# Power series solutions of $(q)$-differential equations

This chapter is published in the homonym paper with A. Bostan, M. Chowdhurry, B. Salvy and É. Schost in the proceedings of *ISSAC'12* [BCL+12].

We provide algorithms computing power series solutions of a large class of differential or $q$-differential equations or systems. Their number of arithmetic operations grows linearly with the precision, up to logarithmic terms.

## 4.1 Introduction

Truncated power series are a fundamental class of objects of computer algebra. Fast algorithms are known for a large number of operations starting from addition, derivative, integral and product and extending to quotient, powering and several more. The main open problem is composition: given two power series $f$ and $g$, with $g(0)=0$, known mod $x^N$, the best known algorithm computing $f(g) \bmod x^N$ has a cost which is roughly that of $\sqrt{N}$ products in precision $N$; it is not known whether quasi-linear (i.e., linear up to logarithmic factors) complexity is possible in general. Better results are known over finite fields [Ber98, KU11] or when more information on $f$ or $g$ is available. Quasi-linear complexity has been reached when $g$ is a polynomial [BK78], an algebraic series [Hoe02], or belongs to a large class containing for instance the expansions of $\exp(x)-1$ and $\log(1+x)$ [BSS08].

One motivation for this work is to deal with the case when $f$ is the solution of a given differential equation. Using the chain rule, a differential equation for $f(g)$ can be derived, with coefficients that are power series. We focus on the case when this equation is linear, since in many cases linearization is possible [BCO+07]. When the order $n$ of the equation is larger than 1, we use the classical technique of converting it into a first-order equation over vectors, so we consider equations of the form

$$x^k\,\delta(F) = A\,F + C, \tag{4.1}$$

where $A$ is an $n \times n$ matrix over the power series ring $\mathbb{k}[[x]]$ ($\mathbb{k}$ being the field of coefficients), $C$ and the unknown $F$ are size $n$ vectors over $\mathbb{k}[[x]]$ and for the moment $\delta$ denotes the differential operator $d/dx$. The exponent $k$ in (4.1) is a non-negative integer that plays a key role for this equation.

By *solving* such equations, we mean computing a vector $F$ of power series such that (4.1) holds modulo $x^N$. For this, we need only to compute $F$ polynomial of degree less than $N+1$ (when $k=0$) or $N$ (otherwise). Conversely, when (4.1) has a power series solution, its first $N$ coefficients can be computed by solving (4.1) modulo $x^N$ (when $k \neq 0$) or $x^{N-1}$ (otherwise).

If $k=0$ and the field $\mathbb{k}$ has characteristic 0, then a formal Cauchy theorem holds and (4.1) has a unique vector of power series solution for any given initial condition. In this situation, algorithms are known that compute the first $N$ coefficients of the solution in quasi-linear complexity [BCO+07]. Also the relaxed algorithm of [Hoe02] applies to this case. In this article, we extend the results of [BCO+07] in three directions:

**Singularities** We deal with the case when $k$ is positive. A typical example is the computation of the composition $F = f(g)$ when $f$ is Gauss' ${}_2F_1$ hypergeometric series. Although $f$ is a very nice power series

$$f = 1 + \frac{a\,b}{c}\,x + \frac{a\,(a+1)\,b\,(b+1)}{c\,(c+1)}\,\frac{x^2}{2!} + \cdots,$$

we exploit this structure indirectly only. We start from the differential equation

$$x\,(x-1)\,f'' + (x\,(a+b+1) - c)\,f' + a\,b\,f = 0 \tag{4.2}$$

and build up and solve the more complicated

$$\frac{g\,(g-1)}{g'^2}\,F'' + \frac{g'^2\,(g\,(a+b+1) - c) + (g - g^2)\,g''}{g'^3}\,F' + a\,b\,F = 0$$

in the unknown $F$, $g$ being given, with $g(0)=0$. Equation (4.2) has a leading term that is divisible by $x$ so that Cauchy's theorem does not apply and indeed there does not exist a basis of two power series solutions. This behavior is inherited by the equation for $F$, so that the techniques of [BCO+07] do not apply — this example is actually already mentioned in [BK78], but the issue with the singularity at 0 was not addressed there. We show in this article how to overcome this singular behavior and obtain a quasi-linear complexity.

**Positive characteristic** Even when $k = 0$, Cauchy's theorem does not hold in positive characteristic and Equation (4.1) may fail to have a power series solution (a simple example is $F' = F$). However, such an equation may have a solution modulo $x^N$. Efficient algorithms for finding such a solution are useful in conjunction with the Chinese remainder theorem. Other motivations for considering algorithms that work in positive characteristic come from applications in number-theory based cryptology or in combinatorics [BMSS08, BSS08, BS09].

Our objectives in this respect are to overcome the lack of a Cauchy theorem, or of a formal theory of singular equations, by giving conditions that ensure the existence of solutions at the required precisions. More could probably be said regarding the $p$-adic properties of solutions of such equations (as in [BGVPS05, LS08]), but this is not the purpose of this chapter.

**Functional Equations** The similarity between algorithms for linear differential equations and for linear difference equations is nowadays familiar to computer algebraists. We thus use the standard technique of introducing $\sigma\colon \mathbb{k}[[x]] \rightarrow \mathbb{k}[[x]]$ a unitary ring morphism and letting $\delta\colon \mathbb{k}[[x]] \rightarrow \mathbb{k}[[x]]$ denote a $\sigma$-derivation, in the sense that $\delta$ is $\mathbb{k}$-linear and that for all $f, g$ in $\mathbb{k}[[x]]$, we have

$$\delta(f g) = f \delta(g) + \delta(f)\, \sigma(g).$$

These definitions, and the above equality, carry over to matrices over $\mathbb{k}[[x]]$. Thus, our goal is to solve the following generalization of (4.1):

$$x^k \delta(F) = A\, \sigma(F) + C. \tag{4.3}$$

As above, we are interested in computing a vector $F$ of power series such that (4.3) holds $\bmod x^N$.

Concerning on-line algorithms, the techniques of [Hoe02] already apply to the positive characteristic case. At the beginning of my thesis, the tools to adapt relaxed algorithms for singular equations did not exist. Our method to deal with singular equations was discovered independently at the same time by [Hoe11]. This paper deals with more general recursive power series defined by algebraic, differential equations or a combination thereof. However, this paper does not consider the case of $(q)$-differential equations and works under more restrictive hypotheses.

One motivation for the generalization to functional equations comes from coding theory. The list-decoding of the folded Reed-Solomon codes [GR08] leads to an equation $Q\,(x, f(x), f\,(q\,x)) = 0$ where $Q$ is a known polynomial. A linearized version of this is of the form (4.3), with $\sigma\colon \phi(x) \mapsto \phi(q\,x)$. In cases of interest we have $k = 1$, and we work over a finite field.

In view of these applications, we restrict ourselves to the following setting:

$$\delta(x) = 1, \qquad \sigma\colon x \mapsto q\,x,$$

for some $q \in \mathbb{k} \setminus \{0\}$. Then, there are only two possibilities:

- $q = 1$ and $\delta\colon f \mapsto f'$ (*differential* case);
- $q \neq 1$ and $\delta\colon f \mapsto \frac{f(qx) - f(x)}{x\,(q-1)}$ (*q-differential* case).

