \tilde{A} / \tilde{B} + O(x^{l+1})$. This proves the correctness. The complexity simply follows from the latter proposition. □

Once the quotient Q of A by B is computed, then $R = A - Q\,B$ may be obtained with $\mathsf{M}(n)$ operations in $\mathbb{A}$.

3.2. Main divide and conquer routine

Theorem 9 gives formulas to compute S_{n_1-1}, Q_{n_1}, S_{n_2}, Q_{n_2}, and S_{n_2-1} by means of the above division algorithm. Let us assume these polynomials already computed, and let us examine how to obtain the other quotients $Q_{n_{i+1}}$ for $i \geqslant 2$. The key observation is that Q_{n_3} only depends on the $n_2 - n_3 + 1$ highest coefficients of S_{n_2} and S_{n_2-1}, so it can be obtained from $\lfloor S_{n_2} \rfloor_{n_3 - \deg Q_{n_3}}$ and $\lfloor S_{n_2-1} \rfloor_{n_3 - \deg Q_{n_3}}$. Then Q_{n_4} only depends on the $n_3 - n_4 + 1$ highest coefficients of S_{n_3} and S_{n_3-1}. But the coefficient of degree j in S_{n_3-1} only depends on the coefficients of S_{n_2} and S_{n_2-1} of degrees $\geqslant j - \deg Q_{n_3}$. Consequently Q_{n_3} and Q_{n_4} may be obtained from $\lfloor S_{n_2} \rfloor_{n_4 - (\deg Q_{n_3} + \deg Q_{n_4})}$ and $\lfloor S_{n_2-1} \rfloor_{n_4 - (\deg Q_{n_3} + \deg Q_{n_4})}$. By induction, we thus prove that Q_{n_3}, ..., Q_{n_i} may be obtained from $\lfloor S_{n_2} \rfloor_{n_i - (\deg Q_{n_3} + \cdots + \deg Q_{n_i})}$ and $\lfloor S_{n_2-1} \rfloor_{n_i - (\deg Q_{n_3} + \cdots + \deg Q_{n_i})}$. By using $n_i = n_2 - (n_2 - n_3) - \cdots - (n_{i-1} - n_i) = n_2 - (\deg Q_{n_3} + \cdots + \deg Q_{n_i})$, the low truncation order $n_i - (\deg Q_{n_3} + \cdots + \deg Q_{n_i})$ rewrites into $n_2 - 2\,(\deg Q_{n_3} + \cdots + \deg Q_{n_i})$.

If we fix an integer $l \geqslant 0$, in order to compute Q_{n_3}, ..., Q_{n_j} with j maximal such that $\deg Q_{n_3} + \cdots + \deg Q_{n_j} \leqslant l$, it suffices to low truncate S_{n_2} and S_{n_2-1} at order $n_2 - 2\,l$. In order to handle negative truncation order, it could be convenient to use Laurent polynomials from a theoretical point of view, but for practice we prefer to keep computations in $\mathbb{A}[x]$. In fact the construction we have just sketched simply works fine by setting $\lfloor A \rfloor_n = A$ for all $n < 0$, thanks to the following lemma:

Lemma 13. *Let $A \neq 0$, $B \in \mathbb{A}[x]$, and let $l \in \mathbb{Z}$. If $l < 0$ then we have $\lfloor A\,B \rfloor_{l + \deg A} = \lfloor A\,\lfloor B \rfloor_l \rfloor_{l + \deg A}$, otherwise we have $\lfloor A\,B \rfloor_{l + \deg A} = \lfloor A\,\lfloor B \rfloor_l \rfloor_{\deg A}$.*

Proof. If $l < 0$ then $\lfloor B \rfloor_l = B$ and the lemma is correct. Now assume $l \geqslant 0$. We need to prove that the coefficient $\sum_{p+q=i, q \geqslant 0} a_p\,b_{q+l}$ of degree i in $A\,\lfloor B \rfloor_l$ equals the coefficient of degree $i + l$ in $A\,B$ for all $i \geqslant \deg A$. This assertion is correct because $p \leqslant \deg A$ implies $q = i - p \geqslant 0$. □

Let $i \geqslant 2$ and $0 \leqslant l \leqslant n_i$. If we are given $\lfloor S_{n_i} \rfloor_\nu$ and $\lfloor S_{n_i-1} \rfloor_\nu$, where $\nu = n_i - 2\,l$, then the same reasoning shows that we may compute $Q_{n_{i+1}}$, ..., Q_{n_j} with j maximal such that $n_i - n_j \leqslant l$. In fact, we shall better compute the subresultants s_{n_i}, ..., $s_{n_{j-1}}$ and the numerators $N_{n_{i+1}}$, ..., N_{n_j} of the atomic transition matrices made from $Q_{n_{i+1}}$, ..., Q_{n_j} respectively, namely $N_{n_{i+1}} = s_{n_i}^2\,M_{n_{i+1}}$. At the same time we shall return the composite transition matrix $M_{n_{i+1}, n_j} = M_{n_j} \cdots M_{n_{i+1}}$, or more precisely its "numerator" $N_{n_{i+1}, n_j} = s_{n_i}^2\,M_{n_{i+1}, n_j}$, which has coefficients in $\mathbb{A}[x]$ according to Lemma 10. We may then recover S_{n_j} and S_{n_j-1} as

$$\begin{pmatrix} S_{n_j} \\ S_{n_j-1} \end{pmatrix} = \left(N_{n_{i+1}, n_j} \begin{pmatrix} S_{n_i} \\ S_{n_i-1} \end{pmatrix} \right) / s_{n_i}^2.$$

Upon this strategy we are now able to present the adaptation of the half-gcd algorithm to subresultants. But before, notice that given l and $\lfloor A \rfloor_l$ one may deduce $\deg A$ whenever $\lfloor A \rfloor_l \neq 0$: this is simply $\deg \lfloor A \rfloor_l$ if $l < 0$, or $l + \deg \lfloor A \rfloor_l$ otherwise.

Algorithm 3

Input. $n_0, ..., n_i$, an integer $l \in \{0, ..., n_i\}$, and $\lfloor S_{n_i} \rfloor_\nu$ and $\lfloor S_{n_i-1} \rfloor_\nu$, where $\nu = n_i - 2\,l$.

Output. $n_{i+1}, ..., n_j$, $s_{n_{i+1}}, ..., s_{n_j}$, $N_{n_{i+1}}, ..., N_{n_j}$, and N_{n_{i+1},n_j} with $j \geqslant i$ maximal such that $n_i - n_j \leqslant l$.

Assumptions: $i \geqslant 2$, f_{n_0}, g_{n_1}, $s_{n_1}, ..., s_{n_w}$ are nonzero divisors in $\mathbb{A}$.

