January 2014

Application of Computer Algebra in List Decoding

Muhammad Foizul Islam Chowdhury
The University of Western Ontario

Supervisor
Eric Schost
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy

Application of Computer Algebra in List Decoding

(Thesis format: Paper)

by

Muhammad Foizul Islam Chowdhury

Graduate Program
in
Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

The School of Graduate and Postdoctoral Studies
Western University
London, Ontario, Canada
August 22, 2013

Abstract

The amount of data that we use in everyday life (social media, stock analysis, satellite communication, etc.) is increasing day by day. As a result, the amount of data that needs to traverse through electronic media as well as to store are rapidly growing and there exist several environmental effects that can damage these important data during travelling or while in storage devices. To recover correct information from noisy data, we use error correcting codes. The most challenging work in this area is to have a decoding algorithm that can decode the code quite fast and that can tolerate highest amount of noise, so that we can use it in practice.

List decoding has been an active research area for last two decades. This research popularise in coding theory after the breakthrough work by Madhu Sudan where he used list decoding technique to correct errors that exceed half the minimum distance of Reed Solomon codes. Towards the direction of code development that can reach theoretical limits of error correction, Guruswami-Rudra introduced folded Reed Solomon codes that reached at $1 - R - \epsilon$ for $\epsilon > 0$. To decode these codes, one has to first interpolate a multivariate polynomial and then to factor out all possible roots. The difficulties that lie here are efficient interpolation, dealing with multiplicities smartly, and efficient factoring. This thesis deals with all these cases in order to have practical folded Reed Solomon codes.

Keywords. List Decoding, Structured matrix multiplication, Newton-Pueisux expansion, Newton-iteration.

Acknowledgments

I would first like to thank my thesis supervisor Associate Professor Éric Schost in the Department of Computer Science at Western University, Canada. The door to Prof. Éric Schost office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently helped me on the way of this thesis and steered me in the right direction whenever he thought I needed it. I am grateful to him for his excellent support to me in all arenas.

Sincere thanks and appreciation are extended to all the members from our Ontario Research Centre for Computer Algebra (ORCCA) lab and the Computer Science Department for their invaluable teaching and assistance.

My deep gratitude goes to my parents along with all my brothers, sisters, sister-in-laws, brother-in-laws, nieces and nephews. Special thanks goes to my brother Dr. Badrul Islam Choudhury and Sister-in-law Sabiha Choudhury Moona. My heartfull thanks goes to my two cutest nieces Manha and Rahmah who provided me lots of recreation during heavy workload.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

In everyday life, large amounts of data travel through various electronic comminication channels. The amount of data is increasing and no communication channel or storage device is error prone: errors can be introduced to these data while they are traversing wired or wireless media and corrupt them; these data can also be damaged while we store them in electronic media like hard disks or CDs / DVDs due to the presence of bad sectors or scratches. The solution to this problem is to apply a coding scheme, so that we can recover correct information even in the presence of errors in the data. Coding theory deals with this question.

A simple communication system is depicted in Figure 1.1; the main source of error is the noisy channel. To recover the correct information that passed through the noisy channel, coding schemes add some redundancy to the original data: if we want to transfer k symbols through the noisy channel, our codding scheme will encode these symbols into n (for $n > k$) symbols, by adding $n - k$ redundant symbols. These n symbols will then traverse the noisy channel, and we expect that the receiver will be able to recover the original symbols correctly, as long as the number of errors falls within some error limits. Both the decoder and encoder use error correcting codes to perform these operations.

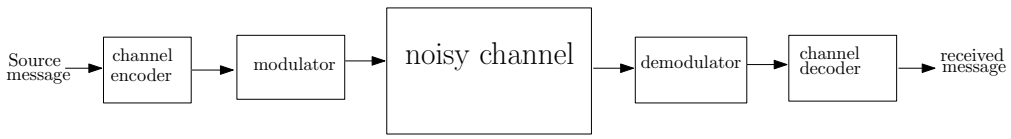


Figure 1.1: A simple communication system.

1.1 Error correcting codes

Error correcting codes tell us about how to add redundant symbols to the original data so that the receiver can recover the original message even if they received corrupted symbols. The following definitions taken from [4] are essentials for error correcting codes.

- **Encoding:** An injective function $E : \delta^k \to \delta^n$ having parameters k, $n \in \mathbb{N}$ that maps a message m consisting of k symbols over some alphabet δ (for example the binary alphabet $\delta = \{0, 1\}$) into a string $E(m)$ of length n over δ, where $n > k$, is known as an *encoding*. The word that is encoded by this function, i.e. $E(m)$, is referred as a codeword. If $|\delta|$ is finite then the alphabet size is often written $q = |\delta|$.

- **Decoding:** The sender sends the encoded codeword $E(m)$ through the noisy channel. The receiver may receive a distorted copy of the transmitted codeword, so she needs to figure out the original message. This role is done by a decoding function, $D : \delta^n \to \delta^k \bigcup \{1\}$, which maps the codeword of length n, which is possibly corrupted, to a string of length k.

- **Error-Correcting code:** It is the set of all codewords $\mathcal{C} \subset \delta^n$ that are obtained by encoding messages, i.e. the image of the encoding function E. Each codeword in $\mathcal{C}$ has the same length n, so we say that $\mathcal{C}$ is a block code of length n. If $\mathcal{C} \subseteq \delta^n$ is a block code of length n with $q = |\delta|$ finite, we say that $\mathcal{C}$ is a q-ary (error-correcting) code and the dimension of this code $\mathcal{C}$ is $k = \log_q |\mathcal{C}|$.

- **Rate:** The ratio $R = \frac{k}{n}$ of the number of symbols in a message to the length of the encoding map in the above definitions is called the *rate* of the code. This measures the amount of redundancy that is added by the encoding.

- **Distance:** Given two vectors x and y of length n, with entries in δ, the *distance* between these two vectors is the number of coordinates that differ from each other, i.e. $\{i | x_i \neq y_i\}$. The *minimum distance* (or simply, distance) of a code is defined to be the smallest distance between two distinct codewords; according to the *Singleton bound,* the minimum distance d of a code $\mathcal{C}$ cannot be greater than $n - k + 1$ and *maximum distance separable* (or MDS) codes are codes that remain within this bound.

The alphabets of our codes are usually taken to be finite fields $\mathbb{F}$ having q elements where q is a prime power, and we often make the assumption that $\mathcal{C} \subseteq \mathbb{F}^n$ is a linear

subspace of an n-dimensional vector space over $\mathbb{F}$; such codes are known as a *linear q-array codes*. Here the dimension of the code $k = \log_q |\mathcal{C}|$ matches the vector space dimension as an $\mathbb{F}$-subspace of $\mathbb{F}^n$. A linear code having length n dimension k and minimum distance d is also known as an $[n, k, d]_q$ code.

Depending on the number of errors, decoding strategies are available: typically, one can consider *unique decoding* or *list decoding*. In the former, the decoder outputs a single candidate for the message; this may be feasible only when few errors occured. List decoding is an alternative approach, due to Elias [3], which returns a list of candidates, one of which is the correct message. In this thesis, we will focus on list decoding algorithms for a particular family of codes, *Reed-Solomon codes*, which we introduce now.

1.2 Reed-Solomon codes

There exist many linear codes; Reed-Solomon codes are one of the most famous ones in coding theory, and will be the main focus of this thesis.

Definition 1. *(***Reed-Solomon Codes***) Let $\mathbb{F}$ be a field with at least n elements, and consider n pairwise distinct values $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}^n$. Given a polynomial $f(x) = \sum_{i=0}^{i=k-1} \delta_i x^i$ in $\mathbb{F}[x]$, where $\delta_i \in \mathbb{F}$ are the message symbols, the Reed-Solomon codeword $(y_1, \ldots, y_n)$ associated to f is obtained by evaluating $f(x)$ at α, so that $y_i = f(\alpha_i)$. This code is denoted*

$$RS_q(n, k, \alpha) = \{(f(\alpha_1), \ldots, f(\alpha_n)) \mid f \in \mathbb{F}[x]_k\},$$

where $\mathbb{F}[x]_k$ is the set of polynomials having coefficients in the base field $\mathbb{F}$ and degree less than k.

Although not strictly necessary, it will be convenient to take α of the form $1, \gamma, \ldots, \gamma^{n-1}$, for some γ in $\mathbb{F} \setminus \{0\}$ of multiplicative order at least n (often, by a slight abuse of expression, we may refer to γ as a primitive element, although strictly speaking this would be the case only if $|\mathbb{F}| = n + 1$).

Since a nonzero polynomial of degree less than k can have at most $k-1$ zeros, every nonzero codeword will have at least $n - k + 1$ nonzero components. The minimum distance of $RS_q(n, k, \alpha)$ is equal to $n - k + 1$, which is the singleton bound, so Reed-Solomon codes are maximum distance separable.

One of main design goals of error correcting codes is to make them highly tolerant to errors: the decoder should be able to correct as many errors as possible. In theory,

the number of errors that can be corrected by a decoder should be as close to $n-k$ as possible; that is, error correction should be possible as long as k symbols are correct. The fraction of errors ρ is defined as the ratio of the number of errors to the length; the information theoretic limit states that this number is at most $(n-k)/n = 1-R$ [5].

Standard unique decoding techniques, using for instance the Berlekamp-Massey algorithm, recover the correct message from the codeword and can decode the correct message as long as the number of erros is less than $(n-k)/2$. *Folded Reed-Solomon codes*, invented by Guruswami and Rudra [5] following previous work by Parvaresh and Vardy [9], are derived from Reed-Solomon codes and can tolerate an error ratio of $1-R-\varepsilon$ for any $\varepsilon > 0$ (using list decoding techniques).

These codes rely on a folding process, characterized by a *folding parameter* $s \geqslant 1$. For the folded case, we will make from the outset the assumption that the evaluation points $(\alpha_1, \ldots, \alpha_n)$ are of the form $(1, \gamma, \ldots, \gamma^{n-1})$.

Definition 2. *(***Folded Reed-Solomon codes***) Let s, n be integers such that s divides n; let $\mathbb{F}$ be a field with at least n elements, and consider n pairwise distinct values $\alpha = (1, \gamma, \ldots, \gamma^{n-1}) \in \mathbb{F}^n$. Given a polynomial $f(x) = \sum_{i=0}^{i=k-1} \delta_i x^i$ in $\mathbb{F}[x]$, where $\delta_i \in \mathbb{F}$ are the message symbols, the folded Reed-Solomon codeword associated to f consists in the n/s elements $Y_1, \ldots, Y_{n/s}$, with $y_i \in \mathbb{F}^s$ given by*

$$Y_i = (y_{(i-1)s+1}, \ldots, y_{(i-1)s+s}), \qquad y_i = f(\alpha_i).$$

In other words, the symbols in the codeword are now s-uples of values in $\mathbb{F}$, obtained by juxtaposing s consecutive "elementary symbols" $f(\gamma^i)$. When $s = 1$, we recover Reed-Solomon codes; Figure 1.2 gives an example of folded Reed-Solomon code taken from [5], where the folded parameter s is equal to 4.

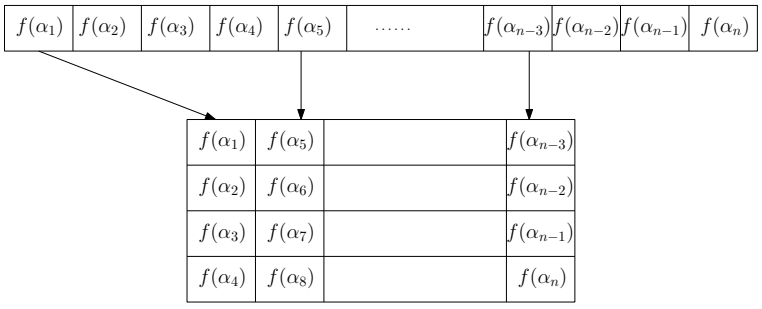


Figure 1.2: Folded Reed-Solomon code for $s = 4$.

The importance of these codes comes from the result (due to Guruswami and

Rudra) that for any $\varepsilon > 0$ and rate $R < 1$, there exists a family of folded Reed-Solomon codes that have rate at least R and which can be *list decoded* up to a fraction $1 - R - \varepsilon$ of errors in an efficient manner. We recall below some of the landmark results in this direction.

Recall first that unique decoding for Reed-Solomon codes can accommodate up to $(n-k)/2$ errors, or equivalently a ratio of $(1-R)/2 = 1 - (R+1)/2$.

The first crucial step toward incorporating a larger error ratio was solved by Madhu Sudan [10], using list decoding techniques. In a nutshell, Sudan's algorithm recovers a (small) list of polynomials that contains the message, provided the error ratio is at most $1 - \sqrt{2R}$. The following shows the main step in Sudan's list decoding algorithm; they form the blueprint of all algorithms that follow.

- Interpolate a bivariate polynomial $Q(x, y)$ such that $Q(\alpha_i, y_i) = 0$ (under suitable degree bounds [10]).
- Find all polynomials $f(x)$ such that $y - f(x)$ is a factor of Q.
- Keep those where $f(\alpha_i) = y_i$ for at least $n - e$ values of i, where e is a bound on the number of errors and the degree of f is less than k.

Correctness of this algorithm follows from imposing suitable degree bounds on Q; we will not need to state these bounds below.

This work was further developed by Guruswami and Sudan in [6] by introducing *multiplicities*: Q must vanish at high enough order at the points (α_i, y_i). For suitable choices of the degree bounds on Q and the multiplicity parameter, Guruswami and Sudan's algorithm allows error ratios up to $1 - \sqrt{R}$ [6].

Following the work of Guruswami and Sudan, Parvaresh and Vardy developed codes that can tolerate errors, and can be seen as precursors of the folded codes we defined above. Again, the algorithm proceeds through two main steps, evaluation and root-finding.

Finally, in 2008, Guruswami and Rudra introduced in [5] the folded Reed-Solmon codes defined above, by formulating a relation between Reed Solomon codes and Parvaresh and Vadry codes; their list-decoding algorithm works as follows:

- Compute a multivariate polynomial $Q(x, z_1, \ldots, z_s)$ such that, for $i = 0, \ldots, n-1$,

$$Q(\alpha_{si+1}, y_{si+1}, \ldots, y_{si+s}) = 0 \tag{1.1}$$

at order m (that is, all derivatives of Q of order up to m vanish as well at this point). Here, $\alpha_1, \ldots, \alpha_n$ are the evaluation points and $y_1, \ldots, y_n$ are the values as in Definition 2.

- Find all polynomials $f(x)$ such that $Q(x, f(x), f(\gamma x), \ldots, f(\gamma^{s-1}x)) = 0$.
- Keep all polynomials f such that $f(\alpha_i) = y_i$ for at least $(n - e)$ consecutive values of $i \in n$.

As for Sudan's algorithm, Q is subject to some degree constraints: upper bounds are given for its total degree, as well as for is *weighted degree*, for a weight where every variable z_i has degree $k - 1$. It will not be necessary for us to make these constraints explicit; Guruswami and Rudra's article gives all details. To conclude, remark that when $s = 1$, no folding occurs and we recover Sudan's algorithm.

1.3 Problem statement and overview of our results

In this work, we consider the two main steps highlighted in the above description of the Sudan / Guruswami-Sudan / Guruswami-Rudra list decoding algorithms: interpolation and root-finding. Although we saw that they are rooted in coding theory, these problems can be stated independently of the framework of error correction; this is the point of view we adopt, considering these questions as interesting by themselves.

Interpolation. The first question is to recover a polynomial Q that satisfies constraints (1.1), under additional requirements on its total degree and its weighted degree, for a well-chosen weight.

This is a linear algebra problem; as such, most early references on the subject (such as Sudan's and Guruswami-Sudan's papers) point out that this problem can be solved using essentially Gaussian elimination. Much work has been devoted to improve on naive linear algebra techniques: standard techniques now employ either fast linear algorithms, or polynomial lattice reduction techniques.

Chapter 3 gives a review of the existing literature and presents a new algorithm, inspired by previous work by Zeh, Gentner and Augot [11], which is the fastest to date (to the best of our knowledge).

The algorithm of Chapter 4 uses as a black box an algorithm for solving structured linear systems based in particular on recent techniques presented in [2]. In Chapter 4, we present these techniques in detail, and give the first report on the experimental behavior of this algorithm.

Root-finding. The second main question is to find all polynomials f that satisfy an equation of the type

$$Q(x, f(x), f(\gamma x), \ldots, f(\gamma^{s-1}x)) = 0, \tag{1.2}$$

for some Q in $\mathbb{F}[x, z_1, \ldots, z_s]$. When $s = 1$, this means that f satisfies $Q(x, f(x)) = 0$, so we are left with a bivariate factorization problem, for which standard solutions exist. For higher values of s, the solution proposed by Guruswami and Rudra (following previous work by Parvaresh-Vardy) is the following.

Assume that γ has multiplicative order precisely $q - 1$, with $q = |\mathbb{F}|$, let P be the irreducible polynomial $P(x) = x^{q-1} - 1$, and let $\mathbb{F}' = \mathbb{F}[x]/P$. Then, Guruswami and Rudra prove that f can be recovered as a root of

$$T = Q(x, z, z^q, \ldots, z^{q(s-1)}),$$

seen as a univariate polynomial in $\mathbb{F}'[z]$. However, the large degree of this polynomial in z makes this approach very expensive in practice.

Following previous work by Pecquet and Augot for the Guruswami-Sudan case [1], we investigate how lifting techniques can be used to compute *power series* (and thus polynomial) solutions of (1.2).

Equations such as (1.2) are often called *q-difference* equations, although in our context we should call them γ-difference equations (traditionally, the scaling factor is written as q rather than as γ; this goes back to at least [8]). It turns out that such equations are very similar to differential equations; the analogy can be seen by noting that, if we were in a context were we could let γ approach 1,

$$\lim_{\gamma \to 1} \frac{f(\gamma x) - f(x)}{(\gamma - 1)x} = f'(x)$$

for any polynomial f. As

$$\frac{f(\gamma x) - f(x)}{(\gamma - 1)x} = \frac{\sum_i f_i(\gamma^i - 1)x^i}{(\gamma - 1)x}$$

that becomes $f'(x)$ when $\gamma \to 1$.

In Chapter 5, we give algorithms that handle simultaneously differential and q-difference cases, using either Newton iteration or divide-and-conquer techniques, in the simplest case where Q is linear in $z_1, \ldots, z_s$. Previous algorithms existed to handle the differential case, under some regularity assumptions; our algorithms extends these

results to (some) singular cases and to the q-difference case (which is needed in our applications to list-decoding).

In Chapter 6, we describe the case of arbitrary Q. When $s = 1$, well-known techniques involve a combination of Newton iteration (when the solutions have no multiplicity) and of a desingularization process called the Newton-Puiseux algorithm in general. We show how these techniques extend to the folded case, and using recent work by Cano and Fortuny Ayuso [7] we propose a heuristic that drastically simplifies the resolution process.

Bibliography

[1] D. Augot and L. Pecquet. A Hensel lifting to replace factorization in list-decoding of algebraic-geometric and Reed-Solomon codes. *IEEE Transactions on Information Theory*, 46(7):2605–2614, 2000.

[2] A. Bostan, C.-P. Jeannerod, C. Mouilleron, and É. Schost. Fast simultaneous multiplication of a structured matrix by vectors. *PrePrint*, 2012.

[3] P. Elias. List decoding for noisy channels. *Technical Report 335*, pages 94–104, September-1957.

[4] V. Guruswami. *List decoding of error-correcting codes*, volume 3282 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[5] V. Guruswami and A. Rudra. Error correction up to the information-theoritic limit. *Communications of the ACM*, 52(3):87–95, 2009.

[6] V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45(6):1757 – 1767, Sep–1999.

[7] P. Fortuny Ayuso J. Cano. Power series solutions of non-linear q-difference equations and the Newton-Puiseux algorithm, 2012. arXiv:1209.0295.

[8] F. H. Jackson. q-difference equations. *American Journal of Mathematics*, 32(4):pp. 305–314, 1910.

[9] F. Parvaresh and A. Vardy. Correcting errors beyond the Guruswami-Sudan radius in polynomial time. In *FOCS'05*, pages 285 – 294. IEEE Computer Society, 2005.

[10] M. Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal Of Complexity*, 13:180–193, 1997.

[11] A. Zeh, C. Gentner, and D. Augot. An interpolation procedure for list decoding ReedSolomon codes based on generalized key equations. *IEEE Transaction on Information Theory*, 57(9):5946–5959, 2011.

Chapter 2

Mathematical preliminaries

This chapter discusses about various mathematical basics that was used through out the thesis.

2.1 Introduction

This chapter begins with the description of field, ring, group. Then we describe about several structured matrix that act as basic for solving our linear system. The presentation of this part follows [1].

2.1.1 Group

A group is a set of elements with a binary operation $\diamond$ that have following properties. Usually the group is denoted by $\{\mathcal{G}, \diamond\}$.

1. Closer: $\boldsymbol{a} \diamond \boldsymbol{b} \in \mathcal{G} \iff \boldsymbol{a}, \boldsymbol{b} \in \mathcal{G}$.
2. Associativity: $\boldsymbol{a} \diamond (\boldsymbol{b} \diamond \boldsymbol{c}) = (\boldsymbol{a} \diamond \boldsymbol{b}) \diamond \boldsymbol{c} \qquad \forall \boldsymbol{a},\ \boldsymbol{b}, \boldsymbol{c} \in \mathcal{G}$.
3. Commutativity: $\boldsymbol{a} \diamond \boldsymbol{b} = \boldsymbol{b} \diamond \boldsymbol{a}\ \forall \boldsymbol{a}, \boldsymbol{b} \in \mathcal{G}$.
4. Identity element: The group $\mathcal{G}$ has an element $\boldsymbol{e}$ for which we have $\boldsymbol{a} \diamond \boldsymbol{e} = \boldsymbol{e} \diamond \boldsymbol{a} = \boldsymbol{a}\ \forall \boldsymbol{a} \in \mathcal{G}$. The element $\boldsymbol{e}$ is known as identity element of that group.
5. Inverse element: there exist an element $\boldsymbol{a}' \in \mathcal{G}$ for each $\boldsymbol{a} \in \mathcal{G}$ such that $\boldsymbol{a} \diamond \boldsymbol{a}' = \boldsymbol{a}' \diamond \boldsymbol{a} = \boldsymbol{e}$ i.e. identity element.

A group $\mathcal{G}$ is called finite when it has finite number of elements which is also known as the order of that group $\mathcal{G}$. When the number of elements are not finite, its known as infinite group.

A group $\mathcal{G}$ is said to be cyclic when each element of that group can be represented as a power of an element of that group. Let $\boldsymbol{a}$ be an element of a group $\mathcal{G}$, then by means of powering of an element of that group, we refer that number of group operations, e.g. $\boldsymbol{a}^3 = \boldsymbol{a} \diamond \boldsymbol{a} \diamond \boldsymbol{a}$. The element of a group $\mathcal{G}$ which can be used to represent all element of $\mathcal{G}$ by this powering operation is known as a primitive element of that group $\mathcal{G}$. It is also called generator of the group $\mathcal{G}$.

2.1.2 Ring

A ring is a set of elements with two binary operations addition and multiplication that have following properties. We represent a ring by $\{\mathcal{R}, +, \times\}$,

1. A ring $\mathcal{R}$ have all the properties of a group. If $\mathcal{R}$ is an additive group then 0 is it's identity element and $-\boldsymbol{a}$ is the inverse of an element $\boldsymbol{a} \in \mathcal{R}$.
2. Closure under multiplication: $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{R} \quad \Rightarrow \boldsymbol{a} \times \boldsymbol{b} \in \mathcal{R}$.
3. Commutativity under multiplication: $\boldsymbol{a} \times \boldsymbol{b} = \boldsymbol{b} \times \boldsymbol{a} \quad \forall \boldsymbol{a}, \boldsymbol{b} \in \mathcal{R}$.
4. Associativity under multiplication: $\forall \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathcal{R}$; we have $\boldsymbol{a} \times (\boldsymbol{b} \times \boldsymbol{c}) = (\boldsymbol{a} \times \boldsymbol{b}) \times \boldsymbol{c}$.
5. Distributivity: $\forall \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathcal{R}$; we have
   (a) $\boldsymbol{a} \times (\boldsymbol{b} + \boldsymbol{c}) = (\boldsymbol{a} \times \boldsymbol{b}) + (\boldsymbol{a} \times \boldsymbol{c})$
   (b) $(\boldsymbol{a} + \boldsymbol{b}) \times \boldsymbol{c} = (\boldsymbol{a} \times \boldsymbol{c}) + (\boldsymbol{b} \times \boldsymbol{c})$

An integral domain is a commutative ring that have following properties in addition to the properties of a ring $\mathcal{R}$.

- Multiplicative identity: $\forall \boldsymbol{a} \in \mathcal{R}$, we have $1 \in \mathcal{R}$ such that $\boldsymbol{a} \times 1 = 1 \times \boldsymbol{a} = \boldsymbol{a}$.
- No zero divisor: $\boldsymbol{a} \times \boldsymbol{b} = 0 \quad \forall \boldsymbol{a}, \boldsymbol{b} \in \mathcal{R} \Rightarrow$ either $\boldsymbol{a} = 0$ or $\boldsymbol{b} = 0$.

2.1.3 Field

A field $\mathcal{F}$ is a set of elements with two operations addition and multiplication that have following properties.

- Integral domain: $\mathcal{F}$ have all the properties defined above.

- Multiplicative inverse: For each element $\boldsymbol{a} \in \mathcal{F} \setminus 0$, we have $\boldsymbol{a}^{-1} \in \mathcal{F}$ such that $\boldsymbol{a}\boldsymbol{a}^{-1} = \boldsymbol{a}^{-1}\boldsymbol{a} = 1$.

We can do all arithmetic operation, i.e. addition, subtraction, multiplication and division, on a a field $\mathcal{F}$ and the result of the operation will be in that field $\mathcal{F}$. Here the division operations is performed as follows

$$\frac{\boldsymbol{a}}{\boldsymbol{b}} = \boldsymbol{a}\boldsymbol{b}^{-1} \quad \forall \boldsymbol{a}, \boldsymbol{b} \in \mathcal{F}.$$

Polynomial Ring

A polynomial ring over a field $\mathcal{F}$, represented by $\mathcal{F}[x]$, is a set of polynomials P of the form

$$P = p_0 + p_1 X + p_2 X^2 + \cdots + P_{m-1} X^{m-1} + P_m X^m.$$

where the coefficients of P, $p_0, \ldots, p_m$, are elements of underlying field $\mathcal{F}$ and X is indeterminate. The degree of P is the highest power in X that has nonzero coefficient.

2.1.4 Notion of Finite field

Let p be a prime and n is a positive integer, then the number of elements of (also known as order of finite field) a finite field is p^n. Here p is said to be the characteristic of the field. Generally we use $GF(p^n)$ or $\mathbb{F}_{p^n}$ to denote a finite field having order p^n. GF stands for Galois Field. The structure of the finite field when $n > 1$ is different than the structure of the finite field when $n = 1$.

For a prime p and $n = 1$, the finite field $\mathbb{F}_p$ $(GF(P))$ is the set $\mathbb{Z}_p$ of integers with arithmetic operation modulo prime p. Here

$$\mathbb{Z}_p = \{0, 1, \ldots, p-1\}.$$

Let $\mathbb{F}$ be a field, then a polynomial $f \in \mathbb{F}[x]$, that have coefficients over the field $\mathbb{F}$, is said to be irreducible over the field $\mathbb{F}$ if and only if $f(x)$ is irreducible as an element over the polynomial ring $\mathbb{F}[x]$.

For a prime p and $n > 1$, the finite field $\mathbb{F}_{p^n}$ $(GF(P^n))$ is defined by using an irreducible polynomial f over finite field $\mathbb{F}_p$ having degree n. In this field, we perform all arithmetic operations modulo the irreducible polynomial $f(x)$.

Example of finite field of the form $\mathbb{F}_{p^n}$

Let ξ be the set of all polynomials having degree less than n over a field $\mathbb{F}_p$. Each polynomial in this set can be represented by

$$f(x) = p_0 + p_1 X + \cdots + p_{n-1} x^{n-1}$$

where $p_i \in \mathbb{F}_p$, for $0 \leqslant i \leqslant n-1$. It is easily verifiable that the set ξ has p^n number of polynomials. The set having this property is a finite field with following arithmetic operations.

- All basic arithmetic operations is executed modulo p for coefficients.
- When the degree of product of two elements from ξ is greater than n, it gets reduced modulo an irreducible polynomial $f(x)$ having degree n;

We gave a simple example of a finite field $\mathbb{F}_{p^n}$ which is taken from [1]. Here we choose $p = 2$ and $n = 3$.

Example of operations on a finite field $\mathbb{F}_{2^3}$

An irreducible polynomial having degree 3 over $\mathbb{F}_2$ is $x^3 + x + 1$ and let the set ξ has polynomials of degree less than 3 over $\mathbb{F}_2$. Here

$$\xi = \{0, 1, x, 1+x, x^2, 1+x^2, x+x^2, 1+x+x^2\}.$$

The arithmetic operation on this elements modulo $x^3 + x + 1$ are shown in following tables 2.1 and table 2.2.

$+$	0	1	x	$1+x$	x^2	$1+x^2$	$x+x^2$	$1+x+x^2$
0	0	1	x	$1+x$	x^2	$1+x^2$	$x+x^2$	$1+x+x^2$
1	1	0	$1+x$	x	$1+x^2$	x^2	$1+x+x^2$	$x+x^2$
x	x	$1+x$	0	1	$x+x^2$	$1+x+x^2$	x^2	$1+x^2$
$1+x$	$1+x$	x	1	0	$1+x+x^2$	$x+x^2$	$1+x^2$	x^2
x^2	x^2	$1+x^2$	$x+x^2$	$1+x+x^2$	0	1	x	$1+x$
$1+x^2$	$1+x^2$	x^2	$1+x+x^2$	$x+x^2$	1	0	$1+x$	x
$x+x^2$	$x+x^2$	$1+x+x^2$	x^2	$1+x^2$	x	$1+x$	0	1
$1+x+x^2$	$1+x+x^2$	$x+x^2$	$1+x^2$	x^2	$1+x$	x	1	0

Table 2.1: Addition modulo $1 + x + x^3$

$\times$	0	1	x	$1+x$	x^2	$1+x^2$	$x+x^2$	$1+x+x^2$
0	0	0	0	0	0	0	0	0
1	0	1	x	$1+x$	x^2	$1+x^2$	$x+x^2$	$1+x+x^2$
x	0	x	x^2	$x+x^2$	$1+x$	1	$1+x+x^2$	$1+x^2$
$1+x$	0	$1+x$	$x+x^2$	$1+x^2$	$1+x+x^2$	x^2	1	x
x^2	0	x^2	$1+x$	$1+x+x^2$	$x+x^2$	x	$1+x^2$	1
$1+x^2$	0	$1+x^2$	1	x^2	x	$1+x+x^2$	$1+x$	$x+x^2$
$x+x^2$	0	$x+x^2$	$1+x+x^2$	1	$1+x^2$	$1+x$	x	x^2
$1+x+x^2$	0	$1+x+x^2$	$1+x^2$	x	1	$x+x^2$	x^2	$1+x$

Table 2.2: Multiplication modulo $1+x+x^3$

2.1.5 Polynomial multiplication

Polynomial consists of variable and coefficients. A polynomial is known as univariate when it has only one variable. When a polynomial has multiple variable, we called it as multivariate polynomial. We write univariate polynomial as

$$P = a_0 + a_1 x^1 + \cdots + a_n x^n$$

where all coefficients a_i's are in a ring $\mathcal{R}$ or in a field $\mathcal{F}$ and x is a variable. The degree of a univariate polynomial is the highest power in x that has nonzero coefficient. A monomial is a polynomial that has only one term.