As a consequence, $\delta(1) = 0$ and for all $i \geq 0$, we have

$$\delta(x^i) = \gamma_i\, x^{i-1} \text{ with } \gamma_0 = 0 \text{ and } \gamma_i = 1 + q + \cdots + q^{i-1}\ (i > 0).$$

By linearity, given $f = \sum_{i \geq 0} f_i\, x^i \in \mathbb{k}[[x]]$,

$$\delta(f) = \sum_{i \geq 1} \gamma_i\, f_i\, x^{i-1}$$

can be computed $\bmod x^N$ in $\mathcal{O}(N)$ operations, as can $\sigma(f)$. Conversely, assuming that $\gamma_1, \ldots, \gamma_n$ are all non-zero in $\mathbb{k}$, given $f$ of degree at most $n-1$ in $\mathbb{k}[x]$, there exists a unique $g$ of degree at most $n$ such that $\delta(g) = f$ and $g_0 = 0$; it is given by $g = \sum_{0 \leq i \leq n-1} f_i / \gamma_{i+1} x^{i+1}$ and can be computed in $\mathcal{O}(N)$ operations. We denote it by $g = \int_q f$. In particular, our condition excludes cases where $q$ is a root of unity of low order.

**Notation and complexity model** We adopt the convention that uppercase letters denote matrices or vectors while lowercase letters denote scalars. The set of $n \times m$ matrices over a ring $R$ is denoted $\mathcal{M}_{n,m}(R)$; when $n = m$, we write $\mathcal{M}_n(R)$. If $f$ is in $\mathbb{k}[[x]]$, its degree $i$ coefficient is written $f_i$; this carries over to matrices. The identity matrix is written $\mathsf{Id}$ (the size will be obvious from the context). To avoid any confusion, the entry $(i, j)$ of a matrix $M$ is denoted $M^{(i,j)}$.

Our algorithms are sometimes stated with input in $\mathbb{k}[[x]]$, but it is to be understood that we are given only truncations of $A$ and $C$ and only their first $N$ coefficients will be used.

The costs of our algorithms are measured by the number of arithmetic operations in $\mathbb{k}$ they use. We let $\mathsf{M}: \mathbb{N} \to \mathbb{N}$ be such that for any ring $R$, polynomials of degree less than $n$ in $R[x]$ can be multiplied in $\mathsf{M}(n)$ arithmetic operations in $R$. We assume that $\mathsf{M}(n)$ satisfies the usual assumptions of [GG03, §8.3]; using Fast Fourier Transform, $\mathsf{M}(n)$ can be taken in $\mathcal{O}(n \log(n) \operatorname{loglog}(n))$ [CK91, SS71]. We note $\omega \in (2, 3]$ a constant such that two matrices in $\mathcal{M}_n(R)$ can be multiplied in $\mathcal{O}(n^\omega)$ arithmetic operations in $R$. The current best bound is $\omega < 2.3727$ ([VW11] following [CW90, Sto10]).

Our algorithms rely on linear algebra techniques; in particular, we have to solve several systems of non-homogeneous linear equations. For $U$ in $\mathcal{M}_n(\mathbb{k})$ and $V$ in $\mathcal{M}_{n,1}(\mathbb{k})$, we denote by $\mathsf{LinSolve}(U\, X = V)$ a procedure that returns $\perp$ if there is no solution, or a pair $F, K$, where $F$ is in $\mathcal{M}_{n,1}(\mathbb{k})$ and satisfies $U F = V$, and $K \in \mathcal{M}_{n,t}(\mathbb{k})$, for some $t \leq n$, generates the nullspace of $U$. This can be done in time $\mathcal{O}(n^\omega)$. In the pseudo-code, we adopt the convention that if a subroutine returns $\perp$, the caller returns $\perp$ too (so we do not explicitly handle this as a special case).

**Main results** Equation (4.3) is linear, non-homogeneous in the coefficients of $F$, so our output follows the convention mentioned above. We call *generators* of the solution space of Eq. (4.3) at precision $N$ either $\perp$ (if no solution exists) or a pair $F, K$ where $F \in \mathcal{M}_{n,1}(\mathbb{k}[x])$ and $K \in \mathcal{M}_{n,t}(\mathbb{k}[x])$ with $t \leq n\, N$, such that for $G \in \mathcal{M}_{n,1}(\mathbb{k}[x])$, with $\deg(G) < N$, $x^k \delta(G) = A\, \sigma(G) + C \bmod x^N$ if and only if $G$ can be written $G = F + K\, B$ for some $B \in \mathcal{M}_{t,1}(\mathbb{k})$.

Seeing Eq. (4.3) as a linear system, one can obtain such an output using linear algebra in dimension $n\, N$. While this solution always works, we give algorithms of much better complexity, under some assumptions related to the spectrum $\operatorname{Spec} A_0$ of the constant coefficient $A_0$ of $A$. First, we simplify our problem: we consider the case $k = 0$ as a special case of the case $k = 1$. Indeed, the equation $\delta(F) = A\, \sigma(F) + C \bmod x^N$ is equivalent to $x\, \delta(F) = P \sigma(F) + Q \bmod x^{N+1}$, with $P = xA$ and $Q = xC$. Thus, in our results, we only distinguish the cases $k = 1$ and $k > 1$.

**Definition 4.1.** *The matrix $A_0$ has* good spectrum at precision $N$ *when one of the following holds:*

- $k = 1$ *and* $\mathrm{Spec}A_0 \cap (q^i\,\mathrm{Spec}A_0 - \gamma_i) = \emptyset$ *for* $1 \le i < N$
- $k > 1$, $A_0$ *is invertible and*
  - $q = 1$, $\gamma_1, \ldots, \gamma_{N-k}$ *are non-zero,* $|\mathrm{Spec}A_0| = n$ *and* $\mathrm{Spec}A_0 \subset \mathbb{k}$*;*
  - $q \neq 1$ *and* $\mathrm{Spec}A_0 \cap q^i\,\mathrm{Spec}A_0 = \emptyset$ *for* $1 \le i < N$.

In the classical case when $\mathbb{k}$ has characteristic 0 and $q = 1$, if $k = 1$, $A_0$ has good spectrum when no two eigenvalues of $A_0$ differ by a non-zero integer (this is e.g. the case when $A_0 = 0$, which is essentially the situation of Cauchy's theorem; this is also the case in our ${}_2F_1$ example whenever $c\,\mathrm{val}(g)$ is not an integer, since $\mathrm{Spec}A_0 = \{0, \mathrm{val}(g)\,(1 - c) - 1\}$).

These conditions could be slightly relaxed, using gauge transformations (see [Bal00, Ch. 2] and [BBP10, BP99]). Also, for $k > 1$ and $q = 1$, we could drop the assumption that the eigenvalues are in $\mathbb{k}$, by replacing $\mathbb{k}$ by a suitable finite extension, but then our complexity estimates would only hold in terms of number of operations in this extension.

As in the non-singular case [BCO+07], we develop two approaches. The first one is a divide-and-conquer method. The problem is first solved at precision $N/2$ and then the computation at precision $N$ is completed by solving another problem of the same type at precision $N/2$. This leads us to the following result, proved in Section 4.2 (see also that section for comparison to previous work). In all our cost estimates, we consider $k$ constant, so it is absorbed in the big-Os.

**Theorem 4.2.** *Algorithm 4.2 computes generators of the solution space of Eq. (4.3) at precision $N$ by a divide-and-conquer approach. Assuming $A_0$ has good spectrum at precision $N$, it performs in time $\mathcal{O}(n^\omega\,\mathsf{M}(N)\log(N))$. When either $k > 1$ or $k = 1$ and $q^i A_0 - \gamma_i\,\mathsf{Id}$ is invertible for $0 \le i < N$, this drops to $\mathcal{O}(n^2\,\mathsf{M}(N)\log(N) + n^\omega N)$.*

Our second algorithm behaves better with respect to $N$, with cost in $\mathcal{O}(\mathsf{M}(N))$ only, but it always involves polynomial matrix multiplications. Since in many cases the divide-and-conquer approach avoids these multiplications, the second algorithm becomes preferable for rather large precisions.

