Introduction to Algebraic Algorithms (FLAG, MU4IN902, EPU-N8-IAL)

Jérémy Berthomieu, Vincent Neiger, Mohab Safey El Din

Introduction

Unless every attending student speaks French, the lectures will be given in English, as well as the tutorials and lab sessions for non-Polytech students. The tutorials and lab sessions for Polytech students will be given in French. You can always ask your questions in French at all times, including during lectures. This document will not be translated into French.

The following is the typical organization of a week, but times and rooms sometimes vary: always check your calendar. The **tutorials** and **lab sessions** take place, depending on your group, on Monday from 8:30 to 12:45 or on Wednesday from 8:30 to 12:45. They will be supervised by **Pierre Pébereau**[1] and **Vincent Neiger**[2]. **Lectures** are taught on Friday from 8:30 to 10:30, by **Vincent Neiger**, in charge of the course.

All communication with the teachers must happen through the forums on the Moodle page on the course. The *only* exception is if you are writing about a matter that requires privacy: then you should contact Vincent Neiger directly by email[a] (or another teacher if necessary).

[a] `firstname.lastname@sorbonne-universite.fr`

The evaluation is done through:

- Moodle quizzes that will take place during the lab sessions (at least 5 quizzes in total), evaluating how you learnt the course and practiced on basic exercises (40% of the final grade). *They must be done from the lab room where the lab session takes place.*
- A final exam evaluating your ability to manipulate the concepts taught in this course, accounting for 60% of the final grade.

[1] `https://polsys.lip6.fr/~pebereau/`
[2] `https://vincent.neiger.science/`

Objectives of the course and outline.

This course presents algorithms for basic computations on algebraic objects such as polynomials and matrices, with a focus on complexity and efficient implementations. The studied computational problems and techniques come from the field of computer algebra (also known as symbolic computation) and find applications for example in cryptography, coding theory, and robotics.

Here is a tentative schedule of the course:

1. Introduction: rings and fields, complexity, matrices, polynomials. Algorithm: division with remainder.
2. Extended Euclidean algorithm. Finite fields.
3. Formal power series and fast division of polynomials.
4. Fast multipoint evaluation and interpolation. Application to Shamir's Secret Sharing
5. Linear recurrent sequences. The Berlekamp-Massey algorithm. Application to sparse linear systems and cyclic codes.
6. Irreducible polynomials, polynomial factorization. Berlekamp's algorithm.
7. Fast algorithms for matrices with structures.
8. Resultant, subresultants. Bivariate polynomial system solving.
9. An opening towards coding theory.

Contents

I. Introduction: polynomials, matrices, and complexity

Cryptography, computer vision, and many other areas require to perform operations on integers, polynomials and matrices of *large* size. Current state-of-the-art software for such purposes often perform operations such as:

- multiply integers with more than 30 million digits;
- multiply univariate polynomials of degree several millions, with coefficients in $\mathbb{Z}/p\mathbb{Z}$ where p is a 60-bit prime number;
- Gaussian elimination of matrices with a few thousand rows and columns, with coefficients in $\mathbb{Z}/p\mathbb{Z}$.

Such operations require fast algorithms and efficient implementations of them.

I.1. Complexity models.

We mainly focus on time complexity; space complexity issues will be mostly ignored in this course.

We will use the Random Access Machine model which is the most standard one. In this model, a program reads and writes integers on different tapes, it is also allowed to use an arbitrary number of integer registers. Elementary operations (which coincide with those supported by an assembly code) are

- write and read (on a tape or a register);
- addition, substraction, multiplication and division;
- three jump instructions: unconditional jump, jump when a register is 0 or jump when a register is not 0.

We will not use subtelties of this model in what follows; the students are encouraged to read a little bit more about this model in the academic litterature.

We will measure two kinds of complexities of algorithms:

Arithmetic complexity. Most of algorithms running on mathematical data are expected to run on an abstract algebraic structure equipped with binary operations.

In this model, we only count the number of such binary operations and predicate tests performed by the algorithm.

This model reflects what is observed in practice when the cost of running binary operations is mostly a constant.

For most of algorithms running over $\mathbb{Q}$, such a model is not so convenient when the size of manipulated numbers grows significantly during the computations.

Note that in this model, we do not take into account copy operations and countings performed for running loops, etc.

Bit complexity. This model is devoted to taking into account the growth of numbers when running algorithms. It is primarily defined for $\mathbb{Z}$: we will see below that every integer can be seen as a *vector* of elements of $\{0, \dots, b-1\}$ for a given integer $b \in \mathbb{N} \setminus \{0, 1\}$. In the bit complexity model, we count the number of operations performed in $\mathbb{Z}/b\mathbb{Z}$.

Again memory access is neglicted in this model (for instance, transposing a matrix, even an extremely large one, is free in this model).

We will use the $O(\cdot)$ notation; in short we say that $f(n) = O(g(n))$ when there exists $K > 0$ and $M > 0$ such that for all $n \geq M$, the following inequality holds:

$$\|f(n)\| \leq K\|g(n)\|.$$

Careful attention must be paid when several parameters are involved in complexity estimates. In those cases, without further precisions, all of them are expected to tend to infinity and the constant K should not depend on any of these parameters.

We may also use the $O\tilde{}(()\cdot)$ notation: we say that $f(n) = O\tilde{}(()g(n))$ when there exists $a \geq 0$ such that

$$f(n) = O\left(g(n) \log_2^a(\max(2, \|g(n)\|)\right).$$

Definition I.1. *Let N be the sum of the sizes of the input and output data. An algorithm is said to be* nearly optimal *(or* quasi-optimal*) when its runtime is bounded by $O\tilde{}(()N)$.*

Problem I.1.

1. *What is the complexity of sorting a list of integers of size n using the bubble sort algorithm? Is this algorithm quasi-optimal?*
2. *Same questions with the quicksort algorithm.*

I.2. Integers.

Here, we present the encoding of integers in computers. Key requirements are uniqueness and generality of the representation.

In the integer encodings presented below, uniqueness is actually a consequence of the uniqueness of the quotient and remainder in the Euclidean division of integers.

Proposition I.2. *Let a and b be in $\mathbb{N}$, with $b \neq 0$. There exists a unique (q, r) in $\mathbb{N} \times \mathbb{N}$ such that $a = bq + r$ and $0 \leq r < b$.*

Proof. Since $b \neq 0$, the sequence $(a - bq)_{q \in \mathbb{N}}$ is decreasing, and takes the value $a \geq 0$ for $q = 0$ and tends to $-\infty$ when $q \longrightarrow +\infty$. Therefore there is q such that $a - bq \geq 0$ and $a - b(q+1) < 0$. This shows the existence: taking this q and $r = a - bq$, by definition of q we have that $r \geq 0$ and $r = a - bq = a - b(q+1) + b < b$.

For the uniqueness, we consider two pairs (q, r) and (q', r') such that $0 \leq r < b$ and $0 \leq r' < b$ and $a = bq + r = bq' + r'$. From this identity we obtain

$$b(q - q') = r' - r. \tag{I.1}$$

On the other hand, from the inequalities on r and r' we obtain

$$-b < r' - r < b. \tag{I.2}$$

Combining (I.1) and (I.2) we obtain $-b < b(q - q') < b$, which implies $-1 < q - q' < 1$ since $b \neq 0$. Since $q - q'$ is an integer, this means $q - q' = 0$. It follows that $q = q'$ and $r = r'$. □

Theorem I.3. *Let $n \in \mathbb{N}$ and $b \in \mathbb{N} \setminus \{0, 1\}$. There exists a unique sequence $a_0, \ldots, a_h$ in $\{0, \ldots, b-1\}$ such that $a_h \neq 0$ and*

$$a_0 + a_1 b^1 + \cdots + a_h b^h = n.$$

This sequence is called the decomposition of n in base b, *and h is called the* height of n in base b.

Proof. Our proof is by induction on n. The cases $n = 0$ and $n = 1$ are obvious. Our induction assumption is that for any $m < n$, the decomposition of m in base b exists and is unique.

Define a_0 as the remainder of the Euclidean division of n by b, and let q be the corresponding quotient. By Proposition I.2, the pair (q, a_0) is unique.

Observe that $q < n$, since $b \geq 2$. Hence we can apply our induction assumption on q. This shows that it can be written uniquely as $q = a_1 + \cdots + a_h b^{h-1}$ with $a_h \neq 0$ and all a_i's in $\{0, \ldots, b-1\}$. This concludes the proof. □

Lemma I.4. *Let $n \in \mathbb{N} \setminus \{0\}$ and $b \in \mathbb{N}$. Then the height of n in basis b is bounded by $\lceil \log_b(n) \rceil$.*

Proof. Using notation from the previous theorem, we obviously have $b^h \leq n$. Exercise: prove that $n < b^{h+1}$, and conclude. □

Problem I.2. *How many bits do you need to store integers of the form a^t for $a \in \mathbb{N}$ and $t \in \mathbb{N}$?*

What is the bit complexity of the basic algorithm computing the n-th term of the Fibonacci sequence? Recall that it is defined by $F_{k+2} = F_{k+1} + F_k$ for all $k \in \mathbb{N}$ with $F_0 = 0$ and $F_1 = 1$.

Bit complexity counts the number of single / bit operations and hence allows us to take into account the growth of coefficients.

Theorem I.5. *Let a and b be integers of respective bit size h_1 and h_2; we also let $h = \max(h_1, h_2)$. There exist algorithms for*

- *adding a and b in bit complexity $O(h)$;*
- *multiplying a and b in bit complexity $O(h_1 h_2)$ (naive algorithm);*
- *multiplying a and b in bit complexity $O(h^{1.59})$ (Karatsuba's algorithm);*
- *multiplying a and b in bit complexity $O(h \log(h) \log(\log(h)))$ (Fast Fourier Transform, to be seen later).*

In the sequel, we assume that an integer multiplication is fixed (for example, one in the above list), and the notation $\mathsf{M}_{\mathbb{Z}}(h)$ stands for the bit complexity of multiplying two integers of bit size $\leq h$ using that algorithm (for example, one of the complexities in the above list).

Problem I.3. *Consider an integer n, and computations in $\mathbb{Z}/n\mathbb{Z}$.*

- *What is the (binary) cost of performing addition and multiplication in $\mathbb{Z}/n\mathbb{Z}$?*
- *Would you implement arithmetic in $\mathbb{Z}/n\mathbb{Z}$ the same way for $n \leq 2^{32}$ and for $n > 2^{32}$?*

I.3. Polynomials.

Let R be a ring. Univariate polynomials in $R[x]$ such as $c_0 + c_1 x + \cdots + c_d x^d$ can be represented by vectors of coefficients $[c_0, \ldots, c_d]$. Hence there is a strong similarity with integers (the variable x in some sense plays the same role as the base b chosen to represent integers).

Remark: this encoding is often, but not always, the most appropriate. For example, one noticeable family is that of *sparse* polynomials: these are polynomials whose number of nonzero coefficients is *negligible* compared to their degree. For such polynomials, the proposed encoding as vectors of coefficients may be quite inefficient: the polynomial $x^{425147} + 7x^{26245} - 11x^{24} + 3$ is stored as a vector of 425148 coefficients, which are all zero except four of them.

Theorem I.6. *Let f and g be two polynomials in $R[x]$ of degree at most n. Then the arithmetic complexity of multiplying f and g is*

- *$O(n^2)$ operations in R using the naive algorithm;*
- *$O(n^{1.59})$ operations in R using Karatsuba's algorithm;*
- *$O(n \log(n) \log(\log(n)))$ operations in R (Fast Fourier Transform, to be seen later).*

The proof of the first statement is obvious.

We take now advantage of the mention of Karatsuba's algorithm to study the important "master theorem" for the complexity of divide-and-conquer algorithms.

First let us recall that Karatsuba's algorithm relies on the very basic following observation: writing

$$f = f_0 + f_1 x \qquad \text{and} \qquad g = g_0 + g_1 x$$

their product $h = fg = h_0 + h_1 x + h_2 x^2$ can be obtained through an interpolation formula at 0, 1 and ∞:

$$h_0 = f_0 g_0, \quad h_2 = f_1 g_1 \quad h_1 = f(1)g(1) - h_0 - h_2.$$

Hence, to obtain the coefficients of h, only 3 multiplications (and 4 additions) are needed.

The idea of reducing the number of multiplications to speed up fundamental algorithms will be used repeatedly in this course and related ones.

One derives a multiplication algorithm by truncating polynomials at half degree. This is a famous example of a divide-and-conquer strategy which is at the foundation of many asymptotically fast algorithms.

On input data of size n, this principle consists in reducing to m recursive calls with input data divided by p (often $p = 2$) and finally recombine the result. The cost of the splitting and recombination steps is bounded by a function T. When the input data has size smaller than s, another algorithm running in time κ is called.

We describe below how the runtime of a divide-and-conquer algorithm can be estimated. For this asymptotic analysis, one can reasonably assume $s \geq p$. The total cost can be written as

$$C(n) = \begin{cases} T(n) + mC\left(\lceil n/p \rceil\right) & \text{when } n \geq s \geq p \\ \kappa & \text{otherwise.} \end{cases} \tag{I.3}$$

We state now the divide-and-conquer lemma, leading to the master theorem.

Lemma I.7 (Divide-and-conquer lemma)**.** *Let C be a function as in (I.3) with $m > 0, \kappa > 0$ and T such that $T(pn) \geq qT(n)$ with $q > 1$. Then, if n is a positive power of p, one has*

$$C(n) \leq \begin{cases} \left(1 - \frac{m}{q}\right)^{-1} T(n) + \kappa n^{\log_p(m)} & \text{if } q > m \\ T(n) \log_p n + \kappa n^{\log_p q} & \text{if } q = m \\ n^{\log_p m} \left(\frac{T(n)}{n^{\log_p q}} \frac{q}{m-q} + \kappa \right) & \text{if } q < m. \end{cases}$$

Proof. The repeated use of (I.3) yields

$$\begin{aligned} C(n) \leq & \qquad T(n) + mC\left(\frac{n}{p}\right) \\ \leq & \qquad T(n) + mT\left(\frac{n}{p}\right) + \cdots + m^{k-1} T\left(\frac{n}{p^{k-1}}\right) + m^k C\left(\frac{n}{p^k}\right) \\ \leq & \qquad T(n)\left(1 + \frac{m}{q} + \cdots + \left(\frac{m}{q}\right)^{k-1}\right) + m^k \kappa \end{aligned}$$

for $k = \left\lfloor \log_p(n/s) \right\rfloor + 1$. This choice implies

$$m^k \kappa \leq n^{\log_p m} \kappa$$

which bounds the second term of the above inequality. It remains to bound the first term.

When $q > m$, the sum in brackets is bounded by the sum of a geometric series.

When $q = m$, this is a sum with all k terms equal to 1.

When $q < m$, it can be rewritten as

$$\left(\frac{m}{q}\right)^{k-1} \left(1 + \frac{q}{m} + \cdots + \left(\frac{q}{m}\right)^{k-1}\right)$$

which we also bound using a geometric series. □

From this lemma, one easily deduces the following theorem.

Theorem I.8 (Master theorem). *Let C be a function as in (I.3) with $m > 0, \kappa > 0$ and T be an* increasing *function such that there exist q and r with $1 < q \leq r$ sastisfying*

$$qT(n) \leq T(pn) \leq rT(n), \qquad \text{for all } n \text{ large enough.}$$

Then

$$C(n) = \begin{cases} O(T(n)) & \text{if } q > m, \\ O(T(n)\log(n)) & \text{if } q = m, \\ O(n^{\log_p m/q} T(n)) & \text{if } q < m. \end{cases}$$

The master formula for Karatsuba's algorithm is

$$K(n) \leq 3K(n/2) + 4n.$$

In the rest of the course, we will assume that polynomials are encoded with vectors of coefficients.

Problem I.4. *In all this exercise, n is the size of the polynomials (they are thus of degree $n-1$).*

1. *Give a complete description in pseudo-code of Karatsuba's algorithm.*
2. *In a language where the memory is handled by the user, like in C, how to implement this algorithm?*

   Be careful concerning the fact that polynomials are encoded as array of coefficients.
3. *If n is a power of 2, give a* bound *on the number of operations performed by Karatsuba's algorithm.*
4. *Let n be a power 2. Give a hybrid multiplication algorithm for two polynomials of degree n in $R[x]$ (where R is a ring) which calls Karatsuba's algorithm if $n > 2^d$ and the naive one otherwise.*

5. *Show that the arithmetic complexity $C(n)$ of this algorithm satisfies $C(n) \leq \gamma(d)n^{\log_2 3} - 8n$ for all $n \geq 2^d$, where γ is a function that only depends on d.*

6. *Find the value d that minimizes γ. What can you deduce from it?*

Problem I.5. *Can we deduce from the above theorem the bit complexity of multiplying polynomials in $\mathbb{Z}[x]$?*

I.4. Matrices.

Matrices will be mostly considered with entries in a field $\mathbb{K}$ (either a finite one like $\mathbb{Z}/p\mathbb{Z}$ with p prime or a field of characteristic 0). Dense matrices are encoded with a two-dimensional array; sparse and structured matrices are encoded with ad-hoc representations minimizing space complexity.

Adding matrices is done naively in linear time:

Lemma I.9. *Let M and N be $p \times q$ matrices with entries in $\mathbb{K}$. Computing $M + N$ is done using $O(pq)$ operations in $\mathbb{K}$.*

Problem I.6. *Prove this lemma.*

However, approaching linear time for matrix multiplication remains an open and very challenging problem, despite decades of research on this topic. Using the naive algorithm with three for loops, we get the following result:

Lemma I.10. *Let M and N be $n \times n$ matrices with entries in $\mathbb{K}$. Computing the product $M \cdot N$ with the naive multiplication algorithm uses $O(n^3)$ operations in $\mathbb{K}$.*

Problem I.7. *Prove this lemma. What is the complexity of the naive multiplication algorithm if M is $n \times q$ and N is $q \times r$?*

Remark I.11. *The size of the input M and N is $O(n^2)$; the size of the output is $O(n^2)$. We deduce that the cost of multiplying M by N is* not *asymptotically optimal. Reducing the exponent 3 as much as possible towards 2 is a major research open question. Corollary: never write and/or say that matrix multiplication is quadratic or quasi-optimal (unless it is proved in the future, which is quite unsure).*

Hence, the quest for nearly optimal algorithms for matrix multiplication is not finished. We shall see in the following problem a first algorithm that can multiply two (2×2)-matrices in fewer than 8 multiplications.

Problem I.8. *Winograd and Waksman's algorithms.*

1. *How many (exactly) additions and multiplications are needed by the naive algorithm for matrices of size n?*

2. *Let $n = 2k$, $v = (v_1, \dots, v_n)$ et $w = (w_1, \dots, w_n)$ two vectors. Let $\sigma(v) = v_1v_2 + \dots + v_{2k-1}v_{2k}$ and $\sigma(w) = w_1w_2 + \dots + w_{2k-1}w_{2k}$.*

   Show that the scalar product of v by w is $(v_1 + w_2)(v_2 + w_1) + \dots + (v_{2k-1} + w_{2k})(v_{2k} + w_{2k-1}) - \sigma(v) - \sigma(w)$.

3. *How many additions and multiplications are done with this formula?*

4. *What can you deduce for the matrix product?*

5. *We now assume that 2 is invertible in the base field (what does it mean?) and that the division by 2 is free. Let $R = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)\left(\begin{smallmatrix} x & y \\ z & t \end{smallmatrix}\right)$. How many additions and multiplications are done with the following formula: $R = \frac{1}{2}\left(\begin{smallmatrix} \alpha_1-\alpha_2 & \beta_1-\beta_2 \\ \gamma_1-\gamma_2 & \delta_1-\delta_2 \end{smallmatrix}\right)$? where $\alpha_1 = (a+z)(b+x)$, $\alpha_2 = (a-z)(b-x)$, $\beta_1 = (a+t)(b+y)$, $\beta_2 = (a-t)(b-y)$, $\gamma_1 = (c+z)(d+x)$, $\gamma_2 = (c-z)(d-x)$, $\delta_1 = (c+t)(d+y)$ and $\delta_2 = (c-t)(d-y)$.*

6. *Using $\gamma_1 + \gamma_2 + \beta_1 + \beta_2 = \alpha_1 + \alpha_2 + \delta_1 + \delta_2 = 2(ab + cd + xz + ty)$, what can you deduce on the number of additions and multiplications needed to compute the product? What do you conclude?*

7. *Can we use this algorithm recursively, like Karatsuba's?*

The first algorithm with a time complexity better than $O(n^3)$ is due to Strassen. Like for Karatsuba's algorithm, it is based on an improvement in the case $n = 2$ (for Karatsuba, we were considering linear polynomials to build the general improvement) and a decrease of the number of multiplications.

So, let

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad \text{and} \qquad M = \begin{pmatrix} x & y \\ z & t \end{pmatrix}.$$

To multiply M by N, compute the following quantities

$$\begin{aligned} q_1 &= a(x+z) & q_2 &= d(y+t) & & \\ q_3 &= (d-a)(z-y) & q_4 &= (b-d)(z+t) & & \\ q_5 &= (b-a)z & q_6 &= (c-a)(x+y) & q_7 &= (c-d)y, \end{aligned}$$

compute the sums

$$\begin{aligned} r_{1,1} &= q_1 + q_5 & r_{1,2} &= q_2 + q_3 + q_4 - q_5 \\ r_{2,1} &= q_1 + q_3 + q_6 - q_7 & r_{2,2} &= q_2 + q_7. \end{aligned}$$

The entry at row i and column j of the product MN is exactly $r_{i,j}$, for $1 \le i, j \le 2$.

Hence, this remark allows us to compute matrix multiplication by splitting $2n \times 2n$ matrices into blocks of size n. It performs 7 recursive calls to the matrix multiplication algorithm and $C = 18$ additions.

The recurrence formula for estimating the cost of multiplying square matrices is

$$S(n) \leq 7S\left(\frac{n}{2}\right) + C\left(\frac{n}{2}\right)^2.$$

Using the above lemma for estimating the cost of divide and conquer strategies with $m = 7$, $p = s = 2$, $\kappa = 1$ and $q = 4$, we deduce

$$S(n) \leq \left(1 + \frac{C}{3}\right) n^{\log_2 7}.$$

This leads to the following theorem (using $\log_2 7 \leq 2.81$).

Theorem I.12. *One can multiply $n \times n$ matrices with entries in a ring R using $O(n^{2.81})$ operations in R.*

Problem I.9. *Check Strassen's formulae.*

Strassen's algorithm becomes competitive for matrices of size around 64 with entries which are integers of small size.

Problem I.10.

1. *Practice multiplying polynomials of size 4 and 6 using Karatsuba's algorithm.*
2. *Practice multiplying matrices of size 2 using Strassen's algorithm.*

Implementation I.1. *Using SageMath, implement the naive multiplication algorithm and Karatsuba's multiplication algorithm, for arbitrary univariate polynomials over a field.*

Test your code by comparing the result to that of SageMath's native polynomial multiplication, on many examples. Note: if `ring` *is your polynomial ring, you can generate a random polynomial* `p` *of degree* `d` *with the command* `p = ring.random_element(degree=d)`.

Measure the performance of your code over a small finite field such as $\mathbb{Z}/97\mathbb{Z}$. You can fill a table showing timings of your two algorithms and the SageMath native multiplication, for various degrees (e.g. powers of 2 from 2 to 2^{14}).

Use these measures to estimate the exponent in the complexity, for each of the three benchmarked algorithms.

Implementation I.2. *Implement basic operations over $\mathbb{Z}/p\mathbb{Z}$ in language C, assuming p is a prime number less than 2^{30}. You should provide at least addition $x+y$, multiplication $x \cdot y$, inversion x^{-1} for $x \neq 0$, and opposite $-x$. For convenience, you can also provide subtraction $x-y$ and division x/y for $y \neq 0$.*

Test your code on many examples, verifying the results using SageMath or some basic calculator.

Implementation I.3. *In language C, implement memory management, addition, and naive multiplication of univariate polynomials for polynomials over $\mathbb{Z}/p\mathbb{Z}$ where p is a prime number less than 2^{30}.*

Then implement Karatsuba's multiplication, and find experimentally the best size of polynomials to stop the recursive calls and instead rely on the already implemented naive algorithm.

Bonus: *improve your Karatsuba's algorithm implementation so that two of the three recursive calls can store their partial results in the allocated memory for the global result.*

II. Euclid's division and GCD algorithms

In this chapter, we mainly present Euclid's algorithms for division with remainder and for computing greatest common divisors. We also describe its immediate applications, essentially to what is called *rational reconstruction.*

II.1. A little bit more on algebraic structures.

Here we introduce algebraic closures and quotient rings.

Lemma II.1. *Let $f \in \mathbb{K}[x]$ and $a \in \mathbb{K}$ such that $f(a) = 0$. Then, $x - a$ divides f.*

Lemma II.2. *Let $\mathbb{K}$ be a field and $f \in \mathbb{K}[x]$ of degree d. Then f has at most d roots in $\mathbb{K}$.*

Problem II.1. *Prove these two lemmas.*

Definition II.3. *Let $\mathbb{K}$ be a field. We say that a field $\bar{\mathbb{K}}$ is an algebraic closure of $\mathbb{K}$ if $\mathbb{K} \subseteq \bar{\mathbb{K}}$ and, for all $f \in \mathbb{K}[x]$ of degree d, f has d roots in $\bar{\mathbb{K}}$ (counted with multiplicity).*

For example, the fields $\mathbb{Q}$ and $\mathbb{R}$ are not algebraically closed: you already know a classical example of a polynomial of degree 2 which has 0 roots (what is this classical example?). On the other hand, $\mathbb{C}$ is an algebraic closure of $\mathbb{Q}$ and of $\mathbb{R}$.

Definition II.4. *Let $\mathcal{R}$ be a ring and $f \in \mathcal{R}[x]$ be a monic polynomial (i.e. its leading coefficient is 1). Let also s and t be polynomials in $\mathcal{R}[x]$.*

- *We say that s and t are equivalent modulo f if f divides $s - t$.*
- *The set of classes of equivalences is denoted by $\mathcal{R}[x]/\langle f \rangle$ and is called a* quotient ring.

II.2. Division algorithm for polynomials in $\mathbb{K}[x]$.

Let $\mathbb{K}$ be a field. For a polynomial $A = a_p x^p + \cdots + a_0 \in \mathbb{K}[x]$ of degree p, i.e. with $a_p \neq 0$, we let $\mathrm{lc}(A) = a_p$, where lc stands for *leading coefficient.*

Recall that $\mathbb{K}[x]$ is a Euclidean domain; actually we take for the map h used to define the division the degree function (the one which associates to a polynomial in $\mathbb{K}[x]$ its degree).

Using it, one can define the Euclidean division in $\mathbb{K}[x]$ of two polynomials. The traditional way to define it is as the output of the division algorithm below, Algorithm 1.

```
Algorithm 1: PolynomialDivisionAlgorithm
Input: Two polynomials $A = a_m x^m + \cdots + a_0$ and $B = b_n x^n + \cdot + b_0$ in $\mathbb{K}[x]$.
Output: Two polynomials $Q = q_{m-n}x^{m-n} + \cdots + q_0$ and $R = r_p x^p + \cdots + r_0$ in $\mathbb{K}[x]$ such
        that $A = BQ + R$ and $p = \deg R < \deg B = n$.
$R := A$, $Q = 0$, $b = \mathrm{lc}(B)$
While $\deg R \geq \deg B$ do
    $a := \mathrm{lc}(R)$
    $Q := Q + \frac{a}{b} x^{\deg R - \deg B}$
    $R := R - \frac{a}{b} x^{\deg R - \deg B} B$
Return $(Q, R)$
```

It is a *rewriting* algorithm: it consists in applying repeatedly the rewriting rule

$$x^q \longrightarrow -\left(\frac{b_{q-1}x^{q-1} + \cdots + b_0}{b_q}\right).$$

When $q = 1$ this matches with the substitution operation. Hence, dividing A by $B = b_1 x + b_0$ (with $b_1 \neq 0$ since $q = 1$) consists in evaluating A at the root $\frac{-b_0}{b_1}$ of B.

More generally for $q \geq 1$, when ρ is a root of B (in $\mathbb{K}$ or in an algebraic closure of $\mathbb{K}$), A and R take the same value at ρ, meaning $A(\rho) = R(\rho)$, where R is the remainder of the Euclidean division of A by B.

Problem II.2. *Prove the assertion in the last paragraph.*

Remark II.5. *If we assume that B is* monic *(i.e. its leading coefficient is $b = 1$), then the Euclidean division with remainder does not require any inversion in the base field $\mathbb{K}$ where the coefficients of the polynomials lie. Thus, in this monic case, one can in fact extend* PolynomialDivisionAlgorithm *to input A and B in $\mathcal{R}[x]$ for any commutative ring $\mathcal{R}$, as long as $\mathrm{lc}(B) = 1$.*

Problem II.3. *Practice dividing polynomials with* PolynomialDivisionAlgorithm, *both in the case of coefficients in $\mathbb{Q}$ and in the case of coefficients in some prime field $\mathbb{Z}/p\mathbb{Z}$ (e.g. $p = 7$).*

Theorem II.6. *On input A and B in $\mathbb{K}[x]$ of respective degrees m and n with $m \geq n$, the above division algorithm,* PolynomialDivisionAlgorithm *performs $O(n(m-n))$ arithmetic operations in $\mathbb{K}$.*

Proof. The while loop performs $O(n)$ arithmetic operations. Observe also that it is performed $m - n + 1$ times in the worst case. This ends the proof of the complexity statement. □

One may remark that the complexity of PolynomialDivisionAlgorithm is the same as that of naive multiplication of B and Q. Yet, because the algorithm only multiply B with some monomials several times, as such it cannot benefit from faster multiplication algorithms like Karatsuba's or *FFT*-based ones.

Observe that when $n \sim \frac{m}{2}$ the algorithm has a complexity which is quadratic in m, while the output is an array of at most $n+1$ coefficients. Hence, a natural question is whether it is feasible to compute the division with remainder with a better complexity, closer to linear in n; and if yes, a natural goal is to design an explicit algorithm doing this. We will answer this question positively and reach this goal in a subsequent chapter of this course, with an algorithm of quasi-optimal complexity (see Theorem IV.13 in Chapter IV):

Theorem II.7. *Let $A = a_m x^m + \cdots + a_0$ and $B = b_n x^n + \cdots + b_0$ be two polynomials in $\mathbb{K}[x]$ of degrees $m \geq n$. Let $m \mapsto \mathsf{M}(m)$ be such that there exists an algorithm for multiplying two polynomials of degree m in operations. Then, one can compute the Euclidean division of A and B in $O(\mathsf{M}(m))$ operations.*

Remark II.8. *This bound is not tight, it can be sharpened depending on n and $m-n$, as we will see in Theorem IV.13.*

Problem II.4. *Provide families of polynomial where the above complexity estimate is better than the one from the naive* PolynomialDivisionAlgorithm *(Algorithm 1). Same question but for a worse complexity estimate.*

One can now investigate some applications and consequences.

Corollary II.9. *Let $\mathcal{R}$ be a ring and f be a monic polynomial in $\mathcal{R}[x]$ of degree d. The quotient $\mathcal{R}[x]/\langle f\rangle$ equipped with the addition and multiplication induced by the addition in $\mathcal{R}[x]$ and the combination of multiplication and Euclidean division by f in $\mathcal{R}[x]$ is a ring in which one can perform addition and multiplication in time $O(\mathsf{M}(d))$.*

If $\mathcal{R} = \mathbb{K}$, a field, then this quotient is also a $\mathbb{K}$-vector space of dimension d.

