A First Course in Computer Algebra

Notes for the M1 Course on Computer Algebra at ENS de Lyon

Bruno Salvy and Gilles Villard

Version: March 28, 2025. Please report bugs to Bruno.Salvy@inria.fr

Contents

Lecture 1

Introduction

Computer Algebra is the art of using a computer to perform exact mathematical computations. The aim of this course is to cover the fundamental algorithms of this field (from the Fast Fourier Transform to Gröbner bases), while providing the students with a practical familiarity with its uses and applications through tutorials using Maple. After this course, no student will think of using pen-and-paper for long mathematical derivations.

The questions on the theoretical side are: "what can we compute *exactly*?" and "how fast?" (in terms of complexity estimates). A related question specific to computer algebra is "how big is the result?".

On the practical side, computer algebra has led to the development of systems with several million users, the most famous ones being Mathematica, Maple, SageMath, Pari-GP. On the theoretical side, this area has known more than 50 years of algorithmic progress of which a selection will be presented in this course[1].

1.1 Effectivity

1.1.1 Logic vs Symbolic Computation

An important result that limits the possibilities of computer algebra is an undecidability theorem.

Theorem 1.1 (Richardson-Matiyasevich)**.** *In the class of expressions built from one variable x and the constant 1 with the operations $+, -, \times$ and composition with the functions* $\sin(\cdot)$ *and absolute value $|\cdot|$, recognizing 0 is undecidable.*

Here, 'recognizing 0' means detecting that an expression built with these rules is the function that is identically 0. That it is undecidable means that no algorithm can really 'simplify', unless either a more precise meaning is given to the word, or one restricts the class of expressions to be addressed. A large part of this course consists in taking the second path.

This theorem will not be proved here, but we refer the interested reader to the nice (and short) book by Yuri Matiyasevich entitled "Hilbert's Tenth Problem", where he explains the solution of this problem that got him the Fields medal, and some of its applications.

1.1.2 Data Structures for Mathematical Objects

In practice, many mathematical objects can be represented in a simple way that allows 0-recognition. This is the case for machine integers, that are simply elements of $\mathbb{Z}/2^{32}\mathbb{Z}$ or $\mathbb{Z}/2^{64}\mathbb{Z}$. Then by taking arrays of coefficients, one gets integers of arbitrary size, where an array $(a_0, \ldots, a_k)$ represents

$$a_0 + a_1 B + \cdots + a_k B^k,$$

[1] The date of birth of the field can be set to 1966 when a first conference in this area took place in Washington. It was called Symsac for "Symbolic and Algebraic Computation".

B being a basis, usually a power of 2 such as 2^{32} (the sign can be stored separately). With arrays, one can also construct vectors and matrices; polynomials (for which fast algorithms will be described in Lectures 2,3,4); truncated power series (Lecture 3) and fractions or rational functions (addition of two fractions A/B and C/D does not need a gcd, and a sum is 0 if and only if its denominator is 0). Thus, one can for instance construct matrices of rational functions in several variables, and zero-recognition propagates through these algebraic constructions.

1.1.3 Equations as a Data-Structure

An important idea of computer algebra is that it is not necessarily useful to 'solve' an equation (assuming that this has a meaning), but instead one can think of an equation as a data-structure for its solutions. Algorithms will then compute properties of these solutions (test equality, or compute series expansions, or evaluate numerically,...) from this data-structure.

Example 1.1. The identity

$$\frac{\sin\frac{2\pi}{7}}{\sin^2\frac{3\pi}{7}} - \frac{\sin\frac{\pi}{7}}{\sin^2\frac{2\pi}{7}} + \frac{\sin\frac{3\pi}{7}}{\sin^2\frac{\pi}{7}} = 2\sqrt{7}$$

has a simple proof using the fact that $\exp(i\pi/7)$ is a root of x^7+1 and resultants; this will be seen in Lecture 5.

Example 1.2. The identity $\sin^2+\cos^2=1$ has a simple proof using the fact that both sin and cos are solutions of the linear differential equation $y''+y=0$. This will be presented in Lecture 6.

Example 1.3. The same principle that gives an algorithm for $\sin^2+\cos^2=1$ lets one prove much more complicated formulas, such as Mehler's formula for the Hermite polynomials

$$\sum_{n=0}^{\infty} H_n(x)H_n(y)\frac{u^n}{n!} = \frac{\exp\left(\frac{4u(xy-u(x^2+y^2))}{1-4u^2}\right)}{\sqrt{1-4u^2}},$$

starting from the linear recurrence

$$H_{n+2} = 2x(n+1)H_{n+1} - 2(n+1)H_n, \quad H_0=1, H_1=2x,$$

which can be seen as a definition of, or a data-structure for, the Hermite polynomials.

Example 1.4. Recent records of computation of π rely on the formula

$$\frac{1}{\pi} = \frac{12}{C^{3/2}}\sum_{n=0}^{\infty}\frac{(-1)^n(6n)!(A+nB)}{(3n)!n!^3C^{3n}},$$

with $A=13591409$, $B=545140134$, $C=640320$. Fast numerical evaluation of this formula is achieved by recognizing that the summands satisfy a 1st order linear recurrence and exploiting this. (Tutorial 2).

Example 1.5. The following recent formula for $\zeta(3)$

$$\zeta(3) = \sum_{n=1}^{\infty}\frac{1}{n^3} = \frac{5}{2}\sum_{n=1}^{\infty}\frac{(-1)^{n-1}}{\binom{2n}{n}n^3}$$

can be proved automatically thanks to a system of 1st order linear recurrences (this will be discussed in Lecture 9.)

Example 1.6. A large family of examples is provided by *polynomial systems* that can be used to encode questions in robotics, geometry, or graph theory. These will be discussed in Lecture 10.

1.2 Efficiency

It is important to keep in mind that with current computers, 1 sec. is a long computation. For instance, with a good polynomial and integer library, 1 sec. is sufficient to multiply integers with 30,000,000 digits; multiply polynomials of degree 650,000 (with coefficients that are integers modulo a small p); multiply two matrices of size 850×850 (again with integers modulo a small p for entries). Thus 1 sec. is in the asymptotic regime of the algorithms.

There are many situations that call for such large objects. A spectacular recent example comes from a work of Bostan and Kauers in 2010 where they showed the existence of a polynomial cancelling a certain power series from combinatorics. The polynomial has degree roughly 45 in 3 variables, with coefficients that are integers with up to 25 digits. Intermediate computations involve series expansion at order 1000, conjecturing a linear differential equation with 1.5 billion coefficients (the algorithm will be given in Lecture 8) and an automatic proof involving very large resultants (Lecture 5).

1.2.1 $O()$ notation

We recall the standard notation for asymptotics that will be used in complexity estimates:

$$\begin{aligned} f(n) \sim g(n) \quad &\text{means} \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = 1; \\ f(n) = O(g(n)) \quad &\text{means} \quad \exists K \exists M \forall n \geq M, |f(n)| \leq K g(n). \end{aligned}$$

Example 1.7. Two things must be remembered: $O()$ is only an upper bound; constant factors that might be of importance in practice are erased by this notation. Thus all the following identities are correct

$$\begin{aligned} \log(2n) &= O(\log n) \\ 10^{10^{10}} n &= O(n) \\ 10^{10^{10}} n + n^2 &= O(n^2) \\ n + n^2 &= O(n^{20}). \end{aligned}$$

Figure 1.1 taken from the book by Moore and Mertens (2011) is important to keep in mind the orders of magnitude implied by complexity estimates.

It is often claimed that Moore's law (that the speed of computers doubles every two years) has an important impact on computing, but this is only the case for algorithms with linear complexity. In the figure, doubling the speed amounts to a small vertical shift. Any algorithmic progress is thus likely to have a more important practical impact than doubling the speed of a computer.