In the differential case, when $k = 0$ and the characteristic is 0, the algorithms in [BCO+07, BK78] compute an invertible matrix of power series solution of the homogeneous equation by a Newton iteration and then recover the solution using variation of the constant. In the more general context we are considering here, such a matrix does not exist. However, it turns out that an associated equation that can be derived from (4.3) admits such a solution. Section 4.3 describes a variant of Newton's iteration to solve it and obtains the following.

**Theorem 4.3.** *Assuming $A_0$ has good spectrum at precision $N$, one can compute generators of the solution space of Eq. (4.3) at precision $N$ by a Newton-like iteration in time $\mathcal{O}(n^\omega\,\mathsf{M}(N) + n^\omega \log(n)\,N)$.*

To the best of our knowledge, this is the first time such a low complexity is reached for this problem. Without the good spectrum assumption, however, we cannot guarantee that this algorithm succeeds, let alone control its complexity.

## 4.2 Divide-and-Conquer

The classical approach to solving (4.3) is to proceed term-by-term by coefficient extraction. Indeed, we can rewrite the coefficient of degree $i$ in this equation as

$$R_i F_i = \Delta_i, \tag{4.4}$$

where $\Delta_i$ is a vector that can be computed from $A$, $C$ and all previous $F_j$ (and whose actual expression depends on $k$), and $R_i$ is as follows:

$$\begin{cases} R_i = (q^i A_0 - \gamma_i \,\mathsf{Id}) & \text{if } k = 1 \\ R_i = q^i A_0 & \text{if } k > 1. \end{cases}$$

Ideally, we wish that each such system determines $F_i$ uniquely that is, that $R_i$ be a unit. For $k = 1$, this is the case when $i$ is not a root of the *indicial equation* $\det(q^i A_0 - \gamma_i \,\mathsf{Id}) = 0$. For $k > 1$, either this is the case for all $i$ (when $A_0$ is invertible) or for no $i$. In any case, we let $\mathcal{R}$ be the set of indices $i \in \{0, \ldots, N-1\}$ such that $\det(R_i) = 0$; we write $\mathcal{R} = \{j_1 < \ldots < j_r\}$, so that $r = |\mathcal{R}|$.

Even when $\mathcal{R}$ is empty, so the solution is unique, this approach takes quadratic time in $N$, as computing each individual $\Delta_i$ takes linear time in $i$. To achieve quasi-linear time, we split the resolution of Eq. (4.3) mod $x^N$ into two half-sized instances of the problem; at the leaves of the recursion tree, we end up having to solve the same Eq. (4.4).

When $\mathcal{R}$ is empty, the algorithm is simple to state (and the cost analysis simplifies; see the comments at the end of this section). Otherwise, technicalities arise. We treat the cases $i \in \mathcal{R}$ separately, by adding placeholder parameters for all corresponding coefficients of $F$ (this idea is already in [BBP10, BP99]; the algorithms in these references use a finer classification when $k > 1$, by means of a suitable extension of the notion of indicial polynomial, but take quadratic time in $N$).

Let $\mathbf{f}_{1,1}, \ldots, \mathbf{f}_{n,r}$ be $n\, r$ new indeterminates over $\mathbb{k}$ (below, all boldface letters denote expressions involving these formal parameters). For $\rho = 1, \ldots, r$, we define the vector $\mathbf{F}_{j_\rho}$ with entries $\mathbf{f}_{1,\rho}, \ldots, \mathbf{f}_{n,\rho}$ and we denote by $\mathscr{L}$ the set of all vectors

$$\mathbf{F} = \varphi_0 + \varphi_1 \mathbf{F}_{j_1} + \cdots + \varphi_r \mathbf{F}_{j_r},$$

with $\varphi_0$ in $\mathcal{M}_{n,1}(\mathbb{k}[x])$ and each $\varphi_\ell$ in $\mathcal{M}_n(\mathbb{k}[x])$ for $1 \leq \ell \leq r$. We also define $\mathscr{L}_i$ the subspace of vectors of the form

$$\mathbf{F} = \varphi_0 + \varphi_1 \mathbf{F}_{j_1} + \cdots + \varphi_{\mu(i)} \mathbf{F}_{\mu(i)},$$

where $\mu(i)$ is defined as the index of the largest element $j_\ell \in \mathcal{R}$ such that $j_\ell < i$; if no such element exist (for instance when $i = 0$), we let $\mu(i) = 0$. A *specialization* $S$: $\mathscr{L} \to \mathcal{M}_{n,1}(\Bbbk[x])$ is simply an evaluation map defined by $\mathbf{f}_{i,\ell} \mapsto f_{i,\ell}$ for all $i, \ell$, for some choice of $(f_{i,\ell})$ in $\Bbbk^{nr}$.

We extend $\delta$ and $\sigma$ to such vectors, by letting $\delta(\mathbf{f}_{i,\ell}) = 0$ and $\sigma(\mathbf{f}_{i,\ell}) = \mathbf{f}_{i,\ell}$ for all $i, \ell$, so that we have, for $\mathbf{F}$ in $\mathscr{L}$

$$\delta(\mathbf{F}) = \delta(\varphi_0) + \delta(\varphi_1)\,\mathbf{F}_{j_1} + \cdots + \delta(\varphi_r)\,\mathbf{F}_{j_r},$$

and similarly for $\sigma(\mathbf{F})$.

**Algorithm 4.1**
Recursive Divide-and-conquer $\mathsf{RDAC}(A, \mathbf{C}, i, N, k)$

```
Input:  $A \in \mathcal{M}_n(\Bbbk[[x]]), \mathbf{C} \in \mathscr{L}_i, i \in \mathbb{N}, N \in \mathbb{N} \setminus \{0\}, k \in \mathbb{N} \setminus \{0\}$
Output: $\mathbf{F} \in \mathscr{L}_{i+N}$

if $N = 1$
    if $(k = 1)$              then $R_i := q^i A_0 - \gamma_i \,\mathsf{Id}$   else $R_i := q^i A_0$
    if $(\det(R_i) = 0)$      then return $\mathbf{F}_i$             else return $-R_i^{-1}\mathbf{C}_0$
else
    $m := \lceil N/2 \rceil$
    $\mathbf{H} := \mathsf{RDAC}(A, \mathbf{C}, i, m, k)$
    $\mathbf{D} := (\mathbf{C} - x^k \delta(\mathbf{H}) + (q^i A - \gamma_i x^{k-1} \mathsf{Id})\, \sigma(\mathbf{H}))$ div $x^m$
    $\mathbf{K} := \mathsf{RDAC}(A, \mathbf{D}, i+m, N-m, k)$
    return $\mathbf{H} + x^m \mathbf{K}$
```

The main divide-and-conquer algorithm first computes $\mathbf{F}$ in $\mathscr{L}$, by simply skipping all equations corresponding to indices $i \in \mathcal{R}$; it is presented in Algorithm 4.2. In a second step, we resolve the indeterminacies by plain linear algebra. For $i \geq 0$, and $\mathbf{F}, \mathbf{C}$ in $\mathscr{L}$, we write

$$E(\mathbf{F}, \mathbf{C}, i) = x^k \delta(\mathbf{F}) - ((q^i A - \gamma_i x^{k-1} \mathsf{Id})\, \sigma(\mathbf{F}) + \mathbf{C}).$$

In particular, $E(\mathbf{F}, \mathbf{C}, 0)$ is a parameterized form of Eq. (4.3). The key to the divide-and-conquer approach is to write $\mathbf{H} = \mathbf{F} \bmod x^m$, $\mathbf{K} = \mathbf{F}$ div $x^m$ and $\mathbf{D} = (\mathbf{C} - E(\mathbf{H}, \mathbf{C}, i))$ div $x^m$. Using the equalities

$$x^k \delta(\mathbf{F}) = x^k \delta(\mathbf{H}) + x^{m+k} \delta(\mathbf{K}) + \gamma_m x^{m+k-1} \sigma(\mathbf{K})$$

and $\gamma_{i+m} = \gamma_m + q^m \gamma_i$, a quick computation shows that

$$E(\mathbf{F}, \mathbf{C}, i) = (E(\mathbf{H}, \mathbf{C}, i) \bmod x^m) + x^m E(\mathbf{K}, \mathbf{D}, i+m). \qquad (4.5)$$