Problem II.5. *Give pseudo-code for the addition and multiplication algorithms in $\mathcal{R}[x]/\langle f\rangle$, assuming polynomials are stored as arrays of coefficients.*

II.3. Euclid's algorithm (and the extended Euclidean algorithm).

From now on, we assume that $\mathcal{R}$ is an integral domain.

Definition II.10. *Let a and b in $\mathcal{R}$. One says that g is a greatest common divisor (gcd) of a and b in $\mathcal{R}$ if*

- *$g \in \mathcal{R}$;*
- *g divides a;*
- *g divides b;*
- *any common divisor of a and b divides g.*

The above definition is rather general. In particular, there exist integral rings containing couples of elements without any gcd.

Example II.11. *Let $\mathcal{R}$ be the ring $\mathbb{Z}[i\sqrt{3}] = \{x + i\sqrt{3}y, x, y \in \mathbb{Z}\}$, $a = 4$ and $b = 2 + 2i\sqrt{3}$. One can establish easily that $g_1 = 2$ divide both a and b; $g_2 = 1 + i\sqrt{3}$ divide both a and b but no multiple of g_1 or g_2 divides both a and b. Thus, there is no gcd to a and b.*

Integral rings admitting gcds for all their couples of elements are called rings with gcd. In general, one prefers to deal with *factorial rings* (readers may learn about these rings in any textbook or other reference). For computational purpose, the subclass of rings with gcds are simply *Euclidean domains.* Recall that a height function h is associated to any element in Euclidean rings. Recall also that for $0 \in \mathcal{R}$, $h(0)$ is less than the height of any other element of $\mathcal{R}$.

The core idea behind Euclid's algorithm is that on input a and b in a Euclidean ring $\mathcal{R}$ equipped with a height function h and for (q, r) such that

$$a = bq + r \qquad \text{with} \qquad h(r) < h(b)$$

one has $\gcd(a, b) = \gcd(b, r)$. Hence one can call recursively the algorithm on input (b, r), yielding a decreasing sequence of height values.

```
Algorithm 2: EuclidAlgorithm
Input: Two elements $a$ and $b$ in a Euclidean domain $\mathcal{R}$ with a height function $h$.
Output: A gcd of $a$ and $b$ in $\mathcal{R}$.
$r_0 := a$, $r_1 := b$, $i := 1$
While $r_i \neq 0$ do
    $r_{i+1} := \mathsf{rem}(r_{i-1}, r_i)$
    $i := i + 1$
Return $r_{i-1}$
```

The sequence $(R_0, \ldots, R_\delta)$ is called the sequence of Euclidean remainders (or Euclidean remainder sequence).

Theorem II.12. *Let $\mathcal{R} = \mathbb{K}[x]$ where $\mathbb{K}$ is a field. Algortihm 2,* EuclidAlgorithm*, on input two polynomials A and B in $\mathbb{K}[x]$ returns a gcd of A and B of degrees $m \geq n$ and performs $O(mn)$ operations in $\mathbb{K}$.*

Proof. Correctness is immediate through a reasoning by induction as the one done before the description of the algorithm. We focus on the complexity statement, assuming that $\deg A \geq \deg B$.

Using the complexity of the naive algorithm performing the Euclidean division, dividing a polynomial S by a polynomial T is done using $O(\deg T(\deg S - \deg T))$. Hence there exists a constant K such that this is done using *at most* $K \deg T(\deg S - \deg T)$.

One deduces that in order to estimate the cost of Euclid's algorithm in $\mathbb{K}[x]$, one needs to upper bound the sum of all

$$K \deg R_i(\deg R_{i-1} - \deg R_i).$$

But all polynomials R_i, for $i \geq 1$, have degree bounded by $\deg(B)$. Hence one needs to bound now

$$K \deg(B) \sum_{i=1}^{\delta} (\deg R_{i-1} - \deg R_i) \leq K \deg B \deg A.$$

This finishes the proof. □

The above complexity bound does reflect the practical behaviour of the algorithm when the degrees of the Euclidean remainder sequence decrease one by one.

Let us go back to the case of a Euclidean ring $\mathcal{R}$ and let a and b in $\mathcal{R}$ with $g = \gcd(a, b)$. Assume that $g = 1$. One then expects to find u and v in $\mathcal{R}$ such that

$$g = ua + vb.$$

The reason of this is rather intuitive and goes back to the case of integers. When a and b are *coprime* integers (e.g. their gcd is 1), one expects to be able to *invert a* modulo b, hence establishing that $\mathbb{Z}/p\mathbb{Z}$ is a field if p is a prime integer.

When a and b are elements of a Euclidean ring $\mathcal{R}$ and $g = \gcd(a, b)$, one expects to find u and v with additional height properties

$$g = ua + vb \qquad \text{and } h(ug) < h(b) \text{ and } h(vg) < h(a).$$

This has *many* applications, in particular in computer algebra, cryptography, computational geometry, etc.

The elements u and v as above are called cofactors; again this terminology comes from the analogy with integers.

The rather intuitive idea for computing the cofactors is as follows. At each step i of Euclid's algorithm, one finds u_i and v_i such that

$$r_i = u_i a + v_i b.$$

Note that for $i = 0$, it suffices to take $u_0 = 1$ and $v_0 = 0$ which trivially yields $1 \cdot a + 0 \cdot b = a = r_0$. For $i = 1$, one also easily obtains $u_1 = 0$ and $v_1 = 1$ which yields $0 \cdot a + 1 \cdot b = b = r_1$. Next, Euclidean division enters in the game with the relation

$$r_{i-1} = q_i r_i + r_{i+1}.$$

Using the recurrence relation, one obtains

$$r_{i+1} = r_{i-1} - q_i r_i = (u_{i-1} - q_i u_i)a + (v_{i-1} - q_i v_i)b.$$

which shows that one can define

$$u_{i+1} = u_{i-1} - q_i u_i \qquad \text{and} \qquad v_{i+1} = v_{i-1} - q_i v_i$$

Problem II.6. *Prove that computing the cofactors as above yields the height requirements on them.*

```
Algorithm 3: ExtendedEuclideanAlgorithm
Input: Two elements $a$ and $b$ in a Euclidean domain $\mathcal{R}$ with a height function $h$.
Output: A gcd of $a$ and $b$ in $\mathcal{R}$ together with the corresponding cofactors.
$r_0 := a, u_0 := 1, v_0 := 0.$
$r_1 := b, u_1 := 0, v_1 := 1, i := 1$
While $r_i \neq 0$ do
    $(q_i, r_{i+1}) :=$ PolynomialDivisionAlgorithm$(r_{i-1}, r_i)$
    $u_{i+1} = u_{i-1} - q_i u_i, v_{i+1} = v_{i-1} - q_i v_i$
    $i := i + 1$
Return $r_{i-1}, u_{i-1}, v_{i-1}$
```

Theorem II.13. *When $\mathcal{R} = \mathbb{K}[x]$, Algorithm 3,* ExtendedEuclideanAlgorithm*, runs on input A and B using $O(\deg(A)\deg(B))$ arithmetic operations in $\mathbb{K}$.*

Problem II.7. *Prove the theorem above.*

II.4. Applications.

The first application of Euclid's algorithm is the computation of the gcd of two elements in a Euclidean domain.

Theorem II.14. *Let a and b two elements of a Euclidean ring $\mathcal{R}$ and let g be the output of* EuclidAlgorithm *on input (a, b). Then g is a gcd of a and b.*

Another *very important* application of the extended Euclidean algorithm is the so-called modular inversion. For a Euclidean ring $\mathcal{R}$, we say that (p, q) in $\mathcal{R} \times \mathcal{R}$ are coprime when their gcd is 1. Note that this gcd is only defined up to a unit, that is up to multiplication by an invertible element in $\mathcal{R}$.

Problem II.8. *For two polynomials a and b in $\mathbb{K}[x]$, show that the output of* EuclidAlgorithm(a, b) *is in $\mathbb{K} \setminus \{0\}$ if and only if a and b are coprime.*

In this case, by Bézout's relation, there exists a pair (u, v) in $\mathcal{R} \times \mathcal{R}$ such that

$$ua + vb = 1,$$

and this pair is easily found from the output of ExtendedEuclideanAlgorithm. It follows that u is the inverse of a modulo b.

Problem II.9. *Working in $\mathbb{Z}/2\mathbb{Z}[x]$, compute the inverse of the polynomial $a = x^4 + x + 1$ modulo the polynomial $x^5 + x^2 + 1$.*

We use this modular inversion to *solve* modular equations. Let $\mathcal{R}$ be a Euclidean ring, p and q be two coprime elements of $\mathcal{R}$ and a, b be elements of $\mathcal{R}$. We aim at solving the *modular equations*

$$x = a \bmod p \qquad \text{and} \qquad x = b \bmod q, \tag{II.1}$$

where x is the unknown (the mod notation means that we take the remainder of the Euclidean division).

Recall that since p and q are coprime, their gcd is 1 and, consequently, there exists a couple (u, v) in $\mathcal{R} \times \mathcal{R}$ such that

$$up + vq = 1.$$

In other words, we have

$$u = p^{-1} \bmod q \qquad \text{and } v = q^{-1} \bmod p.$$

All in all, we easily guess that one solution of the modular equations (II.1) is

$$x = avq + bup \bmod pq.$$

Problem II.10. *Prove that this solution is unique modulo pq.*

Observe that one can apply this easily to a variety of problems such as Chinese Remainder Theorem, polynomial interpolation (i.e. Lagrange interpolation) and manipulation of algebraic parametrizations.

Problem II.11. *Solve the following two systems of modular equations*

$$\begin{aligned} x = 3 \bmod 21 \qquad &\textit{and} \qquad x = 10 \bmod 80, \\ x = 3 \bmod (x-1) \qquad &\textit{and} \qquad x = 10 \bmod (x+1). \end{aligned}$$

Implementation II.1. *Implement the* ExtendedEuclideanAlgorithm *for integers. Use it to implement an inversion function in $\mathbb{Z}/n\mathbb{Z}$.*

Implementation II.2. *Implement the* PolynomialDivisionAlgorithm *for polynomials over $\mathbb{Z}/p\mathbb{Z}$. Use it to implement the* ExtendedEuclideanAlgorithm *for polynomials over $\mathbb{Z}/p\mathbb{Z}$.*

III. Finite fields

In this chapter, we study finite fields, a basic algebraic structure commonly encountered in areas such as cryptography, numerical computing, quantum information, computer vision, etc. These are sets which are equipped with two arithmetic operators (addition and multiplication), acting on elements of these sets, with the properties of $+$ and $\times$ we are all used to.

III.1. Finite fields

III.1.1. Quotient rings

Definition III.1. *Let R be a ring and I be an ideal of R. The quotient ring R/I is the* set of equivalence classes *$a + I = \{a + i \mid i \in I\}$ for all $a \in R$.*

- *The sets $a + I$ and $b + I$ are equal if, and only if, $b - a \in I$.*
- *The quotient ring has a natural ring structure with*
  - *$0 + I = I$ is the identity element for $+$;*
  - *$1 + I$ is the identity element for $\times$*
  - *$(a + I) + (b + I) = (a + b) + I$;*
  - *$(a + I) \times (b + I) = ab + I$.*

We may write $\bar{a}$ or even a for the class of a in A/I.

Example III.2.
- *$\mathbb{Z}/13\mathbb{Z} = \{0, 1, \ldots, 12\}$ with operations modulo 13, e.g. $3 - 5 = -2 = 11$.*
- *$\mathbb{R}[x]/\langle x^2 + 1\rangle = \{a + bx \mid a, b \in \mathbb{R}\}$ with operations modulo $x^2 + 1$, e.g. $(1 + 2x)(3 - 4x) = 3 + 2x - 8x^2$ but $x^2 = -1$, hence $(1 + 2x)(3 - 4x) = 11 + 2x$. This quotient is a field; in fact, it is isomorphic to the field $\mathbb{C}$ of complex numbers.*
- *$\mathbb{R}[x]/\langle x^2\rangle = \{a + bx \mid a, b \in \mathbb{R}\}$ with operations modulo x^2, e.g. $(1 + 2x)(3 - 4x) = 3 + 2x - 8x^2$ but $x^2 = 0$, hence $(1 + 2x)(3 - 4x) = 3 + 2x$. It is* not *a field! Can you give an explicit element of this quotient which is not invertible?*

Problem III.1. *Do the following computations by hand:*

1. *addition $4 + 5$ and multiplication $4 \cdot 5$ in $\mathbb{Z}/7\mathbb{Z}$;*
2. *inversion of 23 in $\mathbb{Z}/31\mathbb{Z}$;*

3. *addition* $(2+2x)+(1+2x)$ *and multiplication* $(2+2x)\times(1+2x)$ *in* $\mathbb{Z}/3\mathbb{Z}[x]/\langle x^2+1\rangle$;

4. *Compute the inverse of* x^4+x+1 *in* $\mathbb{Z}/2\mathbb{Z}[x]/\langle x^5+x^2+1\rangle$.

Hint: *the last question is directly related to Problem II.9.*

Problem III.2. *Let* $d\in\mathbb{N}$ *and* $f\in\mathbb{K}[x]$ *of degree* d. *Let* $p\in\mathbb{K}[x]/\langle f\rangle$, *represented by a polynomial* $p=p_0+p_1x+\cdots+p_nx^n$ *of degree* n, *with possibly* $n\geq d$. *What is the complexity of finding the canonical representation of* p *in* $\mathbb{K}[x]/\langle f\rangle$, *in each of the following situations?*

(i) *General case: no assumption on* f.

(ii) *The case* $f=x^d$.

(iii) *The case* $f=x^d-1$.

Remark: *this problem arises naturally when one is computing in* $\mathbb{K}[x]/\langle f\rangle$. *For example, a multiplication in this quotient is performed by computing a polynomial product, which may have degree up to* $2d-2$, *and then computing the canonical representation of this product.*

III.1.2. Extended Euclidean algorithm

We recall basic facts from the previous chapter.

```
Algorithm 4: ExtendedEuclideanAlgorithm (see Algorithm 3)
Input: Two elements $a$ and $b$ in a Euclidean domain $\mathcal{R}$ with a height function $h$.
Output: A gcd of $a$ and $b$ in $\mathcal{R}$ together with the corresponding cofactors.
$r_0 := a$, $u_0 := 1$, $v_0 := 0$.
$r_1 := b$, $u_1 := 0$, $v_1 := 1$, $i := 1$
While $r_i \neq 0$ do
    $(q_i, r_{i+1}) :=$ PolynomialDivisionAlgorithm$(r_{i-1}, r_i)$
    $u_{i+1} = u_{i-1} - q_iu_i$, $v_{i+1} = v_{i-1} - q_iv_i$
    $i := i+1$
Return $r_{i-1}, u_{i-1}, v_{i-1}$
```

Theorem III.3 (Bézout's relation). *Let* $\mathcal{R}=\mathbb{Z}$ *or* $\mathcal{R}=\mathbb{K}[x]$. *For* a *and* b *in* $\mathcal{R}$, *there exist* u *and* v *in* $\mathcal{R}$ *such that* $au+bv=\gcd(a,b)$.

Corollary III.4. *Let* $\mathcal{R}=\mathbb{Z}$ *or* $\mathcal{R}=\mathbb{K}[x]$. *If* a *and* b *are coprime, then* a *is invertible modulo* b *and* b *is invertible modulo* a.

Thus, $\mathbb{Z}/n\mathbb{Z}$ *is a field, if and only if,* n *is prime.*

III.1.3. Definitions and Properties

Definition III.5. *A finite field is a field with a finite number of elements.*

Theorem III.6. *The characteristic of a finite field is a prime number $p > 0$. A finite field $\mathbb{K}$ of characteristic p has $q = p^r$ elements for some $r \geq 1$.*

- *If $r = 1$, then $\mathbb{K} = \mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$, the unique field of p elements.*
- *Otherwise,*
  - *there is a unique injective ring homomorphism from $\mathbb{Z}/p\mathbb{Z}$ to $\mathbb{K}$;*
  - *$\mathbb{K}$ is the unique, up to an isomorphism, field of $q = p^r$ elements, denoted by $\mathbb{F}_q = \mathbb{F}_{p^r}$;*
  - *$\mathbb{F}_q$ is* not *$\mathbb{Z}/q\mathbb{Z}$.*

III.1.4. Field Extensions

Definition III.7. *Let $\mathbb{K}$ and $\mathbb{L}$ be two fields. If $\mathbb{K} \subseteq \mathbb{L}$, then $\mathbb{K}$ and $\mathbb{L}$ share the same characteristic and $\mathbb{L}$ is said to be an* extension *of $\mathbb{K}$.*

Problem III.3. *Show that any ring homomorphism from $\mathbb{K}$ to $\mathbb{L}$ must be injective.*

Example III.8. *$\mathbb{C}$ is an extension of $\mathbb{R}$ and both $\mathbb{R}$ and $\mathbb{C}$ are extensions of $\mathbb{Q}$.*
For any $r \geq 1$, $\mathbb{F}_{p^r}$ is an extension of $\mathbb{F}_p$.

Definition III.9. *Let $\mathbb{K}$ be a field. Let $P \in \mathbb{K}[x]$ be a nonzero polynomial. P is* irreducible *if $P = QR$ with Q and R in $\mathbb{K}[x]$ implies that Q or R is invertible.*

Problem III.4.
1. *Show that any polynomial of $\mathbb{K}[x]$ of degree 1 is irreducible.*
2. *Show that any polynomial of $\mathbb{K}[x]$ of degree 2 or 3 is irreducible if, and only if, it has no root in $\mathbb{K}$.*

Problem III.5. *List all the irreducible polynomials of degree 2 and 3 in $\mathbb{F}_2[x]$.*

Corollary III.10. *Let $\mathcal{R} = \mathbb{Z}$ or $\mathcal{R} = \mathbb{K}[x]$. If a and b are coprime, then a is invertible modulo b and b is invertible modulo a.*
Thus, $K[x]/\langle P \rangle$ is a field, if and only if, P is irreducible.

Theorem III.11. *The quotient ring $\mathbb{K}[x]/\langle P \rangle$ has a natural $\mathbb{K}$-vector space structure of dimension $d = \deg P$, whose canonical basis is $1, x, \dots, x^{d-1}$.*
We say that $\mathbb{K}[x]/\langle P \rangle$ is an extension of degree *d.*

Example III.12.
1. *$\mathbb{C} \simeq \mathbb{R}[x]/\langle x^2 + 1 \rangle$.*
2. *$\mathbb{F}_4 \simeq \mathbb{F}_2[x]/\langle x^2 + x + 1 \rangle$.*
3. *$\mathbb{F}_8 \simeq \mathbb{F}_2[x]/\langle x^3 + x + 1 \rangle$.*

4. $\mathbb{F}_9 \simeq \mathbb{F}_3[x]/\langle x^2+1\rangle$.

Problem III.6. *Give the addition and multiplication tables of the following rings* $\mathcal{A}_1 = \mathbb{F}_3[x]/(x^2+1)$, $\mathcal{A}_2 = \mathbb{F}_3[x]/(x^2+2)$, $\mathcal{A}_3 = \mathbb{F}_3[x]/(x^2+x+2)$ *and* $\mathcal{A}_4 = \mathbb{F}_3[x]/(x^2+2x+1)$. *Which ones are integral domains? Which ones are fields?*

Problem III.7. *Note: the first four items below are directly related to Problem III.1.*

1. *Compute* $4+5$ *and* 4×5 *in* $\mathbb{F}_7$.
2. *Compute the inverse of* 23 *in* $\mathbb{F}_{31}$.
3. *Compute* $(2+2x)+(1+2x)$ *and* $(2+2x)\times(1+2x)$ *in* $\mathbb{F}_9 = \mathbb{F}_3[x]/\langle x^2+1\rangle$.
4. *Compute the inverse of* x^4+x+1 *in* $\mathbb{F}_{32} = \mathbb{F}_2[x]/\langle x^5+x^2+1\rangle$.
5. *Is* $\mathbb{F}_4[\beta]/\langle \beta^2+\beta+\alpha\rangle$, *with* $\mathbb{F}_4 = \mathbb{F}_2[\alpha]/\langle \alpha^2+\alpha+1\rangle$, *a field?*
6. *Compute the inverse of* $\alpha+\beta$, *if it exists, in* $\mathbb{F}_4[\beta]/\langle \beta^2+\beta+\alpha\rangle$.

Problem III.8.
1. *Let* $\mathbb{F}_8 = \mathbb{F}_2[x]/\langle x^3+x+1\rangle$.
   a. *Show that for all* $z = a+bx+cx^2$ *in* $\mathbb{F}_8$, $z^2+z+1 \neq 0$.
   b. *Deduce that* $\mathbb{F}_4$ *is* not *included in* $\mathbb{F}_8$.
2. a. *Show that* x^4+x+1 *is irreducible in* $\mathbb{F}_2[x]$.
   b. *Let* $\mathbb{F}_{16} = \mathbb{F}_2[x]/\langle x^4+x+1\rangle$ *and let* $z = x^2+x+1$. *Compute* z^2+z+1.
   c. *Deduce that* $\mathbb{F}_4$ is *included in* $\mathbb{F}_{16}$.
   d. *Show that* y^2+y+x *is irreducible in* $\mathbb{F}_4[y]$.
   e. *What is the quotient ring* $\mathbb{F}_4[y]/\langle y^2+y+x\rangle$*?*

Theorem III.13. *Let* $\mathbb{K}$ *be a field. Let* $\mathbb{K}'$ *be an extension of* $\mathbb{K}$ *of degree* d. *Let* $\mathbb{K}''$ *be an extension of* $\mathbb{K}'$ *of degree* d'.

Then, $\mathbb{K}''$ *is an extension of* $\mathbb{K}$ *of degree* dd'.

For $q = p^r$ *and* $s \geq 1$, $\mathbb{F}_{q^s}$ *is an extension of* $\mathbb{F}_q$ *of degree* s *and an extension of* $\mathbb{F}_p$ *of degree* rs.

Problem III.9.
1. *Compute the inverse of* x^2 *in* $\mathbb{F}_{256} = \mathbb{F}_4[x]/\langle x^4+\alpha x^3+x^2+\alpha x+1\rangle$, *with* $\mathbb{F}_4 = \mathbb{F}_2[\alpha]/\langle \alpha^2+\alpha+1\rangle$.
2. *Give all the fields that are included in* $\mathbb{F}_{2^{28}}$, $\mathbb{F}_{3^{42}}$ *and* $\mathbb{F}_{5^{193}}$.

III.2. Matrix viewpoint

As we said before, $\mathbb{K}[x]/\langle P\rangle$ is a $\mathbb{K}$-vector space of dimension $d = \deg P > 0$. As a quotient ring, it is also a ring, thus it is a $\mathbb{K}$-*algebra*: a ring containing $\mathbb{K}$ and a $\mathbb{K}$-vector space.

As a consequence, for any $h \in \mathbb{K}[x]/\langle P\rangle$, the *multiplication map by* h is a $\mathbb{K}$-linear map:

$$\begin{aligned}\mu_h :\ \mathbb{K}[x]/\langle P\rangle &\to \mathbb{K}[x]/\langle P\rangle\\ f &\mapsto fh \mod P.\end{aligned}$$

Since $K[x]/\langle P\rangle$ has dimension d, then μ_h can be represented by a $d\times d$ matrix given by $\mu_h(x^i)$ for all $0 \le i < d$. And by linearity, one can recover all the μ_h thanks to only all μ_{x^i}, $0 \le i < d$.

In fact, the map μ that associates the matrix of μ_h to h is an *algebra homomorphism* (ring homomorphism and $\mathbb{K}$-linear map) between $\mathbb{K}[x]/\langle P\rangle$ and $\mathbb{K}^{d\times d}$. That is $\mu_1 = \mathrm{Id}$, $\mu_{h+h'} = \mu_h + \mu_{h'}$, $\mu_{hh'} = \mu_h\mu_{h'}$ and for all $\lambda \in \mathbb{K}$, $\mu_{\lambda h} = \lambda\mu_h$. Notice that since $\mathbb{K}[x]/\langle P\rangle$ is a commutative ring, all these matrices commute!

Problem III.10. 1. *Show that for a monic polynomial $P = x^d + p_{d-1}x^{d-1} + \cdots + p_0 \in \mathbb{K}[x]$, the matrix of the multiplication by x is the* companion matrix *of P:*

$$\mu_x = \begin{pmatrix} 0 & \cdots & \cdots & 0 & -p_0\\ 1 & \ddots & \ddots & \vdots & \vdots\\ 0 & \ddots & \ddots & \vdots & \vdots\\ \vdots & \ddots & \ddots & \vdots & \vdots\\ 0 & \cdots & 0 & 1 & -p_{d-1}\end{pmatrix}.$$

2. *What is the matrix of the multiplication by x^2?*

3. *Show that P is the minimal polynomial of μ_x.*

Example III.14. *The multiplication map by x in $\mathbb{F}_3[x]/\left\langle x^3 + x + 1\right\rangle$ in the basis $(1, x, x^2)$ is given by*

$$\begin{pmatrix} 0 & 0 & 2\\ 1 & 0 & 2\\ 0 & 1 & 0\end{pmatrix};$$

Theorem III.15. *Let $\mathbb{K}$ be a field and $P \in \mathbb{K}[x]$ be a polynomial of degree $d > 0$. Then, $\mathbb{K}[x]/\langle P\rangle$ is isomorphic to the subalgebra*

$$\mathbb{K}[C_P] = \left\{a_0\,\mathrm{Id} + a_1C_P + \cdots + a_{d-1}C_P^{d-1} \,\middle|\, a_0,\ldots,a_{d-1} \in \mathbb{K}\right\},$$

where C_P is the companion matrix of P.

Problem III.11. 1. *Show that*

$$\mathbb{F}_{27} \simeq \left\{\begin{pmatrix} a & 2c & 2b\\ b & a+2c & 2b+2c\\ c & b & a+2c\end{pmatrix} \,\middle|\, a,b,c \in \mathbb{F}_3\right\}.$$

2. *Are the following matrix algebras finite field? If yes, which ones?*

$$\left\{\left.\begin{pmatrix} a & 2c & 2b+2c \\ b & a+2c & 2b+c \\ c & b+c & a+b \end{pmatrix}\right| a,b,c \in \mathbb{F}_3\right\}, \qquad \left\{\left.\begin{pmatrix} a & 0 & 0 & 0 \\ b & a+d & c & b \\ c & b & a+d & c \\ d & c & b & a+d \end{pmatrix}\right| a,b,c,d \in \mathbb{F}_2\right\}.$$

Problem III.12. 1. *Give the multiplication matrices of μ_α, μ_β and $\mu_{\alpha\beta}$ of respectively α, β and $\alpha\beta$ in $\left(\mathbb{F}_2[\alpha]/\langle\alpha^2+\alpha+1\rangle\right)[\beta]/\langle\beta^2+\beta+\alpha\rangle$.*

2. *Deduce a matrix representation of $\mathbb{F}_{16}$ in $\mathbb{F}_2^{4\times4}$.*

3. *Find an invertible matrix $Q \in \mathbb{F}_2^{4\times4}$ such that $Q\mu_\beta Q^{-1}$ is a companion matrix.*

4. *Deduce that $\mathbb{F}_{16} = \mathbb{F}_2[\beta]/(P)$ with $P \in \mathbb{F}_2[\beta]$ irreducible of degree 4.*

III.3. Cryptography application

In cryptography, many algebraic structure such as lattices or elliptic curves are defined over a finite field. For instance, the AES algorithm performs operations in $\mathbb{F}_{256}$ built as an extension of $\mathbb{F}_2$, which is: $\mathbb{F}_{256} = \mathbb{F}_2[x]/\langle x^8+x^4+x^3+x+1\rangle$.

Problem III.13. *We let $\mathbb{F}_{256}$ be defined as above.*

1. *Compute $(x^6+x+1)(x^4+x)$.*

2. *Compute the inverse of x.*

3. *Compute the inverse of x^4+x^2+x+1.*

Problem III.14 (Frobenius homomorphism). *Let $\mathbb{F}_q$ be a finite field of characteristic p.*

1. *Show that for $0 < i < p$, $\binom{p}{i} = 0 \bmod p$.*

2. *Deduce that for $a, b \in \mathbb{F}_q$, $(a+b)^p = a^p + b^p$. The map $a \mapsto a^p$ is called the Frobenius map or homomorphism.*

3. *For $a, b \in \mathbb{F}_q$, compute $(a+b)^q$.*

4. *Let $\mathbb{F}_{q^s}$ be an extension of $\mathbb{F}_q$. Show that $a \in \mathbb{F}_{q^s}$ satisfies $a^q = a$ if, and only if, $a \in \mathbb{F}_q$.*

Theorem III.16. *For q a power of a prime and $s \in \mathbb{N}^*$, the map $a \in \mathbb{F}_{q^s} \to a^{q^s} \in \mathbb{F}_{q^s}$ is the identity map.*

Furthermore, $a \in \mathbb{F}_{q^s}$ lies in a subfield of $\mathbb{F}_{q^s}$ isomorphic to $\mathbb{F}_q$ if, and only if, $a = a^q$.

Remark III.17. *For $q^s = p$ a prime number, this is Fermat's little theorem.*

Example III.18. *Since $\mathbb{F}_{16} = \mathbb{F}_2[z]/\langle z^4 + z + 1\rangle$ and $16 = 4^2$, we can find the elements that lie in $\mathbb{F}_4$. Let $h = a + bz + cz^2 + dz^3 \in \mathbb{F}_{16}$. Then, $h \in \mathbb{F}^4$ iff $h^4 = h$.*

$$h^4 = (a + b + c + d) + (b + d)z + (c + d)z^2 + dz^3 = a + bz + cz^2 + dz^3 = h.$$

This is equivalent to

$$\begin{cases} b &= c \\ d &= 0 \end{cases}.$$

Hence, the elements in $\mathbb{F}_4$ are 0, 1, $z + z^2$ and $1 + z + z^2$.

Problem III.15. *Let $\mathbb{F}_4 = \mathbb{F}_2[x]/\langle x^2 + x + 1\rangle$ and $\mathbb{F}_{64} = \mathbb{F}_4[y]/\langle y^3 + y + 1\rangle$. Compute the elements of $\mathbb{F}_{64}$ lying in $\mathbb{F}_8$.*

Corollary III.19. *The group $\mathbb{F}_q^*$ is cyclic of order $q - 1$, i.e. there exists $a \in \mathbb{F}_q^*$ such that for all $b \in \mathbb{F}_q^*$, b is a power of a.*

Implementation III.1. *This must be done in C.*

Implement integer arithmetic modulo n with n of at most 32 bits over unsigned integer types: addition, subtraction, multiplication and inversion. Elements of $\mathbb{Z}/n\mathbb{Z}$ shall be represented in $\{0, \dots, n - 1\}$.

Implementation III.2. *Implement polynomial arithmetic over the ring $\mathbb{Z}/n\mathbb{Z}$: addition, subtraction, multiplication, Euclidean division and extended Euclidean algorithm. The polynomials will be stored in a dense fashion as an array of coefficients.*

Implementation III.3. *This must be done in C.*

Implement polynomial arithmetic over $\mathbb{F}_p$ modulo P: addition, subtraction, multiplication and inversion. Elements of $\mathbb{F}_p[x]/\langle P\rangle$ shall be represented as polynomials of degree less than $\deg P$.

