Relaxed lifting of triangular sets

In this chapter, we present a new lifting algorithm for triangular sets over p-adics. Our contribution is to give, for any p-adic triangular set, a shifted algorithm of which the triangular set is a fixed point. Then we can apply the recursive p-adic framework and deduce a relaxed lifting algorithm for this triangular set.

We compare our algorithm with the adaptation of the Newton-Hensel operator for triangular sets of [GLS01, HMW01, Sch02]. Our algorithm always improves the asymptotic cost in the precision for the special case of univariate representations. The general situation is more contrasted.

Finally we implement these algorithms in the C++ library Algebramix of Mathemagix [HLM+02] for the special case of univariate representations. Our new relaxed algorithm compares favorably on the examples. We mention that our on-line algorithm is currently connected to Kronecker inside Mathemagix with the help of G. Lecerf.

This chapter contains work in progress.

6.1 Introduction

6.1.1 Notations

Throughout this chapter, we use the notions and notations of Chapter 1, Section 1.1. In particular, we use the ring of p-adics R_p with its assumption on the length function λ and its complexity model. We recall that $\mathsf{I}(n)$ and $\mathsf{R}(n)$ denotes the cost of multiplying two p-adics of length n by respectively an off-line and an on-line algorithm.

In this chapter, we choose to denote elements by small letters, e.g. $a \in R_p$, vectors by bold fonts, e.g. $\boldsymbol{a} \in (R_p)^n$, and matrices by capital letters, e.g. $A \in \mathcal{M}_n(R_p)$. We denote by $\boldsymbol{v}_1 \cdot \boldsymbol{v}_2$ the inner product between two vectors and $c \times \boldsymbol{v}$ the coefficientwise product of a scalar c by a vector $\boldsymbol{v}$.

Let $\mathsf{M}(d_1, ..., d_n)$ denote the cost of multiplication of dense multivariate polynomials $P \in R[X_1, ..., X_n]$ satisfying $\deg_{X_i}(P) \leqslant d_i$ for all $1 \leqslant i \leqslant n$. By Kronecker substitution, we get that $\mathsf{M}(d_1, ..., d_n) = \mathcal{O}(\mathsf{M}(2^n\, d_1 \cdots d_n))$. We point to Chapter 1, Section 1.2 for details on the cost function M of polynomial multiplication. We denote by $\langle P_1, ..., P_k \rangle$ the ideal spanned by $P_1, ..., P_k \in R[X_1, ..., X_n]$.

Let us introduce the notion of univariate representation of a zero-dimensional ideal $\mathcal{I} \subseteq R[X_1, ..., X_n]$ for any ring R. An element P of $A := R[X_1, ..., X_n]/\mathcal{I}$ will be called *primitive* if the R-algebra $R[P]$ spanned by P is equal to A itself. If Λ is a primitive linear form in A, a *univariate representation* of A consists of polynomials $\mathfrak{P} = (Q, S_1, ..., S_n)$ in $R[T]$ with $\deg(S_i) < \deg(Q)$ such that we have a R-algebra isomorphism

$$\begin{array}{ccc} A = R[X_1, ..., X_n]/\mathcal{I} & \to & R[T]/(Q) \\ X_1, ..., X_n & \mapsto & S_1, ..., S_n \\ \Lambda & \mapsto & T. \end{array}$$

The oldest trace of this representation is to be found in [Kro82] and a few years later in [Kön03]. A good summary of their work can be found in [Mac16]. The shape lemma [GM89] states the existence of such a representation for a generic linear form Λ of a zero-dimensional ideal. Different algorithms compute this representation, from a geometric resolution [GHMP97, GHH+97, GLS01, HMW01] or using a Gröbner basis [Rou99].

When using univariate representations, the elements of $A \simeq R[T]/(Q)$ are then represented as univariate polynomials of degree less than $d := \deg(Q)$. Then, multiplication in A costs $\mathcal{O}(\mathsf{M}(d))$.

A *triangular set* is a set of n polynomials $\boldsymbol{t} = (t_1, ..., t_n) \subseteq R[X_1, ..., X_n]$ such that t_i is in $R[X_1, ..., X_i]$, monic and reduced with respect to $(t_1, ..., t_{i-1})$. The notion of triangular set comes from [Rit66] in the context of differential algebra. Many similar notions were introduced afterwards [Wu84, Laz91, Kal93, ALMM99]. Although all these notions do not coincide in general, they are the same for zero-dimensional ideals.

As it turns out, univariate representations can be seen as a special case of triangular sets. Indeed, with the notations above, the family $(Q(T), X_1 - S_1(T), ..., X_n - S_n(T))$ is a triangular set in the algebra $R[T, X_1, ..., X_n]$.

For any triangular set $\boldsymbol{t}$ in $R[X_1, ..., X_n]$, we define the number e of *essential variables* by $e := \#\{i | d_i > 1\}$ where $d_i := \deg_{X_i}(t_i)$. If r is a reduced normal form modulo $\boldsymbol{t}$, then r is written on e variables, that is $r \in R[X_j]_{j \in \{i | d_i > 1\}}$. Only those variables play a true role in the quotient algebra $A := R[X_1, ..., X_n]/\langle \boldsymbol{t} \rangle$. We define $\mathsf{Rem}(d_1, ..., d_e)$ to be the cost of reducing polynomials $P \in R[X_1, ..., X_n]$ satisfying $\deg_{X_i}(P) \leqslant 2(d_i - 1)$ modulo $\boldsymbol{t}$. As it turns out, the cost of arithmetic operations in the quotient algebra A is $\mathcal{O}(\mathsf{Rem}(d_1, ..., d_e))$ (see Section 6.2). The number e of *essential variables* plays an important role because $\mathsf{Rem}(d_1, ..., d_e)$ is exponential in e.

As in Chapter 3, we denote by ω the exponent of linear algebra on fields, so that we can multiply and invert matrices in $\mathcal{M}_{n \times n}(R)$ in $\mathcal{O}(n^\omega)$ arithmetic operations. We will also need to invert matrices over rings that are not fields, e.g. in quotients of polynomial ring $R[T]/(Q)$. We denote by $\mathcal{O}(n^\Omega)$ the arithmetic complexity of the elementary operations on $n \times n$ matrices over any commutative ring: addition, multiplication, determinant and adjoint matrix. In fact, Ω can be taken less than 2.70 [Ber84, Kal92, KV04]. For the special case of matrices over $R_p[T]/(Q)$, we combine linear algebra over $(R/(p))[T]/(Q)$ and Newton iteration to invert matrices in time $\mathcal{O}((n^\omega \mathsf{I}(N) + n^\Omega)\mathsf{M}(d))$, where $d := \deg_T(Q)$.

In this chapter, we denote by $\boldsymbol{f} = (f_1, ..., f_n) \in R[X_1, ..., X_n]$ a polynomial system given by an s.l.p. with L operations in $\{+, -, *\}$. If L_{f_i} is the evaluation complexity of only the output f_i, then we denote by $L^{\perp} := L_{f_1} + \cdots + L_{f_n}$ the complexity that corresponds to computing f_1, ..., f_n independently, that is without sharing any operations between the computation of different outputs f_i. Since $L_{f_i} \leqslant L$, we always have

$$L \leqslant L^{\perp} \leqslant n\, L.$$

When $\boldsymbol{f}$ is given as an s.l.p., its Jacobian matrix can be computed by an algorithm from Baur and Strassen [BS83]. This method uses $\mathcal{O}(L_{f_i})$ arithmetic operations to compute the gradient of f_i. Therefore, the Jacobian matrix of $\boldsymbol{f}$ can be evaluated in time $\mathcal{O}(L^{\perp})$.

6.1.2 Motivations

Lifting triangular sets (or univariate representations) is a crucial operation. Most implementations of algorithms that compute triangular set on rationals compute this object modulo a prime number, and then lift the representation. For example, the Kronecker software [L+02] for univariate representations and the RegularChains package [LMX05] of Maple for triangular sets use a lifting. Even better, the geometric resolution algorithm [GLS01, HMW01] which is implemented in Kronecker requires yet another lifting: a lifting on power series is employed to compute univariate representations of curves, which is a basic step of the algorithm.

As it turns out, most of the time required to compute triangular sets (or univariate representations) is spend in the lifting. Therefore, any improvement on the lifting complexity will have repercussions on the whole algorithm.

It was shown in Chapter 5 that relaxed algorithms could reduce the cost due to linear algebra when lifting a regular root of a polynomial system compared to offline, or zealous, algorithms. In the same way that the Newton iteration was adapted to lift univariate representations in [GLS01, HMW01] and then triangular sets in [Sch02], we adapt our relaxed approach to lift such objects with the hope of getting rid of the contribution of linear algebra in the complexity.

6.1.3 Results

Let $\boldsymbol{f} = (f_1, ..., f_n)$ be a polynomial system in $R[X_1, ..., X_n]$ and $\boldsymbol{t}_0$ be a triangular set in $R/(p)[X_1, ..., X_n]$ such that:

- $\boldsymbol{f}$ is given as an s.l.p. with inputs X_1, ..., X_n and n outputs corresponding to $f_1, ..., f_n$. This s.l.p. has operations in $\{+, -, *\}$ and can use constants in $A/\langle \boldsymbol{t}_0 \rangle$;

- $\boldsymbol{f} = 0$ in $R/(p)[X_1, ..., X_n]/\langle \boldsymbol{t}_0 \rangle$;
- the determinant of the Jacobian matrix $\mathrm{Jac}_{\boldsymbol{f}}$ in $\mathcal{M}_n(R/(p)[X_1, ..., X_n])$ must be invertible modulo $\boldsymbol{t}_0$.

This last condition is sufficient to have the existence and uniqueness of a triangular set $\boldsymbol{t}$ in $R_p[X_1, ..., X_n]$ which reduces to $\boldsymbol{t}_0$ modulo p and satisfies $\boldsymbol{f} = 0$ in $R_p[X_1, ..., X_n]/\langle \boldsymbol{t} \rangle$. From these inputs, we compute at some precision N this unique triangular set $\boldsymbol{t}$. We call this operation the lifting of the triangular set $\boldsymbol{t}$ at precision N.

Example 6.1. We consider the polynomial system $\boldsymbol{f} = (f_1, f_2)$ in $\mathbb{Z}[X_1, X_2]$ with

$$\begin{aligned} f_1 &:= 33\,X_2^3 + 14699\,X_2^2 + 276148\,X_1 + 6761112\,X_2 - 11842820 \\ f_2 &:= 66\,X_1\,X_2 + X_2^2 - 94\,X_1 - 75\,X_2 - 22. \end{aligned}$$

Let $\boldsymbol{t}_0$ be the triangular set of $(\mathbb{Z}/7\,\mathbb{Z})[X_1, X_2]$ given by

$$\boldsymbol{t}_0 := (X_1^2 + 5\,X_1, 3\,X_1\,X_2 + X_2^2 + 4\,X_1 + 2\,X_2 + 6).$$

We lift the triangular set $\boldsymbol{t}_0$ from $(\mathbb{Z}/7\,\mathbb{Z})[X_1, X_2]$ to a triangular set $\boldsymbol{t}$ in $\mathbb{Z}_7[X_1, X_2]$. At each step of the relaxed lifting, we increment the precision. So at the first step, we have

$$\boldsymbol{t} = (X_1^2 + (5 + 5 \cdot 7)\,X_1 + 7, (3 + 2 \cdot 7)\,X_1\,X_2 + X_2^2 + 4\,X_1 + (2 + 3 \cdot 7)\,X_2 + (6 + 3 \cdot 7))$$

in $(\mathbb{Z}_7/7^2\,\mathbb{Z}_7)[X_1, X_2]$. We iterate again and find

$$\begin{aligned} \boldsymbol{t} = (&X_1^2 + (5 + 5 \cdot 7 + 6 \cdot 7^2)\,X_1 + (7 + 7^2), \\ &(3 + 2 \cdot 7 + 7^2)\,X_1\,X_2 + X_2^2 + (4 + 5 \cdot 7^2)\,X_1 + (2 + 3 \cdot 7 + 5 \cdot 7^2)\,X_2 + \\ &(6 + 3 \cdot 7 + 6 \cdot 7^2)) \end{aligned}$$

in $(\mathbb{Z}_7/7^3\,\mathbb{Z}_7)[X_1, X_2]$. The precision is enough to recover the triangular set

$$\boldsymbol{t} := (X_1^2 - 9\,X_1 + 56, 66\,X_1\,X_2 + X_2^2 - 94\,X_1 - 75\,X_2 - 22) \in \mathbb{Z}[X_1, X_2].$$

Theorem 6.2. *With the former notations and hypotheses, we can lift the triangular set* $\boldsymbol{t}$ *at precision* N *in time*

$$[\mathcal{O}(n\,L\,\mathsf{R}(N)) + n^2 \log(n)^{\mathcal{O}(1)}\,N + \mathcal{O}(n^\Omega)]\,\mathsf{Rem}(d_1, ..., d_n).$$

A different technique improves the dominant asymptotic cost in the precision $n\,L\,\mathsf{R}(N)\,\mathsf{Rem}(d_1, ..., d_n)$ when the number e of essential variables is lower than n. This technique requires to solve a linear system where the matrix has finite precision. Since the definition of this matrix σ_B is quite technical, we just content ourselves with saying that its finite length, denoted by λ, satisfies $\lambda = \tilde{\mathcal{O}}(L\,d_1 \cdots d_n)$ and with

pointing to Formula 6.13 for a recursive definition of the rows of the matrix. In the special case where arithmetic operations in R_p have no carries, this length reduces to $\lambda = 1$.