1.2.2 $\tilde{O}()$ notation

In computer algebra, it is also common to make use of $\tilde{O}$ in complexity estimates. If $g \to \infty$, then $f = \tilde{O}(g)$ means that there exists $p \geq 0$ such that $f = O(g \log^p g)$. A function $f(n)$ is called *quasi-linear* if $f = \tilde{O}(n)$. For instance, later lectures will show that for polynomials of degree at most n or for integers of at most n bits (or digits), the following operations have quasi-linear complexity: multiplication, division, square-root (in power series for the case of polynomials), gcd.

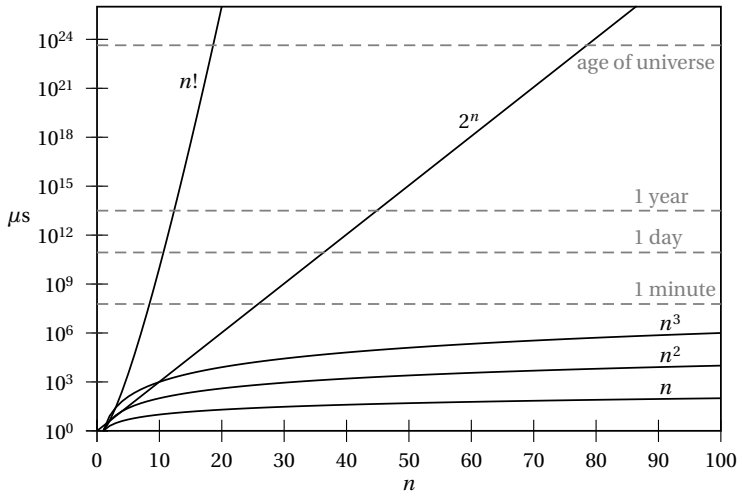


Figure 1.1: Comparisons of orders of complexity

1.2.3 Complexity Model

** This will not be used later in the course and can be skipped. **

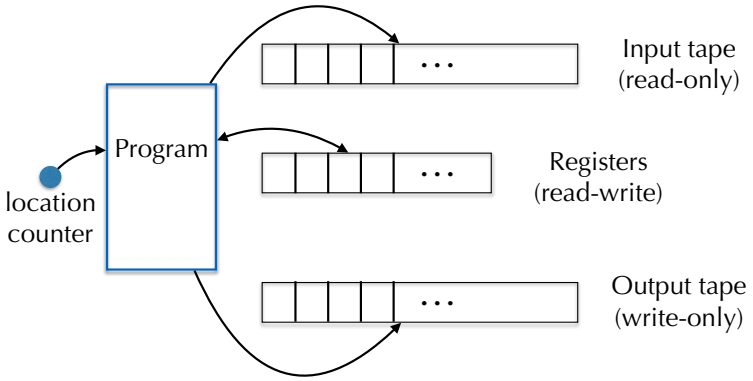


Figure 1.2: Random Access Machine

For definiteness, we give a precise explanation of what the complexity estimates will mean. The model of computer we use in this course is the *Random Access Machine*, described in Fig. 1.2.

The machine has access two three tapes: a read-only tape that contains the input, a write-only tape where it writes the output and a read-write tape that models the memory, storing intermediate computations in cells called registers. All computations take place in the first register, called the accumulator. The program consists of a list of instructions restricted to the set

read, write, load, store, jump, jgtz, jzero, halt, $+, -, \times, \div$.

This set of instructions is sufficient to model real computers. They are used respectively to read on the input tape, write on the output tape, load a register in the accumulator, store the content of the accumulator in a register, jump to a location in the program, jump if the content of the accumulator is greater than zero, jump if it is 0, halt the program, perform one of the arithmetic operations between the content of the accumulator and the content of the given register and store the result in the accumulator. Since the program is not stored in memory, it is a feature of this model that the program cannot modify itself. Each instruction

Problem	Input	Simple Lower Bound	Best known algorithm	Measure
Sorting	n elts	n	O(n log n)	comparisons
Polynomial multiplication	degree n	n	O(n log n)	ops on coeffs
Matrix multiplication	size n x n	n^2	$O(n^{2.373})$	ops on coeffs
Subset sum	n integers	n	$2^{O(n)}$	time

Table 1.1: Simple Lower Bounds for Basic Problems

is assumed to have unit cost. The complexity of a program is thus the number of steps that it performs before halting.

This model leads to two different complexity models depending on what the cells of the tapes are allowed to store. If they hold integers of bounded size, the model is called a *bit complexity*. This reflects closely the computation time, but does not allow for much abstraction in the analyses. We will often make use of the situation where the cells are allowed to store elements of a ring $\mathbb{A}$. That model is called the *algebraic complexity model* and the complexity is expressed in numbers of arithmetic operations in $\mathbb{A}$. It is often called the *arithmetic complexity* of the algorithm. (It will be assumed that the divisions, if any, are defined in $\mathbb{A}$.)

1.2.4 Lower Bounds

By definition, a *lower bound* on the complexity of a *problem* is an upper bound on the complexity of the best possible algorithm for this problem, even if it is as yet unknown. Since the RAM model above counts reading the input and writing the output as unit cost, a simple lower bound on the complexity of algorithms is

$$\text{size(Input)} + \text{size(Output)} \leq \text{complexity}\,.$$

Although very crude, this bound is often quite good, as shown by Table 1.1.

The algorithm for polynomial multiplication will be given in Lecture 2; algorithms with complexity better than the naïve cubic bound for matrix multiplication will be presented in Lecture 7. The subset sum problem is NP complete. This type of problems will not appear in this course.

1.3 Divide-and-Conquer

As divide-and-conquer is a very important paradigm of algorithmic design, we gather here the basic results that will be used in their analysis.

The general pattern of a divide-and-conquer algorithm is illustrated by Fig. 1.3. An input of size n is split into m inputs of size at most n/p on which the algorithm is called recursively. When the size becomes smaller than a threshold s (often equal to 1), another method of computation is used.

The complexity $C(n)$ of the execution of such an algorithm on an input of size n will obey an inequality of the form

$$C(n) \leq mC(\lceil n/p \rceil) + f(n),$$

where $f(n)$ denotes the cost of splitting the input and of recombining the results of the recursive calls. General methods allow to conclude from such an inequality that the complexity has the form $C(n) = O(g(n))$ for a simple function g.

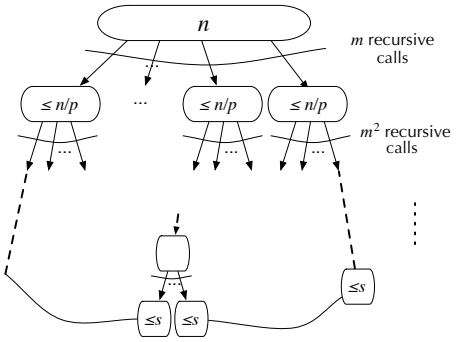


Figure 1.3: Divide and Conquer

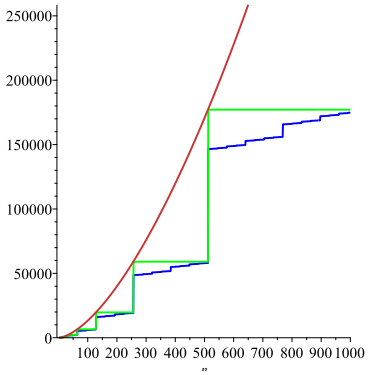


Figure 1.4: Complexity Bounds for Karatsuba's Algorithm

1.3.1 Example: Karatsuba's Algorithm

This algorithm, that will be presented in the next lecture, has a complexity that obeys the inequality

$$C(n) \leq 3C(\lceil n/2 \rceil) + \lambda n.$$

Using this inequality with n replaced by $\lceil n/2 \rceil$ implies

$$C(\lceil n/2 \rceil) \leq 3C(\lceil \lceil n/2 \rceil /2 \rceil + \lambda \lceil n/2 \rceil.$$