**Lemma 4.4.** *Let $A$ be in $\mathcal{M}_n(\Bbbk[x])$ and $\mathbf{C}$ in $\mathscr{L}_i$, and let $\mathbf{F} = \mathsf{RDAC}(A, \mathbf{C}, i, M, k)$ with $i + M \leq N$. Then:*

1. *$\mathbf{F}$ is in $\mathscr{L}_{i+M}$;*

2. *for* $j \in \{0, \ldots, M-1\}$ *such that* $i + j \notin \mathcal{R}$, *the equality* $\operatorname{coeff}(E(\mathbf{F}, \mathbf{C}, i), x^j) = 0$ *holds;*

3. *if* $C$ *and* $F$ *in* $\mathcal{M}_{n,1}(\Bbbk[x])$ *with* $\deg F < M$ *are such that* $E(F, C, i) = 0 \bmod x^M$ *and there exists a specialization* $S$: $\mathscr{L}_i \to \mathcal{M}_{n,1}(\Bbbk[x])$ *such that* $C = S(\mathbf{C})$, *there exists a specialization* $S'$: $\mathscr{L}_{i+M} \to \mathcal{M}_{n,1}(\Bbbk[x])$ *which extends* $S$ *and such that* $F = S(\mathbf{F})$.

$\mathbf{F}$ *is computed in time* $\mathcal{O}((n^2 + r\, n^\omega)\, \mathsf{M}(M) \log(M) + n^\omega M)$.

**Proof.** The proof is by induction on $M$.

**Proof of 1.** For $M = 1$, we distinguish two cases. If $i \in \mathcal{R}$, say $i = j_\ell$, we return $\mathbf{F}_i = \mathbf{F}_{j_\ell}$. In this case, $\mu\,(i+1) = \ell$, so our claim holds. If $i \notin \mathcal{R}$, because $\mathbf{C}_0 \in \mathscr{L}_i$, the output is in $\mathscr{L}_i$ as well. This proves the case $M = 1$.

For $M > 1$, we assume the claim to hold for all ($i$, $M'$), with $M' < M$. By induction, $\mathbf{H} \in \mathscr{L}_{i+m}$ and $\mathbf{K} \in \mathscr{L}_{i+M}$. Thus, $\mathbf{D} \in \mathscr{L}_{i+m}$ and the conclusion follows.

**Proof of 2.** For $M = 1$, if $i \in \mathcal{R}$, the claim is trivially satisfied. Otherwise, we have to verify that the constant term of $E(\mathbf{F}, \mathbf{C}, i)$ is zero. In this case, the output $\mathbf{F}$ is reduced to its constant term $\mathbf{F}_0$, and the constant term of $E(\mathbf{F}, \mathbf{C}, i)$ is (up to sign) $R_i \mathbf{F}_0 + \mathbf{C}_0 = 0$, so we are done.

For $M > 1$, we assume that the claim holds for all $(i, M')$, with $M' < M$. Take $j$ in $\{0, \ldots, M-1\}$. If $j < m$, we have $\operatorname{coeff}(E(\mathbf{F}, \mathbf{C}, i), x^j) = \operatorname{coeff}(E(\mathbf{H}, \mathbf{C}, i), x^j)$; since $i + j \notin \mathcal{R}$, this coefficient is zero by assumption. If $m \leq j$, we have $\operatorname{coeff}(E(\mathbf{F}, \mathbf{C}, i), x^j) = \operatorname{coeff}(E(\mathbf{K}, \mathbf{D}, i), x^{j-m})$. Now, $j + i \notin \mathcal{R}$ implies that $(j - m) + (i + m) \notin \mathcal{R}$, and $j - m < M - m$, so by induction this coefficient is zero as well.

**Proof of 3.** For $M = 1$, if $i \in \mathcal{R}$, say $i = j_\ell$, we have $\mathbf{F} = \mathbf{F}_{j_\ell}$, whereas $F$ has entries in $\Bbbk$; this allows us to define $S'$. When $i \notin \mathcal{R}$, we have $F = S(\mathbf{F})$, so the claim holds as well. Thus, we are done for $M = 1$.

For $M > 1$, we assume our claim for all $(i, M')$ with $M' < M$. Write $H = F \bmod x^m$, $K = F$ div $x^m$ and $D = (C - x^k\,\delta(H) + (q^i A - \gamma_i\, x^{k-1}\, \mathsf{Id})\, \sigma(H))$ div $x^m$. Then, (4.5) implies that $E(H, C, i) = 0 \bmod x^m$ and $E(K, D, i + m) = 0 \bmod x^{M-m}$. The induction assumption shows that $H$ is a specialization of $\mathbf{H}$, say $H = S'(\mathbf{H})$ for some $S'$: $\mathscr{L}_{i+m} \to \mathcal{M}_{n,1}(\Bbbk[x])$ which extends $S$. In particular, $D = S'(\mathbf{D})$. The induction assumption also implies that there exist an extension $S''$: $\mathscr{L}_{i+m} \to \mathcal{M}_{n,1}(\Bbbk[x])$ of $S'$, and thus of $S$, such that $K = S''(\mathbf{K})$. Then $F = S''(\mathbf{F})$, so we are done.

For the complexity analysis, the most expensive part of the algorithm is the computation of $\mathbf{D}$. At the inner recursion steps, the bottleneck is the computation of $A\ \sigma(\mathbf{H})$, where $\mathbf{H}$ has degree less than $M$ and $A$ can be truncated mod $x^M$ (the higher degree terms have no influence in the subsequent recursive calls). Computing $\sigma(\mathbf{H})$ takes time $\mathcal{O}(N\ (n + r\ n^2))$ and the product is done in time $\mathcal{O}((n^2 + r\, n^\omega)\, \mathsf{M}(M))$; recursion leads to a factor $\log(M)$. The base cases use $\mathcal{O}(M)$ matrix inversions of cost $\mathcal{O}(n^\omega)$ and $\mathcal{O}(M)$ multiplications, each of which takes time $\mathcal{O}(r\, n^\omega)$. □

The second step of the algorithm is plain linear algebra: we know that the output of the previous algorithm satisfies our main equation for all indices $i \notin \mathcal{R}$, so we conclude by forcing the remaining ones to zero.

```
Algorithm 4.2
Divide-and-Conquer                                         $\mathsf{DAC}(A, C, N, k)$

Input:  $A \in \mathcal{M}_n(\Bbbk[[x]]), C \in \mathcal{M}_{n,1}(\Bbbk[[x]]),\ N \in \mathbb{N} \setminus \{0\}, k \in \mathbb{N} \setminus \{0\}$
Output: Generators of the solution space of
        $x^k \delta(F) = A\,\sigma(F) + C$ at precision $N$.

$\mathbf{F} := \mathsf{RDAC}(A, C, 0, N, k)$
                    ($\mathbf{F}$ has the form $\varphi_0 + \varphi_1 \mathbf{F}_{j_1} + \cdots + \varphi_r \mathbf{F}_{j_r}$)
$\mathbf{T} := x^k \delta(\mathbf{F}) - A\,\sigma(\mathbf{F}) - C \bmod x^N$
$\Gamma := (\mathbf{T}_i^{(j)}, i \in \mathcal{R}, j = 1, \ldots, n)$
$\Phi, \Delta := \mathsf{LinSolve}(\Gamma = 0)$
$M := [\varphi_1, \ldots, \varphi_r]$
return $\varphi_0 + M\,\Phi, M\,\Delta$
```