Let $a \in \mathbb{F}[x]$ and $b \in \mathbb{F}[x]$ are two polynomials having degree less than n. Then by $\mathsf{M}(n)$, we denote the number of operations required to multiply the polynomials a and b.

The total degree of a monomial $x_1^{j_1} x_2^{j_2} \ldots x_n^{j_n}$ is $i_1 + i_2 + \cdots + i_n$, where x_i's are variables and j_i's are integers for $1 \leqslant i \leqslant n$. The degree of a multivariate polynomial is the highest total degree among all its monomials that has nonzero coefficient. The $(\mu_1, \ldots, \mu_n)$ weighted degree of the previous monomial is equal to $\sum_{k=1}^{n} \mu_k j_k$ where all μ_i's are integers for $1 \leqslant \mathbf{i} \leqslant n$. The weighted degree of a multivariate polynomial is the highest weighted degree among all its monomials that have nonzero coefficient.

2.1.6 Matrix structure

In chapter 3 we explored structured property of our linear problem. In this section we gave example of two structured matrices.

A matrix whose form is

$$\begin{bmatrix} l_{1,1} & & & & 0 \\ l_{2,1} & l_{2,2} & & & \\ t_{3,1} & l_{3,2} & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ l_{n,1} & l_{n,2} & \dots & l_{n,n-1} & l_{n,n} \end{bmatrix} \in \mathbb{F}^{n\times n}$$

is known as lower triangular whereas a matrix having the form

$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \dots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \dots & u_{2,n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & u_{n-1,n} \\ 0 & & & & u_{n,n} \end{bmatrix} \in \mathbb{F}^{n\times n}$$

is known as upper triangular matrix.

A matrix is known as Toeplitz matrix when the elements on diagonals of a matrix are same. It looks like following

$$\begin{bmatrix} t_0 & t_{-1} & t_{-2} & \dots & \dots & t_{-n+1} \\ t_1 & t_0 & t_{-1} & \ddots & & \vdots \\ t_2 & t_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & t_{-1} & t_{-2} \\ \vdots & & \ddots & t_1 & t_0 & t_{-1} \\ t_{n-1} & \dots & \dots & t_2 & t_1 & t_0 \end{bmatrix} \in \mathbb{F}^{n\times n}.$$

A matrix is known as Hankel matrix when the elements on antidiagonals of a matrix are same and looks like following

$$\begin{bmatrix} h_0 & h_1 & h_2 & \dots & \dots & h_{n-1} \\ h_1 & h_2 & h_3 & \ddots & & \vdots \\ h_2 & h_3 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & h_{-(n-2)} & h_{-(n-1)} \\ \vdots & & \ddots & h_{-(n-2)} & h_{-(n-1)} & h_{-n} \\ h_{n-1} & \dots & \dots & h_{-(n-1)} & h_{-n} & h_{-n+1} \end{bmatrix} \in \mathbb{F}^{n\times n}.$$

These matrices can be represented in a compact form. Let A is a Toeplitz matrix, and Z is a matrix of the form

$$Z = \begin{bmatrix} & & & 0 \\ 1 & & & \\ & 1 & & \\ & & \ddots & \\ 0 & & & 1 \end{bmatrix} \in \mathbb{F}^{n \times n},$$

then we have

$$A = L[G_1]U[H_1^t] + L[G_2]U[H_2^t]$$

where

- $L[G_j]$ is a lower triangular Toeplitz matrix;
- $U[H_j^t]$ is a upper triangular Toeplitz matrix;

and

$$G_1 = \begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_{n-1} \end{bmatrix} \quad G_2 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad H_1 = \begin{bmatrix} 1 & \\ 0 & \\ \vdots & \vdots \\ 0 & \end{bmatrix} \quad H_2 = \begin{bmatrix} 0 \\ t_{-1} \\ \vdots \\ t_{-n+1} \end{bmatrix}.$$

So we have

$$A - ZAZ^t = GH^t$$

where $Z - ZAZ^t$ is known as stain operator. Here the matrices G and H are known as generator matrices of A and the rank of G and H are 2. A matrix is known as structured when the rank of its generator matrix is lower than the rank of that matrix. The rank of $A - ZAZ^t$ is known as displacement rank of A.

Bibliography

[1] William Stalling. *Cryptography and Network Security.* Prentice Hall, 2005.

Chapter 3

On the Complexity of Multivariate Interpolation with Multiplicities and of Simultaneous Polynomial Approximations

This chapter is published in the homonym paper with Claude-Pierre Jeannerod, Vincent Neiger, Éric Schost and Gilles Villard in the proceedings of ASCM12.

The interpolation step in the Guruswami-Sudan algorithm has attracted a lot of interest and it is now solved by many algorithms in the literature. This problem of interpolation with multiplicities has been generalized to multivariate polynomials, with links to the list-decoding of folded Reed-Solomon codes. Here, we present two approaches to address this multivariate interpolation which both boil down to solving a structured homogeneous linear system. The first approach has similarities with the derivation of Extended Key Equations presented in [39] while the second one corresponds to solving simultaneous polynomial approximations. In the special case of Reed-Solomon list-decoding, both our approaches have complexity $\mathcal{O}^{\sim}(\ell^{\omega-1}m^2n)$, where ℓ, m, n are the list size and $\mathcal{O}^{\sim}$ hide the logarithmic factor from traditional $\mathcal{O}$ notation, the multiplicity and the number of sample points and ω is the exponent of matrix multiplication.

3.1 Introduction

In this paper, we consider a multivariate interpolation problem which originates from coding theory. In what follows, $\mathbb{K}$ is our base field and, in the coding theory context, s, ℓ, m, n, k, b are respectively known as the *number of variables*, *list size*, *multiplicity*, *code length*, *message length* and as an *agreement parameter* (which is such that $n - b/m$ is an upper bound on the number of errors that are allowed on a received word).

We stress here that we do not address the problem of choosing the parameters s, ℓ, m with respect to n, k, b, as is often done: in our context, these are all input parameters. Similarly, although we will mention them, we do not make some usual assumptions on these parameters; in particular, we do not make any assumption that ensures that our problem admits a solution: the algorithm will detect whether no solution exists.

Here and hereafter, bold face letters are used for vector objects; $\deg_{\boldsymbol{Y}}$ denotes the total degree (summation of exponents of all variables) with respect to variables $\boldsymbol{Y} = Y_1, \ldots, Y_s$ and $\deg_X$ denotes the degree in a single variable X.

Problem 1. MultivariateInterpolation
Input: positive integers s, ℓ, m, n, k, b; points $\{(x_i, y_{i,1}, \ldots, y_{i,s})\}_{1 \leqslant i \leqslant n}$ in $\mathbb{K}^{s+1}$ with the x_i's pairwise distinct.

Output: a polynomial Q in $\mathbb{K}[X, Y_1, \ldots, Y_s]$ satisfying the following conditions:

(i) Q is nonzero

(ii) $\deg_{\boldsymbol{Y}}(Q) \leqslant \ell$

(iii) $\deg_X(Q(X, X^k Y_1, \ldots, X^k Y_s)) < b$

(iv) for $1 \leqslant i \leqslant n$, $Q(x_i, y_{i,1}, \ldots, y_{i,s}) = 0$ with order at least m.

We call conditions (ii), (iii) and (iv) the *list-size* condition, the *weighted-degree* condition and the *vanishing* condition, respectively. Here, we say that a point $(x_i, y_{i,1}, \ldots, y_{i,s})$ is a zero of Q of *order at least* m if the shifted polynomial $Q(X + x_i, Y_1 + y_{i,1}, \ldots, Y_s + y_{i,s})$ has no monomial of total degree less than m. In characteristic zero, or larger than m, this means that all derivatives of Q of order up to $m - 1$ vanish at $(x_i, y_{i,1}, \ldots, y_{i,s})$.

For $\boldsymbol{j}$ in $\mathbb{N}^s$, with $\boldsymbol{j} = (j_1, \ldots, j_s)$, write $|\boldsymbol{j}| = j_1 + \cdots + j_s$. Let further $\Gamma \subset \mathbb{N}^s$ be the set of all $\boldsymbol{j}$ in $\mathbb{N}^s$ such that $|\boldsymbol{j}| \leqslant \ell$ and $k|\boldsymbol{j}| < b$. Then, defining

$$N_{\boldsymbol{j}} = b - k|\boldsymbol{j}| > 0,$$

we see that conditions (ii) and (iii) are equivalent to Q being written as

$$Q(X, \boldsymbol{Y}) = \sum_{\boldsymbol{j} \in \Gamma} Q_{\boldsymbol{j}}(X) \boldsymbol{Y}^{\boldsymbol{j}}, \quad \text{with} \quad \deg(Q_{\boldsymbol{j}}) < N_{\boldsymbol{j}} \text{ for all } \boldsymbol{j}, \tag{3.1}$$

where we write $\boldsymbol{Y}^{\boldsymbol{j}}$ to denote the s-variate monomial $\boldsymbol{Y}^{\boldsymbol{j}} = Y_1^{j_1} \cdots Y_s^{j_s}$. For $\boldsymbol{i}$ in $\mathbb{N}^s$ such that $|\boldsymbol{i}| < m$, let further

$$M_{\boldsymbol{i}} = n(m - |\boldsymbol{i}|)$$

and define finally

$$M = \sum_{|\boldsymbol{i}|<m} M_{\boldsymbol{i}} = \binom{s+m}{s+1} n \quad \text{and} \quad N = \sum_{|\boldsymbol{j}| \in \Gamma} N_{\boldsymbol{j}}. \tag{3.2}$$

Then, under conditions (ii) and (iii), finding Q amounts to finding a non-trivial solution to a homogeneous linear system with N unknowns and M equations. It is customary to assume that $N > M$, in order to guarantee the existence of a non-trivial solution; however, as said above, we do not make this assumption, since our algorithms do not require it.

This problem is a generalization to s variables $Y_1, \ldots, Y_s$ of the interpolation step of list-decoding algorithms based on Sudan's idea and its generalization by Guruswami and Sudan [36, 18]: Sudan's algorithm corresponds to $m = s = 1$ and Guruswami-Sudan's algorithm to $s = 1$. Multivariate interpolation problems, with $s > 1$, correspond for instance to Parvaresh-Vardy codes [29] or folded Reed-Solomon codes [17].

Our solution to Problem 1 relies on a reduction to a simultaneous approximation problem defined below, which generalizes Padé and Hermite-Padé approximation.

Problem 2. SimultaneousPolynomialApproximations

Input: positive integers μ, ν, $(M'_0, \ldots, M'_{\mu-1})$ and $(N'_0, \ldots, N'_{\nu-1})$ and polynomials $(P_i, \boldsymbol{F}_i)_{0 \leqslant i < \mu}$ in $\mathbb{K}[X]$, such that for all i, $\boldsymbol{F}_i = (F_{i,0}, \ldots, F_{i,\nu-1})$, P_i is monic of degree M'_i and $\deg(F_{i,j}) < M'_i$.

Output: polynomials $\boldsymbol{Q} = (Q_0, \ldots, Q_{\nu-1})$ in $\mathbb{K}[X]$ satisfying the following conditions:

(a) the Q_j's are not all zero

(b) for $0 \leqslant j < \nu$, $\deg(Q_j) < N'_j$,

(c) for $0 \leqslant i < \mu$, $\sum_{0 \leqslant j < \nu} Q_j F_{i,j} = 0 \bmod P_i$.

We present two algorithms to solve the latter problem. Both involve a linearization of the univariate equations (c) into a homogeneous linear system over $\mathbb{K}$; if we define

$$M' = \sum_{0 \leqslant i < \mu} M'_i \quad \text{and} \quad N' = \sum_{0 \leqslant j < \nu} N'_j;$$

then this system has M' equations in N' unknowns (remark that as above, we do not assume that $N' > M'$).

Our two algorithms amount to reformulating this set of equations as structured linear systems, which we solve using the algorithm given by Bostan, Jeannerod and Schost in [7]. The first approach, given in Section 3.5, follows the derivation of Extended Key Equations presented in the case $s = 1, m = 1$ by Roth and Ruckenstein [32] and generalized to $s = 1, m \geqslant 1$ by Zeh, Gentner and Augot [39]; the matrix of the system is mosaic-Hankel. In our second approach, presented in Section 3.6, the structured linear system is directly obtained from condition (c), without using key equations described in [39].

Both points of view lead to the same result, which says that Problem 2 can be solved in time quasi-linear in M', multiplied by a subquadratic term in $\rho = \max(\mu, \nu)$. In the following theorems, and the rest of this paper, the soft-O notation $\mathcal{O}^{\tilde{}}(\ )$ indicates that we omit polylogarithmic terms. The exponent ω is so that we can multiply $n \times n$ matrices using $\mathcal{O}(n^\omega)$ ring operations on any ring; the best known bound on ω is $\omega \leqslant 2.3727$ [12, 35, 38]. Finally, the function M is a *multiplication time* function for $\mathbb{K}[X]$: M is such that polynomials of degree at most d in $\mathbb{K}[X]$ can be multiplied in $\mathsf{M}(d)$ operations in $\mathbb{K}$, and such that M satisfies the super-linearity properties of [14, Ch. 8]. It is known that $\mathsf{M}(d)$ can be taken in $\mathcal{O}(d \log(d) \log\log(d))$ [10].

Theorem 3. *There exists a probabilistic algorithm that either computes a solution to Problem 2, or determines that none exists, using* $\mathcal{O}(\rho^{\omega-1}\mathsf{M}(M')\log(M')^2) \subset \mathcal{O}^{\tilde{}}(\rho^{\omega-1}M')$ *operations in* $\mathbb{K}$, *where* $\rho = \max(\mu, \nu)$.

The algorithm chooses $\mathcal{O}(M')$ *elements in* $\mathbb{K}$; *if these elements are chosen uniformly at random in a set* $S \subset \mathbb{K}$ *of cardinality at least* $6(M'+1)^2$, *the probability of success is at least* $1/2$.

The probability analysis is a standard consequence of the Zippel-Schwartz lemma; as usual, the probability of success can be made arbitrarily close to one by increasing the size of S (this remark holds for all probabilistic algorithms mentioned below).

We will use Theorem 3 to solve Problem 1, which leads to the following result.

Theorem 4. *There exists a probabilistic algorithm that either computes a solution to Problem 1, or determines that none exists, using* $\mathcal{O}(r^{\omega-1}\mathsf{M}(M)\log(M)^2) \subset \mathcal{O}^{\sim}(r^{\omega-1}M)$ *operations in* $\mathbb{K}$*, where* $r = \max(|\Gamma|, \binom{s+m-1}{s})$*.*

The algorithm chooses $\mathcal{O}(M)$ *elements in* $\mathbb{K}$*; if these elements are chosen uniformly at random in a set* $S \subset \mathbb{K}$ *of cardinality at least* $6(M+1)^2$*, the probability of success is at least* $1/2$*.*

In order to understand this cost estimate, let us briefly discuss it under some usual assumptions on the input parameters:

$\mathbf{H_1}$: $m \leqslant \ell$,

$\mathbf{H_2}$: $\ell k < b$.

With regards to the first assumption, we mention that the case $m \geqslant \ell$ can easily be reduced to the case $m = \ell$ (see Lemma 6). The second assumption means that we do not take ℓ uselessly large: if $\ell k \geqslant b$, then the weighted-degree constraint implies that some of the coefficients $Q_{\boldsymbol{j}}$ are identically zero.

Under these assumptions, $|\Gamma| = \binom{s+\ell}{s}$, so $r = \binom{s+\ell}{s}$, whereas $M = \binom{s+m}{s+1}n$. Assume for simplicity that s is constant; then, r and M grow respectively like ℓ^s and $m^{s+1}n$. As a particular case, we obtain the following result, which discusses the Guruswami-Sudan algorithm with $s = 1$.

Corollary 5. *Taking* $s = 1$*, if the parameters* ℓ, m, n, k, b *satisfy* $\mathbf{H}_1$ *and* $\mathbf{H}_2$*, there exists a probabilistic algorithm that computes a solution to Problem 1 using* $\mathcal{O}(\ell^{\omega-1}\mathsf{M}(m^2n)\log(mn)^2)$ *operations in* $\mathbb{K}$*, which is* $\mathcal{O}^{\sim}(\ell^{\omega-1}m^2n)$*.*

The algorithm chooses $\mathcal{O}(m^2n)$ *elements in* $\mathbb{K}$*; if these elements are chosen uniformly at random in a set* $S \subset \mathbb{K}$ *of cardinality at least* $24m^4n^2$*, the probability of success is at least* $1/2$*.*

Notation. Regarding Problem 1, several univariate polynomials will be used repeatedly. The polynomial

$$G(X) = \prod_{i=1}^{n}(X - x_i),$$

is called the master polynomial associated to the x_i's; we will also use the s-tuple $\boldsymbol{R} = (R_1, \ldots, R_s)$ of Lagrange interpolation polynomials, defined by the conditions

$$\deg(R_j) < n \quad \text{and} \quad R_j(x_i) = y_{i,j}$$

for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant s$.

Previous work. We are not aware of previous results specific to Problem 2, but several particular cases of it are well known. When all P_i's are of the form X^{M_i}, this problem becomes known as a simultaneous Hermite-Padé approximation problem or vector Hermite-Padé approximation problem [3, 34]. The case $\mu = 1$, with P_1 being given through its roots (and their multiplicities) is known as the M-Padé problem [2].

Regarding Problem 1, previous results focus on the Guruswami-Sudan case $s = 1$; we summarize them in Table 3.1, in which we make assumptions $\mathbf{H_1}$ and $\mathbf{H_2}$. In some cases [30, 1, 5, 11], the complexity was not stated quite exactly in our terms but the translation is straightforward.

For this case, the most significant factor in the running time is its dependency with respect to n, with results either being cubic, quadratic, or quasi-linear. Then, under the assumption $\mathbf{H_1}$: $m \leqslant \ell$, the second most important parameter is ℓ, followed by m. In particular, our result in Corollary 5 compares favorably to the cost $\mathcal{O}^{\sim}(\ell^{\omega} mn)$ from [11], which was, to our knowledge, the best previous bound for this problem.

In the general case $s \geqslant 1$, the result in Theorem 4 improves as well on the best previously known bounds; we discuss those below.

Sudan case ($m = 1$)	
Sudan [36]	$\mathcal{O}(n^3)$
Roth-Ruckenstein [32]	$\mathcal{O}(\ell n^2)$
Olshevsky-Shokrollahi [27]	$\mathcal{O}(\ell n^2)$
This paper (probabilistic)	$\mathcal{O}(\ell^{\omega-1}\mathsf{M}(n)\log(n)^2)$
Guruswami-Sudan case ($m \geqslant 1$)	
Guruswami-Sudan [18]	$\mathcal{O}(m^6 n^3)$
Olshevsky-Shokrollahi [27]	$\mathcal{O}(\ell m^4 n^2)$
Augot-Gentner-Zeh [39]	$\mathcal{O}(\ell m^4 n^2)$
Kötter / McEliece [21, 23]	$\mathcal{O}(\ell m^4 n^2)$
Reinhard [30]	$\mathcal{O}(\ell^3 m^4 n^2)$
Lee-O'Sullivan [22]	$\mathcal{O}(\ell^4 m n^2)$
Trifonov [37] (heuristic)	$\mathcal{O}(m^3 n^2)$
Alekhnovich [1]	$\mathcal{O}(\ell^4 m^4 \mathsf{M}(n)\log(n))$
Beelen-Brander [4]	$\mathcal{O}(\ell^3 \mathsf{M}(\ell mn)\log(\ell mn))$
Bernstein [5]	$\mathcal{O}(\ell^{\omega} \mathsf{M}(\ell n)\log(\ell n))$
Cohn-Heninger [11]	$\mathcal{O}(\ell^{\omega} \mathsf{M}(mn)\log(\ell n))$
This paper (probabilistic)	$\mathcal{O}(\ell^{\omega-1}\mathsf{M}(m^2 n)\log(mn)^2)$

Table 3.1: Overview of previous results, $s = 1$.

Most previous algorithms rely on linear algebra, either over $\mathbb{K}$ or over $\mathbb{K}[X]$. Working over $\mathbb{K}$, a natural idea is to rely on cubic-time general linear system solvers, as in Sudan's and Guruswami-Sudan's original papers. Several papers also cast the

problem in terms of Gröbner basis computation in $\mathbb{K}[X,Y]$, implicitly or explicitly: the incremental algorithms of [21, 26, 23] are particular cases of the Buchberger-Möller algorithm [24], while Alekhnovich's algorithm [1] is a divide-and-conquer change-of-order for bivariate ideals.

Yet another line of work [32, 39] uses Feng-Tzeng's linear system solver [13], combined with a reformulation in terms of syndromes and key equations. We will use (and generalize to the case $s > 1$) some of these results in Section 3.5, but we will rely on the structured linear system solver of [7] in order to prove our main results. Prior to our work, Olshevsky and Shokrollahi also used structured linear algebra techniques [27], but it is unclear to us whether their encoding of the problem could lead to similar results as ours.

As said above, another approach rephrases the problem of computing Q in terms of polynomial matrix computations, that is, as linear algebra over $\mathbb{K}[X]$; this was in particular the basis of the extensions to the multivariate cases $s > 1$ in [9, 8]. Starting from generators of an ad-hoc $\mathbb{K}[X]$-module (or polynomial lattice) that is known to contain a non-trivial Q, the algorithms in [22, 9, 4, 8, 5, 11] compute a Gröbner basis of that lattice, or simply a short vector therein. To achieve quasi-linear time in n (almost linear up to logarithmic factor), the algorithms in [4, 8] use a short vector subroutine due to Alekhnovich [1], while those in [5, 11] rely on a (faster, but probabilistic) algorithm due to Giorgi, Jeannerod and Villard [15]. For a lattice of dimension L, with generators of degree at most d, that algorithm in [4, 8] runs in time $\mathcal{O}(L^\omega \mathsf{M}(d)\log(Ld))$. Note that a recent deterministic algorithm from [16] achieves the same cost; this is the best result known to date.

Two main lattice constructions exist in the literature (Bernstein [5] gives more refined constructions, better adapted to some choices of the parameters). Following [9], we present them directly in the case $s \geqslant 1$; we give the cost bounds that can be obtained using the (fast) algorithms of [15, 16] for lattice reduction. The first construction may be called *banded* (due to the shape of the generators it involves when $s = 1$); its generators derive from the polynomials G and $\boldsymbol{R}$ introduced before:

$$\left\{ G^i \prod_{r=1}^{s} (Y_r - R_r)^{j_r} \;\middle|\; i > 0,\ j_1, \ldots, j_s \geqslant 0,\ i + |\boldsymbol{j}| = m \right\}$$
$$\bigcup \left\{ \prod_{r=1}^{s} (Y_r - R_r)^{j_r} Y_r^{J_r} \;\middle|\; j_1, \ldots, j_s \geqslant 0,\ J_1, \ldots, J_s \geqslant 0,\ |\boldsymbol{j}| = m,\ |\boldsymbol{J}| \leqslant \ell - m \right\},$$

The second construction may be called *triangular*; its generators derive from the

polynomials

$$\left\{ G^i \prod_{r=1}^{s} (Y_r - R_r)^{j_r} \;\middle|\; i > 0,\ j_1, \ldots, j_s \geqslant 0,\ i + |\boldsymbol{j}| = m \right\}$$
$$\bigcup \quad \left\{ \prod_{r=1}^{s} (Y_r - R_r)^{j_r} \;\middle|\; j_1, \ldots, j_s \geqslant 0,\ m \leqslant |\boldsymbol{j}| \leqslant \ell \right\}.$$

When $s = 1$, the first construction is used in [4, Remark 16] and [22, 11] and the second is used in [4, 5]; the latter also appears in [8] for $s \geqslant 1$. In both cases, the actual lattice bases are the coefficient vectors (in $\boldsymbol{Y}$) of the polynomials $h(X, X^k Y_1, \ldots, X^k Y_s)$, for h in either of the sets above.

For the banded basis, we have the following dimension and degree bounds, from [9]:

$$L_b = \binom{s+m-1}{s} + \binom{s+m-1}{s-1}\binom{s+\ell-m}{s} \quad \text{and} \quad d_b = \mathcal{O}(mn);$$

in the triangular case, we have

$$L_t = \binom{s+\ell}{s} \quad \text{and} \quad d_t = \mathcal{O}(\ell n).$$

Under our assumption $m \leqslant \ell$, we always have $L_b \geqslant L_t$ and $d_b \leqslant d_t$; when $s = 1$, we get $L_b = L_t = \ell + 1$. In both cases, we readily deduce the cost of finding a polynomial Q from [15, 16], respectively to $\mathcal{O}(L_b^\omega \mathsf{M}(d_b) \log(L_b d_b))$ and $\mathcal{O}(L_t^\omega \mathsf{M}(d_t) \log(L_t d_t))$.

For $s = 1$, these are the costs reported in [5, 11]. For $s > 1$, the costs reported in [9, 8] are worse, because the short vector algorithms used in thoses references are inferior to the ones we refer to. Under $\mathbf{H_1}$ and $\mathbf{H_2}$, and possibly neglecting logarithmic factors from $\mathcal{O}$ notation, the result in Theorem 4 is an improvement over those of both [9] and [8]. To see this, remark that the cost in our theorem is quasi-linear in $\binom{s+\ell}{s}^{\omega-1}\binom{s+m}{s+1}n$, whereas the costs in [9, 8] are at least $\binom{s+\ell}{s}^{\omega} mn$; a quick simplification proves our claim.

It is interesting to notice that the two main approaches discussed here — solving a linear system over $\mathbb{K}$ or finding a short vector in a polynomial lattice $\mathcal{L}$ — ultimately rely on the same assumptions to ensure success. We have already mentioned that under the linear algebra point of view, the assumption $M < N$ ensures that Problem 1 admits a solution. For lattice-based methods, the list-size and vanishing conditions are consequences of belonging to the lattice; in order to guarantee that the shortest vector in the lattice $\mathcal{L}$ will correspond to a polynomial $Q \in \mathbb{K}[X, \boldsymbol{Y}]$ that satisfies the

weighted-degree condition, the following condition must hold:

$$\frac{\deg(\det(\mathcal{L}))}{\dim \mathcal{L}} \;<\; b.$$

For both lattices described above, assuming as before $m \leqslant \ell$, one can verify that this inequality can be rewritten

$$\frac{1}{\binom{s+\ell}{s}} \left(\sum_{0 \leqslant |\boldsymbol{i}| < m} n(m - |\boldsymbol{i}|) \;+\; \sum_{0 \leqslant |\boldsymbol{j}| \leqslant \ell} |\boldsymbol{j}| k \right) \;<\; b.$$

This is precisely the assumption $M < N$ seen before.

Outline of the paper. The next section briefly discusses the relevance of assumption $\mathbf{H_1}$. Then, after a reminder on algorithms for structured linear systems, we show how to reduce Problem 1 to Problem 2 in Section 3.4, then give two algorithms that both prove Theorem 3, in Sections 3.5 and 3.6.

3.2 Preliminaries: assumption $\mathbf{H_1}$

In this very brief section, we discuss assumption $\mathbf{H_1}$ that was introduced previously for Problem 1. In Theorem 4, we do not make any assumption on m and ℓ, but we mentioned that assumption $\mathbf{H_1}$, that is, $m \leqslant \ell$ is mostly harmless. The following lemma substantiates this claim, by showing that the case $m \geqslant \ell$ can be reduced to the case $m = \ell$. As mentioned in the introduction, we denote by G the master polynomial $\prod_{1 \leqslant i \leqslant n}(X - x_i)$.

Lemma 6. *Suppose that $m \geqslant \ell$. Then, if $b < n(m - \ell)$, Problem 1 with parameters (s, ℓ, m, n, k, b) has no solution. Else, the solutions to that problem are exactly the polynomials of the form $Q = Q^\star\, G^{m-\ell}$, where $Q^\star$ is a solution to Problem 1 with parameters $(s, \ell, \ell, n, k, b - n(m - \ell))$.*

Proof. Let Q be a solution to Problem 1 with parameters (s, ℓ, m, n, k, b). We claim that $b \geqslant n(m - \ell)$, that $G^{m-\ell}$ divides Q, and that $Q^\star = Q/G^{m-\ell}$ is a solution to Problem 1 with parameters $(s, \ell, \ell, n, k, b - (m - \ell))$.

Let i be in $\{1, \ldots, n\}$. By condition (iv), $Q_i = Q(X + x_i, Y_1 + y_{i,1}, \ldots, Y_s + y_{i,s})$ has no monomial of total degree less than m. By condition (ii), every monomial in Q_i has degree at most ℓ in $\boldsymbol{Y}$, so each such monomial is a multiple of $X^{m-\ell}$. Shifting back the coordinates, and considering all i's in $\{1, \ldots, n\}$ proves the first claim.

Let then $Q^\star = Q/G^{m-\ell}$. This polynomial is nonzero, has degree at most ℓ in $\boldsymbol{Y}$, and

$$Q^\star(X, X^k Y_1, \ldots, X^k Y_s) = Q(X, X^k Y_1, \ldots, X^k Y_s)/G^{m-\ell}.$$

Since the numerator on the right-hand side has degree less than b, and $Q^\star$ is nonzero we must in particular have $b \geqslant n(m-\ell)$, as claimed. Besides, for $1 \leqslant i \leqslant n$, the remarks above show that $Q^\star(x_i, y_{i,1}, \ldots, y_{i,s}) = 0$ with multiplicity ℓ. Thus, $Q^\star$ is a solution to Problem 1 with parameters $(s, \ell, \ell, n, k, b - n(m-\ell))$.

Conversely, let Q' be *any* solution to Problem 1 with parameters $(s, \ell, \ell, n, k, b - n(m-\ell))$. Proceeding as in the previous paragraphs, one easily verifies that $Q'\, G^{m-\ell}$ is a solution to the problem with parameters (s, ℓ, m, n, k, b), so the proof is complete. □

3.3 Solving structured linear systems

Our main algorithms rely on solving linear systems over $\mathbb{K}$. In this section, we briefly review useful concepts and results related to *displacement rank* techniques. While these techniques can handle systems with several kinds of structure, we will only need (and discuss) those related to *Toeplitz-like* and *Hankel-like* systems (explained in chapter 2); for a more comprehensive treatment, the reader may consult [28].