In order to simplify the analysis we introduce a notation for iterated ceilings:

$$\lceil x/2 \rceil_1 = \lceil x/2 \rceil, \quad \lceil x/2 \rceil_{k+1} = \lceil \lceil x/2 \rceil_k /2 \rceil.$$

With this notation, injecting the second inequality above into the first one gives

$$C(n) \leq \lambda n + 3\lambda \lceil n/2 \rceil + 9C(\lceil n/2 \rceil_2).$$

Iterating this process $k-1$ times leads to

$$C(n) \leq \lambda N \left(1 + \frac{3}{2} + \cdots + \left(\frac{3}{2}\right)^{k-1}\right) + 3^k C(\lceil n/2 \rceil_k),$$

with N the power of 2 such that $n \leq N < 2n$. Reordering the sum in order to exhibit a convergent geometric series gives

$$C(n) \leq \lambda N \left(\frac{3}{2}\right)^{k-1} \left(1 + 2/3 + \cdots + (2/3)^{k-1}\right) + 3^k C(\lceil n/2 \rceil_k).$$

Bounding the geometric series, this simplifies to

$$C(n) \leq 3^k \left(2\lambda \frac{N}{2^k} + C(\lceil n/2 \rceil_k) \right).$$

Finally, using $k = \lceil \log_2 n \rceil$ gives

$$C(n) \leq (2\lambda + 1) 3^{\lceil \log_2 n \rceil} = O(n^{\log_2 3}).$$

The curves corresponding to the solution of $C(n) = 3C(\lceil n/2 \rceil) + n$, to $(2\lambda+1)3^{\lceil \log_2 n \rceil}$ and to $n^{\log_2 3}$ appear in red, green and blue in Fig. 1.4.

1.3.2 Master Theorem

The analysis of the complexity of Karatsuba's algorithm generalizes and leads to a convenient result, that admits numerous variants.

Theorem 1.2 ('Master Theorem of Divide-and-Conquer'). *Assume*

$$C(n) \leq mC(\lceil n/p \rceil) + f(n), \quad n \geq p,$$

with $f(n)$ and increasing function such that there exist (q, r) obeying

$$q \leq f(pn)/f(n) \leq r,$$

for large enough n. Then, as $n \to \infty$

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

The most important case is when $f(n) = cn^\alpha$ with $\alpha \geq 1$, and then $q = r = p^\alpha$. This is what happens with Karatsuba's algorithm above, where $f(n) = \lambda n$, $q = r = 2$, $m = 3$. The more general version with an increasing function will often be used with $f(n) = \mathsf{M}(n)$, the cost of polynomial multiplication, introduced in Lecture 2.

Proof. The proof follows the steps of the example of Karatsuba's analysis, starting from

$$C(n) \leq mC(\lceil n/p \rceil) + f(n).$$

Iterating once shows

$$C(n) \leq f(n) + mf(\lceil n/p \rceil) + m^2 C(\lceil n/p \rceil_2)$$

with the same notation for iterated ceilings as above. Let N be the power of p such that

$$n \leq N < pn,$$

then since f is increasing we obtain

$$C(n) \leq f(N) + mf(N/p) + m^2 C(\lceil n/p \rceil_2).$$

The hypothesis on f shows that this is bounded by

$$C(n) \leq f(N)(1 + m/q) + m^2 C(\lceil n/p \rceil_2).$$

Iterating $k-1$ times gives

$$C(n) \leq f(N)(1 + m/q + \cdots + (m/q)^{k-1}) + m^k C(\lceil n/p \rceil_k).$$

Choosing $k = \log_p(N)$ then gives

$$C(n) \leq f(N)(1 + m/q + \cdots + (m/q)^{k-1}) + O(N^{\log_p m}),$$

since $m^{\log_p N} = N^{\log_p m}$. The geometric series is bounded in different ways depending on whether $m < q$, $m = q$ or $m > q$, giving the bounds

$$O(1), \quad \log_p N, \quad O(N^{\log_p(m/q)}).$$

The factor $f(N)$ is bounded by $O(f(n))$ thanks to the second inequality on f, using

$$N < pn \Rightarrow f(N) \leq rf(n) = O(f(n)).$$

It remains to show that the last summand $O(N^{\log_p m})$ is smaller than the estimate of the theorem.

Lemma 1.1. *With the hypotheses of the theorem,*

$$n^{\log_p q} = O(f(n)).$$

Proof. This follows from repeating the inequality of the theorem:

$$f(n) \geq qf(n/p) \geq q^2 f(n/p^2) \geq \cdots \geq q^{\log_p n} f(1).$$

The last term is $n^{\log_p q} f(1)$, whence the bound. □

The bound of the lemma allows to conclude directly when $m \leq q$. If $m > q$, then the conclusion comes from

$$n^{\log_p m} = n^{\log_p(m/q)} n^{\log_p q} = n^{\log_p(m/q)} O(f(n)) = O(f(n) n^{\log_p(m/q)}).$$ □

Additional bibliography

More detailed versions of a large part of this course, sometimes at a more advanced mathematical level, are given in

Alin Bostan et al. *Algorithmes Efficaces en Calcul Formel*. Auto-édition, Sept. 2017. ISBN: 979-10-699-0947-2. URL: https://hal.archives-ouvertes.fr/AECF/

In English, the most classical reference for this course is

Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. 3rd edition. New York: Cambridge University Press, 2013, pp. xiv+785. URL: http://www.cambridge.org/fr/knowledge/isbn/item1170826

It is very complete and more advanced than what is needed in a first introduction. It can be used for most of the course, except Lecture 6 and a few topics that are discussed in some of the lectures but not in that book.

Apart from its more elementary level, an important feature of this course that distinguishes it from the references above is the choice to illustrate many of the algorithms by actual Maple code rather than pseudo-code. This makes the algorithms more concrete and makes the presentation more complete. Contrary to pseudo-code, in actual code, no aspect can be swept under the carpet by an imprecise statement. Also, the course is complemented by tutorials in Maple, absent from these notes, whose aim is to make the students fluent in that language and see how the algorithms introduced in the lectures can be put to use in 'concrete' situations.

Lecture 2

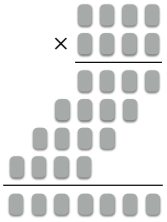
Fast Multiplication

Summary

While naïve multiplication has quadratic complexity, it is actually possible to multiply polynomials much faster. The first algorithm to beat the naïve one has been Karatsuba's algorithm, which is the most efficient one for inputs of intermediate size. For inputs of large size, the fast Fourier transform reaches almost linear complexity.

Introduction

The traditional method of multiplication for two n-digit integers computes one integer of n or $n+1$ digits for each nonzero digit of one of the multiplicands and then adds them up. This can be depicted as follows:



Each of the n intermediate integers is computed in n digit-by-digit multiplications plus $O(n)$ carry computations. The final addition uses $O(n^2)$ digit additions and again $O(n)$ carry computations. In total, the number of digit operations is $O(n^2)$. This is what is called a *quadratic* algorithm. The complexity $O(n^2)$ is also the number of *bit* operations, rather than *digit* operations, since each digit operation uses a constant number of bit operations. Note that the key to using digits in our number representation is that in terms of the numbers themselves, integers smaller than $N > 0$ are represented with $O(\log N)$ bits or digits and therefore this algorithm is only quadratic in the *logarithm* of the quantities involved.

The same reasoning applies to polynomial multiplication. If P and Q are univariate polynomials of degree n_1 and n_2, they have at most n_1+1 and n_2+1 coefficients. Distributing the product leads to $(n_1+1)(n_2+1)$ coefficient multiplications, followed by $O(n_1 n_2)$ additions of coefficients. Thus if both degrees are smaller than n, the algorithm has quadratic complexity $O(n^2)$ again. The situation for polynomials is simpler than that for integers as no carry needs to be propagated. It is a general theme of this course that algorithms for polynomials are usually simpler than their counterparts for integers. We will therefore focus on the polynomial algorithms, that are sufficient to display the ideas of the algorithms.