1. If $\lfloor S_{n_i-1} \rfloor_\nu = 0$ or $l < n_i - n_{i+1}$ then return nothing. *Notice that n_{i+1} is determined from $\lfloor S_{n_i-1} \rfloor_\nu$ and ν when $\lfloor S_{n_i-1} \rfloor_\nu \neq 0$.*
2. Let $h = \lfloor l/2 \rfloor$ and call the algorithm recursively with $n_0, ..., n_i$, h, and $\lfloor S_{n_i} \rfloor_{n_i-2h}$, $\lfloor S_{n_i-1} \rfloor_{n_i-2h}$. Let $n_{i+1}, ..., n_k$, $s_{n_{i+1}}, ..., s_{n_k}$, $N_{n_{i+1}}, ..., N_{n_k}$, and N_{n_{i+1},n_k} be the data obtained in return.
3. Let $\mu = \nu + n_i - n_k$, and compute $\begin{pmatrix} \lfloor S_{n_k} \rfloor_\mu \\ \lfloor S_{n_k-1} \rfloor_\mu \end{pmatrix}$ from $N_{n_{i+1},n_k} \begin{pmatrix} \lfloor S_{n_i} \rfloor_\nu \\ \lfloor S_{n_i-1} \rfloor_\nu \end{pmatrix} / s_{n_i}^2$.
4. If $\lfloor S_{n_k-1} \rfloor_\mu = 0$ or $l < n_i - n_{k+1}$ then return $n_{i+1}, ..., n_k$, $s_{n_{i+1}}, ..., s_{n_k}$, $N_{n_{i+1}}, ..., N_{n_k}$, and N_{n_{i+1},n_k}. *Notice that n_{k+1} is determined from $\lfloor S_{n_k-1} \rfloor_\mu$ and μ when $\lfloor S_{n_k-1} \rfloor_\mu \neq 0$.*
5. If $n_{k+1} < n_k - 1$ then compute $\rho_{n_k} = s_{n_k-1,n_{k+1}}^{n_k-n_{k+1}-1} / s_{n_k}^{n_k-n_{k+1}-2}$, and $\lfloor S_{n_{k+1}} \rfloor_\mu = (\rho_{n_k} \lfloor S_{n_k-1} \rfloor_\mu)/s_{n_k}$, otherwise set $\rho_{n_k} = s_k$.
6. Compute $Q_{n_{k+1}}$ from $(-1)^{n_k-n_{k+1}-1}\ s_{n_k-1,n_{k+1}}\ s_{n_{k+1}}\ \lfloor S_{n_k} \rfloor_\mu$ and $\lfloor S_{n_k-1} \rfloor_\mu$, by means of Algorithm 2.
7. Build $N_{n_k+1} = \begin{pmatrix} 0 & \rho_{n_k} s_{n_k} \\ (-1)^{n_k-n_{k+1}-1} s_{n_k-1,n_{k+1}} s_{n_{k+1}} & -Q_{n_{k+1}} \end{pmatrix}$.
8. Let $\lambda = \mu + n_k - n_{k+1}$, and deduce $\lfloor S_{n_{k+1}-1} \rfloor_\lambda = \left\lfloor (-1)^{n_k-n_{k+1}-1} s_{n_k-1,n_{k+1}} s_{n_{k+1}} S_{n_k} - Q_{n_{k+1}} S_{n_k-1} \right\rfloor_\lambda / s_{n_k}^2$ by means of Lemma 13.
9. Call recursively the algorithm with n_0, ..., n_{k+1}, $l - (n_i - n_{k+1})$, $\lfloor S_{n_{k+1}} \rfloor_\lambda$ and $\lfloor S_{n_{k+1}-1} \rfloor_\lambda$. Let $n_{k+1}, ..., n_j$, $s_{n_{k+2}}, ..., s_{n_j}$, $N_{n_{k+2}}, ..., N_{n_j}$, and N_{n_{k+2},n_j} be the data obtained in return.
10. Return n_{i+1}, ..., n_j, $s_{n_{i+1}}$, ..., s_{n_j}, $N_{n_{i+1}}$, ..., N_{n_j} and $(N_{n_{k+2},n_j}\ ((N_{n_k+1}\ N_{n_{i+1},n_k})/s_{n_k}^2))/s_{n_{k+1}}^2$.

PROPOSITION 14. *Algorithm 3 is correct and takes $O(\mathsf{M}(l) \log l)$ operations in $\mathbb{A}$.*

Proof. The proof is done by strong induction on l. If $l = 0$ then $\nu = n_i$, $\lfloor S_{n_i-1} \rfloor_\nu = 0$, and the output is correct. Now assume $l \geqslant 1$ and that the algorithm is correct up to $l - 1$.

In step 1 the case $\lfloor S_{n_i-1} \rfloor_\nu = 0$ means that $\nu > n_{i+1}$, whence $n_i - n_{i+1} > 2\,l \geqslant l$, and the output is correct. Otherwise we obtain n_{i+1}, and the output is again correct whenever $l < n_i - n_{i+1}$.

In step 2 we necessarily have $0 \leqslant h < l$, which implies that the recursive call is valid and returns a correct result by induction. By Lemma 10 the degrees of the entries of N_{n_{i+1},n_k} are $\leqslant n_i - n_k$. Lemma 13 ensures that we may safely obtain $\lfloor S_{n_k} \rfloor_\mu$ and $\lfloor S_{n_k-1} \rfloor_\mu$ in step 3.

If $\lfloor S_{n_k-1} \rfloor_\mu = 0$ in step 4, then this means that $\mu > n_{k+1}$, whence $n_i - n_{k+1} > n_i - \mu = n_i - \nu - (n_i - n_k) = 2\,l - (n_i - n_k) \geqslant l$ and the output is correct. If $\lfloor S_{n_k-1} \rfloor_\mu \neq 0$ then we may determine n_{k+1} so the output is also correct when $l < n_i - n_{k+1}$.

The computation in step 5 follows from part 4 of Theorem 9. In step 6, we have $n_i - n_{k+1} \leqslant l$, and therefore, from $\nu = n_i - 2\,l$, we obtain $\mu = \nu + n_i - n_k = 2\,n_i - n_k - 2\,l \leqslant n_{k+1} - (n_k - n_{k+1})$ so we may safely obtain $Q_{n_{k+1}}$ by part 6 of Theorem 9.

In step 9, we verify the requested conditions to the recursive call: $\lambda = n_{k+1} - 2\,(l - (n_i - n_{k+1}))$, and $l - (n_i - n_{k+1}) < l$.

In step 10, $s_{n_i}^2\, M_{n_{i+1},n_k+1} = s_{n_i}^2\,(N_{n_k+1} / s_{n_k}^2)\,(N_{n_{i+1},n_k} / s_{n_i}^2) = (N_{n_k+1}\, N_{n_{i+1},n_k}) / s_{n_k}^2$ has its entries in $\mathbb{A}[x]$ by Lemma 10. Repeating this argument we obtain $s_{n_i}^2\, M_{n_{i+1},n_j} = (N_{n_{k+2},n_j}\,((N_{n_k+1}\, N_{n_{i+1},n_k}) / s_{n_k}^2)) / s_{n_{k+1}}^2$. We are done with the correctness.