Let M be a positive integer and let $\mathcal{Z}_M \in \mathbb{K}^{M\times M}$ be the square matrix with ones on the subdiagonal and zeros elsewhere:

$$\mathcal{Z}_M = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix} \in \mathbb{K}^{M\times M}.$$

Given two integers M, N, consider the following operators:

$$\begin{array}{rccc} \Delta_{M,N} : & \mathbb{K}^{M\times N} & \to & \mathbb{K}^{M\times N} \\ & A & \mapsto & A - \mathcal{Z}_M\, A\, \mathcal{Z}_N^T \end{array}$$

and

$$\begin{array}{rccc} \Delta'_{M,N} : & \mathbb{K}^{M\times N} & \to & \mathbb{K}^{M\times N} \\ & A & \mapsto & A - \mathcal{Z}_M\, A\, \mathcal{Z}_N, \end{array}$$

which subtract from A its translate one place along the diagonal, resp. along the anti-diagonal.

Let us discuss $\Delta_{M,N}$ first. If A is a *Toeplitz* matrix, that is, invariant along diagonals, $\Delta_{M,N}(A)$ has rank at most two. As it turns out, Toeplitz systems can be solved much faster than general linear systems, in quasi-linear time in M. The main idea behind algorithms for structured matrices is to extend these algorithmic properties to those matrices A for which the rank of $\Delta_{M,N}(A)$ is small, in which case we call A *Toeplitz-like*. Below, this rank will be called the *displacement rank* of A (with respect to $\Delta_{M,N}$).

Two matrices (V, W) in $\mathbb{K}^{M\times\alpha} \times \mathbb{K}^{\alpha\times N}$ will be called a *generator of length* α for A with respect to $\Delta_{M,N}$ if $\Delta_{M,N}(A) = V\,W$. For the structure we are considering, one can recover A from its generators; in particular, one can use a generator of length α as a way to represent A using $\alpha(M + N)$ field elements. One of the main aspects of structured linear algebra algorithms is to use generators as a compact data structure throughout the whole process.

Up to now, we only discussed the Toeplitz structure. *Hankel-like* matrices are those which have a small displacement rank with respect to $\Delta'_{M,N}$, that is, those matrices A for which the rank of $\Delta'_{M,N}(A)$ is small. As far as solving the system $Au = 0$ is concerned, this case can be easily reduced to the Toeplitz-like case. Define $B = AJ_N$, where J_N is the reversal matrix of size N, all entries of which are zero, except the anti-diagonal which is set to one. Then, one easily checks that the displacement rank of A with respect to $\Delta'_{M,N}$ is the same as the displacement rank of B with respect to $\Delta_{M,N}$, and that if (V, W) is a generator for A with respect to $\Delta'_{M,N}$, (V, WJ_N) is a generator for B with respect to $\Delta_{M,N}$. Using the algorithm for Toeplitz-like matrices gives us a solution v to $Bv = 0$, from which we deduce that $u = J_N v$ is a solution to $Au = 0$.

In this paper, we will not enter the details of algorithms for solving such structured systems. The main result we will rely on is the following proposition, a minor extension of a result by Bostan, Jeannerod and Schost [7], which features the best known complexity for this kind of task, to the best of our knowledge. This algorithm is based on previous work of Bitmead-Anderson [6], Morf [25], Kaltofen [20] and Pan [28], and is probabilistic (it depends on the choice of some parameters in the base field, and success is ensured provided these parameters avoid a strict hypersurface of the parameter space).

The proof of the following proposition occupies the rest of this section. Remark that some aspects of this statement could be improved (for instance, we could re-

duce the cost so that it only depends on M, not $\max(M,N)$), but that would be inconsequential for the applications we make of it.

Proposition 7. *Given a generator (V,W) of length α for a matrix $A \in \mathbb{K}^{M\times N}$, with respect to either $\Delta_{M,N}$ or $\Delta'_{M,N}$, one can find a non-zero element in the nullspace of A, or determine that none exists, by a probabilistic algorithm that uses $\mathcal{O}(\alpha^{\omega-1}\mathsf{M}(P)\log(P)^2)$ operations in $\mathbb{K}$, with $P = \max(M,N)$.*

The algorithm chooses $\mathcal{O}(P)$ elements in $\mathbb{K}$; if these elements are chosen uniformly at random from a set $S \subset \mathbb{K}$ of cardinality at least $6P^2$, the probability of success is at least $1/2$.

Square matrices. In all that follows, we consider only the operator $\Delta_{M,N}$, since we already pointed out that the case of $\Delta'_{M,N}$ can be reduced to it at no extra cost.

When $M = N$, we use directly [7, Theorem 1], which gives the running time reported above. That result does not explicitly state which solution we obtain, as it is written for general non-homogeneous systems. Here, we want to make sure we obtain a nonzero nullspace element (if one exists), so slightly more details are needed.

The algorithm in that theorem chooses $3M-2$ elements in $\mathbb{K}$, the first $2M-2$ of which are used to precondition A by giving it generic rank profile; this is the case when these parameters avoid a hypersurface of $\mathbb{K}^{2M-2}$ of degree at most M^2+M.

Assume this is the case. Then, the output vector u is obtained in a parametric form as $u = \ell(u')$, where u' consists of another set of M parameters chosen in $\mathbb{K}$ and ℓ is a surjective linear mapping with image the nullspace $\ker(A)$ of A. If $\ker(A)$ is trivial, the algorithm returns the zero vector in any case, which is correct. Otherwise, the set of vectors u' such that $\ell(u') = 0$ is contained in a hyperplane of $\mathbb{K}^M$, so it is enough to choose u' outside of that hyperplane to ensure success.

Using the Zippel-Schwartz lemma and the obvious upper bound $M^2+M+1 \leqslant 3M^2$, we conclude that if we choose all parameters uniformly at random in a subset S of $\mathbb{K}$ of cardinality at least $6M^2$, the algorithm succeeds with probability at least $1/2$.

Wide matrices. Suppose now that $M < N$, so that the system is underdetermined. We add $N-M$ zero rows on top of A, obtaining an $N \times N$ matrix A'. Applying the algorithm for the square case to A', we will obtain a nullspace element u for A' and thus A, since these nullspaces are the same. In order to do so, we need to construct a generator for A' from the generator (V,W) we have for A: one simply takes (V',W), where V' is the matrix in $\mathbb{K}^{N\times\alpha}$ obtained by adding $N-M$ zero rows on top of V.

Tall matrices. Suppose finally that $M > N$. This time, we build the matrix $A' \in \mathbb{K}^{M\times M}$ by adjoining $M - N$ zero columns to A on the left. The generator (V, W) of A can be turned into a generator of A' by simply adjoining $M - N$ zero columns to W on the left. We then solve the system $A's = 0$, and return the vector u obtained by discarding the first $M - N$ entries of s.

The cost of this algorithm fits into the requested bound; all that remains to see is that we obtain a nonzero vector in the nullspace $\ker(A)$ of A with nonzero probability. Indeed, the nullspaces of A and A' are now related by the equality $\ker(A') = \mathbb{K}^{M-N} \times \ker(A)$. We mentioned earlier that in the algorithm for the square case, the solution s to $A's = 0$ is obtained in parametric form, as $s = \ell(s')$ for $s' \in \mathbb{K}^M$, with ℓ a surjective mapping $\mathbb{K}^M \to \ker(A')$. Composing with the projection $\pi : \ker(A') \to \ker(A)$, we obtain a parametrization of $\ker(A)$ as $u = (\pi \circ \ell)(s')$. The error probability analysis is then the same as in the square case.

3.4 Reducing Problem 1 to Problem 2

In this section, we show how instances of Problem 1 can be reduced to instances of Problem 2. The main technical ingredient, stated in Lemma 8 below, generalizes to any $s \geqslant 1$ the one given for $s = 1$ by Zeh, Gentner, and Augot in [39, Proposition 3]. To prove it, we use the same steps as in [39]; we rely on the notion of Hasse derivatives, which allow us to write Taylor expansions in positive characteristic (see for instance Hasse [19] or Roth [31, pp. 87, 276]).

In what follows, comparison and addition of s-tuples of integers are defined componentwise. For example, writing $\boldsymbol{i} \leqslant \boldsymbol{j}$ is equivalent to $i_k \leqslant j_k$ for $k = 1, \ldots, s$, and $\boldsymbol{i} - \boldsymbol{j}$ denotes $(i_1 - j_1, \ldots, i_s - j_s)$. Similarly, if $\boldsymbol{y} = (y_1, \ldots, y_s)$ is in $\mathbb{K}[X]^s$ then $\boldsymbol{Y} - \boldsymbol{y} = Y_1 - y_1, \ldots, Y_s - y_s$. Finally, for products of binomial coefficients, we shall write

$$\binom{\boldsymbol{j}}{\boldsymbol{i}} = \binom{j_1}{i_1} \cdots \binom{j_s}{i_s}.$$

Note that this coefficient is zero when $\boldsymbol{i} \not\leqslant \boldsymbol{j}$; note also that the monomial $\boldsymbol{Y}^{\boldsymbol{i}}$ has total degree $\deg_{\boldsymbol{Y}}(\boldsymbol{Y}^{\boldsymbol{i}}) = |\boldsymbol{i}| = i_1 + \cdots + i_s$.

If $\mathbb{A}$ is any commutative ring with unity and $\mathbb{A}[\boldsymbol{Y}]$ denotes the ring of polynomials in $Y_1, \ldots, Y_s$ over $\mathbb{A}$, then for a polynomial $P(\boldsymbol{Y}) = \sum_{\boldsymbol{j}} P_{\boldsymbol{j}} \boldsymbol{Y}^{\boldsymbol{j}}$ in $\mathbb{A}[\boldsymbol{Y}]$ and a multi-index $\boldsymbol{i}$ in $\mathbb{N}^s$, the *order-$\boldsymbol{i}$ Hasse derivative* of P is the polynomial $P^{[\boldsymbol{i}]}$ in $\mathbb{A}[\boldsymbol{Y}]$ defined

by

$$P^{[\boldsymbol{i}]} = \sum_{\boldsymbol{j} \geqslant \boldsymbol{i}} \binom{\boldsymbol{j}}{\boldsymbol{i}} P_{\boldsymbol{j}} \boldsymbol{Y}^{\boldsymbol{j}-\boldsymbol{i}}.$$

The Hasse derivative satisfies the following property (Taylor expansion): for all $\boldsymbol{a}$ in $\mathbb{A}^s$,

$$P(\boldsymbol{Y}) = \sum_{\boldsymbol{i}} P^{[\boldsymbol{i}]}(\boldsymbol{a})(\boldsymbol{Y} - \boldsymbol{a})^{\boldsymbol{i}}.$$

The following lemma shows how Hasse derivatives can help rephrase the vanishing condition (iv) of Problem 1. Below, the polynomials G and $\boldsymbol{R}$ are as defined in the introduction.

Lemma 8. *For any polynomial Q in $\mathbb{K}[X, \boldsymbol{Y}]$, Q satisfies the vanishing condition (iv) of Problem 1 if and only if for all $\boldsymbol{i}$ in $\mathbb{N}^s$ such that $|\boldsymbol{i}| < m$,*

$$Q^{[\boldsymbol{i}]}(X, \boldsymbol{R}) = 0 \bmod G^{m-|\boldsymbol{i}|}.$$

Proof. Since the x_r's defining $G = \prod_{r=1}^{n}(X - x_r)$ are pairwise distinct, it suffices to prove, for $1 \leqslant r \leqslant n$, the following equivalence for the point $(x_r, \boldsymbol{y}_r)$: $Q(x_r, \boldsymbol{y}_r) = 0$ with order at least m if and only if for all $\boldsymbol{i}$ in $\mathbb{N}^s$ such that $|\boldsymbol{i}| < m$, $Q^{[\boldsymbol{i}]}(X, \boldsymbol{R}) = 0 \bmod (X - x_r)^{m-|\boldsymbol{i}|}$. Now, up to a shift one can assume that this point is $\boldsymbol{0} \in \mathbb{K}^{s+1}$; in other words, it suffices to show that for $\boldsymbol{R}(0) = \boldsymbol{0} \in \mathbb{K}^s$, we have $Q(0, \boldsymbol{0}) = 0$ with order at least m if and only if, for all $\boldsymbol{i}$ in $\mathbb{N}^s$ such that $|\boldsymbol{i}| < m$, $X^{m-|\boldsymbol{i}|}$ divides $Q^{[\boldsymbol{i}]}(X, \boldsymbol{R})$.

Assume first that $\boldsymbol{0} \in \mathbb{K}^{s+1}$ is a root of Q of order at least m. Then, $Q(X, \boldsymbol{Y}) = \sum_{\boldsymbol{j}} Q_{\boldsymbol{j}} \boldsymbol{Y}^{\boldsymbol{j}}$ has only monomials of total degree at least m, so that for $\boldsymbol{j} \geqslant \boldsymbol{i}$, each nonzero $Q_{\boldsymbol{j}} \boldsymbol{Y}^{\boldsymbol{j}-\boldsymbol{i}}$ has only monomials of total degree at least $m - |\boldsymbol{i}|$. Now, $\boldsymbol{R}(0) = \boldsymbol{0} \in \mathbb{K}^s$ implies that X divides each component of $\boldsymbol{R}$. Consequently, $X^{m-|\boldsymbol{i}|}$ divides $Q_{\boldsymbol{j}} \boldsymbol{R}^{\boldsymbol{j}-\boldsymbol{i}}$ for each $\boldsymbol{j} \geqslant \boldsymbol{i}$, and thus $Q^{[\boldsymbol{i}]}(X, \boldsymbol{R})$ as well.

Conversely, let us assume that for all $\boldsymbol{i}$ in $\mathbb{N}^s$ such that $|\boldsymbol{i}| < m$, $X^{m-|\boldsymbol{i}|}$ divides $Q^{[\boldsymbol{i}]}(X, \boldsymbol{R})$, and show that Q has no monomial of total degree less than m. Writing the Taylor expansion of Q with $\mathbb{A} = \mathbb{K}[X]$ and $\boldsymbol{a} = \boldsymbol{R}$, we obtain

$$Q(X, \boldsymbol{Y}) = \sum_{\boldsymbol{i}} Q^{[\boldsymbol{i}]}(X, \boldsymbol{R})(\boldsymbol{Y} - \boldsymbol{R})^{\boldsymbol{i}}.$$

Each component of $\boldsymbol{R}$ being a multiple of X, we deduce that for the multi-indices $\boldsymbol{i}$ such that $|\boldsymbol{i}| \geqslant m$ every nonzero monomial in $Q^{[\boldsymbol{i}]}(X, \boldsymbol{R})(\boldsymbol{Y} - \boldsymbol{R})^{\boldsymbol{i}}$ has total degree

at least m. Using our assumption, the same conclusion follows for the multi-indices such that $|\boldsymbol{i}| < m$. □

For $\boldsymbol{i}$ in $\mathbb{N}^s$, with $|\boldsymbol{i}| < m$, define the polynomials

$$P_{\boldsymbol{i}} = G^{m-|\boldsymbol{i}|}$$

as well as $\boldsymbol{F}_{\boldsymbol{i}} = (F_{\boldsymbol{i},\boldsymbol{j}})_{\boldsymbol{j}\in\Gamma}$, with

$$F_{\boldsymbol{i},\boldsymbol{j}} \quad = \quad \binom{\boldsymbol{j}}{\boldsymbol{i}} \boldsymbol{R}^{\boldsymbol{j}-\boldsymbol{i}} \mod P_{\boldsymbol{i}} \quad = \quad \binom{j_1}{i_1} R_1^{j_1-i_1} \cdots \binom{j_s}{i_s} R_s^{j_s-i_s} \mod P_{\boldsymbol{i}}.$$

Then, the previous lemma implies that Q satisfies properties (ii)-(iv) if and only if it can be written as $Q = \sum_{\boldsymbol{j}\in\Gamma} Q_{\boldsymbol{j}} \boldsymbol{Y}^{\boldsymbol{j}}$, with $\deg(Q_{\boldsymbol{j}}) < N_{\boldsymbol{j}}$ for all $\boldsymbol{j}$ and, for all $\boldsymbol{i}$ in $\mathbb{N}^s$ such that $|\boldsymbol{i}| < m$,

$$\sum_{\boldsymbol{j}\in\Gamma} Q_{\boldsymbol{j}} F_{\boldsymbol{i},\boldsymbol{j}} \;=\; 0 \bmod P_{\boldsymbol{i}}.$$

The latter conditions express the problem of finding such a Q as an instance of Problem 2. In order to make the reduction completely explicit, define further

$$\mu = \binom{s+m-1}{s}, \quad \nu = |\Gamma|,$$

and choose arbitrary orders on the set of indices $\{\boldsymbol{i} \in \mathbb{N}^s \mid |\boldsymbol{i}| < m\}$ and Γ, that is, bijections

$$\phi : \{0, \ldots, \mu-1\} \to \{\boldsymbol{i} \in \mathbb{N}^s \mid |\boldsymbol{i}| < m\} \quad \text{and} \quad \psi : \{0, \ldots, \nu-1\} \to \Gamma.$$

To i in $\{0, \ldots, \mu-1\}$, we can then associate $M'_i = M_{\phi(i)}$, and similarly to j in $\{0, \ldots, \nu-1\}$ we associate $N'_j = N_{\psi(j)}$; we also set $P'_i = P_{\phi(i)}$ and $F'_{i,j} = F_{\phi(i),\psi(j)}$.

Proposition 9. *Let (s, ℓ, m, n, k, b) be parameters for Problem 1, and let the parameters $\mu, \nu, \boldsymbol{M}' = (M'_0, \ldots, M'_{\mu-1}), \boldsymbol{N}' = (N'_0, \ldots, N'_{\nu-1})$ be as above.*

Then, one can reduce an instance of Problem 1 with parameters (s, ℓ, m, n, k, b) to an instance of Problem 2 with parameters $(\mu, \nu, \boldsymbol{M}', \boldsymbol{N}')$ and input polynomials $(P'_i, \boldsymbol{F}'_i)$ using $\mathcal{O}(r\mathsf{M}(M)\log(M))$ operations in $\mathbb{K}$, where we write $r = \max(\binom{s+m-1}{s}, |\Gamma|) = \max(\mu, \nu)$.

Proof. The only thing left to do is the complexity analysis. First, we need to compute $P_{\boldsymbol{i}} = G^{m-|\boldsymbol{i}|}$ for every $\boldsymbol{i}$ such that $|\boldsymbol{i}| < m$. This involves only m different polynomials,

namely $G, \ldots, G^m$, so it can be done using $\mathcal{O}(m\mathsf{M}(mn))$ operations; this will be dominated by the cost of the third step below.

Then, we have to compute the interpolation polynomials $\boldsymbol{R} = (R_1, \ldots, R_s)$(using Lagrange Interpolation). Each of them can be computed in $\mathcal{O}(\mathsf{M}(n)\log(n))$ operations in $\mathbb{K}$, for a total of $\mathcal{O}(s\mathsf{M}(n)\log(n))$, which is $\mathcal{O}(s\mathsf{M}(M)\log(M))$. We have $|\Gamma| \geqslant s$, so that this cost is $\mathcal{O}(r\mathsf{M}(M)\log(M))$, except in the extreme case when $\Gamma = \{(0, \ldots, 0)\}$, so that $|\Gamma| = 1$; in that case, however, we need not compute $\boldsymbol{R}$ and we can omit the cost of interpolation.

Finally, we compute $F_{\boldsymbol{i},\boldsymbol{j}} \bmod P_{\boldsymbol{i}}$ for every $\boldsymbol{i}, \boldsymbol{j}$. This is done by fixing $\boldsymbol{i}$ and computing all products $F_{\boldsymbol{i},\boldsymbol{j}} \bmod P_{\boldsymbol{i}}$ incrementally, starting from $R_1, \ldots, R_s$. Each product takes $\mathcal{O}(\mathsf{M}(M_{\boldsymbol{i}}))$ operations in $\mathbb{K}$. Summing over all $\boldsymbol{j}$ leads to a cost of $\mathcal{O}(|\Gamma|\mathsf{M}(M_{\boldsymbol{i}}))$ per index $\boldsymbol{i}$. Summing over all $\boldsymbol{i}$ and using the super-linearity of M leads to a total cost of $\mathcal{O}(|\Gamma|\mathsf{M}(M))$, which is $\mathcal{O}(r\mathsf{M}(M))$. □

Thus, in order to prove Theorem 4, it is enough to prove Theorem 3: in view of the cost reported in Theorem 3 for solving the linear system, the cost of reduction given above will always be negligible.

3.5 Solving Problem 2 through a mosaic-Hankel linear system

In this section, we give our first solution to Problem 2, thereby proving Theorem 3.

This approach extends to arbitrary s the derivation of Extended Key Equations presented in [32, 39] for $s = 1$; contrary to those references, however, we apply the algorithm of Section 3.3 to solve the resulting mosaic-Hankel linear system, since it features the best cost we are aware of for this task.

We consider input polynomials $(P_i, \boldsymbol{F}_i)_{0 \leqslant i < \mu}$ with, for all i, P_i monic of degree M'_i and $\boldsymbol{F}_i$ a vector of ν polynomials $(F_{i,0}, \ldots, F_{i,\nu-1})$, all of degree less than M'_i. Given degree bounds $N'_0, \ldots, N'_{\nu-1}$, we look for polynomials $\boldsymbol{Q} = (Q_0, \ldots, Q_{\nu-1})$ in $\mathbb{K}[X]$ such that the following holds:

(a) the Q_j's are not all zero

(b) for $0 \leqslant j < \nu$, $\deg(Q_j) < N'_j$,

(c) for $0 \leqslant i < \mu$, $\sum_{0 \leqslant j < \nu} Q_j F_{i,j} = 0 \bmod P_i$.

Our goal here is to linearize the problem into a linear system involving M' linear equations with N' unknowns. Let us start by making a few simplifying assumptions.

- We can assume without loss of generality that for all i in $\{0, \ldots, \mu - 1\}$, the vector of polynomials $\boldsymbol{F}_i$ is not identically zero: if that were the case, the corresponding equation would become $0 = 0$, and could be discarded.

- Without loss of generality, we may as well assume that $N' \leqslant M' + 1$. Indeed, if $N' \geqslant M' + 1$, the instance of Problem 2 we are considering has more unknowns than equations. We may set the last $N' - (M' + 1)$ unknowns to zero, while keeping the system underdetermined. This simply amounts to replacing the degree bounds $N'_0, \ldots, N'_{\nu-1}$ by $N'_0, \ldots, N'_{\nu'-2}, N''_{\nu'-1}$, for $\nu' \leqslant \nu$ and $N''_{\nu'-1} \leqslant N'_{\nu'-1}$ such that $N'_0 + \cdots + N'_{\nu'-2} + N''_{\nu'-1} = M' + 1$. In particular, ν may only decrease through this process.

In what follows, we will work with the reversals of the polynomials P_i and $\boldsymbol{F}_i$, defined by

$$\overline{P_i} = X^{M'_i} P_i(X^{-1}) \quad \text{and} \quad \overline{\boldsymbol{F}} = (\overline{F_{i,j}})_{i,j}, \quad \text{with} \quad \overline{F_{i,j}} = X^{M'_i - 1} F_{i,j}(X^{-1}).$$

Similarly, for j in $\{0, \ldots, \nu - 1\}$, we associate to the unknown polynomial Q_j its reversal $\overline{Q_j} = X^{N'_j - 1} Q_j(X^{-1})$. For i and j as above, we define further

$$\delta_i = M'_i + \max\{N'_j \mid j \in \{0, \ldots, \nu - 1\}, F_{i,j} \neq 0\} - 1 \quad \text{and} \quad \gamma_{i,j} = \delta_i - (M'_i + N'_j - 1) \geqslant 0.$$

Since all N'_j's are positive, and since by assumption we take the maximum of a non-empty set, the inequality $\delta_i \geqslant M'_i$ holds for all i. Finally, for all i, j, we define $S_{i,j}$ as

$$S_{i,j} = \frac{X^{\gamma_{i,j}} \overline{F_{i,j}}}{\overline{P_i}} \bmod X^{\delta_i};$$

we see $S_{i,j}$ as an element of $\mathbb{K}[X]$, which is valid since $\gamma_{i,j} \geqslant 0$ and $\overline{P_i}$ is a unit modulo X^{δ_i}.

Lemma 10. *Suppose that $\boldsymbol{Q} = (Q_0, \ldots, Q_{\nu-1})$ satisfies (b). Then, $\boldsymbol{Q}$ satisfies condition (c) if and only if for all i in $\{0, \ldots, \mu - 1\}$, there exists a polynomial T_i in $\mathbb{K}[X]$ such that*

$$\sum_{0 \leqslant j < \nu} \overline{Q_j} S_{i,j} = T_i \bmod X^{\delta_i} \qquad \textit{and} \qquad \deg(T_i) < \delta_i - M'_i. \tag{3.3}$$

Proof. Condition (c) holds if and only if for all i in $\{0, \ldots, \mu - 1\}$, there exists a polynomial B_i in $\mathbb{K}[X]$ such that

$$\sum_{0 \leqslant j < \nu} Q_j F_{i,j} = B_i P_i. \tag{3.4}$$

For all i, j, the summand $Q_j F_{i,j}$ has degree less than $N'_j + M'_i - 1$, so the left-hand term above has degree less than δ_i. Since P_i has degree M'_i, this implies that whenever a polynomial B_i as above exists, we must have $\deg(B_i) < \delta_i - M'_i$. Now, by substituting $1/X$ for X and multiplying by $X^{\delta_i - 1}$ we can rewrite the identity in (3.4) as

$$\sum_{0 \leqslant j < \nu} \overline{Q_j}\, \overline{F_{i,j}} X^{\gamma_{i,j}} = T_i \overline{P_i}, \tag{3.5}$$

where T_i is the polynomial of degree less than $\delta_i - M'_i$ given by $T_i = X^{\delta_i - M'_i - 1} B_i(X^{-1})$. Since the degrees of both sides of (3.5) are less than δ_i, one can consider the above identity modulo X^{δ_i} without loss of generality, and since $\overline{P_i}(0) = 1$ one can further divide by $\overline{P_i}$ modulo X^{δ_i}. This shows that (3.5) is equivalent to the identity in (3.3) and the proof is complete. □

Following [32, 39], we are going to rewrite the latter conditions as a linear system in the coefficients of the polynomials $\boldsymbol{Q}$, eliminating the unknowns T_i from the outset. Let us first define the *coefficient vector* of a vector $\boldsymbol{Q} = (Q_0, \ldots, Q_{\nu-1})$ that satisfies (b). For any such polynomials, and for any j in $\{0, \ldots, \nu - 1\}$, we denote by

$$x_j = \left[Q_j^{(0)}, Q_j^{(1)}, \ldots, Q_j^{(N'_j - 1)}\right]^T \in \mathbb{K}^{N'_j}$$

the vector of coefficients of Q_j and we define the *coefficient vector of* $\boldsymbol{Q}$ as the column vector in $\mathbb{K}^{N'}$ obtained by concatenating $x_0, \ldots, x_{\nu-1}$.

For i in $\{0, \ldots, \mu-1\}$ and j in $\{0, \ldots, \nu-1\}$, let also $S_{i,j}^{(0)}, S_{i,j}^{(1)}, \ldots$ be the coefficients of $S_{i,j}$. Using these coefficients, we define the Hankel matrix

$$A_{i,j} = [A_{i,j}^{u,v}]_{0 \leqslant u < M'_i, 0 \leqslant v < N'_j} = [S_{i,j}^{(u+v+\gamma_{i,j})}]_{0 \leqslant u < M'_i, 0 \leqslant v < N'_j} \in \mathbb{K}^{M'_i \times N'_j},$$

and the mosaic-Hankel matrix $A = [A_{i,j}]_{0 \leqslant i < \mu, 0 \leqslant j < \nu} \in \mathbb{K}^{M' \times N'}$.

Lemma 11. *A nonzero vector of $\mathbb{K}^{N'}$ is in the nullspace of A if and only if it is the coefficient vector of a solution $\boldsymbol{Q}$ to Problem 2.*

Proof. It is sufficient to consider a polynomial vector $\boldsymbol{Q}$ that satisfies (b). Then,

looking at the high-degree terms in equations (3.3), we see that condition (c) is equivalent to the following system of so-called Extended Key Equations: for all i in $\{0, \ldots, \mu - 1\}$ and all δ in $\{\delta_i - M'_i, \ldots, \delta_i - 1\}$,

$$\sum_{0 \leqslant j < \nu} \sum_{0 \leqslant r < N'_j} Q_j^{(N'_j - 1 - r)} S_{i,j}^{(\delta - r)} = 0.$$

The matrix obtained by considering all these equations is precisely A. □

We will use the algorithm of Section 3.3 to find a nullspace element for A, with respect to the displacement operator $\Delta'_{M',N'}$. Not only do we need to prove that the displacement rank of A with respect to $\Delta'_{M',N'}$ is not too large, we also have to compute generators for A, that is, matrices V and W such that $A - \mathcal{Z}_{M'} A \mathcal{Z}_{N'} = VW$. We will see that here, computing these generators boils down to computing the coefficients of the polynomials $S_{i,j}$.

The cost incurred by computing these generators is summarized in the following lemma; combined with Lemma 11 and Proposition 7, this proves Theorem 3.

Lemma 12. *The displacement rank of A with respect to $\Delta'_{M',N'}$ is at most $\mu + \nu$. Furthermore, one can compute generators for A using $\mathcal{O}\left((\mu + \nu)\mathsf{M}(M')\right)$ operations in $\mathbb{K}$.*

Proof. We are going to exhibit two matrices $V \in \mathbb{K}^{M' \times (\mu+\nu)}$ and $W \in \mathbb{K}^{(\mu+\nu) \times N'}$ such that $A - \mathcal{Z}_{M'} A \mathcal{Z}_{N'} = VW$. Because of the structure of A, at most μ rows and ν columns of the matrix $A - \mathcal{Z}_{M'} A \mathcal{Z}_{N'} = A - (A$ shifted left and down by one unit) are nonzero. More precisely, only the first row and the last column of each $(M'_i \times N'_j)$ block of this matrix can be nonzero. Indexing the rows, resp. columns, of $A - \mathcal{Z}_{M'} A \mathcal{Z}_{N'}$ from 0 to $M' - 1$, resp. from 0 to $N' - 1$, only the μ rows with indices of the form $r_i = \sum_{i' < i} M'_{i'}$, for $i = 0, \ldots, \mu - 1$, can be nonzero, and only the ν columns with indices of the form $c_j = -1 + \sum_{j' \leqslant j} N'_{j'}$, for $j = 0, \ldots, \nu - 1$, can be nonzero.