2.1 Karatsuba's Algorithm

Karatsuba's algorithm is based on a divide-and-conquer method. By itself, divide-and-conquer is not sufficient to beat the quadratic method: a natural attempt would be the algorithm of Fig. 2.1.

1. If $n = 1$ return FG
2. Let $k := \lceil n/2 \rceil$
3. Split $F = F_0 + x^k F_1, G = G_0 + x^k G_1$
   F_0, F_1, G_0, G_1 of degree $< k$
4. Compute recursively
   $H_0 := F_0 G_0$, $H_1 := F_0 G_1$, $H_2 := F_1 G_0$, $H_3 := F_1 G_1$
5. Return $H_0 + x^k(H_1 + H_2) + x^{2k} H_3$

Figure 2.1: A quadratic divide-and-conquer multiplication algorithm

Let $C(n)$ be the complexity for the multiplication by this algorithm in terms of the number of arithmetic operations on the coefficients. If $n > 1$, no operation on coefficients occur in step 3: splitting reduces to copying coefficients; in step 4 there are at most $4C(\lceil n/2 \rceil)$ operations due to the recursive calls; and finally step 5 uses $O(n)$ additions of coefficients. Thus,

$$C(n) \leq 4C(\lceil n/2 \rceil) + \lambda n.$$

By the 'master theorem' of the previous lecture or by a simple direct derivation, it follows that $C(n) = O(n^2)$.

As a first introduction to Maple code, here is the relatively straightforward Maple implementation of the previous pseudo-code:

```
naivedacmult:=proc(F,G,n,x)
local k,f0,f1,g0,g1,h0,h1,h2,h3;
  if n=1 then return F*G fi;
  k:=ceil(n/2);
  # split into quotient and remainder
  f1:=quo(F,x^k,x,'f0');
  g1:=quo(G,x^k,x,'g0');
  # recursive calls
  h0:=thisproc(f0,g0,k,x);
  h1:=thisproc(f0,g1,k,x);
  h2:=thisproc(f1,g0,k,x);
  h3:=thisproc(f1,g1,k,x);
  # recombination
  expand(h0+x^k*(h1+h2)+x^(2*k)*h3)
end:
```

One way of obtaining a better complexity is by looking first at polynomials of degree 1,

$$F = f_0 + f_1 T, \quad G = g_0 + g_1 T, \quad H = FG = h_0 + h_1 T + h_2 T^2.$$

The naive algorithm corresponds to writing

$$H = f_0 g_0 + (f_0 g_1 + f_1 g_0) T + f_1 g_1 T^2.$$

It uses 4 multiplications and 1 addition of coefficients.

Since the result has degree 2, it can also be reconstructed by interpolation from 3 values. It is convenient to take $0, 1$ and ∞ for these values, with the convention that ∞ governs the leading coefficients of the

polynomials. Looking at those values gives

$$h_0 = F(0)G(0) = f_0 g_0,$$
$$h_2 = \text{“}F(\infty)G(\infty)'' = f_1 g_1,$$
$$\tilde{h}_1 = h_0 + h_1 + h_2 = F(1)G(1) = (f_0 + f_1)(g_0 + g_1).$$

Then FG is reconstructed by

$$FG = h_0 + (\tilde{h}_1 - h_0 - h_2)T + h_2 T^2.$$

This computation costs only 3 multiplications (one at each of $0, 1, \infty$), 2 additions and 2 subtractions. While the total number of operations has thus increased (from 5 to 7), the key is that the number of multiplications has decreased and that this idea can be used recursively, with $T = x^k$ and f_i and g_i becoming polynomials of degree smaller than k. The resulting algorithm is given in Fig. 2.2.

1. If n is small, use naive multiplication
2. Let $k := \lceil n/2 \rceil$
3. Split $F = F_0 + x^k F_1, G = G_0 + x^k G_1$
   F_0, F_1, G_0, G_1 of degree $< k$
4. Compute recursively
   $H_0 := F_0 G_0$, $H_2 := F_1 G_1$, $\tilde{H}_1 := (F_0 + F_1)(G_0 + G_1)$
5. Return $H_0 + x^k(\tilde{H}_1 - H_0 - H_2) + x^{2k} H_2$

Figure 2.2: Karatsuba's multiplication algorithm

Its Maple implementation would look like the code below, which is only given for pedagogical purpose, as Maple has its own much faster multiplication implemented in C in its kernel.

```
# Input/Output: same as before
# but now in O(n^1.58) ops.
macro(TK=15): # Threshold for other algorithm
karamult:=proc(F,G,n,x)
local k,f0,f1,g0,g1,h0,h1,h2;
        if n<TK then return expand(F*G) fi;
        k:=ceil(n/2);
        # split into quotient and remainder
        f1:=quo(F,x^k,x,'f0');
        g1:=quo(G,x^k,x,'g0');
        # 3 recursive calls
        h0:=thisproc(f0,g0,k,x);
        h2:=thisproc(f1,g1,k,x);
        h1:=thisproc(f0+f1,g0+g1,k,x);
        # recombination
        expand(h0+x^k*(h1-h0-h2)+x^(2*k)*h2)
end:
```

The complexity analysis is very similar to that of the quadratic divide-and-conquer algorithm, except that now there are only three recursive calls, leading to

$$C(n) \leq 3C(\lceil n/2 \rceil) + \lambda n.$$

The complexity analysis was detailed in Lecture 1. It leads to

$$C(n) = O(n^{\log_2 3}).$$

Since $\log_2 3 \simeq 1.58$, this is a big improvement over the quadratic algorithm.

2.2 Faster Polynomial Multiplication

The starting point of Karatsuba's algorithm above is the reconstruction of degree 2 polynomials from 3 values. A generalized form of this follows from considering evaluation and interpolation in terms of linear maps.

2.2.1 Evaluation and Interpolation

Given $k+1$ points $a_0, \ldots, a_k$ and a polynomial $P = p_0 + p_1 x + \cdots + p_k x^k$ of degree at most k, the simultaneous evaluation of P at $a_0, \ldots, a_k$ is given by the matrix-vector product

$$\begin{pmatrix} P(a_0) \\ \vdots \\ P(a_k) \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & a_0 & \cdots & a_0^k \\ & \cdots & \cdots & \\ 1 & a_k & \cdots & a_k^k \end{pmatrix}}_{V} \begin{pmatrix} p_0 \\ \vdots \\ p_k \end{pmatrix}.$$

The matrix V in this equation is known as the *Vandermonde* matrix.

Lemma 2.1. $\det V = \prod_{i<j}(a_j - a_i)$.