**Proposition 4.5.** *On input $A$, $C$, $N$, $k$ as specified in Algorithm 4.2, this algorithm returns generators of the solution space of (4.3) mod $x^N$ in time $\mathcal{O}((n^2 + r\,n^\omega)\,\mathsf{M}(N)\log(N) + r^2 n^\omega N + r^\omega n^\omega)$.*

**Proof.** The first claim is a direct consequence of the construction above, combined with Lemma 4.4. For the cost estimate, we need to take into account the computation of $\mathbf{T}$, the linear system solving, and the final matrix products. The computation of $\mathbf{T}$ fits in the same cost as that of $\mathbf{D}$ in Algorithm 4.1, so no new contribution comes from here. Solving the system $\Gamma = 0$ takes time $\mathcal{O}((r\,n)^\omega)$. Finally, the product $[\varphi_1 \cdots \varphi_r]\,\Delta$ involves an $n \times (r\,n)$ matrix with entries of degree $N$ and an $(r\,n) \times t$ constant matrix, with $t \leq r\,n$; proceeding coefficient by coefficient, and using block matrix multiplication in size $n$, the cost is $\mathcal{O}(r^2 n^\omega N)$. □

When all matrices $R_i$ are invertible, the situation becomes considerably simpler: $r = 0$, the solution space has dimension 0, there is no need to introduce formal parameters, the cost drops to $\mathcal{O}(n^2\ \mathsf{M}(N)\ \log\ (N) + n^\omega\ N)$ for Lemma 4.4, and Proposition 4.5 becomes irrelevant.

When $A_0$ has good spectrum at precision $N$, we may not be able to ensure that $r = 0$, but we always have $r \leq 1$. Indeed, when $k = 1$, the good spectrum condition implies that for all $0 \leq i < N$ and for $j \in \mathbb{N}$, the matrices $R_i$ and $R_j$ have disjoint spectra so that at most one of them can be singular. For $k > 1$, the good spectrum condition implies that all $R_i$ are invertible, whence $r = 0$. This proves Thm. 4.2.

**Previous work.** As said above, Barkatou and Pflügel [BP99], then Barkatou, Broughton and Pflügel [BBP10], already gave algorithms that solve such equations term-by-term, introducing formal parameters to deal with cases where the matrix $R_i$ is singular. These algorithms handle some situations more finely than we do (e.g., the cases $k \geq 2$), but take quadratic time; our algorithm can be seen as a divide-and-conquer version of these results.

In the particular case $q \neq 1$, $n = 1$ and $r = 0$, another forerunner to our approach is Brent and Traub's divide-and-conquer algorithm [BT80]. That algorithm is analyzed for a more general $\sigma$, of the form $\sigma(x) = x\, q(x)$, as such, they are more costly than ours; when $q$ is constant, we essentially end up with the approach presented here.

Let us finally mention van der Hoeven's paradigm of relaxed algorithms [Hoe02, Hoe09, Hoe11], which allows one to solve systems such as (4.3) in a term-by-term fashion, but in quasi-linear time. The cornerstone of this approach is fast *relaxed multiplication*, otherwise known as *online multiplication*, of power series.

In [Hoe02, Hoe03], van der Hoeven offers two relaxed multiplication algorithms (the first one being similar to that of [FS74]); both take time $\mathcal{O}(\mathsf{M}(n) \log(n))$. When $r = 0$, this yields a complexity similar to Prop. 4.5 to solve Eq. (4.3), but it is unknown to us how this carries over to arbitrary $r$.

When $r = 0$, both our divide-and-conquer approach and the relaxed one can be seen as "fast" versions of quadratic time term-by-term extraction algorithms. It should appear as no surprise that they are related: as it turns out, at least in simple cases (with $k = 1$ and $n = 1$), using the relaxed multiplication algorithm of [Hoe03] to solve Eq. (4.3) leads to doing exactly the same operations as our divide-and-conquer method, without any recursive call. We leave the detailed analysis of these observations to future work.

For suitable "nice" base fields (e.g., for fields that support Fast Fourier Transform), the relaxed multiplication algorithm in [Hoe02] was improved in [Hoe07, Hoe12], by means of a reduction of the $\log(n)$ overhead. This raises the question whether such an improvement is available for divide-and conquer techniques.

## 4.3 Newton Iteration

### 4.3.1 Gauge Transformation

Let $F$ be a solution of Eq. (4.3). To any invertible matrix $W \in \mathcal{M}_n(\Bbbk[x])$, we can associate the matrix $Y = W^{-1} F \in \mathcal{M}_n(\Bbbk[[x]])$. We are going to choose $W$ in such a way that $Y$ satisfies an equation simpler than (4.3). The heart of our contribution is the efficient computation of such a $W$.

**Lemma 4.6.** *Let $W \in \mathcal{M}_n(\Bbbk[x])$ be invertible in $\mathcal{M}_n(\Bbbk[[x]])$ and let $B \in \mathcal{M}_n(\Bbbk[x])$ be such that*

$$B = W^{-1}(x^k \delta(W) - A\,\sigma(W)) \bmod x^N. \tag{4.6}$$

*Then $F$ in $\mathcal{M}_{n,1}(\Bbbk[x])$ satisfies*

$$x^k \delta(F) = A\,\sigma(F) + C \bmod x^N \tag{4.7}$$

*if and only if* $Y = W^{-1} F$ *satisfies*

$$x^k \delta(Y) = B \sigma(Y) + W^{-1} C \bmod x^N. \tag{4.8}$$

**Proof.** Differentiating the equality $F = WY$ gives

$$x^k \delta(F) = x^k \delta(W) \sigma(Y) + x^k W \delta(Y).$$

Since $x^k \delta(W) = A \sigma(W) - WB \bmod x^N$, we deduce

$$x^k \delta(F) - A \sigma(F) - C = W (x^k \delta(Y) - B \sigma(Y) - W^{-1} C) \bmod x^N.$$

Since $W$ is invertible, the conclusion follows. □

The systems (4.3) and (4.8) are called equivalent under the gauge transformation $Y = WF$. Solving (4.3) is thus reduced to finding a simple $B$ such that (4.8) can be solved efficiently and such that the equation

$$x^k \delta(W) = A \sigma(W) - WB \bmod x^N \tag{4.9}$$

that we call *associated* to (4.3) has an *invertible* matrix $W$ solution that can be computed efficiently too.

As a simple example, consider the differential case, with $k = 1$. Under the good spectrum assumption, it is customary to choose $B = A_0$, the constant coefficient of $A$. In this case, the matrix $W$ of the gauge transformation must satisfy

$$x W' = A W - W A_0 \bmod x^N.$$

It is straightforward to compute the coefficients of $W$ one after the other, as they satisfy $W_0 = \mathsf{Id}$ and, for $i > 0$,

$$(A_0 - i \, \mathsf{Id}) W_i - W_i A_0 = -\sum_{j<i} A_{i-j} W_j.$$

However, using this formula leads to a quadratic running time in $N$. The Newton iteration presented in this section computes $W$ in quasi-linear time.