Theorem 6.3. *We keep the former notations and hypotheses. We can lift the triangular set $\boldsymbol{t}$ at precision N in time*

$$\mathcal{O}([e\,L\,\mathsf{R}(N) + N\,\mathsf{MMR}(n,1,\lambda)/\lambda + n\,L\,N + n^{\Omega}]\,\mathsf{Rem}(d_1,...,d_n)),$$

that is

$$[\mathcal{O}(e\,L\,\mathsf{R}(N)) + n^2 \log(n)^{\mathcal{O}(1)} \log(\lambda)^{\mathcal{O}(1)}\,N + \mathcal{O}(n\,L\,N + n^{\Omega})]\,\mathsf{Rem}(d_1,...,d_n)$$

where λ satisfies $\lambda = \tilde{\mathcal{O}}(L\,d_1 \cdots d_n)$.

We deduce the following important corollary for univariate representations.

Corollary 6.4. (of Theorem 6.3) *Let $\boldsymbol{f} = (f_1, ..., f_n)$ be a polynomial system in $R[X_1, ..., X_n]$ given by an s.l.p. Γ and $\mathfrak{P}_0 = (Q_0, S_{1,0}, ..., S_{n,0})$ a univariate representation in $R/(p)[X_1, ..., X_n]$ such that $f(S_{1,0}, ..., S_{n,0}) = 0$ in $R/(p)[X]/Q_0$.*

Then there exists an integer λ satisfying $\lambda = \tilde{\mathcal{O}}(L\,d)$ such that we can lift the univariate representation $\mathfrak{P}$ at precision N in time

$$\mathcal{O}([L\,\mathsf{R}(N) + N\,\mathsf{MMR}(n,1,\lambda)/\lambda + n\,L\,N + n^{\Omega}]\,\mathsf{M}(d)).$$

Let us compare the relaxed approach to the off-line methods of Section 6.3. We focus on the asymptotic behavior in the precision N. For triangular sets, we have to compare the relaxed cost $n\,L\,\mathsf{R}(N)\,\mathsf{Rem}(d_1,...,d_n)$ to the zealous bound $\mathcal{O}(((L^{\perp} + n^{\omega})\,\mathsf{I}(N) + n^{\Omega})\,\mathsf{Rem}(d_1,...,d_n))$. In this case, we can hope for an improvement only when $n\,L \ll n^{\omega}$ and for precisions N where the ratio $\mathsf{R}(N)/\mathsf{I}(N)$ is moderate.

The relaxed approach for univariate representations is more profitable. The relaxed cost $L\,\mathsf{R}(N)\,\mathsf{M}(d_n)$ always compares favorably to the zealous cost $\mathcal{O}((L^{\perp} + n^{\Omega})\,\mathsf{I}(N)\,\mathsf{M}(d_n))$ for precisions N where the ratio $\mathsf{R}(N)/\mathsf{I}(N)$ is moderate.

6.2 Quotient and remainder modulo a triangular set

This section deals with Euclidean division modulo a triangular set. From now on, we denote by $\boldsymbol{t} = (t_1, ..., t_n)$ a triangular set of $R[X_1, ..., X_n]$. Computing remainders is a basic operation necessary to be able to compute with the quotient algebra $A := R[X_1, ..., X_n]/\langle \boldsymbol{t} \rangle$. We are also interested in the quotients of the division since we will need them later.

We start by defining quotients and remainder of the Euclidean division by $\boldsymbol{t}$ in a unique manner. Then we focus on computing this objects. We circumvent the fact that the size of the quotients is exponential in the size $d_1 \cdots d_n$ of the quotient algebra A by computing only reductions of the quotients modulo a triangular set. This leads us to Algorithms `Rem_triangular` and `Rem_quo_triangular`.

Canonical quotients and remainder For any $P \in R[X_1, ..., X_n]$, the existence of $r, q_1, ..., q_n \in R[X_1, ..., X_n]$ satisfying $P = r + q_1 t_1 + \cdots + q_n t_n$ and $\deg_{X_i}(r) < d_i$ is guaranteed because the elements of a triangular set are monic. The quotients $q_1, ...,$ q_n are not unique. For $1 \leqslant i < j \leqslant n$, let $z_{i,j}$ be the vector of $R[X_1, ..., X_n]^n$ with only t_j in the i-th position and $-t_i$ in the j-th position. We can add to any choice of quotients an element of the syzygy $R[X_1, ..., X_n]$-module spanned by the $(z_{i,j})_{1 \leqslant i < j \leqslant n}$ in $R[X_1, ..., X_n]^n$. Nevertheless, a canonical choice of quotient can be made, as for the division by a standard, or Gröbner, basis

Lemma 6.5. *For all $P \in R[X_1, ..., X_n]$, there exists a unique vector of polynomials $(r, q_1, ..., q_n)$ in $R[X_1, ..., X_n]^{n+1}$ such that*

$$P = r + q_1 t_1 + \cdots + q_n t_n$$

and for all $1 \leqslant i \leqslant n$, $\deg_{X_i}(r) < d_i$ and for all $1 \leqslant i < j \leqslant n$, $\deg_{X_j}(q_i) < d_j$.

Proof. Take any Euclidean decomposition $P = r + q_1 t_1 + \cdots + q_n t_n$ with $\deg_{X_i}(r) < d_i$. Then use the syzygies $(z_{1,i})_{1<i\leqslant n}$ to reduce the degree of q_1 in $X_2, ..., X_n$. Again use the syzygies $(z_{2,i})_{2<i\leqslant n}$ to reduce the degree of q_2 in $X_3, ..., X_n$. This last action do not change q_1. Continuing the process until we reduce the degree of q_{n-1} in X_n by $z_{n-1,n}$, we have exhibited a Euclidean decomposition satisfying the hypothesis of the lemma.

Now let us prove the uniqueness of $(r, q_1, ..., q_n)$. Because r is unique, we have to prove that if $q_1 t_1 + \cdots + q_n t_n = 0$ with $\deg_{X_j}(q_i) < d_j$ for all $1 \leqslant i < j \leqslant n$, then $q_1 = \cdots = q_n = 0$. By contradiction, we suppose there is such a decomposition with a non-zero q_i. Let j be the maximal index of a non-zero q_j. Then $\deg_{X_j}(q_j t_j) \geqslant d_j$ and in the same time

$$\begin{aligned}
\deg_{X_j}(q_j t_j) &= \deg_{X_j}(q_1 t_1 + \cdots + q_{j-1} t_{j-1}) \\
&\leqslant \max_{i<j}(\deg_{X_j}(q_i t_i)) \\
&\leqslant \max_{i<j}(\deg_{X_j}(q_i)) \\
&< d_j.
\end{aligned}$$

Contradiction. □

We call canonical quotients and remainder, those which satisfies the conditions of Lemma 6.5. We denote by $P \operatorname{rem} \boldsymbol{t}$ the remainder of P modulo $\boldsymbol{t}$. We can not compute the q_i because they suffer from the phenomenon of *intermediate expression swell*; in the computations of the remainder of a polynomial modulo a triangular set, the size of intermediate expressions, e.g. the size of the quotients, increases too

much. A quick estimate gives that the size of the quotients is exponential in the size $d_1 \cdots d_n$ of the quotient algebra A.

Fast multivariate Euclidean division by a triangular set Therefore we use a variant of the remainder algorithm of [LMMS09] that do not compute the entire quotients but modular reductions of them. Then we describe a second algorithm that keeps the quotient modulo another triangular set, avoiding once again to pay the cost due to their sizes.

We mention a different approach to compute remainders modulo a triangular set whose basic idea is to an evaluation / interpolation on the points of the variety defined by the triangular set [BCHS11]. The motivation behind this approach is to circumvent the exponential factor in the complexity. But because this approach can not be adapted to obtain the quotients, we will not use it here.

We denote by $d_i := \deg_{X_i}(t_i)$ the degree in X_i. For the sake of simplicity in our forthcoming algorithms, we will assume that the set of indices $\{i | d_i > 1\}$ of essential variables is $\{1, ..., e\}$, so that any reduced normal form r belongs to $R[X_1, ..., X_e]$.

Our algorithm `Rem_triangular` is a slight improvement of the algorithm of [LMMS09]: it does $3\,d_e$ recursive calls instead of $4\,d_e$. As a consequence, the exponential factor in the complexity is 3^e instead of 4^e. Algorithm `Rem_triangular` is meant to reduce the product of two reduced elements. Therefore we suppose that the input polynomial $P \in R[X_1, ..., X_e]$ satisfies $\deg_{X_i}(P) \leqslant 2\,(d_i - 1)$.

The forthcoming algorithm is a triangular version of the fast univariate division with remainder (see [GG03, Section 9.1]). If $P \in R[X_1, ..., X_e]$, we denote by $P[X_e^i] \in R[X_1, ..., X_{e-1}]$ the coefficient of P in X_e^i. If $\deg_{X_e}(P) = d$, we denote by $\mathrm{rev}_{X_e}(P)$ its reverse polynomial w.r.t. X_e defined by $\mathrm{rev}_{X_e}(P) := \sum_{i=0}^{d} (P[X_e^{d-i}])\, X_e^i$.

Algorithm `Rem_triangular`

Input: $\boldsymbol{t}$ and $P \in R[X_1, ..., X_e]$ such that $\deg_{X_i}(P) \leqslant 2\,(d_i - 1)$
Output: $r \in R[X_1, ..., X_e]$ reduced modulo $\boldsymbol{t}$ such that

$$r = P \operatorname{rem} \boldsymbol{t}.$$

1. Let $\boldsymbol{t}' := (t_1, ..., t_{e-1})$ and $R' := R[X_1, ..., X_{e-1}]/\langle \boldsymbol{t}' \rangle$.
   Compute the quotient q_e of P by t_e in $R'[X_e]$:
   a. $q_e := \sum_{i=d_e}^{2d_e-1}$ `Rem_triangular`$(\boldsymbol{t}', P[X_e^i])\, X_e^i$
   b. Precompute $I := 1/\mathrm{rev}_{X_e}(t_e) \operatorname{rem} X_e^{d_e-1}$ in $R'[X_e]$
   c. $q_e := \mathrm{rev}_{X_e}(q_e)\, I \operatorname{rem} X_e^{d_e}$ in $R'[X_e]$
   d. $q_e := \mathrm{rev}_{X_e}(q_e)$
2. $r := (P - q_e\, t_e) \operatorname{rem} X_e^{d_e}$ in $R[X_1, ..., X_{e-1}][X_e]$
3. $r := \sum_{i=0}^{d_e-1}$ `Rem_triangular`$(r[X_e^i], \boldsymbol{t}')\, X_e^i$
4. **return** r

The precomputation of step 1.b means that, as the object I depends only on $\boldsymbol{t}$, we compute it once and for all at the first call of Algorithm `Rem_triangular`.

In accord with the introduction, we denote by $\mathsf{Rem}(d_1, ..., d_e)$ the complexity of Algorithm `Rem_triangular`.

Proposition 6.6. *The algorithm* `Rem_triangular` *is correct and runs in time* $\mathsf{Rem}(d_1, ..., d_e) = \mathcal{O}(\mathsf{M}(3^e\, d_1 \cdots d_e))$.

Proof. Since Algorithm `Rem_triangular` is very similar to their algorithm, we refer to [LMMS09] for the proof of correction of our algorithm.

Step 1.c involves a multiplication in $R[X_1, ..., X_e]$ and a reduction by $\boldsymbol{t}'$ of the coefficients in X_e^i for $i < d_e$. So multivariate multiplications are used in steps 1.c and 2 and d_e reductions by $\boldsymbol{t}'$ are done in steps 1.a, 1.c and 2. Thus the complexity analysis becomes

$$\mathsf{Rem}(d_1, ..., d_e) \;=\; 3\, d_e\, \mathsf{Rem}(d_1, ..., d_{e-1}) + 2\, \mathsf{M}(d_1, ..., d_e),$$

and

$$\begin{aligned}
& \mathsf{Rem}(d_1, ..., d_e) \;=\; \mathcal{O}\left(\sum_{i=1}^{e} 3^{e-i}\, \mathsf{M}(d_1, ..., d_i)\, d_{i+1} \cdots d_e \right) \\
\Rightarrow\quad & \mathsf{Rem}(d_1, ..., d_e) \;=\; \mathcal{O}\left(\sum_{i=1}^{e} 3^{e-i}\, \mathsf{M}(2^i\, d_1 \cdots d_i)\, d_{i+1} \cdots d_e \right) \\
\Rightarrow\quad & \mathsf{Rem}(d_1, ..., d_e) \;=\; \mathcal{O}\left(\sum_{i=1}^{e} \mathsf{M}\left(3^e \left(\frac{2}{3} \right)^i d_1 \cdots d_e \right) \right) \\
\Rightarrow\quad & \mathsf{Rem}(d_1, ..., d_e) \;=\; \mathcal{O}(\mathsf{M}(3^e\, d_1 \cdots d_e)).
\end{aligned}$$

The precomputation of step 1.b costs $\mathcal{O}(\mathsf{M}(d_1, ..., d_e) + d_e\ \mathsf{Rem}(d_1, ..., d_{e-1}))$ by Newton iteration for the inversion, that is $\mathcal{O}(\mathsf{Rem}(d_1, ..., d_e))$. □

Remark that non-essential variables do not impact the complexity of our algorithm, *i.e.* $\mathsf{Rem}(d_1, ..., d_e, 1, ..., 1) = \mathsf{Rem}(d_1, ..., d_e)$. Indeed if $d_i = 1$, then we condition $\deg_{X_i}(P) \leqslant 2\,(d_i - 1)$ implies that P does not depend on X_i.