For any integers $0 \leqslant i < K$, define $\mathcal{O}_{i,K} = [0 \cdots 0\ 1\ 0 \cdots 0]^T \in \mathbb{K}^K$ with 1 at position i, and

$$\mathcal{O}^{(V)} = [\mathcal{O}_{r_i, M'}]_{0 \leqslant i < \mu} \in \mathbb{K}^{M' \times \mu}, \quad \mathcal{O}^{(W)} = \left[\mathcal{O}_{c_j, N'}\right]^T_{0 \leqslant j < \nu} \in \mathbb{K}^{\nu \times N'}.$$

For given i in $\{0, \ldots, \mu - 1\}$ and j in $\{0, \ldots, \nu - 1\}$, we will consider $v_{i,j} \in \mathbb{K}^{M'_i \times 1}$ and $w_{i,j} \in \mathbb{K}^{1 \times N'_j}$ which are respectively the last column and the first row of the block (i, j) in $A - \mathcal{Z}_{M'} A \mathcal{Z}_{N'}$, up to a minor point: the first entry of $v_{i,j}$ is set to zero. Their

coefficients

$$v_{i,j} = [v_{i,j}^{(r)}]_{0 \leqslant r < M'_i} \quad \text{and} \quad w_{i,j} = [w_{i,j}^{(r)}]_{0 \leqslant r < N'_j}^T$$

are as follows:

$$v_{i,j}^{(r)} = \begin{cases} 0 & \text{if } r = 0 \\ A_{i,j}^{r,N'_j-1} - A_{i,j+1}^{r-1,0} & \text{otherwise,} \end{cases}$$

$$w_{i,j}^{(r)} = \begin{cases} A_{i,j}^{0,r} - A_{i-1,j}^{M'_{i-1}-1,r+1} & \text{if } r < M'_i - 1 \\ A_{i,j}^{0,M'_i-1} - A_{i-1,j+1}^{M'_{i-1}-1,0} & \text{if } r = M'_i - 1. \end{cases}$$

Note that here, we use the convention that an indexed object is zero when the index is out of the allowed bounds for this object.

Then, we define V_j and W_i as

$$V_j = \begin{bmatrix} v_{0,j} \\ \vdots \\ v_{\mu-1,j} \end{bmatrix} \in \mathbb{K}^{M' \times 1} \quad \text{and} \quad W_i = [w_{i,0} \ \cdots \ w_{i,\nu-1}]_{0 \leqslant j < \nu} \in \mathbb{K}^{1 \times N'},$$

and

$$V' = [\cdots \ V_j \ \cdots]_{0 \leqslant j < \nu} \in \mathbb{K}^{M' \times \nu} \quad \text{and} \quad W' = \begin{bmatrix} \vdots \\ W_i \\ \vdots \end{bmatrix}_{0 \leqslant i < \mu} \in \mathbb{K}^{\mu \times N'}.$$

Now, one can easily verify that the matrices

$$V = \begin{bmatrix} V' & \mathcal{O}^{(V)} \end{bmatrix} \in \mathbb{K}^{M' \times (\mu+\nu)} \qquad \text{and} \qquad W = \begin{bmatrix} \mathcal{O}^{(W)} \\ W' \end{bmatrix} \in \mathbb{K}^{(\mu+\nu) \times N'}$$

are generators for A, that is, $A - \mathcal{Z}_M A \, \mathcal{Z}_N = VW$.

We notice that all we need to compute the generators V and W are the highest $M'_i + N'_j - 1$ coefficients of $S_{i,j}$ for every i in $\{0, \ldots, \mu - 1\}$ and j in $\{0, \ldots, \nu - 1\}$. Now, recall that

$$S_{i,j} = \frac{X^{\gamma_{i,j}} \overline{F_{i,j}}}{\overline{P_i}} \bmod X^{\delta_i} = \frac{X^{\delta_i - (M'_i + N'_j - 1)} \overline{F_{i,j}}}{\overline{P_i}} \bmod X^{\delta_i}.$$

Thus, the first $\delta_i - (M'_i + N'_j - 1)$ coefficients of $S_{i,j}$ are zero, and the last $M'_i + N'_j - 1$

coefficients of $S_{i,j}$ are the coefficients of

$$S^{\star}_{i,j} = \frac{\overline{F_{i,j}}}{\overline{P_i}} \bmod X^{M'_i+N'_j-1},$$

which can be computed in $\mathcal{O}(\mathsf{M}(M'_i + N'_j))$ operations in $\mathbb{K}$ by fast power series division. By expanding products, we see that $\mathsf{M}(M'_i + N'_j) = \mathcal{O}(\mathsf{M}(M'_i) + \mathsf{M}(N'_j))$. Summing the costs, we obtain an upper bound of the form

$$\mathcal{O}\left(\sum_{0\leqslant i<\mu}\sum_{0\leqslant j<\nu} \mathsf{M}(M'_i) + \mathsf{M}(N'_j)\right).$$

Using the super-linearity of M, this is in $\mathcal{O}(\nu\mathsf{M}(M') + \mu\mathsf{M}(N'))$. Since we assumed that $N' \leqslant M' + 1$, this is $\mathcal{O}((\mu + \nu)\mathsf{M}(M'))$. □

3.6 A direct solution to Problem 2

In this section, we propose an alternative solution to Problem 2 with the same asymptotic running time as in the previous section. As above, our input is the polynomials $(P_i, \boldsymbol{F}_i)_{0\leqslant i<\mu}$ and we look for polynomials $(Q_0, \ldots, Q_{\nu-1})$ in $\mathbb{K}[X]$ such that for $0 \leqslant i < \mu$, $\sum_{0\leqslant j<\nu} Q_j F_{i,j} = 0 \bmod P_i$, with the same degree constraints as previously.

In addition, we denote by $F^{(r)}_{i,j}$ and $P^{(r)}_i$ the coefficients of $F_{i,j}$ and P_i, respectively, and we define $\mathcal{C}_i$ as the $M'_i \times M'_i$ companion matrix of P_i; if B is a polynomial of degree less than M'_i with coefficient vector $\alpha \in \mathbb{K}^{M'_i}$, then the product $\mathcal{C}_i\alpha \in \mathbb{K}^{M'_i}$ is the coefficient vector of the polynomial $XB \bmod P_i$. Explicitly, we have

$$\mathcal{C}_i = \begin{bmatrix} 0 & 0 & \cdots & 0 & -P_i^{(0)} \\ 1 & 0 & \cdots & 0 & -P_i^{(1)} \\ 0 & 1 & \cdots & 0 & -P_i^{(2)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -P_i^{(M'_i-1)} \end{bmatrix}.$$

We are going to see that solving Problem 2 is equivalent to finding a nonzero solution to a homogeneous linear system whose matrix is $A' = (A'_{i,j}) \in \mathbb{K}^{M'\times N'}$, where for every $i < \mu$ and $j < \nu$, $A'_{i,j} \in \mathbb{K}^{M'_i\times N'_j}$ is a matrix which depends on the coefficients of $F_{i,j}$ and P_i. Without loss of generality, we make the same assumptions as in the previous section: for all i in $\{0, \ldots, \mu - 1\}$, the vector of polynomials $\boldsymbol{F}_i$ is not identically zero, and $N' \leqslant M' + 1$ holds.

For i, j as above and for $r \in \mathbb{N}$, let $\alpha_{i,j}^{(r)} \in \mathbb{K}^{M_i'}$ be the coefficient vector of the polynomial $X^r F_{i,j} \bmod P_i$, so that these vectors are given by

$$\alpha_{i,j}^{(0)} = \begin{bmatrix} F_{i,j}^{(0)} \\ \vdots \\ F_{i,j}^{(M_i'-1)} \end{bmatrix} \quad \text{and} \quad \alpha_{i,j}^{(r+1)} = \mathcal{C}_i \alpha_{i,j}^{(r)}.$$

Let then $A' = (A'_{i,j}) \in \mathbb{K}^{M' \times N'}$, where for every $i < \mu$ and $j < \nu$, the block $A'_{i,j} \in \mathbb{K}^{M_i' \times N_j'}$ is defined by

$$A'_{i,j} = \begin{bmatrix} \alpha_{i,j}^{(0)} & \cdots & \alpha_{i,j}^{(N_j'-1)} \end{bmatrix};$$

in particular, $A'_{i,j}$ is the matrix of the mapping $Q \mapsto QF_{i,j} \bmod P_i$, for Q of degree less than N_j'.

Consider now a vector of polynomials $\boldsymbol{Q} = (Q_0, \ldots, Q_{\nu-1})$ that satisfies the degree constraint (b). By construction, applying A' to the coefficient vector of $\boldsymbol{Q}$ outputs the coefficients of the remainders $\sum_{0 \leqslant j < \nu} Q_j F_{i,j} \bmod P_i$, for $i = 0, \ldots, \mu - 1$. This proves in particular that a nonzero vector of $\mathbb{K}^{N'}$ is in the nullspace of A' if and only if it is the coefficient vector of a solution $\boldsymbol{Q}$ to Problem 2.

The following lemma shows that A' possesses a Toeplitz-like structure, with displacement rank at most $\mu + \nu$. Together with Proposition 7, this gives our second proof of Theorem 3.

Lemma 13. *The displacement rank of A' with respect to $\Delta_{M',N'}$ is at most $\mu + \nu$. Furthermore, one can compute generators for A' using $\mathcal{O}((\mu+\nu)\mathsf{M}(M'))$ operations.*

Proof. We are going to exhibit matrices $Y \in \mathbb{K}^{M' \times (\mu+\nu)}$ and $Z \in \mathbb{K}^{(\mu+\nu) \times N'}$ such that $A' - \mathcal{Z}_{M'} A' \mathcal{Z}_{N'}^T = YZ$. Define first the matrix

$$\mathcal{C} = \begin{bmatrix} \mathcal{C}_0 & 0 & \cdots & 0 \\ 0 & \mathcal{C}_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathcal{C}_{\mu-1} \end{bmatrix} \in \mathbb{K}^{M' \times M'}.$$

Up to μ columns, $\mathcal{C}$ coincides with $\mathcal{Z}_{M'}$; we make this explicit as follows. For i in

$\{0, \ldots, \mu - 1\}$, define

$$v_i = \begin{bmatrix} P_i^{(0)} \\ \vdots \\ P_i^{(M'_i-1)} \end{bmatrix} \in \mathbb{K}^{M'_i}, \quad V_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ v_i \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{K}^{M'} \quad \text{and} \quad W_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{K}^{M'},$$

with the coefficient 1 in V_i and W_i at respective indices $\sum_{i' \leqslant i} M'_{i'}$ and $-1+\sum_{i' \leqslant i} M'_{i'}$. Then, if we define $V = [V_0 \cdots V_{\mu-1}] \in \mathbb{K}^{M' \times \mu}$ and $W = [W_0 \cdots W_{\mu-1}] \in \mathbb{K}^{M' \times \mu}$, we obtain

$$\mathcal{C} = \mathcal{Z}_{M'} - V_0 W_0^T - \cdots - V_{\mu-1} W_{\mu-1}^T \; = \; \mathcal{Z}_{M'} - V W^T.$$

As before, we use the convention that an indexed object is zero when the index is out of the allowed bounds for this object. For j in $\{0, \ldots, \nu - 1\}$, let us further define

$$V'_j = \begin{bmatrix} \alpha_{0,j}^{(0)} \\ \vdots \\ \alpha_{\mu-1,j}^{(0)} \end{bmatrix} - \begin{bmatrix} \alpha_{0,j-1}^{(N'_{j-1})} \\ \vdots \\ \alpha_{\mu-1,j-1}^{(N'_{j-1})} \end{bmatrix} \in \mathbb{K}^{M'} \quad \text{and} \quad W'_j = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{K}^{N'},$$

with the coefficient 1 in W'_j at index $\sum_{j'<j} N'_{j'}$, and the compound matrices

$$V' = [V'_0 \cdots V'_{\nu-1}] \in \mathbb{K}^{M' \times \nu} \quad \text{and} \quad W' = [W'_0 \cdots W'_{\nu-1}] \in \mathbb{K}^{N' \times \nu}.$$

Then, we claim that matrices

$$Y = [\, -V \quad V' \,] \in \mathbb{K}^{M' \times (\mu+\nu)} \quad \text{and} \quad Z = \begin{bmatrix} W^T A' \mathcal{Z}_{N'}^T \\ W'^T \end{bmatrix} \in \mathbb{K}^{(\mu+\nu) \times N'}$$

are generators for A' for the Toeplitz-like displacement structure, *i.e.*, that

$$A' - \mathcal{Z}_{M'}\, A'\, \mathcal{Z}_{N'}^T = YZ.$$

By construction, we have $\mathcal{C}\, A' = (B_{i,j})_{i<\mu, j<\nu} \in \mathbb{K}^{M'\times N'}$, with $B_{i,j}$ given by

$$B_{i,j} \;=\; \mathcal{C}_i A'_{i,j} \;=\; \left[\alpha_{i,j}^{(1)} \;\cdots\; \alpha_{i,j}^{N'_j-1}\; \alpha_{i,j}^{(N'_j)}\right] \in \mathbb{K}^{M'_i\times N'_j}.$$

As a consequence, $A' - \mathcal{C}\, A'\, \mathcal{Z}_{N'}^T = V'W'^T$, so finally we get, as claimed,

$$\begin{aligned} A' - \mathcal{Z}_{M'}\, A'\, \mathcal{Z}_{N'}^T &= A' - (\mathcal{C} + VW^T)A'\mathcal{Z}_{N'}^T \\ &= A' - \mathcal{C}\, A'\, \mathcal{Z}_{N'}^T \;-\; VW^T A'\mathcal{Z}_{N'}^T \\ &= V'W'^T \;-\; VW^T A\, \mathcal{Z}_{N'}^T \\ &= YZ. \end{aligned}$$

To compute Y and Z, the only non-trivial steps are those giving V' and W^TA'. For the former, we have to compute the coefficients of $X^{N'_j}F_{i,j} \bmod P_i$ for all every $i < \mu$ and $j < \nu$. For fixed i and j, this can be done using fast Euclidean division in $\mathcal{O}(\mathsf{M}(M'_i + N'_j))$ operations in $\mathbb{K}$, which is $\mathcal{O}(\mathsf{M}(M'_i) + \mathsf{M}(N'_j))$. Summing over the indices $i < \mu$ and $j < \nu$, this gives a total cost of $\mathcal{O}(\nu\mathsf{M}(M') + \mu\mathsf{M}(N'))$ operations. Since $N' \leqslant M' + 1$, this is $\mathcal{O}((\mu+\nu)\mathsf{M}(M'))$.

Finally, we show that W^TA' can be computed using $\mathcal{O}((\mu+\nu)\mathsf{M}(M'))$ operations as well. Computing this matrix amounts to computing the rows of A' of indices $-1 + \sum_{i'\leqslant i} M'_{i'}$, for $i < \mu$. By construction of A', this means that we want to compute the coefficients of degree $M'_i - 1$ of $X^rF_{i,j} \bmod P_i$ for $r = 0, \ldots, N'_j - 1$ and for all i, j. Unfortunately, the naive approach leads to a cost proportional to $M'N'$ operations, which is not acceptable. However, for i and j fixed, Lemma 14 below shows how to do this computation using only $\mathcal{O}(\mathsf{M}(M'_i) + \mathsf{M}(N'_j))$ operations, which leads to the announced cost by summing over i and j. □

Lemma 14. *Let $P \in \mathbb{K}[X]$ be monic of degree m, let $F \in \mathbb{K}[X]$ be of degree less than m, and for $i \geqslant 0$ let c_i denote the coefficient of degree $m-1$ of $X^iF \bmod P$. For $n \geqslant 1$ we can compute $c_0, \ldots, c_{n-1}$ using $\mathcal{O}(\mathsf{M}(m) + \mathsf{M}(n))$ operations in $\mathbb{K}$.*

Proof. Writing $F = \sum_{0\leqslant j<m} f_jX^j$ we have $X^iF \bmod P = \sum_{0\leqslant j<m} f_j(X^{i+j} \bmod P)$. Hence $c_i = \sum_{0\leqslant j<m} f_jb_{i+j}$, with b_i denoting the degree $m-1$ coefficient of $X^i \bmod P$. Since $b_0 = \cdots = b_{m-2} = 0$ and $b_{m-1} = 1$, we can deduce $c_0, \ldots, c_{n-1}$ from

$b_{m-1}, \ldots, b_{m+n-2}$ in time $\mathcal{O}(\mathsf{M}(n))$ by multiplication by the lower triangular Toeplitz matrix $[f_{m+j-i}]_{i,j}$ of order $n-1$.

Thus, we are left with the question of computing $b_{m-1}, \ldots, b_{m+n-2}$. Writing P as $P = \sum_{0 \leqslant j < m} p_j X^j + X^m$ and using the fact that $X^i P \bmod P = 0$ for all $i \geqslant 0$, we see that the b_i's are generated by a linear recurrence of order m with constant coefficients:

$$\sum_{0 \leqslant j < m} p_j b_{i+j} + b_{i+m} = 0 \quad \text{for all } i \geqslant 0.$$

Consequently, $b_m, \ldots, b_{m+n-2}$ can be deduced from $b_0, \ldots, b_{m-1}$ in time $\mathcal{O}(\frac{n}{m}\mathsf{M}(m))$, which is $\mathcal{O}(\mathsf{M}(m)+\mathsf{M}(n))$, by $\lceil \frac{n-1}{m} \rceil$ calls to Shoup's algorithm for extending a linearly recurrent sequence [33, Theorem 3.1]. □

Acknowledgments. M.F.I.C. and É.S. were supported by NSERC and by the Canada Research Chairs program. We thank the three reviewers for their helpful comments, and especially the second one for suggesting a shorter proof of Lemma 14.

Bibliography

[1] M. Alekhnovich. Linear diophantine equations over polynomials and soft decoding of Reed-Solomon codes. *IEEE Transactions on Information Theory*, 51(7):2257–2265, July 2005.

[2] B. Beckermann. A reliable method for computing M-Padé approximants on arbitrary staircases. *J. Comput. Appl. Math.*, 40(1):19–42, 1992.

[3] Bernhard Beckermann and George Labahn. A uniform approach for the fast computation of matrix-type Padé approximants. *SIAM J. Matrix Anal. Appl.*, 15(3):804–823, July 1994.

[4] P. Beelen and K. Brander. Key equations for list decoding of Reed-Solomon codes and how to solve them. *Journal of Symbolic Computation*, 45(7):773–786, 2010.

[5] D. J. Bernstein. Simplified high-speed high-distance list decoding for alternant codes. In *PQCrypto'11*, pages 200–216, Berlin, Heidelberg, 2011. Springer-Verlag.

[6] R. R. Bitmead and B. D. O. Anderson. Asymptotically fast solution of Toeplitz and related systems of linear equations. *Linear Algebra Appl.*, 34:103–116, 1980.

[7] A. Bostan, C.-P. Jeannerod, and É. Schost. Solving structured linear systems with large displacement rank. *Theor. Comput. Sci.*, 407(1-3):155–181, November 2008.

[8] K. Brander. *Interpolation and List Decoding of Algebraic Codes*. PhD thesis, Technical University of Denmark, 2010.

[9] P. Busse. *Multivariate List Decoding of Evaluation Codes with a Gröbner Basis Perspective*. PhD thesis, University of Kentucky, 2008.

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

[11] H. Cohn and N. Heninger. Ideal forms of Coppersmith's theorem and Guruswami-Sudan list decoding. In Bernard Chazelle, editor, *ICS*, pages 298–308. Tsinghua University Press, 2011.

[12] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comp.*, 9(3):251–280, 1990.

[13] G. L. Feng and K. K. Tzeng. A generalization of the Berlekamp-Massey algorithm for multisequence shift-register synthesis with applications to decoding cyclic codes. *IEEE Transactions on Information Theory*, 37(5):1274–1287, 1991.

[14] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra (second ed.)*. Cambridge University Press, 2003.

[15] P. Giorgi, C.-P. Jeannerod, and G. Villard. On the complexity of polynomial matrix computations. In *Proceedings of the 2003 international symposium on Symbolic and algebraic computation*, ISSAC '03, pages 135–142, New York, NY, USA, 2003. ACM.

[16] S. Gupta, S. Sarkar, A. Storjohann, and J. Valeriote. Triangular x-basis decompositions and derandomization of linear algebra algorithms over $K[x]$. *J. Symb. Comput.*, 47(4):422–453, April 2012.

[17] V. Guruswami and A. Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions on Information Theory*, 54(1):135–150, 2008.

[18] V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999.

[19] H. Hasse. Theorie der höheren Differentiale in einem algebraischen Funktionenkörper mit vollkommenem Konstantenkörper bei beliebiger Charakteristik. *Journal für die reine und angewandte Mathematik*, 175:50–54, 1936.

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

[21] R. Kötter. Fast generalized minimum-distance decoding of algebraic-geometry and Reed-Solomon codes. *IEEE Transactions on Information Theory*, 42(3):721 –737, may 1996.

[22] K. Lee and M. E. O'Sullivan. List decoding of Reed-Solomon codes from a Gröbner basis perspective. *Journal of Symbolic Computation*, 43(9):645 – 658, 2008.

[23] R. J. McEliece. The Guruswami-Sudan decoding algorithm for Reed-Solomon codes, 2003. IPN Progress Report 42-153.

[24] H. M. Möller and Buchberger B. The construction of multivariate polynomials with preassigned zeros. In *EUROCAM'82*, volume 144 of *Lecture Notes in Computer Science*, pages 24–31. Springer, 1982.

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

[26] R. R. Nielsen and T. Høholdt. Decoding Reed-Solomon codes beyond half the minimum distance. In *Coding Theory, Cryptography and Related Areas*, pages 221–236. Springer-Verlag, 2000.

[27] V. Olshevsky and M. A. Shokrollahi. A displacement approach to efficient decoding of algebraic-geometric codes. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, STOC '99, pages 235–244, New York, NY, USA, 1999. ACM.

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

[29] F. Parvaresh and A. Vardy. Correcting errors beyond the Guruswami-Sudan radius in polynomial time. In *FOCS*, pages 285–294, 2005.

[30] J.-R. Reinhard. Algorithme LLL polynomial et applications. Master's thesis, École Polytechnique, Paris, France, 2003.

[31] R. M. Roth. *Introduction to Coding Theory*. Cambridge University Press, 2007.

[32] R. M. Roth and G. Ruckenstein. Efficient decoding of Reed-Solomon codes beyond half the minimum distance. *IEEE Transactions on Information Theory*, 46(1):246 –257, January 2000.

[33] V. Shoup. A fast deterministic algorithm for factoring polynomials over finite fields of small characteristic. In *ISSAC'91*, pages 14–21. ACM, 1991.

[34] Arne Storjohann. Notes on computing minimal approximant bases. In Wolfram Decker, Mike Dewar, Erich Kaltofen, and Stephen Watt, editors, *Challenges in Symbolic Computation Software*, number 06271 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[35] A. Stothers. *On the Complexity of Matrix Multiplication*. PhD thesis, University of Edinburgh, 2010.

[36] M. Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *J. Complexity*, 13(1):180–193, March 1997.

[37] P. V. Trifonov. Efficient interpolation in the Guruswami-Sudan algorithm. *IEEE Transactions on Information Theory*, 56(9):4341 –4349, September 2010.

[38] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 887–898. ACM, 2012.

[39] A. Zeh, C. Gentner, and D. Augot. An interpolation procedure for list decoding Reed-Solomon codes based on generalized key equations. *IEEE Transactions on Information Theory*, 57(9):5946–5959, September 2011.

Chapter 4

Efficient Solution of Structured Linear Systems

4.1 Introduction

The interpolation step of list decoding for (folded) Reed Solomon codes essentially needs to solve a linear system to obtain a multivariate polynomial; in the previous chapter, we saw how such systems can be solved efficiently by structured linear system solvers. In this chapter, we give more details on such algorithms, and present the first implementation of a central subroutine, structured matrix multiplication.

In [1], Bitmap and Anderson gave an algorithm to solve "Toeplitz-like"linear systems (see the definition below) efficiently using the notion of displacement operators and displacement rank; Morf gave essentially the same algorithm in [7], so this algorithm is known as the MBA algorithm.

In 1994, Kaltofen gave a variant of the MBA algorithm [6] where he introduced randomization to remove some previously needed genericity assumptions on the input matrix. The complexity of Kaltofen's version of the MBA algorithm is $O(\alpha^2\mathsf{M}(n)\log n)$ where α is the displacement rank of the system we want to solve, n is its size and M denotes a function such that univariate polynomials of degree at most n can be multiplied in $\mathsf{M}(n)$ operations in the base field. Later on, Bostan, Jeannerod and Schost gave another variant of the MBA algorithm in [3] with a complexity $O(\alpha^{\omega-1}\mathsf{M}(n)\log^2 n)$, where $\omega \leqslant 3$ is the exponent of matrix multiplication, that is, is such that matrices of size α can be multiplied in $O(\alpha^\omega)$ base field operations. Using for instance Strassen's matrix multiplication algorithm, or many subsequent better

algorithms yielding $\omega < 3$, this latter result is thus better when α is large compared to n.

All these variants are required to "compress" structured matrices, that is, to minimize the size of the internal representation of some intermediate matrices. Though this step does not effect the asymptotic complexity, timings can be greately improved if compression can be avoided. Jeannerod and Mouilleron showed in [5] how to modify the algorithm so that no compression is required.

Finally, in a most recent preprint [2], Bostan, Jeannerod, Mouilleron and Schost gave an algorithm having complexity $O(\alpha^{\omega-1}\mathsf{M}(n)\log n)$ for solving Toeplitz-like structuted systems of size n with displacement rank α; this is unconditionally better than Kaltofen's, at least asymptotically.

In this chapter, we present in detail the algorithm of [2] for solving structured linear systems, and we report on the first implementation we are aware of a critical step, structured matrix multiplication. For all standard results used but not proved below, one may consult [9].

4.2 Basics on structured linear systems

Let $\mathbb{F}$ be our base field. In what follow, we introduce the basic concepts related to structured linear systems (displacement rank, displacement operators, generators), for the *Toeplitz* structure, which is one of the most important in practice.

An $n \times n$ Toeplitz matrix, with coefficients in $\mathbb{F}$, is a matrix of the following shape:

$$T = \begin{bmatrix} t_0 & t_{-1} & t_{-2} & \cdots & \cdots & t_{-n+1} \\ t_1 & t_0 & t_{-1} & \ddots & & \vdots \\ t_2 & t_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & t_{-1} & t_{-2} \\ \vdots & & \ddots & t_1 & t_0 & t_{-1} \\ t_{n-1} & \cdots & \cdots & t_2 & t_1 & t_0 \end{bmatrix} \in \mathbb{F}^{n \times n}.$$

It has been known for long that one can solve the linear system $Tu = v$ in quasi-linear time in n. The idea behind algorithms for structured linear systems amounts to obtain similar quasi-linear results for matrices A that are "close" to being Toeplitz matrices. Although our example above was square, this is actually not necessary, so we may consider matrices of size $m \times n$ below.

We measure how close, or far, a matrix $A \in \mathbb{F}^{m\times n}$ is to being Toeplitz using its *displacement rank*, that is, the rank of its image under a *displacement operator*.

There exist two broad classes of displacement operators that can be used to define the displacement rank: *Sylvester* operators, of the form

$$\nabla_{M,N} : A \in \mathbb{F}^{m\times n} \mapsto MA - AN \in \mathbb{F}^{m\times n},$$

for some fixed square matrices M and N, of respective sizes m and n, and *Stein* operators of the form

$$\Delta_{M,N} : A \in \mathbb{F}^{m\times n} \mapsto A - MAN \in \mathbb{F}^{m\times n},$$

with the same constraints on M and N. For such an operator, say $\mathcal{L}$, the rank α of $\mathcal{L}(A)$ is known as the $\mathcal{L}$-*displacement rank* of A and A is called structured if its $\mathcal{L}$-displacement rank of A, α is small compared to m and n.

Two matrices $G, H \in \mathbb{F}^{m\times\alpha'}$ are called a $\mathcal{L}$-*generator* of A if $\mathcal{L}(A) = GH^t$ where H^t is the transpose of H (remark that $\alpha \leqslant \alpha'$, but $\alpha < \alpha'$ is possible). Provided $\mathcal{L}$ is invertible, so that we can recover A from $\mathcal{L}(A)$, such generator matrices form a compact representation of matrix A. Remark that for all notation introduced above, when the operator $\mathcal{L}$ is clear from the context, we will omit it.

Let us consider the particular case of the Stein operator with $m = n$, M an $n \times n$ matrix of the form

$$M = \begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix} \in \mathbb{F}^{n\times n}$$

and $N = M^t$. Then for the Toeplitz matrix $T \in \mathbb{F}^{n\times n}$ seen above, and for the Stein operator $\Delta_{M,N}$, we have

$$\Delta_{M,N}(T) = T - MTN = GH^t, \tag{4.1}$$

where $G, H \in \mathbb{F}^{n\times\alpha}$ are as follows:

$$G = \begin{bmatrix} t_0 & 1 \\ t_1 & 0 \\ \vdots & \vdots \\ t_{n-1} & 0 \end{bmatrix} \quad H = \begin{bmatrix} 1 & 0 \\ 0 & t_{-1} \\ \vdots & \vdots \\ 0 & t_{-n+1} \end{bmatrix};$$

in particular, the displacement rank of a Toeplitz matrix for this operator is at most 2 (and exactly 2 in general). Besides, this operator is invertible: letting (G_j) and (H_j) be the columns of respectively G and H, the quation

$$\Delta_{M,N}(A) = GH^t$$

admits the unique solution for matrix A given by

$$A = \sum_{j=1}^{\alpha} L[G_j]\ U[H_j^t], \tag{4.2}$$

where $L[G_j]$ is a lower triangular Toeplitz matrix that have G_j as its first column and $U[H_j^t]$ is a upper triangular Toeplitz matrix having H_j^t as its first row. This shows in particular how we can compute products of the form Au, for a vector u, or AB, for a matrix B, from the knowledge of the generators of A only, and u or B.

To analyze the complexity of this operation, it will be convenient to rewrite the above equality, introducing the antidiagonal matrix Z of size n, such that AZ coincides with A with columns in the reverse order, and ZA coincides with A with rows in the reverse order. Remark that for all j, we have

$$Z\ L[G_j]\ Z = U[G_j^t] \quad \text{and} \quad Z\ U[H_j^t]\ Z = L[H_j];$$

since Z^2 is the identity matrix, we obtain

$$A = Z \sum_{j=1}^{\alpha} U[G_j^t]\ L[H_j]\ Z. \tag{4.3}$$

Multiplying by Z is free (it is just a reversal), so we can focus on the computation of a product of the form

$$u \mapsto v = \sum_{j=1}^{\alpha} U[G_j^t]\ L[H_j]\ u,$$

where u is a vector of length n. For $j = 1, \ldots, \alpha$, let γ_j be the polynomial whose coefficients are the entries of G_j in the reverse order; let as well η_j be the polynomial whose coefficients are the entries of H_j. Finally, let f be the polynomial whose coefficients are the entries of u. Then, the entries of v are seen to be the coefficients of the polynomial

$$\sum_{j=1}^{\alpha} \gamma_j(\eta_j f \bmod x^n) \operatorname{div} x^n.$$

In terms of complexity, if we let M be such that polynomials of degree n can be multiplied in $\mathsf{M}(n)$ operations, we deduce from this formula that the product Au, for u a vector of length n, can be done in $O(\alpha\mathsf{M}(n))$ operations. If we multiply by a matrix B of size $n \times \beta$, the cost thus becomes $O(\alpha\beta\mathsf{M}(n))$. The same approach, up to minor differences, shows that we can compute products of the form $A^t u$ or $A^t B$ in the same amount of time.