Proof. By the expansion of determinants as sums over permutations, the determinant of V is a polynomial in $\mathbb{Z}[a_0, \ldots, a_k]$, whose total degree is at most $1+2+\cdots+k = k(k+1)/2$. If two a_is are equal then two rows of the matrix are equal, making the determinant vanish. This implies that $a_j - a_i$ divides the determinant for all $i < j$. The product of all these polynomials has degree $k(k+1)/2$. Thus this product differs from the determinant by at most a constant factor. The coefficient of $a_0^0 a_1^1 \cdots a_k^k$ in the determinant is 1 (those are the elements along the diagonal); it is also 1 in the product (to obtain a_k^k one has to take a_k in all $a_k - a_i$ and then to obtain a_{k-1}^{k-1} one has to take a_{k-1} in all $a_{k-1} - a_i$, $i < k-1$ and so on.). □

Interpolation is the operation of constructing a polynomial P of degree $\leq k$ from its values $(v_0, \ldots, v_k)$ at $(a_0, \ldots, a_k)$. It follows from the invertibility of the Vandermonde matrix when the a_i are distinct elements of a field $\mathbb{K}$ (by the lemma), that there is a unique such polynomial, whose coefficients are given by

$$V^{-1} \begin{pmatrix} v_0 \\ \vdots \\ v_k \end{pmatrix}.$$

2.2.2 A Karatsuba-like algorithm

Any three distinct points lead to a Karatsuba-like algorithm. For instance, with $(a_0, a_1, a_2) = (-1, 0, 1)$, one gets the following Vandermonde matrix and its inverse

$$V = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}, \qquad V^{-1} = \frac{1}{2}\begin{pmatrix} 0 & 2 & 0 \\ -1 & 0 & 1 \\ 1 & -2 & 1 \end{pmatrix}.$$

In terms of polynomials, the matrix V^{-1} implies that for any polynomial P of degree at most 2,

$$P(x) = \frac{1}{2}\left(2P(0) + (P(1) - P(-1))x + (P(1) - 2P(0) + P(-1))x^2\right).$$

By the same reasoning as before, this leads to a multiplication algorithm when used recursively, given in Fig. 2.3.

1. If n is small, use naive multiplication
2. Let $k := \lceil n/2 \rceil$
3. Split $F = F_0 + x^k F_1, G = G_0 + x^k G_1$, with $\deg F_i, G_i < k$
4. Compute recursively
$H_0 := F_0 G_0$, $\tilde{H}_1 := (F_0 + F_1)(G_0 + G_1)$, $\tilde{H}_{-1} := (F_0 - F_1)(G_0 - G_1)$
5. Return $\frac{1}{2}\left(2H_0 + x^k(\tilde{H}_1 - \tilde{H}_{-1}) + x^{2k}(\tilde{H}_1 - 2H_0 + \tilde{H}_{-1})\right)$

Figure 2.3: A variant of Karatsuba's multiplication algorithm

This variant is not as good as Karatsuba's algorithm: it requires a division by 2 and slightly more operations. Nonetheless, its complexity analysis runs exactly as before, leading to

$$C(n) \leq 3C(\lceil n/2 \rceil) + \lambda n$$

and therefore again

$$C(n) = O(n^{\log_2 3}).$$

The complexity exponent is the same as in Karatsuba's algorithm.

2.2.3 Higher degree

The same idea extends. Using the 5 points $(a_0, \ldots, a_4) = (-2, \ldots, 2)$ gives the matrices

$$V = \begin{pmatrix} 1 & -2 & 4 & -8 & 16 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \end{pmatrix}, \qquad V^{-1} = \frac{1}{24} \begin{pmatrix} 0 & 0 & 24 & 0 & 0 \\ 2 & -16 & 0 & 16 & -2 \\ -1 & 16 & -30 & 16 & -1 \\ -2 & 4 & 0 & -4 & 2 \\ 1 & -4 & 6 & -4 & 1 \end{pmatrix}.$$

With 5 points, one can reconstruct polynomials of degree 4. Thus if F and G have degree at most 2, the

Algorithm:
1. For small degrees, use simpler algorithm
2. Split F and G into 3 parts
3. Compute 5+5 linear combinations of these parts
4. Compute 5 products recursively
5. Recover the result by linear combinations

Figure 2.4: Multiplication algorithm with 5 recursive calls

algorithm proceeds by evaluating F and G at $0, \pm 1, \pm 2$ which amounts to computing linear combinations of their coefficients; pairwise multiplications of the results; interpolation from these 5 values by linear combinations given by V^{-1}. This in turn gives the recursive algorithm of Fig. 2.4, which is possibly clearer in the more explicit Maple version below.

```
Toom3:=proc(F,G,n,x)
local k,f0,f1,f2,g0,g1,g2,h0,h1,h2,hm1,hm2;
  if n<TK then return expand(F*G) fi;
  k:=ceil(n/3);
  # split into 3 parts
  f2:=quo(F,x^(2*k),x,'f1'); f1:=quo(f1,x^k,x,'f0');
  g2:=quo(G,x^(2*k),x,'g1'); g1:=quo(g1,x^k,x,'g0');
  # 5 recursive calls
  h0:=thisproc(f0,g0,k,x);
  h1:=thisproc(f0+f1+f2,g0+g1+g2,k,x);
  h2:=thisproc(f0+2*f1+4*f2,g0+2*g1+4*g2,k,x);
  hm1:=thisproc(f0-f1+f2,g0-g1+g2,k,x);
  hm2:=thisproc(f0-2*f1+4*f2,g0-2*g1+4*g2,k,x);
  # recombination
  expand(h0
    +x^k*(2*hm2-16*hm1+16*h1-2*h2)/24
    +x^(2*k)*(-hm2+16*hm1-30*h0+16*h1-h2)/24
    +x^(3*k)*(-2*hm2+4*hm1-4*h1+2*h2)/24
    +x^(4*k)*(hm2-4*hm1+6*h0-4*h1+h2)/24)
end:
```

The complexity analysis is very similar. It leads to

$$C(n) \leq 5C(\lceil n/3 \rceil) + \lambda n = O(n^{\log_3 5})$$

coefficient operations. Since $\log_3 5 \simeq 1.46$, this is an improvement over Karatsuba's algorithm.

2.2.4 Toom-Cook algorithm

This principle extends to any $N > 0$. Provided the field $\mathbb{K}$ contains $2N-1$ distinct values $(a_0, \ldots, a_{2N-1})$, one precomputes once and for all the corresponding Vandermonde matrix V and its inverse. Then, given two polynomials of degree smaller than N,

$$F = f_0 + \cdots + f_{N-1}x^{N-1}, \quad G = g_0 + \cdots + g_{N-1}x^{N-1},$$

the values of F and G at $(a_0, \ldots, a_{2N-1})$ are given by linear combinations of the coefficients of F and G, given by the product by V, in at most $(2N-1)^2$ operations. Next, the values of $H = FG$ at these points are obtained by pairwise multiplication in $2N-1$ multiplications. Finally, from these values, the coefficients of H are obtained by multiplication by V^{-1}, again in at most $(2N-1)^2$ operations. Using this process recursively gives the algorithm of Fig. 2.5.

Toom-Cook Algorithm:

1. For small degrees, use simpler algorithm
2. Split F and G into N parts
3. Compute linear combinations of these parts
4. Compute $2N-1$ products recursively
5. Recover the result by linear combinations

Figure 2.5: Toom-Cook algorithm

The same steps of the complexity analysis lead now to

$$C(n) \leq (2N-1)C(\lceil n/N \rceil) + \lambda n.$$

The constant λ depends on N in a quadratic way, as indicated above, but that does not affect the exponent of the complexity, which is given by the 'master theorem' as

$$C(n) = O(n^{\log_N(2N-1)}).$$

As $N \to \infty$, this exponent rewrites

$$\log_N(2N-1) = \frac{\log(2N-1)}{\log N} \leq \frac{\log N + \log 2}{\log N} = 1 + o(1).$$

This means that asymptotically, this method leads to multiplication algorithms that are as close to linear as desired. (More formally, for any $\epsilon > 0$, we deduce an algorithm in complexity $O(n^{1+\epsilon})$).

2.3 Integer multiplication

As mentioned in the introduction, integers behave like polynomials and variants of the algorithms for polynomials adapt to integers. Still, this must be done on a case by case basis, as no theorem of equivalence of complexities is available.