Recall that since the product of two reduced element in the quotient algebra $R[X_1, ..., X_e]/\langle \boldsymbol{t} \rangle$ satisfies the degree condition of the input of Algorithm `Rem_triangular`, arithmetic operations in this quotient algebra cost $\mathsf{M}(d_1, ..., d_n) + \mathsf{Rem}(d_1, ..., d_n)$, that is $\mathcal{O}(\mathsf{Rem}(d_1, ..., d_n))$.

Now we adapt Algorithm `Rem_triangular` to keep the quotients modulo another triangular set $\boldsymbol{t}^2$. In order to simplify the algorithm and because it suits our future needs, we assume that for all i, $\deg_{X_i}(t_i^1) = \deg_{X_i}(t_i^2)$ and still denote by d_i this degree.

Algorithm `Rem_quo_triangular`

Input:

- Triangular sets $\boldsymbol{t}^1$, $\boldsymbol{t}^2$
- $P \in R[X_1, ..., X_e]$ such that $\deg_{X_i}(P) \leqslant 2(d_i - 1)$

Output: $r, q_1, ..., q_e \in R[X_1, ..., X_e]$ reduced modulo $\boldsymbol{t}^1$ (or $\boldsymbol{t}^2$) such that

$$P = r + \sum_{i=1}^{e} q_i\, t_i^1 \quad \text{modulo } \langle \boldsymbol{t}^1 \rangle \langle \boldsymbol{t}^2 \rangle.$$

1. Let $\boldsymbol{t}^{2'} := (t_1^2, ..., t_{e-1}^2)$ and $R' := R[X_1, ..., X_{e-1}]/\langle \boldsymbol{t}^{2'} \rangle$. Compute the quotient q_e of P by t_e^1 in $R'[X_e]$:
   a. $q_e := \sum_{i=d_e}^{2d_e-1}$ `Rem_triangular`$(\boldsymbol{t}^{2'}, P[X_e^i])\, X_e^i$
   b. Precompute $I := 1/\mathrm{rev}_{X_e}(t_e^1) \operatorname{rem} X_e^{d_e-1}$ in $R'[X_e]$
   c. $q_e := (\mathrm{rev}_{X_e}(q_e)\, I) \operatorname{rem} X_e^{d_e}$ in $R'[X_e]$
   d. $q_e := \mathrm{rev}_{X_e}(q_e)$
2. $r := (P - q_e\, t_e^1)$ in $R[X_1, ..., X_e]$
3. $r_1 := r \operatorname{rem} X_e^{d_e}$ and $r_2 = r - r_1$
4. $0, q_1, ..., q_{e-1} := \sum_{i=d_e}^{2d_e-1}$ `Rem_quo_triangular`$(r_2[X_e^i], \boldsymbol{t}^{2'}, (t_1^1, ..., t_{e-1}^1))\, X_e^i$
5. **for** i from 1 to $e-1$
   $q_i' :=$ `Rem_triangular`$(q_i, \boldsymbol{t}^1)$
6. $r := r_1 + q_1' t_1^2 + \cdots + q_{e-1}' t_{e-1}^2$ in $R[X_1, ..., X_e]$
7. $r, q_1, ..., q_{e-1} := \sum_{i=0}^{d_e-1}$ `Rem_quo_triangular`$(r[X_e^i], \boldsymbol{t}^{1'}, \boldsymbol{t}^{2'})\, X_e^i$
8. **return** $r, q_1, ..., q_{e-1}, q_e$

We denote by $\mathsf{RemQuo}(d_1, ..., d_e)$ the complexity of `Rem_quo_triangular` for triangular sets $\boldsymbol{t}^1$ and $\boldsymbol{t}^2$ of same degrees $d_1, ..., d_e$.

Lemma 6.7. *If r is reduced modulo $\boldsymbol{t}^1$ and $P = r + \sum_{i=1}^{e} q_i\, t_i^1$ modulo the product ideal $\langle \boldsymbol{t}^1 \rangle \langle \boldsymbol{t}^2 \rangle$, then r is the reduced normal form of P modulo $\boldsymbol{t}^1$ and*

$$P = r + \sum_{i=1}^{e} q_i\, t_i^1 \quad \text{modulo } \boldsymbol{t}^2.$$

Proof. Since the product ideal $\langle \boldsymbol{t}^1 \rangle \langle \boldsymbol{t}^2 \rangle$ is included in both the ideals $\langle \boldsymbol{t}^1 \rangle$ and $\langle \boldsymbol{t}^2 \rangle$, the relation $P = r + \sum_{i=1}^{e} q_i\, t_i^1$ stands modulo both these ideals. So $P = r$ modulo $\boldsymbol{t}^1$ and since r is reduced, it is the reduced normal form of P. □

Proposition 6.8. *The algorithm* `Rem_quo_triangular` *is correct and its costs verifies* $\mathsf{RemQuo}(d_1, ..., d_e) = \mathcal{O}(e\, \mathsf{Rem}(d_1, ..., d_e))$.

Proof. We proceed recursively on the number e on variables involved in P. In one variable, our algorithm coincides with the fast univariate division with remainder (see [GG03, Section 9.1]). So it is correct and $\mathsf{RemQuo}(d_1) = \mathsf{Rem}(d_1)$.

Let's suppose that we have proved our claims in less that e variables. Since q_e is the quotient of P by t_e^1 in $(R[X_1, ..., X_{e-1}]/\langle \boldsymbol{t}^{2'} \rangle)[X_e]$, we have $r_2 = 0$ in $(R[X_1, ..., X_{e-1}]/\langle \boldsymbol{t}^{2'} \rangle)[X_e]$. By assumption, the recursive call of step 4 gives the decomposition

$$r_2 = \sum_{i=1}^{e-1} q_i\, t_i^2 \quad \text{modulo } \langle \boldsymbol{t}^{1'} \rangle\, \langle \boldsymbol{t}^{2'} \rangle$$

and also modulo $\langle \boldsymbol{t}^1 \rangle\, \langle \boldsymbol{t}^2 \rangle$. The reduction of the quotient of step 5 gives

$$r_2 = \sum_{i=1}^{e-1} q_i'\, t_i^2 \quad \text{modulo } \langle \boldsymbol{t}^1 \rangle\, \langle \boldsymbol{t}^2 \rangle$$

where the polynomials q_i' are reduced modulo $\boldsymbol{t}^1$. Therefore the polynomial $r_2' := \sum_{i=1}^{e-1} q_i'\, t_i^2$ has degree $\deg_{X_e}(r_2') < d_e$ and $\deg_{X_i}(r_2') < 2\, d_i$ for $i < e$. Because r_1 satisfies the same degree conditions, they are still satisfied by $r = r_1 + r_2'$. By the induction hypothesis, at step 7, we have for all $0 \leqslant i < d_e$ and $1 \leqslant j \leqslant e-1$, that $q_j[X_e^i]$ is reduced modulo $\boldsymbol{t}^{1'}$. Since q_j has degree less than d_e in X_e, it is reduced modulo $\boldsymbol{t}^1$. The last quotient q_e is also reduced because it was computed in $(R[X_1, ..., X_{e-1}]/\langle \boldsymbol{t}^{2'} \rangle)[X_e]$ and $\deg_{X_e}(q_e) = \deg_{X_e}(P) - \deg_{X_e}(t_e^1) < d_e$. Finally

$$\begin{aligned} P = r_1 + r_2 + q_e\, t_e^1 &= (r_1 + r_2') + q_e\, t_e^1 && \text{modulo } \langle \boldsymbol{t}^1 \rangle\, \langle \boldsymbol{t}^2 \rangle \\ &= \left(r + \sum_{i=1}^{e-1} q_i\, t_i^1 \right) + q_e\, t_e^1 && \text{modulo } \langle \boldsymbol{t}^1 \rangle\, \langle \boldsymbol{t}^2 \rangle. \end{aligned}$$

Concerning the complexity analysis, we have

$$\begin{aligned} \mathsf{RemQuo}(d_1, ..., d_e) &= 2\, d_e\, \mathsf{RemQuo}(d_1, ..., d_{e-1}) + 2\, d_e\, \mathsf{Rem}(d_1, ..., d_{e-1}) + \\ &\quad (e-1)\, \mathsf{Rem}(d_1, ..., d_e) + (e+1)\, \mathsf{M}(d_1, ..., d_e) \end{aligned}$$

which gives

$$\mathsf{RemQuo}(d_1, ..., d_e) = \mathcal{O}(e\, \mathsf{Rem}(d_1, ..., d_e)). \qquad \square$$

As for the remainder algorithm, we have $\mathsf{RemQuo}(d_1, ..., d_e, 1, ..., 1)$ equals to $\mathsf{RemQuo}(d_1, ..., d_e)$, so that

$$\mathsf{RemQuo}(d_1, ..., d_n) = \mathsf{RemQuo}(d_1, ..., d_e) = \mathcal{O}(e\, \mathsf{Rem}(d_1, ..., d_e)) = \mathcal{O}(e\, \mathsf{Rem}(d_1, ..., d_n)).$$

Also we notice that the remainder and quotients modulo $\langle \boldsymbol{t}^1 \rangle\, \langle \boldsymbol{t}^2 \rangle$ of the product of two reduced element in A costs $\mathcal{O}(\mathsf{RemQuo}(d_1, ..., d_e))$.

When we apply Algorithm `Rem_quo_triangular` to the triangular sets $\boldsymbol{t}$ and $\boldsymbol{t}_0$ in Section 6.4, the reductions modulo $\boldsymbol{t}^2 = \boldsymbol{t}_0$ will be cheaper than reductions modulo $\boldsymbol{t}^1 = \boldsymbol{t}$ since they can be done coefficientwise. However the overall costs of Algorithm `Rem_quo_triangular`$(\boldsymbol{t}, \boldsymbol{t}_0, \_)$ will remain bounded by $\mathcal{O}(e)$ times the cost of reduction by $\boldsymbol{t}$.

Finally, we point out the situation would have been different with naïve algorithms for the remainder and quotients. The naïve remainder algorithm reduces the leading terms in X_e one by one, as would do a Gröbner basis reduction algorithm for the lexicographical monomial ordering with $X_1 \ll \cdots \ll X_n$. This algorithm implicitly computes the whole quotients and therefore $\mathsf{RemQuo}(d_1, \ldots, d_n) = \mathsf{Rem}(d_1, \ldots, d_n)$ with naïve algorithms.

Shift index The shift index is a theoretical tool used to prove the correctness of the computation of recursive p-adic numbers (see Proposition 2.17). We assess the shift index of the two previous algorithms with respect to the p-adic coefficients of the triangular sets.

Lemma 6.9. *Let Γ be an s.l.p. with n inputs and one output which satisfies $\mathrm{sh}(\Gamma) \geqslant 0$. Let $\Gamma(\boldsymbol{t})$ denote the output of Γ on the inputs $\boldsymbol{t}$.*

Then one has, for any triangular set $\boldsymbol{t}^2$,

$$\mathrm{sh}(\boldsymbol{t} \mapsto \texttt{Rem\_triangular}\,(\boldsymbol{t}, \Gamma(\boldsymbol{t}))) \geqslant 0$$

$$\mathrm{sh}(\boldsymbol{t} \mapsto \texttt{Rem\_quo\_triangular}\,(\boldsymbol{t}, \boldsymbol{t}^2, \Gamma(\boldsymbol{t}))) \geqslant 0.$$

In other words, Lemma 6.9 states that if the nth p-adic coefficient of a polynomial $\Gamma(\boldsymbol{t})$ involves only the ith coefficients of $\boldsymbol{t}$ for $i \leqslant n$, then so it is for the polynomials `Rem_triangular` $(\boldsymbol{t}, \Gamma(\boldsymbol{t}))$ and `Rem_quo_triangular` $(\boldsymbol{t}, \boldsymbol{t}^2, \Gamma(\boldsymbol{t}))$. The notation $\boldsymbol{t} \mapsto \texttt{Rem\_triangular}\,(\boldsymbol{t}, \Gamma(\boldsymbol{t}))$ refers to an s.l.p. which takes as input $\boldsymbol{t}$ and outputs `Rem_triangular` $(\boldsymbol{t}, \Gamma(\boldsymbol{t}))$ (see Remark 2.2). The entries $\boldsymbol{t}$ are given by the list of their polynomial coefficients, so that we can reverse polynomials.

Proof. We prove it for `Rem_quo_triangular`, the other case being similar. We proceed by induction on the number n of variables involved in the input of $\boldsymbol{t}^1$. If no variables are involved, then our algorithm does nothing and its shift index is the one of Γ. From now on, let us assume that the result is valid for input of less than n variables.

First, we prove that the computations that leads to $I := 1/\mathrm{rev}_{X_n}(t_n) \operatorname{rem} X_n^{d_n - 1}$ in $R'[X_n]$ have a non-negative shift. Define $I_0 := 1$ and $I_\ell := I_{\ell-1} - I_{\ell-1}\,(\mathrm{rev}_{X_\ell}(t_\ell)\, I_{\ell-1} - 1)$ in $R'[X_n]/\langle X_n^\ell \rangle$. Thereby $I = I_{\lceil \log_2(d_n - 1) \rceil}$ modulo $X_n^{d_n - 1}$. Since I_0 has a non-negative shift index and since I_ℓ is obtained from $I_{\ell-1}$ by multiplication and reduction modulo $\boldsymbol{t}'$ of operands which have a non-negative shift, we deduce that I has itself a non-negative shift by the induction hypothesis.