### 4.3.2 Polynomial Coefficients

Our approach consists in reducing efficiently the resolution of (4.3) to that of an equivalent equation where the matrix $A$ of power series is replaced by a matrix $B$ of polynomials of low degree. This is interesting because the latter can be solved in *linear* complexity by extracting coefficients. This subsection describes the resolution of the equation

$$x^k \delta(Y) = P \sigma(Y) + Q, \tag{4.10}$$

where $P$ is a polynomial matrix of degree less than $k$.

```
Algorithm 4.3
                                   PolCoeffsDE($P, Q, k, N$)
Input:   $P \in \mathcal{M}_n(\mathbb{k}[x])$ of degree less than $k$,
         $Q \in \mathcal{M}_{n,1}(\mathbb{k}[[x]])$, $N \in \mathbb{N} \setminus \{0\}$, $k \in \mathbb{N} \setminus \{0\}$
Output:  Generators of the solution space of
         $x^k \delta(Y) = P\sigma(Y) + Q$ at precision $N$.

for $i = 0, \ldots, N-1$
    $C := Q_i + (P_1 q^{i-1} Y_{i-1} + \ldots + P_{k-1} q^{i-k+1} Y_{i-k+1})$
    if $(k = 1)$
        $Y_i, M_i :=$ LinSolve($(\gamma_i \mathsf{Id} - q^i P_0) X = C$)
    else
        $Y_i, M_i :=$ LinSolve($-q^i P_0 X = C - \gamma_{i-k+1} Y_{i-k+1}$)
return $Y_0 + \ldots + Y_{N-1} x^{N-1}$, $[M_0 \; M_1 x \cdots M_{N-1} x^{N-1}]$
```

**Lemma 4.7.** *Suppose that $P_0$ has good spectrum at precision $N$. Then Algorithm 4.3 computes generators of the solution space of Eq. (4.10) at precision $N$ in time $\mathcal{O}(n^\omega N)$, with $M \in \mathcal{M}_{n,t}(\mathbb{k})$ for some $t \leq n$.*

**Proof.** Extracting the coefficient of $x^i$ in Eq. (4.10) gives

$$\gamma_{i-k+1} Y_{i-k+1} = q^i P_0 Y_i + \ldots + q^{i-k+1} P_{k-1} Y_{i-k+1} + Q_i.$$

In any case, the equation to be solved is as indicated in the algorithm. For $k = 1$, we actually have $C = Q_i$ for all $i$, so all these systems are independent. For $k > 1$, the good spectrum condition ensures that the linear system has full rank for all values of $i$, so all $M_i$ are empty. For each $i$, computing $C$ and solving for $Y_i$ is performed in $\mathcal{O}(n^\omega)$ operations, whence the announced complexity. □

### 4.3.3 Computing the Associated Equation

Given $A \in \mathcal{M}_n(\mathbb{k}[[x]])$, we are looking for a matrix $B$ with polynomial entries of degree less than $k$ such that the associated Equation (4.9), which does not depend on the non-homogeneous term $C$, has an invertible matrix solution.

In this article, we content ourselves with a simple version of the associated equation where we choose $B$ in such a way that (4.9) has an invertible solution $V \bmod x^k$; thus, $V$ and $B$ must satisfy $A\ \sigma(V) = V B \bmod x^k$. The invertible matrix $V$ is then lifted at higher precision by Newton iteration (Algorithm 4.6) under regularity conditions that depend on the spectrum of $A_0$. Other cases can be reduced to this setting by the polynomial gauge transformations that are used in the computation of formal solutions [BBP10, Was65].

When $k = 1$ or $q \neq 1$, the choice

$$B = A \bmod x^k, \quad V = \mathsf{Id}$$

solves our constraints and is sufficient to solve the associated equation. When $q = 1$, $k > 1$ (in particular when the point 0 is an irregular singular point of the equation), this is not be the case anymore. In that case, we use a known technique called the *splitting lemma* to prepare our equation. See for instance [Bal00, Ch. 3.2] and [BBP10] for details and generalizations.

**Lemma 4.8. (Splitting Lemma)** *Suppose that* $k > 1$*, that* $|\mathrm{Spec} A_0| = n$ *and that* $\mathrm{Spec} A_0 \subset \mathbb{k}$*. Then one can compute in time* $\mathcal{O}(n^\omega)$ *matrices* $V$ *and* $B$ *of degree less than* $k$ *in* $\mathcal{M}_n(\mathbb{k}[x])$ *such that the following holds:* $V_0$ *is invertible;* $B$ *is diagonal;* $A\,V = V B \bmod x^k$*.*

**Proof.** We can assume that $A_0$ is diagonal: if not, we let $P$ be in $\mathcal{M}_n(\mathbb{k})$ such that $D = P^{-1} A P$ has a diagonal constant term; we find $V$ using $D$ instead of $A$, and replace $V$ by $PV$. Computing $P$ and $PV$ takes time $\mathcal{O}(n^\omega)$, since as per convention, $k$ is considered constant in the cost analyses.

Then, we take $B_0 = A_0$ and $V_0 = \mathsf{Id}$. For $i > 0$, we have to solve $A_0 V_i - V_i A_0 - B_i = \Delta_i$, where $\Delta_i$ can be computed from $A_1, \ldots, A_i$ and $B_1, \ldots, B_{i-1}$ in time $\mathcal{O}(n^\omega)$. We set the diagonal of $V_i$ to 0. Since $A_0$ is diagonal, the diagonal $B_i$ is then equal to the diagonal of $\Delta_i$, up to sign. Then the entry $(\ell, m)$ in our equation reads $(r_\ell - r_m)\, V_i^{(\ell,m)} = \Delta_i^{(\ell,m)}$, with $r_1, \ldots, r_n$ the (distinct) eigenvalues of $A_0$. This can always be solved, in a unique way. The total time is $\mathcal{O}(n^\omega)$. □

### 4.3.4 Solving the Associated Equation

Once $B$ and $V$ are determined as in §4.3.3, we compute a matrix $W$ that satisfies the associated Equation (4.9); this eventually allows us to reduce (4.3) to an equation with polynomial coefficients. This computation of $W$ is performed efficiently using a suitable version of Newton iteration for Eq. (4.9); it computes a sequence of matrices whose precision is roughly doubled at each stage. This is described in Algorithm 4.6; our main result in this section is the following.

**Proposition 4.9.** *Suppose that* $A_0$ *has good spectrum at precision* $N$*. Then, given a solution of the associated equation mod* $x^k$*, invertible in* $\mathcal{M}_n(\mathbb{k}[[x]])$*, Algorithm 4.6 computes a solution of that equation* $\bmod x^N$*, also invertible in* $\mathcal{M}_n(\mathbb{k}[[x]])$*, in time* $\mathcal{O}(n^\omega\, \mathsf{M}(N) + n^\omega \log(n)\, N)$*.*

Before proving this result, we show how to solve yet another type of equations that appear in an intermediate step:

$$x^k\, \delta(U) = B\, \sigma(U) - U B + \Gamma \bmod x^N, \tag{4.11}$$

where all matrices involved have size $n \times n$, with $\Gamma = 0 \bmod x^m$. This is dealt with by Algorithm 4.4 when $k = 1$ or $q \neq 1$ and Algorithm 4.5 otherwise.

For Algorithm 4.4, remember that $B = A \bmod x^k$. The algorithm uses a routine $\mathsf{Sylvester}$ solving *Sylvester equations*. Given matrices $Y$, $V$, $Z$ in $\mathcal{M}_n(\mathbb{k})$, we are looking for $X$ in $\mathcal{M}_n(\mathbb{k})$ such that $YX - XV = Z$. When $(Y, V)$ have disjoint spectra, this system admits a unique solution, which can be computed $\mathcal{O}(n^\omega \log(n))$ operations in $\mathbb{k}$ [Kir01].