IV. Formal power series and fast polynomial division algorithm

This chapter introduces the algebraic object called “formal power series”, the definition of basic operations with these, and a criterion for invertibility. A fast inversion algorithm is described, based on Newton iteration. This is then applied in order to perform polynomial division with remainder in quasi-optimal complexity. Finally we study the problem of rational reconstruction, which also serves as an opening towards future aspects of the course.

In all this chapter, $\mathbb{K}$ is a field and $\mathsf{M}(n)$ is a function for the complexity of multiplying univariate polynomials of degree less than n with coefficients in $\mathbb{K}$.

IV.1. Introduction

The definition of (formal) power series is fairly straightforward:

Definition IV.1. *A* (formal) power series *over the field $\mathbb{K}$ is an infinite sequence $S = (s_i)_{i\in\mathbb{N}}$ with $s_i \in \mathbb{K}$ for all i, indexed by the set of nonnegative integers $\mathbb{N} = \{0, 1, 2, \dots\}$.*

Now, for algebraic and algorithmic purposes that will become clearer later in the course, we want to be able to perform some basic operations with this type of object. We are interested for example in the addition, multiplication, and inversion of power series.

Proposition IV.2. *The set of formal power series with coefficients in $\mathbb{K}$ is a ring, when equipped with the following operations. Let $S = (s_i)_{i\in\mathbb{N}}$ and $T = (t_i)_{i\in\mathbb{N}}$ be two power series over $\mathbb{K}$. The addition is defined in a natural coefficient-wise manner, as follows:*

$$S + T = (s_i + t_i)_{i\in\mathbb{N}}.$$

The multiplication is of the so-called “convolution” type, defined as follows:

$$ST = \left(\sum_{k=0}^{i} s_k t_{i-k}\right)_{i\in\mathbb{N}}.$$

Problem IV.1. *Show the above proposition, and determine the 0 and 1 elements in the ring of power series.*

One may notice a strong similarity with univariate polynomials, which can be seen as (and can be in fact defined as) infinite sequences of elements in $\mathbb{K}$ with only a finite number of nonzero coefficients, with the exact same operations of addition and multiplication as above. That is to say, univariate polynomials over $\mathbb{K}$ are exactly power series over $\mathbb{K}$ which have *finite support.*

Problem IV.2. *Show that $\mathbb{K}[x]$ is a subring of $\mathbb{K}[\![x]\!]$.*

By similarity with univariate polynomials, we will therefore usually write the power series $S = (s_i)_{i\in\mathbb{N}}$ as $S = \sum_{i\in\mathbb{N}} s_i x^i$, for some indeterminate x. The ring of power series over $\mathbb{K}$ is then concisely denoted by $\mathbb{K}[\![x]\!]$.

To highlight again the link with polynomials, remark that the above definition of multiplication of power series can be rewritten as follows. For $S = \sum_{i\in\mathbb{N}} s_i x^i$ and $T = \sum_{j\in\mathbb{N}} t_j x^j$ both in $\mathbb{K}[\![x]\!]$,

$$ST = \left(\sum_{i\in\mathbb{N}} s_i x^i\right)\left(\sum_{j\in\mathbb{N}} t_j x^j\right) = \sum_{i\in\mathbb{N}}\left(\sum_{k=0}^{i} s_k t_{i-k}\right) x^i.$$

One may already anticipate some obstacles if such an object must be represented in a computer: we cannot store an infinite list of coefficients in a finite memory! As a consequence, for most computational purposes, we deal with power series that admit some other (finite) representations of power series: fractions of two polynomials; a truncated list of only the first n coefficients for some suitable n; a solution of some differential equations; etc. Here is a specific example of a power series in $\mathbb{K}[\![x]\!]$ that can be represented as a rational fraction, i.e., a fraction of two univariate polynomials in $\mathbb{K}[x]$.

Example IV.3 (Generating series and fraction expansion)**.** *Recall that the Fibonacci sequence $(f_i)_{i\in\mathbb{N}}$ is defined over the field of rational numbers $f_i \in \mathbb{K} = \mathbb{Q}$, by the initial terms $f_0 = 0$ and $f_1 = 1$ and the recurrence $f_{i+2} = f_{i+1} + f_i$ for all $i \in \mathbb{N}$. We can associate to this sequence the power series $F = \sum_{i\in\mathbb{N}} f_i x^i$. It is called the* generating series *of the sequence. Using the above definition of multiplication of power series, one verifies that F satisfies*

$$\begin{aligned}(1 - x - x^2)F &= \sum_{i\in\mathbb{N}} f_i x^i - \sum_{i\in\mathbb{N}} f_i x^{i+1} - \sum_{i\in\mathbb{N}} f_i x^{i+2}\\ &= f_0 + f_1 x - f_0 x + \sum_{i\in\mathbb{N}} \underbrace{(f_{i+2} - f_{i+1} - f_i)}_{=0} x^{i+2}\\ &= x.\end{aligned}$$

Thus, F can be represented as a rational fraction, explicitly, $F = \frac{x}{1-x-x^2} \in \mathbb{K}(x)$. When this situation arises, one says that $F = \sum_{i\in\mathbb{N}} f_i x^i$ is the power series expansion *of this fraction.* □

When representing power series as a truncated list of the first n coefficients, one does not have access to the coefficients beyond these n first ones. Therefore one has to consider that two power series S and T are "equal at precision n" as soon as these first n coefficients are the same. This means precisely that the first n coefficients of $S - T$ are zero, i.e. x^n divides $S - T$. In that case, one says that $S - T$ has *valuation* at least n, and more generally:

Definition IV.4. *For a power series $S = \sum_{i\in\mathbb{N}} s_i x^i \in \mathbb{K}[\![x]\!]$, the* ($x$-adic) valuation *of S, denoted by $\mathrm{v}_x(S)$, is the largest integer $i \in \mathbb{N}$ such that x^i divides S, or, equivalently, the smallest integer i such that $s_i \neq 0$. For $S = 0$, we let $\mathrm{v}_x(0) = \infty$.*

Example IV.5. $\mathrm{v}_x(x^2) = 2$ *and* $\mathrm{v}_x(x^3 + 2x) = 1$.

The following related notation is often used. For two series S and T in $\mathbb{K}[\![x]\!]$, we write $S = T + O(x^n)$ or $S = T \bmod x^n$ if the valuation of $S - T$ is at least n. In this case we say that S and T are equal at precision n.

Problem IV.3. *Given two power series S and T in $\mathbb{K}[\![x]\!]$ known at precision n, show that one can compute $S + T \bmod x^n$ in $O(n)$ operations in $\mathbb{K}$, and one can compute $S \cdot T \bmod x^n$ in $\mathsf{M}(n)$ operations in $\mathbb{K}$.*

Problem IV.4. *Consider the formal power series $A = \sum_{i\in\mathbb{N}} x^i$ over some field $\mathbb{K}$. Compute $(1-x)A$. Is there an inverse of $1 - x$ in $\mathbb{K}[\![x]\!]$?*

Problem IV.5. *Is there an inverse of x in $\mathbb{K}[\![x]\!]$?*

Remark IV.6. *The following remarks are for culture; more details can be found in algebra textbooks. The* norm *associated to the x-adic valuation is defined as $|A| = \mathrm{e}^{-\mathrm{v}_x(A)}$. In the example above, $|x^2| = \mathrm{e}^{-2}$ and $|0| = 0$. The ring of polynomials $\mathbb{K}[x]$, seen as a $\mathbb{K}$-vector space, is not* complete *for this norm. Indeed, the sequence $(A_n)_{n\in\mathbb{N}}$ defined by $A_n = \sum_{i=0}^{n} x^i$ is a Cauchy sequence: $|A_m - A_n| \leq \mathrm{e}^{-m-1}$ for all $m < n$, yet it has no limit in $\mathbb{K}[x]$. The ring of power series $\mathbb{K}[\![x]\!]$ is the* completion *of $\mathbb{K}[x]$ for this norm. As an exercise, one can show that the above norm satisfies the following* ultrametric *property: $|A + B| \leq \max(|A|, |B|)$ for all $A, B \in \mathbb{K}[x]$.*

IV.2. Power series inversion: Newton iteration

We have seen that truncated power series can be efficiently added and multiplied, relying on operations on polynomials. We now turn to inversion. This is an important operation for power series. We did not focus on it for polynomials, since the only polynomials that are invertible are the ones that are elements of $\mathbb{K} \setminus \{0\}$ (they are inverted in 1 operation in $\mathbb{K}$). In constrast, many power series are invertible, and inverting them is nontrivial: the next lemma states that a power series $s_0 + s_1 x + \cdots$ is invertible exactly when its constant coefficient s_0 is nonzero.

Lemma IV.7. *A power series $S = \sum_{i\in\mathbb{N}} s_i x^i \in \mathbb{K}[\![x]\!]$ is invertible in $\mathbb{K}[\![x]\!]$ if and only if $s_0 \neq 0$.*

Proof. Assume $s_0 = 0$, i.e. $S = xT$ where $T = \sum_{i\in\mathbb{N}} s_{i+1} x^i$. Any inverse $U \in \mathbb{K}[\![x]\!]$ for S would satisfy $1 = US = xUT$, which is impossible since the constant coefficient of xUT is 0. Hence S is not invertible.

Conversely, assume $s_0 \neq 0$. We are going to describe a method for constructing the inverse $U = u_0 + u_1 x + u_2 x^2 + \cdots$ of S. Using the definition of multiplication for power series, we are looking for such a U which satisfies:

$$1 = u_0 s_0$$

$$\begin{aligned} 0 &= u_0 s_1 + u_1 s_0 \\ 0 &= u_0 s_2 + u_1 s_1 + u_2 s_0 \\ 0 &= u_0 s_3 + u_1 s_2 + u_2 s_1 + u_3 s_0 \\ \text{etc}\ldots \end{aligned}$$

Since s_0 is nonzero, the first equation defines u_0 uniquely as $u_0 = s_0^{-1}$. From this, the second equation defines u_1 uniquely as $u_1 = -u_0 s_1 s_0^{-1} = -s_1 s_0^{-2}$. Then the third equation defines u_2 uniquely as $u_2 = -(u_1 s_1 + u_0 s_2) s_0^{-1} = -(-s_1^2 s_0^{-2} + s_0^{-1} s_2) s_0^{-1} = s_1^2 s_0^{-3} - s_2 s_0^{-2}$. And so on and so forth: this constructs an inverse U for S, hence S is invertible. □

Problem IV.6. *Consider an invertible power series $S = \sum_{i\in\mathbb{N}} s_i x^i \in \mathbb{K}[\![x]\!]$, known at some precision n. Based on the above proof, write an algorithm for computing its inverse at the same precision n, i.e., $S^{-1} \bmod x^n$. What is the complexity of this algorithm?*

Problem IV.7. *Consider the power series $S = 3 + 2x^2 + x^3 + O(x^5) \in \mathbb{K}[\![x]\!]$, for $\mathbb{K} = \mathbb{F}_5$, known at precision 5. Compute the inverse of S at precision 5, using the above algorithm.*

Problem IV.8. *Show that for any nonzero ideal I of $\mathbb{K}[\![x]\!]$, there exists a unique integer $v \geq 0$ such that $I = \langle x^v \rangle$.* Hint: Show that a nonzero series S is in I if and only if $x^{\nu_x(S)} \in I$.

The above proof of Lemma IV.7 and the corresponding Problem IV.6 provide a direct iterative method to compute the inverse of an invertible power series. Yet, such inverses can be computed much more efficiently using the so-called *Newton iteration* method, based on the following result. It states that from an inverse of S truncated at precision n, one can deduce an inverse truncated at precision $2n$ using a few additions and multiplications of truncated power series, which allows us to rely on fast multiplication of truncated power series (which can be done via fast multiplication of univariate polynomials).

Lemma IV.8. *Let $S \in \mathbb{K}[\![x]\!]$ be an invertible power series, and let $T \in \mathbb{K}[\![x]\!]$ be such that $T = S^{-1} + O(x^n)$. Then the power series $U = T + (1 - TS)T$ satisfies $U = S^{-1} + O(x^{2n})$.*

Proof. By assumption, there exists $H \in \mathbb{K}[\![x]\!]$ such that $T = S^{-1} + x^n H$. From this it follows that $1 - TS = -x^n HS$. These identities yield

$$\begin{aligned} U &= T + (1 - TS)T \\ &= \left(S^{-1} + x^n H\right) + (1 - TS)\left(S^{-1} + x^n H\right) \\ &= \left(S^{-1} + x^n H\right) - x^n HS\left(S^{-1} + x^n H\right) \\ &= S^{-1} + x^n H - x^n H - x^{2n} HSH \\ &= S^{-1} + O(x^{2n}). \end{aligned}$$ □

Remark IV.9. *Since $1 - TS = -x^n HS$ is $O(x^n)$, we have $U = T + O(x^n)$, that is explicitly: $u_0 = t_0, \ldots, u_{n-1} = t_{n-1}$. Thus, the first n terms of $U = T + (1 - TS)T$ are known from T, and the next terms $u_n, \ldots, u_{2n-1}$ are given by the terms $n, \ldots, 2n-1$ of $(1 - TS)T$. Furthermore, we know that $TS = 1 + O(x^n)$, hence we only need to compute its terms of degrees $n, \ldots, 2n-1$.*

Remark IV.10. *For a differentiable function F, the Newton operator of F is defined as follows: for an approximated root x_k of $F(x) = 0$,*

$$x_{k+1} = x_k - \frac{F(x_k)}{F'(x_k)}.$$

In the present case, since the inverse of the power series S is the root of the function $F(x) = \frac{1}{x} - S$, we have the following Newton operator:

$$x_{k+1} = x_k - \frac{F(x_k)}{F'(x_k)} = x_k + (1 - x_k S)x_k.$$

Thus, in the above lemma, U is the power series obtained from this Newton operator applied to T.

We obtain the following algorithm. Here, the representation of a truncated power series at precision n is simply a polynomial of degree less than n.

```
**Algorithm 5:** Power Series Inversion via Newton iteration
**Input:** A power series $S = s_0 + \cdots + s_{n-1}x^{n-1} + O(x^n) \in \mathbb{K}[[x]]$ at precision $n$, with $s_0 \neq 0$.
**Output:** The power series $U \in \mathbb{K}[[x]]$, at precision $n$, which satisfies $U = S^{-1} + O(x^n)$.
**If** $n = 1$ **then Return** $s_0^{-1}$
Recursive call: find $T$ the inverse of $s_0 + \cdots + s_{m-1}x^{m-1} + O(x^m)$ at precision $m = \lceil \frac{n}{2} \rceil$
$U := T + (1 - TS)T$, with all operations at precision $2m$
**Return** $U + O(x^n)$
```

Proposition IV.11. *Algorithm 5 is correct: on input S at precision n, it returns S^{-1} at precision n. If n is a power of 2, it uses $3\mathsf{M}(n) + O(n)$ operations in $\mathbb{K}$. Otherwise, it uses $O(\mathsf{M}(n))$ operations in $\mathbb{K}$.*

Proof. The algorithm is obviously correct in the base case, i.e. at precision $n = 1$. Then we prove correctness by induction on the precision n: we assume the algorithm is correct for all input series at precision less than n, and we show it is also correct for all series at precision $n + 1$.

Let $S = s_0 + \cdots + s_{n-1}x^{n-1} + O(x^n) \in \mathbb{K}[[x]]$ be an invertible power series known at precision n, for some $n \geq 2$. Then it is easily verified that $m = \lceil \frac{n}{2} \rceil$ is less than n. So, by assumption, the recursive call correctly computes T at precision m such that $TS = 1 + O(x^m)$. Thus, by Lemma IV.8, the computed series U is the inverse of S at precision $2m$. By construction, $2m \geq n$ (in fact, $2m$ is either n or $n + 1$), so the returned value is indeed the inverse of S at precision n.

For complexity, let $C(n)$ be the number of operations performed by the algorithm on input precision n. In the base case $n = 1$, the algorithm does a single inversion in $\mathbb{K}$, so $C(1) = 1$. For $n \geq 2$, the algorithm uses:

- a recursive call at precision $m = \lceil n/2 \rceil$: cost $C(m)$;
- two additions of power series at precision $2m$: cost $\leq 4m$;
- two multiplications of power series at precision $2m$: cost $2\mathsf{M}(2m)$.

This yields $C(n) \le C(m) + 2\mathsf{M}(2m) + 4m$. If $n = 2^\ell$ is a power of 2, one has $m = n/2$ so this becomes $C(n) \le C(n/2) + 2\mathsf{M}(n) + 2n$. Unrolling this recurrence yields

$$C(n) \le C(1) + \sum_{k=0}^{\ell-1} \left(2\mathsf{M}\left(\frac{n}{2^k}\right) + 2\frac{n}{2^k}\right).$$

Now we will use the superlinearity of $\mathsf{M}(\cdot)$: one has $\mathsf{M}(a) + \mathsf{M}(b) \le \mathsf{M}(a+b)$ for any $a, b \ge 1$. Therefore

$$\sum_{k=0}^{\ell-1} \mathsf{M}\left(\frac{n}{2^k}\right) \le \mathsf{M}\left(\sum_{k=0}^{\ell-1} \frac{n}{2^k}\right).$$

It remains to use the geometric sum identity

$$\sum_{k=0}^{\ell-1} \frac{1}{2^k} = \frac{1 - \left(\frac{1}{2}\right)^\ell}{1 - \frac{1}{2}} \le \frac{1}{1 - \frac{1}{2}} = 2.$$

So

$$C(n) \le 1 + 2\mathsf{M}\left(\sum_{k=0}^{\ell-1} \frac{n}{2^k}\right) + 2\sum_{k=0}^{\ell-1} \frac{n}{2^k}$$
$$C(n) \le 1 + 2\mathsf{M}(2n) + 2n.$$

For quasi-linear $\mathsf{M}(n) = n \log(n) \log\log(n)$, this is bounded by $4\mathsf{M}(n) + O(n)$. Being more careful about the available coefficients and about the needed coefficients in the computation of U (see Remark IV.9 for such considerations), one can show that the constant 4 can be replaced by 3. Using a so-called middle-product operation, one can further lower this to 2. □

Example IV.12. *Consider the power series $S = 3 + 2x^2 + x^3 + x^7 \in \mathbb{F}_5[[x]]$, known at precision 8. Let us compute the inverse of S at precision 8, using Algorithm 5.*

1. *$n = 8$: recursive call on $S = 3 + 2x^2 + x^3 + O(x^4)$ at precision 4*
   a. *$n = 4$: recursive call on $S = 3 + O(x^2)$ at precision 2*
      i. *$n = 2$: recursive call on $S = 3 + O(x)$ at precision 1*
         . *$n = 1$: return $T = 3^{-1} + O(x) = 2 + O(x)$*
      ii. *return $U = T + (1 - TS)T + O(x^2) = 2 + 0 \cdot T + O(x^2) = 2 + O(x^2)$*
   b. *return $V = U + (1 - US)U + O(x^4) = 2 - (4x^2 + 2x^3)2 + O(x^4) = 2 + 2x^2 + x^3 + O(x^4)$*
2. *return $W = V + (1 - VS)V + O(x^8) = 2 + 2x^2 + x^3 - (x^4 + x^5 + 4x^6 + 3x^7)(2 + 2x^2 + x^3) + O(x^8) = 2 + 2x^2 + x^3 + 2x^4 + 2x^5 + 4x^7 + O(x^8)$.*

Problem IV.9. *Compute the inverse of $S = \sum_{i\in\mathbb{N}} (i + 2)\, x^i \in \mathbb{F}_7[[x]]$ at precision 8.*

Implementation IV.1. Iterative inversion of power series.

- *Using SageMath, implement an iterative algorithm for power series inversion, based on the proof of Lemma IV.7 and the corresponding Problem IV.6. Hint: in SageMath, power series coefficients are immutable; it may be easier to compute the coefficients of T in a list, and then convert this list to a power series.*
- *Implement the version of this algorithm that builds the (triangular) matrix of the linear system, and directly calls a linear system solver to find the inverse.*
- *Compare running times for both versions, and observe their behaviour (is it quadratic in the precision? cubic?).*

Implementation IV.2. Newton inversion of power series.

- *Using SageMath, implement Algorithm 5. Hint: after the recursive call, be careful with the precision of T when computing U.*
- *Compare running times of Newton inversion to the running times of Implementation IV.1; is the behaviour quasi-linear, quadratic, ...?*
- *Compare running times of Newton inversion to those of SageMath's native implementation.*

IV.3. Fast polynomial division

We are now going to use the above fast inversion of power series to give an algorithm of quasi-optimal complexity for the division with remainder for univariate polynomials.

Let us first give the rough idea underlying the algorithm. As input, we have two polynomials A and B with B nonzero; let m be the degree of A and n be the degree of B. We seek polynomials Q and R such that $A = BQ + R$ and $\deg(R) < \deg(B) = n$. In what follows we assume $m \geq n$, otherwise the solution is $(Q, R) = (0, A)$. Going to rational fractions, this relation can be rewritten as (here we use $B \neq 0$):

$$\frac{A}{B} = Q + \frac{R}{B}.$$

Now, imagine that the base field is the real numbers $\mathbb{K} = \mathbb{R}$ and temporarily see the polynomials as functions that we can look at from an mathematical analysis point of view. Then, because of the degree constraints $\deg(R) < \deg(B)$, the fraction $\frac{R(x)}{B(x)}$ tends to 0 when $x \longrightarrow +\infty$. Therefore Q corresponds to the asymptotic expansion of $\frac{A}{B}$ at infinity. Hence, one can obtain Q by computing the Taylor expansion at infinity of the fraction $\frac{A}{B}$.

Let us come back to our algebraic setting, with an abstract field $\mathbb{K}$ where "$x \longrightarrow \infty$" does not make sense. Yet we can still consider Taylor expansions at 0. For example we have seen above that $\frac{1}{1-x} = \sum_{i \in \mathbb{N}} x^i$. More generally, we have seen the inversion of power series, which gives a way to expand as a power series any rational fraction $\frac{A}{B}$ (for polynomials A and B) as soon as the constant coefficient of B is nonzero.

To adapt the above approach of an expansion at ∞, it suffices (1) to perform the change of variables $y \leftarrow \frac{1}{x}$:

$$\frac{A(1/x)}{B(1/x)} = Q(1/x) + \frac{R(1/x)}{B(1/x)},$$

which is still an identity between rational fractions, and (2) to multiply each side by x^{m-n}, to ensure that we only manipulate polynomials in numerators and denominators:

$$\frac{x^m A(1/x)}{x^n B(1/x)} = x^{m-n} Q(1/x) + \frac{x^m R(1/x)}{x^n B(1/x)}.$$

Observe that $m - n$ is precisely the degree of Q, and that here all numerators and denominators are polynomials. Furthermore, the polynomial $x^n B(1/x)$ is invertible as a power series since its constant coefficient is the coefficient of degree $n = \deg(B)$ of B, which is nonzero by assumption. Since $\deg(R) < n$ and $m \geq n$, the polynomial $x^m R(1/x) = x^{m-n+1} x^{n-1} R(1/x)$ has valuation at least $m - n + 1$, which is greater than the degree of the polynomial $x^{m-n} Q(1/x)$. Thus, the expansion of $\frac{x^m A(1/x)}{x^n B(1/x)}$ at precision $m - n + 1$ will give us all coefficients of the polynomial $x^{m-n} Q(1/x)$, from which we can deduce Q. Then R can be obtained as $R = A - BQ$.

More formally, this yields the following algorithm.

```
FastPolynomialDivisionAlgorithm($A, B$)

    Input:  Polynomials $A$ and $B$ in $\mathbb{K}[x]$ with $B$ nonzero.

    Output:  Polynomials $(Q, R)$ in $\mathbb{K}[x]$ such that $A = BQ + R$ and $\deg(R) < \deg(B)$.

    1. Let $m = \deg(A)$ and $n = \deg(B)$
    2. If $m < n$, return $(0, A)$
    3. Compute the reversals $\tilde{A} = x^m\, A(1/x)$ and $\tilde{B} = x^n\, B(1/x)$
       (this step does not require any arithmetic operation in $\mathbb{K}$)
    4. Compute $\tilde{Q} = \tilde{A}/\tilde{B} \mod x^{m-n+1}$ by inverting a formal power series and performing a
       power series multiplication, both at precision $m - n + 1$
    5. Deduce $Q$ by reverting the coefficients of $\tilde{Q}$ (note that the reversal must be done with
       respect to degree $\deg(Q) = m - n$, even though $\tilde{Q}$ might have degree less than $m - n$)
    6. Deduce $R$ by computing $A - BQ$
    7. Return $(Q, R)$
```

Theorem IV.13. *Let A and B be two polynomials in $\mathbb{K}[x]$ of respective degrees m and n with $m \geq n \geq 0$. The Euclidean division with remainder of A and B can be computed in $O(\mathsf{M}(m-n) + \mathsf{M}(n))$ operations in $\mathbb{K}$.*

Problem IV.10. *Prove Theorem IV.13.*

Problem IV.11. *Compute the Euclidean division of $A = x^9 + x^3 + 2$ and $B = x^3 + 2x + 2$ in $\mathbb{F}_3[x]$, using the above fast algorithm.*

Implementation IV.3. Fast polynomial division with remainder.

- *Using operations on univariate polynomials and power series in SageMath, implement the fast polynomial division algorithm presented in this section.*
- *Measure running times for dividing A of degree $2n$ by B of degree n. How do these times grow when n grows? (is it quasi-linear, quadratic, …?)*
- *Compare it to:*
  - *your own implementation of the Euclidean division algorithm for polynomials, done in a previous lab;*
  - *SageMath's routine* `quo_rem` *;*
  - *the time for multiplying two polynomials of degree n in SageMath.*

IV.4. Rational reconstruction

As already observed above, for computational purposes, one cannot store as such the infinite sequence of coefficients of a formal power series. Above, we have already exploited two representations:

- An exact representation, where we manage to represent the power series S as a fraction $\frac{A}{B}$ of two polynomials (whose lists of coefficients are finite, and hence can be stored). For example, we have seen that $\frac{1}{1-x} = \sum_{i\in\mathbb{N}} x^i$, and that $F = \frac{x}{1-x-x^2}$ for F the generating series of the Fibonacci sequence.
- An approximate representation, where we consider the first n coefficients of the power series: $S = s_0 + s_1 x + \cdots + s_{n-1}x^{n-1} + O(x^n)$, in which case we say that S is known at precision n. We used the latter representation of power series to design a fast algorithm for polynomial division with remainder, based on Newton inversion (Algorithm 5).

A natural question is then: *how efficiently can we switch between the two representations?* This contains two questions, depending which representation is the input and which one is the output. The first question can be reformulated as: how fast can we find the power series expansion of a fraction $\frac{A}{B}$ of two polynomials (with $b_0 \neq 0$ for invertibility)? We have already studied this above: combining the multiplication and the inversion of power series at precision n, we obtain the first n coefficients of the expansion of $\frac{A}{B}$ in complexity $O(\mathsf{M}(n))$. In this section, we study the second question:

> *given $S = s_0 + s_1 x + \cdots + s_{n-1}x^{n-1} + O(x^n)$ known at precision n, how fast can we find polynomials A and B such that $\frac{A}{B} = S + O(x^n)$?*

Observe that a given fraction $\frac{A}{B}$ with $b_0 \neq 0$ will define a power series uniquely, via its expansion. However, in the latter question, suitable polynomials A and B may depend on n:

- Given a power series S, there is no guarantee that there exists polynomials A and B such that $S = \frac{A}{B}$ at full precision (i.e. at precision n for all $n \in \mathbb{N}$).

- Even when such polynomials exist, without further information on their degrees we cannot hope to find them from the first n terms of S. For example, assume a power series is known at precision 1, and its constant coefficient is 1: one cannot know if this power series is $\sum_{i\in\mathbb{N}} x^i = \frac{1}{1-x}$, or $\sum_{i\in\mathbb{N}} 2^i x^i = \frac{1}{1-2x}$, or $F = \frac{1}{1-x-x^2} = \sum_{i\in\mathbb{N}} f_{i+1}x^i$ where $(f_i)_{i\in\mathbb{N}}$ is the Fibonacci sequence.

So, without further knowledge on the series S whose n first terms are known, A and B may depend on n. But then, one would be tempted to answer the question by taking the trivial solution $A = s_0 + s_1 x + \cdots + s_{n-1}x^{n-1}$ and $B = 1$ (note that A depends on n), which satisfy $\frac{A}{B} = S + O(x^n)$. However, as one may expect, this solution is generally not interesting and brings no information compared to the knowledge of S at precision n. The related algorithmic problem that is indeed interesting and useful for other tasks is called *rational reconstruction*, and requires further degree constraints on A and B which forbid this trivial solution:

Definition IV.14. *Let X and S in $\mathbb{K}[x]$ be polynomials with $n = \deg(X) > \deg(S)$. For $k \in \{1, \dots, n\}$, the* rational reconstruction of S modulo X *is a pair $(A, B) \in \mathbb{K}[x]^2$ satisfying*

$$\gcd(B, X) = 1,\ \deg(A) < k,\ \deg(B) \le n - k,\ \frac{A}{B} = S \bmod X.$$

In particular, we call (A, B) a Padé approximant *whenever $X = x^n$, and a* Cauchy interpolant *whenever $X = \prod_{i=0}^{n-1}(x - x_i)$ with $x_0, \dots, x_{n-1}$ pairwise distinct elements of $\mathbb{K}$.*

Remark IV.15. *One may observe the following facts, on the problem of computation a rational reconstruction of S modulo X:*

1. *If $S = 0$, then $(A, B) = (0, 1)$ is a solution to the problem.*

2. *If $k = n$, then $(A, B) = (S, 1)$ is a solution of the problem. This is the direct generalization of the trivial solution mentioned in the paragraph above Definition IV.14.*

3. *If $k < n$, it is possible that the problem has no solution. This is the case for $n = 3, k = 2, X = x^3$ and $S = x^2 + 1$. Indeed, by hypotheses, $B = ax + b$ with $b \neq 0$, thus $A = (ax + b)(x^2 + 1) = bx^2 + ax + b \bmod x^3$ and $\deg(A) > 1$. This is a contradiction.*

4. *If the problem has a solution (A, B), then the fraction $\frac{A}{B}$ corresponding to all nonzero solutions (A, B) is unique.*

Problem IV.12. *Prove the last assertion.*

Observe that if (A, B) is a solution of the rational reconstruction problem, then it is also a solution of the following problem

$$\deg(A) < k, \ \deg(B) \leq n - k, \ A = SB \bmod X. \tag{IV.1}$$

Here again, all solutions $(A, B) \neq (0, 0)$ give rise to the same fraction $\frac{A}{B}$. However, this version of the problem always has a solution (A, B) which is nontrivial, i.e. a solution other than $(0, 0)$.

Problem IV.13. *Prove the last assertion, by rewriting Eq. (IV.1) as a homogeneous linear system over $\mathbb{K}$ with $n + 1$ unknowns and n equations.*

From $A = SB \bmod X$, we deduce that there exists $U \in \mathbb{K}[x]$ such that $A = XU + SB$ which is a relation of Bézout type, already encountered in the context of extended GCDs in Chapter II.