Our algorithm uses only recursive calls in less variables, addition and multiplication. Since all these operations preserve a non-negative shift, we deduce that q_e, r and finally $r, q_1, \ldots, q_e$ have non-negative shift indices. □

6.3 Overview of off-line lifting algorithms

We present three existing lifting algorithms, which are off-line, in increasing order of generality (and complexity). First algorithm lifts only a regular root, so it applies only to triangular sets with $d_1 = \cdots = d_n = 1$. Second algorithm lifts a univariate representation, that is a triangular set with $d_1 = \cdots = d_{n-1} = 1$ and any degree d_n. And finally we present an algorithm that lift any triangular set.

6.3.1 Hensel-Newton local lifting of a root

We start by recalling the local Newton iterator, that lift a regular root of an algebraic system into the completion ring R_p. It was first introduced by [New36] for finding power series solutions of univariate polynomials with coefficients in $\Bbbk[[X]]$. This method allows a local study of an algebraic variety. A relaxed version of this algorithm is presented in Chapter 5.

We detail the Newton iteration that doubles the precision of a regular solution of the algebraic system $\boldsymbol{f}$.

Algorithm `Local_Newton_step`

Input:

- System of equation $\boldsymbol{f}$ and its Jacobian $\mathrm{Jac}_{\boldsymbol{f}}$ as an s.l.p.
- A root $\boldsymbol{S} = (S_1, ..., S_n)$ of $\boldsymbol{f}$ in $R/\left(p^{2^{m-1}}\right)$
- Inverse $\mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S})$ of $\mathrm{Jac}_{\boldsymbol{f}}(\boldsymbol{S})$ in $\mathcal{M}_n\left(R/\left(p^{2^{m-2}}\right)\right)$

Output:

- A root $\boldsymbol{S}' = (S'_1, ..., S'_n)$ of $\boldsymbol{f}$ in $R/(p^{2^m})$
- Inverse $\mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S})'$ of $\mathrm{Jac}_{\boldsymbol{f}}(\boldsymbol{S})$ in $\mathcal{M}_n\left(R/\left(p^{2^{m-1}}\right)\right)$

1. In $\mathcal{M}_n\left(R/\left(p^{2^{m-1}}\right)\right)$, compute

   $\mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S})' := \mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S}) - \mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S})\left(\mathrm{Jac}_{\boldsymbol{f}}(\boldsymbol{S}) \cdot \mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S}) - \mathrm{Id}_n\right)$
2. $\boldsymbol{S}' := \boldsymbol{S} - \mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S})' \cdot \boldsymbol{f}(\boldsymbol{S})$ in $(R/(p^{2^m})[T])^n$
3. **return** $\boldsymbol{S}', \mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S})'$

Proposition 6.10. *The algorithm* `Local_Newton_step_univariate` *is correct and costs* $\mathcal{O}((L^{\perp} + n^{\omega})\,\mathsf{I}(N) + n^{\Omega})$ *to lift a regular root in* R_p *at precision* N.

Proof. The proof of correctness is classical and can be found in [GG03]. The cost $\mathcal{O}(n^{\Omega})$ is to compute the inverse of the Jacobian matrix modulo p. The Jacobian matrix $\mathrm{Jac}_{\boldsymbol{f}}$ can be evaluated in $\mathcal{O}(L^{\perp})$ operations in $R/(p^{2^m})$, which each costs $\mathsf{I}(2^m)$, so the result follows. □

6.3.2 Hensel-Newton global lifting of univariate representation

The following algorithm lifts univariate representations under the condition that the Jacobian matrix is invertible modulo the univariate representation over $R/(p)$. This algorithm is a slight modification from the local Newton iterator. It was first introduced in [GLS01, HMW01] and generalizes the previous approach.

Algorithm `Global_Newton_step_univariate`

Input:

- System of equation $\boldsymbol{f}$ and its Jacobian $\mathrm{Jac}_{\boldsymbol{f}}$ as an s.l.p.
- $u = \lambda_1 X_1 + \cdots + \lambda_n X_n$ a linear form with $\lambda_i \in R$
- Univariate representation $\boldsymbol{S} = (S_1, \ldots, S_n)$ and Q in $R/\big(p^{2^{m-1}}\big)[T]$
- Inverse $\mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S})$ of $\mathrm{Jac}_{\boldsymbol{f}}(\boldsymbol{S})$ in $\mathcal{M}_n\big(R/\big(p^{2^{m-2}}\big)[T]/Q\big)$

Output:

- Univariate representation $\boldsymbol{S}' = (S'_1, \ldots, S'_n)$ and Q' in $R/(p^{2^m})[T]$
- Inverse $\mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S})'$ of $\mathrm{Jac}_{\boldsymbol{f}}(\boldsymbol{S})$ in $\mathcal{M}_n\big(R/\big(p^{2^{m-1}}\big)[T]/Q\big)$

1. In $\mathcal{M}_n\big(R/\big(p^{2^{m-1}}\big)[T]/Q\big)$, compute
$$\mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S})' := \mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S}) - \mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S})\,(\mathrm{Jac}_{\boldsymbol{f}}(\boldsymbol{S}) \cdot \mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S}) - \mathrm{Id}_n)$$
2. $\boldsymbol{S}' := (\boldsymbol{S} - \mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S})' \cdot \boldsymbol{f}(\boldsymbol{S})) \operatorname{rem} Q$ in $(R/(p^{2^m})[T])^n$
3. $\Delta := u(\boldsymbol{S}') - T$ in $(R/(p^{2^m})[T])^n$
4. $\boldsymbol{S}' := \boldsymbol{S}' - \Big(\Big(\frac{\partial \boldsymbol{S}'}{\partial T}\Delta\Big) \operatorname{rem} Q\Big)$ in $(R/(p^{2^m})[T])^n$
5. $Q' := Q - \big(\big(\frac{\partial Q}{\partial T}\Delta\big) \operatorname{rem} Q\big)$ in $R/(p^{2^m})[T]$
6. **return** $\boldsymbol{S}'$, Q' and $\mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{S})'$

Proposition 6.11. *The algorithm* `Global_Newton_step_univariate` *is correct and costs* $\mathcal{O}((L^{\perp}+n^{\omega})\,\mathsf{M}(d)\,\mathsf{I}(N)+n^{\Omega})$ *to lift a univariate representation at precision* N *under the condition that* $\mathrm{Jac}_{\boldsymbol{f}}(\boldsymbol{S})$ *is invertible in* $R/(p)[T]/Q$.

We refer to [GLS01] for a proof of this proposition.

6.3.3 Hensel-Newton global lifting of triangular sets

The following algorithm was introduced in [Sch02]. Roughly speaking, it can be seen as a classical Newton iteration for finding a zero of the function $\Phi\colon \boldsymbol{Y} \mapsto B \cdot \boldsymbol{Y} - \boldsymbol{f}$ where B is an element of $\mathcal{M}_n(R[X_1, \ldots, X_n])$ satisfying $\boldsymbol{f} = B \cdot \boldsymbol{t}$. This algorithm lifts any triangular set under an inversibility condition of the Jacobian matrix. In the special case of univariate representations, this algorithm does the same computations as Algorithm `Global_Newton_step_univariate`, but they are presented differently, with only matrix multiplications, and more concisely.

Example 6.12. We consider the polynomial system $\boldsymbol{f} = (f_1, f_2)$ in $\mathbb{Z}[X_1, X_2]$ with

$$\begin{aligned} f_1 &:= 33\,X_2^3 + 14699\,X_2^2 + 276148\,X_1 + 6761112\,X_2 - 11842820 \\ f_2 &:= 66\,X_1\,X_2 + X_2^2 - 94\,X_1 - 75\,X_2 - 22. \end{aligned}$$

Let $\boldsymbol{t}_0$ be the triangular set of $(\mathbb{Z}/7\,\mathbb{Z})[X_1, X_2]$ given by

$$\boldsymbol{t}_0 := (X_1^2 + 5\,X_1, 3\,X_1\,X_2 + X_2^2 + 4\,X_1 + 2\,X_2 + 6).$$

We lift the triangular set $\boldsymbol{t}_0$ from $(\mathbb{Z}/7\,\mathbb{Z})[X_1, X_2]$ to a triangular set $\boldsymbol{t}$ in $\mathbb{Z}_7[X_1, X_2]$. At each step of the off-line lifting, we double the precision. So at the first step, we have

$$\boldsymbol{t} = (X_1^2 + 40\,X_1 + 7, 17\,X_1\,X_2 + X_2^2 + 4\,X_1 + 23\,X_2 + 27) \in (\mathbb{Z}/7^2\,\mathbb{Z})[X_1, X_2].$$

We iterate again and find

$$\boldsymbol{t} = (X_1^2 + 2392\,X_1 + 56, 66\,X_1\,X_2 + X_2^2 + 2307\,X_1 + 2326\,X_2 + 2379)$$

in $(\mathbb{Z}/7^4\,\mathbb{Z})[X_1, X_2]$. The precision is enough to recover the triangular set

$$\boldsymbol{t} := (X_1^2 - 9\,X_1 + 56, 66\,X_1\,X_2 + X_2^2 - 94\,X_1 - 75\,X_2 - 22) \in \mathbb{Z}[X_1, X_2].$$

Algorithm `Global_Newton_step_triangular`

Input:

- System of equation $\boldsymbol{f}$ and its Jacobian $\mathrm{Jac}_{\boldsymbol{f}}$ as an s.l.p.
- Triangular set $\boldsymbol{t} = (t_1, ..., t_n)$ in $R/\big(p^{2^{m-1}}\big)[X_1, ..., X_n]$
- Inverse $\mathrm{IJac}_{\boldsymbol{t}}$ of $\mathrm{Jac}_{\boldsymbol{t}}$ in $\mathcal{M}_n\big(R/\big(p^{2^{m-2}}\big)[X_1, ..., X_n]/\langle \boldsymbol{t} \rangle\big)$

Output:

- Triangular set $\boldsymbol{t}' = (t'_1, ..., t'_n)$ in $R/(p^{2^m})[X_1, ..., X_n]$
- Inverse $\mathrm{IJac}_{\boldsymbol{f}}$ of $\mathrm{Jac}_{\boldsymbol{f}}$ in $\mathcal{M}_n\big(R/\big(p^{2^{m-1}}\big)[X_1, ..., X_n]/\langle \boldsymbol{t} \rangle\big)$

1. In $\mathcal{M}_n\big(R/\big(p^{2^{m-1}}\big)[X_1, ..., X_n]/\langle \boldsymbol{t} \rangle\big)$, compute
$$\mathrm{IJac}'_{\boldsymbol{f}} := \mathrm{IJac}_{\boldsymbol{f}} - \mathrm{IJac}_{\boldsymbol{f}}\,(\mathrm{Jac}_{\boldsymbol{f}} \cdot \mathrm{IJac}_{\boldsymbol{f}} - \mathrm{Id}_n)$$
2. $\delta\boldsymbol{t} := \mathrm{Jac}_{\boldsymbol{t}} \cdot \mathrm{IJac}_{\boldsymbol{f}}(\boldsymbol{t})' \cdot \boldsymbol{f}$ in $(R/(p^{2^m})[X_1, ..., X_n]/\langle \boldsymbol{t} \rangle)^n$
3. $\boldsymbol{t}' := \boldsymbol{t} + \delta\boldsymbol{t}$ in $(R/(p^{2^m})[X_1, ..., X_n]/\langle \boldsymbol{t} \rangle)^n$
4. **return** $\boldsymbol{t}'$

Proposition 6.13. *The algorithm* `Global_Newton_step_triangular` *is correct and costs* $\mathcal{O}((L^{\perp} + n^{\omega})\,\mathsf{Rem}(d_1, ..., d_n)\,\mathsf{I}(N))$ *to lift a triangular set at precision* N *under the condition that* $\mathrm{Jac}_{\boldsymbol{f}}$ *is invertible in* $R/(p)[X_1, ..., X_n]/\langle \boldsymbol{t} \rangle$.

6.4 Relaxed lifting of triangular sets

Let $\boldsymbol{t}_0$ be a triangular set of $R/(p)[X_1, ..., X_n]$. Define the $R/(p)$-algebra A_0 by $A_0 := R/(p)[X_1, ..., X_n]/\langle \boldsymbol{t}_0 \rangle$. Let $\boldsymbol{f}$ be given as an s.l.p. Γ with inputs $X_1, ..., X_n$ and n outputs corresponding to $f_1, ..., f_n$. The s.l.p. Γ has operations in $\{+, -, *\}$ and can use constants in $A/\langle \boldsymbol{t}_0 \rangle$. We assume that the triangular set $\boldsymbol{t}_0$ satisfies the property that $\mathrm{Jac}(\boldsymbol{f}_0)$ is invertible in $\mathcal{M}_n(A_0)$. Then there exists a unique triangular set $\boldsymbol{t}$ in $R_p[X_1, ..., X_n]$ which reduces to $\boldsymbol{t}_0$ modulo p and satisfies $\boldsymbol{f} = 0$ in $R_p[X_1, ..., X_n]/\langle \boldsymbol{t} \rangle$.

In this section we detail two relaxed algorithms that lift $\boldsymbol{t}$ at precision N. The first algorithm of Section 6.4.1 should be used for generic triangular set. We refine this algorithm in Section 6.4.2 for triangular sets with few essential variables, e.g. for univariate representations.