This result allows us in particular to multiply matrices given by their generators. The following is taken from [6, Prop. 2]; it applies to the operator $\Delta_{M,N}$, although extensions exist to further structures. Suppose that A, A' are matrices of size n, that A is given by means of generators (G, H) and that A' is given by means of generators (G', H'). Then, $B = AA'$ admits the generators (P, Q), with

$$P = [G \mid MAM^t G' \mid -g] \qquad Q = [A'^t H \mid H' \mid h],$$

where g is the last column of MA and h is the last column of MA'^t. In particular, if A and A' have displacement ranks at most α, then AA' has displacement rank at most $2\alpha + 1$. Besides, using Equation (4.3), we can compute generators for it by multiplying A or A' by suitable matrices of size $n \times \alpha$ on the left or on the right. Using the remark in the previous paragraph on the cost of multiplication of A by a vector, we see that we can obtain P and Q in $O(\alpha^2\mathsf{M}(n))$ operations in $\mathbb{F}$. Section 4.4 will give a better algorithm for this task.

4.3 Structured matrix inversion

If A is structured with respect to either a Sylvester operator or a Stein operator, say $\mathcal{L}$, and A is invertible then its inverse is structured with respect to the operator $\mathcal{L}'$ defined by

$$\mathcal{L} = \nabla_{M,N} \Rightarrow \mathcal{L}' = \nabla_{N,M}, \quad \mathcal{L} = \Delta_{M,N} \Rightarrow \mathcal{L}' = \Delta_{N,M};$$

in other words, the roles of M and N are exchanged. This claims says that the ranks of $\mathcal{L}(A)$ and $\mathcal{L}'(A^{-1})$ are the same [9, Th. 1.5.1].

In what follows, we will restrict our presentation of the MBA algorithm to invertible matrices A; the extension to arbitrary matrices can be found in [6]. Then, all variants of the MBA algorithm compute $\mathcal{L}'$-generator matrices for A^{-1} from $\mathcal{L}$-generators matrices for A. We consider the Toeplitz structure in the following section, taking $\mathcal{L} = \Delta_{M,N}$ for M, N as in the previous section.

For dense matrices, it is known that matrix inversion can be reduced to matrix

multiplication, by means of block Gaussian elimination; we review this algorithm in the first subsection. In the second subsection, we give an overview of the MBA algorithm, which can be seen as a structured version of block Gaussian elimination, where all matrices are represented by means of their generators. In particular, this will show that structured matrix inversion can be reduced to structured matrix multiplication (which will be the focus of the next section).

4.3.1 Matrix inversion using block Gaussian elimination

Let $A \in \mathbb{F}^{n\times n}$ be an invertible matrix. Assume further that n_1 and n_2 are positive integers such that $n = n_1 + n_2$. Then, we can partition the matrix A as follows:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \in \mathbb{F}^{n\times n} \quad \text{where} \quad A_{ij} \in \mathbb{F}^{n_i \times n_j} \quad \text{for} \quad i,j \in \{1,2\}.$$

Suppose that A_{11} itself is invertible. Then, we define the Schur complement of A_{11} in A as

$$S = A_{22} - A_{21}A_{11}^{-1}A_{12}.$$

Then, the inverse of A is

$$A^{-1} = \begin{bmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}S^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}S^{-1} \\ -S^{-1}A_{21}A_{11}^{-1} & S^{-1} \end{bmatrix} \in \mathbb{F}^{n\times n}.$$

Define further two block matrices E and F as

$$E = \begin{bmatrix} \mathbb{I}_{n_1} & 0 \\ -A_{21}A_{11}^{-1} & \mathbb{I}_{n_2} \end{bmatrix} \quad \text{and} \quad F = \begin{bmatrix} \mathbb{I}_{n_1} & -A_{11}^{-1}A_{12} \\ 0 & \mathbb{I}_{n_2} \end{bmatrix},$$

where $\mathbb{I}_s$ is the identity matrix of size s. Note that E and F are nonsingular and that

$$EAF = \begin{bmatrix} A_{11} & \\ & S \end{bmatrix},$$

so S is nonsingular if A_{11} and A are nonsingular. Hence we can write a factorization of the inverse of A as follows [9, p. 157]

$$A^{-1} = F \begin{bmatrix} A_{11}^{-1} & \\ & S^{-1} \end{bmatrix} E.$$

This leads naturally to an algorithm that computes the inverse of A recursively, provided the upper left corners of A_{11} and S are invertible as well (and so on). One can prove that this is the case if and only if A is strongly regular, that is, all square upper-left matrices extracted from A are invertible.

When this is the case, the algorithm deduced from the formula above involves two recursive calls and a constant number of matrix multiplications in size at most n. Let ω be such that one can multiply matrices of size n in $O(n^\omega)$ operations in $\mathbb{F}$. Then, taking $n_1 = \lfloor n/2 \rfloor$, the cost $C(n)$ of computing A^{-1} satisfies

$$C(n) = 2C(\frac{n}{2}) + O(n^\omega),$$

which leads to $C(n) = O(n^\omega)$. In other words, the cost of matrix inversion is asymptotically the same as that of matrix multiplication, up to a constant factor.

4.3.2 Structured matrix inversion

Next, we describe the analogue of the previous algorithm for structured matrices; our matrices will thus be represented by means of their generators, as per our convention.

Let A be an $n \times n$ matrix with entries in $\mathbb{F}$; suppose furthermore that (G, H) are generators of A for the operator $\Delta_{M,N}$ given above. For simplicity, we are going to assume that n is even (making this assumption hold for further recursive calls will lead us to assume that n is actually a power of 2; all cases can be reduced to this case by padding dummy rows and columns); in the subdivision below, we directly take $n_1 = n_2 = n/2$. Divide A, operator matrices M and N, and the generator matrices G and H of A as follows:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad G = \begin{bmatrix} G_1 \\ G_2 \end{bmatrix}, \quad H = \begin{bmatrix} H_1 \\ H_2 \end{bmatrix},$$

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}, \qquad N = \begin{bmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{bmatrix},$$

where A_{ij}, M_{ij} and N_{ij} are matrices in $\mathbb{F}^{n/2 \times n/2}$ for $i, j \in \{1, 2\}$ and G_i and H_i are matrices in $\mathbb{F}^{n/2 \times \alpha}$ for $i \in \{1, 2\}$; in particular, M_{12} and N_{21} are the zero matrices of

size $n/2 \times n/2$, whereas M_{21} and N_{12} are the rank-one matrices

$$M_{21} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} 0 & \cdots & 0 & 1 \end{bmatrix} = U_2 V_1^t, \quad N_{12} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix} = U_1 V_2^t.$$

Finally, $M_{11} = M_{22}$ are similar matrices to M, but in size $n/2$, and similarly for $N_{11} = N_{22}$. Then we have the following property from [9, Proposition 4.4] or[8, Property 1.7]:

Property 15. *From $\nabla_{M,N}(A) = GH^t$ and the partitioning above, we have the following for $i, j \in \{1, 2\}$*

$$\nabla_{M_{ij},N_{ij}}(A_{ij}) = G_{ij} H_{ij}^t$$

where

$$\begin{aligned} G_{11} &= G_1 \in \mathbb{F}^{n/2 \times \alpha}, \\ H_{11} &= H_1 \in \mathbb{F}^{n/2 \times \alpha}, \end{aligned}$$

$$\begin{aligned} G_{12} &= [G_1 | A_{11} U_1] \in \mathbb{F}^{n/2 \times (\alpha+1)}, \\ H_{12} &= [H_2 | V_2] \in \mathbb{F}^{n/2 \times (\alpha+1)}, \end{aligned}$$

$$\begin{aligned} G_{21} &= [G_2 | -U_2] \in \mathbb{F}^{n/2 \times (\alpha+1)}, \\ H_{21} &= [H_1 | A_{11}^t V_1] \in \mathbb{F}^{n/2 \times (\alpha+1)}, \end{aligned}$$

$$\begin{aligned} G_{22} &= [G_2 | -U_2 | A_{21} U_1] \in \mathbb{F}^{n/2 \times (\alpha+2)} \\ H_{22} &= [H_2 | A_{12}^t V_1 | V_2] \in \mathbb{F}^{n/2 \times (\alpha+2)}. \end{aligned}$$

This property tells us that the submatrices of a structured matrix are also almost as structured as the initial matrix. Besides, one can compute the generators of all these submatrices by padding the initial generators with some "simple" vectors, typically obtained from particular rows or columns of A (which themselves can be obtained using Equation 4.3).

We can then present the MBA algorithm, which is a "structured" version of the

block Gaussian elimination algorithm, using the above formulas to extract generators of submatrices of A. Some steps involve multiplications of matrices: for all such steps, we know generators for the matrices involved, so we can apply the multiplication algorithm described at the end of Section 4.2.

Algorithm 1: MBA algorithm invert Toeplitz-like matrices

Require: $G, H \in \mathbb{F}^{n\times\alpha}$ such that $\Delta_{\mathbb{Z}_{n,0},\mathbb{Z}^t_{n,o}}(A) = GH^t$
Assumption: A is strongly regular
Ensure: Y and Z such that $\Delta_{\mathbb{Z}^t_{n,0},\mathbb{Z}_{n,o}}(A^{-1}) = YZ^t$.
1: Compute length-α generators G_{11}, H_{11} for A_{11}
2: $(Y_{11}, Z_{11}) \leftarrow \text{MBA}(G_{11}, H_{11})$ (those are generators for A_{11}^{-1})
3: Compute generators $(\widetilde{G_s}, \widetilde{H_s})$ for $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$
4: Deduce from $(\widetilde{G_s}, \widetilde{H_s})$ generators (G_s, H_s) for S of minimal length by compression
5: $(Y_s, Z_s) \leftarrow \text{MBA}(G_s, H_s)$
6: Compute generators for $-A_{11}^{-1}A_{12}S^{-1}$, $-S^{-1}A_{21}A_{11}^{-1}$ and $A_1 1 - 1 + A_{11}^{-1}A_{12}S^{-1}A_{21}A_{11}^{-1}$
7: Deduce generators $(\widetilde{Y}, \widetilde{Z})$ for $A^{-1} = \begin{bmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}S^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}S^{-1} \\ -S^{-1}A_{21}A_{11}^{-1} & S^{-1} \end{bmatrix}$
8: Deduce from $(\widetilde{Y}, \widetilde{Z})$ generators (Y, Z) for A^{-1} of minimal length by compression
9: **return** Y and Z

Throughout this algorithm, the length of the generators may increase (due to the multiplication algorithm mentioned above). It is possible however to reduce the size of these generators for some particular matrices: we know that A^{-1} admits generators of the same length α as A, and Kaltofen proved that it is also the case for the Schur complement S. Thus, the algorithm performs "compression steps" (which are done by linear algebra, see [6]) in order to maintain generators of length α for the recursive calls.

Thus, this algorithm performs two recursive calls in size $n/2$, with generators of the same length; all other operations are products or additions of structured matrices. We see in Section 4.2 that the latter operations can be done in $O(\alpha^2\mathsf{M}(n))$ operations in $\mathbb{F}$; this leads to the following recursion on the running time $C'(n)$ of this algorithm:

$$C'(n) = 2C'(\frac{n}{2}) + O(\alpha^2\mathsf{M}(n)),$$

so that $C'(n) = O(\alpha^2\mathsf{M}(n)\log(n))$, as mentioned before.

As seen above, one technical issue of this algorithm is that the length of generators computed in the intermediate steps grows as a result of successive additions and multiplications of structured matrices. This issue was explicitely studied in [8, 5];

the authors proposed an algorithm that systemetically avoids the two compression steps of algorithm 1. They also show how to get directly the generator matrices of intermediate submatrices required to do the multiplication [5, Lemmas 3, 4, 5]. These simplifications reduce the overall running time by a constant factor; we will not address them further in this thesis.

4.4 Structured matrix multiplication

In this section, we revisit the multiplication algorithm described in Section 4.2, presenting an improvement due to Bostan, Jeannerod, Mouilleron and Schost [2] and describe a first implementation of it.

Suppose we have to compute a product of the form AB, where A is a matrix of size n given by means of generators (G, H) of length α and B is a matrix of size $n \times \beta$. One main application is the multiplication of two structured matrices A and A' both of displacement rank at most α; as explained in Section 4.2, this mainly boils down to operations such as computing AB, where B is derived from the generators of A'. In particular, one should have in mind the case where $\beta \simeq \alpha$.

For $j = 1, \ldots, \alpha$, let G_j and H_j be the columns of respectively G and H; let further γ_j be the polynomial whose coefficients are the entries of G_j in the reverse order; let as well η_j be the polynomial whose coefficients are the entries of H_j. Finally, let $B_1, \ldots, B_\beta$ be the columns of B and for all i let f_i be the polynomial whose coefficients are the entries of B_i. Then, we have mentioned that the entries of AB_i are the coefficients of the polynomial

$$c_i = \sum_{k=1}^{\alpha} \gamma_k (\eta_k f_i \bmod x^n) \operatorname{div} x^n.$$

To compute c_i, we can sum it up first and then apply the division by x^n, so the core problem is to compute

$$r_i = \sum_{k=1}^{\alpha} \gamma_k (\eta_k f_i \bmod x^n)$$

for all i. The naive approach computes all sums independently and thus has with cost $O(\alpha^2 \mathsf{M}(n))$ assuming $\alpha = \beta$. The algorithm given below, taken from [2], reduces the complexity to $O(\alpha^{\omega-1} \mathsf{M}(n))$, where ω is the exponent of matrix multiplication.

The problem can be rephrased in terms of computations with matrices with poly-

nomial entries. Let

$$\mathsf{U} \in \mathbb{F}[x]^{\alpha\times 1} \quad \mathsf{V} \in \mathbb{F}[x]^{\alpha\times 1} \quad \mathsf{W} \in \mathbb{F}[x]^{\beta\times 1}$$

be the polynomial vectors whose entries are γ_i, η_i and f_i respectively. Then our problem is to compute the row vector $\mathsf{R} \in \mathbb{F}[x]^{1\times\beta}$ such that $\mathsf{R} = \mathsf{U}^{\mathsf{t}}(\mathsf{V}\mathsf{W}^t \bmod x^n)$. Let us assume $\alpha = \beta$ for simplicity. Then, the following lemma is the basis of the algorithm.

Lemma 16. *Let $\alpha, \mu, \nu \in \mathbb{N}_{>0}$ with ν even and $\nu' = \frac{\nu}{2}$. Let $\mathsf{V}, \mathsf{W} \in \mathbb{F}[x]^{\alpha\times\mu}$, of degree less than ν, define*

$$\mathsf{V}_0 = \mathsf{V} \bmod x^{\nu'}, \quad \mathsf{V}_1 = \mathsf{V} \operatorname{div} x^{\nu'}$$

and

$$\mathsf{W}_0 = \mathsf{W} \bmod x^{\nu'}, \quad \mathsf{W}_1 = \mathsf{W} \operatorname{div} x^{\nu'}.$$

Then the matrices $[\mathsf{V}_0\ \mathsf{V}_1]$ and $[\mathsf{W}_1\ \mathsf{W}_0]$ are in $\mathbb{F}[x]^{\alpha\times 2\mu}$, of degree less than ν', and we have

$$\mathsf{V}\mathsf{W}^t \bmod x^{\nu} = \mathsf{V}_0\mathsf{W}_0^t + x^{\nu'}([\mathsf{V}_0\ \mathsf{V}_1][\mathsf{W}_1\ \mathsf{W}_0]^t \mod x^{\nu'}).$$

The proof of this lemma is given in [2].

This lemma will be applied recursively starting from $\nu = n$ and $\mu = 1$. Thus, we will assume for convenience that both n and α are powers of 2.

- If n is not a power of two, then define $\bar{n} = 2^{\lceil \log n \rceil}$ and $n' = \bar{n} - n$ and it can be easily checked that $a \bmod b = x^{-n'}((x^{n'}a) \bmod (x^{n'}b))$ for a and nonzero $b \in \mathbb{F}[x]$. Application of this technique to R gives us

  $$\mathsf{R} = x^{-n'}\mathsf{U}^t(\mathsf{V}(x^{n'}\mathsf{W})^t \bmod x^{\bar{n}}),$$

  where V and $x^{n'}\mathsf{W}$ have degree less than $\bar{n}$.

- If α is not a power of two, we define $\bar{\alpha} = 2^{\lceil \log \alpha \rceil}$ and $\alpha' = \bar{\alpha} - \alpha$. Then we can introduce α' dummy polynomials $(U_i), (V_i)$ and (W_i) all equals to zero without affecting the value of $r_1, \ldots, r_\alpha$.

Algorithms 2 and 3 together depict the whole process, where Algorithm 3 handles the case when n and α are not powers of two by introducing $\bar{n}$, $\bar{\alpha}$ as defined above and then call Algorithm 2. Algorithm 2 initially set $\nu = \bar{n}$ and $\mu = 1$. The width of matrices V and W are increased in twofold in each recursive call whereas the degree of the entries decreases by half; thus, we maintain the invariant $\mu\nu = \bar{n}$.

Algorithm 2: MUL_REC($\mathsf{U}, \mathsf{V}, \mathsf{W}, n, \nu, \bar{\alpha}, \mu$)

Require: $\mathsf{U} \in \mathbb{F}[x]^{\bar{\alpha}\times 1}$ of degree less than n, $\mathsf{V}, \mathsf{W} \in \mathbb{F}[x]^{\bar{\alpha}\times\mu}$ of degree less than ν
Assumption: $\bar{\alpha}, \mu, \nu$ are powers of 2 and $\mu \leqslant \bar{\alpha}$
Ensure: $\mathsf{R} \in \mathbb{F}[x]^{1\times\bar{\alpha}}$ of degree less than $n+\nu-1$ such that $\mathsf{R} = \mathsf{U}^t(\mathsf{V}\mathsf{W}^t \bmod x^\nu)$.

if $\mu = \bar{\alpha}$ **then**
  $\mathsf{R}' = \mathsf{V}\mathsf{W}^t \bmod x^\nu$
  $R = \mathsf{U}^t\mathsf{R}'$
else
  $\nu' = \frac{\nu}{2}; \mu' = 2\mu$
  $\mathsf{V}_0 = \mathsf{V} \bmod x^{\nu'}, \mathsf{V}_1 = \mathsf{V} \operatorname{div} x^{\nu'}, \mathsf{V}' = [\mathsf{V}_0\mathsf{V}_1]$
  $\mathsf{W}_0 = \mathsf{W} \bmod x^{\nu'}, \mathsf{W}_1 = \mathsf{W} \operatorname{div} x^{\nu'}, \mathsf{W}' = [\mathsf{W}_1\mathsf{W}_0]$
  $\mathsf{R}' =$ MUL_REC($\mathsf{U}, \mathsf{V}', \mathsf{W}', n, \nu', \bar{\alpha}, \mu'$)
  $\mathsf{R} = \mathsf{U}^t\mathsf{V}_0\mathsf{W}_0 + x^{\nu'}\mathsf{R}'$
end if
return R

Algorithm 3: MUL($\mathsf{U}, \mathsf{V}, \mathsf{W}, n, \alpha$)

Require: $\mathsf{U}, \mathsf{V}, \mathsf{W} \in \mathbb{F}[x]^{\alpha\times 1}$ of degree less than n
Assumption: $\alpha \leqslant n$
Ensure: $\mathsf{R} \in \mathbb{F}[x]^{1\times\alpha}$ of degree less than $2n-1$ such that $\mathsf{R} = \mathsf{U}^t(\mathsf{V}\mathsf{W}^t \bmod x^n)$.

$\bar{n} = 2^{\lceil \log n \rceil}; n' = \bar{n} - n$
$\bar{\alpha} = 2^{\lceil \log \alpha \rceil}; \alpha' = \bar{\alpha} - \alpha$
$\bar{\mathsf{U}} = \mathsf{U}$ augmented with α' zero rows
$\bar{\mathsf{V}} = \mathsf{V}$ augmented with α' zero rows
$\bar{\mathsf{W}} = x^{n'}\mathsf{W}$ augmented with α' zero rows
$\bar{\mathsf{R}} =$ MUL_REC($\bar{\mathsf{U}}, \bar{\mathsf{V}}, \bar{\mathsf{W}}, \bar{n}, \bar{\alpha}, 1$)
$\mathsf{R} = x^{-n'}\bar{\mathsf{R}}$
return first α entries of R

The complexity analysis is taken from [2]. For the base case of algorithm 2, i.e. when $\mu = \bar{\alpha}$, we have $\mathsf{V}, \mathsf{W} \in \mathbb{F}[x]^{\bar{\alpha}\times\bar{\alpha}}$ of degree less than ν, and the algorithm first computes $\mathsf{R}' = \mathsf{V}\mathsf{W}^t \bmod x^\nu$ in time $O(\bar{\alpha}^\omega \mathsf{M}(\nu))$, simply by multiplying polynomial matrices. Then, the computation of $\mathsf{R} = \mathsf{U}^t\mathsf{R}'$, where U in $\mathbb{F}[x]^{\bar{\alpha}\times 1}$ of degree less than n and $\mathsf{R}' \in \mathbb{F}[x]^{\bar{\alpha}\times\bar{\alpha}}$ of degree less than ν is done in two steps. We rewrite $\mathsf{U}^t = [1\ x^c\ x^{2c} \dots x^{(\bar{\alpha}-1)c}]\mathsf{U}'^t$ where $c = \lceil \frac{n}{\bar{\alpha}} \rceil$ and $\mathsf{U}' \in \mathbb{F}[x]^{\bar{\alpha}\times\bar{\alpha}}$ with entries of degree less than c. Thus, we can compute $\mathsf{Q} = \mathsf{U}'^t\mathsf{R}'$ at first step in time $O(\bar{\alpha}^\omega \mathsf{M}(\max\{c, \nu\}))$ and then can deduce R from Q in time $O(\bar{\alpha}(n+\nu))$. In summary the total time

required at base case of algorithm 2 is

$$\begin{aligned} & O(\bar{\alpha}^{\omega}\mathsf{M}(\nu)) + O(\bar{\alpha}^{\omega}\mathsf{M}(\max\{c,\nu\})) + O(\bar{\alpha}(n+\nu)) \\ = \;& O(\bar{\alpha}^{\omega}\mathsf{M}(\max\{c,\nu\})). \end{aligned}$$

We always have $\nu = \frac{\bar{n}}{\mu} \leqslant \frac{2n}{\mu}$, so at the base case of Algorithm 2, we have $\nu \leqslant \frac{2n}{\bar{\alpha}}$. Similarly, we have by definition $c \leqslant \frac{n}{\bar{\alpha}} + 1$ which is at most $\frac{3n}{\bar{\alpha}}$ since $\bar{\alpha} \leqslant 2\alpha \leqslant 2n$. Thus the cost of base case of Algorithm 2 is $O(\bar{\alpha}^{\omega}\mathsf{M}(\frac{n}{\bar{\alpha}})) = O(\bar{\alpha}^{\omega-1}\mathsf{M}(n))$, using the superlinearity assumption on M from [4]. This in turn is $O(\alpha^{\omega-1}\mathsf{M}(n))$.

Let $C(k)$ denotes the cost of algorithm 2 called upon parameters $\bar{\alpha}$ and μ such that $\bar{\alpha} = 2^k\mu$. Then the cost of algorithm 3 will be $C(\mathsf{k})$ where $\mathsf{k} = \lceil \log \alpha \rceil$. When $k = 0$, we have the cost of base case. Now, let us investigate the case $k \geqslant 1$.

In algorithm 2, given V, W in $\mathbb{F}[x]^{\bar{\alpha}\times\mu}$, we can compute V_i and W_i for free. Then algorithm 2 computes R' in time $C(k-1)$ recursively and we are left with estimating the cost of computing $\mathsf{R} = \mathsf{Q} + x^{\nu'}\mathsf{R}'$ with $\mathsf{Q} = \mathsf{U}^t\mathsf{V}_0\mathsf{W}_0$.

Let $D(k)$ denotes the time to compute Q. Then, given $\mathsf{R}' \in \mathbb{F}[x]^{1\times\alpha}$ and $\mathsf{Q} \in \mathbb{F}[x]^{1\times\alpha}$, both of degrees less than $n+\nu-1$, the addition can be done with $\bar{\alpha}(n+\nu)$ operations. Since $\nu = \frac{\bar{n}}{\mu} \leqslant 2n$, we have

$$C(k) \leqslant C(k-1) + D(k) + O(\bar{\alpha}n)$$

for $k \geqslant 1$.

In order to estimate $D(k)$, let us rewrite $\mathsf{U}^t = [1 \; x^c \; x^{2c} \ldots x^{(\bar{\alpha}-1)c}]\mathsf{U}'^t$ as before, where $c = \lceil \frac{n}{\mu} \rceil$ and $\mathsf{U}' \in \mathbb{F}[x]^{\bar{\alpha}\times\mu}$, with entries of degree less than c. Then, we can compute Q as $\mathsf{Q} = [1 \; x^c \; x^{2c} \ldots x^{(\bar{\alpha}-1)c}](\mathsf{U}'^t\mathsf{V}_0\mathsf{W}_0^t)$. The computation of the matrix product $\mathsf{U}'^t\mathsf{V}_0\mathsf{W}_0^t$ involves three matrices of dimensions $\mu\times\bar{\alpha}$, $\bar{\alpha}\times\mu$ and $\mu\times\bar{\alpha}$ respectively having polynomial entries of degree less than $\max(c,\nu)$. As before, we can prove that $\max(c,\nu) \leqslant \frac{3n}{\mu}$, so, the computation of $\mathsf{U}'^t\mathsf{V}_0\mathsf{W}_0^t$ can be done in time

$$O\big(\frac{\bar{\alpha}}{\mu}\; \mu^{\omega}\mathsf{M}(\frac{n}{\mu})\big)$$

using [3, Lemma 7]. Since $\bar{\alpha} = 2^k\mu$, we can rewrite the above complexity as

$$O\big(2^k\; (\frac{\bar{\alpha}}{2^k})^{\omega}\mathsf{M}(\frac{2^k n}{\bar{\alpha}})\big).$$

The time required to reconstruct Q once $\mathsf{U}'^t\mathsf{V}_0\mathsf{W}_0^t$ is known is the same as before,

$O(\bar{\alpha}n)$. Taking all $k = 0, \ldots, \mathsf{k}$ into account we have

$$\begin{aligned} & O(\sum_{k=0}^{\mathsf{k}} 2^k \ (\frac{\bar{\alpha}}{2^k})^{\omega} \mathsf{M}(\frac{2^k n}{\bar{\alpha}})) \\ = \ & O(\bar{\alpha}^{\omega} \mathsf{M}(\frac{n}{\bar{\alpha}})) \text{ neglecting negative exponent} \\ = \ & O(\bar{\alpha}^{\omega-1} \mathsf{M}(n)) \end{aligned}$$

Since $\bar{\alpha} \leqslant 2\alpha$, we deduce that the complexity for any α is

$$O(\alpha^{\omega-1} \mathsf{M}(n)).$$

This is to be compared to the cost $O(\alpha^2 \mathsf{M}(n))$ resulting from the naive algorithm: the new algorithm is better, at least in theory, as soon as $\alpha < 3$, that is, as soon as we use a matrix multiplication algorithm better than the naive algorithm.

In case when $\alpha \neq \beta$ this algorithm will be called c times where c will be either $\lceil \frac{\beta}{\alpha} \rceil$ or $\lceil \frac{\alpha}{\beta} \rceil$. Figure 4.1 shows the performance improvement between naive reduction and algorithm 2. We implemented these algorithms in C++ using NTL [10] and run them on a 2.1 GHz AMD Athlon 64 processor. Our base field is FFT prime [4] and we used FFT approach as a subroutine to multiply matrices of algorithm 2. We used naive matrix multiplication algorithm instead of Strassen matrix multiplication algorithm. Figures 4.1, 4.2 and 4.3 summarize the results where we see that algorithm 2 outperforms the naive reduction when we increase the degree for fixed α. Here the degrees are represented in logarithmic scale and alpha was reduced by dividing the original value by 10. In Figure 4.3, the degree was fixed at 40 whereas alpha was fixed at 40 in Figure 4.2. Observe that the algorithm 2 performs better than the naive reduction when we increase the degree keeping α fixed which is shown in Figure 4.2. On the other hand, Figure 4.3 shows that we will not get any improvement if we increase α keeping the degree fixed.

Bibliography

[1] R. R. Bitmead and B. D. O. Anderson. Asymptotically fast solution of Toeplitz and related systems of linear equations. *Linear Algebra and its Applications*, 34:103–116, 1980.

[2] A. Bostan, C.-P. Jeannerod, C. Mouilleron, and É. Schost. Fast simultaneous

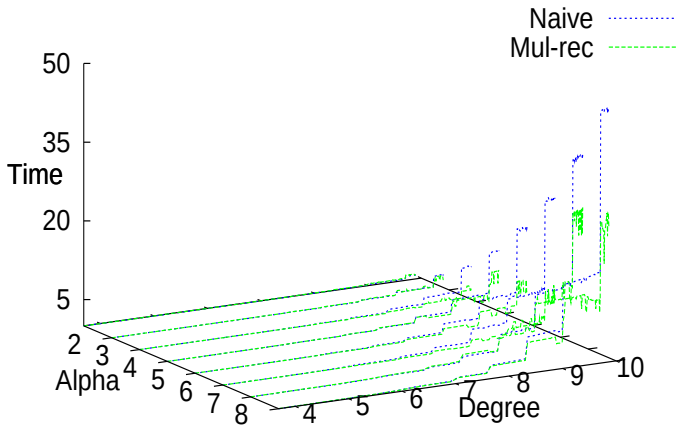


Figure 4.1: Algorithm MUL_REC vs naive reduction

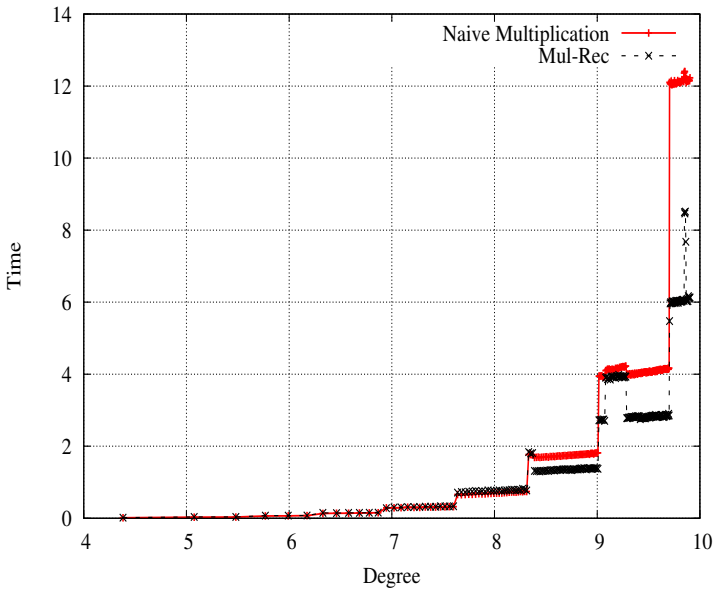


Figure 4.2: α fixed

multiplication of a structured matrix by vectors. *PrePrint*, 2012.