Let $\mathsf{C}(l)$ represent the cost of the algorithm called with l. Steps 1, 4, 5 and 7 involve $O(l)$ operations in $\mathbb{A}$. Step 2 costs $\mathsf{C}(h)$. Steps 3, 8 and 10 perform $O(\mathsf{M}(l))$ operations in $\mathbb{A}$. In step 6, we appeal to Proposition 12 to get a cost $O(\mathsf{M}(n_k - n_{k+1}) \log(n_k - n_{k+1}))$. Step 9 amounts to $\mathsf{C}(l - (n_i - n_{k+1}))$: notice that $n_i - n_{k+1} \geqslant h+1$ implies $l - (n_i - n_{k+1}) \leqslant l - (h+1) < l/2$. Overall there exists a constant $c > 0$ such that

$$\mathsf{C}(l) \leqslant \mathsf{C}(h) + \mathsf{C}(l - (n_i - n_{k+1})) + c\,(\mathsf{M}(l) + \mathsf{M}(n_k - n_{k+1}) \log(n_k - n_{k+1})).$$

Since h and $l - (n_i - n_{k+1})$ are $\leqslant l/2$, it is classical to deduce

$$\mathsf{C}(l) = O\left(\mathsf{M}(l) \log l + \sum_{k=i}^{j-1} \mathsf{M}(n_k - n_{k+1}) \log(n_k - n_{k+1}) \right) = O(\mathsf{M}(l) \log l). \qquad \square$$

3.3. Top level algorithm

We are now ready to present the main algorithm for computing all the atomic transition matrices.

Algorithm 4

Input. F and G in $\mathbb{A}[x]$ of respective degrees $n_0 \geqslant n_1$.

Output. $n_0, ..., n_w$, $s_{n_1}, ..., s_{n_w}$, $N_{n_3}, ..., N_{n_w}$, S_{n_1-1}, Q_{n_1}, ρ_{n_1}, Q_{n_2}.

Assumption: f_{n_0}, g_{n_1}, $s_{n_1}, ..., s_{n_w}$ are nonzero divisors in $\mathbb{A}$.

1. If $F = 0$ then $w = -1$ and return nothing — none of the quantities are defined.
2. If $G = 0$ then $w = 0$ and return n_0 — the other quantities are not defined.
3. Compute $S_{n_1-1} = \mathrm{prem}(F, Q)$, $Q_{n_1} = \mathrm{pquo}(F, G)$, and $s_{n_1} = g_{n_1}^{n_0 - n_1}$.
4. If $S_{n_1-1} = 0$ then $w = 1$ and return n_0, n_1, s_{n_1}, S_{n_1-1}, Q_{n_1} — the other quantities are not defined.
5. Let $n_2 = \deg S_{n_1-1}$. If $n_2 < n_1 - 1$ then compute $S_{n_2} = (\rho_{n_1}\, S_{n_i-1}) / s_{n_1}$, where $\rho_{n_1} = s_{n_1-1,n_2}^{n_1-n_2-1} / s_{n_1}^{n_1-n_2-2}$, otherwise let $\rho_{n_1} = s_{n_1}$.
6. Perform the division $(-1)^{n_1-n_2-1}\, s_{n_1-1,n_2}\, s_{n_2}\, G = Q_{n_2}\, S_{n_1-1} + g_{n_1}\, s_{n_1}\, S_{n_2-1}$, in order to obtain Q_{n_2} and S_{n_2-1}.
7. If $S_{n_2-1} = 0$ then $w = 2$ and return n_0, n_1, n_2, $s_{n_1}, s_{n_2} = \mathrm{lc}(S_{n_2})$, S_{n_1-1}, Q_{n_1}, ρ_{n_1}, Q_{n_2}— the other quantities are not defined.
8. Call Algorithm 3 with n_0, n_1, n_2, $l = n_2$, $\lfloor S_{n_2} \rfloor_\nu$ and $\lfloor S_{n_2-1} \rfloor_\nu$, where $\nu = -n_2$. Let $n_3, ..., n_j$, $s_{n_3}, ..., s_{n_j}$, $N_{n_3}, ..., N_{n_j}$, and N_{n_3,n_j} represent the data obtained in return.
9. Return $n_0, ..., n_j$, $s_{n_1}, ..., s_{n_j}$, $N_{n_3}, ..., N_{n_j}$, S_{n_1-1}, Q_{n_1}, ρ_{n_1}, Q_{n_2}.

Theorem 15. *Algorithm 4 is correct and takes $O(\mathsf{M}(n_0) \log n_0)$ operations in $\mathbb{A}$.*

Proof. The correctness follows from Theorem 9 and Proposition 14 after noticing that j necessarily coincides to w in step 8. The divisions in steps 2 and 5 take $O(\mathsf{M}(n_0) \log n_0)$ by Proposition 12. The cost of step 8 is given in Proposition 14. $\square$

Example 16. Let us briefly illustrate the algorithm with $\mathbb{A}=\mathbb{Z}$, $n_0=5$, $n_1=4$,

$$F=-3\,x^5-5\,x^4+3\,x^3-2\,x^2+4\,x+2, \qquad G=-5\,x^4+4\,x^3-2\,x^2+3\,x-1.$$

We obtain $S_{n_1-1} = -43\,x^3 -21\,x^2 +4\,x+87$, $n_2 = 3$, and $S_{n_2} = S_{n_1-1}$. Then we have $S_{n_2-1}=-415\,x^2-482\,x+890$, and we enter Algorithm 3 with S_{n_2}, S_{n_2-1} and $l=3$. In a recursive call with $\lfloor S_{n_2}\rfloor_1$, $\lfloor S_{n_2-1}\rfloor_1$ and $l=1$ we obtain Q_{n_3}, which has degree 1. Then we deduce $S_{n_3-1}=-11348\,x+13885$, and then $Q_{n_3}=4709420\,x+11232011$. Finally we have $n_4=1$, $Q_{n_4}=2724234924\,x-3333274755$, and $S_{n_4-1}=-240063$.

4. Bivariate case

In this section we study the complexity of Algorithm 4 when $\mathbb{A}$ is a polynomial ring $\mathbb{B}[t]$, where $\mathbb{B}$ is a commutative ring with unity and endowed with its partially defined division routine. First, we illustrate the coefficient growth issue. Then we detail the computations of the ρ_{n_i} and analyze the cost of the divisions in $\mathbb{A}[x]$. We are interested in deterministic algorithms that do not rely on fast evaluation/interpolation schemes, which would require specific assumptions on $\mathbb{B}$.

We recall that the product of two polynomials in $\mathbb{B}[t][x]$ of partial degrees $\leqslant d$ in t and $\leqslant n$ in x may be achieved with $O(\mathsf{M}(d\,n))$ operations in $\mathbb{B}$ by means of the classical Kronecker substitution [24, Section 8.4].

4.1. Coefficient growth

The sizes of the coefficients of the subresultant polynomials may be bounded from their defining determinant.