Theorem IV.16. *Let $X \in \mathbb{K}[x]$ of degree $n > 0$ and $S \in \mathbb{K}[x]$ with $\deg S < n$. Let $k \in \{1, 2, \ldots, n\}$ and let R_j, U_j, V_j be the remainder and the associated cofactors obtained at jth iteration of the extended Euclidean algorithm called on X and S. If j is minimal such that $\deg(R_j) < k$, then*

1. *There exists a solution $(A, B) \neq (0, 0)$ to Eq. (IV.1), which is $(A, B) = (R_j, V_j)$. If furthermore $\gcd(X, V_j) = \gcd(R_j, V_j) = 1$, then (R_j, V_j) is a solution of the rational reconstruction problem in Definition IV.14.*
2. *If the rational reconstruction problem has a solution, let $\frac{A}{B} \in \mathbb{K}(x)$ be an irreducible fraction coming from a solution (A, B), then there exists $c \in \mathbb{K}^*$ such that $A = cR_j$ and $B = cV_j$.*

As a result, the rational reconstruction of S modulo X has a solution if and only if $\gcd(X, V_j) = 1$.

Example IV.17. *Let $n = 8$, $S = x^5 + 2x^4 + x^3 + x^2 + 2x + 1 \in \mathbb{F}_3[x]$ and we seek a rational fraction $\frac{A}{B} \in \mathbb{F}_3(x)$ with $k = 3$. We unroll the extended Euclidean algorithm called on x^8 and S, finding*

i	R_i	U_i	V_i
0	x^8	1	0
1	$x^5 + 2x^4 + x^3 + x^3 + 2x + 1$	0	1
2	$x^4 + 2x^3 + x^2 + x + 2$	1	$2x^3 + 2x^2 + 2$
3	1	$2x$	$x^4 + x^3 + x + 1$

Observe that $\deg R_3 = 0 < k$, hence we stop the algorithm. We check that R_3 and V_3 are coprime. We conclude that $S = \frac{R_3}{V_3} \bmod x^8$.

Problem IV.14. *Give all the rational reconstructions of $E = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + O(x^4) \in \mathbb{F}_5[[x]]$.*

Problem IV.15. *Let $\mathbf{u} = (u_i)_{i \in \mathbb{N}}$ be the Fibonacci sequence modulo 11. That is $u_0 = 0$, $u_1 = 1$ and for all $i \in \mathbb{N}$, $u_{i+2} = u_{i+1} + u_i \bmod 11$. Let $S = u_0x^7 + u_1x^6 + \cdots + u_6x + u_7 \in \mathbb{F}_{11}$. Compute its rational reconstruction.*

Problem IV.16. *For $n = 4$ and $k = 2$, we want to find the irreducible rational fraction $\frac{A}{B} \in \mathbb{F}_7(x)$ such that*

$$\frac{A(1)}{B(1)} = 5, \ \frac{A(3)}{B(3)} = 0, \ \frac{A(2)}{B(2)} = 2, \ \frac{A(6)}{B(6)} = 2.$$

1. *Compute X and S.*
2. *Compute A and B.*
3. *Check the result.*
4. *Estimate the complexity for computing X, S, A and B.*

Implementation IV.4. Rational reconstruction.

- *Using SageMath's univariate polynomial operations, implement a solution to the rational reconstruction problem based on Theorem IV.16. It takes as input two polynomials X and S and a parameter k and returns — if any — the pair of polynomials (A, B) as in Definition IV.14.*

  Hint: *you can start from your code for the extended Euclidean algorithm.*

- *Measure the running time of your implementation for various values of n, taking for example $k = \lceil n/2 \rceil$.*

- *Compare these running times to SageMath's native routine for this task (it is a method defined among others for univariate polynomials, and called* `rational_reconstruction`*).*

  Hints: *You can see examples of how to call this method by defining a polynomial* `f` *and typing* `f.rational_reconstruction?` *followed by* `Enter`*. You can see the source code of this method by typing* `f.rational_reconstruction??` *followed by* `Enter`*.*

IV.5. Hensel lifting

In here, we extend Section IV.2. Let F be a polynomial in $\mathbb{K}[\![x]\!][y]$, we want to compute its roots in $\mathbb{K}[\![x]\!]$. Hensel's Lemma allows us to determine the roots of F in $\mathbb{K}[\![x]\!]$ from the approximated roots in $\mathbb{K}$.

Lemma IV.18 (Hensel's Lemma)**.** *Let $F \in \mathbb{K}[\![x]\!][y]$ and let $s_0 \in \mathbb{K}$ be a simple root of $F \bmod x = 0$. Then, there exists a unique solution $S \in \mathbb{K}[\![x]\!]$ of $F(y) = 0$ such that $S = s_0 \bmod x$.*

Furthermore, we can compute S by applying iteratively the Newton operator starting with s_0 and its quadratic convergence is ensured

If an approximated root x_k is known in precision n and $x_{k+1} = x_k - \frac{F(x_k)}{F'(x_k)}$, then the terms of degrees $n, \ldots, 2n-1$ of x_{k+1} are given by the quotient $-\frac{F(x_k)}{F'(x_k)}$

Example IV.19. *Let $F = y^2 - (1+x) \in \mathbb{F}_3[[x]][y]$. Modulo x, The equation $F(y) = 0$ becomes $y^2 - 1 = 0$ whose (simple) solutions are $s_0 = 1$ and $\sigma_0 = -1$. The derivative, in y, of F is $2y$, hence the Newton operator is*

$$x_{k+1} = x_k - \frac{x_k^2 - (1+x)}{2x_k}.$$

Starting with $x_0 = s_0 + O(x) = 1 + O(x)$, we find

1. $x_1 = 1 + 2x + O(x^2)$;

2. $x_2 = 1 + 2x + x^2 + x^3 + O(x^4)$;

3. $x_3 = 1 + 2x + x^2 + x^3 + 2x^4 + x^5 + O(x^8)$;

4. …

Starting with σ_0, we would have found the other solution $\mathbb{F}_3[[x]]$.

Problem IV.17. *Compute the other root of Example IV.19.*

Problem IV.18. *Compute the roots of $F = y^2 + y + x \in \mathbb{F}_2[[x]][y]$.*

Implementation IV.5. *Implement the Hensel lifting algorithm over $\mathbb{F}_q[[z]]$ taking as an input an approximated root, the polynomial whose root we are computing and an integer for the precision of the root.*

Test the functions with Problem IV.18.

V. Guessing linear recurrence relations

In this chapter, we study more systematically some aspects that have appeared above in the particular case of the Fibonacci sequence. Remember that this sequence is defined by 2 initial terms and a degree-2 linear recurrence. As we have seen, one can associate to this sequence a so-called *generating power series*, which happens to be a fraction of two polynomials. We will observe that this holds more generally for all linearly recurrent sequences. Then, we will consider the situation where we have a certain number of terms of a sequence, and from these we want to guess a linear recurrence satisfied by the whole sequence; this can be solved using rational reconstruction. Finally, we will apply this to give an algorithm that efficiently solves *sparse* linear systems, that is, systems whose matrices have most entries equal to zero.

V.1. Linearly recurrent sequences and generating series

Definition V.1. *A sequence* $\mathbf{b} = (b_i)_{i\in\mathbb{N}}$ *with terms in a field* $\mathbb{K}$ *is said to be* linearly recurrent *if there exists* $d \in \mathbb{N}$ *and* $v_0, \ldots, v_{d-1} \in \mathbb{K}$ *such that*

$$\forall\, i \in \mathbb{N}, \quad b_{i+d} + v_{d-1}b_{i+d-1} + \cdots + v_0 b_i = 0.$$

Any such integer $d \in \mathbb{N}$ *is called an* order *of the linear recurrence.*

Example V.2.

1. *The only linearly recurrent sequence of order* $d = 0$ *is the zero sequence* $\mathbf{b} = (0)_{i\in\mathbb{N}}$.

2. *For a linearly recurrent sequence* $\mathbf{b}$ *of order* $d = 1$, *we have* $b_{i+1} + v_0 b_i = 0$ *for all* $i \in \mathbb{N}$. *Thus* $\mathbf{b}$ *is a geometric sequence:* $b_i = (-v_0)^i b_0$, *for all* $i \in \mathbb{N}$. *Notice that* $\mathbf{b}$ *also satisfies a recurrence of order* d *for any integer* $d > 0$.

3. *The sequence* $\mathbf{b} = (i)_{i\in\mathbb{N}}$ *satisfies* $b_{i+1} - b_i - 1 = 0$ *for all* $i \in \mathbb{N}$. *It is not linearly recurrent of order* 1, *but it also satisfies* $b_{i+2} - 2b_{i+1} + b_i = 0$ *for all* $i \in \mathbb{N}$, *and is recurrent of order* 2.

Problem V.1. *Show that for any linearly recurrent sequence* $\mathbf{b} = (b_i)_{i\in\mathbb{N}}$ *with coefficients in* $\mathbb{K}$, *there is a unique smallest integer* d *and unique* $v_0, \ldots, v_{d-1} \in \mathbb{K}$ *such that for all* $i \in \mathbb{N}$,

$$b_{i+d} + v_{d-1}b_{i+d-1} + \cdots + v_0 b_i = 0.$$

Problem V.2. *Show that for two linearly recurrent sequences* $\mathbf{b}$ *and* $\mathbf{c}$, *of respective orders* d *and* e, *the sum* $\mathbf{b} + \mathbf{c}$ *admits a linear recurrence of order at most* $d + e$.

The next proposition gives a link between the fact that a sequence $(b_i)_{i\in\mathbb{N}}$ is linearly recurrent and the fact that the corresponding *generating series* $\sum_{i=0}^{\infty} b_i x^i$ is a proper rational fraction (that is, a fraction of polynomials with numerator of degree strictly less than the degree of the denominator). Furthermore, the irreducible such fraction has denominator given by the coefficients of the recurrence.

Proposition V.3. *A sequence* $\mathbf{b} = (b_i)_{i\in\mathbb{N}}$ *over* $\mathbb{K}$ *is linearly recurrent of order* d *of smallest relation*

$$b_{i+d} + v_{d-1}b_{i+d-1} + \cdots + v_0 b_i = 0$$

if, and only if, there exists a polynomial $R \in \mathbb{K}[x]$ *such that* $\deg(R) < d$ *and*

$$\sum_{i=0}^{\infty} b_i x^i = \frac{R}{1 + v_{d-1}x + \cdots + v_0 x^d}.$$

Problem V.3. *Prove Proposition V.3.*

Remark V.4. *For a sequence* $\mathbf{b}$ *whose terms are "not given by a formula", testing that a linear recurrence relation is satisfied by* $\mathbf{b}$ *is not possible (in fact, think about it: how would you define this algorithmic problem, formally? what is the input?).*

Guessing *a linear recurrence relation satisfied by* $\mathbf{b}$ *is the problem of computing a suitable linear recurrence relation based on finitely many terms of* $\mathbf{b}$.

V.2. Guessing linearly recurrent sequences

An algorithm cannot take *all* the sequence terms as input since there are infinitely many; it is often mathematically meaningful and algorithmically efficient to only consider the first D terms, $b_0, \ldots, b_{D-1}$. If D is sufficiently large, one can hope that the found linear recurrence actually holds for the whole (hypothetical) infinite sequence $b_0, b_1 \ldots,$. In practical cases where such computations are used (see for example Section V.3), one usually knows a bound on the order of the recurrence of the considered sequence, which allows to select a suitable D which ensures that a recurrence for the first D terms actually gives a recurrence for the whole sequence.

Computing a linear recurrence relation of order d satisfied by $\mathbf{b} = (b_i)_{i\in\mathbb{N}}$, from its first D terms, comes down to finding $v_0, \ldots, v_{d-1}$ such that

$$\begin{cases} v_0 b_0 + \cdots + v_{d-1}b_{d-1} + b_d & = 0 \\ v_0 b_1 + \cdots + v_{d-1}b_d + b_{d+1} & = 0 \\ & \vdots \\ v_0 b_{D-1-d} + \cdots + v_{d-1}b_{D-2} + b_{D-1} & = 0. \end{cases}$$

Thus, we look for the smallest $d \in \{0, \dots, D-1\}$ such that the *Hankel matrix*

$$\begin{pmatrix} b_0 & \cdots & b_{d-1} & b_d \\ b_1 & \cdots & b_d & b_{d+1} \\ \vdots & & \vdots & \vdots \\ b_{D-1-d} & \cdots & b_{D-2} & b_{D-1} \end{pmatrix}$$

has a right kernel which contains a vector of the form $\begin{pmatrix} v_0 \\ \vdots \\ v_{d-1} \\ 1 \end{pmatrix}$.

The matrix-vector product

$$\begin{pmatrix} b_0 & \cdots & b_{d-1} & b_d \\ b_1 & \cdots & b_d & b_{d+1} \\ \vdots & & \vdots & \vdots \\ b_{D-1-d} & \cdots & b_{D-2} & b_{D-1} \end{pmatrix} \begin{pmatrix} v_0 \\ \vdots \\ v_{d-1} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

can be extended into

$$\begin{pmatrix} b_0 & \cdots & b_{d-1} & b_d \\ b_1 & \cdots & b_d & b_{d+1} \\ \vdots & & \vdots & \vdots \\ b_{D-1-d} & \cdots & b_{D-2} & b_{D-1} \\ b_{D-d} & \cdots & b_{D-1} & 0 \\ \vdots & \iddots & \iddots & \vdots \\ b_{D-1} & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} v_0 \\ \vdots \\ v_{d-1} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ r_{d-1} \\ \vdots \\ r_0 \end{pmatrix},$$

with $r_0, \dots, r_{d-1}$ unknown. This product corresponds to the multiplication of two polynomials, namely $B_D = \sum_{i=0}^{D-1} b_i x^{D-1-i}$ and $V = x^d + \sum_{i=0}^{d-1} v_i x^i$, truncated at precision x^D.

This means that we look for a polynomial V, monic of degree d, such that $B_D V \bmod x^D$ has the form $R = r_{d-1}x^{d-1} + \cdots + r_0$, i.e., it has degree less than $d = \deg(V)$. In other words, we look for V and R such that

$$x^D U + B_D V = R, \quad \deg V > \deg R, \quad \text{for some polynomial } U.$$

This can also be rewritten as

$$R = B_D V \bmod x^D, \ \deg(V) = d, \ \deg(R) < d,$$

which is a particular case of Eq. (IV.1) called *Padé approximation*. Note that the translation from the latter equation takes $n = D$ and $k = d$ and $d \le n - k = D - d$, hence we must have $D \ge 2d$ if we wish to be able to find a solution with the results of Section IV.4.

Historically, guessing linearly recurrent sequences has been done with the Berlekamp-Massey algorithm. Several variations of the original Berlekamp-Massey algorithm have been described. Among them, the above viewpoint allows us to rely on the rational reconstruction method of Section IV.4, which is essentially based on the extended Euclidean algorithm: we call the extended Euclidean algorithm on x^D and $B_D = \sum_{i=0}^{D-1} b_i x^{D-1-i}$, and we stop as soon as the degree of the current remainder R becomes smaller than the degree of the Bézout coefficient V of B_D.

Theorem V.5. *Guessing a linear recurrence from the first D terms of a sequence can be done in $O(D^2)$ operations in $\mathbb{K}$. Using algorithms elaborating upon the fast extended Euclidean algorithm, this problem can be solved in complexity $O(\mathsf{M}(D)\log(D))$.*

Example V.6. *Let $(f_i)_{i\in\mathbb{N}}$ be the Fibonacci sequence with $f_0 = f_1 = 1$ and let $D = 6$. We have $B_6 = f_0 x^5 + \cdots + f_5 = x^5 + x^4 + 2x^3 + 3x^2 + 5x + 8$.*

i	R_i	U_i	V_i	Q_i
0	x^6	1	0	
1	P_5	0	1	$x-1$
2	$-x^4-x^3-2x^2-3x+8$	1	$-x+1$	$-x$
3	$13x+8$	x	$-x^2+x+1$	

Since $\deg(13x+8) < \deg(-x^2+x+1)$, *we stop here and return* $-x^2+x+1$ *meaning that the polynomial of the relation is* $-x^2+x+1$, *that is* $-f_{i+2}+f_{i+1}+f_i = 0$, *or* $f_{i+2}-f_{i+1}-f_i = 0$ *for all i.*

Problem V.4. *Let $\mathbf{u} = (u_i)_{i\in\mathbb{N}} = (2^i + i^2 - 1)_{i\in\mathbb{N}}$ be a sequence defined over $\mathbb{F}_7$.*

1. *Show that $\mathbf{u}$ satisfies the linear recurrence relation for all $i \in \mathbb{N}$, $u_{i+21} - u_i = 0$.*
2. *Guess a linear recurrence relation of order 4 satisfied by $\mathbf{u}$, using only its first 10 terms.*
3. *Is it guaranteed that the relation you guessed is in fact correct for the whole sequence $\mathbf{u}$ (in particular, beyond the 10 first terms)?*
4. *In the present situation, how to prove this relation is correct for the whole sequence?*

V.3. Application to operations on sparse matrices

A matrix is said to be *sparse* if many of its coefficients are zero. In general, we consider that a collection of matrices $(M_n)_{n\in\mathbb{N}}$, where $M_n \in \mathbb{K}^{n\times n}$, is a collection of *sparse* matrices if the number of nonzero entries of M_n is in $O(n^{1+\varepsilon})$ with $\varepsilon < 1$.

Definition V.7. *A* sparse representation *of a matrix is such that only nonzero entries are stored.*

Example V.8. *A matrix M is stored a list of triplets $(a_{i,j}, i, j)$ with $a_{i,j} \neq 0$.*

Over $\mathbb{F}_2$, we even can avoid writing the nonzero coefficient, since it is 1! The M4RI library proposes the JCF format based on that and a text input must follow this template

```
nrows ncols modulus
nonzero_entries_upper_bound
column_index
```

where `nrows` *(resp.* `ncols`*) is the number of rows (resp. of columns) of the matrix,* `modulus` *is the modulus (always* 2*, here) and* `nonzero_entries_upper_bound` *is an upper bound on the number of nonzero entries of the matrix. This format does not allow the matrix to have a zero row. The number* `column_index` *is the column index of the coefficient; the row index is implicit, it is increased by* 1 *each time* `column_index` *is negative.*

```
 3 7 2
 8

-2
 4
 7
-1
 2
 3
-2
 3
```

represents the matrix

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

V.3.1. Multiplication

Given a matrix with m nonzero entries, stored with a sparse representation, its product with a vector requires $O(m)$ operations in the base field.

Problem V.5. *Prove this assertion.*

For matrix product, we could hope that, likewise, the complexity depends on m if both input matrices have $O(m)$ nonzero entries. Recall that with a dense representation, if the matrices are $n \times n$, the Coppersmith–Winograd algorithm has complexity $O(n^{2.38})$.

Take two $n \times n$ matrices A and B, and assume they have at most m_A and m_B nonzero entries, respectively. The naive algorithm can take some advantage of this sparsity, since the product $a_{i,k}b_{k,j}$ is zero whenever $a_{i,k} = 0$ or $b_{k,j} = 0$. Let $\bar{a}_k$ (resp. $\bar{b}_k$) be the number of nonzero entries of the kth column (resp. row) of A (resp. B). In particular, $\sum_{k=1}^n \bar{a}_k = m_A$ and $\sum_{k=1}^n \bar{b}_k = m_B$. Then the number of multiplications in the base ring that one does in the naive matrix multiplication AB is $\sum_{k=1}^n \bar{a}_k\bar{b}_k \leq (\sum_{k=1}^n \bar{a}_k)(\sum_{k=1}^n \bar{b}_k) \leq m_A m_B$. Note that one should also consider the number of additions, and that the details of such a naive sparse matrix multiplication algorithm will be highly dependent on the chosen sparse representation format.

V.3.2. Sparse linear system solving

The common strategy to solve a general linear system is to compute the PLU decomposition of the matrix and then to solve two triangular systems.

PLU decomposition

We know that for dense matrices, PLU decomposition and inversion have computation complexities equivalent to that of matrix product. However, a sparse matrix can have a dense PLU decomposition

The family $(M_n)_{n\in\mathbb{N}}$ of sparse matrices, defined over rational numbers,

$$M_n = \begin{pmatrix} 1 & 2 & 2 & \cdots & 2 \\ 2 & 1 & 0 & \cdots & 0 \\ 2 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 2 & 0 & \cdots & 0 & 1 \end{pmatrix} \in \mathbb{Q}^{n\times n}$$

has the following PLU decomposition (with identity permutation):

$$\begin{aligned} M_n &= L_n U_n \\ &= \begin{pmatrix} 1 & & & & \\ 2 & 1 & & & \\ 2 & \frac{4}{3} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ 2 & \frac{4}{3} & \cdots & \frac{4}{4n-9} & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 2 & 2 & \cdots & 2 \\ & -3 & -4 & -4 & \cdots & -4 \\ & & \frac{7}{3} & \frac{4}{3} & \cdots & \frac{4}{3} \\ & & & \frac{11}{7} & \ddots & \vdots \\ & & & & \ddots & \frac{4}{4n-13} \\ & & & & & \frac{4n-5}{4n-9} \end{pmatrix}. \end{aligned}$$

These matrices L_n and U_n are not sparse. Yet, observe that on this specific example we can also obtain a sparse PLU decomposition, by swapping the first and last rows.

The Wiedemann algorithm

The goal of this algorithm is to efficiently compute a nonzero vector in the kernel of a matrix. This is a probabilistic algorithm.

In all this subsection, $M \in \mathbb{K}^{n\times n}$ is a sparse matrix. We let v_0 be a randomly picked vector of $\mathbb{K}^n$ and we let $v = Mv_0$. We also pick at random a vector $u \in \mathbb{K}^n$.

The matrix M has a minimal polynomial $P = z^r + p_{r-1}z^{r-1} + \cdots + p_0$ of degree $r \leq n$, that is, there exist $r \in \mathbb{N}$ minimal and $p_0, \ldots, p_{r-1} \in \mathbb{K}$ such that

$$M^r + p_{r-1}M^{r-1} + \cdots + p_0 \operatorname{Id} = 0.$$

Multiplying on the right this equality by v, we obtain $s_r + p_{r-1}s_{r-1} + \cdots + p_0 s_0 = 0$, where $s_i = M^i v$. We denote $\mathbf{s} = (s_i)_{i\in\mathbb{N}} = (v, Mv, M^2v, \ldots) = (M^i v)_{i\in\mathbb{N}}$. The terms of this vector sequence are computed recursively, $s_0 = v$ and for all $i \in \mathbb{N}$, $s_{i+1} = Ms_i$.

Since $v = Mv_0$, then

$$s_r + p_{r-1}s_{r-1} + \cdots + p_0 s_0 = M(M^r v_0 + p_{r-1}M^{r-1}v_0 + \cdots + p_0 v_0) = 0.$$

Hence, $M^r v_0 + p_{r-1}M^{r-1}v + \cdots + p_0 v_0$ is a vector in the kernel of M. Unfortunately, this vector is the zero one.

Let us assume, however, that there exists $d \in \mathbb{N}$ such that $d < r$ and $q_0, \ldots, q_{d-1} \in \mathbb{K}$ such that

$$s_d + q_{d-1}s_{d-1} + \cdots + q_0 s_0 = M(M^d v_0 + q_{d-1}M^{d-1}v_0 + \cdots + q_0 v_0) = 0$$
$$M^d v_0 + q_{d-1}M^{d-1}v_0 + \cdots + q_0 v_0 \neq 0.$$

Then, $M^d v_0 + q_{d-1}M^{d-1}v_0 + \cdots + q_0 v_0$ is a nonzero vector in the kernel of M.

Wiedemann's idea is to compute these $q_0, \ldots, q_{d-1}$ by multiplying first this equation on the left by $u^{\mathrm{T}}M^i$ for all i. We, thus, obtain

$$\forall i \in \mathbb{N}, \quad b_{i+d} + q_{d-1}b_{i+d-1} + \cdots + q_0 b_i = 0,$$

where $b_i = u^{\mathrm{T}}M^i v$ and $\mathbf{b} = (b_i)_{i\in\mathbb{N}} = (u^{\mathrm{T}}v, u^{\mathrm{T}}Mv, u^{\mathrm{T}}M^2 v, \ldots) = (u^{\mathrm{T}}M^i v)_{i\in\mathbb{N}}$. These sequence terms can be computed as follows: $b_i = u^{\mathrm{T}}s_i$.

In other words, the sequence $\mathbf{b}$ satisfies the *linear recurrence relation* for all i, $b_{i+d}+q_{d-1}u_{i+d-1}+\cdots+q_0 b_i = 0$.

To compute these d and $q_0, \ldots, q_{d-1}$, we use rational reconstruction to determine the linear recurrence relation of smallest order satisfied by the sequence.

Theorem V.9. *The Wiedemann algorithm, called on an $n \times n$ sparse matrix with at most m nonzero entries, $m \geq n$, uses $O(nm)$ operations in the base field.*

Proof. The computation of the sequence $\mathbf{s}$ is done in $O(nm)$ operations. Then, deducing the sequence $\mathbf{b}$ is done in $O(n^2)$ operations, with $n^2 \leq nm$.

Determining the linear recurrence relation uses $O(\mathsf{M}(n)\log n)$ operations in the base field. One can take $\mathsf{M}(n)$ to be quasi-linear in n, so that this is in $O(n^2)$, hence this costs $O(nm)$ field operations (in fact, even the Karatsuba algorithm is enough for $O(\mathsf{M}(n)\log n)$ to be in $O(n^2)$; however, the naive quadratic algorithm would not be suitable).

In short, building the sequence is the bottleneck of the algorithm. □

Problem V.6. *We want to solve the linear system over $\mathbb{F}_{13}$*

$$Av = \begin{pmatrix} 1 & 2 & 2 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = b.$$

1. *Show that this comes down to computing a vector, whose coordinates depend on v_1, v_2 and*

v_3, in the kernel of

$$A' = \begin{pmatrix} 1 & 2 & 2 & 1 \\ 2 & 1 & 0 & 1 \\ 2 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

2. *Compute the vector sequence of the Wiedemann algorithm for $v_0 = \begin{pmatrix} 11 \\ 11 \\ 10 \\ 10 \end{pmatrix}$ and the scalar sequence for $u = \begin{pmatrix} 4 \\ 12 \\ 3 \\ 12 \end{pmatrix}$.*
3. *Determine the smallest linear recurrence relation satisfied by* $\mathbf{u}$.
4. *Deduce a potential vector in the kernel of A'.*
5. *Check that this vector is indeed in the kernel of A'.*
6. *Deduce the solution of the original system.*

Problem V.7. *Let $\mathbf{u} = (u_i)_{i \in \mathbb{N}}$ be a sequence and let $H_{m,n} = (h_{i,j})_{\substack{0 \le i \le m \\ 0 \le j \le n}} = (u_{i+j})_{\substack{0 \le i \le m \\ 0 \le j \le n}}$ a Hankel matrix with $m \ge n$. We consider the property $\mathcal{P}$: "the last column of the matrix is linearly independent from the previous ones".*

Let us assume that $H_{m,n}$ has rank $n-1$ and does not satisfy property $\mathcal{P}$. Let us assume that the matrix $H_{m+1,n}$ does satisfy $\mathcal{P}$ and thus has rank n.

Show that all matrices $H_{m+1-p,n+p}$, for $0 \le p \le m-n+1$, satisfy $\mathcal{P}$ but that $H_{n-1,m+2}$ does not.

Hint: *we can start checking that this is indeed the case for the sequence* $\mathbf{u} = (0, 1, 1, 2, 3, 5, 9, \ldots)$ *for $n = 2$ and $m = 3$.*

Implementation V.1. *Using SageMath:*

- *Use your rational reconstruction code to write a function for guessing an order d linear recurrence from D terms of a sequence, with $D \ge 2d$.*
- *Implement the Wiedemann algorithm for solving sparse linear systems.*
- *Working over prime fields $\mathbb{Z}/p\mathbb{Z}$, use experiments to estimate its probability of success depending on the bitlength of the prime p and on the matrix size n.*

VI. Fast polynomial evaluation and interpolation

VI.1. Introduction and reminders

Let $\mathbb{K}$ be a field. We denote by $\mathsf{M}(n)$ the complexity of multiplying two polynomials of degree less than n over $\mathbb{K}$. For an arbitrary $\mathbb{K}$, we can take $\mathsf{M}(n) = O(n \log(n) \log(\log(n)))$ using an FFT-based algorithm due to Cantor&Kaltofen. We denote by ω a feasible exponent of the complexity of matrix multiplication over $\mathbb{K}$, meaning that two $n \times n$ matrices over $\mathbb{K}$ can be multiplied in $O(n^\omega)$ operations. We can take $\omega \leq \log_2(7) \approx 2.81$ using the Strassen algorithm. The best known upper bound for ω is $\omega \leq 2.37286$ (Alman and Vassilevska Williams, 2021). The naive lower bound on ω is $\omega \geq 2$. Note that $\omega = 2$ would correspond to a algorithm with linear complexity, since the size of an $n \times n$ matrix is n^2 coefficients from $\mathbb{K}$.

This chapter concerns the two following problems, which are inverses of each other.

Question VI.1 (Multipoint evaluation)**.** *Input: n elements $x_0, \ldots, x_{n-1} \in \mathbb{K}$ and a polynomial $P = p_{n-1}x^{n-1} + \cdots + p_0 \in \mathbb{K}[x]$ of degree less than n. How to efficiently compute $y_i = P(x_i)$ for $0 \leq i < n$?*

Question VI.2 (Interpolation)**.** *Input: n pairwise distinct elements $x_0, \ldots, x_{n-1} \in \mathbb{K}$ and n elements $y_0, \ldots, y_{n-1} \in \mathbb{K}$. How to efficiently compute a polynomial $P = p_{n-1}x^{n-1} + \cdots + p_0 \in \mathbb{K}[x]$ such that $P(x_i) = y_i$ for all $0 \leq i < n$?*

Note that, unlike in the interpolation problem, the points $x_0, \ldots, x_{n-1}$ need not be distinct in the multipoint evaluation problem.

These problems can be seen from a linear algebra viewpoint, since multipoint evaluation can be realized by matrix-vector multiplication where the vector contains the polynomial coefficients and the matrix is a so-called Vandermonde matrix built from the evaluation points:

$$\begin{pmatrix} 1 & x_0 & \cdots & x_0^{n-1} \\ 1 & x_1 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}.$$

This matrix is called the *Vandermonde matrix* of $(x_0, \ldots, x_{n-1})$. It is known to be invertible when $x_i \neq x_j$ holds for all $i \neq j$, which is among our assumptions for the interpolation problem: in that case, one can solve the interpolation problem via linear system solving.

Problem VI.1. *What is the complexity of multipoint evaluation using this linear algebra point of view?*

Problem VI.2. *Show that, indeed, the Vandermonde matrix of $(x_0, \ldots, x_{n-1})$ over $\mathbb{K}$ is invertible if, and only if, $x_0, \ldots, x_{n-1}$ are pairwise distinct. When that is the case, what is the complexity of interpolation using this linear algebra point of view?*

Hence these problems can be solved respectively in $O(n^2)$ operations via a matrix-vector product, and in $O(n^\omega)$ operations via linear system solving. In this chapter, we will see how to obtain quasi-optimal complexity for both problems, i.e. complexity linear in n up to some extra logarithmic factors.

The results of this chapter also show that matrix-vector products and linear system solving can be performed in complexity $\tilde{O}(n)$ when the matrix is Vandermonde one. Observe that the Vandermonde matrix has a special structure: it is defined by only n field elements instead of n^2, which reflects the fact that it comes from the evaluation/interpolation problems with only n points as input. It is this specificity which allows us to perform basic operations faster than for an arbitrary $n \times n$ matrix. In a later chapter of this course, we will show similar results (quasi-linear complexity for matrix-vector products and linear system solving) for a much more general class of structured matrices.