Throughout this section, we denote by $P_{-n}, ..., P_L$ the result sequence of the s.l.p. Γ on the input $X_1, ..., X_n$. Let r_i and $\boldsymbol{b}_i$ be the canonical remainder and quotients of P_i for $-n \leqslant i \leqslant L$. So we have $P_i = r_i + \boldsymbol{b}_i \, \boldsymbol{t} \in R[X_1, ..., X_n]$. Let $i_1, ..., i_n$ be the indices of the n outputs of Γ, so that we have $f_j = P_{i_j}$ for $1 \leqslant j \leqslant n$. We denote by $B \in \mathcal{M}_n(R[X_1, ..., X_n])$ the matrix whose jth row is $\boldsymbol{b}_{i_j}$. Therefore one has $\boldsymbol{f} = B \, \boldsymbol{t} \in \mathcal{M}_{n,1}(R[X_1, ..., X_n])$.

6.4.1 Using the quotient matrix

We define two maps σ and δ from R_p to R_p by $\sigma(a) = a_0$ and $\delta(a) := \frac{a - a_0}{p}$ for any $a = \sum_{i \in \mathbb{N}} a_i \, p^i \in R_p$. For any $a \in R_p$, we have $a = \sigma(a) + p \, \delta(a)$. We extend the definition of σ and δ to $A := R_p[X_1, ..., X_n]$ by mapping X_i to itself and to $\mathcal{M}_{r,s}(A)$ by acting componentwise. Thus $\sigma(\boldsymbol{t})$, also denoted by $\boldsymbol{t}_0$, is defined by $\sigma(\boldsymbol{t}) := (\sigma(t_1), ..., \sigma(t_n))$.

Recursive formula for $\boldsymbol{t}$ The triangular set $\boldsymbol{t}$ is a recursive p-adic vector of polynomials.

Lemma 6.14. *The matrix $\sigma(B) \in \mathcal{M}_n(R_p[X_1, ..., X_n])$ is invertible modulo $\boldsymbol{t}_0$. Moreover the triangular set $\boldsymbol{t}$ satisfies the recursive equation*

$$\boldsymbol{t} - \boldsymbol{t}_0 = \sigma(B)^{-1} \, (\boldsymbol{f} - p^2 \, (\delta(B) \cdot \delta(\boldsymbol{t}))) \text{ rem } \boldsymbol{t}_0 \tag{6.1}$$

in $\mathcal{M}_{n,1}(R_p[X_1, ..., X_n])$.

Proof. For any $P \in R[X_1, ..., X_n]$, let r and $\boldsymbol{a}$ be the canonical remainder and quotients of P by $\boldsymbol{t}$ so that

$$P - r = \boldsymbol{a} \cdot \boldsymbol{t} = \sigma(\boldsymbol{a}) \cdot \boldsymbol{t} + p \, \delta(\boldsymbol{a}) \cdot \boldsymbol{t} = \sigma(\boldsymbol{a}) \cdot \boldsymbol{t} + p \, \delta(\boldsymbol{a}) \cdot \boldsymbol{t}_0 + p^2 \, \delta(\boldsymbol{a}) \cdot \delta(\boldsymbol{t}).$$

Thus we have

$$\sigma(\boldsymbol{a}) \cdot \boldsymbol{t} = P - (r + p^2 \, \delta(\boldsymbol{a}) \cdot \delta(\boldsymbol{t}))$$

in $R[X_1, ..., X_n]/\langle \boldsymbol{t}_0 \rangle$. Now if $P \in \langle \boldsymbol{t} \rangle$, then $r = 0$ and we get

$$\sigma(\boldsymbol{a}) \cdot \boldsymbol{t} = P - p^2 \, (\delta(\boldsymbol{a}) \cdot \delta(\boldsymbol{t}))$$

in $R[X_1,...,X_n]/\langle \boldsymbol{t}_0\rangle$. We apply this to the equations $\boldsymbol{f}$ and get

$$\sigma(B)\cdot \boldsymbol{t} = \boldsymbol{f} - p^2\,(\delta(B)\cdot\delta(\boldsymbol{t})) \tag{6.2}$$

in $R[X_1,...,X_n]/\langle \boldsymbol{t}_0\rangle$.

By differentiating the equality $\boldsymbol{f}_0 = \sigma(B)\ \boldsymbol{t}_0 \in \mathcal{M}_{n,1}(R/(p)[X_1,\ ...,\ X_n])$, we get $\mathrm{Jac}(\boldsymbol{f}_0) = \sigma(B)\ \mathrm{Jac}(\boldsymbol{t}_0)$ in $\mathcal{M}_n(A_0)$. Since $\mathrm{Jac}(\boldsymbol{f}_0)$ is invertible in $\mathcal{M}_n(A_0)$ by hypothesis, B_0 and $\mathrm{Jac}(\boldsymbol{t}_0)$ are invertible in $\mathcal{M}_n(A_0)$ and

$$\sigma(B) = \mathrm{Jac}(\boldsymbol{f}_0)\,\mathrm{Jac}(\boldsymbol{t}_0)^{-1} \tag{6.3}$$

in $\mathcal{M}_n(A_0)$. Because its zeroth p-adic coefficient is invertible, we deduce that $\sigma(B)$ is invertible in $\mathcal{M}_n(R_p[X_1,...,X_n]/\langle \boldsymbol{t}_0\rangle)$. After inverting $\sigma(B)$ in Equation (6.2), it remains to notice that $\boldsymbol{t}-\boldsymbol{t}_0$ is the remainder of $\boldsymbol{t}$ by $\boldsymbol{t}_0$ to conclude. □

Let us explain the idea behind our algorithm. If one takes the coefficient in p^m of Equation (6.1) for $m \geqslant 1$, the left-hand side is the p-adic coefficient $\boldsymbol{t}_m := (t_{1,m},...,t_{n,m})$ of $\boldsymbol{t}$ but the right-hand side depends only on the p-adic coefficients B_i and $\boldsymbol{t}_i$ with $i<m$. Since the matrix B is made of quotients of $\boldsymbol{f}$ by the triangular basis $\boldsymbol{t}$, its coefficient B_i only depends on the coefficients $\boldsymbol{t}_j$ with $j \leqslant i$. So we can deduce $\boldsymbol{t}_m$ from the previous p-adic coefficients of $\boldsymbol{t}$, and compute $\boldsymbol{t}$ at any precision. Informally speaking, we have introduced a shift in the p-adic coefficients of $\boldsymbol{t}$ in the right-hand side.

Computation of $\boldsymbol{B}$ modulo $\boldsymbol{t_0}$ In this paragraph, we explain how to compute the remainder r_i and quotients $\boldsymbol{b}_i$ by $\boldsymbol{t}$ of any element P_i of the result sequence. Since Equation (6.1) is modulo $\boldsymbol{t}_0$, this quantities are only required modulo $\boldsymbol{t}_0$. We proceed recursively on the index i for $-n \leqslant i \leqslant L$.

First, for $-n < i \leqslant 0$, let $\bar{\imath} := i+n$ such that $P_i = X_{\bar{\imath}}$. We distinguish two cases :

- if $\deg_{X_{\bar{\imath}}}(t_{\bar{\imath}}) = 1$, then $\boldsymbol{b}_i := (0,...,0,1,0,...,0)$ with only a one in position $\bar{\imath}$ and $r_i = t_{\bar{\imath}} - X_{\bar{\imath}}$.
- if $\deg_{X_{\bar{\imath}}}(t_{\bar{\imath}}) > 1$, then $X_{\bar{\imath}}$ is already reduced modulo $\boldsymbol{t}$, we put $r_i := X_{\bar{\imath}}$ and $\boldsymbol{b}_i := (0,...,0)$.

Secondly, if the i-th result P_i is a constant in $A/\langle \boldsymbol{t}_0\rangle$, then it is reduced modulo $\boldsymbol{t}$ because $\deg_{X_i}(\sigma(t_i)) = \deg_{X_i}(t_i)$ for any $1 \leqslant i \leqslant n$. Consequently, we take $r_i := P_i$ and $\boldsymbol{b}_i := (0,...,0)$.

Let us consider the final case when $P_i = P_j \,\mathrm{op}\, P_k$ with $\mathrm{op} \in \{+,-,*\}$ and $j,k<i$. The case where op is the addition is straightforward

$$\begin{aligned} r_i &:= r_j + r_k \\ \boldsymbol{b}_i &:= \boldsymbol{b}_j + \boldsymbol{b}_k. \end{aligned}$$

The case of the subtraction is similar. Let us deal with the case of the multiplication. Let

$$s, \boldsymbol{q} := \texttt{Rem\_quo\_triangular}(\boldsymbol{t}, \boldsymbol{t}_0, r_j r_k)$$

be the reductions modulo $\boldsymbol{t}_0$ of the canonical remainder and quotients of $r_j r_k$ by $\boldsymbol{t}$. They satisfy

$$\begin{aligned} r_j r_k &= s && \text{in } A/\langle \boldsymbol{t}\rangle \\ r_j r_k &= s + \boldsymbol{q}\cdot\boldsymbol{t} && \text{in } A/\langle \boldsymbol{t}_0\rangle. \end{aligned}$$

Then one has in $A/\langle \boldsymbol{t}_0 \rangle$

$$\begin{aligned} P_j P_k &= r_j r_k + [(r_j + \boldsymbol{b}_j \cdot \boldsymbol{t}) \times \boldsymbol{b}_k + r_k \times \boldsymbol{b}_j] \cdot \boldsymbol{t} \\ &= s + [\boldsymbol{q} + (r_j + \boldsymbol{b}_j \cdot \boldsymbol{t}) \times \boldsymbol{b}_k + r_k \times \boldsymbol{b}_j] \cdot \boldsymbol{t} \end{aligned}$$

which implies, still in $A/\langle \boldsymbol{t}_0 \rangle$,

$$\begin{aligned} r_i &:= s \\ \boldsymbol{b}_i &:= \boldsymbol{q} + (r_j + \boldsymbol{b}_j \cdot \boldsymbol{t}) \times \boldsymbol{b}_k + r_k \times \boldsymbol{b}_j. \end{aligned} \tag{6.4}$$

We put all these formulas together to form an algorithm that computes all the remainders r_i and quotients $\boldsymbol{b}_i$ modulo $\boldsymbol{t}_0$. We describe this algorithm as a straight-line program, in order to prove that it is a part of a shifted algorithm.

Let L be the length of the s.l.p. Γ of $\boldsymbol{f}$. We define recursively in i such that $-n < i \leqslant L$ some s.l.p.'s ε^i with n inputs. These s.l.p.'s ε^i compute, on the entries $\boldsymbol{t}$ given as the list of their polynomial coefficients, the remainders r_j and quotients $\boldsymbol{b}_j$ of P_j for $j < i$. We call ρ^i and $\boldsymbol{\alpha}^i = (\alpha^i_1, ..., \alpha^i_n)$ the outputs of ε^i corresponding to r_i and $\boldsymbol{b}_i$.

Definition 6.15. *Let us initiate the induction for $-n < i \leqslant 0$ and $\bar{\imath} := i + n$:*

- *if $\deg_{X_{\bar{\imath}}}(t_{\bar{\imath}}) = 1$, then we define $\varepsilon^i := (-r_{\bar{\imath}}, 0, 1)$ where $r_{\bar{\imath}} := t_{\bar{\imath}} - X_{\bar{\imath}}$. The output ρ^i points to $-r_{\bar{\imath}}$ and α^i_m points to 0 if $m \neq \bar{\imath}$ or 1 otherwise;*
- *if $\deg_{X_{\bar{\imath}}}(t_{\bar{\imath}}) > 1$, then we define $\varepsilon^i := (X_{\bar{\imath}}, 0)$. The output ρ^i points to $X_{\bar{\imath}}$ and α^i_m points to 0 for any $1 \leqslant m \leqslant n$.*

Now recursively for $0 < i \leqslant L$, depending on the operation type of Γ_i:

- *if $\Gamma_i = (P^c)$ with $P \in A$ reduced modulo $\boldsymbol{t}_0$, then we define $\varepsilon^i := (P, 0)$. The output ρ^i points to P and α^i_m points to 0 for any $1 \leqslant m \leqslant n$;*
- *if $\Gamma_i = (+; u, v)$, then we build ε^i on top of ε^u and ε^v in such a manner that one has $\rho^i := \rho^u + \rho^v$ and $\boldsymbol{\alpha}^i := \boldsymbol{\alpha}^u + \boldsymbol{\alpha}^v$;*
- *if $\Gamma_i = (-; u, v)$, then we build ε^i on top of ε^u and ε^v in such a manner that one has $\rho^i := \rho^u - \rho^v$ and $\boldsymbol{\alpha}^i := \boldsymbol{\alpha}^u - \boldsymbol{\alpha}^v$;*
- *if $\Gamma_i = (*; u, v)$, we define ε^i accordingly to formula (6.4). First, we compute $s, \boldsymbol{q} := \texttt{Rem\_quo\_triangular}(\rho^u(\boldsymbol{t})\ \rho^v(\boldsymbol{t}), \boldsymbol{t}, \boldsymbol{t}_0)$. Then $\rho^i := s$ and $\boldsymbol{\alpha}^i$ is defined by*

$$\boldsymbol{q} + (\rho^u(\boldsymbol{t}) + \boldsymbol{\alpha}^u(\boldsymbol{t}) \cdot \boldsymbol{t}) \times \boldsymbol{\alpha}^v(\boldsymbol{t}) + \rho^v(\boldsymbol{t}) \times \boldsymbol{\alpha}^u(\boldsymbol{t}).$$

Finally, we set $\varepsilon = \varepsilon^L$.