An important tool that lets one transfer a complexity result from the world of integers to that of polynomials is *Kronecker's substitution*. The idea is that a product of two polynomials $F(x)$ and $G(x)$ with coefficients smaller than 2^k can be reconstructed from the integer $F(2^{2k+1})G(2^{2k+1})$. This is best illustrated in base 10: the coefficients of the product of

$$F(x) = 72x + 43 \quad \text{and} \quad G(x) = 51x + 8,$$

i.e., $F(x)G(x) = 3672x^2 + 2769x + 344$ can be read off from the product

$$F(10^5)G(10^5) = 720043 \times 710008 = \underbrace{3672}\,\underbrace{2769}\,\underbrace{0344}.$$

To deal with negative coefficients, one can split F and G into a sum of positive and negative parts.

2.3.1 Karatsuba's algorithm

The algorithm used for multiplication of polynomials requires very little change for integers smaller than 2^n (Fig. 2.6). The starting point is to change X into 2. In order to have the last recursive call on input whose size is halved, it is simpler to use the variant of Karatsuba's algorithm based on the points $(0, -1, \infty)$ and track signs separately.

1. If n is small, use naive multiplication
2. Let $k := \lceil n/2 \rceil$
3. Split $F = F_0 + 2^k F_1, G = G_0 + 2^k G_1$
   $F_0, F_1, G_0, G_1 < 2^k$
4. Compute recursively
   $H_0 := F_0 G_0, \quad H_2 := F_2 G_2, \quad \tilde{H}_1 := (F_0 + F_1)(G_0 + G_1)$
5. Return $H_0 + 2^k(\tilde{H}_1 - H_0 - H_2) + 2^{2k} H_2$

Figure 2.6: Karatsuba's algorithm for integers

Then, this is not exactly the same algorithm as the polynomial one, but the complexity analysis is the same. One gets the same inequality

$$C(n) \leq 3C(\lceil n/2 \rceil) + \lambda n,$$

where now $C(n)$ denotes the number of *bit* operations required to multiply two n-bit long integers. The same $O(n^{\log_2 3})$ complexity estimate therefore holds.

2.3.2 A collection of algorithms

In practice, none of the multiplication algorithms is better than all the other ones all the time. The Gnu Multiprecision Library (GMP) has been the best integer arithmetic library available for many years. It comes with many multiplication algorithms and automatically selects the most appropriate depending on the size of the input. The thresholds it uses are given in Table 2.1.

# 64-bit words	approx # digits	Algorithm
0	**0**	Naive
26	**500**	Karatsuba
73	**1,400**	Toom - 3
208	**4,000**	Toom - 4
4736	**90,000**	FFT

Table 2.1: Length thresholds for switching between multiplication algorithms in GMP

For instance, for integers of about 5,000 decimal digits, the library uses Toom-4 (the variant of Toom-Cook with $N = 4$). This is a recursive algorithm. As soon as the size drops below 4,000 decimal digits, Toom-3 is used (the variant with $N = 3$) and then Karatsuba's algorithm when the input is below 1,400 decimal digits. Finally, in the recursive calls, when the size drops below 500 decimal digits, the naïve quadratic algorithm is used.

For very large sizes, GMP uses the fast Fourier transform (FFT), which is our next topic.

2.4 Discrete Fourier Transform

The Fourier transform is a classical tool of digital signal processing. It decomposes periodic signals into linear combinations of sines and cosines. The FFT (Fast Fourier Transform) performs this decomposition efficiently for discrete signals. It has been listed as one of the top 10 algorithms of the 20th century by the *IEEE Journal of Computing in Science & Engineering*. It is based on special roots of 1, that we first introduce.

2.4.1 Primitive roots of unity

Definition 2.1. *An element ω of a field $\mathbb{K}$ is called an* nth root of unity *if $\omega^n = 1$ (n is called the* order *of the root). It is called* primitive *if moreover*

$$\omega^t \neq 1, \quad t \in \{1, \ldots, n-1\}.$$

Example 2.1. If $\mathbb{K} = \mathbb{C}$, then $\exp(2\pi i/n)$ is a primitive root of unity.

Example 2.2. The number $N = 5 \times 2^{25} + 1$ is a prime. This implies that $\mathbb{K} = \mathbb{Z}/N\mathbb{Z}$ is a field. In this field, it turns out that 17 is a primitive root of unity of order $2^{25} > 3 \times 10^7$. This will have for consequence that polynomials of up to that degree can be multiplied efficiently in $\mathbb{K}$.

The main properties of primitive roots of unity are summarized in the following.

Proposition 2.1. *If ω is a primitive nth root of unity, then*

1. *so is ω^{-1};*
2. *if $n = pq$, then ω^p is a primitive qth root of unity;*
3. *for $\ell \in \{1, \dots, n-1\}$,*

$$\sum_{j=0}^{n-1} \omega^{\ell j} = 0.$$

Proof. 1. First, since $\omega^n = \omega^{n-1}\omega = 1$, it follows that ω invertible, with $\omega^{-1} = \omega^{n-1}$. It is an nth root of unity since $\omega^{-n}\omega^n = \omega^{-n} = 1$. It is primitive since for $t \in \{1, \dots, n-1\}$

$$\omega^{-t} - 1 = -\omega^{-t}(\omega^t - 1) \neq 0.$$

2. ω^p is a qth root of unity: $(\omega^p)^q = \omega^{pq} = \omega^n = 1$. It is primitive: if $(\omega^p)^t - 1$ is 0 for $t \in \{1, \dots, q-1\}$, then so is $\omega^{pt} - 1$ and $pt < n$ shows that this is impossible.

3. Multiplying the sum by $(1 - \omega^\ell)$ gives

$$(1 - \omega^\ell)(1 + \omega^\ell + \cdots + \omega^{(n-1)\ell}) = 1 - \omega^{n\ell} = 0.$$

However, since ω is primitive, $1 - \omega^\ell$ is not 0, which implies that the sum is 0. □

2.4.2 Discrete Fourier Transform

Definition 2.2. *If ω is a primitive nth root of unity in the field $\mathbb{K}$, the* discrete Fourier transform *(DFT) is the map*

$$\begin{aligned} \mathrm{DFT}_\omega : \mathbb{K}[X] &\to \mathbb{K}^n \\ A &\mapsto (A(1), A(\omega), \dots, A(\omega^{n-1})). \end{aligned}$$

In view of Section 2.2.1, this is a linear map, whose matrix is the Vandermonde matrix

$$V_\omega = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{n-1} \\ & \cdots & \cdots & \\ 1 & \omega^{n-1} & \cdots & \omega^{(n-1)^2} \end{pmatrix}.$$

In that case, the inverse of the Vandermonde matrix can be made explicit.

Lemma 2.2. $V_\omega V_{\omega^{-1}} = n \,\mathrm{Id}_n$.

Proof. The entry (i, j) of the product $V_\omega V_{\omega^{-1}}$ is the sum

$$\sum_{k=0}^{n-1} \omega^{ik}\omega^{-jk} = \sum_{k=0}^{n-1} \omega^{(i-j)k}.$$

If $i = j$ all summands equal 1, giving the terms on the diagonal of $n \,\mathrm{Id}_n$. Otherwise, the sum is 0 by the proposition above. □

The practical consequence of this lemma is that an algorithm computing the discrete Fourier transform can also be used to compute its inverse, by using it with ω^{-1} in place of ω, and dividing by n at the end.

2.4.3 Application to signals

The DFT of a signal $(a_0, \dots, a_{n-1})$ is defined to be that of the polynomial $a_0 + \cdots + a_{n-1}X^{n-1}$.