VI.1.1. Naive multipoint evaluation

The very naive algorithm to evaluate a polynomial $P = p_{n-1}x^{n-1} + \cdots + p_0 \in \mathbb{K}[x]$ at a point x_i is to compute the powers of x_i and perform a dot product with the coefficients of P. This uses roughly $2n$ multiplications and n additions. *This naive algorithm should never be used.*

What you should use instead for evaluation at a single point is a less naive (but very classical and very simple to state and implement) algorithm called the Horner scheme. It uses about n multiplication and n additions and is based on the writing:

$$P(x_i) = (\cdots((p_{n-1}x_i + p_{n-2})x_i + p_{n-3})x_i + \cdots)x_i + p_0.$$

To evaluate P at several points $x_0, \ldots, x_{n-1}$, one can repeatedly use Horner's scheme, leading to $O(n^2)$ operations in $\mathbb{K}$ in total. Observe that with the algorithm (not to be used) in the first paragraph, this is basically the same as building the Vandermonde matrix and performing the matrix-vector product as explained above.

VI.1.2. Lagrange interpolation

A first approach for interpolation is to rely on the Lagrange formula

$$P(x) = \sum_{0 \le i < n} y_i \prod_{\substack{0 \le j < n \\ j \ne i}} \frac{x - x_j}{x_i - x_j}.$$

Let us analyse the complexity of this approach. More precisely, for computing P we will use the above formula rewritten as follows:

$$P(x) = \sum_{i=0}^{n-1} y_i \frac{L_i(x)}{L_i(x_i)} \quad \text{where } L_i(x) = \prod_{\substack{0 \le j < n \\ j \ne i}} (x - x_j).$$

We compute first $A(x) = \prod_{j=0}^{n-1}(x - x_j)$ by n successive multiplications of a polynomial of degree less than n and a polynomial of degree 1; in total, this takes $O(n^2)$ operations. Then, we compute for $0 \le i < n$ the polynomial $L_i(x)$; using the fact that $L_i = \frac{A(x)}{x - x_i}$, each L_i can be found in $O(n)$ operations. Thus computing all L_i's costs $O(n^2)$ operations in total. Next, we evaluate each L_i at the point x_i, again at a cost of $O(n^2)$ operations in total. At this stage, we can compute $\frac{y_i}{L_i(x_i)} L_i(x)$ for all i, at a total cost of $O(n^2)$, and add these again at the same cost to obtain P.

Using the Lagrange formula, we have obtained an algorithm in $O(n^2)$ operations in $\mathbb{K}$ for interpolation. This is faster than solving the corresponding Vandermonde linear system using a general system solving algorithm! In other words, solving a Vandermonde system can be done faster than solving a general linear system.

Remark VI.1. *The Lagrange formula is a special instance of the Chinese Remainder Theorem in* $\mathbb{K}[x]$ *for solving the congruence system*

$$\begin{cases} P(x) & = y_0 \bmod (x - x_0) \\ & \vdots \\ P(x) & = y_{n-1} \bmod (x - x_{n-1}). \end{cases}$$

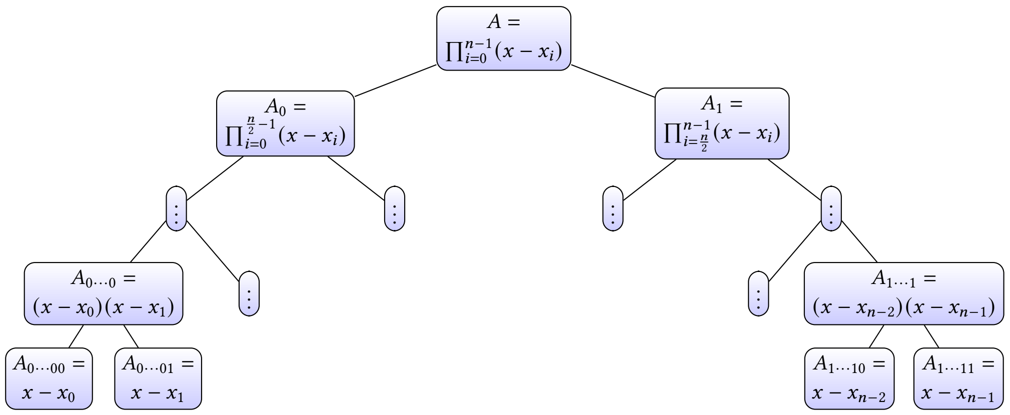
Indeed, recall that $P(x_0) = P(x) \bmod (x - x_0)$.

VI.2. Fast multipoint evaluation

Computing $y_i = P(x_i)$ for $0 \le i < n$ comes down to computing $P \bmod (x - x_i)$ for $0 \le i < n$. To do so, we proceed by a divide and conquer approaches. First, we compute $A = \prod_{i=0}^{n-1}(x - x_i)$ and a corresponding *subproducts tree*. Then we compute $P \bmod (x - x_i)$ for $0 \le i < n$ by exploiting the subproducts tree. For simplicity, below we assume that $n = 2^k$.

VI.2.1. Subproducts tree

We make the subproducts tree as follows, going up

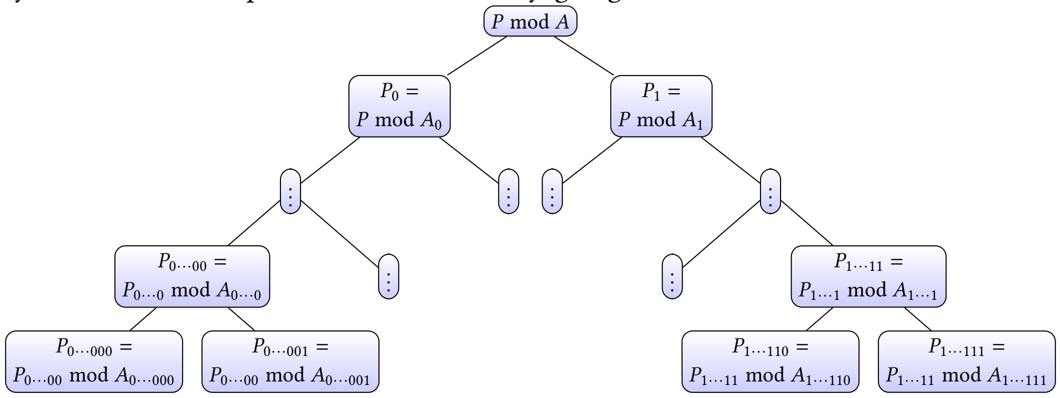

$A = \prod_{i=0}^{n-1}(x-x_i)$
$A_0 = \prod_{i=0}^{\frac{n}{2}-1}(x-x_i)$
$A_1 = \prod_{i=\frac{n}{2}}^{n-1}(x-x_i)$
$A_{0\cdots 0} = (x-x_0)(x-x_1)$
$A_{1\cdots 1} = (x-x_{n-2})(x-x_{n-1})$
$A_{0\cdots 00} = x-x_0$
$A_{0\cdots 01} = x-x_1$
$A_{1\cdots 10} = x-x_{n-2}$
$A_{1\cdots 11} = x-x_{n-1}$

Proposition VI.2. *Building the subproducts tree from its leaves to its root yields all the polynomials indicated in its nodes in a total of* $O(\mathsf{M}(n)\log(n))$ *operations in* $\mathbb{K}$.

Proof. Let $\mathsf{T}(n)$ denote the complexity of building the subproducts tree from n points $x_0, \ldots, x_{n-1}$. The divide and conquer approach builds the two subproducts trees for $n/2$ points $x_0, \ldots, x_{n/2-1}$ and $x_{n/2}, \ldots, x_{n-1}$, and gets the missing polynomial at the root by multiplying those at the roots of these two trees recursively computed. The complexity therefore satisfies $\mathsf{T}(n) \leq 2\mathsf{T}(\frac{n}{2}) + \mathsf{M}(\frac{n}{2})$, where $\mathsf{M}(\frac{n}{2})$ comes from the product of two polynomials of degree $\frac{n}{2}$. Unrolling this recurrence or using the master theorem concludes the proof. □

VI.2.2. Remainders tree

Now assume we have computed the above subproducts tree. To compute $P(x_0), \ldots, P(x_{n-1})$, which is the same as $P \bmod (x - x_0), \ldots, P \bmod (x - x_{n-1})$, we compute P modulo all the polynomials of the subproducts tree recursively, going down from the root.


$P \bmod A$
$P_0 = P \bmod A_0$
$P_1 = P \bmod A_1$
$P_{0\cdots 00} = P_{0\cdots 0} \bmod A_{0\cdots 0}$
$P_{1\cdots 11} = P_{1\cdots 1} \bmod A_{1\cdots 1}$
$P_{0\cdots 000} = P_{0\cdots 00} \bmod A_{0\cdots 000}$
$P_{0\cdots 001} = P_{0\cdots 00} \bmod A_{0\cdots 001}$
$P_{1\cdots 110} = P_{1\cdots 11} \bmod A_{1\cdots 110}$
$P_{1\cdots 111} = P_{1\cdots 11} \bmod A_{1\cdots 111}$

Proposition VI.3. *Recall that we assume* $\deg(P) < n$. *Having computed the subproducts tree, computing all the remainders in the remainders tree from its root to its leaves uses* $O(\mathsf{M}(n)\log(n))$ *operations in* $\mathbb{K}$.

Proof. The analysis is similar to the one for subproduct trees. Let $\mathsf{T}(n)$ denote the complexity of building the remainders tree from P of degree $< n$, the subproducts tree for $x_0, \ldots, x_{n-1}$ being known. Then $\mathsf{T}(n) \leq 2\mathsf{T}(\frac{n}{2}) + O(\mathsf{M}(\frac{n}{2}))$, where $O(\mathsf{M}(\frac{n}{2}))$ comes from performing the Euclidean division of a polynomial of degree n by a polynomial of degree $\frac{n}{2}$. Unrolling this recurrence or using the master theorem concludes the proof. □

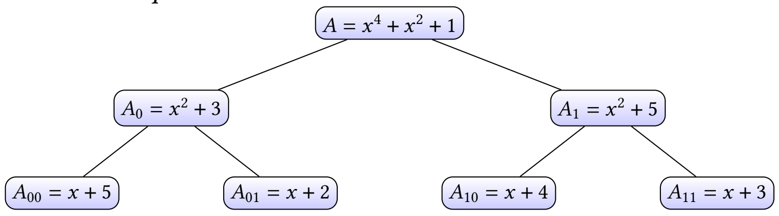
VI.2.3. Conclusion

```
Algorithm 6: Fast multipoint evaluation
Input: An integer $n > 0$, a polynomial $P \in \mathbb{K}[x]$ of degree $< n$, and $x_0, \ldots, x_{n-1}$ in $\mathbb{K}$.
Output: $P(x_0), \ldots, P(x_{n-1})$.
Compute the subproducts tree of $x_0, \ldots, x_{n-1}$.
Compute the remainders tree of $P$ with respect $x_0, \ldots, x_{n-1}$, using the subproducts tree.
Return the remainders at the leaves of the remainders tree.
```

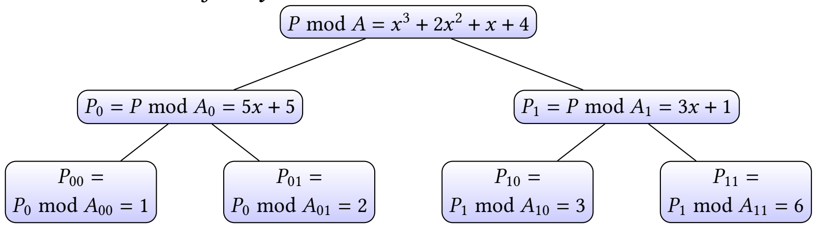
Theorem VI.4. *Fast polynomial multipoint evaluation can be performed in $O(\mathsf{M}(n)\log(n))$ operations in $\mathbb{K}$ using Algorithm 6.*

Proof. This is a consequence of the previous propositions. □

Example VI.5. *We wish to evaluate $P = x^3 + 2x^2 + x + 4 \in \mathbb{F}_7[x]$ in $x_0 = 2$, $x_1 = 5$, $x_2 = 3$ and $x_3 = 4$. The subproducts tree is*



The remainders of P by the successive moduli are


$P \bmod A = x^3 + 2x^2 + x + 4$
$P_0 = P \bmod A_0 = 5x + 5$
$P_1 = P \bmod A_1 = 3x + 1$
$P_{00} = P_0 \bmod A_{00} = 1$
$P_{01} = P_0 \bmod A_{01} = 2$
$P_{10} = P_1 \bmod A_{10} = 3$
$P_{11} = P_1 \bmod A_{11} = 6$

Hence, $P(2) = 1$, $P(5) = 2$, $P(3) = 3$ and $P(4) = 6$.

Problem VI.3. *Use the fast evaluation algorithm on $P = x^3 + x + \alpha$ and all the points of $\mathbb{F}_4 = \mathbb{F}_2[\alpha]/\langle \alpha^2 + \alpha + 1 \rangle$.*

Problem VI.4. *Until now we have assumed that $\deg(P) < n$, where n is the number of points. Here, we consider more generally the case where $\deg(P)$ may be arbitrary. Let $P \in \mathbb{K}[x]$ and $x_0, \ldots, x_{n-1} \in \mathbb{K}$. Let m be the degree of P. Propose an algorithm for multipoint evaluation of P at the x_i's and give complexity bounds which depends on m and n.*

Implementation VI.1. *Use SageMath, and use the provided file for binary trees in order to build subproducts tree. Implement first the algorithm which, given n points, constructs the corresponding subproducts tree. Implement two multipoint evaluation functions that take as input a polynomial $P \in \mathbb{F}_p[x]$ of degree at most $n-1$ and elements $x_0, \dots, x_{n-1} \in \mathbb{F}_p$, and return $P(x_0), \dots, P(x_{n-1})$. One will use Horner scheme, while the other uses Algorithm 6. Compare their performances.*

VI.3. Fast interpolation

Recall that, in this context, the points $x_0, \dots, x_{n-1}$ are pairwise distinct.

VI.3.1. Ideas

Again, we aim for a divide and conquer approach. Using notation from above (Section VI.1.2), the polynomial P is

$$P(x) = \sum_{i=0}^{n-1} (y_i/L_i(x_i))L_i(x) = A(x) \sum_{i=0}^{n-1} \frac{y_i/L_i(x_i)}{x - x_i}.$$

From these formulas we see that P is exactly the numerator of a sum of fractions of the form

$$S = \sum_{i=0}^{n-1} \frac{c_i}{x - x_i},$$

for some fixed elements $c_0, \dots, c_{n-1} \in \mathbb{K}$. Specifically here, $c_i = \frac{y_i}{L_i(x_i)}$. The first observation is that we can get these elements efficiently.

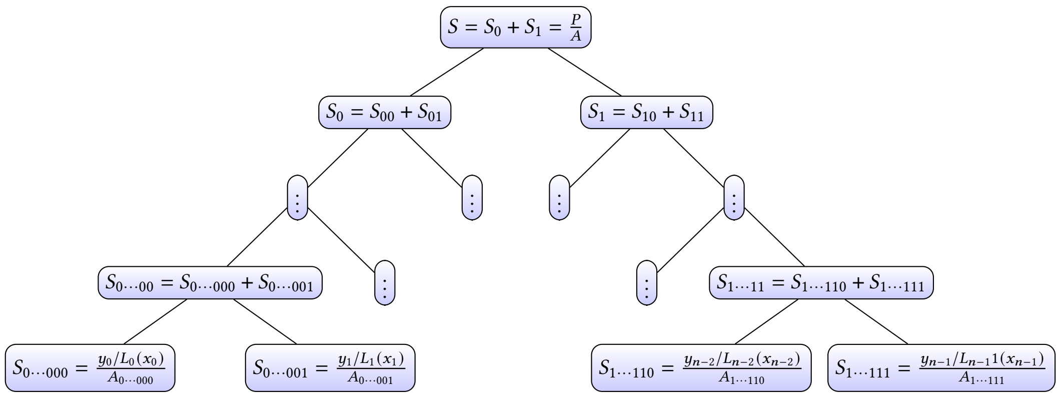
Proposition VI.6. *For all $0 \le i < n$, we have $L_i(x_i) = A'(x_i)$, where A' is the derivative of A.*

Problem VI.5. *Prove Proposition VI.6. Deduce the complexity of computing c_i for $0 \le i < n$.*

Having now both the x_i's and the c_i's, we are looking for the numerator of the fraction S, or more precisely, at the polynomial AS. We perform this computation recursively, setting $S_0 = \sum_{0 \le i < n/2} \frac{c_i}{x - x_i}$ and $S_1 = \sum_{n/2 \le i < n} \frac{c_i}{x - x_i}$, and returning $S = S_0 + S_1$. We observe that the common denominator of S_0 is A_0, and that of S_1 is A_1; recall that we already know that we can compute the whole subproducts tree in $O(\mathsf{M}(n)\log(n))$ operations.

VI.3.2. Partial sums tree

We perform the sum of fractions by divide and conquer, going up, using the fact that we know the denominators.


$S = S_0 + S_1 = \frac{P}{A}$
$S_0 = S_{00} + S_{01}$
$S_1 = S_{10} + S_{11}$
$S_{0\cdots 00} = S_{0\cdots 000} + S_{0\cdots 001}$
$S_{1\cdots 11} = S_{1\cdots 110} + S_{1\cdots 111}$
$S_{0\cdots 000} = \frac{y_0/L_0(x_0)}{A_{0\cdots 000}}$
$S_{0\cdots 001} = \frac{y_1/L_1(x_1)}{A_{0\cdots 001}}$
$S_{1\cdots 110} = \frac{y_{n-2}/L_{n-2}(x_{n-2})}{A_{1\cdots 110}}$
$S_{1\cdots 111} = \frac{y_{n-1}/L_{n-1}1(x_{n-1})}{A_{1\cdots 111}}$

Proposition VI.7. *Computing all the numerators in the partial sums tree from the leaves to the root uses $O(\mathsf{M}(n)\log(n))$ operations in $\mathbb{K}$.*

Proof. Again, the complexity $\mathsf{T}(n)$ for this task satisfies the recurrence $\mathsf{T}(n) \leq 2\mathsf{T}(\frac{n}{2}) + 2\mathsf{M}(\frac{n}{2})$, where $2\mathsf{M}(\frac{n}{2})$ comes from two products each of two polynomials of degree at most $\frac{n}{2}$. Unrolling this recurrence or using the master theorem concludes the proof. ☐

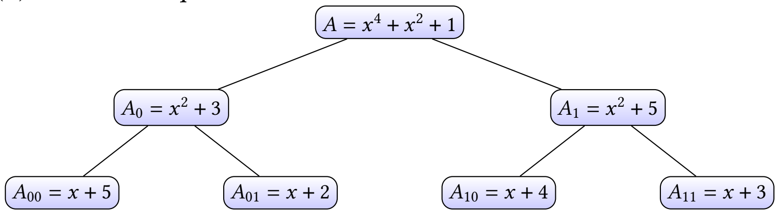
VI.3.3. Conclusion

```
Algorithm 7: Fast interpolation
Input: An integer $n > 0$; $x_0, \ldots, x_{n-1}$ pairwise distinct in $\mathbb{K}$; and $y_0, \ldots, y_{n-1}$ in $\mathbb{K}$.
Output: $P \in \mathbb{K}[x]$ of degree less than $n$ such that $P(x_0) = y_0, \ldots, P(x_{n-1}) = y_{n-1}$.
Compute the subproducts tree of $x_0, \ldots, x_{n-1}$ and set $A$ as the root of the tree.
Compute the remainders tree of $A'$ and $x_0, \ldots, x_{n-1}$.
Compute the partial sums tree of $\frac{y_0/A'(x_0)}{x-x_0}, \ldots, \frac{y_{n-1}/A'(x_{n-1})}{x-x_{n-1}}$.
Return the numerator of the root of the partial sums tree.
```

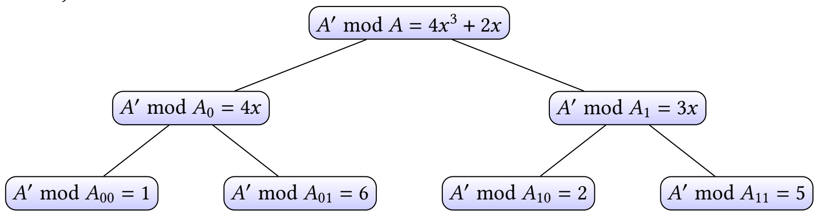
Theorem VI.8. *Polynomial interpolation can be performed in $O(\mathsf{M}(n)\log(n))$ operations in $\mathbb{K}$ using Algorithm 7.*

Proof. This is a consequence of the previous propositions. ☐

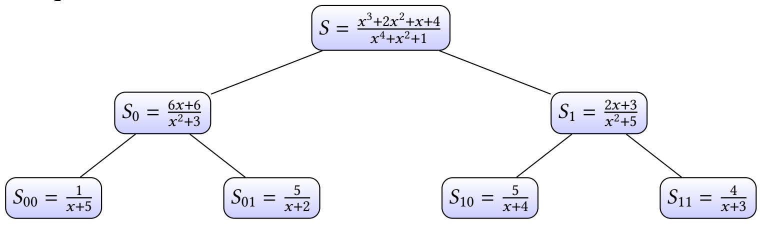
Example VI.9. *We want to interpolate $P \in \mathbb{F}_7[x]$ such that $P(2) = 1, P(5) = 2, P(3) = 3$ and $P(4) = 6$. The subproducts tree is*



Thus, $A' = 4x^3 + 2x$ and the remainders tree is



The numerators of the leaves of the partial sums tree are $\frac{1}{1} = 1, \frac{2}{6} = 5, \frac{3}{2} = 5, \frac{6}{5} = 4$. The partial sums tree is then,



Hence, $P = x^3 + 2x^2 + x + 4$.

Problem VI.6. *Show that in Algorithm 7, no 0 can appear in the leaves of the remainders tree.*

Problem VI.7. *We let $\mathbb{F}_9 = \mathbb{F}_3[\alpha]/\langle \alpha^2 + 1 \rangle$. Interpolate the polynomial of smallest degree $P \in \mathbb{F}_9[x]$ such that $P(1) = 0$, $P(\alpha) = \alpha$, $P(2\alpha) = 2\alpha$ and $P(2\alpha + 1) = 2$.*

Problem VI.8.

1. *Give the complexity of a naive algorithm computing the evaluation of $P = p_{n-1}x^{n-1} + \cdots + p_0$ and its $n - 1$ first derivatives $P', \ldots, P^{(n-1)}$ in a single point x_0.*

2. *Design an algorithm with quasi-linear complexity to solve this problem.*

Implementation VI.2. *Implement two interpolation functions that take as input pairwise distinct $x_0, \ldots, x_{n-1} \in \mathbb{F}_p$ and $y_0, \ldots, y_{n-1}\mathbb{F}_p$, and return $P \in \mathbb{F}_p[x]$ such that $P(x_0) = y_0, \ldots, P(x_{n-1}) = y_{n-1}$. One will use the Lagrange interpolation algorithm, and the other one will use Algorithm 7. Compare their performances.*

VI.4. Application: Shamir's secret sharing

Here we will assume our probability claims make sense for the fixed field $\mathbb{K}$. For simplicity, take for $\mathbb{K}$ a finite field, so that uniform distribution is possible.

This is one of the first cryptography schemes for sharing a secret S. This secret is divided into n shares $S_0, \ldots, S_{n-1}$ such that

1. knowing at least k shares makes S easy to compute;

2. knowing at most $k - 1$ shares makes S completely undetermined (more precisely: all the possible values are equiprobable).

This number k is called the threshold, and Shamir's secret sharing is an ideal and perfect (k, n)-threshold scheme. Observe that if $k = n$, then every share is needed to reconstruct the secret.

The idea is that the evaluations at exactly k distinct points are needed to reconstruct a polynomial of degree $k - 1$. To exploit this, we pick at random $k - 1$ elements $p_1, \dots, p_{k-1}$ in the field $\mathbb{K}$ and we set $p_0 = S$. We let $P = p_0 + p_1 x + \cdots + p_{k-1} x^{k-1}$ and for n distinct points $a_0, \dots, a_{n-1} \neq 0$, we build the shares $(a_i, P(a_i))$.

Given at least k shares, we can reconstruct P by interpolation, and thus S. However, given ℓ shares $(b_0, P(b_0)), \dots, (b_{\ell-1}, P(b_{\ell-1}))$ with $\ell < k$, we can only reconstruct a polynomial R of degree at most $\ell - 1$ that does not give any information on the constant term of P. Indeed, P and R coincide on $b_0, \dots, b_{\ell-1}$, hence there exists a polynomial Q of degree $k - \ell$ such that $P = Q \prod_{i=0}^{\ell-1}(x - b_i) + R$. The constant term of the right-hand side member is $q_0(-1)^\ell b_0 \cdots b_{\ell-1} + r_0$. Since none of the b_i's is 0 and q_0 is uniformly distributed in $\mathbb{K}$, then so is this term. Thus, no information on $p_0 = S$ can be extracted.

Problem VI.9. *An electronic vote is organized to elect the president of an association. Five poll counters are appointed and share the secret to access the results of the vote using Shamir's secret sharing scheme. Fruthermore, only one poll counter can be absent in order to access the results.*

Their shares are respectively $(1, 1)$, $(2, 1)$, $(3, 3)$, $(4, 6)$, $(5, 2)$ *in* $\mathbb{F}_7$.

1. *Give the parameters* (k, n) *of the sharing.*
2. *The first poll counter is not here, what is the secret to access the results of the vote?*

VII. Univariate and bivariate resultants

Source: this chapter is inspired from Section 6.2 of the book "Algorithmes Efficaces en Calcul Formel" (in French), written by Chyzak et al. (2018), freely available at:
https://hal.archives-ouvertes.fr/AECF/

In this course we will define the resultant and give basic algebraic properties of it and algorithms related to it. We will work with univariate polynomials in x over a ring $\mathcal{R}$, that is, elements of $\mathcal{R}[x]$. More precisely we will only consider:

- *univariate* resultants where the ring is a field $\mathcal{R} = \mathbb{K}$;
- *bivariate* resultants where the ring is the univariate polyomial $\mathcal{R} = \mathbb{K}[y]$.

Note that another similar and interesting case would be $\mathcal{R} = \mathbb{Z}$ (we will not consider it here). All these rings have a Euclidean division, are principal ideal domains, and have a fraction field (which of course is $\mathbb{K}$ itself when $\mathcal{R} = \mathbb{K}$).

We will see that univariate resultants are defined as the determinant of a *structured* matrix, are directly related to the GCD of two univariate polynomials, and can be computed by a Euclidean-like algorithm. For *bivariate* resultants, we will study the important application to the problem of solving bivariate polynomial systems: given two polynomials $F(x,y)$ and $G(x,y)$ in $\mathbb{K}[x,y]$, find all common roots $(\alpha,\beta) \in \mathbb{K}^2$ such that $F(\alpha,\beta) = G(\alpha,\beta) = 0$.

VII.1. Definition: Sylvester matrix, resultant

Let $A, B \in \mathcal{R}[x]$ be two polynomials,

$$A = a_m x^m + \cdots + a_0 \quad \text{and} \quad B = b_n x^n + \cdots + b_0$$

of degree respectively m and n (meaning that $a_m \neq 0$ and $b_n \neq 0$).

We define the *Sylvester matrix* of (A,B) as the following $(m+n)\times(m+n)$ matrix over $\mathcal{R}$:

$$\operatorname{Syl}(A,B) = \begin{bmatrix} a_m & \cdots & a_1 & a_0 & & \\ & \ddots & & & \ddots & \\ & & a_m & a_{m-1} & \cdots & a_0 \\ b_n & \cdots & b_1 & b_0 & & \\ & \ddots & & & \ddots & \\ & & b_n & b_{n-1} & \cdots & b_0 \end{bmatrix} \in \mathcal{R}^{(m+n)\times(m+n)}. \tag{VII.1}$$

Here the first n rows represent the coefficient vectors of the first n shifts of A, precisely: $x^{n-1}A, \dots, xA, A$. Similarly the first m rows represent the coefficient vectors of the first m shifts of B, that is, $x^{m-1}B, \dots, xB, B$.

Note that this matrix has a particular structure, and its $(m+n)^2$ entries are defined from only $m+n$ elements of $\mathcal{R}$, which are the coefficients of the polynomials A and B. Because of this structure, by construction, vector-matrix products of the form $W\,\mathrm{Syl}(A,B)$ for a vector $W \in \mathcal{R}^{1\times(m+1)}$ correspond to polynomial operations.

Let us make the last remark more explicit, as it will play a central role in the rest of the chapter. Let $U, V \in \mathcal{R}[x]$ be two polynomials such that $\deg(U) < \deg(B) = n$ and $\deg(V) < \deg(A) = m$. Then we can define the vector

$$W = \begin{bmatrix} u_{n-1} & \cdots & u_1 & u_0 & v_{m-1} & \cdots & v_1 & u_0 \end{bmatrix} \in \mathcal{R}^{1\times(m+1)}, \tag{VII.2}$$

where the u_i's and v_j's are the coefficients of U and V. The observation is that the vector-matrix product $W\,\mathrm{Syl}(A,B)$ gives the vector of coefficients of

$$(u_{n-1}x^{n-1}A + \cdots + u_1xA + u_0A) + (v_{m-1}x^{m-1}B + \cdots + v_1xB + v_0B) = AU + BV.$$

In other words, $\mathcal{R}$-linear combinations of the rows of $\mathrm{Syl}(A,B)$ allow us to represent all such polynomial combinations $AU + BV$, when we restrict to $\deg(U) < n$ and $\deg(V) < m$.

In particular, if $\mathcal{R} = \mathbb{K}$ then the Bézout relation tells us that one of these combinations yields the GCD of A and B, that is, $AU + BV = \gcd(A,B)$ for U and V with degrees as above. This GCD is the polynomial of smallest degree which can be obtained as such a polynomial combination. In terms of $\mathcal{R}$-linear algebra on the Sylvester matrix, this means that the vector of coefficients of the GCD can be retrieved as the last nonzero row in a row echelon form of $\mathrm{Syl}(A,B)$.

Example VII.1. *The following example has been discussed during the course using SageMath for various computations. Take the field of rational numbers $\mathcal{R} = \mathbb{K} = \mathbb{Q}$ and the two polynomials*

$$A = x^4 - x^3 - 7x^2 + 2x + 3 \quad \textit{and} \quad B = x^3 - 4x^2 + 2x + 3.$$

Then the Sylvester matrix of (A,B) is

$$\mathrm{Syl}(A,B) = \begin{bmatrix} 1 & -1 & -7 & 2 & 3 & 0 & 0 \\ 0 & 1 & -1 & -7 & 2 & 3 & 0 \\ 0 & 0 & 1 & -1 & -7 & 2 & 3 \\ 1 & -4 & 2 & 3 & 0 & 0 & 0 \\ 0 & 1 & -4 & 2 & 3 & 0 & 0 \\ 0 & 0 & 1 & -4 & 2 & 3 & 0 \\ 0 & 0 & 0 & 1 & -4 & 2 & 3 \end{bmatrix} \in \mathbb{K}^{7\times7},$$

and one row echelon form of it (in fact, the unique reduced one) is

$$E = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -729 \\ 0 & 1 & 0 & 0 & 0 & 0 & -243 \\ 0 & 0 & 1 & 0 & 0 & 0 & -81 \\ 0 & 0 & 0 & 1 & 0 & 0 & -27 \\ 0 & 0 & 0 & 0 & 1 & 0 & -9 \\ 0 & 0 & 0 & 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \in \mathbb{K}^{7\times 7}.$$

In particular, this echelon form E shows that $\mathrm{Syl}(A, B)$ *is singular (its determinant is zero, it is not invertible), since E has a zero row. Furthermore we observe that the last nonzero row* $[0 \ \cdots \ 0 \ 1 \ -3]$ *gives the coefficients of* $\gcd(A, B) = x - 3$.