Shifted algorithm In this paragraph, we prove that formula (6.1) gives rise to a shifted algorithm to compute $\boldsymbol{t}$. Mainly, we have to prove that the p-adic coefficient in p^m of $p^2\,(\delta(B) \cdot \delta(\boldsymbol{t}))$, that is the coefficient in p^{m-2} of $\delta(B) \cdot \delta(\boldsymbol{t})$, depend at most in the coefficients $\boldsymbol{t}_i$ of $\boldsymbol{t}$ with $i < m$. For that matter, we will compute the shift index of the computation of $p^2\,(\delta(B) \cdot \delta(\boldsymbol{t}))$ and prove that it is positive.

Since the s.l.p. ε computes the matrix B on the entries $\boldsymbol{t}$, we can build an s.l.p. $\boldsymbol{\Lambda}$ on top of ε such that

$$\boldsymbol{\Lambda}: \boldsymbol{t} \mapsto \boldsymbol{t}_0 + [\sigma(B)^{-1} (\boldsymbol{f} - p^2 \times (\delta(B) \cdot \delta(\boldsymbol{t}))) \text{ rem } \boldsymbol{t}_0].$$

In the s.l.p. $\boldsymbol{\Lambda}$, the resolution of the linear system

$$\sigma(B)\, \boldsymbol{a} = (\boldsymbol{f} - p^2 \times (\delta(B) \cdot \delta(\boldsymbol{t}))) \in \mathcal{M}_{n,1}(A/\langle \boldsymbol{t}_0 \rangle)$$

in $\boldsymbol{a}$ is performed by the relaxed algorithm of Chapter 3, Section 3.3.2. Indeed, $\sigma(B)$ has length 1 and this algorithm is adapted to low length matrices.

Lemma 6.16. *The s.l.p. $\boldsymbol{\Lambda}$ is a shifted algorithm of which $\boldsymbol{t}$ is a fixed point when the computations are done in the algebra $R_p[X_1, ..., X_n]$.*

Proof. The triangular set $\boldsymbol{t}$ is a fixed point of the s.l.p. $\boldsymbol{\Lambda}$ over $R_p[X_1, ..., X_n]$ because of Equation (6.1).

Since the s.l.p. ε uses only additions, subtractions, multiplications, calls to `Rem_triangular` and `Rem_quo_triangular`, and since all these operations preserve a non-negative shift index (Lemma 6.9), we know that $\mathrm{sh}(\boldsymbol{t} \mapsto B) \geqslant 0$. Besides

$$\begin{aligned} \mathrm{sh}(\boldsymbol{t} \mapsto p^2 \times (\delta(B) \cdot \delta(\boldsymbol{t}))) &= 2 + \mathrm{sh}(\boldsymbol{t} \mapsto \delta(B) \cdot \delta(\boldsymbol{t})) \\ &= 2 + \min(\mathrm{sh}(\boldsymbol{t} \mapsto \delta(B)), \mathrm{sh}(\boldsymbol{t} \mapsto \delta(\boldsymbol{t}))) \\ &= 1 + \min(\mathrm{sh}(\boldsymbol{t} \mapsto B), \mathrm{sh}(\boldsymbol{t} \mapsto \boldsymbol{t})) \\ &\geqslant 1. \end{aligned}$$

Furthermore, notice that $\boldsymbol{f} \operatorname{rem} \boldsymbol{t}_0$ and $\sigma(B)$ depend only on $\boldsymbol{t}_0$. Finally the resolution of the linear system does not change the shift, hence we have proved that $\mathrm{sh}(\boldsymbol{\Lambda}) > 0$. □

Proof. (of Theorem 6.2) The triangular set $\boldsymbol{t}$ is a fixed point of the s.l.p. $\boldsymbol{\Lambda}$, which is a shifted algorithm by Lemma 6.16. Proposition 2.17 shows that we can compute $\boldsymbol{t}$ in time the number of operations in $\boldsymbol{\Lambda}$.

We count the number of operations of $\boldsymbol{\Lambda}$:

- Computation of the remainder r and the quotients $\boldsymbol{b}$ at each step of the computation of $\boldsymbol{f}$:

  We focus on the steps which correspond to a multiplication $*$ in the s.l.p. $\boldsymbol{f}$ because they have the worst complexity. The remainder and quotients require a call to Algorithm `Rem_quo_triangular`. Then $\boldsymbol{b}$ uses an inner product and scalar vector multiplications $\times$. The inner product costs less than a call to `Rem_quo_triangular`, since this latter algorithm does an inner product. Summing up, the total cost is

  $$\mathcal{O}(L\, \mathsf{R}(N)\, \mathsf{RemQuo}(d_1, ..., d_n) + n\, L\, \mathsf{R}(N)\, \mathsf{Rem}(d_1, ..., d_n))$$

  that is $\mathcal{O}(n\, L\, \mathsf{R}(N)\, \mathsf{Rem}(d_1, ..., d_n))$ (see Proposition 6.8);
- Computation of $\boldsymbol{f} \operatorname{rem} \boldsymbol{t}_0$ in time $\mathcal{O}(L\, \mathsf{Rem}(d_1, ..., d_n)\, \mathsf{R}(N))$;
- Computation of $p^2 \times (\delta(B) \cdot \delta(\boldsymbol{t}))$ requires n inner products $\delta(b_i) \cdot \delta(\boldsymbol{t})$, whose costs are dominated by $\mathcal{O}(n\, \mathsf{R}(N)\, \mathsf{RemQuo}(d_1, ..., d_n))$, which is bounded by $\mathcal{O}(n\, L\, \mathsf{R}(N)\, \mathsf{Rem}(d_1, ..., d_n))$ since $L \geqslant n$;

- Resolution of the linear system in $\sigma(B)$:
  Since $\sigma(B)$ has length one, Proposition 3.6 solves the linear system in time $\mathcal{O}(N\,\mathsf{MMR}(n,1,1)/1+n^\Omega)=\tilde{\mathcal{O}}(n^2)\,N+\mathcal{O}(n^\Omega)$. □

6.4.2 By-passing the whole quotient matrix

In the algorithm of Section 6.4.1, we computed the whole quotient B. This raised a component $\mathcal{O}(n\,L\,\mathsf{R}(N)\,\mathsf{Rem}(d_1,\ldots,d_n))$ in the complexity. We also had to call `Rem_quo_triangular` for each multiplication in the s.l.p. of $\boldsymbol{f}$, leading to a cost of $\mathcal{O}(e\,L\,\mathsf{R}(N)\,\mathsf{Rem}(d_1,\ldots,d_n))$. These two costs are balanced when $e\simeq n$.

However, when $e\ll n$, we can benefit from not computing the whole quotient B. We present in this section a new method to compute $\delta(B)\cdot\delta(\boldsymbol{t})$ without computing B, thus leading to an asymptotic complexity of $\mathcal{O}(L\,\mathsf{R}(N)\,\mathsf{Rem}(d_1,\ldots,d_n))$ plus some calls to `Rem_quo_triangular`. That is how we reach a total complexity of $\mathcal{O}(e\,L\,\mathsf{R}(N)\,\mathsf{Rem}(d_1,\ldots,d_n))$.

Nevertheless, this new method makes it harder to deal with the carries involved in the computation of B. We introduce the notion of shifted decomposition to solve this issue. In return, we increase the subdominant part of the complexity when N tends to infinity.

Shifted decomposition Recall that σ and δ were defined by $\sigma(a)=a_0$ and $\delta(a):=\frac{a-a_0}{p}$ and that, for any $a\in R_p$, we have $a=a_0+p\,\delta(a)$.

To our great regret, σ and δ are not ring homomorphisms. To remedy this fact, we call a *shifted decomposition* of $a\in R_p$ a pair $(\sigma_a,\,\delta_a)\in R_p^2$ such that $a=\sigma_a+p\,\delta_a$. Shifted decompositions are not unique. For any $a\in R_p$, the pair $(\sigma(a),\delta(a))$ is called the *canonical* shifted decomposition of a. Because σ and δ are not ring homomorphisms, we will use another shifted decomposition that behaves better with respect to arithmetic operations.

Lemma 6.17. *Let $a,b\in R_p$ and $(\sigma_a,\delta_a),(\sigma_b,\delta_b)\in R_p^2$ be shifted decompositions of a and b. Then*

1. *$(\sigma_a+\sigma_b,\delta_a+\delta_b)$ is a shifted decomposition of $a+b$;*
2. *$(\sigma_a-\sigma_b,\delta_a-\delta_b)$ is a shifted decomposition of $a-b$;*
3. *$(\sigma_a\,\sigma_b,\delta_a\,\sigma_b+a\,\delta_b)$ and $(\sigma_a\,\sigma_b,\delta_a\,b+\sigma_a\,\delta_b)$ are shifted decompositions of $a\,b$.*

Proof. These shifted decompositions are direct consequences of the relations

$$a+b = \sigma_a+\sigma_b+p\,(\delta_a+\delta_b) \tag{6.5}$$

$$-a = -\sigma_a+p\,(-\delta_a) \tag{6.6}$$

$$\begin{aligned} a\,b &= \sigma_a\,\sigma_b+p\,(\delta_a\,b+\sigma_a\,\delta_b)\\ &= \sigma_a\,\sigma_b+p\,(\delta_a\,\sigma_b+a\,\delta_b). \end{aligned} \tag{6.7}$$

□

We extend the notion of shifted decomposition naturally to polynomials $R_p[X_1, ..., X_n]$, vectors $(R_p)^n$ and matrices $\mathcal{M}_{r,s}(R_p)$.

Recursive formula for $\boldsymbol{t}$ The recursive formula (6.1) for $\boldsymbol{t}$ adapts well to shifted decomposition.

Lemma 6.18. *Let (σ_B, δ_B) be any shifted decomposition of the quotient matrix $B \in \mathcal{M}_n(R_p[X_1, ..., X_n])$.*

Then the matrix $\sigma_B \in \mathcal{M}_n(R_p[X_1, ..., X_n])$ is invertible modulo $\boldsymbol{t}_0$. Moreover the triangular set $\boldsymbol{t}$ satisfies the recursive equation

$$\boldsymbol{t} - \boldsymbol{t}_0 = \sigma_B^{-1} \left(\boldsymbol{f} - p^2 \left(\delta_B \cdot \delta(\boldsymbol{t}) \right) \right) \text{ rem } \boldsymbol{t}_0 \tag{6.8}$$

in $\mathcal{M}_{n,1}(R_p[X_1, ..., X_n])$.

Proof. We proceed similarly to the proof of Lemma 6.14. For any $P \in R[X_1, ..., X_n]$, let r and $\boldsymbol{a}$ be the canonical remainder and quotients of P by $\boldsymbol{t}$. For any shifted decomposition $(\sigma_{\boldsymbol{a}}, \delta_{\boldsymbol{a}})$ of $\boldsymbol{a}$, one has

$$P - r = \boldsymbol{a} \cdot \boldsymbol{t} = \sigma_{\boldsymbol{a}} \cdot \boldsymbol{t} + p\, \delta_{\boldsymbol{a}} \cdot \boldsymbol{t} = \sigma_{\boldsymbol{a}} \cdot \boldsymbol{t} + p\, \delta_{\boldsymbol{a}} \cdot \boldsymbol{t}_0 + p^2\, \delta_{\boldsymbol{a}} \cdot \delta(\boldsymbol{t}).$$

We apply this to the equations $\boldsymbol{f}$ and get

$$\sigma_B \cdot \boldsymbol{t} = \boldsymbol{f} - p^2 \left(\delta_B \cdot \delta(\boldsymbol{t}) \right)$$

in $R_p[X_1, ..., X_n]/\langle \boldsymbol{t}_0 \rangle$.

Since the zeroth p-adic coefficient of σ_B is the one of B which in invertible in $\mathcal{M}_n(A_0)$ (see the proof of Lemma 6.14), we deduce that σ_B is invertible in $\mathcal{M}_n(R_p[X_1, ..., X_n]/\langle \boldsymbol{t}_0 \rangle)$. It remains to invert σ_B and to notice that $\boldsymbol{t} - \boldsymbol{t}_0 = \boldsymbol{t}$ rem $\boldsymbol{t}_0$ to conclude. □

Computation of $\boldsymbol{r}$, $\boldsymbol{\sigma_B}$ and $\boldsymbol{\delta_B \cdot \delta(t)}$ For every multiplication of the s.l.p. Γ of $\boldsymbol{f}$, we did n calls to `Rem_triangular` and one call to `Rem_quo_triangular` to compute the corresponding quotients with our first method of subsection 6.4.1. In this paragraph, we present a method that does only $\mathcal{O}(1)$ calls to `Rem_triangular` and one call to `Rem_quo_triangular` in the same situation.

We denote by $(\sigma_{\boldsymbol{b}_i}, \delta_{\boldsymbol{b}_i})$ a shifted decomposition of the quotients $\boldsymbol{b}_i$. The main idea of our new method is to deal with $\delta_{\boldsymbol{b}_i} \cdot \delta(\boldsymbol{t}) \in R_p$ instead of $\delta_{\boldsymbol{b}_i} \in (R_p)^n$. Let us explain how to compute r_i, $\sigma_{\boldsymbol{b}_i}$ and $\delta_{\boldsymbol{b}_i} \cdot \delta(\boldsymbol{t})$. We proceed recursively on the index i for $-n < i \leqslant L$.