Lemma 17. *Assume $\mathbb{A}=\mathbb{B}[t]$. If $\deg_t F\leqslant d_0$ and $\deg_t G\leqslant d_1$ then we have $\deg_t s_{n_1}\leqslant (n_0-n_1)\,d_1$, and, for $0\leqslant k<n_1$, the partial degrees of S_k, U_k and V_k are $\leqslant (n_1-k)\,d_0+(n_0-k)\,d_1$.*

Proof. This follows from the definitions in equations (6), (7), and (8). □

Example 18. Let $\mathbb{A}=\mathbb{Z}[t]$. We consider the following family of polynomials parametrized by the integer $m\geqslant 2$:

$$F \;=\; t\,x^{3m}-t\,x^{2m}+x^m-1, \qquad G=t\,x^{3m}+t\,x^{2m}+x^m.$$

We have $n_0=n_1=3\,m$. The convention for S_{n_1} and formula (9) lead to:

$$\begin{aligned} S_{n_1} \;&=\; t^{-1}\,G \;=\; x^{3m}+x^{2m}+t^{-1}\,x^m \;\in\; t^{-1}\,\mathbb{Z}[t],\\ S_{n_1-1} \;&=\; \operatorname{prem}(F,G) \;=\; t\,F-t\,G \;=\; -2\,t^2\,x^{2m}-t. \end{aligned}$$

We obtain $n_2=2\,m$, and formulas (1) and (2) give:

$$\begin{aligned} S_{n_2} \;&=\; (\rho_{n_1}\,S_{n_1-1})/s_{n_1} \;=\; (-2\,t^2)^{m-1}\,S_{n_1-1} \;=\; (-2\,t^2)^m\,x^{2m}-(-2)^{m-1}\,t^{2m-1},\\ S_{n_2-1} \;&=\; \operatorname{prem}(S_{n_1},-S_{n_1-1})/s_{n_1}^{m+1} \;=\; (-2)^m\,t^{2m+1}\,x^m-t\,(-2\,t^2)^m. \end{aligned}$$

Then we get $n_3=m$, and

$$\begin{aligned} S_{n_3} \;&=\; (s_{n_2-1,n_3}^{m-1}\,S_{n_2-1})/\,s_{n_2}^{m-1} \;=\; ((-2\,t^2)^m\,t\,/\,(-2\,t^2)^m)^{m-1}\,S_{n_2} \;=\; (-2)^m\,t^{3m}\,(x^m-1),\\ S_{n_3-1} \;&=\; \operatorname{prem}(S_{n_2},-S_{n_2-1})/(-2\,t^2)^{m(m+1)} \;=\; (-2)^m\,t^{3m+1}-(-2)^{m-1}\,t^{3m}. \end{aligned}$$

Finally $n_4 = 0$, and

$$S_{n_4} \;=\; -(1+2\,t)^{m-1}\,(2\,t+1)\,t^{3m}.$$

This example shows that the intermediate sizes grow linearly with the gap sizes, when computing subresultants by means of formulas (1) and (2). Of course, for normal sequences of subresultants, these formulas simplify to $S_{n_{i+1}} = \mathrm{prem}(S_{n_i}, S_{n_i-1}) / s_{n_i}^2 = \mathrm{rem}(s_{n_{i-1}}^2 S_{n_i}, S_{n_i-1})$, and intermediate coefficient sizes just increase by a factor of three.

In the rest of this section, we show that the combination of Lazard's lemma and of the Lickteig–Roy division allows Algorithm 4 to preserve a coefficient growth bounded by a constant factor only.

4.2. Computation of the ρ_{n_i}

Here we analyze the computation of $S_{n_{i+1}}$ from S_{n_i-1} when $n_{i+1} < n_i - 1$. Example 18 shows that ρ_{n_i} should not be computed as the division of $s_{n_i-1,n_{i+1}}^{n_i-n_{i+1}-1}$ by $s_{n_i}^{n_i-n_{i+1}-2}$. Instead, we appeal to the "divide and conquer" algorithm based on Lazard's lemma (part 3 of Theorem 9), as described by Ducos in [21, p. 338], and which is reminiscent of the classical *binary powering algorithm*.

Algorithm 5

Input. s_{n_i} and $s_{n_i-1,n_{i+1}}$ in $\mathbb{A}$, for $1 \leqslant i \leqslant w$, and an integer $1 \leqslant l \leqslant n_i - n_{i+1} - 1$.

Output. $s_{n_i-1,n_{i+1}}^{l} / s_{n_i}^{l-1} \in \mathbb{A}$.

Assumption: $n_{i+1} < n_i - 1$.

1. If $l = 1$ then return $s_{n_i-1,n_{i+1}}$.
2. Let $h = \lfloor l/2 \rfloor$.
3. Recursively compute $a = s_{n_i-1,n_{i+1}}^{h} / s_{n_i}^{h-1}$, and then $b = a^2 / s_{n_i}$.
4. If l is even then return b else return $(s_{n_i-1,n_{i+1}}\, b) / s_{n_i}$.

Lemma 19. *Algorithm 5 is correct and takes $O(\log(n_i - n_{i+1}))$ operations in $\mathbb{A}$. If $\mathbb{A} = \mathbb{B}[t]$, $\deg_t F \leqslant d_0$, and $\deg_t G \leqslant d_1$, then the algorithm costs $O(\mathsf{M}(D)\log(D)\log(n_i - n_{i+1}))$ operations in $\mathbb{B}$, where $D = n_1\,d_0 + n_0\,d_1$. If $\mathbb{B}$ is a field then the latter cost simplifies to $O(\mathsf{M}(D)\log(n_i - n_{i+1}))$.*

Proof. The correctness is a consequence of part 3 of Theorem 9. When $\mathbb{A} = \mathbb{B}[t]$, we use the degree bound provided by Lemma 17, and part 3 of Theorem 9 also gives that $\deg_t a = O(D)$ holds during the execution. Then the cost analysis is rather standard: each product takes $O(\mathsf{M}(D))$, and each division costs $O(\mathsf{M}(D)\log D)$ by Proposition 12 (the factor $\log D$ may be discarded when $\mathbb{B}$ is a field). □

4.3. Polynomial divisions

We need now to revisit the costs of the division algorithms of Section 3.1 when $\mathbb{A} = \mathbb{B}[t]$.

Proposition 20. *Assume $\mathbb{A} = \mathbb{B}[t]$. Let $\Phi, \Gamma \in \mathbb{A}[\![x]\!]/(x^n)$ be such that the division of Φ by Γ is well defined, and let d bound the degrees in t of Φ, Γ, and Ψ. Then, Algorithm 1 takes $O(\mathsf{M}(d\,n)\log n + n\,\mathsf{M}(d)\log d)$ operations in $\mathbb{B}$. If $\mathbb{B}$ is a field then the latter cost simplifies to $O(\mathsf{M}(d\,n)\log n)$.*

Proof. We use the notation of Algorithm 1, and write $\mathsf{C}(n)$ for its cost. A straightforward induction shows that the degrees in t of the values of Γ and of the output Ψ are always at most d throughout the recursive calls. This implies that the degrees in t of the values of Φ always remain bounded by $2\,d$ throughout the recursive calls. Consequently, there exists a constant $c>0$ such that

$$\mathsf{C}(n) \leqslant \mathsf{C}(h) + \mathsf{C}(n-h) + c\,\mathsf{M}(d\,n).$$

In general we have $\mathsf{C}(1) = O(\mathsf{M}(d)\,\log\,d)$ by Proposition 12, which simplifies to $\mathsf{C}(1) = O(\mathsf{M}(d))$ when $\mathbb{B}$ is a field. The conclusion classically follows by unrolling the latter inequality. □