[3] A. Bostan, C.-P. Jeannerod, and É. Schost. Solving structured linear systems with large displacement rank. *Theorerical Compututer Science*, 407(1-3):155–181, 2008.

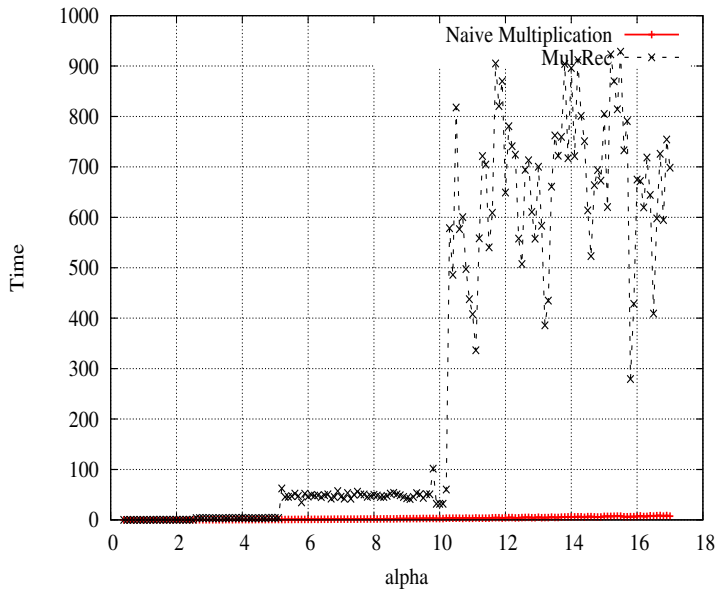


Figure 4.3: Degree fixed

[4] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

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

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

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

[8] C. Mouilleron. *Efficient computation with structured matrices and arithmetic expressions*. PhD thesis, 2011.

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

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

Chapter 5

Power Series Solutions of Singular (q)-Differential Equations

This chapter is published in the homonym paper with A. Bostan, R. Lebreton, B. Salvy and É. Schost in the proceedings of ISSAC12.

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.

5.1 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 $f(g)$ mod 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 [4, 25] or when more information on f or g is available. Quasi-linear complexity has been reached when g is a polynomial [11], an algebraic series [19], or belongs to a large class containing for instance the expansions of $\exp(x)-1$ and $\log(1+x)$ [8].

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 [5]. 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

$$x^k \delta(F) = AF + C, \tag{5.1}$$

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 δ denotes the differential operator d/dx. The exponent k in (5.1) 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 (5.1) holds modulo x^N. For this, we need only compute F polynomial of degree less than $N+1$ (when $k=0$) or N (otherwise). Conversely, when (5.1) has a power series solution, its first N coefficients can be computed by solving (5.1) 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 (5.1) 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 [5]. In this article, we extend the results of [5] 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

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

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

$$x(x-1)f'' + (x(a+b+1) - c)f' + abf = 0 \tag{5.2}$$

and build up and solve the more complicated

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

in the unknown F, g being given, with $g(0)=0$. Equation (5.2) 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 [5] do not apply — this example is actually

already mentioned in [11], 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 Eq. (5.1) 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 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 [7, 8, 10].

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 [6, 27]), but this is not the purpose of this paper.

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 $\sigma : \mathbb{K}[[x]] \to \mathbb{K}[[x]]$ a unitary ring morphism and letting $\delta : \mathbb{K}[[x]] \to \mathbb{K}[[x]]$ denote a σ-derivation, in the sense that δ is $\mathbb{K}$-linear and that for all f, g in $\mathbb{K}[[x]]$, we have

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

These definitions, and the above equality, carry over to matrices over $\mathbb{K}[[x]]$. Thus, our goal is to solve the following generalization of (5.1):

$$x^k\delta(F) = A\sigma(F) + C. \tag{5.3}$$

As above, we are interested in computing a vector F of power series such that (5.3) holds mod x^N.

One motivation for this generalization comes from coding theory. The list-decoding of the folded Reed-Solomon codes [18] leads to an equation $Q(x, f(x), f(qx)) = 0$ where Q is a known polynomial. A linearized version of this is of the form (5.3), with $\sigma : \phi(x) \mapsto \phi(qx)$. 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:

$$\delta(x) = 1, \qquad \sigma : x \mapsto qx,$$

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

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

As a consequence, $\delta(1) = 0$ and for all $i \geqslant 0$, 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 $f = \sum_{i \geqslant 0} f_i x^i \in \mathbb{K}[[x]]$,

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

can be computed mod x^N in $O(N)$ 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 $g = \sum_{0 \leqslant i \leqslant n-1} f_i/\gamma_{i+1} x^{i+1}$ and can be computed in $O(N)$ operations. We denote it by $g = \mathbb{I}f$. 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 $\mathscr{M}_{n,m}(R)$; when $n = m$, we write $\mathscr{M}_n(R)$. 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 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 [17, S8.3]; using Fast Fourier Transform, $\mathsf{M}(n)$ can be taken in $O(n \log(n) \log\log(n))$ [13, 28]. We note $\omega \in (2, 3]$ a constant

such that two matrices in $\mathscr{M}_n(R)$ can be multiplied in $O(n^\omega)$ arithmetic operations in R. The current best bound is $\omega < 2.3727$ ([31] following [14, 30]).

Our algorithms rely on linear algebra techniques; in particular, we have to solve several systems of non-homogeneous linear equations. For U in $\mathscr{M}_n(\mathbb{K})$ and V in $\mathscr{M}_{n,1}(\mathbb{K})$, we denote by $\mathsf{LinSolve}(UX = V)$ a procedure that returns $\perp$ if there is no solution, or a pair F, K, where F is in $\mathscr{M}_{n,1}(\mathbb{K})$ and satisfies $UF = V$, and $K \in \mathscr{M}_{n,t}(\mathbb{K})$, for some $t \leqslant n$, generates the nullspace of U. This can be done in time $O(n^\omega)$. 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 (5.3) 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. (5.3) at precision N either $\perp$ (if no solution exists) or a pair F, K where $F \in \mathscr{M}_{n,1}(\mathbb{K}[x])$ and $K \in \mathscr{M}_{n,t}(\mathbb{K}[x])$ with $t \leqslant nN$, such that for $G \in \mathscr{M}_{n,1}(\mathbb{K}[x])$, with $\deg(G) < N$, $x^k\delta(G) = A\sigma(G) + C \bmod x^N$ if and only if G can be written $G = F + KB$ for some $B \in \mathscr{M}_{t,1}(\mathbb{K})$.

Seeing Eq. (5.3) as a linear system, one can obtain such an output using linear algebra in dimension nN. 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 1. *The matrix A_0 has* good spectrum at precision N *when one of the following holds:*

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

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 $\operatorname{Spec} A_0 = \{0, \mathrm{val}(g)(1-c) - 1\}$).

These conditions could be slightly relaxed, using gauge transformations (see [1, Ch. 2] and [2, 3]). 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 [5], 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 5.2 (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 1. *Algorithm 5 computes generators of the solution space of Eq. (5.3) at precision N by a divide-and-conquer approach. Assuming A_0 has good spectrum at precision N, it performs in time $O(n^\omega \mathsf{M}(N)\log(N))$. When either $k > 1$ or $k = 1$ and $q^i A_0 - \gamma_i \mathsf{Id}$ is invertible for $0 \leqslant i < N$, this drops to $O(n^2\mathsf{M}(N)\log(N) + n^\omega N)$.*

Our second algorithm behaves better with respect to N, with cost in $O(\mathsf{M}(N))$ 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 [5, 11] 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 (5.3) admits such a solution. Section 5.3 describes a variant of Newton's iteration to solve it and obtains the following.

Theorem 2. *Assuming A_0 has good spectrum at precision N, one can compute generators of the solution space of Eq. (5.3) at precision N by a Newton-like iteration in time $O(n^\omega \mathsf{M}(N) + n^\omega \log(n)N)$.*

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.

5.2 Divide-and-Conquer

The classical approach to solving (5.3) is to proceed term-by-term by coefficient extraction. Indeed, we can rewrite the coefficient of degree i in this equation as

$$R_i F_i = \Delta_i, \tag{5.4}$$

where Δ_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 $\mathcal{R} = \{j_1 < \cdots < j_r\}$, 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 Δ_i takes linear time in i. To achieve quasi-linear time, we split the resolution of Eq. (5.3) 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. (5.4).

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 [2, 3]; 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 nr 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 φ_0 in $\mathscr{M}_{n,1}(\mathbb{K}[x])$ and each φ_ℓ in $\mathscr{M}_n(\mathbb{K}[x])$ for $1 \leqslant \ell \leqslant r$. We also define $\mathscr{L}_i$ the

Algorithm 4: Recursive Divide-and-Conquer

RDAC$(A, \mathbf{C}, i, N, k)$
input : $A \in \mathscr{M}_n(\mathbb{K}[[x]]), \mathbf{C} \in \mathscr{L}_i$, $i \in \mathbb{N}, N \in \mathbb{N} \setminus \{0\}, k \in \mathbb{N} \setminus \{0\}$
output: $\mathbf{F} \in \mathscr{L}_{i+N}$
if $N = 1$ **then**
  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
  $m := \lceil N/2 \rceil$
  $\mathbf{H} :=$ 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
  $\mathbf{K} :=$ RDAC$(A, \mathbf{D}, i + m, N - m, k)$
  return $\mathbf{H} + x^m \mathbf{K}$
end

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* $S : \mathscr{L} \to \mathscr{M}_{n,1}(\mathbb{K}[x])$ is simply an evaluation map defined by $\mathbf{f}_{i,\ell} \mapsto f_{i,\ell}$ for all i, ℓ, for some choice of $(f_{i,\ell})$ in $\mathbb{K}^{nr}$.

We extend δ and σ to such vectors, by letting $\delta(\mathbf{f}_{i,\ell}) = 0$ and $\sigma(\mathbf{f}_{i,\ell}) = \mathbf{f}_{i,\ell}$ for all i, ℓ, 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 $\sigma(\mathbf{F})$.

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 5. In a second step, we resolve the indeterminacies by plain linear algebra. For $i \geqslant 0$, and $\mathbf{F}, \mathbf{C}$ in $\mathscr{L}$, we write

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

In particular, $E(\mathbf{F}, \mathbf{C}, 0)$ is a parameterized form of Eq. (5.3). 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

$$E(\mathbf{F}, \mathbf{C}, i) = (E(\mathbf{H}, \mathbf{C}, i) \bmod x^m) + x^m E(\mathbf{K}, \mathbf{D}, i+m). \tag{5.5}$$

Lemma 1. *Let A be in $\mathscr{M}_n(\mathbb{K}[x])$ and $\mathbf{C}$ in $\mathscr{L}_i$, and let $\mathbf{F} = \mathsf{RDAC}(A, \mathbf{C}, i, M, k)$ with $i + M \leqslant N$. 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 $\mathscr{M}_{n,1}(\mathbb{K}[x])$ with $\deg F < M$ are such that $E(F, C, i) = 0 \bmod x^M$ and there exists a specialization $S : \mathscr{L}_i \to \mathscr{M}_{n,1}(\mathbb{K}[x])$ such that $C = S(\mathbf{C})$, there exists a specialization $S' : \mathscr{L}_{i+M} \to \mathscr{M}_{n,1}(\mathbb{K}[x])$ which extends S and such that $F = S(\mathbf{F})$.*

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

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 $m \leqslant j$, 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.

Algorithm 5: Divide-and-Conquer

$\mathsf{DAC}(A, C, N, k)$
input : $A \in \mathscr{M}_n(\mathbb{K}[[x]]), C \in \mathscr{M}_{n,1}(\mathbb{K}[[x]])$, $N \in \mathbb{N} \setminus \{0\}, k \in \mathbb{N} \setminus \{0\}$
output: Generators of the solution space of $x^k\delta(F) = A\sigma(F) + C$ 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 $\varphi_0 + M\Phi, M\Delta$

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 $\mathbb{K}$; 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 $H = F \bmod x^m$, $K = F \operatorname{div} x^m$ and $D = (C - x^k\delta(H) + (q^iA - \gamma_i x^{k-1}\mathsf{Id})\sigma(H)) \operatorname{div} x^m$. Then, (5.5) 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 $S' : \mathscr{L}_{i+m} \to \mathscr{M}_{n,1}(\mathbb{K}[x])$ which extends S. In particular, $D = S'(\mathbf{D})$. The induction assumption also implies that there exist an extension $S'' : \mathscr{L}_{i+m} \to \mathscr{M}_{n,1}(\mathbb{K}[x])$ 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 $O(N(n+rn^2))$ and the product is done in time $O((n^2+rn^\omega)\mathsf{M}(M))$; recursion leads to a factor $\log(M)$. The base cases use $O(M)$ matrix inversions of cost $O(n^\omega)$ and $O(M)$ multiplications, each of which takes time $O(rn^\omega)$. □

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.

Proposition 1. *On input A, C, N, k as specified in Algorithm 5, this algorithm returns generators of the solution space of (5.3) mod x^N in time $O((n^2 + rn^\omega)\mathsf{M}(N)\log(N) + r^2n^\omega N + r^\omega n^\omega)$.*

Proof. The first claim is a direct consequence of the construction above, combined with Lemma 1. 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, so no new contribution comes from here. Solving the system $\Gamma = 0$ takes time $O((rn)^\omega)$. Finally, the product $[\varphi_1 \cdots \varphi_r]\Delta$ involves an $n \times (rn)$ matrix with entries of degree N and an $(rn) \times t$ constant matrix, with $t \leqslant rn$; proceeding coefficient by coefficient, and using block matrix multiplication in size n, the cost is $O(r^2 n^\omega N)$. □

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 $O(n^2\mathsf{M}(N)\log(N)+n^\omega N)$ for Lemma 1, and Proposition 1 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 $r \leqslant 1$. Indeed, when $k = 1$, the good spectrum condition implies that for all $0 \leqslant i < N$ 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. 1.

Previous work. As said above, Barkatou and Pflügel [3], then Barkatou, Broughton and Pflügel [2], 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 $k \geqslant 2$), 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 [12]. That algorithm is analyzed for a more general σ, of the form $\sigma(x) = xq(x)$, 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 [19, 22, 23], which allows one to solve systems such as (5.3) 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 [19, 20], van der Hoeven offers two relaxed multiplication algorithms (the first one being similar to that of [16]); both take time $O(\mathsf{M}(n)\log(n))$. When $r = 0$, this yields a complexity similar to Prop. 1 to solve Eq. (5.3), 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 [20] to solve Eq. (5.3) 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 [19] was improved in [21, 24], by means of a reduction of the $\log(n)$ overhead. This raises the question of whether such an improvement is available for divide-and conquer techniques.

5.3 Newton Iteration

5.3.1 Gauge Transformation

Let F be a solution of Eq. (5.3). To any invertible matrix $W \in \mathscr{M}_n(\mathbb{K}[x])$, we can associate the matrix $Y = W^{-1}F \in \mathscr{M}_n(\mathbb{K}[[x]])$. We are going to choose W in such a way that Y satisfies an equation simpler than (5.3). The heart of our contribution is the efficient computation of such a W.

Lemma 2. *Let $W \in \mathscr{M}_n(\mathbb{K}[x])$ be invertible in $\mathscr{M}_n(\mathbb{K}[[x]])$ and let $B \in \mathscr{M}_n(\mathbb{K}[x])$ be such that*

$$B = W^{-1}(x^k\delta(W) - A\sigma(W)) \bmod x^N. \tag{5.6}$$

Then F in $\mathscr{M}_{n,1}(\mathbb{K}[x])$ satisfies

$$x^k\delta(F) = A\sigma(F) + C \bmod x^N \tag{5.3}$$

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

$$x^k\delta(Y) = B\sigma(Y) + W^{-1}C \bmod x^N. \tag{5.7}$$

Proof. Differentiating the equality $F = WY$ gives

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

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 (5.3) and (5.7) are called equivalent under the gauge transformation $Y = WF$. Solving (5.3) is thus reduced to finding a simple B such that (5.7) can be solved efficiently and such that the equation

$$x^k\delta(W) = A\sigma(W) - WB \bmod x^N \tag{5.8}$$

that we call *associated* to (5.3) 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

$$xW' = AW - WA_0 \bmod x^N.$$

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

$$(A_0 - i\mathsf{Id})W_i - W_iA_0 = -\sum_{j<i} A_{i-j}W_j.$$

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.

5.3.2 Polynomial Coefficients

Our approach consists in reducing efficiently the resolution of (5.3) 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

$$x^k\delta(Y) = P\sigma(Y) + Q, \tag{5.9}$$

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

Algorithm 6: PolCoeffsDE

PolCoeffsDE(P, Q, k, N)
input : $P \in \mathscr{M}_n(\mathbb{K}[x])$ of degree less than k, $Q \in \mathscr{M}_{n,1}(\mathbb{K}[[x]])$, $N \in \mathbb{N} \setminus \{0\}$, $k \in \mathbb{N} \setminus \{0\}$
output: Generators of the solution space of $x^k\delta(Y) = P\sigma(Y) + Q$ at precision N.
for $i = 0, \ldots, N-1$ **do**
  $C := Q_i + (P_1 q^{i-1} Y_{i-1} + \cdots + P_{k-1} q^{i-k+1} Y_{i-k+1})$
  if $k = 1$ **then**
    $Y_i, M_i := \mathsf{LinSolve}((\gamma_i \mathsf{Id} - q^i P_0)X = C)$
  else
    $Y_i, M_i := \mathsf{LinSolve}(-q^i P_0 X = C - \gamma_{i-k+1} Y_{i-k+1})$
  end
end
return $Y_0 + \cdots + Y_{N-1}x^{N-1}$, $[M_0 \; M_1 x \cdots M_{N-1}x^{N-1}]$

Lemma 3. *Suppose that P_0 has good spectrum at precision N. Then Algorithm 6 computes generators of the solution space of Eq. (5.9) at precision N in time $O(n^\omega N)$, with $M \in \mathscr{M}_{n,t}(\mathbb{K})$ for some $t \leqslant n$.*

Proof. Extracting the coefficient of x^i in Eq. (5.9) gives

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

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 $O(n^\omega)$ operations, whence the announced complexity. □

5.3.3 Computing the Associated Equation

Given $A \in \mathscr{M}_n(\mathbb{K}[[x]])$, we are looking for a matrix B with polynomial entries of degree less than k such that the associated equation (5.8), 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 (5.8) has an invertible solution V mod x^k; thus, V and B must satisfy $A\sigma(V) = VB \bmod x^k$. The invertible matrix V is then lifted at higher precision by Newton iteration (Algorithm 9) 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 [2, 33].

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 the case anymore. In that case, we use a known technique called the *splitting lemma* to prepare our equation. See for instance [1, Ch. 3.2] and [2] for details and generalizations.

Lemma 4 (Splitting Lemma)**.** *Suppose that $k > 1$, that $|\operatorname{Spec} A_0| = n$ and that $\operatorname{Spec} A_0 \subset \mathbb{K}$. Then one can compute in time $O(n^\omega)$ matrices V and B of degree less than k in $\mathscr{M}_n(\mathbb{K}[x])$ such that the following holds: V_0 is invertible; B is diagonal; $AV = VB \bmod x^k$.*

Proof. We can assume that A_0 is diagonal: if not, we let P be in $\mathscr{M}_n(\mathbb{K})$ such that $D = P^{-1}AP$ has a diagonal constant term; we find V using D instead of A, and replace V by PV. Computing P and PV takes time $O(n^\omega)$, 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 $A_0V_i - V_iA_0 - B_i = \Delta_i$, where Δ_i can be computed from $A_1, \ldots, A_i$ and $B_1, \ldots, B_{i-1}$ in time $O(n^\omega)$. 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 Δ_i, up to sign. Then the entry (ℓ, 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 $O(n^\omega)$. □

5.3.4 Solving the Associated Equation

Once B and V are determined as in **S**5.3.3, we compute a matrix W that satisfies the associated equation (5.8); this eventually allows us to reduce (5.3) to an equation with polynomial coefficients. This computation of W is performed efficiently using a suitable version of Newton iteration for Eq. (5.8); it computes a sequence of matrices whose precision is roughly doubled at each stage. This is described in Algorithm 9; our main result in this section is the following.

Proposition 2. *Suppose that A_0 has good spectrum at precision N. Then, given a solution of the associated equation mod x^k, invertible in $\mathscr{M}_n(\mathbb{K}[[x]])$, Algorithm 9*

Algorithm 7: Solving Eq. (5.10) when $k = 1$ or $q \neq 1$

DiffSylvester(Γ, m, N)
input : $\Gamma \in x^m \mathscr{M}_n(\mathbb{K}[[x]]), m \in \mathbb{N} \setminus \{0\}, N \in \mathbb{N} \setminus \{0\}$
output: $U \in x^{m-k}\mathscr{M}_n(\mathbb{K}[x])$ solution of (5.10).
for $i = m, \ldots, N-1$ **do**
  $C := (B_1 q^{i-1} U_{i-1} + \cdots + B_{k-1} q^{i-k+1} U_{i-k+1})$
    $-(U_{i-1}B_1 + \cdots + U_{i-k+1}B_{k-1}) + \Gamma_i$
  if $k = 1$ **then**
    $U_i := \mathsf{Sylvester}(XB_0 + (\gamma_i \mathsf{Id} - q^i B_0)X = C)$
  else
    $U_i := \mathsf{Sylvester}(XB_0 - q^i B_0 X =$
      $C - \gamma_{i-k+1} U_{i-k+1})$
  end
end
return $U_m x^m + \cdots + U_{N-1} x^{N-1}$

computes a solution of that equation $\bmod x^N$*, also invertible in* $\mathscr{M}_n(\mathbb{K}[[x]])$*, in time* $O(n^\omega \mathsf{M}(N) + n^\omega \log(n) N)$.

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

$$x^k \delta(U) = B\sigma(U) - UB + \Gamma \bmod x^N, \tag{5.10}$$

where all matrices involved have size $n \times n$, with $\Gamma = 0 \bmod x^m$. This is dealt with by Algorithm 7 when $k = 1$ or $q \neq 1$ and Algorithm 8 otherwise.

For Algorithm 7, remember that $B = A \bmod x^k$. The algorithm uses a routine Sylvester solving *Sylvester equations*. Given matrices Y, V, Z in $\mathscr{M}_n(\mathbb{K})$, we are looking for X in $\mathscr{M}_n(\mathbb{K})$ such that $YX - XV = Z$. When (Y, V) have disjoint spectra, this system admits a unique solution, which can be computed $O(n^\omega \log(n))$ operations in $\mathbb{K}$ [26].

Lemma 5. *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* $k \leqslant m < N$*, then Algorithm 7 computes a solution* U *to Eq. (5.10) that satisfies* $U = 0 \bmod x^{m-k+1}$ *in time* $O(n^\omega \log(n) N)$.

Proof. Extracting the coefficient of x^i in (5.10) 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 7. In both cases $k = 1$ and $k > 1$, this gives a Sylvester

Algorithm 8: Solving Eq. (5.10) when $k > 1$ and $q = 1$

DiffSylvesterDifferential(Γ, m, N)
input : $\Gamma \in x^m \mathscr{M}_n(\mathbb{K}[[x]]), m \in \mathbb{N} \setminus \{0\}, N \in \mathbb{N} \setminus \{0\}$
output: $U \in x^{m-k}\mathscr{M}_n(\mathbb{K}[x])$ solution of (5.10).
for $i = 1, \ldots, n$ **do**
  for $j = 1, \ldots, n$ **do**
    if $i = j$ **then** $U^{(i,i)} := x^k \int (x^{-k}\Gamma^{(i,i)}) \bmod x^N$
    else
      $U^{(i,j)} :=$ PolCoeffsDE$(B^{(i,i)} - B^{(j,j)}, \Gamma^{(i,j)}, k, N)$
    end
  end
end
return U

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 Γ is $0 \bmod x^m$, so is U (so we can start the loop at index m). The total running time is $O(n^\omega \log(n) N)$ operations in $\mathbb{K}$. □

This approach fails in the differential case ($q = 1$) when $k > 1$, since then the Sylvester systems are all singular. Algorithm 8 deals with this issue, using the fact that in this case, B is diagonal, and satisfies the conditions of Lemma 4.

Lemma 6. *Suppose that $k > 1$, $q = 1$ and that A_0 has good spectrum at precision N. If $\Gamma = 0 \bmod x^m$, with $k \leqslant m < N$, then Algorithm 8 computes a solution U to Eq. (5.10) that satisfies $U = 0 \bmod x^{m-k+1}$ in time $O(n^2 N)$.*

Proof. Since B is diagonal, the (i, j)th entry of (5.10) 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 3 applies (and since $k > 1$, the solution is unique, as pointed out in its proof). □

We now prove the correctness of Algorithm 9 for Newton iteration. Instead of doubling the precision at each step, there is a slight loss of $k - 1$.

Algorithm 9: Newton iteration for Eq. (5.8)

NewtonAE(V, N)
input : $V \in \mathscr{M}_n(\mathbb{K}[x])$ solution of (5.8) modx^k invertible in $\mathscr{M}_n(\mathbb{K}[[x]])$,
$N \in \mathbb{N} \setminus \{0\}$
output: $W \in \mathscr{M}_n(\mathbb{K}[x])$ solution of (5.8) modx^N invertible in $\mathscr{M}_n(\mathbb{K}[[x]])$,
with $W = V \bmod x^k$
if $N \leqslant k$ **then** return V
else
  $m := \lceil \frac{N+k-1}{2} \rceil$
  $H :=$ NewtonAE(V, m)
  $R := x^k\delta(H) - A\sigma(H) + HB$
  if $k = 1$ or $q \neq 1$ **then**
    $U :=$ DiffSylvester$(-H^{-1}R, m, N)$
  else
    $U :=$ DiffSylvesterDifferential$(-H^{-1}R, m, N)$
  end
  return $H + HU$
end

Lemma 7. *Let $m \geqslant k$ and let $H \in \mathscr{M}_n(\mathbb{K}[x])$ be invertible in $\mathscr{M}_n(\mathbb{K}[[x]])$ and satisfy (5.8) modx^m. Let N be such that $m \leqslant N \leqslant 2m - k + 1$. Let R and U be as in Algorithm 9 and suppose that A_0 has good spectrum at precision N.*

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

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

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

Using either Lemma 5 or Lemma 6, $U = 0 \bmod x^{m-k+1}$, so $\sigma(U) = 0 \bmod x^{m-k+1}$. Thus, the latter expression is 0, since $2m - k + 1 \geqslant N$. Finally, since $HU = 0 \bmod x^{m-k+1}$, and $m \geqslant k$, $H + HU$ remains invertible in $\mathscr{M}_n(\mathbb{K}[[x]])$. The various matrix products and inversions take a total number of $O(n^\omega \mathsf{M}(N))$ operations in $\mathbb{K}$ (using Newton iteration to invert H). Adding the cost of Lemma 5, resp. Lemma 6, we get the announced complexity. □

We can now prove Proposition 2. Correctness is obvious by repeated applications

of the previous lemma. The cost $C(N)$ of the computation up to precision N satisfies

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

Using the super-additivity properties of the function M as in [17, Ch. 9], we obtain the claimed complexity.

We can now conclude the proof of Thm. 2. In order to solve Equation (5.3), we first determine B and V as in §5.3.3; the cost will be negligible. Then, we use Proposition 2 to compute a matrix W that satisfies (5.8) $\bmod x^N$. Given C in $\mathscr{M}_{n,1}(\mathbb{K}[[x]])$, 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 3 gives us generators of the solution space of this equation $\bmod x^N$. If it is inconsistent, we infer that Eq. (5.3) is. Else, from the generators (Y, M) obtained in Lemma 3, we deduce that $(WY, WM) \bmod x^N$ is a generator of the solution space of Eq. (5.3) $\bmod x^N$. Since the matrix M has few columns (at most n), the cost of all these computations is dominated by that of Proposition 2, as reported in Thm. 2.

5.4 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 [29]. 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 [32], see also [15]) or evaluation-interpolation techniques [9].

Our implementation follows the descriptions given above, up to a few optimizations for algorithm 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

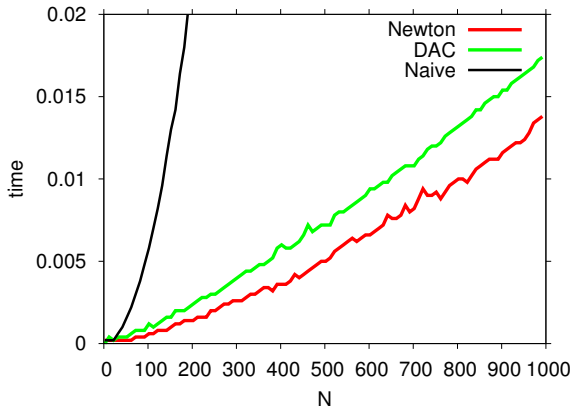


Figure 5.1: Timings with $n = 1$, $k = 1$, $q \neq 1$

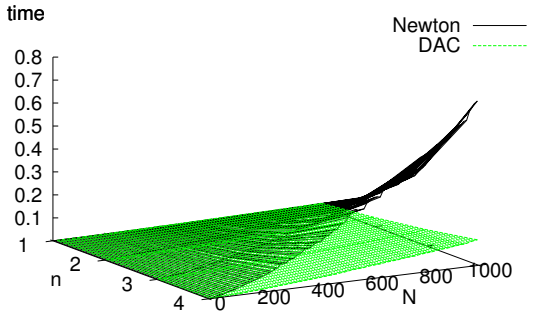


Figure 5.2: Timings with $k = 1$, $q \neq 1$

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. 5.1, 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. 5.2 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. 1).

Finally, Table 5.1 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 5.1: Timings with $k = 3$, $q \neq 1$

Bibliography

[1] W. Balser. *Formal power series and linear systems of meromorphic ordinary differential equations*. Universitext. Springer-Verlag, New York, 2000.