Example VII.2. *The following example has also been discussed during the course using SageMath for various computations; the polynomials were chosen "at random". Take the field of rational numbers $\mathcal{R} = \mathbb{K} = \mathbb{Q}$ and the two polynomials*

$$A = 6x^4 - x^3 + 3x^2 - 2x - 6 \quad \text{and} \quad B = x^3 - \frac{1}{9}x^2 + \frac{19}{2}.$$

Then the Sylvester matrix of (A, B) is

$$\mathrm{Syl}(A, B) = \begin{bmatrix} 6 & -1 & 3 & -2 & -6 & 0 & 0 \\ 0 & 6 & -1 & 3 & -2 & -6 & 0 \\ 0 & 0 & 6 & -1 & 3 & -2 & -6 \\ 1 & -\frac{1}{9} & 0 & \frac{19}{2} & 0 & 0 & 0 \\ 0 & 1 & -\frac{1}{9} & 0 & \frac{19}{2} & 0 & 0 \\ 0 & 0 & 1 & -\frac{1}{9} & 0 & \frac{19}{2} & 0 \\ 0 & 0 & 0 & 1 & -\frac{1}{9} & 0 & \frac{19}{2} \end{bmatrix} \in \mathbb{K}^{7\times 7},$$

and one row echelon form of it (in fact, the unique reduced one) is the identity matrix

$$E = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix} \in \mathbb{K}^{7\times 7}.$$

In particular, this echelon form E shows that $\mathrm{Syl}(A, B)$ *is invertible, since E is itself invertible. Furthermore we observe that the last nonzero row* $[0 \ \cdots \ 0 \ 1]$ *represents* $\gcd(A, B) = 1$.

Playing with SageMath, one interesting remark is that the determinant of $\mathrm{Syl}(A, B)$ *is* $\frac{3803532689}{1944}$*, and we ask SageMath to compute the "resultant" (which we have not defined yet!) of A and B, we get* $\frac{3803532689}{1944}$ *as well.*

The last remark leads us to the next definition.

Definition VII.3. *Given nonzero polynomials A and B in $\mathcal{R}[x]$, the resultant of (A, B) is the determinant of the Sylvester matrix* $\mathrm{Syl}(A, B)$. *We denote it by* $\mathrm{Res}_x(A, B)$. *It is an element of $\mathcal{R}$.*

Note that the order matters: $\mathrm{Res}_x(A, B)$ is not necessarily equal to $\mathrm{Res}_x(B, A)$, this depends on the parity of the degrees m and n. Theorem VII.5 below will make this more explicit.

In this course, until now we have mostly considered polynomials and matrices whose coefficients are in a field. Here, we consider more generally polynomials and matrices whose coefficients are in a "nice" ring $\mathcal{R}$ (see the comment on this at the beginning of this chapter). Note that all the usual properties of the determinant still work for matrices over $\mathcal{R}$: the determinant of the identity matrix is 1, we can expand with respect to a column or a row, it does not change when we perform "elementary operations" on the rows or the columns of the matrix, we get the opposite determinant when we swap two columns or two rows, it can be expressed as the sum over all permutations, etc.

Before continuing with general properties, we first state a basic fact which is a consequence of what has been written above in the case $\mathcal{R} = \mathbb{K}$.

Lemma VII.4. *Here, $\mathcal{R} = \mathbb{K}$. Let A and B be nonzero polynomials in $\mathbb{K}[x]$. Then, A and B are coprime if and only if* $\mathrm{Res}_x(A, B) \neq 0$.

Proof. We could rely on the remarks above on the echelon form of $\mathrm{Syl}(A, B)$, but since we have not proved these remarks rigorously, we will give a different proof. We will use the above link between vector-matrix products $W \,\mathrm{Syl}(A, B)$ and the combinations $AU + BV$ for U of degree $< n$ and V of degree $< m$.

First assume A and B are coprime. We want to prove that $\mathrm{Res}_x(A, B) \neq 0$, which is equivalent to say that $\mathrm{Syl}(A, B)$ is an invertible matrix. For this, it suffices to show that the homogeneous linear system $W \,\mathrm{Syl}(A, B) = 0$ (where $W \in \mathbb{K}^{1\times(m+n)}$ is the unknown vector) has no solution other than $W = 0$. Let $W \in \mathbb{K}^{1\times(m+n)}$ be such that $W \,\mathrm{Syl}(A, B) = 0$, and write W as the concatenation of the coefficients of two polynomials U and V, as in Eq. (VII.2). Then $AU + BV = 0$. This implies that A divides BV, but since A is coprime with B, then A divides V. By construction $\deg(V) < m = \deg(A)$, so the only possibility is $V = 0$. Then, $AU = 0$, with $A \neq 0$, so $U = 0$ as well. This shows $W = 0$, hence $\mathrm{Syl}(A, B)$ is invertible.

Now assume $\mathrm{Res}_x(A, B) \neq 0$, which is equivalent to say that $\mathrm{Syl}(A, B)$ is invertible. We want to show that A and B are coprime. The polynomial 1 is represented by the vector $[0 \ \cdots \ 0 \ 1] \in \mathbb{K}^{1\times(m+n)}$, and since $\mathrm{Syl}(A, B)$ is invertible there is a unique solution $W \in \mathbb{K}^{1\times(m+n)}$ to the linear system $W \,\mathrm{Syl}(A, B) = [0 \ \cdots \ 0 \ 1]$ (explicitly, this solution is $W = [0 \ \cdots \ 0 \ 1] \,\mathrm{Syl}(A, B)^{-1}$). Writing as above W as the concatenation of the coefficients of two polynomials U and V, we obtain that $AU + BV = 1$. Thus A and B are coprime. □

VII.2. Properties of the resultant

Theorem VII.5 (Poisson's formula)**.** *Assume that the polynomials A and B in $\mathcal{R}[x]$ factorize as*

$$A = a(x - \alpha_1) \cdots (x - \alpha_m) \quad \text{and} \quad B = b(x - \beta_1) \cdots (x - \beta_n),$$

where $a, b, \alpha_i, \beta_j \in \mathcal{R}$. Then the resultant of (A, B) is

$$\begin{aligned}\operatorname{Res}_x(A, B) &= a^n b^m \prod_{i,j}(\alpha_i - \beta_j)\\ &= (-1)^{mn} b^m \prod_{1\le j\le n} A(\beta_j) = a^n \prod_{1\le i\le m} B(\alpha_i) = (-1)^{mn} \operatorname{Res}_x(B, A).\end{aligned}$$

We will not provide a proof of this result here. But we will study important consequences.

Corollary VII.6. *The resultant is* multiplicative*: for nonzero polynomials A, B, C in $\mathcal{R}[x]$,*

$$\operatorname{Res}_x(AB, C) = \operatorname{Res}_x(A, C) \operatorname{Res}_x(B, C).$$

The proof of this corollary is rather direct from Theorem VII.5. To make it possible to apply this theorem, one thing to notice is that, for the rings $\mathcal{R}$ we consider here, there is an associated fraction field, and we can see A, B, C as polynomials whose coefficients are in the algebraic closure of this fraction field. This ensures that these polynomials can be written in the completely factorized form in Theorem VII.5. Since the resultant does not change when we do this embedding (because of the definition as the determinant of the Sylvester matrix), we can then apply Theorem VII.5 over this algebraic closure to prove the result over $\mathcal{R}$.

The following result shows that, in the general case over $\mathcal{R}$, the resultant is a polynomial combination of A and B by polynomials whose coefficients are in the ring $\mathcal{R}$.

Proposition VII.7. *For nonzero polynomials A and B in $\mathcal{R}[x]$, there exist U and V in $\mathcal{R}[x]$ such that $\operatorname{Res}_x(A, B) = AU + BV$ and $\deg(U) < n = \deg(B)$ and $\deg(V) < m = \deg(A)$.*

Proof. If $\operatorname{Res}_x(A, B) = 0$, it suffices to take $U = V = 0$. Otherwise, the matrix $\operatorname{Syl}(A, B)$ is nonsingular (it has an inverse when considered over the fraction field). We have the classical inversion formula

$$\operatorname{adj}(\operatorname{Syl}(A, B)) \cdot \operatorname{Syl}(A, B) = \det(\operatorname{Syl}(A, B))\, I_{m+n} = \operatorname{Res}_x(A, B)\, I_{m+n}.$$

Here I_{m+n} is the identity matrix of size $(m + n) \times (m + n)$, and we denote by $\operatorname{adj}(\operatorname{Syl}(A, B))$ the adjugate of the Sylvester matrix, which is the transpose of its cofactor matrix. The entries of this matrix are essentially determinants of submatrices of $\operatorname{Syl}(A, B)$, which are thus in the ring $\mathcal{R}$. So $\operatorname{adj}(\operatorname{Syl}(A, B))$ is in $\mathcal{R}^{(m+n)\times(m+n)}$.

Then, denoting by $W \in \mathcal{R}^{1\times(m+n)}$ the last row of $\operatorname{adj}(\operatorname{Syl}(A, B))$, considering the last row in the above equation gives

$$W \cdot \operatorname{Syl}(A, B) = [0 \ \ \cdots \ \ 0 \ \ \operatorname{Res}_x(A, B)].$$

It remains to interpret this in terms of polynomials as we have done several times above, and we obtain $AU + BV = \operatorname{Res}_x(A, B)$ with the sought degree constraints on U and V. ◻

Here is now an important remark showing that the resultant does not always agree with specialization (in the example below, “specialization” is replacing a by 0).

Example VII.8. *For $a, b, c, d, e \in \mathcal{R}$, the resultant of ax^2+bx+c and $dx+e$ is ae^2+cd^2-bde, whereas the resultant of $bx+c$ and $dx+e$ is $be-cd$. Note that, when $a=0$, $ae^2+cd^2-bde = d(cd-be)$.*

Still, the following shows various conditions under which specialization works well with resultants (note: the next remark and example have not been presented in class).

Remark VII.9. *Let $\mathcal{R}$ and $\mathcal{R}'$ be two commutative rings, and $\varphi : \mathcal{R} \longrightarrow \mathcal{R}'$ be a ring homomorphism. We extend φ into a polynomial ring homomorphism $\varphi : \mathcal{R}[x] \longrightarrow \mathcal{R}'[x]$, in the natural way by setting $\varphi(x) = x$. Let A and B be nonzero polynomials in $\mathcal{R}[x]$, of respective degrees m and n.*

- *If $\deg(\varphi(A)) = m$ and $\deg(\varphi(B)) = n$, then*
$$\varphi(\mathrm{Res}_x(A,B)) = \mathrm{Res}_x(\varphi(A), \varphi(B)).$$
- *If $\deg(\varphi(A)) = m$ and $\deg(\varphi(B)) = n' < n$, then*
$$\varphi(\mathrm{Res}_x(A,B)) = \varphi(f_m)^{n-n'}\,\mathrm{Res}_x(\varphi(A), \varphi(B)).$$
- *If $\deg(\varphi(A)) = m' < m$ and $\deg(\varphi(B)) = n$, then*
$$\varphi(\mathrm{Res}_x(A,B)) = (-1)^{(m-m')n}\varphi(g_n)^{m-m'}\,\mathrm{Res}_x(\varphi(A), \varphi(B)).$$
- *If $\deg(\varphi(A)) < m$ and $\deg(\varphi(B)) < n$, then $\varphi(\mathrm{Res}_x(A,B)) = 0$ but nothing can be said in general about $\mathrm{Res}_x(\varphi(A), \varphi(B))$.*

Example VII.10. *Note: in this example, we take the integer ring $\mathcal{R} = \mathbb{Z}$.*

1. *Let $A = 3x^3-2x^2+7x+1$, $B = 5x^2-x+1$ and $\mathcal{R} = \mathbb{Z}$. If $S = \mathbb{Z}/3\mathbb{Z}$, then $\varphi(A) = x^2+x+1$, $\varphi(B) = 2x^2+2x+1$. We have $\varphi(\mathrm{Res}_x(A,B)) = \varphi(1337) = 2$ and*
$$\varphi(g_2)^{3-2}\,\mathrm{Res}_x(\varphi(A), \varphi(B)) = 2\det\left(\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 2 & 2 & 1 & 0 \\ 0 & 2 & 2 & 1 \end{bmatrix}\right) = 2.$$
2. *Let $A = 4x^2-1$, $B = 2x^2-1$ and $\mathcal{R} = \mathbb{Z}$, then the roots of A (in the algebraic closure $\mathbb{C}$ of the fraction field $\mathbb{Q}$ of $\mathbb{Z}$) are $\pm\frac{1}{2}$, while the roots of B are $\pm\frac{\sqrt{2}}{2}$. Thus $\mathrm{Res}_x(A,B) = 2^2 4^2 \left(\frac{1-\sqrt{2}}{2}\right)\left(\frac{1+\sqrt{2}}{2}\right)\left(\frac{-1-\sqrt{2}}{2}\right)\left(\frac{-1+\sqrt{2}}{2}\right) = 4$.*

   If $S = \mathbb{Z}/2\mathbb{Z}$, then $\varphi(\mathrm{Res}_x(A,B)) = \varphi(4) = 0$ but $\mathrm{Res}_x(\varphi(A), \varphi(B)) = \mathrm{Res}_x(1,1) = 1$.

VII.3. Univariate resultant: algorithms

We consider the case of the univariate resultant, $\mathcal{R} = \mathbb{K}$.

Following the definition, the naive way to compute $\mathrm{Res}_x(A,B)$ is by linear algebra, to find the determinant of the Sylvester matrix. Using "slow" Gaussian elimination, this costs $O((m+n)^3))$

operations in $\mathbb{K}$. Faster algorithms reduce the determinant to matrix multiplication, with a cost of $O((m+n)^{2.81})$ field operations with Strassen's multiplication, or $O((m+n)^{2.38})$ with the current best known algorithms.

This linear algebra approach does not exploit at all the specific structure of the Sylvester matrix, mentioned above. We have seen strong links between operations on this matrix and operations in $\mathbb{K}[x]$. For example, using linear algebra, computing the vector-matrix product $W\,\mathrm{Syl}(A,B)$ uses $O((m+n)^2)$ operations in $\mathbb{K}$. However, we have seen that this can be rewritten as a combination of polynomials of the form $AU+BV$: using this interpretation, this vector-matrix product can be computed much faster, in time quasi-linear in $m+n$ (specifically, in $O((m+n)\log(m+n)\log\log(m+n))$ operations in $\mathbb{K}$).

The next result gives the basic property showing that the resultant can be computed in a way very similar to the GCD via the quadratic Euclidean algorithm (indeed, using the same notation as below, this algorithm is based on the formula $\gcd(A,B)=\gcd(B,R)$).

Theorem VII.11. *Let A and B be nonzero polynomials in $\mathbb{K}[x]$. Let Q and R be polynomials in $\mathbb{K}[x]$ such that $A=BQ+R$ and $r=\deg(R)<\deg(B)=n$. Then,*

$$\mathrm{Res}_x(A,B) = (-1)^{mn} b_n^{m-r}\,\mathrm{Res}_x(B,R),$$

where b_n is the leading coefficient of B.

This leads to a Euclid-like algorithm for computing the resultant of two univariate polynomials over $\mathbb{K}$. This algorithm has complexity $O((m+n)^2)$, which is better than the above linear algebra approach.

Problem VII.1. *Describe the algorithm in detail, and prove the above claim about its complexity.*

Remark: using fast GCD algorithms, based on quasi-linear polynomial multiplication, it has been showed that one can obtain an algorithm for computing the univariate resultant in time quasi-linear in $m+n$.

VII.4. Bivariate resultant: algorithms

We now consider the case of the bivariate resultant: $\mathcal{R}=\mathbb{K}[y]$. This means that the coefficients of A and B are themselves univariate polynomials in y, and the Sylvester matrix is an $(m+n)\times(m+n)$ matrix with entries in $\mathbb{K}[y]$. For such matrices, computing the determinant by usual linear algebra techniques is not appropriate. Indeed, since $\mathbb{K}[y]$ is not a field, Gaussian elimination is not well-suited to this context: we are not allowed to divide by a non-constant polynomial (otherwise, we get fractions which are not in $\mathbb{K}[y]$ anymore). Even if we accepted to work over the field of fractions $\mathbb{K}(y)$, the computations using usual Gaussian elimination quickly lead to numerators and denominators having very large degrees, making this prohibitive in terms of complexity.

Instead, the classical approach is to compute the determinant of a matrix over $\mathbb{K}[y]$ using *evaluation-interpolation*. This is studied in the next exercise, in general for a square matrix over $\mathbb{K}[y]$.

Problem VII.2. *Let $S \in \mathbb{K}[y]^{m\times m}$ be a matrix whose entries are in $\mathbb{K}[y]$. Let $d \in \mathbb{N}$ be an upper bound on their degree: all entries of S have degree at most d. To simplify matters, we assume the field is "large enough": if we need to pick k distinct points in $\mathbb{K}$, we assume this is feasible.*

1. *Give an upper bound on the degree of* $\det(S)$.
2. *Is this bound tight (i.e. is there a matrix S for which is is reached)?*
3. *Suppose you are given n pairwise distinct points $\alpha_1, \dots, \alpha_n$ in $\mathbb{K}$, with $n > d$. How much does it cost to compute $S(\alpha_k)$ for each $1 \le k \le n$?*
4. *Now suppose $n > \det(S)$, and you already know the evaluation $\beta_k = \det(S)(\alpha_k)$ for each $1 \le k \le n$. How much does it cost to compute $\det(S)$?*
5. *Give a complete algorithm for computing $\det(S)$, and describe its complexity.*

VII.5. Bivariate resultant: solving bivariate polynomial systems

Let A and B be two polynomials in $\mathbb{K}[x, y]$. Here we will assume that the field $\mathbb{K}$ is algebraically closed. We want to find all points $(\alpha, \beta) \in \mathbb{K}^2$ such that $A(\alpha, \beta) = B(\alpha, \beta) = 0$. Geometrically, each of the equations $A = 0$ and $B = 0$ describes a curve in the plane $\mathbb{K}^2$. Solving the system $S : A = B = 0$ is equivalent to finding the points at the intersection of both curves.

Similarly to Gaussian elimination, the main idea is to "eliminate" a variable, thus reducing the problem to finding roots of univariate polynomials.

1. We consider the system S_x, which is the same as S except that A and B are seen as univariate polynomials in x over $\mathcal{R} = \mathbb{K}[y]$, that is, $A, B \in \mathbb{K}[y][x]$. Solving S_x means finding the common "x roots" of both polynomials.
2. We know that A and B have a common root if their resultant is 0; this resultant is a polynomial in y, indeed we have seen $\mathrm{Res}_x(A, B) \in \mathcal{R} = \mathbb{K}[y]$. (Here, we see the importance of x in the notation $\mathrm{Res}_x(A, B)$: it indicates we eliminate the variable x.)
3. Since $\mathrm{Res}_x(A, B) = AU + BV$ for some $U, V \in \mathcal{R}[x]$, if (α_0, β_0) is a solution of S, then $\mathrm{Res}_x(A, B)$ evaluated at $y = \beta_0$ vanishes. In other words, $A(x, \beta_0)$ and $B(x, \beta_0)$, seen as polynomials in x, have at least one common root α_0. Thus, their resultant is 0 and β_0 is a zero of $\mathrm{Res}_x(A, B)$.

   Beware: $\mathrm{Res}_x(A, B)$ can vanish at some β_{-1} without the existence of a solution $(\alpha_{-1}, \beta_{-1})$ of the system S! Such a root of $\mathrm{Res}_x(A, B)$ is called a parasite solution, or extra solution.

4. For each root β_i of $\mathrm{Res}_x(A, B)$, we find the list of α_i's such that $A(\alpha_i, \beta_i) = B(\alpha_i, \beta_i) = 0$.

Problem VII.3. *Here we consider the base field $\mathbb{K} = \mathbb{Q}$ (and its algebraic closure $\mathbb{C}$). We want to solve the system given by*

$$S = \begin{cases} A = -xy^2 + x^2 - x = 0 \\ B = x^3y^2 - x^2y^2 - 4x + 4 = 0 \end{cases} .$$

1. *Describe the curves defined by A and B.*
2. *Show that $\mathrm{Res}_y(A, B) = x^2(x-1)^2(x-2)^2(x^2+x+2)^2$ (note this is the resultant with respect to y!).*
3. *Give the roots of $\mathrm{Res}_y(A, B)$.*
4. *For each rational root α_i of $\mathrm{Res}_y(A, B)$, give all possible β_i such that $A(\alpha_i, \beta_i) = B(\alpha_i, \beta_i) = 0$.*
5. *Conclude on the rational solutions of the system S.*

VIII. Structured linear algebra

Source: this chapter is strongly inspired from Chapter 10 of the book "Algorithmes Efficaces en Calcul Formel" (in French), written by Chyzak et al. (2018), freely available at: https://hal.archives-ouvertes.fr/AECF/

Matrices with a specific structure are ubiquitous in computer algebra. These are matrices whose elements are repeated following certain patterns, or satisfy some types of relations. More formally, an $n \times n$ structured matrix may typically be described from only $O(n)$ field elements, instead of n^2 for a dense matrix, and it can be multiplied by a vector in $\tilde{O}(n)$ arithmetic operations instead of $O(n^2)$ for a dense matrix. This chapter presents a unified algorithm framework to handle matrices with a structure of type Toeplitz, Hankel, Vandermonde, Sylvester, and Cauchy. Computations with matrices of these types are directly related to computations with polynomials, which allows us to leverage fast polynomial multiplication to accelerate operations with these matrices. For example, a linear system defined by such a matrix, assumed invertible, can be solved in $\tilde{O}(n)$ operations in the base field.

VIII.1. Introduction and definitions

We have seen that dense $n \times n$ matrices with entries in a field $\mathbb{K}$ can be manipulated in $O(n^\omega)$ operations in $\mathbb{K}$, where ω is a real number between 2 and 3. By "manipulated" we refer to multiplication, inversion, Gaussian elimination, determinant computation, linear system solving, etc. In some situations, these matrices exhibit some structure that generic algorithms try neither to detect nor to exploit. Here are some common examples of such matrix structures.

Definition VIII.1. *A matrix $A \in \mathbb{K}^{n\times n}$ is said to be a* Toeplitz *matrix if it is invariant along the diagonals, meaning that its entries $a_{i,j}$ satisfy $a_{i,j} = a_{i+k,j+k}$ for all k.*

Remark that a Toeplitz matrix is entirely described by its first row and its first column, that is, by only $2n-1$ coefficients from $\mathbb{K}$.

Definition VIII.2. *A matrix $A \in \mathbb{K}^{n\times n}$ is said to be a* Hankel *matrix if it is invariant along the antidiagonals, meaning that its entries $a_{i,j}$ satisfy $a_{i,j} = a_{i-k,j+k}$ for all k.*

Thus, a Hankel matrix is entirely described by its first row and its last column, that is, again by only $2n-1$ coefficients from $\mathbb{K}$.

Example VIII.3. *The matrix (in the canonical bases) of the operator of multiplication by a given univariate polynomial, in fixed degree, is a Toeplitz matrix. For example, multiplying a polynomial*

$b(x) \in \mathbb{K}[x]$ of degree at most 2 by the given polynomial $a_0 + a_1x + a_2x^2$ in $\mathbb{K}[x]$ translates, in matrix terms, as

$$\begin{pmatrix} a_0 & 0 & 0 \\ a_1 & a_0 & 0 \\ a_2 & a_1 & a_0 \\ 0 & a_2 & a_1 \\ 0 & 0 & a_2 \end{pmatrix} \times \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} a_0b_0 \\ a_0b_1 + a_1b_0 \\ a_0b_2 + a_1b_1 + a_2b_0 \\ a_1b_2 + a_2b_1 \\ a_2b_2 \end{pmatrix}.$$

In the above example, the Toeplitz matrix has a particular shape, called *striped*. In fact, any Toeplitz matrix can be seen as a submatrix of a striped Toeplitz matrix, which leads to the following result.

Lemma VIII.4. *Multiplying a Toeplitz matrix (or a Hankel matrix) in $\mathbb{K}^{n\times n}$ by a vector in $\mathbb{K}^{n\times 1}$ can be done in $O(\mathsf{M}(n))$ operations in $\mathbb{K}$.*

Proof. For each $0 \le i \le n-1$, the entry c_i of the matrix-vector product

$$\begin{pmatrix} a_{n-1} & \cdots & a_0 \\ \vdots & \ddots & \vdots \\ a_{2n-2} & \cdots & a_{n-1} \end{pmatrix} \times \begin{pmatrix} b_0 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} c_0 \\ \vdots \\ c_{n-1} \end{pmatrix}$$

is the coefficient of x^{n-1+i} in the product of polynomials

$$(a_0 + \cdots + a_{2n-2}x^{2n-2}) \times (b_0 + \cdots + b_{n-1}x^{n-1}). \tag{VIII.1}$$

The proof is identical for a Hankel matrix. □

Remark VIII.5. *The above proof shows that $2\mathsf{M}(n) + O(n)$ operations in $\mathbb{K}$ suffice to multiply a $n\times n$ Toeplitz or Hankel matrix by a vector. This complexity bound can be decreased to $\mathsf{M}(n)+O(n)$ thanks to the observation that only the middle part of the polynomial product in (VIII.1) is needed, and using for this task the specific* middle product *algorithm.*

Example VIII.6. *A Sylvester matrix, as in Eq. (VII.1), is the concatenation of two striped Toeplitz matrices.*

Here are two other important families of structured matrices.

Definition VIII.7. *A matrix $A = (a_{i,j})_{i,j=0}^{n-1}$ in $\mathbb{K}^{n\times n}$ is a* Vandermonde *matrix if its entries can be written $a_{i,j} = a_i^j$ for $a_0, \dots, a_{n-1} \in \mathbb{K}$.*

Definition VIII.8. *A matrix $A = (a_{i,j})_{i,j=0}^{n-1}$ in $\mathbb{K}^{n\times n}$ is a* Cauchy *matrix if its entries can be written $a_{i,j} = 1/(a_i - b_j)$ for $a_i, b_j \in \mathbb{K}$, with $a_i \ne b_j$ for all i, j.*

One can observe a number of common features between these examples. The matrix can be represented by only $O(n)$ elements from the field $\mathbb{K}$; a matrix-vector product can be performed more efficiently (in time softly linear in n, see below) than for arbitrary dense matrices; the matrix-matrix product with another $n \times n$ matrix can be performed more efficiently (in time softly quadratic in n, via n matrix-vector products) than for arbitrary dense matrices. In short, matrix-vector and matrix-matrix products can be performed in time quasi-optimal in the size of the output, using FFT-based polynomial multiplication.

Indeed, in all above cases the matrix-vector Av has a polynomial interpretation:

- polynomial multiplication for Toeplitz, Hankel, and Sylvester matrices;
- multipoint evaluation of polynomials for Vandermonde matrices;
- for Cauchy matrices, multipoint evaluation of rational fractions in $\mathbb{K}(x)$ of the form $\sum_{j=0}^{n-1} c_j/(x-b_j)$.

Problem VIII.1. *Show that a matrix-vector product with a Cauchy matrix, as follows:*

$$Av = \begin{bmatrix} \frac{1}{a_0-b_0} & \cdots & \frac{1}{a_0-b_{n-1}} \\ \vdots & & \vdots \\ \frac{1}{a_{n-1}-b_0} & \cdots & \frac{1}{a_{n-1}-b_{n-1}} \end{bmatrix} \begin{bmatrix} v_0 \\ \vdots \\ v_{n-1} \end{bmatrix},$$

corresponds to the multipoint evaluation of some rational fraction that you will describe. Give an algorithm and complexity bound for computing this matrix-vector product.

Observing these examples, one could be tempted to define a *structured* matrix as one such that a matrix-vector product can be performed in $\tilde{O}(n)$ operations. Yet, this definition alone is difficult to exploit for further algorithms: as a natural question, would this notion of structure help to efficiently solve the linear system $Ax = b$? In the above examples, this system solving also has a polynomial interpretation:

i. guessing linear recurrences with constant coefficients, for Toeplitz or Hankel matrices;

ii. interpolation of polynomials, for Vandermonde matrices;

iii. interpolation of rational fractions, for Cauchy matrices.

Problem VIII.2. *Assume that a sequence $(a_n)_n$ in $\mathbb{K}^{\mathbb{N}}$ satisfies an (unknown) linear recurrence equation with constant coefficients, of the form*

$$a_{n+d} = p_{d-1}a_{n+d-1} + \cdots + p_0 a_n, \qquad n \geq 0.$$

Show that finding the coefficients p_i in this recurrence equation is the same as solving a Hankel linear system.

Another encouraging point is that, for these three operations, we have quasi-optimal algorithms with complexity $O(\mathsf{M}(n)\log(n))$.

Yet, in general, if A is an invertible matrix such that the matrix-vector product Av can be computed in say L operations in $\mathbb{K}$, there is no known way to deduce the existence of an algorithm with complexity $O(L)$ for computing $A^{-1}v$ (i.e., to solve a linear system). For example, for a sparse matrix A with $O(n)$ nonzero entries, a matrix-vector product can be computed in $O(n)$ operations, yet the fastest known algorithm for linear system solving with A is the Wiedemann algorithm, whose complexity is quadratic in n.

Another negative point is that the above types of structured matrices are not stable by inversion: the inverse of a Toeplitz (resp. Vandermonde) matrix is not a Toeplitz (resp. Vandermonde) matrix. Therefore, this first attempt at a notion of "structured matrix" lacks some more hypotheses.

The notion that is now classical is based on a generalization of the invariance along the diagonals in Toeplitz matrices. It comes from the following observation: if A is a Toeplitz matrix, then the "displaced" matrix

$$\phi(A) = A - (A \text{ shifted by 1 to the bottom and to the right})$$

has a square submatrix of dimensions $(n-1)\times(n-1)$ with zero entries. In particular this matrix has rank at most 2. We say that ϕ is a *displacement operator*, and that A has a *displacement rank* (with respect to ϕ) which is at most 2.

Therefore we can represent $\phi(A)$ in a concise form, as a product $G \cdot {}^tH$, with G and H rectangular matrices of dimensions $n \times 2$.

Example VIII.9. *Take the* 3×3 *matrix*

$$A = \begin{pmatrix} c & d & e \\ b & c & d \\ a & b & c \end{pmatrix},$$

with $d \neq 0$. *Then* $\phi(A)$ *is*

$$\phi(A) = \begin{pmatrix} c & d & e \\ b & 0 & 0 \\ a & 0 & 0 \end{pmatrix} = \begin{pmatrix} c & d \\ b & 0 \\ a & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & e/d \end{pmatrix}.$$

Definition VIII.10. *The matrices* (G, H) *in* $\mathbb{K}^{n\times 2}$ *such that* $\phi(A) = G\cdot H^t$ *are called* displacement generators *for the Toeplitz matrix A.*

These definitions can be generalized to an arbitrary matrix, which allows one to define the notion of a *quasi-Toeplitz* matrix.