Proposition 21. *Assume $\mathbb{A}=\mathbb{B}[t]$. Let $A, B \in \mathbb{A}[x]$ be of respective degrees $n \geqslant m \geqslant 0$, and such that the division of A by B is well defined, and let d bound the degrees in t of A, B, and of their quotient Q. Then, Q may be obtained with $O(\mathsf{M}(d\,(n-m))\log(n-m) + (n-m)\ \mathsf{M}(d)\ \log\ d)$ operations in $\mathbb{B}$. If $\mathbb{B}$ is a field then the latter cost simplifies to $O(\mathsf{M}(d\,(n-m))\log(n-m))$. Then the remainder may be deduced from Q with $O(\mathsf{M}(d\,n))$ additional operations in $\mathbb{B}$.*

Proof. The quotient Q may be obtained with the claimed complexity thanks to the latter proposition by using Algorithm 2. Then the remainder is obtained as $A - Q\,B$ with $O(\mathsf{M}(d\,n))$ additional operations in $\mathbb{B}$. □

Lemma 22. *Assume $\mathbb{A}=\mathbb{B}[t]$. Let $A, B \in \mathbb{A}[x]$ be of respective degrees $n \geqslant m \geqslant 0$. Then the degree in t of* $\mathrm{pquo}(A,B)$ *is $\leqslant \deg_t A + (n-m)\deg_t B$.*

Proof. We follow the naive pseudo-division algorithm of A by B:

1. Set $R=A$ and $Q=0$;
2. For l from n down to m replace Q by $b_m\,Q + r_l\,x^{l-m}$ and R by $b_m\,R - r_l\,x^{l-m}\,B$.

At the end, Q and R are respectively the pseudo-quotient and pseudo-remainder. At each step in the loop the degrees in t of R and Q increase by at most $\deg_t B$. □

4.4. Main complexity bounds

In order to apply Proposition 21, we need to bound the degrees in t of all the quotients Q_{n_i} arising in the subresultant algorithm. Some care is necessary since $\mathbb{A}$ is not assumed to be integral. Our sole assumption is that f_{n_0}, g_{n_1}, $s_{n_1}, \ldots, s_{n_w}$ are nonzero divisors in $\mathbb{A}$.

Lemma 23. *Assume $\mathbb{A}=\mathbb{B}[t]$, $\deg_t F \leqslant d_0$, $\deg_t G \leqslant d_1$, and let $D = n_1\,d_0 + n_0\,d_1$. Then, $\deg Q_{n_1} \leqslant d_0 + n_0\,d_1$, and for all $2 \leqslant i \leqslant w$ the degree in t of Q_{n_i} is $O(D)$.*

Proof. The bound for Q_{n_1} follows from Lemma 22. By equation (14) the degree bound for Q_{n_2} follows from the one for U_{n_2-1} given in Lemma 17. For $i \geqslant 3$, we combine equation (13), Proposition 8, and Lemma 17. □

Lemma 24. *Assume $\mathbb{A}=\mathbb{B}[t]$, $\deg_t F \leqslant d_0$, $\deg_t G \leqslant d_1$, and let $D = n_1\,d_0 + n_0\,d_1$. Then, for all $1 \leqslant i < j \leqslant w$ the degrees in t of the entries of $s_{n_i}^2\,M_{n_j}\cdots M_{n_{i+1}}$ are $O(D)$.*

Proof. This is a consequence of Lemmas 10 and 17. □

Theorem 25. *Assume $\mathbb{A}=\mathbb{B}[t]$, $\deg_t F \leqslant d_0$, $\deg_t G \leqslant d_1$, and let $D = n_1\,d_0 + n_0\,d_1$. Then, Algorithm 4 takes*

$$\begin{aligned} &O(\mathsf{M}((d_0 + d_1\,n_0)\,n_0)\log n_0 + \underline{n_0\,\mathsf{M}(d_0 + d_1\,n_0)\log(d_0 + d_1\,n_0)}) \\ &\quad + O(\mathsf{M}(D\,n_1)\log n_1 + \underline{n_1\,\mathsf{M}(D)\log D\log n_1}). \end{aligned}$$

operations in $\mathbb{B}$. *In addition, the underlined expressions may be discarded when* $\mathbb{B}$ *is a field.*

Proof. We first analyse the cost $\mathsf{C}(l)$ of Algorithm 3 in terms of operations in $\mathbb{B}$:

- steps 1 and 4 involve $O(D\,l)$ operations in $\mathbb{B}$,
- step 2 takes $\mathsf{C}(h)$,
- step 3 takes $O(\mathsf{M}(D\,l) + \underline{l\,\mathsf{M}(D)\log D})$,
- step 5 takes $O(\mathsf{M}(D)\,\underline{\log D}\log l + l\,\mathsf{M}(D)\,\underline{\log D})$ by using Lemma 19,
- step 6 takes $O(\mathsf{M}(D\ (n_k - n_{k+1}))\ \log(n_k - n_{k+1}) + \underline{(n_k - n_{k+1})\ \mathsf{M}(D)\ \log\ D})$ by combining Lemma 23 to Proposition 21,
- step 7, 8 and 10 take $O(\mathsf{M}(D\,l) + \underline{l\,\mathsf{M}(D)\log D})$ by using Lemma 24,
- step 9 takes $\mathsf{C}(l - (n_i - n_{k+1}))$.

Consequently, there exists a constant $c > 0$ such that

$$\begin{aligned}\mathsf{C}(l) \ \leqslant\ & \mathsf{C}(h) + \mathsf{C}(l - (n_i - n_{k+1})) + c\,(\mathsf{M}(D\,l) + \underline{l\,\mathsf{M}(D)\log D} \\ & + \mathsf{M}(D\,(n_k - n_{k+1}))\log(n_k - n_{k+1}) + \underline{(n_k - n_{k+1})\,\mathsf{M}(D)\log D})).\end{aligned}$$

Since h and $l - (n_i - n_{k+1})$ are $< l/2$, it is classical to deduce

$$\mathsf{C}(l) = O(\mathsf{M}(D\,l)\log l + \underline{l\,\mathsf{M}(D)\log D\log l}).$$

Then we analyze the cost of Algorithm 4 in terms of operations in $\mathbb{B}$:

- steps 1, 2 take $O(d_0\,n_0 + d_1\,n_1)$,
- step 3 takes $O(\mathsf{M}((d_0 + d_1\,n_0)\,n_0)\log n_0 + \underline{n_0\,\mathsf{M}(d_0 + d_1\,n_0)\log(d_0 + d_1\,n_0)})$ by combining Lemma 22 to Proposition 21,
- step 4 just performs $O((d_0 + d_1\,n_0)\,n_0)$ operations in $\mathbb{B}$,
- step 5 takes $O(\mathsf{M}(D)\,\underline{\log D}\log n_1 + n_1\,\mathsf{M}(D)\,\underline{\log D}$ by using Lemma 19,
- step 6 takes $O(\mathsf{M}(D\,n_1)\log n_1 + \underline{n_1\,\mathsf{M}(D)\log D})$ by combining Lemma 23 to Proposition 21,
- steps 7 takes $O(D\,n_1)$,
- step 8 takes $\mathsf{C}(n_2)$.