First, for an index i corresponding to an input, *i.e.* $-n < i \leqslant 0$, we set $\bar{\imath} := i + n$. Therefore $P_i = X_{\bar{\imath}}$ and we distinguish two cases:

- if $\deg_{X_{\bar{\imath}}}(t_{\bar{\imath}}) = 1$, then we set $r_i := t_{\bar{\imath}} - X_{\bar{\imath}} \in R_p[X_1, ..., X_{i-1}]$ reduced with respect to $t_1, ..., t_{i-1}$. Also we set $\sigma_{\boldsymbol{b}_i} := (0, ..., 0, 1, 0, ..., 0)$ the vector with only a one at position $\bar{\imath}$ and $\delta_{\boldsymbol{b}_i} \cdot \delta(\boldsymbol{t}) = 0$.
- if $\deg_{X_{\bar{\imath}}}(t_{\bar{\imath}}) > 1$, then $X_{\bar{\imath}}$ is already reduced modulo $\boldsymbol{t}$ and we take

$$r_i := X_{\bar{\imath}}, \quad \sigma_{\boldsymbol{b}_i} := (0, ..., 0), \quad \delta_{\boldsymbol{b}_i} \cdot \delta(\boldsymbol{t}) = 0. \tag{6.9}$$

Now let $0 < i \leqslant L$ that corresponds to operations in Γ. If the i-th result P_i is a constant in $A/\langle \boldsymbol{t}_0 \rangle$, then, as before, we take

$$r_i := P_i, \quad \sigma_{\boldsymbol{b}_i} := (0, ..., 0), \quad \delta_{\boldsymbol{b}_i} \cdot \delta(\boldsymbol{t}) = 0. \tag{6.10}$$

Consider the final case when $P_i = P_j \,\mathrm{op}\, P_k$ with $\mathrm{op} \in \{+, -, *\}$ and $j, k < i$. The case where op is the addition is straightforward; using Lemma 6.17, one takes

$$\begin{aligned} r_i &:= r_j + r_k \\ \sigma_{\boldsymbol{b}_i} &:= \sigma_{\boldsymbol{b}_j} + \sigma_{\boldsymbol{b}_k} \\ \delta_{\boldsymbol{b}_i} \cdot \delta(\boldsymbol{t}) &:= \delta_{\boldsymbol{b}_j} \cdot \delta(\boldsymbol{t}) + \delta_{\boldsymbol{b}_k} \cdot \delta(\boldsymbol{t}). \end{aligned} \tag{6.11}$$

The case of subtraction is similar. Let us deal with the more complicated case of multiplication. We start by computing the remainder and quotients

$$s, \boldsymbol{q} := \texttt{Rem\_quo\_triangular}(\boldsymbol{t}, \boldsymbol{t}_0, r_j r_k)$$

of $r_j r_k$ by $\boldsymbol{t}$ modulo $\boldsymbol{t}_0$. They satisfy

$$\begin{aligned} r_j r_k &= s && \text{in } A/\langle \boldsymbol{t} \rangle \\ r_j r_k &= s + \boldsymbol{q} \cdot \boldsymbol{t} && \text{in } A/\langle \boldsymbol{t}_0 \rangle. \end{aligned}$$

Thus we still have over $A/\langle \boldsymbol{t}_0 \rangle$

$$\begin{aligned} r_i &:= s \\ \boldsymbol{b}_i &:= \boldsymbol{q} + (r_j + \boldsymbol{b}_j \cdot \boldsymbol{t}) \times \boldsymbol{b}_k + r_k \times \boldsymbol{b}_j. \end{aligned} \tag{6.12}$$

We use the formulas of Lemma 6.17 to compute the shifted decomposition of $\boldsymbol{b}_i$ from shifted decompositions of its operands. Shifted decompositions of $\boldsymbol{b}_j$ and $\boldsymbol{b}_k$ were computed at a previous step of the recursion. We choose to take the canonical shifted decomposition for $r_j, r_k, \boldsymbol{q}$ and $\boldsymbol{t}$. Since the scalar multiplication operator $\times$ and the inner product $\cdot$ are made of additions and multiplications, we deduce that we can take

$$\begin{aligned} \sigma_{\boldsymbol{b}_i} &= \boldsymbol{q}_0 + ((r_j)_0 + \sigma_{\boldsymbol{b}_j} \cdot \boldsymbol{t}_0) \times \sigma_{\boldsymbol{b}_k} + (r_k)_0 \times \sigma_{\boldsymbol{b}_j} \\ \delta_{\boldsymbol{b}_i} &= \delta(\boldsymbol{q}) + (\delta(r_j) + \delta_{\boldsymbol{b}_j} \cdot \boldsymbol{t}_0 + \boldsymbol{b}_j \cdot \delta(\boldsymbol{t})) \times \boldsymbol{b}_k + ((r_j)_0 + \sigma_{\boldsymbol{b}_j} \cdot \boldsymbol{t}_0) \times \delta_{\boldsymbol{b}_k} + \delta(r_k) \times \boldsymbol{b}_j + \\ &\quad (r_k)_0 \times \delta_{\boldsymbol{b}_j}. \end{aligned}$$

Because we work in $A/\langle \boldsymbol{t}_0 \rangle$, this decomposition simplifies and we define

$$\begin{aligned} \sigma_{\boldsymbol{b}_i} &:= \boldsymbol{q}_0 + (r_j)_0 \times \sigma_{\boldsymbol{b}_k} + (r_k)_0 \times \sigma_{\boldsymbol{b}_j} \\ \delta_{\boldsymbol{b}_i} &:= \delta(\boldsymbol{q}) + \delta(r_k) \times \boldsymbol{b}_j + (r_k)_0 \times \delta_{\boldsymbol{b}_j} + \\ &\quad (\delta(r_j) + \boldsymbol{b}_j \cdot \delta(\boldsymbol{t})) \times \boldsymbol{b}_k + (r_j)_0 \times \delta_{\boldsymbol{b}_k}. \end{aligned} \tag{6.13}$$

Now that we have computed r_i and $\sigma_{\boldsymbol{a}_i}$, it remains to compute $\delta_{\boldsymbol{b}_i} \cdot \delta(\boldsymbol{t})$. Using $\sigma_{\boldsymbol{b}_j}$, $\sigma_{\boldsymbol{b}_k}, \delta_{\boldsymbol{b}_j} \cdot \delta(\boldsymbol{t}), \delta_{\boldsymbol{b}_k} \cdot \delta(\boldsymbol{t})$ and other known polynomials, we compute

$$\begin{aligned} \delta_{\boldsymbol{b}_i} \cdot \delta(\boldsymbol{t}) &:= \delta(\boldsymbol{q}) \cdot \delta(\boldsymbol{t}) + \delta(r_k)\,(\boldsymbol{b}_j \cdot \delta(\boldsymbol{t})) + (r_k)_0\,(\delta_{\boldsymbol{b}_j} \cdot \delta(\boldsymbol{t})) + \\ &\quad (\delta(r_j) + \boldsymbol{b}_j \cdot \delta(\boldsymbol{t}))\,(\boldsymbol{b}_k \cdot \delta(\boldsymbol{t})) + (r_j)_0\,(\delta_{\boldsymbol{b}_k} \cdot \delta(\boldsymbol{t})) \end{aligned} \tag{6.14}$$

where $\boldsymbol{b}_j \cdot \delta(\boldsymbol{t}) := \sigma_{\boldsymbol{b}_j} \cdot \delta(\boldsymbol{t}) + p\,(\delta_{\boldsymbol{b}_j} \cdot \delta(\boldsymbol{t}))$ and the same for $\boldsymbol{b}_k \cdot \delta(\boldsymbol{t})$. This formula is new and admits no equivalents for canonical shifted decompositions when the p-adics have carries.

We sum up all these computations in an algorithm. We define recursively for $-n < i \leqslant L$ some s.l.p.'s ξ^i with n inputs. These s.l.p.'s ξ^i compute, on the entries $\boldsymbol{t}$ given as the list of their polynomial coefficients, the remainder r_j and the quantities $\sigma_{\boldsymbol{b}_j}$ and $\delta_{\boldsymbol{b}_j} \cdot \delta(\boldsymbol{t})$ for $j < i$. We name ρ^i, $\boldsymbol{\alpha}^i = (\alpha_1^i, ..., \alpha_n^i)$ and θ^i the outputs of ξ^i corresponding to r_i, $\sigma_{\boldsymbol{b}_i}$ and $\delta_{\boldsymbol{b}_i} \cdot \delta(\boldsymbol{t})$.

Definition 6.19. *Let us initiate the induction for $-n < i \leqslant 0$ and $\bar{\imath} := i + n$:*

- *if $\deg_{X_{\bar{\imath}}}(t_{\bar{\imath}}) = 1$, then we define $\xi^i := (-r_{\bar{\imath}}, 0, 1)$ where $r_{\bar{\imath}} := t_{\bar{\imath}} - X_{\bar{\imath}}$. The output ρ^i points to $-r_{\bar{\imath}}$, α_m^i points to 0 if $m \neq \bar{\imath}$ or 1 otherwise and θ^i points to 0;*
- *if $\deg_{X_{\bar{\imath}}}(t_{\bar{\imath}}) > 1$, then we define $\xi^i := (X_{\bar{\imath}}, 0)$. The output ρ^i points to $X_{\bar{\imath}}$, θ^i and α_m^i points to 0 for any $1 \leqslant m \leqslant n$.*

Now recursively for $0 < i \leqslant L$, depending on the operation type of Γ_i:

- *if $\Gamma_i = (P^c)$ with $P \in A$ reduced modulo $\boldsymbol{t}_0$, then we define $\xi^i := (P, 0)$. The output ρ^i points to P and α_m^i points to 0 for any $1 \leqslant m \leqslant n$;*
- *if $\Gamma_i = (+; u, v)$, then we build ξ^i on top of ξ^u and ξ^v in such a manner that one has $\rho^i := \rho^u + \rho^v$, $\boldsymbol{\alpha}^i := \boldsymbol{\alpha}^u + \boldsymbol{\alpha}^v$ and $\theta^i := \theta^u + \theta^v$;*
- *if $\Gamma_i = (-; u, v)$, then we build ξ^i on top of ξ^u and ξ^v in such a manner that one has $\rho^i := \rho^u - \rho^v$, $\boldsymbol{\alpha}^i := \boldsymbol{\alpha}^u - \boldsymbol{\alpha}^v$ and $\theta^i := \theta^u - \theta^v$;*
- *if $\Gamma_i = (*; u, v)$, we define ξ^i accordingly to formulas (6.12,6.13,6.14). First, we compute $s, \boldsymbol{q} := \texttt{Rem\_quo\_triangular}(\rho^u(\boldsymbol{t})\,\rho^v(\boldsymbol{t}), \boldsymbol{t}, \boldsymbol{t}_0)$. Then $\rho^i := s$, $\boldsymbol{\alpha}^i$ is defined by*

$$\sigma(\boldsymbol{q}) + (\rho^u(\boldsymbol{t}) + \boldsymbol{\alpha}^u(\boldsymbol{t}) \cdot \boldsymbol{t}) \times \boldsymbol{\alpha}^v(\boldsymbol{t}) + \rho^v(\boldsymbol{t}) \times \boldsymbol{\alpha}^u(\boldsymbol{t})$$

and θ^i is defined by

$$\delta(\boldsymbol{q}) \cdot \delta(\boldsymbol{t}) + \delta(\rho^v)\,(\Theta^u) + (\rho^v)_0\,(\theta^u) + (\delta(\rho^u) + \Theta^u)\,(\Theta^v) + (\rho^u)_0\,(\theta^v)$$

where $\Theta^u := \boldsymbol{\alpha}^u \cdot \delta(\boldsymbol{t}) + p \times \theta^u$ and the same for Θ^v.

Finally, we set $\xi = \xi^L$.

Shifted algorithm Similarly to Section 6.4.1, we prove that Lemma 6.18 gives rise to a shifted algorithm to compute $\boldsymbol{t}$. For that matter, we will compute the shift index of the computation of $p^2\,(\delta_B \cdot \delta(\boldsymbol{t}))$ and prove that it is positive.

Lemma 6.20. *For any $-n < i \leqslant L$, one has*

$$\mathrm{sh}(\boldsymbol{t} \mapsto \rho^i(\boldsymbol{t})) \geqslant 0, \quad \mathrm{sh}(\boldsymbol{t} \mapsto \boldsymbol{\alpha}^i(\boldsymbol{t})) \geqslant 0, \quad \mathrm{sh}(\boldsymbol{t} \mapsto \theta^i(\boldsymbol{t})) \geqslant -1.$$

Proof. We proceed recursively on i for $-n < i \leqslant L$.