Definition VIII.11. *The* displacement operator ϕ_+ *is the application* $A \mapsto A - Z \cdot A \cdot Z^t$, *where* A *is an arbitrary matrix in* $\mathbb{K}^{n\times n}$ *and* Z *is the* shift *matrix defined as*

$$Z = \begin{pmatrix} 0 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ \vdots & \ddots & \dots & \vdots \\ 0 & \cdots & 1 & 0 \end{pmatrix} \in \mathbb{K}^{n\times n}.$$

(Thus ZA is the matrix A shifted by one row to the bottom, and $A \cdot Z^t$ *is the matrix A shifted by one column to the right. Hence, this definition coincides with the above definition for Toeplitz matrices.)*

The displacement rank *of A is the integer $\alpha_+(A) = \operatorname{rank}(\phi_+(A))$. Then there exists a pair of $n \times \alpha$ matrices (G, H) such that $\phi_+(A) = G \cdot {}^tH$. Any such pair (G, H) is called a* displacement generator *(with respect to the operator ϕ_+). If $\alpha_+(A) \ll n$, we say that A is* quasi-Toeplitz*, and the integer $\alpha_+(A)$ is also called the* length *of the generator (G, H).*

Remark VIII.12. *The notation with index $+$ is used because, later in this document, we will use another notation with the index $-$.*

Intuitively, the displacement rank measures how far the matrix A is from a Toeplitz matrix, or from the concatenation of a few Toeplitz matrices.

Example VIII.13. *A Toeplitz matrix is quasi-Toeplitz, with displacement rank* 2.

Problem VIII.3. *Show that a Sylvester matrix is quasi-Toeplitz with displacement rank* 2.

This definition captures a wide notion of structured matrix, when the structure has some similarity with Toeplitz matrices. However, this definition does not cover matrices with structures similar to Vandermonde or Cauchy matrices. For these, we need to slightly adapt the displacement operator. This leads to the definition of *quasi-Vandermonde* and *quasi-Cauchy* matrices, as follows:

- $\mathbf{V}_\mathbf{a} \mapsto \mathbf{V}_\mathbf{a} - \operatorname{Diag}_\mathbf{a} \cdot \mathbf{V}_\mathbf{a} \cdot Z^t$ for quasi-Vandermonde matrices;
- $\mathbf{C}_{\mathbf{a},\mathbf{b}} \mapsto \mathbf{C}_{\mathbf{a},\mathbf{b}} - \operatorname{Diag}_\mathbf{a}^{-1} \cdot \mathbf{C}_{\mathbf{a},\mathbf{b}} \cdot \operatorname{Diag}_\mathbf{b}$ for quasi-Cauchy matrices.

Here, for two vectors $\mathbf{a} = (a_0, \dots, a_{n-1})$ and $\mathbf{b} = (b_0, \dots, b_{n-1})$, we write $\mathbf{V}_\mathbf{a}$ for the Vandermonde matrix $(a_i^j)_{i,j=0}^{n-1}$, we write $\operatorname{Diag}_\mathbf{a}$ for the diagonal matrix with entries given by $\mathbf{a}$, and we write $\mathbf{C}_{\mathbf{a},\mathbf{b}}$ for the Cauchy matrix $(1/(a_i - b_j))_{i,j=0}^{n-1}$.

In all cases, the operator $\phi_{M,N}$ is defined as $\phi_{M,N}(A) = A - MAN$, with M and N chosen according to the target structure; and the displacement rank of A (with respect to $\phi_{M,N}$) is defined as the rank of the matrix $\phi_{M,N}(A)$.

Problem VIII.4.
1. *Compute the displacement rank of a Vandermonde matrix and that of a Cauchy matrix, for the displacement operators defined above.*
2. *In both cases, give the corresponding generators (G, H).*

VIII.2. Main result

The key idea behind fast algorithms for structured matrices with displacement rank α is to use the displacement generators as a concise data structure. Indeed it has size $O(\alpha n)$, linear in the size n of the matrix when $\alpha \ll n$. The main result about structured matrices in this course is the following.

Theorem VIII.14. *(Structures: quasi-Toeplitz, quasi-Vandermonde, quasi-Cauchy). Let A be a matrix in $\mathbb{K}^{n\times n}$ with displacement rank α (with respect to a suitable displacement operator), and given by displacement generators (G, H) with both matrices G and H of dimensions $n \times \alpha$. Then, one can:*

1. *compute the determinant of A;*
2. *compute the rank of A;*
3. *for a given vector b in $\mathbb{K}^{n\times 1}$, solve the system $Ax = b$ or certify it has no solution,*

using $\tilde{O}(\alpha^2 n)$ operations in $\mathbb{K}$. More precisely, this complexity bound can be expressed in terms of the complexity of polynomial multiplication, as follows:

1. *$O(\alpha^2\,\mathsf{M}(n)\log(n))$ in the quasi-Toeplitz case;*
2. *$O(\alpha^2\,\mathsf{M}(n)\log^2(n))$ in the quasi-Vandermonde and quasi-Cauchy cases.*

This theorem yields a *unified algorithmic framework* with quasi-optimal complexity for several problems on polynomials and power series.

Corollary VIII.15. *One can compute, in $O(\mathsf{M}(n)\log n)$ operations in $\mathbb{K}$:*

1. *The extended GCD and the resultant of two polynomials of degree at most n;*
2. *a Padé approximant (f, g) such that $S = \frac{f}{g} \bmod x^{2n}$, where $\deg(f)$ and $\deg(g)$ are polynomials of degree at most n, of a given truncated series $S \bmod x^{2n}$.*

Proof. The resultant of two polynomials $A, B \in \mathbb{K}[x]$ is the determinant of their Sylvester matrix $\mathrm{Syl}(A, B)$. As seen above, this matrix has displacement rank at most 2.

The degree of $G = \gcd(A, B)$ is obtained from the rank of the Sylvester matrix: $\deg(G) = \deg(A) + \deg(B) - \operatorname{rank}\mathrm{Syl}(A, B)$. Once this degree is known, the Bézout equation $UA + VB = G$, with degree constraints $\deg(U) < \deg(B) - \deg(G)$ and $\deg(V) < \deg(A) - \deg(G)$, translates as a linear system in the coefficients of U, V and G, whose matrix is the concatenation of three Toeplitz matrices. Such a matrix is quasi-Toeplitz, with displacement rank at most 4. Similar considerations lead to the second item about computing Padé approximants. □

Problem VIII.5. *Let A of degree n and B of degree m. Considering the sketch of proof above, we take for granted the following claim: "the degree of $G = \gcd(A, B)$ is obtained from the rank of the Sylvester matrix: $\deg(G) = m + n - \operatorname{rank}\mathrm{Syl}(A, B)$", and also the fact that there exists (U, V) with $\deg(U) < m - \deg(G)$ and $\deg(V) < n - \deg(G)$.*

Under these assumptions, give a complete, detailed proof of the following statement: "Once this degree is known, the Bézout equation $UA + VB = G$, with degree constraints $\deg(U) < m - \deg(G)$ and $\deg(V) < n - \deg(G)$, translates as a linear system in the coefficients of U, V and G, whose matrix is the concatenation of three Toeplitz matrices. Such a matrix is quasi-Toeplitz, with displacement rank at most 4."

Corollary VIII.16. *Given m power series $f_1, \dots, f_m$ in $\mathbb{K}[[x]]$, known at precision $\sigma = \sum_{1 \le i \le m}(d_i + 1) - 1$, one can compute a Hermite-Padé approximant $(p_1, \dots, p_m)$ of type $(d_1, \dots, d_m)$ of these series using $O(m^2 \mathsf{M}(\sigma) \log(\sigma))$ operations in $\mathbb{K}$. This is a list of polynomials $(p_1, \dots, p_m)$ in $\mathbb{K}[x]$ with $\deg(p_i) < d_i$ for $1 \le i \le m$ and $p_1 f_1 + \cdots + p_m f_m = 0 \bmod x^\sigma$.*

Proof. This problem is linear in the coefficients in $\mathbb{K}$ of the sought approximant, which is a tuple of polynomials $(p_1, \dots, p_m)$. This rewrites as a linear system with a quasi-Toeplitz matrix, whose displacement rank is at most m. □

In what follows, we will sketch the main ideas and arguments behind the proof of Theorem VIII.14, in the quasi-Toeplitz case and for assuming the matrix A is invertible and "sufficiently generic" (all the submatrices that we want to invert are indeed invertible).

Problem VIII.6. *Let $P(x, y) = \sum_{j=0}^{2} \sum_{i=0}^{n-1} p_{i,j}\, x^i\, y^j \in \mathbb{K}[x, y]$ a bivariate polynomial that we seek. We consider points $(x_i, y_j) \in \mathbb{K}^2$ for $0 \le i \le n-1$ and $0 \le j \le 2$.*

1. *Give the matrix $W_{\mathrm{x,y}}$ which corresponds to evaluating P at the points (x_i, y_j). Give the determinant of this matrix.*
2. *How much does it cost to evaluate P at all (x_i, y_j)?*
3. *Show that this matrix is quasi-Vandermonde. What is its displacement rank?*
4. *Deduce a factorisation $\varphi(W_{\mathrm{x,y}}) = G\, H^{\mathrm{T}}$.*
5. *Given values $z_{i,j}$ for each $0 \le i < n$ and $0 \le j \le 2$, how much does it cost to find P such that $P(x_i, y_j) = z_{i,j}$ for all i, j?*

VIII.3. The quasi-Toeplitz case

In this section we prove the above main result, only focusing on the quasi-Toeplitz case since it contains the main algorithmic ideas and covers many applications. The quasi-Vandermonde and quasi-Cauchy cases are more technical, and in fact reduce to the quasi-Toeplitz case.

We will show that the above-defined notion of quasi-Toeplitz matrix is a good notion of structure, since it has the following properties:

(P1) the matrix-vector product can be performed in quasi-optimal time for a quasi-Toeplitz matrix;

(P2) the sum and the product of two quasi-Toeplitz matrices both remain quasi-Toeplitz;

(P3) the inverse of an invertible quasi-Toeplitz matrix remains a quasi-Toeplitz matrix.

The important fact is that these features form the basic requirements for the design of a "Strassen-like" inversion algorithm working with the concise representation based on displacement generators.

VIII.3.1. Matrix-vector product, quasi-Toeplitz case

The following result is the key point to prove the first property (P1) above.

Proposition VIII.17 (ΣLU formula)**.** *The operator $\phi_+ : A \mapsto A - Z \cdot A \cdot {}^tZ$ is invertible. More precisely, the following formula (called ΣLU formula) holds:*

$$A - Z \cdot A \cdot Z^t = G \cdot H^t \quad \textit{if and only if} \quad A = \sum_{i=1}^{\alpha} L(x_i) \cdot U(y_i),$$

where the x_i's (resp. y_i's) are columns of the generator G (resp. H) and where, for a column vector $v = [v_0 \cdots v_{n-1}]^t$, the matrix $L(v)$ is the lower triangular Toeplitz matrix

$$\begin{pmatrix} v_0 & 0 & \dots & 0 \\ v_1 & v_0 & \dots & 0 \\ \vdots & \ddots & \dots & \vdots \\ v_{n-1} & \cdots & v_1 & v_0 \end{pmatrix}$$

and $U(v)$ is the upper triangular Toeplitz matrix $L(v)^t$.

Proof. It can be observed that, by linearity, it is enough to prove the case $\alpha = 1$. Write $a = x_1$ for the (single) column of G and $b = y_1$ for the (single) column of H. If $C = L(a)U(b) = (c_{i,j})$, then a direct calculation shows $c_{i,j} = a_i b_j + a_{i-1} b_{j-1} + \cdots$, hence $c_{i,j} - c_{i-1,j-1} = a_i b_j$ and $\phi_+(C) = (a_i b_j)_{i,j=0}^{n-1} = a \cdot {}^t b$.

The proof of the converse follows from the fact that ϕ_+ is injective, which we leave as an exercise. □

This Proposition VIII.17 allows us to give an alternative, equivalent definition for the displacement rank.

Definition VIII.18. *The number $\alpha_+(A)$ is the smallest nonnegative integer α such that there exists a decomposition of the form*

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

where $L_1, \dots, L_\alpha$ are lower triangular Toeplitz matrices, and $U_1, \dots, U_\alpha$ are upper triangular Toeplitz matrices.

Example VIII.19. *Let A be a Toeplitz matrix. Denote by A_{inf} the lower triangular part of A and A_{ssup} the strictly upper triangular part of A. Then we have a decomposition $A = A_{inf} \cdot I_n + I_n \cdot A_{ssup}$. This implies that $\alpha_+(A) \leq 2$.*

Definition VIII.18 leads to a proof of property (P1) in Page 81.

Corollary VIII.20. *If A is given by a concise representation, via a pair of displacement generators (G, H) of dimensions $n \times \alpha$, then the matrix-vector product Av can be performed in $O(\alpha \mathsf{M}(n))$ operations in $\mathbb{K}$.*

Proof. The product Av is the sum of α terms, each of the form $L(x_i)(U(y_i)v)$. On the other hand, thanks to Lemma VIII.4, each of these terms can be computed in $O(\mathsf{M}(n))$ operations in $\mathbb{K}$. □

VIII.3.2. Addition and multiplication in concise representation via generators

To explain property (P2) in Page 81, we will further show that the concise representation via displacement generators allow an efficient (quasi-linear time) computation of both the addition and the multiplication of two quasi-Toeplitz matrices.

Proposition VIII.21 (Matrix operations in concise representation)**.**
Let (T, U) be a displacement generator of A of length α, and (G, H) be a displacement generator of B of length β. Then,

1. *$([T \mid G], [U \mid H])$ are displacement generators of $A + B$ of length $\alpha + \beta$;*
2. *$([T \mid W \mid a], [V \mid H \mid -b])$ are displacement generators for AB of length $\alpha + \beta + 1$;*

where $V := B^t \cdot U$, $W := Z \cdot A \cdot Z^t \cdot G$, and where the vector a (resp. b) is the last column of $Z \cdot A$ (resp. of $Z \cdot B^t$).

Problem VIII.7. *Prove the above proposition.*

Corollary VIII.22. *Using a concise representation via displacement generators of length at most α for both A and B, we can compute*

1. *the sum $A + B$ in $O(\alpha n)$ operations in $\mathbb{K}$;*
2. *the product AB in $\mathsf{Mul}(n, \alpha) := O(\alpha^2\, \mathsf{M}(n))$ operations in $\mathbb{K}$.*

Proof. The only nontrivial point is the computation of the matrices V and W and of the vectors a and b. This relies on the ΣLU formula. If $B = \sum_{i=1}^{\beta} L(x_i)U(y_i)$, then its transpose can be written $B^t = \sum_{i=1}^{\beta} L(y_i)U(x_i)$, so that the computation of $V = B^t \cdot U$ boils down to $2\alpha\beta$ polynomial multiplications. Each of these multiplications costs $O(\mathsf{M}(n))$. The same argument applies to the computation of W. Finally, computing a can be seen as multiplying A by the column vector $[0, \ldots, 0, 1]^t$, and a similar observation for b concludes the proof. □

VIII.3.3. Inversion using the concise representation with generators

To prove property (P3) in page 81, it will be convenient to use the following *dual displacement operator.*

Definition VIII.23. *The displacement operator ϕ_- of a matrix A in $\mathbb{K}^{n\times n}$ is defined by*

$$\begin{aligned}\phi_-(A) &= A - Z^t \cdot A \cdot Z \\ &= A - (A \text{ shifted by 1 towards the top and the left}).\end{aligned}$$

The corresponding displacement rank α_- is defined as $\alpha_-(A) = \operatorname{rank}(\phi_-(A))$.

The operators ϕ_+ and ϕ_- are related, and similarly to ϕ_+, there is an equivalent characterisation in terms of a ΣUL formula for ϕ_-.

Lemma VIII.24.

1. *$\alpha_-(A)$ is the smallest nonnegative integer α such that A can be written $A = \sum_{i=1}^{\alpha} U_i L_i$, where the L_i's are lower triangular Toeplitz matrices, and the U_i's are upper triangular Toeplitz matrices.*

2. *The following formula, called ΣUL formula, holds:*

$$A - Z^t \cdot A \cdot Z = \sum_{i=1}^{\alpha} x_i \cdot y_i^t \quad \text{if and only if} \quad A = \sum_{i=1}^{\alpha} U(rev(x_i)) \cdot L(rev(y_i)).$$

Here, for $v = [v_0, \dots, v_{n-1}]^t$, we write $rev(v) = [v_{n-1}, \dots, v_0]^t$.

3. *A Toeplitz matrix for ϕ_+ is a Toeplitz matrix for ϕ_-.*

The proof of this lemma follows from the next proposition.

Proposition VIII.25 (Conversion $\Sigma LU \leftrightarrow \Sigma UL$)**.** *For any matrix A, we have the inequality $|\alpha_+(A) - \alpha_-(A)| \leq 2$. Moreover, converting from a ΣLU representation to a ΣUL representation, and conversely, can be done in $O(\alpha\, \mathsf{M}(n))$ operations in $\mathbb{K}$.*

Proof. It suffices to prove the identity

$$L(x) \cdot U(y) = I_n \cdot L(y') + U(x') \cdot I_n - U(x'') \cdot L(y''),$$

where $x'' = Z \cdot \mathrm{rev}(x)$, $y'' = Z \cdot \mathrm{rev}(y)$, $y' = \mathrm{rev}(U(y)^t \cdot L(x)^t \cdot f)$, $x' = \mathrm{rev}(L(x) \cdot U(y) \cdot f)$, and $f = [0, \dots, 0, 1]^t$. □

Theorem VIII.26. *Let $A \in \mathbb{K}^{n \times n}$ be an invertible quasi-Toeplitz matrix. Then, its inverse is also quasi-Toeplitz, with $\alpha_+(A^{-1}) = \alpha_-(A)$.*

Proof. We have the following chain of equalities:

$$\begin{aligned}\alpha_-(A) &= \mathrm{rank}(A - Z^t \cdot A \cdot Z) = \mathrm{rank}(I_n - A^{-1} \cdot Z^t \cdot A \cdot Z) = \mathrm{rank}(I_n - Z \cdot A^{-1} Z^t \cdot A) \\ &= \mathrm{rank}(A^{-1} - Z \cdot A^{-1} \cdot Z^t) = \alpha_+(A^{-1}).\end{aligned}$$

Here we have used the following classical result from linear algebra: if A, B are two matrices, then $\mathrm{rank}(I_n - A \cdot B) = \mathrm{rank}(I_n - B \cdot A)$. □

VIII.3.4. Fast solving of quasi-Toeplitz linear systems

Theorem VIII.26 does not tell us *how* to compute generators for A^{-1}. This is done in the last result of this section (Theorem VIII.27), which is substantiated by the algorithm in Fig. VIII.1.

Theorem VIII.27. *Let $A \in \mathbb{K}^{n \times n}$ be an invertible quasi-Toeplitz matrix, with displacement rank α, given by generators G and H of length α. Then, one can compute generators of length α for A^{-1} using $O(\alpha^2\, \mathsf{M}(n) \log(n))$ operations in $\mathbb{K}$.*

From this representation of the inverse of A and from a vector $b \in \mathbb{K}^{n\times 1}$, the linear system $Ax = b$ can be solved using an extra $O(\alpha\,\mathsf{M}(n))$ operations in $\mathbb{K}$.

Proof. The idea is to adapt to the structured matrix case a recursive inversion algorithm due to Strassen. The difference here only lies in the choice of data structure used to represent and compute with quasi-Toeplitz matrices. Indeed, instead of the classical dense representation of matrices, here we use the data structure provided by displacement generators for the operators ϕ_+ and ϕ_- (or, equivalently, the ΣLU and ΣUL representations). This is a more concise representation in the case of quasi-Toeplitz matrices.

The approach, following the divide and conquer paradigm, is summarized in the algorithm in Fig. VIII.1. Its correction inherits from that of Strassen's inversion algorithm, which we will not detail here. The genericity assumption on A helps to ensure that all minors that must be inverted during the algorithm are indeed invertible.

The complexity $\mathsf{C}(n)$ of this algorithm satisfies the following recurrence definition: $C(1) = 1$, and for all $n \geq 2$,

$$\mathsf{C}(n) = 2\,\mathsf{C}(n/2) + O(\mathsf{Mul}(n,\alpha)) + O(\alpha^2 n + \alpha\,\mathsf{M}(n)).$$

The cost in $O(\alpha^2 n + \alpha\,\mathsf{M}(n))$ comes from the conversions between representations with respect to ϕ_+ and to ϕ_-, and the computations for minimizing the lengths of generators. The notation $\mathsf{Mul}(n,\alpha)$ stands for the complexity of multiplying two $n \times n$ matrices given by generators of length α. To conclude the proof, it remains to use the cost estimate

$$\mathsf{Mul}(n,\alpha) = O(\alpha^2\,\mathsf{M}(n)),$$

which implies $\mathsf{C}(n) = O(\alpha^2\,\mathsf{M}(n)\log(n))$. □

Problem VIII.8. *Let $A \in \mathbb{K}^{n\times n}$ be a quasi-Toeplitz matrix with displacement rank $\alpha \ll n$, concisely represented by displacement generators (G, H) of dimensions $n \times \alpha$. Recall that this means $\phi_+(A) = G \cdot H^t$.*

1. *Let $v_1, \dots, v_\alpha$ be column vectors in $\mathbb{K}^{n\times 1}$. Using the ΣLU formula, show that computing all matrix-vector products $A \cdot v_\ell$ for $1 \leq \ell \leq \alpha$ reduces to the following problem:*

   (P) *Given polynomials $G_j, H_j, V_j \in \mathbb{K}[x]$ for $j \leq \alpha$, each of degree less than n, compute*

   $$A_\ell = \sum_{j=1}^{\alpha} G_j\,(H_j V_\ell \bmod x^n) \text{ for } 1 \leq \ell \leq \alpha.$$

2. *Give a first algorithm to solve Problem **(P)**, and give a complexity bound.*

3. *Show that **(P)** can be reformulated as a matrix computation, as follows:*

   (MP) *Given univariate polynomial matrices $\mathbf{G}, \mathbf{V}$ in $\mathbb{K}[x]^{\alpha\times 1}$ and $\mathbf{H}$ in $\mathbb{K}[x]^{1\times\alpha}$, all of maximal degree less than n, compute $(\mathbf{VH} \bmod x^n)\,\mathbf{G}$.*

Input. A "generic" matrix $A \in \mathbb{K}^{n\times n}$, with $n = 2^k$, represented by generators with respect to the displacement operator ϕ_+.

Output. The inverse A^{-1}, represented by generators with respect to the displacement operator ϕ_-.

```
1. If $n = 1$, then return $A^{-1}$.
2. Compute $\phi_+$-generators for submatrices $a, b, c, d \in \mathbb{K}^{n/2\times n/2}$, where $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$.
3. Compute (recursively) $\phi_-$-generators for $e := a^{-1}$.
4. Compute $\phi_+$-generators for $S = d - ceb$.
5. Compute (recursively) $\phi_-$-generators for $t := S^{-1}$.
6. Return $\phi_-$-generators of $A^{-1} = \begin{pmatrix} x & y \\ z & t \end{pmatrix}$ via Strassen's formulas $y := -ebt$, $z := -tce$ et $x := e + ebtce$.
```

Figure VIII.1.. Strassen-based algorithm for inverting a quasi-Toeplitz matrix.

4. *Let* $\mathbf{A}, \mathbf{B}$ *and* $\mathbf{C}$ *polynomial matrices of dimensions* $(n \times p)$, $(p \times n)$ *and* $(n \times p)$ *and with degree at most* d*. Show that the product* $\mathbf{ABC}$ *can be computed in* $O(\frac{n}{p}\,\mathsf{MM}(p,d))$ *operations in* $\mathbb{K}$*.*
5. *Propose a divide and conquer algorithm (with the recursion on* n*) which solves Problem* ***(MP)*** *in complexity* $O(\frac{1}{\alpha}\,\mathsf{MM}(\alpha,n))$*.*
6. *Conclude that, using the representation with displacement generators, we can multiply two quasi-Toeplitz matrices in* $\mathbb{K}^{n\times n}$ *with displacement rank at most* α *in* $O(\frac{1}{\alpha}\,\mathsf{MM}(\alpha,n))$ *operations in* $\mathbb{K}$*.*

IX. Error correcting codes; decoding algorithms

Source: the beginning of this chapter is inspired from Chapter 7 of the book "Modern Computer Algebra", written by Joachim von zur Gathen and Jürgen Gerhard (3rd edition, 2013).

Coding theory deals with the detection and correction of errors during the transmission of digital information. The scenario is that a message is sent over a transmission channel, and due to noise on the channel some of the symbols in the received message may be different from those in the original message. How can we detect and/or correct these erroneous symbols? How to guarantee efficiency, both by avoiding to significantly increase the volume of data compared to the original message, and by relying on fast algorithms such as those seen in previous chapters?

IX.1. Introduction to error correcting codes

This problem has been studied for many decades, after pioneering work by Hamming and Shannon in the end of the 1940s. Error correcting codes (of the Reed-Müller family) have notably been used in the 1970s for deep-space telecommunications, with Voyager missions which had to transmit colored images and scientific information from Jupiter and Saturn. Error correcting codes are also useful for data storage; for example, variants of Reed-Solomon codes have been used in the Compact Disc standard publicly introduced at the beginning of the 1980s.

Error correcting codes are ubiquitous nowadays in everyday life, through various real-life applications. They are incorporated in the internet TCP/IP stack; they arise in satellite broadcasts (e.g. for HDTV); they help to reduce the amount of data stored and the risks due to storage failures in the context of distributed storage; they are used to make the use of QR codes faster and more robust; they can also ensure that large-scale computations can go through and give correct errors even if a small number of intermediate computations were erroneous.

To fix ideas, we will focus on a context of digital transmission of a message. For this purpose, the *message* is generally split into a sequence of smaller, fixed-length *words*. We will focus on the transmission of a single word, with possible errors on each symbol of this word. Then, the transmission of an arbitrary-length message is simply a series of transmissions of a single word. Let us write w for the word to be sent over the channel, and r for the received message.

$$\underset{w}{\bullet} \xrightarrow{\text{transmission channel}} \underset{r}{\bullet}$$

The fundamental idea behind error detection and correction is based on the introduction of *redundancy.* A basic strategy is to send w three or five times, and take a majority vote on each symbol. Of course, if errors occur too frequently, then this may not help much. But the usual assumption is that errors occur only with fairly small probability (this probability can be known from properties of the communication channel), and then this strategy will give an erroneous result only with much smaller probability than accepting r as is.

However, the cost of transmission (i.e. the total volume of transmitted data) has increased by a factor of three or five. (Similarly, in a context of storage, this means we would need to store three to five times more data than what the actual data requires.) The fundamental task of coding theory is to see whether small error probability can be achieved at reasonable cost. Error correcting codes must not be confused with cryptography, which can be roughly summarized as the art of sending secret messages that only the intended receiver can read/interpret/work with.

It turns out that the tools of algebra provide many useful codes. We describe a particular, but still quite general and useful class of such codes: *linear* codes. Let $\mathbb{F}_q$ be a finite field with q elements, and let $k < n$ be integers. Then, a *linear code* C of parameters (n, k) over $\mathbb{F}_q$ is an $\mathbb{F}_q$-linear subspace of $\mathbb{F}_q^n$ of dimension k. Any basis of C provides an isomorphism $\mathbb{F}_q^k \longrightarrow C$, and one picks a suitable such isomorphism $\varepsilon : \mathbb{F}_q^k \longrightarrow C$ which will be the *encoding map.* The number n is the length of C, k is the *dimension* of C, and the ratio $\frac{k}{n} \leq 1$ is the *rate* of C.

To transmit a word w, we first identify it with an element of $\mathbb{F}_q^k$. If, say, $q = 2$ and $k = 64$, and we want to transmit messages in ASCII, then each ASCII letter can be identified with an 8-bit string, and a block of 8 letters is a word $w \in \mathbb{F}_2^{64}$. In this situation, the simple code which sends each word three times has length 192, dimension 64, and rate $\frac{64}{192}$. Here, $\varepsilon : \mathbb{F}_2^{64} \longrightarrow \mathbb{F}_2^{192}$ is such that $\varepsilon(w) = (w, w, w)$.

For $w \in \mathbb{F}_2^{64}$, this encoding of w into $c = \varepsilon(w) \in C$, yields the *codeword* c associated to w. Codewords are simply elements of C; note that there are elements in $\mathbb{F}_q^n \setminus C$, which are in $\mathbb{F}_q^n$ but are not codewords. This codeword $c = \varepsilon(w)$ contains the same information as w, but redundancy has been introduced; c uses n symbols whereas w uses only k symbols, with $k < n$. The rate $\frac{k}{n}$ measures the ratio of the actual data versus the data augmented with redundancy.

$$\begin{tikzcd}
\bullet \arrow[r, "\text{encoding}"] & \bullet \arrow[r, "\text{transmission channel}"] & \bullet \\
w & c = \varepsilon(w) & r
\end{tikzcd}$$

Upon receiving r, the goal is to determine if there has been errors, and in that case, to correct them. Errors mean that some components of r differ from the corresponding ones in c: they have been altered during transmission. Finding w from r (or sometimes finding c, from which one then deduces w), is called the *decoding stage* of the error correcting codes.

The target is therefore to maximize the rate (minimizing redundancy, making communication cheaper), while also maximizing the number of errors one can detect and correct. Of course, these are conflicting goals: the smaller the redundancy, the smaller the number of errors we can hope to correct (as we will see below, with the Singleton bound). Yet, we can hope for codes that are much more efficient than the above repetition code $\varepsilon(w) = (w, w, w)$, where the redundancy is large and the correction capacity rather limited. Of course, another target is to ensure that both the encoding and decoding stages can be performed in an efficient way.

For an element $a = (a_1, \dots, a_n) \in \mathbb{F}_q^n$, we denote by

$$\operatorname{hweight}(a) = \#\{i \mid 1 \leq i \leq n, a_i \neq 0\},$$

the *Hamming weight* of a, and by

$$\mathrm{d}_{\min}(C) = \min_{a \in C \setminus \{0\}} \operatorname{hweight}(a)$$

the *minimal distance* of the code C. The terminology of “distance” comes from the fact that the classical *Hamming distance* between $a \in \mathbb{F}_q^n$ and $b \in \mathbb{F}_q^n$, defined as the number of coordinates i such that $a_i \neq b_i$, is equal to $\operatorname{hweight}(a - b)$. Since C is a $\mathbb{F}_q$-linear subspace, we have $\mathrm{d}_{\min}(C) \leq \operatorname{hweight}(a - b)$ for all distinct codewords $a, b \in C$.

On receiving a word $r \in \mathbb{F}_q^n$, it is decoded as $c \in C$ such that the distance between r and c, which is $\operatorname{hweight}(r - c)$, is minimal. Since fewer errors are more probable, this is called *maximum likelihood decoding*. If less than $\mathrm{d}_{\min}(C)/2$ errors have occurred in transmitting the word, then this decoding procedure will work correctly.