The total cost of Algorithm 4 is therefore

$$\begin{aligned}O(\mathsf{M}((d_0 + d_1\,n_0)\,n_0)\log n_0 + \underline{n_0\,\mathsf{M}(d_0 + d_1\,n_0)\log(d_0 + d_1\,n_0)}) \\ + O(\mathsf{M}(D\,n_1)\log n_1 + \underline{n_1\,\mathsf{M}(D)\log D\log n_1}).\end{aligned}$$

operations in $\mathbb{B}$. □

Corollary 26. *Assume* $\mathbb{A} = \mathbb{B}[t]$. *Let* $F, G \in \mathbb{A}[x]$ *be of respective degrees* $n_0 \geqslant n_1$, *let* $d_0 = \deg_t F$, $d_1 = \deg_t G$, *and* $D = d_1\,n_0 + d_0\,n_1$. *If* f_{n_0}, g_{n_1}, $s_{n_1}, \ldots, s_{n_w}$ *are nonzero divisors in* $\mathbb{A}$, *then any one subresultant polynomial of* F *and* G *with its associated cofactors may be obtained with*

$$\begin{aligned}O(\mathsf{M}(D\,n_0) + \mathsf{M}((d_0 + d_1\,n_0)\,n_0)\log n_0 + \underline{n_0\,\mathsf{M}(D)\log(D)}) \\ + O(\mathsf{M}(D\,n_1)\log n_1 + \underline{n_1\,\mathsf{M}(D)\log D\log n_1})\end{aligned}$$

operations in $\mathbb{B}$. *The sole computation of one subresultant without its associated cofactors takes*

$$\begin{aligned}O(\mathsf{M}((d_0 + d_1\,n_0)\,n_0)\log n_0 + \underline{n_0\,\mathsf{M}(d_0 + d_1\,n_0)\log(d_0 + d_1\,n_0)}) \\ + O(\mathsf{M}(D\,n_1)\log n_1 + \underline{n_1\,\mathsf{M}(D)\log D\log n_1}).\end{aligned}$$

In addition, the underlined expressions may be discarded when $\mathbb{B}$ *is a field.*

Proof. After calling Algorithm 4, we have the atomic transition matrices at our disposal, and we may easily build T_{n_2} via formula (14). Now assume we want to obtain S_{n_i}, S_{n_i-1} and T_{n_i} for $2 \leqslant i \leqslant w$. For this purpose we compute T_{n_i} first, and then use

$$\begin{pmatrix} S_{n_i} \\ S_{n_i-1} \end{pmatrix} = T_{n_i} \begin{pmatrix} F \\ G \end{pmatrix}.$$

Since $T_{n_i} = M_{n_i} \cdots M_{n_3} T_{n_2}$, we shall compute $s_{n_2}^2 M_{n_i} \cdots M_{n_3}$ by the following divide and conquer algorithm that computes $s_{n_i}^2 M_{n_j} \cdots M_{n_{i+1}}$ for $i < j$ as follows:

1. If $j = i+1$ then return $s_{n_i}^2 M_{n_{i+1}}$ (available as $N_{n_{i+1}}$ from the output of Algorithm 4);
2. Let $h = \lfloor (i+j)/2 \rfloor$, and recursively compute $A = s_{n_i}^2 M_{n_h} \cdots M_{n_{i+1}}$ and $B = s_{n_h}^2 M_{n_j} \cdots M_{n_{h+1}}$;
3. Return $(B\,A)/s_{n_h}^2$.

The correctness is ensured by Lemma 10, which also provides us with the degree bound $n_i - n_j$ in x for $s_{n_i}^2 M_{n_j} \cdots M_{n_{i+1}}$. In addition the degree in t of the latter matrix is $O(D)$ by Lemma 24. Therefore the computation of $s_{n_2}^2 M_{n_i} \cdots M_{n_3}$ costs $O(\mathsf{M}(D\,n_2) \log n_2 + \underline{n_2\,\mathsf{M}(D) \log(D) \log n_2})$. Finally, the computation of T_{n_2} just requires $O(\mathsf{M}(D\,n_0) + \underline{n_0\,\mathsf{M}(D) \log(D)})$ operations in $\mathbb{B}$. If the cofactors are not needed then we may truncate T_{n_2} to precision $O(x^{n_2+1})$, which yields a cost $O(\mathsf{M}(D\,n_2) + \underline{n_2\,\mathsf{M}(D) \log(D)})$. □

If $\mathbb{B}$ is a field that contains sufficiently many elements (for instance when the characteristic is zero), then we may use fast evaluation and interpolation algorithms [24, Chapter 10]. To simplify the situation we reformulate the costs in terms of $d = \max(d_0, d_1)$. In this way, a subresultant of degree k may be interpolated from $D+1$ values of t with $O(k\,\mathsf{M}(d\,n_0) \log(d\,n_0))$. All the specializations of F and G may be obtained with

$$O(n_0^2\,\mathsf{M}(d) \log d),$$

and each specialized subresultant requires $O(\mathsf{M}(n_0) \log n_0)$. Thus the total cost is

$$O(k\,\mathsf{M}(d\,n_0) \log(d\,n_0) + n_0^2\,\mathsf{M}(d) \log d + d\,n_0\,\mathsf{M}(n_0) \log n_0). \tag{15}$$

If $k \simeq n_0$, then the dominant term is $k\,\mathsf{M}(d\,n_0) \log(d\,n_0)$, which is asymptotically higher than the cost $O(\mathsf{M}(d\,n_0^2) \log n_0)$ of Corollary 26 when $\mathsf{M}(n) = n \log n \log\log n$ because

$$\frac{\mathsf{M}(d\,n_0^2) \log n_0}{n_0\,\mathsf{M}(d\,n_0) \log(d\,n_0)} = \frac{d\,n_0^2 \log(d\,n_0) \log\log(d\,n_0) \log n_0}{d\,n_0^2 \log^2(d\,n_0) \log\log(d\,n_0)} = O\left(\frac{\log n_0}{\log(d\,n_0)}\right).$$

For any k, the ratio of the cost of Corollary 26 over the one of (15) when $\mathsf{M}(n) = n \log n \log\log n$ becomes bounded by

$$\frac{\mathsf{M}(d\,n_0^2) \log n_0}{n_0^2\,\mathsf{M}(d) \log d + d\,n_0\,\mathsf{M}(n_0) \log n_0} = \frac{\log(d\,n_0) \log\log(d\,n_0) \log n_0}{\log^2 d \log\log d + \log^2 n_0 \log\log n_0}.$$

If $n_0 \leqslant d$ this ratio is $O\left(\frac{\log n_0}{\log d}\right) = O(1)$. Otherwise, if $d \leqslant n_0$, then the ratio is again $O(1)$. Consequently, in all case the cost of Corollary 26 is never asymptotically higher (up to a constant factor) than with the evaluation/interpolation strategy.

4.5. Applications

Gcd is a very classical application of subresultants. Our Corollary 26 leads to the following deterministic result.