**Algorithm 4.4**

Solving Eq. (4.11) when $k=1$ or $q\neq 1$ $\quad$ $\mathsf{DiffSylvester}(\Gamma, m, N)$

```
Input:  $\Gamma \in x^m \mathcal{M}_n(\mathbb{k}[[x]]), m \in \mathbb{N} \setminus \{0\}, N \in \mathbb{N} \setminus \{0\}$
Output: $U \in x^{m-k} \mathcal{M}_n(\mathbb{k}[x])$ solution of (4.11).

for $i = m, \ldots, N-1$
    $C$ := $(B_1 q^{i-1} U_{i-1} + \ldots + B_{k-1} q^{i-k+1} U_{i-k+1})$
           $-(U_{i-1} B_1 + \ldots + U_{i-k+1} B_{k-1}) + \Gamma_i$
    if $(k=1)$
        $U_i := \mathsf{Sylvester}(X B_0 + (\gamma_i \mathsf{Id} - q^i B_0) X = C)$
    else
        $U_i := \mathsf{Sylvester}(X B_0 - q^i B_0 X = C - \gamma_{i-k+1} U_{i-k+1})$
return $U_m x^m + \ldots + U_{N-1} x^{N-1}$
```

**Lemma 4.10.** *Suppose that $k=1$ or $q\neq 1$ and that $A_0$ has good spectrum at precision $N$. If $\Gamma = 0 \bmod x^m$, with $k \leq m < N$, then Algorithm 4.4 computes a solution $U$ to Eq. (4.11) that satisfies $U = 0 \bmod x^{m-k+1}$ in time $\mathcal{O}(n^\omega \log(n) N)$.*

**Proof.** Extracting the coefficient of $x^i$ in (4.11) gives

$$\gamma_{i-k+1} U_{i-k+1} = q^i B_0 U_i - U_i B_0 + C,$$

with $C$ as defined in Algorithm 4.4. In both cases $k=1$ and $k>1$, this gives a Sylvester equation for each $U_i$, of the form given in the algorithm. Since $B_0 = A_0$, the spectrum assumption on $A_0$ implies that these equations all have a unique solution. Since $\Gamma$ is $0 \bmod x^m$, so is $U$ (so we can start the loop at index $m$). The total running time is $\mathcal{O}(n^\omega \log(n) N)$ operations in $\mathbb{k}$. □

**Algorithm 4.5**

Solving Eq. (4.11) when $k>1$ or $q=1$ $\quad$ $\mathsf{DiffSylvesterDifferential}(\Gamma, m, N)$

```
Input:  $\Gamma \in x^m \mathcal{M}_n(\mathbb{k}[[x]]), m \in \mathbb{N} \setminus \{0\}, N \in \mathbb{N} \setminus \{0\}$
Output: $U \in x^{m-k} \mathcal{M}_n(\mathbb{k}[x])$ solution of (4.11).

for $i = 1, \ldots, n$
    for $j = 1, \ldots, n$
        if $(i = j)$
            $U^{(i,i)} := x^k \int (x^{-k} \Gamma^{(i,i)}) \bmod x^N$
        else
            $U^{(i,j)} := \mathsf{PolCoeffsDE}(B^{(i,i)} - B^{(j,j)}, \Gamma^{(i,j)}, k, N)$
return $U$
```

This approach fails in the differential case ($q=1$) when $k>1$, since then the Sylvester systems are all singular. Algorithm 4.5 deals with this issue, using the fact that in this case, $B$ is diagonal, and satisfies the conditions of Lemma 4.8.

**Lemma 4.11.** *Suppose that $k>1$, $q=1$ and that $A_0$ has good spectrum at precision $N$. If $\Gamma = 0 \bmod x^m$, with $k \leq m < N$, then Algorithm 4.5 computes a solution $U$ to Eq. (4.11) that satisfies $U = 0 \bmod x^{m-k+1}$ in time $\mathcal{O}(n^2 N)$.*

**Proof.** Since $B$ is diagonal, the $(i,j)$th entry of (4.11) is

$$x^k \delta(U^{(i,j)}) = (B^{(i,i)} - B^{(j,j)})\, U^{(i,j)} + \Gamma^{(i,j)} \bmod x^N.$$

When $i=j$, $B^{(i,i)} - B^{(j,j)}$ vanishes. After dividing by $x^k$, we simply have to compute an integral, which is feasible under the good spectrum assumption (we have to divide by the non-zero $\gamma_1 = 1, \ldots, \gamma_{N-k} = N-k$). When $i \neq j$, the conditions ensure that Lemma 4.7 applies (and since $k>1$, the solution is unique, as pointed out in its proof). □

We now prove the correctness of Algorithm 4.6 for Newton iteration. Instead of doubling the precision at each step, there is a slight loss of $k-1$.

```
Algorithm 4.6
Newton iteration for Eq. (4.9)                                  NewtonAE($V, N$)

Input:   $V \in \mathcal{M}_n(\Bbbk[x])$ solution of (4.9) $\bmod x^k$ invertible in $\mathcal{M}_n(\Bbbk[[x]])$,
         $N \in \mathbb{N} \setminus \{0\}$
Output:  $W \in \mathcal{M}_n(\Bbbk[x])$ solution of (4.9) $\bmod x^N$ invertible in $\mathcal{M}_n(\Bbbk[[x]])$,
         with $W = V \bmod x^k$

if ($N \leqslant k$)
    return $V$
else
    $m := \lceil \frac{N+k-1}{2} \rceil$
    $H :=$ NewtonAE($V, m$)
    $R := x^k \delta(H) - A\,\sigma(H) + H\,B$
    if ($k=1$) or ($q \neq 1$)
        $U :=$ DiffSylvester($-H^{-1} R, m, N$)
    else
        $U :=$ DiffSylvesterDifferential($-H^{-1} R, m, N$)
    return $H + H\,U$
```

**Lemma 4.12.** *Let $m \geq k$ and let $H \in \mathcal{M}_n(\Bbbk[x])$ be invertible in $\mathcal{M}_n(\Bbbk[[x]])$ and satisfy (4.9) $\bmod x^m$. Let $N$ be such that $m \leq N \leq 2m - k + 1$. Let $R$ and $U$ be as in Algorithm 4.6 and suppose that $A_0$ has good spectrum at precision $N$.*

*Then $H + H\,U$ is invertible in $\mathcal{M}_n(\Bbbk[[x]])$ and satisfies the associated equation $\bmod x^N$. Given $H$, $U$ can be computed in time $\mathcal{O}(n^\omega \mathsf{M}(N) + n^\omega \log(n)\, N)$.*

**Proof.** By hypothesis, $R = 0 \bmod x^m$. Then

$$\begin{aligned} x^k \delta(H + HU) - A\,\sigma(H + H\,U) + (H + H\,U)\,B \\ = (x^k \delta(H) - A\,\sigma(H) + H\,B)\,(\mathsf{Id} + \sigma(U)) \\ + H\,(x^k \delta(U) + U\,B - B\,\sigma(U)) \\ = R\,(\mathsf{Id} + \sigma(U)) - R \bmod x^N = R\,\sigma(U) \bmod x^N. \end{aligned}$$

Using either Lemma 4.10 or Lemma 4.11, $U = 0 \bmod x^{m-k+1}$, so $\sigma(U) = 0 \bmod x^{m-k+1}$. Thus, the latter expression is 0, since $2m - k + 1 \geq N$. Finally, since $H\,U = 0 \bmod x^{m-k+1}$, and $m \geq k$, $H + H\,U$ remains invertible in $\mathcal{M}_n(\mathbb{k}[[x]])$. The various matrix products and inversions take a total number of $\mathcal{O}(n^\omega\,\mathsf{M}(N))$ operations in $\mathbb{k}$ (using Newton iteration to invert $H$). Adding the cost of Lemma 4.10, resp. Lemma 4.11, we get the announced complexity. □

We can now prove Proposition 4.9. Correctness is obvious by repeated applications of the previous lemma. The cost $C(N)$ of the computation up to precision $N$ satisfies

$$C(N) = C(m) + \mathcal{O}(n^\omega\,\mathsf{M}(N) + n^\omega \log n\,N), \quad N > k.$$

Using the super-additivity properties of the function $\mathsf{M}$ as in [GG03, Ch. 9], we obtain the claimed complexity.