Figure IX.1.. *Encoding stage: words in $\mathbb{F}_q^k$ (red dots on the left) are transformed, using the injective encoding map ε, into code words (blue dots on the right). The blue discs show all words at distance at most e of codewords, which are all possible received words if the number of errors during transmission does not exceed e. We see that, on this visual example, unique decoding is possible since there is only one codeword in each blue disc.*

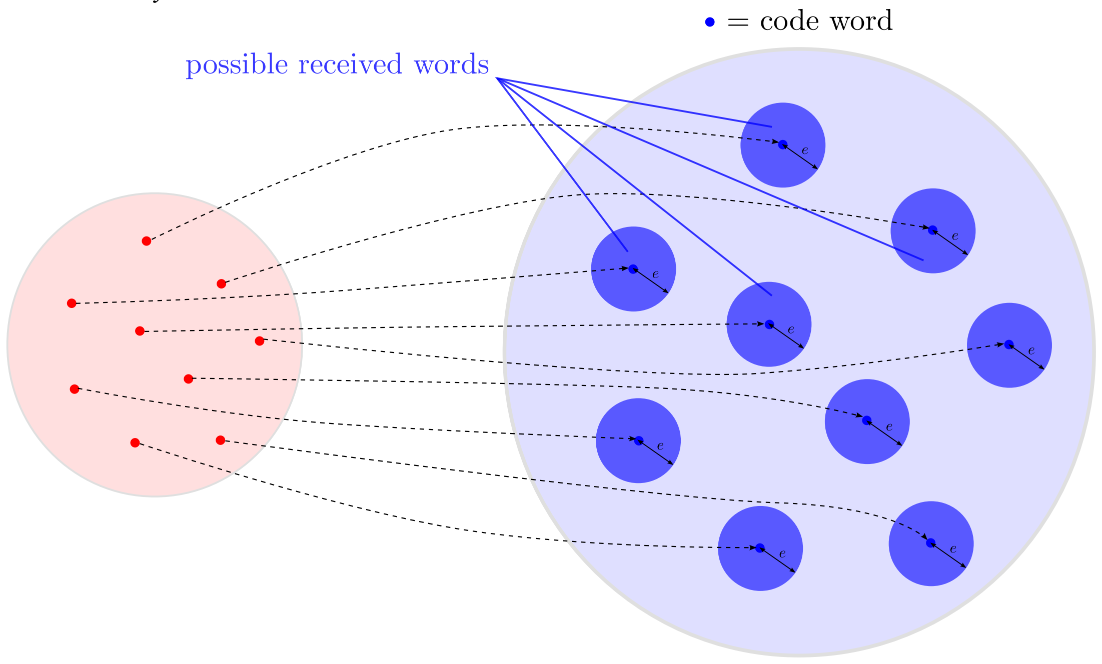
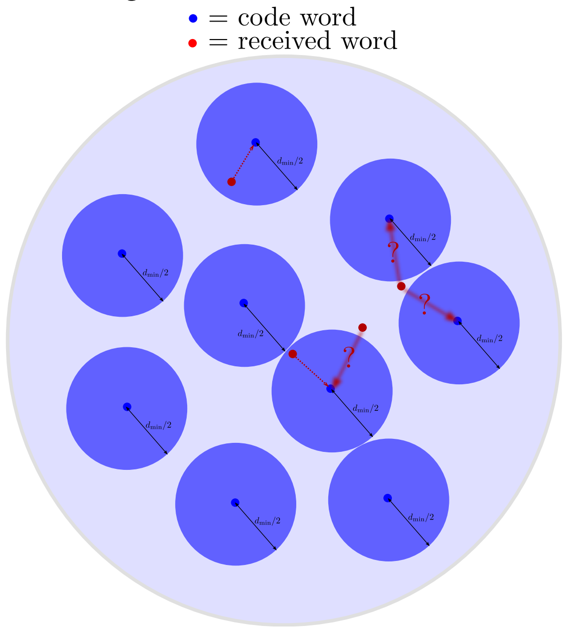


Figure IX.2.. Feasibility of the (unique) decoding: as long as there are less than $\mathrm{d}_{\min}(C)/2$ *errors, there is a single codeword at minimum distance from the received word, and it is the codeword we are looking for in the decoding procedure. However, If there has been* $\mathrm{d}_{\min}(C)/2$ *errors or more, there might be several code words at minimum distance of the received word, or there might be a single one but which is not the right one.*



The following important result states that, for any linear code, the minimum distance cannot exceed $n - k + 1$.

Lemma IX.1 (Singleton bound)**.** *If* C *is an* $\mathbb{F}_q$*-linear code of length* n *and dimension* k*, then its minimum distance satisfies* $\mathrm{d}_{\min}(C) \leq n - k + 1$.

Proof. Define $\mathbb{K} = \mathbb{F}_q$. Let $G \in \mathbb{K}^{k\times n}$ be a matrix whose rows form a basis of the $\mathbb{K}$-vector space C. Since G has rank k, there exist an invertible matrix $P \in \mathbb{K}^{k\times k}$ and a permutation matrix $Q \in \mathbb{K}^{n\times n}$ such that $G = P \begin{bmatrix} I_k & H \end{bmatrix} Q$. Then, each row of $\begin{bmatrix} I_k & H \end{bmatrix} Q$ is a codeword in C, and has Hamming weight at most $n - k + 1$. So, the minimal distance of C cannot be greater than $n - k + 1$. □

A corollary of this result is that, in the unique decoding context we consider here, we cannot hope to find error correcting codes (along with a correct unique decoding algorithm) when there might be $\frac{n-k+1}{2}$ errors or more. Observe that $n - k$ is the number of symbols we introduced to add redundancy: intuitively, this says that if there are more errors than about half the amount of redundancy, then the received word is too far from the sent codeword, and there is no possibility of decoding uniquely. On the other hand, in the most favorable case, $\mathrm{d}_{\min}(C) = n - k + 1$ and then we can hope to correct up to $\lfloor \frac{n-k}{2} \rfloor$ errors. Codes satisfying this are called *maximum distance separable* codes.

Problem IX.1. *Consider the above repetition codes* $C_3 = \{(w, w, w) \in \mathbb{F}_2^{192} \mid w \in \mathbb{F}_2^{64}\}$ *and* $C_5 = \{(w, w, w, w, w) \in \mathbb{F}_2^{320} \mid w \in \mathbb{F}_2^{64}\}$.

(i) Give the minimal distance of C_3*, and that of* C_5*. Compare these to the best possible minimal distance, given by the Singleton bound. Would you say these codes are "good" codes?*

(ii) For each of these codes, explain the above claim: if less than $\mathrm{d}_{\min}(C)/2$ *errors have occurred in transmitting the word, then maximum likelihood decoding will work correctly.*

(iii) For each of these codes, prove that if there can be $\mathrm{d}_{\min}(C)/2$ *errors or more, then there exists no algorithm capable of decoding correctly.*

IX.2. Reed-Solomon codes an their decoding

This final section of the course is presented as a step-by-step problem. As you will observe, many of the algorithmic ideas presented above in this document arise in the single problem of the *decoding of Reed-Solomon codes.*

Warm-up. General properties.

Definition IX.2 (Reed-Solomon codes.)**.** *Let* $k \leqslant n$ *be two positive integers. Assume* $n \leqslant q$ *and choose* $x_1, \ldots, x_n$ *which are* n *distinct elements in* $\mathbb{F}_q$ (evaluation points)*. The encoding function (which depends on* $(\mathbb{F}_q, n, k, x)$*) is defined as* $\varepsilon : \mathbb{F}_q^k \longrightarrow \mathbb{F}_q^n$ *is defined as follows: a message* $\mathbf{w} = (w_0, \ldots, w_{k-1}) \in \mathbb{F}_q^k$ *is mapped to a polynomial* $w = \sum_{i=0}^{k-1} w_i X^i \in \mathbb{F}_q[X]$*, and then*

$$\varepsilon(\mathbf{w}) = (w(x_1), \ldots, w(x_n)).$$

The code itself is $\mathrm{RS}_{\mathbb{F}_q,n,k,x} = \{\varepsilon(\mathbf{w}), \mathbf{w} \in \mathbb{F}_q^k\}$.

1. Show that a Reed-Solomon code is a linear code, and give its length and dimension.
2. Recall the definition of the minimum distance $d = \mathrm{d}_{\min}(\mathrm{RS}_{\mathbb{F}_q,n,k,x})$, and show that Reed-Solomon codes are maximal distance separable codes: $d = n - k + 1$.
3. If we are to design a decoding algorithm, what is the maximal number of errors we can accept during transmission in order to be able to correct the received word?

Suppose a friend of yours encodes the word $\mathbf{w}$ *into a codeword* $\mathbf{c} = (w(x_1), \ldots, w(x_n))$ *and sends it to you. Unfortunately there are errors during the transmission; you receive a word* $(y_1, \ldots, y_n)$*, and you know that no symbol was erased or inserted, and that there are at most* e *erroneous symbols, that is,* $|\{i \mid w(x_i) \neq y_i\}| \leqslant e$ *(for some* $e \leqslant n$ *that you have deduced from the channel properties).*

First half. Brute-force attempts.

1. Give two algorithms (a naïve one, a fast one) that can be used to encode a word $\mathbf{w} \in \mathbb{F}_q^k$. For a practical implementation, how would you choose n and the points $x_1, \ldots, x_n$?

2. Actually, this time you are very lucky and you know that there was absolutely no error during the transmission: $e = 0$. How can you decode the received word $(y_1, \ldots, y_n)$? Give a complexity bound.

3. Now, you are just a bit less lucky: there can be at most one erroneous symbol: $e = 1$. Assuming $k < n - 1$, give an algorithm to decode the received word $(y_1, \ldots, y_n)$.

   Hint: chase the (possible) error location.

   Give a complexity bound, and explain why this idea cannot be generalized for correcting more errors with a reasonable cost.

Half-time. Time to think and write down the equations.

Now, we assume that the upper bound on the number of errors allows unique decoding:

$$|\{i \mid w(x_i) \neq y_i\}| \leqslant e \leqslant \left\lfloor \frac{d-1}{2} \right\rfloor = \left\lfloor \frac{n-k}{2} \right\rfloor .$$

We define the "error-locator" polynomial Λ and the "master" polynomial G as follows:

$$\Lambda(X) = \prod_{i \mid y_i \neq w(x_i)} (X - x_i), \quad G(X) = \prod_{1 \leqslant i \leqslant n} (X - x_i).$$

We denote by R the interpolation polynomial of degree less than n such that $R(x_i) = y_i$.

1. Give a degree bound on Λ; show that if you find w, you can deduce Λ.

2. Show that if you know Λ, you can deduce w.

3. Show that we have the following set of n quadratic equations in the unknown $e + k - 1$ coefficients of Λ and w:

$$\text{for every } i \in \{1, \ldots, n\}, \quad \Lambda(x_i) y_i = \Lambda(x_i) w(x_i). \qquad \text{(IX.1)}$$

   Such quadratic equations are hard to solve in general; however here, we will be able to solve them efficiently. The main idea is to linearize these equations by forgetting the factorization of the right-hand side: we replace the quadratic unknown Λw by a single unknown ω of degree $< e + k$. This will prove sufficient to solve the problem.

Second half. Use what you learned…

1. **… about linear systems.**

a. Show that $\lambda = \Lambda, \omega = \Lambda w$ form a solution of the "linearized" system:

$$\begin{cases} \text{for every } i \in \{1, \ldots, n\}, \quad \lambda(x_i)y_i = \omega(x_i), \\ \deg(\lambda) \leqslant e, \quad \deg(\omega) < e + k, \quad \lambda \text{ monic.} \end{cases} \tag{IX.2}$$

b. Show that if λ_1, ω_1 and λ_2, ω_2 are two solutions to Eq. (IX.2), then $\omega_1/\lambda_1 = \omega_2/\lambda_2$.

   Hint: here comes the unique-decoding assumption into play!

c. Give an algorithm to find Λ and w in $O(n^3)$ operations in $\mathbb{K}$.

2. **… about rational reconstruction.**

a. Show that $\lambda = \Lambda, \omega = \Lambda w$ form a solution of the *Key Equation*:

$$\begin{cases} \lambda R = \omega \mod G, \\ \deg(\lambda) \leqslant e, \quad \deg(\omega) < e + k, \quad \lambda \text{ monic.} \end{cases} \tag{IX.3}$$

b. Show that if λ_1, ω_1 and λ_2, ω_2 are two solutions to (IX.3), then $\omega_1/\lambda_1 = \omega_2/\lambda_2$.

   Hint: try not to go back to the definition of G; degree considerations are enough to conclude.

c. Recall (briefly) how to solve a rational reconstruction problem such as (IX.3).

d. Give an algorithm to decode the received word $(y_1, \ldots, y_n)$, and a complexity bound.

3. **… about structured matrices.**

Here, we assume that all points x_i are nonzero. We define the reciprocal polynomials $\overline{R}(X) = X^{n-1}R(X^{-1})$, $\overline{G}(X) = X^n G(X^{-1})$, and the polynomial $S(X) = \overline{R}/\overline{G} \mod X^{n-k}$.

a. Why is $S(X)$ well-defined? Give an algorithm to compute $S(X)$, and a complexity bound.

b. Show that (IX.3) is equivalent to

$$\begin{cases} \overline{\lambda} S = \overline{\mu} \mod X^{n-k}, \\ \deg(\overline{\lambda}) \leqslant e, \quad \deg(\overline{\mu}) < e, \quad \overline{\lambda}(0) = 1. \end{cases} \tag{IX.4}$$

c. Show that you can find a solution to (IX.4) by solving a homogeneous linear system whose matrix is a $(n-k-e) \times (n-k-e)$ Toeplitz matrix.

d. Give an algorithm to decode the received word, and a complexity bound.

   Hint: we have previously seen how to solve an $N \times N$ Toeplitz system using $O(N^2)$ operations in the base field, or even $O(\mathsf{M}(N)\log(N))$ operations with the best known algorithms.

Post-match celebration.

If you liked this section about the (unique) decoding of Reed-Solomon codes, you may be interested in reading about the list-decoding of Reed-Solomon codes; don't hesitate to ask your teachers for some references for further reading on the subject.

A. Basic algebraic structures

In this appendix, we study basic algebraic structures which are commonly encountered in areas such as cryptography, numerical computing, quantum information, computer vision, etc. These are sets which are equipped with *binary operations* (which play the role of *arithmetic operators*) acting on elements of these sets. The properties of these operators are of first importance since they determine what can be computed (and how) as well as intrinsic properties of the sets under consideration.

To be precise, letting E be a set, a binary operator $\star$ acting on E is a map

$$\begin{aligned} \star\colon E \times E &\longrightarrow E \\ (e_1, e_2) &\longmapsto (e_1 \star e_2). \end{aligned}$$

The very early binary operator that all kids learn is the *addition* over the integers. A second one is the *multiplication* over the integers.

Hence, denoting by $\star$ a binary operator is taking a "*multiplicative*" notation. We could indeed denote by $\oplus$ (or simply $+$) this operator, hence using an "*additive*" notation without any impact on the theory. Further, we use the multiplicative notation.

A.1. Groups

Definition A.1 (Group). *Let G be a set equipped with the binary operation $\star$. One says that $(G, \star)$ is a group if the following hold:*

1. *the operation $\star$ is* associative*: for all x, y, z in G,*

$$x \star (y \star z) = (x \star y) \star z;$$

2. *there exists $e \in G$, called* identity element*, such that for all $x \in G$,*

$$x \star e = e \star x = x;$$

3. *for all $x \in G$, there exists $y \in G$, called* inverse of x *such that*

$$x \star y = y \star x = e.$$

   Usually, such an element y is denoted by x^{-1}.

Example A.2. $(\mathbb{Z}, +)$, $(\mathbb{R}^*, \times)$ *are groups.* $(\mathbb{N}, +)$ *and* $(\mathbb{Z}, \times)$ *are not groups.*

Definition A.3.

1. *A group is said to be* abelian *(or* commutative*) if the binary operation is* commutative*: for all $x, y \in G$,*
$$x \star y = y \star x.$$

2. *A group is said to be* finite *when its cardinality is finite.*

Remark A.4.

1. *If there is no ambiguity on $\star$, we may only talk about group G.*

2. *If $\star$ is a multiplication arithmetic operator, it can also be denoted by the $\times$ symbol, the $\cdot$ symbol or even by juxtaposition. The identity element is then denoted by Id_G, 1_G or even Id or 1.*

3. *If $\star$ is an addition arithmetic operator, then G needs to be commutative. The identity element is then denoted by 0_G or even 0. The inverse of x is denoted by $-x$ and is called its* opposite*. Furthermore, $x + (-y)$ is shortened to $x - y$.*

Problem I.1.

1. *Show that $(\{1\}, \cdot)$ and $(\{1, -1\}, \cdot)$ are groups.*

2. *Show that the identity element is unique in a group.*

3. *Show that if for $x \in G$, there exists $y \in G$ such that $x \star y = e$, the identity element, then y* is *the inverse of x. That is, we only need to check that y is the inverse on one side.*

4. *Show that the inverse of x is unique in G.*

5. *Show that for a and b be in $(G, \cdot)$, $(a \cdot b)^{-1} = b^{-1} \cdot a^{-1}$.*

6. *Show that the set of invertible matrices 2×2 matrices is a multiplicative group. Is it abelian ?*

Definition A.5 (Subgroup)**.** *Let $(G, \star)$ be a group. $(H, \star)$ is a* subgroup *of $(G, \star)$ if*

1. *H is a subset of G containing e_G;*

2. *for all $h, h' \in H$, $h \star h' \in H$;*

3. *for all $h \in H$, $h^{-1} \in H$.*

Theorem A.6 (Lagrange)**.** *Let G be a finite group. For all subgroup H of G, the size (or* order*) of H divides the size (or* order*) of G.*

Corollary A.7. *Let G be a finite group. For any $g \in G$, there exists a nonnegative integer n such that $g^n = e_G$. The smallest such n is called the* order *of g and it divides the order of G.*

Problem I.2. *Prove Lagrange's theorem and its corollary.*

Problem I.3. *Let p be a prime number and consider the set $\frac{\mathbb{Z}}{p\mathbb{Z}} = \{0, \dots, p-1\}$ (sometimes it is denoted by $\mathbb{F}_p$) – it is the set of remainders of any integer in $\mathbb{Z}$ modulo p. Further we denote by $\frac{\mathbb{Z}}{p\mathbb{Z}}^*$ the set $\{1, \dots, p-1\}$.*

Consider the multiplication operator $\star : (a, b) \in \frac{\mathbb{Z}}{p\mathbb{Z}}^2 \mapsto (a \times b) \mod p$ where $\times$ is the classical integer multiplication.

1. *Prove that $\left(\frac{\mathbb{Z}}{p\mathbb{Z}}^*, \star\right)$ is a finite group.*
2. *We say that ω is an nth root of unity if $\omega^n = 1$.*

   Assume that n divides $p-1$. Is there a root of unity of order n in $\frac{\mathbb{Z}}{p\mathbb{Z}}^*$?

Such a property is used in cryptography for implementing fast algorithms for polynomial multiplication of univariate polynomials with prime field coefficients. These are algorithms based on the so-called FFT (Fast Fourier Transform).

Problem I.4.

1. *Show that $2\mathbb{Z}$, the set of even integers, is a subgroup of $\mathbb{Z}$.*
2. *Show that $\mathbb{R}_+^*$ is a subgroup of $\mathbb{R}^*$.*
3. *Show that the set of upper triangular matrices of size n with rational coefficients is a subgroup of $\mathbb{Q}^{n\times n}$.*

Definition A.8 (Group homomorphism)**.** *Let $(G, \star)$ and $(H, \circ)$ be two groups. A map $f : (G, \star) \longrightarrow (H, \circ)$ is a* group homomorphism *if for any pair $(x, y) \in G \times G$, $f(x \star y) = f(x) \circ f(y)$.*

Moreover, when f is a bijection, then f is said to be a group isomorphism*; besides G and H are said to be* isomorphic.

Problem I.5. *Let f be a group homomorphism between G and H.*

1. *Show that $\ker f = \{x \in G | f(x) = e_H\}$ is a subgroup of G and that f is* injective *if, and only if, $\ker f = \{e_G\}$.*
2. *Show that $\operatorname{Im} f = \{f(x) | x \in G\}$ is a subgroup of H and that f is* surjective *if, and only if, $\operatorname{Im} f = H$.*
3. *Show that $\exp$ is a group isomorphism between $\mathbb{R}$ and $\mathbb{R}_+^*$.*
4. *Show that if G is finite, then $|G| = |\ker f| \times |\operatorname{Im} f|$, where $|G|$ means the size of G.*

Problem I.6. *We denote by $(\mathbb{Z}/n\mathbb{Z}, +)$ the group of integers* modulo n.

1. *Show that for $n \in \{1, 2, 3, 5, 7\}$, this is the only group of size n up to an isomorphism.*
2. *Show that there is only one other group of size 4, up to an isomorphism. Is it isomorphic to $(\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}, +)$?*

3. *Show that there is only one other group of size 6, up to an isomorphism. Is it isomorphic to* $(\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/3\mathbb{Z}, +)$*?*

Problem I.7. *Let* $(G, \cdot)$ *be a group. We denote by* $\operatorname{Aut} G$ *the set of automorphisms of* G*, i.e. the set of bijective homomorphisms from* G *to* G*.*

1. *Show that it is a group for the composition* $\circ$*.*
2. *Let* $g \in G$*, show that* $f_g \colon G \longrightarrow G$ *such that for all* $x \in G$, $f_g(x) = gxg^{-1}$ *is an automorphism of* G*.*
3. *Show that* $f \colon G \longrightarrow \operatorname{Aut} G$ *such that* $f(g) = f_g$ *is a homomorphism of groups.*
4. *Determine* $\ker f$*. What can we say about* $\ker f$ *when* G *is* abelian*?*

A.2. Rings

We introduce now rings which enrich the model of groups since we are now considering a set equipped with two binary operations, still with extra properties.

Definition A.9 (Ring)**.** *Let* R *be a set equipped with two binary operations* $+$ *and* $\times$*. One says that* $(R, +, \times)$ *is a ring if the following holds:*

1. $(R, +)$ *is an abelian group whose identity element is denoted* 0_R*;*
2. $\times$ *is an associative binary operation whose identity element is denoted* 1_R*;*
3. $\times$ *is distributive with respect to* $+$*: for all* $x, y, z \in R$*,*

$$x \times (y + z) = x \times y + x \times z, \quad (x + y) \times z = x \times z + y \times z.$$

Problem I.8.

1. *Show that* $(\mathbb{Z}, +, \times)$ *and* $(\mathbb{Z}/n\mathbb{Z}, +, \times)$*, for all* $n \in \mathbb{N}^*$*, are rings.*
2. *Show that for all* $x \in R$, $0_R \times x = 0_R$*.*
3. *Show that* 1_R *is unique.*
4. *We let* -1_R *be the opposite of* 1_R *for* $+$*. Show that for all* $x \in R$, $-1_R \times x$ *is the opposite of* x*. Compute* $(-1_R)^2$*.*

Definition A.10.

- *A ring is said to be* commutative *when* $\times$ *is commutative;*
- *A ring is said to be an* integral domain *when it is commutative and when for all* $x, y \in R$, $x \times y = 0$ *implies* $x = 0$ *or* $y = 0$*.*

  When this occurs, one says that R *does not admit a* zero divisor*.*

Problem I.9.

1. *Is $\mathbb{Z}^{n\times n}$ (the set of $n \times n$ matrices with entries in $\mathbb{Z}$) a ring? a commutative one? an integral domain?*

2. *Give a condition on $n \in \mathbb{N}^*$ so that $\mathbb{Z}/n\mathbb{Z}$ is an integral domain.*

Definition A.11 (Subring). *Let $(R, +, \times)$ be a ring. $(S, +, \times)$ is a* subring *of $(R, +, \times)$ if*

1. *S is a subset of R containing 0_R and 1_R;*

2. *for all $s, s' \in S$, $s + s' \in S$ and $s \times s' \in S$;*

3. *for all $s \in S$, $-s \in S$.*

Definition A.12 (Ideal). *Let $(R, +, \times)$ be a ring. I is an* ideal *of R if*

1. *I is a subset of R containing 0_R;*

2. *for all $s, s' \in I$ and $r \in R$, $s + s' \in S$ and $s \times r \in S$;*

3. *for all $s \in S$, $-s \in S$.*

Example A.13. *For $a \in R$, the set $\langle a\rangle = \{ar | r \in R\}$ is an ideal of R, called* principal.

Problem I.10. *Show that $\mathbb{Z}$ has only principal ideals.*

Problem I.11. *For R a ring and $x \notin R$, let $R[x]$ be the set of polynomials in x with coefficients in R.*

1. *Show that $R[x]$ is a ring.*

2. *Show that $\langle 2, x\rangle = \{2a + xb | a, b \in \mathbb{Z}[x]\}$ is an ideal of $\mathbb{Z}[x]$.*

3. *Show that $\langle 2, x\rangle$ is not a principal ideal.*

Definition A.14 (Ring homomorphism). *Let $(R, +, \times)$ and $(S, \star, \circ)$ be two rings. One says that a map $f : R \longrightarrow S$ is a* ring homomorphism *if for all $(x, y) \in R \times R$, the following holds:*

- *$f(x + y) = f(x) \star f(y)$;*

- *$f(1_R) = 1_S$;*

- *$f(x \times y) = f(x) \circ f(y)$.*

Moreover, when f is a bijection, then f is said to be a ring isomorphism*; besides R and S are said to be* isomorphic.

Problem I.12. *Let f be a ring homomorphism between R and S.*

1. *Show that* $\ker f = \{x \in R | f(x) = 0\}$ *is an ideal of* R.
2. *Show that* $\operatorname{Im} f = \{f(x) | x \in R\}$ *is a subring of* S.
3. *Let* R *be a ring. Show that there is a unique ring homomorphism from* $\mathbb{Z}$ *to* R *whose kernel is* $n\mathbb{Z}$ *with* $n \in \mathbb{N}$. *We then say that* R *has* characteristic n.
4. *Show that a finite ring must have positive characteristic, i.e. nonzero.*

Observe that not all rings are ordered. This yields some difficulty for defining a (Euclidean) division over rings. To bypass this difficulty, one introduces a function which embeds the considered ring in $\mathbb{N}$.

Definition A.15 (Euclidean division)**.** *Let* $(R, +, \times)$ *be a ring. One says that* R *is equipped with a Euclidean division if there exists a map* $h : R - \{0\}$ *in* $\mathbb{N}$ *such that for all* a, b *in* $R - \{0\}$ *the following holds:*

- $h(a \times b) \geq h(a)$;
- *there exists a couple* $(q, r) \in R \times R$ *such that* $a = b \times q + r$ *with either* $r = 0$ *or* $h(r) < h(b)$.

Using the above notations, q *is said to be the* quotient *and* r *is said to be the* remainder *of the Euclidean division.*

Definition A.16. *A ring* R *is said to be* Euclidean *if it is an integral domain and it is equipped with a Euclidean division.*

A.3. Fields

Definition A.17 (Field)**.** *Let* $\mathbb{K}$ *be a set equipped with two binary operations* $+$ *and* $\times$. *On says that* $(\mathbb{K}, +, \times)$ *is a field if the following holds:*

1. $(\mathbb{K}, +, \times)$ *is a commutative ring;*
2. *the identity elements of* $+$ *and* $\times$ *(denoted respectively* $0_{\mathbb{K}}$ *and* $1_{\mathbb{K}}$*) do not coincide;*
3. $(\mathbb{K}^*, \times)$ *is a group (where* $\mathbb{K}^* = \mathbb{K} \setminus \{0_{\mathbb{K}}\}$*).*

A field is said to be finite when its cardinality is finite.

Theorem A.18. *Any field is an integral domain. Any finite integral domain is a field.*

Problem I.13. *Prove Theorem A.18.*

Definition A.19 (Subfield)**.** *A subset* $\mathbb{K}$ *of a field* $(\mathbb{L}, +, \times)$ *is a subfield of* $\mathbb{L}$ *if the following holds:*

- $1_{\mathbb{L}} \in \mathbb{K}$;

- *$(\mathbb{K},+)$ is a subgroup of $(\mathbb{L},+)$;*
- *$(\mathbb{K}^*,\times)$ is a subgroup of $(\mathbb{L}^*,\times)$.*

Problem I.14. *Let $\mathbb{K}[x]$ be the ring of polynomials in x over $\mathbb{K}$, see also Problem I.11. Show that $\mathbb{K}(x) = \left\{\frac{P}{Q} \middle| P, Q \in \mathbb{K}[x], Q \neq 0\right\}$ is a field, called the* field of rational fractions.

Problem I.15. *Show that a ring homomorphism f from a field $\mathbb{K}$ to any ring R is necessarily injective, that is* $\ker f = \{0_{\mathbb{K}}\}$.

Problem I.16. *Show that the sets of matrices*

$$\mathcal{R}_1 = \left\{\begin{pmatrix} a & b \\ b & a+b \end{pmatrix}, a, b \in \mathbb{Z}/2\mathbb{Z}\right\}, \quad \mathcal{R}_2 = \left\{\begin{pmatrix} a & b \\ b & a \end{pmatrix}, a, b \in \mathbb{Z}/2\mathbb{Z}\right\}$$

are commutative subrings of $(\mathbb{Z}/2\mathbb{Z})^{2\times 2}$. Are they fields?

A.4. Vector spaces

Definition A.20 (Vector space). *A set $(E,+,\cdot)$ is a vector space over a field $(\mathbb{K},+,\times)$ (also called $\mathbb{K}$-vector space) if the following holds, for all $(\alpha,\beta) \in \mathbb{K}\times\mathbb{K}$, and $(u,v) \in E \times E$:*

- *$(E,+)$ is a group with identity element 0_E;*
- *$\alpha \cdot (u+v) = \alpha \cdot u + \alpha \cdot v$;*
- *$(\alpha+\beta)\cdot v = \alpha \cdot v + \beta \cdot v$;*
- *$(\alpha \times \beta)\cdot v = \alpha \cdot (\beta \cdot v)$;*
- *$1_{\mathbb{K}} \cdot v = v$.*

Elements of the field $\mathbb{K}$ will be called scalars *while those of E will be called* vectors.

Definition A.21. *Let E be a $\mathbb{K}$-vector space. One says that elements $v_1, \dots, v_\ell$ are* linearly dependent *if there exist $\alpha_1, \dots, \alpha_\ell$ in $\mathbb{K}$, not all zero, which satisfy*

$$\alpha_1 v_1 + \cdots + \alpha_\ell v_\ell = 0_E.$$

Vectors of the above form are called linear combinations of $v_1, \dots, v_\ell$.

Lemma A.22. *We reuse the notations introduced above. The vectors $v_1, \dots, v_\ell$ in E are linearly dependent if either $v_1 = 0_E$ or for some r, v_r is a linear combination of $v_1, \dots, v_{r-1}$.*

Definition A.23. *Let E be a $\mathbb{K}$-vector space. A subset $\mathcal{F}$ of E is said to span E if for any $v \in E$, there exist $\alpha_1, \dots, \alpha_n$ in $\mathbb{K}$ and there exist $v_1, \dots, v_n$ in $\mathcal{F}$ such that*

$$v = \alpha_1 v_1 + \cdots + \alpha_n v_n.$$

We say that $\mathcal{F}$ forms a basis of E if all elements of $\mathcal{F}$ are linearly independent. The cardinality of a basis of E is called the dimension *of the vector space E. A vector space with a finite basis is called finite-dimensional (otherwise we say that F is infinite-dimensional); the cardinality of such bases is unique.*

Equipped with these notions, one can define linear maps and in the case of maps between finite dimensional vector spaces, one can represent them with matrices.

Proposition A.24. *Let $(\mathbb{L}, +, \times)$ be a field and $(\mathbb{K}+, \times)$ be a subfield of $\mathbb{L}$. Then, $\mathbb{L}$ is a $\mathbb{K}$-vector space.*

Example A.25. *$\mathbb{C}$ is a $\mathbb{R}$-vector space of dimension 2 and $\mathbb{R}$ is a $\mathbb{Q}$-vector space of infinite dimension.*