Corollary 27. *Let $\mathbb{K}$ be a field, and let A and B be two polynomials in $\mathbb{K}[t,x]$ of degrees $\leqslant d$ in t and $\leqslant n$ in x. Then, the gcd of A and B may be computed with $O(\mathsf{M}(d\,n^2)\log n + n\,\mathsf{M}(d\,n)\log d)$ operations in $\mathbb{K}$, which simplifies to $O(\mathsf{M}((d\,n)^{3/2})\log d)$ whenever $n \leqslant d$.*

Proof. First we compute the contents and primitive parts of A and B seen in $\mathbb{K}[t][x]$ with $O(n\ \mathsf{M}(d)\ \log\ d)$ operations in $\mathbb{K}$. Then we deduce the gcd of the contents with $O(\mathsf{M}(d)\log d)$ operations in $\mathbb{K}$. By Corollary 26, we may compute the last nonzero subresultant of the primitive parts with $O(\mathsf{M}(d\,n^2)\log n)$ operations in $\mathbb{K}$. It has degree $\leqslant n$ in x and $O(d\,n)$ in t, hence its primitive part may be obtained with $O(n\,\mathsf{M}(d\,n)\log(d\,n))$ more operations. □

Let us mention that complexities in terms of convex hulls of the supports of A and B may further be derived thanks to the algorithm in [10]. Previously such deterministic costs for bivariate gcds were involving hypotheses on the cardinality of $\mathbb{K}$ in order to use fast evaluation/interpolation strategies, as in [24, Chapter 10].

The next corollary concerns the cost of the bivariate multi-gcd problem, which is for instance useful for computing separable decompositions [34].

Corollary 28. *Let $\mathbb{K}$ be a field, and let A and $B_1, ..., B_s$ be polynomials in $\mathbb{K}[t,x]$. Let $n = \deg_x A$, $d = \deg_t A$, $n_i = \deg_x B_i$, $d_i = \deg_t B_i$, and assume $n_1 + \cdots + n_s = O(n)$, $s = O(n)$, and $d_1 + \cdots + d_s = O(d)$. Then, the gcds of A and B_1, A and B_2,..., A and B_s may be computed with $O(\mathsf{M}(d\,n^2)\log n + n\,\mathsf{M}(d\,n)\log d)$ operations in $\mathbb{K}$.*

Proof. First we compute the content a and the primitive part $\hat{A}$ of A seen in $\mathbb{K}[t][x]$ with $O(n\,\mathsf{M}(d)\log d)$ operations in $\mathbb{K}$. Now let $1 \leqslant i \leqslant s$. We compute the content b_i and the primitive part $\widehat{B_i}$ of B_i with $O(n_i\,\mathsf{M}(d_i)\log d_i)$ operations in $\mathbb{K}$. Then we deduce the gcd of a and b_i with $O(\mathsf{M}(d)\log d)$ operations in $\mathbb{K}$. By Corollary 26, we may compute the last nonzero subresultant of $\hat{A}$ and $\widehat{B_i}$ with $O(\mathsf{M}((d + d_i\,n)\,n)\log n + \mathsf{M}(D_i\,n_i)\log n_i)$ operations in $\mathbb{K}$, where $D_i = d_i\,n + d\,n_i = O(d\,n)$. It has degree $\leqslant n_i$ in x and $O(D_i)$ in t, hence its primitive part may be obtained with $O(n_i\,\mathsf{M}(D_i)\log(D_i)) = O(n_i\,\mathsf{M}(d\,n)\log(d\,n))$. The total cost for computing all the gcds is $O(n\ \mathsf{M}(d)\ \log\ d + \sum_{i=1}^{s}[n_i\ \mathsf{M}(d_i)\ \log\ d_i + \mathsf{M}((d\ +\ d_i\ n)\ n)\ \log\ n\ +\ \mathsf{M}(d\ n\ n_i)\ \log\ n_i + n_i\ \mathsf{M}(d\ n)\ \log(d\ n)]) = O(\mathsf{M}(d\ n^2)\ \log\ n\ +\ n\,\mathsf{M}(d\,n)\log d)$. □

An other application of Corollary 26 is the computation of characteristic polynomials in a $\mathbb{A}$-algebra of the form $\mathbb{A}[x]/(A(x))$.

Corollary 29. *Let $\mathbb{A}$ be an integral domain endowed with its partially defined division, and let A and B be two polynomials in $\mathbb{A}[x]$ of degrees $\leqslant n$. Then, the resultant $\chi(t) = \mathrm{Res}_x(A(x), t - B(x))$ may be computed with $O(\mathsf{M}(n^2)\log n + n\,\mathsf{M}(n)\log^2 n)$ operations in $\mathbb{A}$. If $\mathbb{A}$ is a field then the cost simplifies to $O(\mathsf{M}(n^2)\log n)$.*

Proof. It suffices to use Corollary 26 with $d_0 = 0$ and $d_1 = 1$. □

If $\mathbb{A}$ is a field, written $\mathbb{K}$, which contains sufficiently many elements, then χ may be interpolated from $n\ +\ 1$ values of t. For each value a of t the specialized resultant $\mathrm{Res}_x(A(x),\ a\ -\ B(x))$ takes $O(\mathsf{M}(n)\ \log\ n)$ operations in $\mathbb{K}$. This leads to a total cost $O(n\ \mathsf{M}(n)\ \log\ n)$, which is smaller than the general bound of the latter corollary. However, if $\mathbb{K} = \mathbb{F}_2$, we need to perform this evaluation/interpolation procedure over $\mathbb{L} = \mathbb{F}_{2^\kappa}$ with κ being the first integer such that $2^\kappa \geqslant n+1$: the resultant χ may be thus obtained with $O(n\,\mathsf{M}(n)\,\mathsf{M}(\log n)\log n)$ operations in $\mathbb{F}_2$. With $\mathsf{M}(n) = n\log n\log\log n$, this cost rewrites

$$O(n^2\log^3 n\,(\log\log n)^2\log\log\log n),$$

which is higher than the one of the latter corollary. Notice that there also exist algorithms with subquadratic costs based on *power projections* and the *Newton–Girard* formula [26, Section 2], but they also require hypotheses on $\mathbb{K}$ (such as characteristic zero, or to be a finite field, etc).

Implementation.

The algorithms presented in this article have been implemented in the C++ library called Algebramix of Mathemagix [30] from 2009. The source code is available from our SVN server `http://gforge.inria.fr/projects/mmx/`. For the design of the C++ libraries see [31, Section 4].

When $\mathbb{A}$ is a field, the simplest algorithm with quadratic cost is adapted from the naive version of the Euclidean algorithm: it is available from `polynomial_naive.hpp`. The half-gcd is implemented in `polynomial_dicho.hpp`.

Over rings, the file `polynomial_ring_naive.hpp` contains the following implementations of subresultants: over any commutative ring we appeal to Berkowitz' algorithm [9] to compute the defining determinants without division; when a partial division routine is available in the ground ring, we use Ducos' algorithm [22].

Algorithm 4 can be found in `polynomial_ring_dicho.hpp`. The latter file includes the fast polynomial division routines. These C++ implementations are rather intricate because of several optimizations for various contexts, taking into account for instance which subresultants or cofactors are actually requested. For the sake of convenience, in revision 10523, we added a simple implementation of Algorithm 4 in the Mathemagix language [28]: see `gregorix/mmx/subresultant_lickteig_roy.mmx`, and `gregorix/mmx/subresultant_test.mmx` for the tests.