[2] M. Barkatou, G. Broughton, and E. Pflügel. A monomial-by-monomial method for computing regular solutions of systems of pseudo-linear equations. *Math. Comput. Sci.*, 4(2-3):267–288, 2010.

[3] M. Barkatou and E. Pflügel. An algorithm computing the regular formal solutions of a system of linear differential equations. *J. Symb. Comput.*, 28(4-5):569–587, 1999.

[4] D. J. Bernstein. Composing power series over a finite ring in essentially linear time. *J. Symb. Comput.*, 26(3):339–341, 1998.

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

[6] A. Bostan, L. González-Vega, H. Perdry, and É. Schost. From Newton sums to coefficients: complexity issues in characteristic p. In *MEGA'05*, 2005.

[7] A. Bostan, F. Morain, B. Salvy, and É. Schost. Fast algorithms for computing isogenies between elliptic curves. *Math. of Comp.*, 77(263):1755–1778, 2008.

[8] A. Bostan, B. Salvy, and É. Schost. Power series composition and change of basis. In *ISSAC'08*, pages 269–276. ACM, 2008.

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

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

[11] R. P. Brent and H. T. Kung. Fast algorithms for manipulating formal power series. *J. ACM*, 25(4):581–595, 1978.

[12] R. P. Brent and J. F. Traub. On the complexity of composition and generalized composition of power series. *SIAM J. Comput.*, 9:54–66, 1980.

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

[14] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[15] C.-É. Drevet, M. Islam, and É. Schost. Optimization techniques for small matrix multiplication. *Theoretical Computer Science*, 412:2219–2236, 2011.

[16] Fischer and Stockmeyer. Fast on-line integer multiplication. *J. of Computer and System Sciences*, 9, 1974.

[17] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[18] V. Guruswami and A. Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Trans. on Information Theory*, 54(1):135–150, 2008.

[19] J. van der Hoeven. Relax, but don't be too lazy. *J. Symb. Comput.*, 34(6):479–542, 2002.

[20] J. van der Hoeven. Relaxed multiplication using the middle product. In *ISSAC'03*, pages 143–147. ACM, 2003.

[21] J. van der Hoeven. New algorithms for relaxed multiplication. *J. Symb. Comput.*, 42(8):792–802, 2007.

[22] J. van der Hoeven. Relaxed resolution of implicit equations. Technical report, HAL, 2009. http://hal.archives-ouvertes.fr/hal-00441977.

[23] J. van der Hoeven. From implicit to recursive equations. Technical report, HAL, 2011. http://hal.archives-ouvertes.fr/hal-00583125.

[24] J. van der Hoeven. Faster relaxed multiplication. Technical report, HAL, 2012. http://hal.archives-ouvertes.fr/hal-00687479.

[25] K. S. Kedlaya and C. Umans. Fast polynomial factorization and modular composition. *SIAM J. Comput.*, 40(6):1767–1802, 2011.

[26] P. Kirrinnis. Fast algorithms for the Sylvester equation $AX - XB^t = C$. *Theoretical Computer Science*, 259(1–2):623–638, 2001.

[27] R. Lercier and T. Sirvent. On Elkies subgroups of ℓ-torsion points in elliptic curves defined over a finite field. *Journal de Théorie des Nombres de Bordeaux*, 20(3):783–797, 2008.

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

[29] V. Shoup. NTL 5.5.2: A library for doing number theory, 2009. www.shoup.net/ntl.

[30] A. Stothers. *On the Complexity of Matrix Multiplication*. PhD thesis, University of Edinburgh, 2010.

[31] V. Vassilevska Williams. Breaking the Coppersmith-Winograd barrier. Technical report, 2011.

[32] A. Waksman. On Winograd's algorithm for inner products. *IEEE Trans. On Computers*, C-19:360–361, 1970.

[33] W. Wasow. *Asymptotic expansions for ordinary differential equations*. John Wiley & Sons, 1965.

Chapter 6

Polynomial Root-Finding for Nonlinear Equations

List decoding, for the Sudan / Guruswami-Sudan as well as for the Rudra-Guruswami algorithms, requires to perform bivariate, resp. multivariate polynomial root-finding for nonlinear polynomials, whose roots may additionally have multiplicities. There are several barriers to incorporate directly fast root-finding techniques based on Newton iteration; in this chapter, we analyze these questions using Newton polygons and introduce in particular a heuristic to deal with the folded case. Experimental results show the soundness of this approach.

6.1 Introduction

In this chapter, we consider the problem of root-finding for Reed-Solomon codes and folded Reed-Solomon codes. As input, we are given a polynomial Q in $\mathbb{F}[x, z_1, \ldots, z_s]$, for some field $\mathbb{F}$, and a field element γ; this polynomial was obtained by interpolation at a set of points of the form $(\alpha_i, y_{s(i-1)+1}, \ldots, y_{si})$ that lies in $\mathbb{F}^{s+1}$. The output is a polynomial f such that

$$Q(x, f(x), f(\gamma x), \ldots, f(\gamma^{s-1}x)) = 0; \tag{6.1}$$

the case $s = 1$ corresponds to the Sudan and Guruswami-Sudan algorithms, and $s > 1$ holds for the Guruswami-Rudra folded codes.

Following the ideas of Chapter 5, we will compute f using lifting techniques. In that chapter, we were dealing with linear equations, and under suitable assumptions, we were able to give fast algorithms (quasi-linear time in the degree of f). In this

chapter, we deal with more general situations, dropping in particular the linearity assumption. Our main results are the following:

- For non-linear equations, we prove using Newton iteration techniques that provided certain smoothness assumptions hold, we can compute a root of Eq. (6.1) in time quasi-linear in the degree of f. This is well-known for the "algebraic" case $s = 1$, where the equation we solve becomes simply $Q(x, f(x)) = 0$.

- Without the smoothness assumption, we show how Newton polygon constructions can be applied; for this, we rely on previous results due to Kung-Traub [16] and Cano-Fortuny Ayuso [9]. We relate this approach to previous algorithms due to Roth and Ruckenstein [23] (for $s = 1$) and Beelen and Brander [2] for the folded case.

- Finally, we present a heuristic dedicated to the folded case, that greatly simplifies the computations based on Newton polygon techniques, and works remarkably well in practice.

For convenience, we describe first in Section 6.2 the non-folded case, with $s = 1$; in that case, almost all results we give were known, and our main contribution is to highlight the connection between a previous algorithm due to Roth and Ruckenstein [23] and classical Newton polygon construction. In Section 6.3 we show how these techniques extend to the folded case; finally, in the last section, we introduce our heuristic for the folded case.

In this chapter, 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 [28, **S**8.3]; using Fast Fourier Transform, $\mathsf{M}(n)$ can be taken to be $O(n \log(n) \log\log(n))$ [29, 30].

6.2 The classical case ($s = 1$)

This section describes root finding techniques in the non-folded case, that is, when $s = 1$. Most of the contents is well-known, but we make a few remarks about the relations between some existing algorithms that seem to be new.

We are given here a polynomial Q in $\mathbb{K}[x, z]$, and we look for polyomials f of degree less than k in $\mathbb{K}[x]$ such that $Q(x, f(x)) = 0$. A direct approach consists in factoring Q, since in this bivariate case, $z - f(x)$ must be a factor of Q in $\mathbb{K}[x, z]$; this is how the polynomial time complexity results are achieved in [24] and [13]. Now,

bivariate factorization algorithms boil down to univariate factorization, lifting and recombination; it is thus natural to remark that in order to find roots of the form $z = f(x)$, one may directly lift power series solutions from the known values (α_i, y_i), and identify those that are polynomials of degree less than k. We discuss this idea here, first in the case of Sudan's algorithm, then for the Guruswami-Sudan algorithm.

6.2.1 Root-finding when $m = 1$

We start by briefly describing the case $m = 1$. This case was investigated in [1] by Augot and Pecquet, who showed how to use fast root-finding techniques. That paper discusses more than Reed-Solomon codes (the authors consider algebraic geometric codes); for the particular case of Reed-Solomon codes, the lifting process boils down to a standard form of Newton iteration.

Fix an index i in $\{1, \ldots, n\}$, so that $Q(x_i, \alpha_i) = 0$. After replacing Q by $Q_i = Q(x + x_i, z + \alpha_i)$, the algorithm computes a sequence of polynomials $f_{(j)}$ in $\mathbb{K}[x]$ such that $\deg(f_{(j)}) < 2^j$ and $Q_i(x, f_{(j)}) = 0 \bmod x^{2^j}$ holds for $j = 0, \ldots$ This is done as follows:

- For $j = 0$, take $f_{(0)} = 0$.
- To deduce $f_{(j+1)}$ from $f_{(j)}$, write $f_{(j+1)} = f_{(j)} + h_{(j)}$, with $h_{(j)} = 0 \bmod x^{2^j}$. Writing the Taylor expansion of Q_i at $(x, f_{(j)})$, and using the fact that $h_{(j)}^2 = 0 \bmod x^{2^{j+1}}$, we deduce

$$Q_i(x, f_{(j+1)}) = Q_i(x, f_{(j)}) + h_{(j)} \frac{\partial Q_i}{\partial z}(x, f_{(j)}) \bmod x^{2^{j+1}}.$$

  Assuming that $\partial Q_i / \partial z(0, 0)$ is nonzero, we deduce

$$h_{(j)} = -\frac{Q_i(x, f_{(j)})}{\frac{\partial Q_i}{\partial z}(x, f_{(j)})} \bmod x^{2^{j+1}}.$$

The above process is none other than Newton iteration applied to Q_i; in order for the iteration to be well-defined, the partial derivative $\partial Q_i / \partial z(0, 0) = \partial Q / \partial z(x_i, \alpha_i)$ must be nonzero. While we expect this condition to hold "in general" for all i, the assumption $Q(x, f(x)) = 0$ is not enough to guarantee it. Theorem 5 in [1] proves that if Q is chosen with minimal z-degree, there exists at least one index i for which $\partial Q / \partial z(x_i, \alpha_i)$ is nonzero.

Let $\ell = \deg(Q, z)$. When the required condition $\partial Q / \partial z(x_i, \alpha_i) \neq 0$ holds, then the jth step of this Newton iteration takes $O(\ell)$ operations (additions, subtractions,

divisions) on power series of precision 2^j. Each such power series operation takes $O(\mathsf{M}(2^j))$ operations in $\mathbb{K}$, for a total of $O(\ell\mathsf{M}(2^j))$. If we stop the iteration as soon as $2^\ell \geqslant k$, the total is thus $O(\ell\mathsf{M}(k))$ operations in $\mathbb{K}$.

6.2.2 Root-finding when $m > 1$

By construction, in Guruswami-Sudan's algorithm, the partial derivatives $\partial Q/\partial z(x_i, \alpha_i)$ all vanish, so we cannot apply Newton iteration directly in that case. In this subsection, we present workarounds to this problem, first by means of the Roth-Ruckenstein algorithm, then using an older algorithm due to Kung and Traub; we then point out that these two algorithms are essentially the same, with a slight advantage for the latter.

As in the previous subsection, we fix an index i once and for all and we translate the origin to (x_i, α_i). In order to avoid keeping track of the index i, we still write Q for the bivariate polynomial obtained after translation, so that our problem boils down to finding a polynomial $f(x)$ such that $f(0) = 0$ and $Q(x, f(x)) = 0$. Remark that already for such simple polynomials as $Q(x, z) = z^2 - x^2$, there is no way to uniquely compute a root $z = f(x)$ such that $f(0) = 0$ (here, both $z = x$ and $z = -x$ are solutions).

Roth and Ruckenstein's algorithm. We start with Roth and Ruckenstein's algorithm [23], which computes the coefficients of the message polynomial f one after another. Following the original presentation, the algorithm uses an array ψ of size k to store its results, and outputs candidates for f one after the other (formally, they may be appended to an output list). The top-level call starts at index $i = 0$, and subsequent calls involve higher values of i. Finally, in order to distinguish its behavior from the one of the upcoming Newton-Puiseux algorithm, the input is called here M.

Algorithm 10: ROTH & RUCKENSTEIN(M, k, ψ, i)

find the largest integer r such that x^r divides M
$M = \frac{M}{x^r}$
$\wp$ = all roots of the univariate polynomial $M(0, z)$ in $\mathbb{K}$
for each $\varsigma \in \wp$ **do**
  $\psi[i] = \varsigma$
  if $i = k - 1$ **then**
    output ψ
  else
    $\widetilde{M}(x, z) = M(x, xz + \varsigma)$
    ROTH & RUCKENSTEIN$(\widetilde{M}, k, \psi, i + 1)$
  end if
end for

The following example illustrates this algorithm. Let

$$\begin{aligned} M(x,z) &= z^4 + (-3x^3 - 2x^2 - 3x)z^3 + (2x^6 + 3x^5 + 10x^4 + 6x^3 + 2x^2)z^2 \\ &\quad + (-6x^7 - 9x^6 - 9x^5 - 4x^4)z + 4x^8 + 6x^7 + 2x^6 \\ &= (z - x)(z - 2x)(z - x^2 - x^3)(z - x^2 - 2x^3), \end{aligned}$$

and consider finding roots $z = f(x)$ with $f(0) = 0$. Of course, the input of the algorithm is the expanded form given first; on the factored form, we see that there are four such roots. Figure 6.1 shows the recursion tree of Algorithm 10 on this input, with the value at each node corresponding to the root ς chosen to enter the corresponding recursive call, and with the recursion depth i given vertically. Nodes with several children correspond to recursive steps where several roots ς exist; the children correspond to roots that are "separated" at the given recursive call.

In what follows, we will refer to the *label* of a node to describe which root was used to enter the corresponding recursive call (here, the labels of the children of the root are 1, 2, 0); to each node N is also associated the polynomial M_N that was used as input to said node.

The Newton-Puiseux algorithm. Next, following Kung and Traub's presentation [16], we present a simplified version of the so-called *Newton-Puiseux* algorithm, that achieves the same purpose, and which follows the classical constructive proof that the field of Puiseux series is algebraically closed (this approach was already

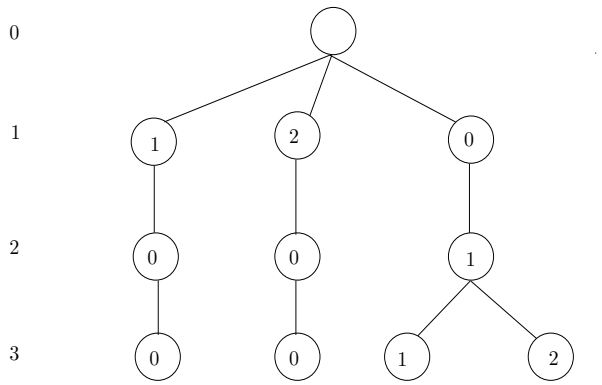


Figure 6.1: Execution of Roth and Ruckenstein's algorithm

mentioned in Pecquet's thesis [19]). For all facts not proved below, and in particular for the correctness of the algorithm, please see [16].

The algorithm relies on a construction called the Newton polygon of Q: this is a polygon in the plane, obtained as the convex hull of all sets $(i, j) + \mathbb{N}^2$, for (i, j) such that $x^i z^j$ appears as a monomial in Q with a nonzero coefficient. The Newton polygon for the polynomial Q in the previous example is given in Figure 6.2, where the vertices are $(0, 4)$, $(2, 2)$ and $(6, 0)$, corresponding to monomials z^4, $x^2 z^2$ and x^6.

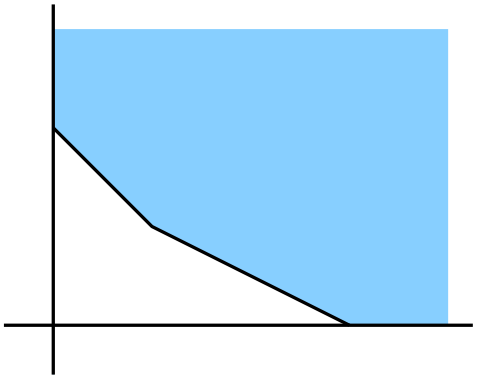


Figure 6.2: The Newton polygon of polynomial Q

The final notion we need is the *slopes* of the Newton polygon. For us, those will be positive rational numbers η, such that the Newton polygon of Q admits an edge of slope η; the only slopes that we will consider are those of the form $\eta = -1/\lambda$, for λ either a non-negative integer (in our example, there are two such slopes, -1 and $-1/2$). We also will consider vertical edges, which correspond to $\eta = -\infty$ and $\lambda = 0$. The key property of these slopes is that if $f = f_i x^i + \cdots$ is a solution of $Q(x, f(x)) = 0$, with $f_i \neq 0$, then $-1/i$ is a slope of the Newton polygon, and conversely.

After these preliminaries, we can present the Newton-Puiseux algorithm: it finds all slopes η as before, and for any such slope finds the corresponding coefficient f_i; the process is then continued recursively. Each call to the algorithm adds one new coefficient to the current solution f; the initial call is made with $f = 0$. For correctness, see for instance [16] or [27]; remark that the original algorithms compute more than we do (they output Puiseux series, that is, series with fractional exponents; we do not need them here).

As before, the algorithm enters with a current solution, stored in an array ψ.

Algorithm 11: NEWTON-PUISEUX EXPANSION(Q, k, ψ, i)

if Q is of the form cz^n for some c in $\mathbb{K}$ and $n \geqslant 1$ **then**
  output ψ
end if
$P = \text{NewtonPolygon}(Q)$
$\Lambda = \{-1/\eta \mid \eta \in \text{Slopes}(P) \text{ such that } -1/\eta \text{ is in } \mathbb{N}\}$
for each integer slope $\lambda \in \Lambda$ **do**
  if $i + \lambda \geqslant k$ **then**
    output ψ
  else
    let $Q_e = Q(x, x^\lambda w)$, where w is a new variable
    find the largest integer r such that x^r divides Q_e
    let $\wp'$ be the roots in $\mathbb{K}$ of the coefficient of x^r in Q_e
    for each root $\varsigma \neq 0 \in \wp'$ **do**
      $\psi[i + \lambda] = \varsigma$
      $\widetilde{Q} = Q(x, x^\lambda(\varsigma + z))/x^r$
      NEWTON-PUISEUX EXPANSION$(\widetilde{Q}, k, \psi, i + \lambda)$
    end for
  end if
end for

As an example, consider again the polynomial

$$Q(x, z) = (z - x)(z - 2x)(z - x^2 - x^3)(z - x^2 - 2x^3)$$

given before, while keeping in mind that if this polynomial was given as input to the algorithm, it would be given in its expanded form. Figure 6.3 shows the recursion tree for Algorithm 11, where each child corresponds to an output, the left-hand side

labels represent powers of x and the value at a node is the coefficient we obtained prior to entering that node.

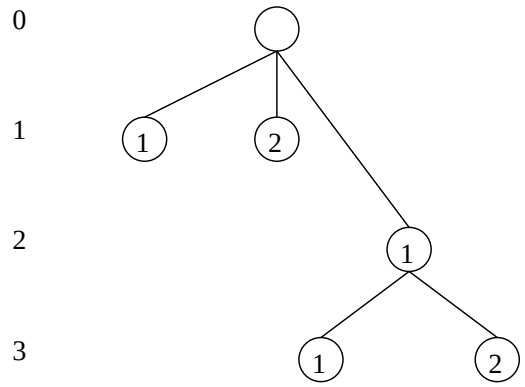


Figure 6.3: Execution of the Newton-Puiseux algorithm

A comparison. The following precise relation between these two algorithms seems to have been unnoticed before: we claim that Algorithms 10 and 11 compute essentially the same polynomials. In order to prove this claim, we will need the following lemma on the Newton-Puiseux algorithm.

Lemma 1. *Suppose that A, B are in $\mathbb{K}[x, z]$, and that $A(x, z) = x^e B(x, xz)$, for some e in $\mathbb{Z}$. Then, the executions of Algorithm 11 with input A and B are the same, except that the outputs are shifted by one power of x, and that the branch of the recursion tree corresponding to the slope $-\infty$ for B (if it exists) does not appear for A.*

Proof. That the outputs are shifted by one power of x follows directly from the fact that the roots of A in z are exactly the roots of B divided by x. The second claim follows from drawing both Newton polygons, and observing that slopes of the form $1/u$ for B become slopes of the form $1/(u-1)$ for A. □

We can now justify our claim, and we thus suppose that a polynomial Q is given as input to both algorithm. Our claim is the following: *there exist a one-to-one correspondance between the nodes N of Roth and Ruckenstein's execution tree that are not labelled by 0 and the nodes N' of the Newton-Puiseux algorithm, such that the polynomials M_N and $Q_{N'}$ at corresponding nodes are related as in Lemma 1.*

We do a proof by induction on the depth of the execution trees. On input Q, Roth and Ruckenstein's algorithm will compute a set of roots $\wp$, which may contain zero. We discuss the nonzero and zero roots separately.

- The nonzero roots correspond to those with $\lambda = 0$ in the Newton-Puiseux algorithm. For such a root ς_i, Roth and Ruckenstein's algorithm will do a recursive call on $\widetilde{M}_i = Q(x, xz + \varsigma_i)$, whereas the Newton-Puiseux algorithm will call itself with the polynomial $\widetilde{Q}_i = Q(x, z + \varsigma_i)/x^{r_i}$, for some integer r_i

  We have $\widetilde{M}_i = x^{r_i}\widetilde{Q}_i(x, xz)$ and by construction, the polynomial $\widetilde{Q}_i$ does not have any infinite slope, so by Lemma 1, the execution of Newton-Puiseux's algorithm on input $\widetilde{Q}_i$ is the same as the one on input $\widetilde{M}_i$. We are thus comparing the Roth-Ruckenstein and Newton-Puiseux's algorithms on the same input $\widetilde{M}_i$, so by induction we have correspondence between the execution trees anchored at the polynomials $\widetilde{M}_i$, as claimed.

- Consider now the zero root (if it belongs to $\wp$), for which Roth and Ruckenstein's algorithm will do a recursive call on $\widetilde{M}_0 = Q(x, xz)$; remark that the Newton-Puiseux algorithm does not make a recursive call on this polynomial.

  The induction assumption implies that if we run both algorithms with input $\widetilde{M}_0$, there is a one-to-one correspondance between the execution trees of both algorithms, where for the Roth-Ruckenstein algorithm only nodes with nonzero labels should be considered.

  By Lemma 1, we deduce that executing the Newton-Puiseux algorithm on $\widetilde{M}_0 = Q(x, xz)$ is the same as calling it on Q and considering only non-infinite slopes. This allows us to conclude the proof, noting the top-level of the Newton-Puiseux algorithm will be connected to the top-level ot the Roth-Ruckenstein algorithm, and that the node corresponding to $\widetilde{M}_0$ will not appear in our correspondance (as it is attached to the zero root).

In particular, both algorithms involve essentially the same polynomials Q at the internal nodes, except for the fact that the Roth-Ruckenstein algorithm computes some unnecessary polynomials at nodes corresponding to zero roots; the relation between Figures 6.1 and 6.3 is thus of a general nature.

A refinment. An issue with these algorithms is that it only computes one new coefficient at a time, whereas we saw that the number of correct terms obtained through Newton iteration doubles at each step. As it turns out, it is possible to obtain the same convergence rate as for Newton iteration even for situations with multiplicities.

The key idea is that, even though the coefficients of f may not be uniquely determined for the first few indices, thus preventing the application of Newton iteration,

it becomes possible to run this iteration after a suitable number of coefficients have been computed, so that all roots have been "separated".

This was noticed by both Roth-Ruckenstein and Kung-Traub. For instance, in the Roth-Ruckenstein algorithm, once $M(0, z)$ has only one simple root at step i, this will remain the case for all further indices; the same holds for the Newton-Puiseax algorithm: at this stage, as pointed out by Kung and Traub, it becomes possible to apply Newton iteration. It is even possible to bound the first index i at which this happens, see [16].

6.3 The folded case ($s > 1$)

We now consider folded Reed Solomon codes, so that $s > 1$. Let thus γ be in $\mathbb{K}$; usually, we take γ of high order, typically a primitive element of $\mathbb{K}$. In the folded case, we specifically take $\alpha_i = \gamma^{i-1}$ for $1 \leqslant i \leqslant n$, and we assume (for simplicity) that s divides n. In order to list-decode folded Reed-Solomon codes, we follow the same two steps as before: first, compute a multivariate polynomial $Q(x, z_1, \ldots, z_s)$ such that

$$Q(\alpha_i, y_{s(i-1)+1}, \ldots, y_{si}) = 0$$

holds with multiplicity at least m for all $1 \leqslant i \leqslant n/s$ (for simplicity we assumed that the folding parameter s and the number of variables $\mathbf{z}$ are the same; further generalizations are possible). After interpolating such a polynomial, we have to find a polynomial $f(x)$ such that

$$Q(x, f(x), f(\gamma x), \ldots, f(\gamma^{s-1} x)) = 0,$$

and keep it if $f(\alpha_i) = y_i$ for at least $(n - e)$ number of $i \in \{1, \ldots, n\}$ [2] where e is the number of tolerable errors.

The original approach to root-finding in this case uses a reduction to polynomial factorization over the extension of $\mathbb{K}$ defined by $\mathbb{K}'(\eta) = \mathbb{K}[x]/(x^{q-1} - \gamma)$, where $q = |\mathbb{K}|$; the polynomial to factor is $Q(\eta, z, z^q, \ldots, z^{q^{s-1}})$. While this allows for polynomial-time factoring (provided we see q and s as constant), the degree expansion makes the root-finding very challenging in practice.

In what follows, we propose methods which do not involve such large degree polynomals, but whose complexity can unfortunately not be controlled as precisely. Our methods are inspired by the case $s = 1$ seen above, but require adaptations, since the equation we have to solve is not algebraic anymore. Such equations are

often called *q-difference* equations; as it turns out, an approach based on Newton polygon techniques has recently been developed for this context by Cano and Fortuny Ayuso [9] over $\mathbb{K} = \mathbb{C}$, inspired by similar techniques first introduced for differential equations (compare for instance with [8, 10] and references therein). We will follow these ideas, and point out however where they fail in our context where $\mathbb{K}$ is finite.

In all that follows, in order to facilite the root-finding, *we assume that* $f(0) = 0$, that is, we assume that we know the first term of f; this is a mild restriction, as we could recover f_0 by factoring the univariate polynomial $Q(0, z, \ldots, z)$ (provided this polynomial is nonzero).

6.3.1 The linear case

As when $s = 1$, our algorithms will rely on linearization techniques; in the present case, we will use the divide-and-conquer method presented in Chapter 5 to solve linear q-difference equations: given polynomials $A_0, \ldots, A_s$ and an integer ℓ, we want to find solutions h to

$$A_s h(\gamma^{s-1}x) + \cdots + A_1 h(x) + A_0 = 0 \bmod x^\ell.$$

The algorithm first goes through a recursive divide-and-conquer process, and outputs a solution that is parametrized by a vector of indeterminates h_i. Then, we resolve the possible relations between the h_i as a post-processing by solving a linear system. In order to highlight the potential difficulties, we recall here the first step (divide-and-conquer) of this algorithm (see Algorithm 12).

For $j = 1, \ldots, s$, let u_j be the constant coefficient of A_j. Then, we can remark that the divide-and-conquer algorithm attempts divisions by elements of the form

$$u_s\gamma^{(s-1)i} + \cdots + u_2\gamma^i + u_1,$$

for $i = 0, \ldots, \ell - 1$ (just like a direct, iterative algorithm would). For further use, if we introduce the polynomial

$$P = u_1 + u_2 x + \cdots + u_s x^{s-1}, \tag{6.2}$$

we observe that these values are of the form $P(\gamma^i)$; P will be called the *critical equation*. Let us illustrate this algorithm on the example

$$h(-x) + (1 + x^2)h(x) + A(x) = 0 \bmod x^4,$$

Algorithm 12: RDAC$(A_0, \ldots, A_s, \gamma, i, \ell)$

Require: $A_0, \ldots, A_s, R \in \mathbb{K}[x]$, γ in $\mathbb{K}$
Ensure: $h(x)$ such that $A_s h(\gamma^{s-1}x) + \cdots + A_1 h(x) + A_0 = 0 \bmod x^\ell$

if $\ell = 1$ **then**
  if $(A_s + \cdots + A_1)(0) \neq 0$ **then**
    return $-\frac{A_0(0)}{(A_s+\cdots+A_1)(0)}$
  else
    return a new variable h_i
  end if
else
  let $n = \lceil \ell/2 \rceil$ and $p = \ell - n$
  $h_0 = \mathrm{RDAC}(A_0, \ldots, A_s, i, n)$
  for $i = 1, \ldots, s$ **do**
    $A'_i = \gamma^{n(i-1)} A_i$
  end for
  $R(x) = (A_s h_0(\gamma^{s-1}x) + \cdots + A_1 h_0(x) + A_0)/x^n$
  $h_1 = \mathrm{RDAC}(-R, A'_1, \ldots, A'_s, i+n, p)$
  return $h = h_0 + x^n h_1$
end if

where we took $s = 2$ and $\gamma = -1$ (which is not a primitive element, but the algorithm would not use this fact in any case); we write $A = \sum_i a_i x^i$. In this case, we have $u_1 = u_2 = 1$, so that $P = 1 + x$; the divide-and-conquer process attempts divisions by the values $P((-1)^i)$, which are $2, 0, 2, 0, \ldots$. The output of the recursive process is

$$h = \frac{-a_0}{2} + h_1 x + \frac{a_0 - 2a_2}{4} x^2 + h_3 x^3,$$

where h_1 and h_3 are indeterminates introduced for $i = 1$ and $i = 3$. In the post-processing step, we evaluate the given equation at such an h, which gives

$$a_1 x + (h_1 + a_3) x^3 = 0.$$

Thus, if $a_1 \neq 0$, there is no solution. If $a_1 = 0$, the solutions are

$$h = \frac{-a_0}{2} - a_3 x + \frac{a_0 - 2a_2}{4} x^2 + h_3 x^3,$$

for any h_3.

We already discussed the complexity of this algorithm in Chapter 5; when no value of the form $P(\gamma^i)$ vanishes, the running time is quasi-linear, of the form $O(s\mathsf{M}(k)\log(k))$.

In general, the running time will depend on the number of coefficients of the form $P(\gamma^i)$ that vanish, for $i = 0, \ldots, k-1$. We cannot give a sharp estimate, but we remark that if γ has multiplicative order at least k, then there are less than s such indices: indeed, P has degree less than s, so it has less than s roots in $\mathbb{K}$, and for any such root r, there is at most one value i in $\{0, \ldots, k-1\}$ such that $\gamma^i = r$.