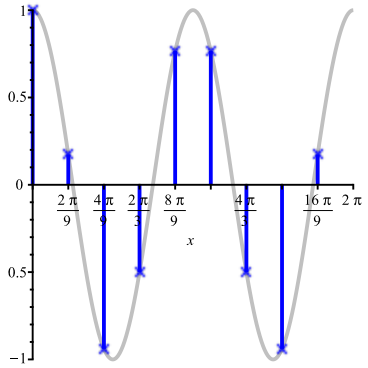


Figure 2.7: Sampling a signal

The key property that DFT recovers frequencies can be seen on an example. We take $n = 9$, $\omega = \exp(-2\pi i/9)$. The signal is $f = \cos(2x)$ and the sample points are $(0, 2\pi/9, 4\pi/9, \dots, 16\pi/9)$ (see Fig. 2.7). The sampled values are

$$\begin{aligned}(a_0, \dots, a_8) &= \left(1, \cos(\frac{4\pi}{9}), \dots, \cos(\frac{32\pi}{9})\right), \\ &= \frac{1}{2}(1, \omega^2, \dots, \omega^{16}) + \frac{1}{2}(1, \omega^{-2}, \dots, \omega^{-16}).\end{aligned}$$

Returning to the definition of DFT_ω, we see that the jth entry of $\mathrm{DFT}_\omega(a_0, \dots, a_8)$ is

$$a_0 + a_1\omega^j + a_2\omega^{2j} + \cdots = \frac{1 + \omega^{2+j} + \omega^{4+2j} + \cdots}{2} + \frac{1 + \omega^{-2+j} + \omega^{-4+2j} + \cdots}{2} = \begin{cases} n/2 & \text{if } j = 2 \text{ or } n-2, \\ 0 & \text{otherwise.} \end{cases}$$

In summary,

$$\mathrm{DFT}_\omega(a_0, \dots, a_8) = 9/2 \times (0, 0, 1, 0, 0, 0, 0, 1, 0).$$

The only nonzero entries are at indices 2 and $n - 2$: the frequency 2 has been recovered from the sample. By linearity, other frequencies appearing in the signal would have been found.

2.5 Fast Fourier Transform

The Fast Fourier Transform algorithm was found by Gauss in 1805, but was not published until much later, in 1866, when it was not seen as important. It was rediscovered in 1965 by Cooley and Tuckey in the US when they were studying seismic signals to detect nuclear tests from Soviet Union.

The principle of the algorithm is a divide-and-conquer approach. It relies on the following.

Lemma 2.3. *If ω is a primitive nth root of unity, with $n = 2k$, then $\omega^k = -1$.*

Proof. Write

$$0 = 1 - \omega^n = (1 - \omega^k)(1 + \omega^k) \tag{2.1}$$

and use the fact that $1 - \omega^k \neq 0$ since ω is primitive. □

The DFT requires the values of a polynomial A at the powers of ω. If this polynomial A has degree less than n, it can first be split into two parts as $A = A_\ell + X^k A_h$, where the indices stand for 'low' and 'high', and both $\deg A_\ell$ and $\deg A_h$ are smaller than k. Next, the polynomial A is written in two different ways

$$A = (A_\ell + A_h) + (X^k - 1)A_h = (A_\ell - A_h) + (X^k + 1)A_h.$$

For any nonnegative integer ℓ, if ℓ is even, then ℓk is a multiple of n and $\omega^{\ell k} = 1$. Conversely, if ℓ is odd, then $\ell k \equiv k \bmod n$ and $\omega^{\ell k} = \omega^k = -1$. Thus, evaluating the identities above at $X = \omega^\ell$ gives

$$A(\omega^\ell) = \begin{cases} R_e(\omega^\ell), & \text{if } \ell \text{ is even,} \\ R_o(\omega^\ell), & \text{otherwise,} \end{cases}$$

where

$$R_e = A_\ell + A_h, \quad R_o = A_\ell - A_h$$

(the indices 'e' and 'o' stand for *even* and *odd*.)

This is the basis of the divide-and-conquer approach: evaluating the polynomial A of degree smaller than n at the n points $(1, \omega, \ldots, \omega^{n-1})$ splits into evaluating the polynomials $R_e(X)$ and $R_o(\omega X)$ of degree smaller than $k = n/2$ at the $n/2$ points $(1, \omega^2, \ldots, \omega^{2(k-1)})$.

Turning this observation into an efficient algorithm requires a good way of computing R_e and R_o. This is achieved by a simple manipulation:

$$\begin{aligned} A(X) &= a_0 + a_1 X + \cdots + a_{n-1} X^{n-1}, \\ &= \underbrace{(a_0 + \cdots + a_{k-1} X^{k-1})}_{Q_\ell} + X^k \underbrace{(a_k + \cdots + a_{n-1} X^{k-1})}_{Q_h} \\ &= (Q_\ell + Q_h) + (X^k - 1)Q_h = R_e + (X^k - 1)Q_h, \\ &= (Q_\ell - Q_h) + (X^k + 1)Q_h = R_o + (X^k + 1)Q_h, \end{aligned}$$

which shows that $R_e = Q_\ell + Q_h$ and $R_o = Q_\ell - Q_h$ are both obtained in k operations each from Q_ℓ and Q_h, whose computation does not require any arithmetic operation.

Input. $A = a_0 + \cdots + a_{n-1}X^{n-1} \in \mathbb{A}[X]$; $(1, \omega, \ldots, \omega^{n-1})$ with ω principal n th root of 1, n power of 2.

Output. $\mathrm{DFT}_\omega(A) = \big(A(1), \ldots, A(\omega^{n-1})\big)$

1. If $n = 1$, return (a_0)
2. Set $k := n/2$ and compute
$$R_e(X) = \sum_{j=0}^{k-1} (a_j + a_{j+k})X^j, \quad S_o(X) := R_o(\omega X) = \sum_{j=0}^{k-1} \omega^j (a_j - a_{j+k})X^j.$$
3. Compute recursively $\mathrm{DFT}_{\omega^2}(R_e), \mathrm{DFT}_{\omega^2}(S_o)$
4. Return $\big(R_e(1), S_o(1), R_e(\omega^2), \ldots, S_o(\omega^{2(k-1)})\big)$

Figure 2.8: Fast Fourier Transform

The FFT algorithm of Fig. 2.8 follows. It requires n to be a power of 2 so that Lemma 2.3 can be used at each of the recursive steps. The computation of R_e and R_o in step 2 comes from the discussion above. The evaluation of S_o at the even powers of ω recovers the evaluation of R_o at the odd ones. A more explicit Maple implementation is as follows.

```
# Input:
#         . A: polynomial in x of degree < n
#         . n: a power of 2
#         . x: variable in A
#         . L: list [1,ω,...,ω^(n-1)] with ω a primitive nth root of 1
# Output:
#         [A(1),...,A(ω^(n-1))]
FastFourierTransform:=proc(A,n,x,L)
local k,j,L2,Re,So;
        if n=1 then return [A] fi;
        k:=n/2;
        Re:=add((coeff(A,x,j)+coeff(A,x,j+k))*x^j,j=0..k-1);
        So:=add((coeff(A,x,j)-coeff(A,x,j+k))*x^j*L[j+1],j=0..k-1);
        L2:=[seq(L[2*j+1],j=0..k-1)];
        Re:=thisproc(Re,k,x,L2);
        So:=thisproc(So,k,x,L2);
        [seq(op([Re[j],So[j]]),j=1..k)]
end:
```

Let $C(n)$ denote the number of arithmetic operations in $\mathbb{K}$ used by this algorithm. Then there are $3k$ operations in step 2 and two recursive calls, leading to

$$C(n) \leq \frac{3n}{2} + 2C(n/2).$$

The 'master theorem' gives $C(n) = O(n \log n)$. A more detailed analysis, proceeding step-by-step, gives more information on the constants involved: by induction, the inequality above leads to

$$C(n) \leq \frac{3n}{2}k + 2^k C(n/2^k).$$

Choosing $k = \log_2 n$ gives

$$C(n) = \frac{3}{2} n \log_2 n + O(n).$$

2.5.1 Multiplication of Polynomials

The principle is to compute the product of polynomials by evaluation and interpolation at roots of unity. Given a fast algorithm for the DFT, this boils down to the algorithm of Fig. 2.9.