Concerning efficiencies, unfortunately our implementations of Algorithm 4 did not reveal to be competitive to Ducos' algorithm even in large sizes for bivariate polynomials. In the special case of Corollary 29, the evaluation/interpolation strategy is in general observed to be faster than the direct use of Algorithm 4.

Bibliography

[1] J. Abdeljaoued, G. M. Diaz-Toca, and L. Gonzalez-Vega. Minors of Bezout matrices, subresultants and the parameterization of the degree of the polynomial greatest common divisor. *Int. J. Comput. Math.*, 81(10):1223–1238, 2004.

[2] J. Abdeljaoued, G. M. Diaz-Toca, and L. Gonzalez-Vega. Bezout matrices, subresultant polynomials and parameters. *Appl. Math. Comput.*, 214(2):588–594, 2009.

[3] A. V. Aho, J. E. Hopcroft, and Ullman J. D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., 1974.

[4] A. G. Akritas. A new method for computing polynomial greatest common divisors and polynomial remainder sequences. *Numer. Math.*, 52(2):119–127, 1988.

[5] A. G. Akritas. Sylvester's forgotten form of the resultant. *Fibonacci Quart.*, 31(4):325–332, 1993.

[6] A. G. Akritas, E. K. Akritas, and G. I. Malaschonok. Matrix computation of subresultant polynomial remainder sequences in integral domains. *Reliab. Comput.*, 1(4):375–381, 1995.

[7] A. G. Akritas and G. I. Malaschonok. Fast matrix computation of subresultant polynomial remainder sequences. In V. G. Ganzha, E. W. Mayr, and E. V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing (CASC 2000)*, pages 1–11. Springer-Verlag, 2000.

[8] A. Arnold, M. Giesbrecht, and D. S. Roche. Faster sparse multivariate polynomial interpolation of straight-line programs. *J. Symbolic Comput.*, 75:4–24, 2016.

[9] S. J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Inform. Process. Lett.*, 18:147–150, 1984.

[10] J. Berthomieu and G. Lecerf. Reduction of bivariate polynomials from convex-dense to dense, with application to factorizations. *Math. Comp.*, 81(279), 2012.

[11] D. A. Bini and L. Gemignani. Fast fraction-free triangularization of Bezoutians with applications to sub-resultant chain computation. *Linear Algebra Appl.*, 284(1):19–39, 1998.

[12] M. Bôcher. *Introduction to higher algebra*. The MacMillan Compagny, 1907.
[13] R. P. Brent, F. G. Gustavson, and D. Y. Y. Yun. Fast solution of Toeplitz systems of equations and computation of Padé approximants. *J. Algorithms*, 1(3):259–295, 1980.
[14] W. S. Brown. On Euclid's algorithm and the computation of polynomial greatest common divisors. *J. ACM*, 18(4):478–504, 1971.
[15] W. S. Brown and J. F. Traub. On Euclid's algorithm and the theory of subresultants. *J. ACM*, 18(4):505–514, 1971.
[16] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*. Springer-Verlag, 1997.
[17] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Infor.*, 28(7):693–701, 1991.
[18] H. Cohen. *A Course in Computational Algebraic Number Theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, 1993.
[19] G. E. Collins. Subresultants and reduced polynomial remainder sequences. *J. ACM*, 14(1):128–142, 1967.
[20] G. E. Collins. The calculation of multivariate polynomial resultants. In S. R. Petrick, editor, *Proceedings of the second ACM symposium on Symbolic and algebraic manipulation (SYMSAC '71)*, pages 212–222. ACM, 1971.
[21] L. Ducos. Algorithme de Bareiss, algorithme des sous-résultants. *Informatique théorique et applications*, 30(4):319–347, 1996.
[22] L. Ducos. Optimizations of the subresultant algorithm. *J. Pure Appl. Algebra*, 145(2):149–163, 2000.
[23] M. El Kahoui. An elementary approach to subresultants theory. *J. Symbolic Comput.*, 35(3):281–292, 2003.
[24] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, New York, 3rd edition, 2013.
[25] J. von zur Gathen and T. Lücking. Subresultants revisited. *Theoret. Comput. Sci.*, 297(1):199–239, 2003.
[26] B. Grenet, J. van der Hoeven, and G. Lecerf. Deterministic root finding over finite fields using Graeffe transforms. *Appl. Algebra Engrg. Comm. Comput.*, 27(3):237–257, 2016.
[27] W. Habicht. Eine Verallgemeinerung des Sturmschen Wurzelzählverfahrens. *Comment. Math. Helv.*, 21(1):99–116, 1948.
[28] J. van der Hoeven and G. Lecerf. *Mathemagix User Guide*. École polytechnique, France, 2013. `http://hal.archives-ouvertes.fr/hal-00785549`.
[29] J. van der Hoeven and G. Lecerf. Sparse polynomial interpolation in practice. *ACM Commun. Comput. Algebra*, 48(4), 2014. In section "ISSAC 2014 Software Presentations".
[30] J. van der Hoeven, G. Lecerf, B. Mourrain, et al. Mathemagix, from 2002. Software available at `http://www.mathemagix.org`.
[31] J. van der Hoeven, G. Lecerf, and G. Quintin. Modular SIMD arithmetic in Mathemagix. *ACM Trans. Math. Softw.*, 43(1), 2016. Article 5.
[32] M. Kerber. Division-free computation of subresultants using Bezout matrices. *Int. J. Comput. Math.*, 86(12):2186–2200, 2009.
[33] D. Knuth. The analysis of algorithms. In *Actes du congrès international des mathématiciens 1970*, volume 3, pages 269–274. Gauthier-Villars, 1971.
[34] G. Lecerf. Fast separable factorization and applications. *Appl. Algebra Engrg. Comm. Comput.*, 19(2):135–160, 2008.
[35] D. H. Lehmer. Euclid's algorithm for large numbers. *Amer. Math. Monthly*, pages 227–233, 1938.
[36] Th. Lickteig and M.-F. Roy. Cauchy index computation. *Calcolo*, 33(3-4):337–351, 1996.
[37] Th. Lickteig and M.-F. Roy. Sylvester–Habicht sequences and fast Cauchy index computation. *J. Symbolic Comput.*, 31(3):315–341, 2001.
[38] H. Lombardi, M.-F. Roy, and M. Safey El Din. New structure theorem for subresultants. *J. Symbolic Comput.*, 29(4–5):663–690, 2000.
[39] R. Loos. Generalized polynomial remainder sequences. In B. Buchberger, G. E. Collins, and R. Loos, editors, *Computer algebra. Symbolic and Algebraic Computation*, volume 4 of *Computing Supplementa*, pages 115–137. Springer-Verlag, 1982.
[40] R. T. Moenck. Fast computation of GCDs. In *Proceedings of the fifth annual ACM symposium on Theory of computing (STOC '73)*, pages 142–151. ACM, 1973.
[41] D. Reischert. Asymptotically fast computation of subresultants. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, ISSAC '97, pages 233–240. ACM, 1997.
[42] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica*, 1(2):139–144, 1971.
[43] C. K. Yap. *Fundamental Problems in Algorithmic Algebra*. Oxford University Press, New York, 2000.