6.3.2 The regular case

We now discuss the case of non-linear equations, first under a regularity assumption. Let as above Q be the given interpolating polynomial, and assume we look for f such that

$$Q(x, f(x), f(\gamma x), \ldots, f(\gamma^{s-1}x)) = 0.$$

Our main regularity assumption, inspired by that given in the case $s = 1$, is the following: *the partial derivatives $\partial Q/\partial z_i$ do not all vanish at $(0, \ldots, 0)$.*

As before, we assume that $f(0) = 0$, so that f is known modulo x. More generally, suppose that we know $\tilde{f} = f \bmod x^\ell$, for some precision $\ell \in \mathbb{N}$; we look for a solution f with higher precision of the form $f = \tilde{f} + h$, where $h = 0 \bmod x^\ell$. Starting from

$$Q(x, \tilde{f}(x) + h, \tilde{f}(\gamma x) + h(\gamma x), \ldots, \tilde{f}(\gamma^{s-1}x) + h(\gamma^{s-1}x)) = 0$$

a Taylor series expansion gives us

$$\begin{aligned}
& \frac{\partial Q}{\partial z_1}(x, \tilde{f}(x), \tilde{f}(\gamma x), \ldots, \tilde{f}(\gamma^{s-1}x))h(x) \\
+ \; & \frac{\partial Q}{\partial z_2}(x, \tilde{f}(x), \tilde{f}(\gamma x), \ldots, \tilde{f}(\gamma^{s-1}x))h(\gamma x) \\
+ \; & \ldots \\
+ \; & \frac{\partial Q}{\partial z_s}(x, \tilde{f}(x), \tilde{f}(\gamma x), \ldots, \tilde{f}(\gamma^{s-1}x))h(\gamma^{s-1}x) \\
+ \; & Q(x, \tilde{f}(x), \tilde{f}(\gamma x), \ldots, \tilde{f}(\gamma^{s-1}x)) = 0 \bmod x^{2\ell},
\end{aligned}$$

since any product of the form $h(\gamma^i x)h(\gamma^j x)$ is zero modulo $x^{2\ell}$. Let

$$\begin{aligned}
A_0(x) &= Q(x, \tilde{f}(x), \tilde{f}(\gamma x), \ldots, \tilde{f}(\gamma^{s-1}x)) \bmod x^{2\ell} \\
A_1(x) &= \frac{\partial Q}{\partial z_1}(x, \tilde{f}(x), \tilde{f}(\gamma x), \ldots, \tilde{f}(\gamma^{s-1}x)) \bmod x^{2\ell} \\
A_2(x) &= \frac{\partial Q}{\partial z_2}(x, \tilde{f}(x), \tilde{f}(\gamma x), \ldots, \tilde{f}(\gamma^{s-1}x)) \bmod x^{2\ell} \\
&\ldots \\
A_s(x) &= \frac{\partial Q}{\partial z_s}(x, \tilde{f}(x), \tilde{f}(\gamma x), \ldots, \tilde{f}(\gamma^{s-1}x)) \bmod x^{2\ell}.
\end{aligned}$$

Thus, we are looking for $h(x)$ such that

$$A_s(x)h(\gamma^{s-1}x) + \cdots + A_2(x)h(\gamma x) + A_1(x)h(x) + A_0(x) = 0 \bmod x^{2\ell},$$

which can be done using the algorithm for the linear case given previously. This leads to Algorithm 13; in Algorithm 14, we present the post-processing: applied to a polynomial f, some of whose coefficients are indeterminates, it outputs all possible values of these indeterminates such that f is a root of Q. This algorithm uses a black-box called Solve for solving non-linear equations over $\mathbb{K}$; many solutions are available to this effect, from Gröbner bases techniques to triangular decomposition algorithms.

Algorithm 13: Root Computation(Q, s, k, γ)

1: $t = 2$
2: $\tilde{f} = 0$
3: **while** $t < k$ **do**
4: $\quad A_0 = Q(x, \tilde{f}(x), \ldots, \tilde{f}(\gamma^{s-1}x))$
5: $\quad$ **for** $i = 1, \ldots, s$ **do**
6: $\quad\quad A_i = \partial Q/\partial z_i(x, \tilde{f}(x), \ldots, \tilde{f}(\gamma^{s-1}x))$
7: $\quad$ **end for**
8: $\quad f_{tmp} = \mathrm{RDAC}(A_{s+1}(x), A_s(x), \ldots, A_2(x), A_1(x), 0, t)$
9: $\quad \tilde{f} = f_{tmp} + \tilde{f}$
10: $\quad t = 2t$
11: **end while**
12: **return** $\tilde{f}$

Similarly the previous subsection, let $u_1, \ldots, u_s$ be the values $\partial Q/\partial z_i(0, \ldots, 0)$, for $i = 1, \ldots, s$. Then, for every call to Algorithm RDAC, the critical equation P will always be the same, namely $P = u_1 + u_2 x + \cdots + u_s x^{s-1}$. Our regularity assumption

Algorithm 14: FILTER(Q, f, k, γ)

1: let $C = (c_1, \ldots, c_s)$ be the coefficients in x of $Q(x, f(x), f(\gamma x), \ldots, f(\gamma^{s-1}x))$
2: let $(w_i)_{i \in I}$ be the indeterminates in C
3: **return** SOLVE(C)

implies that P is nonzero, so it has finitely many roots in $\mathbb{K}$, so as in the case of linear equations, we can assert that overall, only $O(k)$ divisions by zero will be attempted, so $O(k)$ indeterminates will appear in algorithm FILTER. However, the cost of the whole process remains difficult to pin down, due to the non-linear manner in which the indeterminate coefficients are handled.

6.3.3 The general case

When the regularity assumption of the previous subsection does not hold, *every* step will entail a division by zero, so that the algorithm does not do any sensible computation and just return a series with indeterminate coefficients, and that the post-processing phase will have to do all the work. In this subsection, we give an algorithm inspired by the Newton-Puiseux algorithm presented before, that can handle such situations better (although some steps will still require the introduction of indeterminates).

In [2], Beelen and Brander gave a recursive algorithm of Hensel lifting that can compute f, inspired by Roth and Ruckenstein's algorithm; at steps where the value of the coefficient f_i cannot be determined, Beelen and Brander test all elements of the underlying field (which is similar to our use of indeterminate coefficients for such steps). Compared to that algorithm, the main difference in ours lies in the use of a Newton polygon construction, making our algorithm an extension of the Newton-Puiseux algorithm presented before for the case $s = 1$.

Let Q be a polynomial in $\mathbb{K}[x, z_1, z_2, \ldots, z_s]$, which we write as

$$Q = \sum_{(\alpha, \boldsymbol{\rho}) \in S_Q} a_{\alpha, \boldsymbol{\rho}} x^\alpha z_1^{\rho_1} z_2^{\rho_2} \ldots z_s^{\rho_s},$$

where S_Q is the *support* of Q, each $a_{\alpha,\boldsymbol{\rho}}$ is in $\mathbb{K} - \{0\}$ and $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_s)$. Let us define the *points* of Q to be the set

$$\mathcal{P}(Q) = \{P_{\alpha,\boldsymbol{\rho}} \mid (\alpha, \boldsymbol{\rho}) \in S_Q\},$$

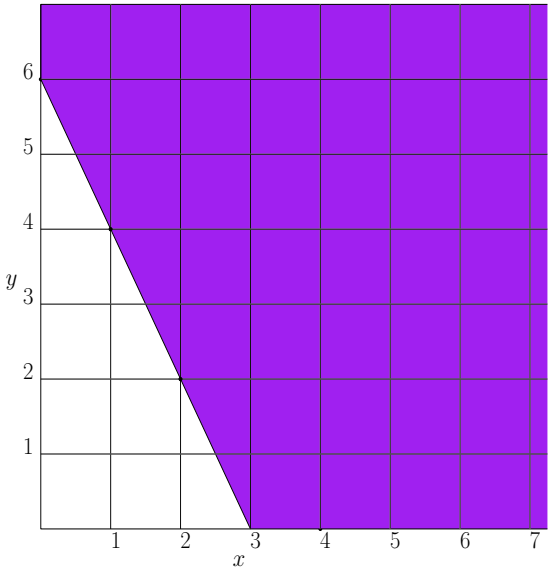
where we write

$$P_{\alpha,\boldsymbol{\rho}} = (\alpha, \rho_1 + \rho_2 + \cdots + \rho_s) \in \mathbb{N}^2.$$

The *Newton polygon* $\mathcal{N}(Q)$ of Q is the convex hull of the set

$$\bigcup_{P \in \mathcal{P}(Q)} (P + \mathbb{N}^2) \quad \mathbb{N} \text{ is integer ring;}$$

the *slopes* of this polygon are defined as in the previous section. Figure 6.4 gives an example of a Newton polygon of a equation having three variables.



Newton polygon for the equation :

$$x^4 - x^3y^2 - x^3y - x^3z^2 - x^3z + x^2y^3 + x^2y^2z^2 + x^2yz + x^2yz^2 + x^2yz + x^2z^3 - xy^3z^2 - xy^3z - xy^2z^3 - xyz^3 + y^3z^3$$

Figure 6.4: Newton polygon

Inspired by the Newton-Puiseux algorithm given before, Algorithm 15 below computes the power series roots of Q using its Newton polygon. As before, the computation is based on the geometry of the Newton polygon; this time however, not only the edges but also the vertices of the Newton polygon may contribute to the solutions (this is because in this 2-dimensional representation, several monomials of Q may correspond to a given vertex). Thus, for $\lambda \in \mathbb{N}$, we define the following:

- $L(Q, \lambda)$ is the straight line with slope $-\frac{1}{\lambda}$ which intersects $\mathcal{N}(Q)$ at either a vertex or an edge;
- $\Phi_{Q,\lambda}(w) = \sum_{(\alpha,\boldsymbol{\rho})} a_{\alpha,\boldsymbol{\rho}} \gamma^{(\rho_2+2\rho_3+\cdots+(s-1)\rho_s)\lambda} w^{\rho_1+\rho_2+\cdots+\rho_s}$, for $P_{\alpha,\boldsymbol{\rho}} \in L(Q, \lambda) \cap$

$\mathcal{N}(Q)$; this is a polynomial in w, which (when nonzero) can be seen as the coefficient of the lowest degree term in x of

$$Q(x, x^\lambda w, (\gamma x)^\lambda w, \dots, (\gamma^{s-1}x)^\lambda w),$$

or equivalently of

$$Q(x, f(x), \dots, f(\gamma^{s-1}x))$$

for f of the form $f = x^\lambda(w + w'x + w''x^2 + \cdots)$.

Remark that the nonzero roots of $\Phi_{Q,\lambda}$ give the possible values w such that our equation may admit a solution of the form $f = x^\lambda(w + w'x + w''x^2 + \cdots)$. As a small example, consider the polynomial $Q = z_1 - z_2$, whose Newton polygon has only one vertex $(0, 1)$ and $\gamma = -1$. When $\lambda = 1$, we obtain $\Phi_{Q,1} = 2w$, which has no nonzero root; this indicates that there is no power series solution f of $f(x) = f(-x)$ of the form $wx + \cdots$, for w nonzero. On the contrary, for $\lambda = 2$, we find $\Phi_{Q,2} = 0$; this shows that, as far as the lowest coefficient in concerned, any w may be suitable for a solution f of $f(x) = f(-x)$ of the form $wx^2 + \cdots$

As this example shows, in this algorithm, as in the cases of the two previous subsections, we may of course come up with outputs f that have indeterminates as coefficients. When this is the case, we use the same post-processing step as in the previous subection.

Correctness follows from the discussion preceding the algorithm; alternatively, one may consult [9], where a more complex algorithm is given (to compute more general solutions than power series).

To conclude, let us mention an analogue to the last remark of the previous section. As we noted there, after sufficiently many initial steps, the solutions get "separated", and the coefficients f_i become uniquely determined. A similar phenomenon happens in the folded case, up to a minor modification; to describe it, we follow Cano and Fortuny Ayuso [9].

Given a solution $f(x) = \sum_{i \geqslant 0} f_i x^{\mu_i} \in \mathbb{K}[x]$ of (6.1), define the sequence of polynomials $Q_0 = Q$ and, for $i \geqslant 0$,

$$Q_{i+1} = Q_i(x, f_i x^{\mu_i} + z_1, \dots, f_i \gamma^{(s-1)i} x^{\mu_i} + z_s).$$

These polynomials are essentially the ones seen during the Newton-Puiseux algorithm (up to change of variables of the form $x \mapsto x^\eta x$, for suitable values of η), but are better suited for the discussion below.

Algorithm 15: NEWTON-PUISEUX EXPANSION(Q, i, k, ψ)

if Q is of the form $cz_1^{\rho_1} \cdots z_s^{\rho_s}$ for some c in $\mathbb{K}$ **then**
  output ψ
end if
if $i \geqslant k$ **then**
  output ψ
end if
replace Q by Q/x^r, where r is the largest integer such that x^r divides Q
for $\lambda = i, \ldots, k-1$ **do**
  if $\Phi_{Q,\lambda}(w) = 0$ **then**
    $\psi[i+\lambda] = w_i$ (w_i is a placeholder for the coefficient)
    $\widetilde{Q} = Q(x, x^\lambda(w_i + z_1), \gamma x^\lambda(w_i + z_2), \ldots, \gamma^{s-1} x^\lambda(w_i + z_s))$
    NEWTON-PUISEUX EXPANSION$(\widetilde{Q}, i+\lambda, k, \psi)$
  else
    let $\wp$ be the roots in $\mathbb{K}$ of $\Phi_{Q,\lambda}$
    for each root $\varsigma \neq 0 \in \wp$ **do**
      $\psi[i+\lambda] = \varsigma$
      $\widetilde{Q} = Q(x, x^\lambda(\varsigma + z_1), \gamma^\lambda x^\lambda(\varsigma + z_2), \ldots, \gamma^{(s-1)\lambda} x^\lambda(\varsigma + z_s))$
      NEWTON-PUISEUX EXPANSION$(\widetilde{Q}, i+\lambda, k, \psi)$
    end for
  end if
end for

For any $i \geqslant 0$, let $p_i = (\alpha_i, \beta_i)$ be the point with highest ordinate at the intersection of the line $L(Q_i, i)$ and the Newton polygon of Q_i. Cano and Fortuny Ayuso proved the following:

- the sequence β_i is non-decreasing;
- there exists $i_0 \geqslant 0$ such that for $i \geqslant i_0$, $\beta_i = \beta_{i_0}$;
- $x^{\alpha_i} z_1^{\rho_1} \cdots z_s^{\rho_s}$ appears with a nonzero coefficient in Q_i, with $\rho_1 + \cdots + \rho_s = \beta_i$ and $\rho_j \geqslant 1$ for some index j, then f is also a root of $R = \partial^{\rho_1 + \cdots + \rho_s - 1} Q / \partial z_1^{\rho_1} \cdots \partial z_j^{\rho_j - 1} \cdots \partial z_s^{\rho_s}$;
- if in addition $\alpha_i = 0$, then R satisfies the assumptions of Subsection 6.3.2

In other words, up to replacing Q by a well-chosen derivative, we are reduced to the regular case; recall however that even in the regular case, some coefficients f_i of f may still be undertermined for those indices i that cancel the critical equation.

The point p_{i_0} is called the *pivot point* associated to Q and f.

6.4 A heuristic

Finally, we propose a heuristic for the folded case, based on the last remarks in the previous section. Based on extensive experiments, our heuristic is the following. Consider $f \in \mathbb{K}[x]$ a polynomial with degree less than k, such and $f(0) = 0$ and $f'(0) \neq 0$; that is, $f = f_1 x + \cdots$. Consider the following assumptions:

- the index i_0 of the pivot point p_{i_0} of Q is equal to zero, i.e. the pivot point (α_0, β_0) associated to Q and f is the point with highest ordinate on $L(Q, 1) \bigcap \mathcal{N}(Q)$
- this pivot point has abcissa 0, that is, $\alpha_0 = 0$.
- the critical equation for the polynomial R obtained by differentiating Q (as in the above section) has no root of the form γ^i, for any $0 < i < k$.

In view of the discussion in the previous section, in that case, it is enough to replace Q by a polynomial R obtained as a suitable derivative of Q in order to be under the assumptions of the "regular" case. Then, we actually do not need to apply the Newton Puiseux algorithm, and we know furthermore that no "division by zero" will occur; the process is summarized in Algorithm 13.

Algorithm 16: LIFT ROOT FROM FRS Q(Q, s, k, γ)

Require:

1: $NP = \mathcal{N}(Q)$
2: let (α, ρ) be the point with highest ordinate on $L(Q, 1) \bigcap \mathcal{N}(Q)$
3: **if** $\rho > 1$ **then**
4: Find a term $a_{j,\rho} x^a z^\rho$ in Q such that $a_{j,\rho} \neq 0$
5: $Q = \frac{\partial^{\rho-1} Q}{\partial z^{\rho-1}}$
6: **end if**
7: **return** ROOT COMPUTATION(Q, s, k, γ)

We implemented the algorithm in magma [3] and run it on a 2.1 GHz AMD Athlon 64 processor; the following tables summarize our results.

We observed that our assumptions above systematically held when Q is obtained by solving a linear system and picking the solution with lowest degree in x. There exist several ways to find such polynomials: a possible approach is to compute a basis of the solution space and find an echelon form (the shape resulting of a Gaussian elimination) that reveals the minimal degree in x; another approach is simply to put

WC		multiplicity			
		3	4	5	%
k	8	938	909	934	92.7
	16	966	843	843	88.4
	32	965	844	715	91.6
	"%"	95.6	86.5	90.5	
C		multiplicity			
		3	4	5	%
k	8	062	91	66	7.3
	16	034	157	157	11.6
	32	035	156	39	8.4
	"%"	04.4	13.5	09.5	

Table 6.1: Number of times a new polynomial Q was needed

an extra constraint that enforces a coefficient of a term x^d to be nonzero and do a binary search over possible values of d.

In our experiments, multiplicity was in the range of $2 \leqslant m \leqslant 5$, the folding parameter was $s = 2$ and the degree of f was in $\{3, 7, 15, 31\}$; the rate was chosen in $\{1/2, 9/10\}$; n was chosen accordingly.

We took our base field $\mathbb{F}$ on next prime of n and run the experiment 1000 times for each k, m and tested for random $f(x)$ with $f(x) \bmod x = 0$. We introduced random errors under the bound given in [14, 2], computed the interpolated polynomial by solving system of linear equations, and then we recover f by applying our algorithm.

Additionally, we always performed the following test on interpolated polynomial Q before calling our algorithm 16: compute the critical equation P for the derivative R of Q and test whether any γ^i for $0 < i \leqslant k$ is a root of it. When we saw any γ^i, for $1 \leqslant i < k$, is a root of P we discarded that interpolated polynomial Q and replaced it by a new polynomial Q: the coefficients of P depend linearly on those of Q, so we could add a linear constraint to our system in order to ensure that P would change.

Table 6.1 shows for how many instances we needed to change a coefficient in P and thus the interpolation polynomial Q. In this table, WC stands for *without change* whereas C stands for *change*: in almost all cases, there was no need to change Q.

Table 6.2 shows the timing required to compute interpolation polynomial as well as computation of root and figure 6.5 depicts the good timing of our algorithm compared to Beelen and Brander's algorithm [2].

Interpolation		multiplicity		
		3	4	5
k	8	.353	1.352	4.365
	16	1.331	5.823	20.295
	32	5.276	28.486	91.414

Root computation		multiplicity		
		3	4	5
k	8	0.001	0.003	0.003
	16	.007	.007	.019
	32	.019	.023	.066

Table 6.2: Timing of Interpolation and Root-Computation

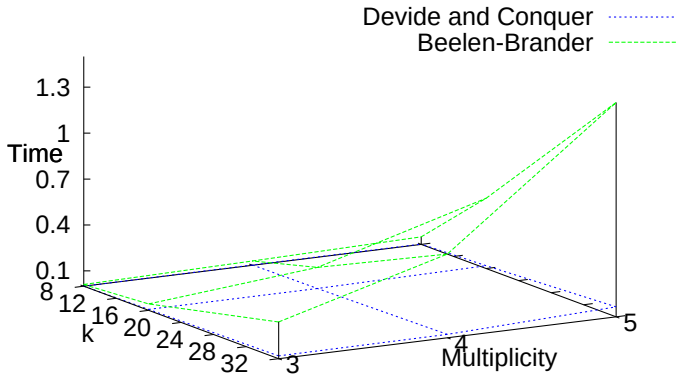


Figure 6.5: Timing between our algorithm and Beelen and Brander's algorithm

Bibliography

[1] D. Augot and L. Pecquet. A Hensel lifting to replace factorization in list-decoding of algebraic-geometric and Reed-Solomon codes. *IEEE Transaction on Information Theory*, 46(7):2605–2614, November-2000.

[2] P. Beelen and K. Brander. Decoding folded Reed-Solomon codes using Hensel lifting. *M. Sala et al. (eds.), Gröbner Bases, Coding, and Cryptography*, pages 389–394, 2009.

[3] W. Bosma, J. Cannon, and C Playoust. The Magma algebra system. I. The user language. *J. Symb. Comp.*, 24(3-4):235–265, 1997.

[4] A. Bostan, G. Lecerf, B. Salvy, É. Schost, and B. Wiebelt. Complexity issues in bivariate polynomial factorization. In ISSAC'04, pages 42–49, ACM, 2004.

[5] A. Bostan, M. F. I. Chowdhury, R. Lebreton, B. Salvy and É. Schost. Power Series Solutions of Singular (q)-Differential Equations. In ISSAC'12. pages 107–113, ACM, 2012.

[6] M. F. I. Chowdhury, C.-P. Jeannerod, V. Neiger, É. Schost and G. Villard. On the Complexity of Multivariate Interpolation with Multiplicities and of Simultaneous Polynopmial Approximations.

[7] R. P. Brent and J.F. Traub. On the complexity of composition and generalized composition of power series. *Siam J. of Computing*, 9(1):54–66, February-1980.

[8] José Cano. On the series defined by differential equations, with an extension of the Puiseux polygon construction to these equations. *Analysis*, 13:103–119, 1993.

[9] José Cano, P. Fortuny Ayuso, Power Series Solutions of Non-Linear q-Difference Equations and the Newton-Puiseux Algorithm. arXiv:1209.0295 [math.AG], Sep-2012.

[10] José Cano. The Newton polygon method for differential equations. 3519/2005:93–114, 2005.

[11] Peter Elias. List decoding for noisy channels. *Technical Report 335*, pages 94–104, September-1957.

[12] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, ISBN: 0521826462, 2003

[13] V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Trans. on Info. Theory,*, 45(6):1757 – 1767, Sep–1999.

[14] V. Guruswami and A. Rudra. Error correction up to the information-theoretic limit. *Commun. ACM*, 52:87–95, March 2009.

[15] J. van der Hoeven. *Transseries and real differential algebra*, volume 1888 of *Lecture Notes in Mathematics*. Springer-Verlag, 2006.

[16] H. T. Kung and J. F. Traub. All algebraic functions can be computed fast. *J. ACM*, pages 245–260, 1978.

[17] K. Lee and M. E. O. Sullivan. List decoding of Reed-Solomon codes from a Gröbner basis perspective. *J. Symb. Comput.*, 43:645–658, September 2008.

[18] F. Parvaresh and A. Vardy. Correcting errors beyond the Guruswami-Sudan radius in polynomial time. In *Proceedings of 46th Annual IEEE Symposium on Foundations of Computer Science FOCS'05*, pages 285 – 294. IEEE Computer Society, 2005.

[19] L. Pecquet *List Decoding of Algebraic Geometric Codes*. PhD Thesis, Université Paris 6, 2001.

[20] A. Poteaux and M. Rybowicz. Good reduction of Puiseux series and complexity of the Newton-Puiseux algorithm over finite fields. In *ISSAC'08*, pages 239–246. ACM, 2008.

[21] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, Jun-1960.

[22] R. M. Roth. *Introduction to Coding Theory*. Cambridge University Press, 2006.

[23] R. M. Roth and G. Ruckenstein. Efficient decoding of Reed-Solomon codes beyond half the minimum distance. *IEEE Transaction on Information Theory*, 46(1):246–257, January-2000.

[24] M. Sudan. Decoding of reed solomon codes beyond the error-correction bound. *JOURNAL OF COMPLEXITY*, 13(CM970439):180193, 1997.

[25] P. V. Trifonov. Efficient interpolation in the Guruswami-Sudan algorithm. *IEEE Transactions on Information Theory,*, 56:4341–4349, August 2010.

[26] N. J. Willis. *Newton-Puiseux algorithm*. MS Thesis, Texas Tech University, 2003.

[27] R. J. Walker. Algebraic Curves. Springer-Verlag, 1950.

[28] J. von zur Gathen and J. Gerhard. Modern Computer Algebra. Cambridge University Press, 1999, 0-521-64176-4.

[29] Cantor, D. G. and Kaltofen, E. On fast multiplication of polynomials over arbitrary algebras. Acta Informatica, 28(7): 693–701, 1991,0001-5903.

[30] Schoenage, A. and Strassen V. Schnelle Multiplikation grosser Zahlen. Computing, 7:281–292, 1971

Chapter 7

Conclusions and future work

In conclusion, we can apply our proposed algorithms for folded Reed Solomon code decoding. Our heuristic in Chapter 6 showed that there exist an interpolated polynomial where we can apply fast root lifting algorithm described in Chapter 5. In order to interpolate such a multivariate polynomial efficiently, we can apply the algorithm described in Chapter 4. The following example shows how to apply fast root lifting algorithm described in Chapter 5. As for example, let our interpolated polynomial is in $\mathbb{F}[x, z_1, z_2, z_3]$ and $f(x)$ be the message polynomial. Then by using the idea of lemma 10 of Chapter 6 and Taylor series expansion, where the known part of f is denoted by $\tilde{f}$ unknown part is denoted by h and ℓ is the precission, we can deduce

$$\tilde{A}(x)h(\gamma^2 x) + \tilde{B}(x)h(\gamma x) + \tilde{C}(x)h(x) = \tilde{D}\ (x) \tag{7.1}$$

where

$$\begin{aligned}
\tilde{D}(x) &= Q(x, \tilde{f}(x), \tilde{f}(\gamma x), \tilde{f}(\gamma^2 x)) \bmod x^{2\ell} \\
\tilde{C}(x) &= \frac{\partial Q}{\partial z_1}(x, \tilde{f}(x), \tilde{f}(\gamma x), \tilde{f}(\gamma 2x)) \bmod x^{2\ell} \\
\tilde{B}(x) &= \frac{\partial Q}{\partial z_2}(x, \tilde{f}(x), \tilde{f}(\gamma x), \tilde{f}(\gamma 2x)) \bmod x^{2\ell} \\
\tilde{A}(x) &= \frac{\partial Q}{\partial z_3}(x, \tilde{f}(x), \tilde{f}(\gamma x), \tilde{f}(\gamma^2 x)) \bmod x^{2\ell}.
\end{aligned}$$

Let $B(x) = \frac{\tilde{B}(x)}{\tilde{A}(x)}, C(x) = \frac{\tilde{C}(x)}{\tilde{A}(x)}, D(x) = \frac{\tilde{D}(x)}{\tilde{A}(x)}$, then we have

$$h(\gamma^2 x) + B(x)h(\gamma x) + c(x)h(x) = d(x)$$

which is equivalent to

$$x\delta(F) = AF + C$$

of Chapter 5 where

$$F = \begin{bmatrix} h(x) \\ h(\gamma x) \end{bmatrix} \qquad A = \begin{bmatrix} -\frac{1}{\gamma-1} & \frac{1}{\gamma-1} \\ -C(x) & 1 - B(x) \end{bmatrix} \qquad C = \begin{bmatrix} 0 \\ D(x) \end{bmatrix}.$$

In general, for s number of variables, this can be constructed simillarly as follows

$$F = \begin{bmatrix} h(x) \\ h(\gamma x) \\ \vdots \\ h(\gamma^{s-1}x) \end{bmatrix} \qquad A = \begin{bmatrix} -\frac{1}{\gamma-1} & \frac{1}{\gamma-1} & 0 & \dots \\ 0 & -\frac{1}{\gamma-1} & \frac{1}{\gamma-1} & \dots \\ \vdots & & & \\ -A_2(x) & -A_3(x) & \dots & 1 - A_s(x) \end{bmatrix} \in \mathbb{F}[x]^{s-1 \times s-1}$$

and

$$C = \begin{bmatrix} 0 \\ \vdots \\ A_1(x) \end{bmatrix} \in \mathbb{F}[x]^{1 \times s-1}.$$

where

$$\begin{aligned} \tilde{A}_1(x) &= Q(x, \tilde{f}(x), \tilde{f}(\gamma x), \dots, \tilde{f}(\gamma^{s-1}x)) \bmod x^{2\ell} \\ \tilde{A}_2(x) &= \frac{\partial Q}{\partial z_1}(x, \tilde{f}(x), \tilde{f}(\gamma x), \dots, \tilde{f}(\gamma^{s-1}x)) \bmod x^{2\ell} \\ \tilde{A}_3(x) &= \frac{\partial Q}{\partial z_2}(x, \tilde{f}(x), \tilde{f}(\gamma x), \dots, \tilde{f}(\gamma^{s-1}x)) \bmod x^{2\ell} \\ &\vdots \\ \tilde{A}_{s+1}(x) &= \frac{\partial Q}{\partial z_s}(x, \tilde{f}(x), \tilde{f}(\gamma x), \dots, \tilde{f}(\gamma^{s-1}x)) \bmod x^{2\ell}. \end{aligned}$$

and

$$\begin{aligned} A_1(x) &= \frac{\tilde{A}_1(x)}{A_{s+1}(x)} \\ A_2(x) &= \frac{\tilde{A}_2(x)}{A_{s+1}(x)} \\ A_3(x) &= \frac{\tilde{A}_3(x)}{A_{s+1}(x)} \\ &\vdots \\ A_s(x) &= \frac{\tilde{A}_s(x)}{A_{s+1}(x)}. \end{aligned}$$

here γ can be read as q in chapter 5.

7.1 Future work

For future directions, we can think the following.

1. As parallel architectures are becoming available now a days, the interpolation task as well as the root lifting task of folded Reed Solomon code decoding can be implemented in a parallel architecture by following already developed parallel algorithm for the interpolation problem.

2. A complete software implementation based on the algorithms described here.

3. The transformed interpolation problem described in Chapter 3 works for folded Reed Solomon codes. This idea can be extended to the interpolation problem of folded Algebraic Geometric codes.

Curriculum Vitae

Name: **Muhammad Foizul Islam Chowdhury**

Post-Secondary Education and Degrees:

Western University, Canada
London, Ontario, Canada
Ph.D. Computer Science, November 2013

The University of Western Ontario
London, Ontario, Canada
M.Sc. Computer Science, Apr. 2009

Khulna University of Engineering & Technology
Khulna, Bangladesh
B.Sc. in Computer Science and Engineering Sept. 2004

Working Experience:

Research Assistant, Teaching Assistant.
University of western Ontario, London, Canada.
Jan. 2008 - Aug. 2013

Software engineer.
Phoenix Interactive, London, Canada.
April. 2013 - November 2013

Full time faculty member
Dept. of computer science & Engineering.
Leading University, Sylhet, Bangladesh.
Jan. 2005 - Aug. 2007