We can now conclude the proof of Thm. 4.3. In order to solve Equation (4.3), we first determine $B$ and $V$ as in §4.3.3; the cost will be negligible. Then, we use Proposition 4.9 to compute a matrix $W$ that satisfies (4.9) mod $x^N$. Given $C$ in $\mathcal{M}_{n,1}(\mathbb{k}[[x]])$, we next compute $\Gamma = W^{-1}\,C \bmod x^N$. By the previous lemma, we conclude by solving

$$x^k\,\delta(Y) = B\,\sigma(Y) + \Gamma \bmod x^N.$$

Lemma 4.7 gives us generators of the solution space of this equation $\bmod x^N$. If it is inconsistent, we infer that Eq. (4.3) is. Else, from the generators $(Y, M)$ obtained in Lemma 4.7, we deduce that $(W\,Y, W\,M) \bmod x^N$ is a generator of the solution space of Eq. (4.3) $\bmod x^N$. Since the matrix $M$ has few columns (at most $n$), the cost of all these computations is dominated by that of Proposition 4.9, as reported in Thm. 4.3.

## 4.4 Implementation

We implemented the divide-and-conquer and Newton iteration algorithms, as well as a quadratic time algorithm, on top of NTL 5.5.2 [S+90]. In our experiments, the base field is $\mathbb{k} = \mathbb{Z}/p\mathbb{Z}$, with $p$ a 28 bit prime; the systems were drawn at random. Timings are in seconds, averaged over 50 runs; they are obtained on a single core of a 2 GHz Intel Core 2.

Our implementation uses NTL's built-in `zz_pX` polynomial arithmetic, that is, works with "small" prime fields (of size about $2^{30}$ over 32 bit machines, and $2^{50}$ over 64 bits machines). For this data type, NTL's polynomial arithmetic uses a combination of naive, Karatsuba and FFT arithmetic.

There is no built-in NTL type for polynomial matrices, but a simple mechanism to write one. Our polynomial matrix product is naive, of cubic cost. For small sizes such as $n = 2$ or $n = 3$, this is sufficient; for larger $n$, one should employ improved schemes (such as Waksman's [Wak70], see also [DIS11]) or evaluation-interpolation techniques [BS05].

Our implementation follows the descriptions given above, up to a few optimizations for algorithm $\mathsf{NewtonAE}$ (which are all classical in the context of Newton iteration). For instance, the inverse of $H$ should not be recomputed at every step, but simply updated; some products can be computed at a lower precision than it appears (such as $H^{-1}\,R$, where $R$ is known to have a high valuation).

In Fig. 4.1, we give timings for the scalar case, with $k = 1$ and $q \neq 1$. Clearly, the quadratic algorithm is outperformed for almost all values of $N$; Newton iteration performs better than the divide-and-conquer approach, and both display a subquadratic behavior. Fig. 4.2 gives timings when $n$ varies, taking $k = 1$ and $q \neq 1$ as before. For larger values of $n$, the divide-and-conquer approach become much better for this range of values of $N$, since it avoids costly polynomial matrix multiplication (see Thm. 4.2).

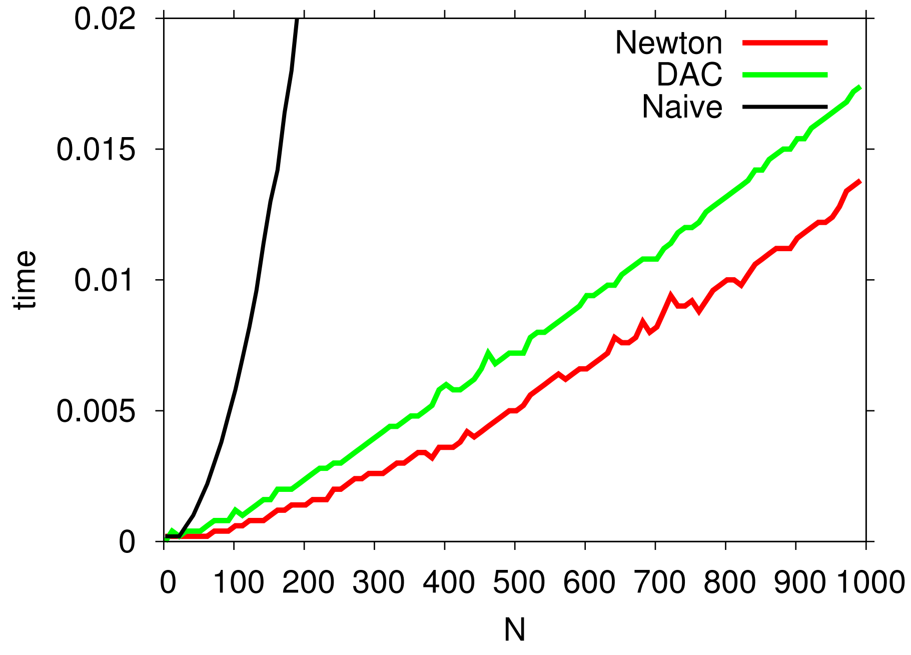

**Figure 4.1.** Timings with $n = 1$, $k = 1$, $q \neq 1$

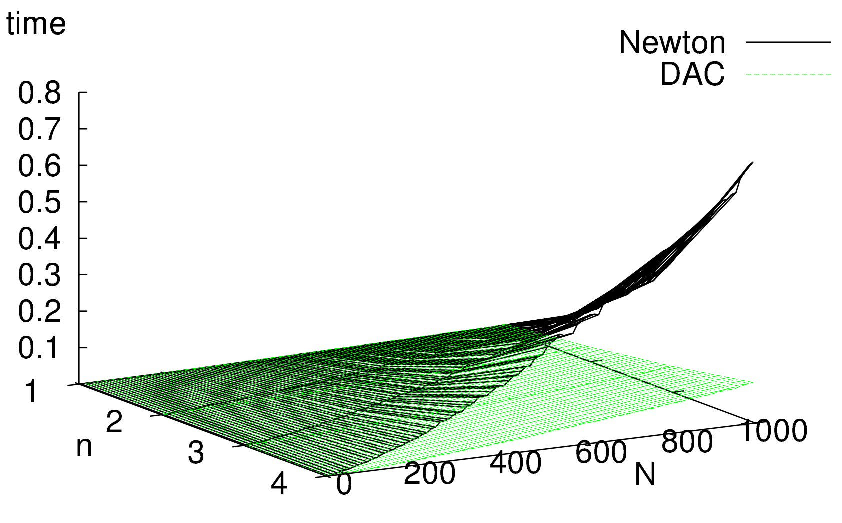

**Figure 4.2.** Timings with $k = 1$, $q \neq 1$

Finally, Table 4.1 gives timings obtained for $k = 3$, for larger values of $n$ (in this case, a plot of the results would be less readable, due to the large gap between the divide-and-conquer approach and Newton iteration, in favor of the former); DAC stands for "divide-and-conquer". In all cases, the experimental results confirm to a very good extent the theoretical cost analyses.

| Newton | | $n$ | | | |
|---|---|---|---|---|---|
| | | 5 | 9 | 13 | 17 |
| $N$ | 50 | 0.01 | 0.11 | 0.32 | 0.72 |
| | 250 | 0.22 | 1.2 | 3.7 | 8.1 |
| | 450 | 0.50 | 2.8 | 8.3 | 18 |
| | 650 | 0.93 | 5.1 | 16 | 34 |

| DAC | | $n$ | | | |
|---|---|---|---|---|---|
| | | 5 | 9 | 13 | 17 |
| $N$ | 50 | 0.01 | 0.01 | 0.02 | 0.04 |
| | 250 | 0.03 | 0.07 | 0.15 | 0.25 |
| | 450 | 0.06 | 0.16 | 0.32 | 0.52 |
| | 650 | 0.10 | 0.27 | 0.53 | 0.88 |

**Table 4.1.** Timings with $k=3$, $q\neq 1$

# Partie III

# Algebraic lifting