We initialize the induction for any $-n < i \leqslant 0$. One has

$$\mathrm{sh}(\boldsymbol{t} \mapsto \boldsymbol{\alpha}^i(\boldsymbol{t})) = \mathrm{sh}(\boldsymbol{t} \mapsto \theta^i(\boldsymbol{t})) = +\infty, \quad \mathrm{sh}(\boldsymbol{t} \mapsto \rho^i(\boldsymbol{t})) = \begin{cases} +\infty & \text{if } \deg_{X_{\bar{\imath}}}(t_{\bar{\imath}}) > 1 \\ 0 & \text{otherwise} \end{cases}.$$

Now recursively for $0 < i \leqslant L$, depending on the type of the i-th operation of Γ:

- if $\Gamma_i = (P^c)$ with $P \in A$ reduced modulo $\boldsymbol{t}_0$, one has
$$\mathrm{sh}(\boldsymbol{t} \mapsto \rho^i(\boldsymbol{t})) = \mathrm{sh}(\boldsymbol{t} \mapsto \boldsymbol{\alpha}^i(\boldsymbol{t})) = \mathrm{sh}(\boldsymbol{t} \mapsto \theta^i(\boldsymbol{t})) = +\infty.$$
- if $\Gamma_i = (\omega; u, v)$ with $\omega \in \{+, -, *\}$ then we proceed as follows. The s.l.p. ξ^i uses only additions, subtractions, multiplications, shifts $p \times \_$ by p, and calls to `Rem_triangular` and `Rem_quo_triangular`. These operations preserve a non-negative shift index, so
$$\mathrm{sh}(\boldsymbol{t} \mapsto \rho^i(\boldsymbol{t})) \geqslant 0, \quad \mathrm{sh}(\boldsymbol{t} \mapsto \boldsymbol{\alpha}^i(\boldsymbol{t})) \geqslant 0.$$
Now θ^i is an arithmetic expression in $\delta(\boldsymbol{q})$, $\delta(\boldsymbol{t})$, $(\rho^u)_0$, $\delta(\rho^u)$, $\boldsymbol{\alpha}^u$, θ^u, $p \times \theta^u$ and the same for v. All this quantities have a shift index greater or equal to -1 and so it is for θ^i. □

Since the s.l.p. ξ computes on the entries $\boldsymbol{t}$ the p-adic vector $\delta_B \cdot \delta(\boldsymbol{t})$, we can build an s.l.p. $\boldsymbol{\Delta}$ on top of ξ such that

$$\boldsymbol{\Delta}: \boldsymbol{t} \mapsto \boldsymbol{t}_0 + [\sigma_B^{-1}\,(\boldsymbol{f} - p^2 \times (\delta_B \cdot \delta(\boldsymbol{t}))) \text{ rem } \boldsymbol{t}_0].$$

The resolution of the linear system in $\boldsymbol{\Delta}$ is done by the relaxed algorithm of Chapter 3, Section 3.3.3.

Lemma 6.21. *The s.l.p. $\boldsymbol{\Delta}$ is a shifted algorithm of which $\boldsymbol{t}$ is a fixed point when the computations are done in the algebra $R_p[X_1, ..., X_n]$.*

Proof. The s.l.p. $\boldsymbol{\Delta}$ compute $\boldsymbol{t}$ on the entries $\boldsymbol{t}$ in the algebra $R_p[X_1, ..., X_n]$ thanks to Lemma 6.18 and because the formulas that define ξ in Definition 6.19 match formulas (6.9) to (6.14).

A direct consequence of Lemma 6.20 is that

$$\mathrm{sh}(\boldsymbol{t} \mapsto p^2 \times (\delta_B \cdot \delta(\boldsymbol{t}))) \geqslant (2 + \mathrm{sh}(\boldsymbol{t} \mapsto \delta_B \cdot \delta(\boldsymbol{t}))) \geqslant 1.$$

Since $\boldsymbol{f} \operatorname{rem} \boldsymbol{t}_0$ and σ_B depend only on $\boldsymbol{t}_0$, and since the resolution of the linear system does not impact the shift, we have proved $\boldsymbol{\Delta}$ has a positive shift index. □

Proof. (of Theorem 6.3) By Lemma 6.21, the triangular set $\boldsymbol{t}$ is a fixed point of the shifted algorithm $\boldsymbol{\Delta}$. Proposition 2.17 shows that we can compute $\boldsymbol{t}$ in time the number of operations in $\boldsymbol{\Delta}$. Let us count the number of operations in $\boldsymbol{\Delta}$.

Cost of $\boldsymbol{\sigma_B}$. We start by evaluating the maximal length of the entries of σ_B. We look at the effect of one operation of the s.l.p. of $\boldsymbol{f}$ on $\sigma_{\boldsymbol{b}_i}$. The worst case happens for the multiplication $*$. In this case, recall from Formula 6.13 that

$$\sigma_{\boldsymbol{b}_i} = \boldsymbol{q}_0 + (r_j)_0 \times \sigma_{\boldsymbol{b}_k} + (r_k)_0 \times \sigma_{\boldsymbol{b}_j}.$$

The multiplication modulo a triangular set increase the length of the p-adics by a factor $\tilde{\mathcal{O}}(d_1 \cdots d_n)$ (see [Lan91, Theorem 3]), so that

$$\lambda(\sigma_{\boldsymbol{b}_i}) \leqslant \max(\lambda(\sigma_{\boldsymbol{b}_k}), \lambda(\sigma_{\boldsymbol{b}_j})) + 2\,\tilde{\mathcal{O}}(d_1 \cdots d_n) + 2\,.$$

There are L operations and consequently $\lambda := \lambda(\sigma_B) = \tilde{\mathcal{O}}(L\,d_1 \cdots d_n)$.

The multiplication of two p-adics of length 1 and λ costs $\mathcal{O}(\min(\lambda, N))$ and so does the addition of two p-adics of length λ. Put it all together for a total cost of $\mathcal{O}(n\,L \min(\lambda, N)\,\mathsf{Rem}(d_1, ..., d_n))$.

Computation of $\boldsymbol{f}$ rem $\boldsymbol{t_0}$ in time $\mathcal{O}(L\,\mathsf{Rem}(d_1, ..., d_n)\,\mathsf{R}(N))$.

Computation of $\boldsymbol{r_i}$ and $\boldsymbol{\delta_{b_i} \cdot \delta(t)}$ at each step of the computation of $\boldsymbol{f}$. We focus on the operations of Γ which are multiplications because they induce the more operations in $\boldsymbol{\Delta}$. The remainder s and quotients $\boldsymbol{q}$ of $r_j\,r_k$ require a call to Algorithm `Rem_quo_triangular`. Then $\delta_B \cdot \delta(\boldsymbol{t})$ use an inner product $\delta(\boldsymbol{q}) \cdot \delta(\boldsymbol{t})$ and $\mathcal{O}(1)$ multiplications in $R_p[X_1, ..., X_n]/\langle \boldsymbol{t}_0 \rangle$. The inner product costs less than a call to `Rem_quo_triangular`, since this latter algorithm does an inner product $\boldsymbol{q} \cdot \boldsymbol{t}$. Summing up, the total cost is $\mathcal{O}(L\,\mathsf{R}(N)\,(\mathsf{RemQuo}(d_1, ..., d_n) + \mathsf{Rem}(d_1, ..., d_n)))$, that is $\mathcal{O}(e\,L\,\mathsf{R}(N)\,\mathsf{Rem}(d_1, ..., d_n))$.

Resolution of the linear system in $\boldsymbol{\sigma_B}$: Since the matrix σ_B has finite length λ, Proposition 3.6 tells us that the cost of solving the linear system is

$$\mathcal{O}([N\,\mathsf{MMR}(n, 1, \lambda)/\lambda + n^{\Omega}]\,\mathsf{Rem}(d_1, ..., d_n)),$$

that is $[N n^{2+o(1)} \log(\lambda)^{\mathcal{O}(1)} + \mathcal{O}(n^{\Omega})]\,\mathsf{Rem}(d_1, ..., d_n)$. □

6.5 Implementation in `Mathemagix`

The computer algebra software MATHEMAGIX [HLM+02] provides a C++ library named ALGEBRAMIX implementing relaxed power series or p-adic numbers [Hoe02, Hoe07, BHL11, BL12]. We implemented the lifting of univariate representations over the power series ring $\mathbb{F}_p[[X]]$ for both the off-line and the relaxed approach. The implementations over the p-adic integers $\mathbb{Z}_p$ are still in progress. Although they work, they still require some efforts to be competitive.

Our implementation is available in the files whose name begins with `lift_` in the C++ library GREGORIX of MATHEMAGIX. We mention that our on-line algorithm is currently connected to KRONECKER inside MATHEMAGIX with the help of G. LECERF.

We now give some implementation details:

- for the multiplication of polynomials of power series in $(\mathbb{F}_p[[X]])[Y]$, we first converted them as power series of polynomial in $(\mathbb{F}_p[Y])[[X]]$. Then the relaxed multiplication algorithm reduces to multiplications of finite precision power series of polynomials, that is polynomials of polynomials in $(\mathbb{F}_p[Y])[X]$. We classically used a Kronecker substitution to reduce these products to multiplications of univariate polynomials in $\mathbb{F}_p[Z]$;
- since remainder and quotients modulo $\boldsymbol{t}$ or $\boldsymbol{t}_0$ are often used, we stored the precomputation of the inverse of these elements in Algorithms `Rem_triangular` and `Rem_quo_triangular`;
- for the matrix multiplication modulo Q inside the off-line Algorithm `Global_Newton_step_univariate`, we delayed the reductions until after the matrix multiplication to reduce their numbers;

- as we mentioned before, Algorithms `Rem_triangular` and `Rem_quo_triangular` greatly simply with univariate representations since we just have to compute quotient and remainder of univariate polynomials.

6.5.1 Benchmarks

We report the timings of our implementation in milliseconds. Timings are measured using one core of an Intel Xeon X5650 at 2.67 GHz running Linux 64 bits, Gmp 5.0.2 [G+91] and setting $p = 16411$ a 15-bit prime number.

We start by giving some comparison of timings between the relaxed and zealous product in $(\mathbb{F}_p[[X]])[Y]$ depending on the degree in Y and the precision N of power series.

	Degree 32 in Y		Degree 64 in Y		Degree 128 in Y		Degree 256 in Y	
N	zealous	relaxed	zealous	relaxed	zealous	relaxed	zealous	relaxed
8	0	0	0	1	1	1	2	4
16	0	1	0	3	1	5	3	11
32	0	3	1	7	2	14	6	34
64	1	8	3	18	6	41	12	100
128	3	21	6	49	12	110	30	270
256	6	56	12	130	29	300	70	700
512	12	150	30	340	71	790	170	1800
1024	29	370	71	860	170	2000	340	4500
2048	72	920	170	2100	350	4800	750	11000

Table 6.1. Timings of zealous and relaxed multiplication in $(\mathbb{F}_p[[X]])[Y]$

We observe that the ratio of the timings between the relaxed and zealous algorithms grows as $\log(N)$. As a future work, we will apply the new relaxed multiplications of [Hoe07], generalized to more general ring in [Hoe12], to keep the ratio $\mathsf{R}(N)/\mathsf{I}(N)$ smaller.

To be fair, we mention that our zealous lifting implementation could also be improved. Especially the multiplication of matrices with polynomial entries could have benefit from an interpolation/evaluation scheme, especially since our matrices have reasonable size $n \times n$ and big entries in $(\mathbb{F}_p[[X]])[Y]$.

We tried our algorithm on two family of examples. The Katsura polynomials systems comes from a problem of magnetism in physics [Kat90]. The system Katsura-n has $n+1$ unknowns $X_0, \ldots, X_n$ and $n+1$ equations:

$$\text{for } 0 \leqslant m < n, \quad \sum_{\ell=-n}^{n} X_{|\ell|}\, X_{|m-\ell|} = X_m$$

and $X_0 + 2\sum_{\ell=1}^{n} X_\ell = 1$.

The other family of polynomial system MulLinForm-n has n unknowns and n equations of the form

$$(\lambda_1 X_1 + \cdots + \lambda_n X_n)\,(\mu_1 X_1 + \cdots + \mu_n X_n) = \alpha$$

where the λ_i, μ_i and α are random coefficients in $\mathbb{F}_p$.

We indicate with a bold font the theoretical bound for the precision of power series required in the KRONECKER algorithm.

	KATSURA-3		KATSURA-4		KATSURA-5		KATSURA-6	
N	zealous	relaxed	zealous	relaxed	zealous	relaxed	zealous	relaxed
2	21	7	75	20	250	58	780	170
4	31	11	106	29	350	78	1100	220
8	**49**	**18**	170	48	550	130	1700	360
16	82	36	**290**	**92**	940	240	1900	700
32	140	74	510	200	**1700**	**530**	5200	1500
64	260	160	970	440	3300	1200	**10000**	**3600**
128	510	360	1900	1000	6600	2800	21000	8600
256	1000	820	4000	2400				
512	2200	1900	8600	5500				

Table 6.2. Timings of zealous/relaxed lifting of univariate representations for KATSURA-n.

	MULLINFORM-4		MULLINFORM-5		MULLINFORM-6	
N	zealous	relaxed	zealous	relaxed	zealous	relaxed
2	44	16	160	45	520	130
4	64	23	230	63	720	180
8	96	38	340	100	1000	300
16	**150**	**69**	520	180	1700	540
32	230	140	**850**	**380**	2900	1000
64	370	180	1400	780	**5200**	**2300**
128	670	580	2600	1600	9500	4800

Table 6.3. Zealous/relaxed lifting timings of univariate representations of MULLINFORM-n.

6.5.2 Conclusion

As a conclusion, we remark that a relaxed approach has generated new algorithms for the lifting of triangular sets. Our hope was to save the cost of linear algebra in the dominant part of the complexity when the precision N tends to the infinity. Besides, previous experiences, with e.g. the relaxed lifting of regular roots, showed that we could expect to do less multiplications in the relaxed model than in the zealous one. Therefore, whenever the precision N gives a measured ratio of complexity between relaxed and zealous multiplication, we can expect better timings from the relaxed approach.

In view of our hopes, we are not completely satisfied with the lifting of general triangular sets, for it does more multiplications when $n\,L \geqslant n^\omega$. On the contrary, the lifting of univariate representations always improve the asymptotic number of multiplications.