Input. F and G in $\mathbb{K}[X]$ with $\deg F + \deg G < n = 2^k$, ω primitive nth root of 1 in $\mathbb{K}$, 2 invertible in $\mathbb{K}$.

Output. $F \times G \bmod (X^n - 1) = F \times G$

1. Compute $\omega^2, \ldots, \omega^{n-1}$.
2. Compute
   $(\hat{f}_0, \ldots, \hat{f}_{n-1}) = \mathrm{DFT}_\omega(F)$, $(\hat{g}_0, \ldots, \hat{g}_{n-1}) = \mathrm{DFT}_\omega(G)$
3. Multiply: $\hat{h} := (\hat{f}_0\hat{g}_0, \ldots, \hat{f}_{n-1}\hat{g}_{n-1})$
4. Return $\frac{1}{n}\mathrm{DFT}_{\omega^{-1}}(\hat{h})$

Figure 2.9: Polynomial Multiplication by Fast Fourier Transform

The final result is the following.

Theorem 2.1. *If n is a power of 2, given a primitive nth root of unity in $\mathbb{K}$, one can compute the product of two polynomials of $\mathbb{K}[X]$ whose degrees sum to less than n in $O(n \log n)$ arithmetic operations in $\mathbb{K}$, assuming 2 is invertible in $\mathbb{K}$.*

In words, multiplication has quasi-linear complexity!

Proof. The algorithm first evaluates F and G at the powers of ω by DFT. Multiplying their values pairwise gives the values of $H = FG$ at the powers of ω. Since $\deg H < n$, it is recovered in a unique way by interpolation, which is computed by one more DFT with respect to ω^{-1} followed by a division by n (invertible by hypothesis), thanks to Lemma 2.2.

In terms of complexity, the first and the third steps use $O(n)$ operations in $\mathbb{K}$. So does the last division by n in step 4. The powers of $\omega^{-1} = \omega^{n-1}$ can also be obtained in the same complexity. Finally, the 3 DFT each have a cost of $3n \log n/2 + O(n)$, whence the result. □

2.5.2 Further FFT-related algorithms

Theorem 2.1 establishes that multiplication of polynomials of degree at most n in $\mathbb{K}[x]$ can be performed in $O(n \log n)$ arithmetic operations in $\mathbb{K}$ when $\mathbb{K}$ has primitive nth roots of unity and 2 is invertible in $\mathbb{K}$. This algorithm goes back to Cooley and Tuckey in 1965. The basic principles of the FFT algorithm have then been extended to other contexts where the hypotheses of Theorem 2.1. First, in 1971, Schönhage and Strassen found a way to multiply two n-bit integers in $O(n \log n \log \log n)$ *bit* operations, which also gives an algorithm of the same complexity for multiplication of polynomials when $\mathbb{K}$ does not have primitive nth roots of unity. The case when 2 is not invertible is important in cryptographic applications. A first extension of FFT when 2 is not invertible but 3 is was given by Shönhage in 1977. Finally, in 1991, Cantor and Kaltofen extended these ideas to multiplication of polynomials of degree at most n in $\mathbb{A}[X]$ when $\mathbb{A}$ is only a ring, still in $O(n \log n \log \log n)$ arithmetic operations.

For integers, the final theoretical improvement is very recent: in 2021, Harvey and van der Hoeven, gave a theoretical algorithm with only $O(n \log n)$ bit complexity. This is not intended to be practical, but it closes the gap between polynomials and integers.

2.6 Multiplication Functions

2.6.1 Summary for Polynomials

Let $\mathrm{Mul}(n)$ be a bound on the number of arithmetic operations in $\mathbb{A}$ needed to multiply two polynomials of degree at most n in $\mathbb{A}[X]$. We have obtained

$$\mathrm{Mul}(n) = \begin{cases} O(n^2) & \text{by the naive algorithm;} \\ O(n^{\log_2 3}) & \text{by Karatsuba's algorithm;} \\ O(n^{\log_k (2k-1)}) & \text{by Toom-Cook's algorithm (}\mathbb{A}\text{large);} \\ O(n \log n) & \text{by FFT (with primitive roots of 1);} \\ O(n \log n \log \log n) & \text{by the Schönhage-Strassen algorithm.} \end{cases}$$

All these complexity estimates satisfy

$$\mathrm{Mul}(n_1) + \mathrm{Mul}(n_2) \leq \mathrm{Mul}(n_1 + n_2), \quad \mathrm{Mul}(mn) \leq m^2 \, \mathrm{Mul}(n).$$

2.6.2 Summary for Integers

Let $\mathrm{Mul}_{\mathbb{Z}}(n)$ be a bound on the number of bit operations needed to multiply two integers of at most n bits. Then, we have obtained

$$\mathrm{Mul}_{\mathbb{Z}}(n) = \begin{cases} O(n^2) & \text{by the naive algorithm;} \\ O(n^{\log_2 3}) & \text{by Karatsuba's algorithm;} \\ O(n^{\log_k(2k-1)}) & \text{by Toom-Cook's algorithm;} \\ O(n \log n) & \text{by FFT (not in the course).} \end{cases}$$

All these complexity estimates satisfy

$$\mathrm{Mul}_{\mathbb{Z}}(n_1) + \mathrm{Mul}_{\mathbb{Z}}(n_2) \leq \mathrm{Mul}_{\mathbb{Z}}(n_1 + n_2), \ \mathrm{Mul}_{\mathbb{Z}}(mn) \leq m^2 \, \mathrm{Mul}_{\mathbb{Z}}(n).$$

2.6.3 Multiplication Functions

Since all these algorithms are used simultaneously in the implementations, it is useful to express the complexity of algorithms relying on multiplication in a way that does not depend on the actual multiplication algorithm that is used. For this purpose, one makes use of multiplication functions.

Definition 2.3. *The function* $\mathsf{M} : \mathbb{N}^\star \to \mathbb{R}^\star$ *is a* multiplication function *for the ring* $\mathbb{A}[X]$ *if*

1. *one can multiply two polynomials of degree* $\leq n$ *in at most* $\mathsf{M}(n)$ *arithmetic operations in* $\mathbb{A}$*;*
2. $\mathsf{M}(n + n') \geq \mathsf{M}(n) + \mathsf{M}(n')$*;*
3. $\mathsf{M}(mn) \leq m^2 \mathsf{M}(n)$.

The second condition implies that the complexity is at least linear, the last one that it is at most quadratic.

Similarly, for integer multiplication, the definition is as follows.

Definition 2.4. *The function* $\mathsf{M}_{\mathbb{Z}} : \mathbb{N}^\star \to \mathbb{R}^\star$ *is a* multiplication function *for* $\mathbb{Z}$ *if*

1. *one can multiply to integers with* $\leq n$ *bits (or digits) in* $\leq \mathsf{M}_{\mathbb{Z}}(n)$ *word operations;*
2. *and 3. as above.*

For future uses of this notion, it is useful to check that these functions satisfy the hypotheses of the 'master theorem' of the previous lecture.

2.7 Example: Multiplication of Bivariate Polynomials

The multiplication of two polynomials

$$F(x, y) = \sum_{i=0}^{n-1} f_i(y) x^i, \quad G(x, y) = \sum_{i=0}^{n-1} g_i(y) x^i,$$

in $\mathbb{K}[x, y]$, with coefficients f_i and g_i of degree smaller than d can be reduced to the multiplication of univariate polynomials so that its complexity can be estimated in terms of the multiplication function M. The idea is called *Kronecker's substitution*. It consists in computing

$$H(x, y) = F(x, y)G(x, y) = \sum_{i=0}^{2n-1} h_i(y) x^i$$

from the univariate product

$$H(x^{2d}, x) = F(x^{2d}, x)G(x^{2d}, x). \tag{2.2}$$

Indeed, this product is

$$H(x^{2d}, x) = h_0(x) + x^{2d}h_1(x) + x^{4d}h_2(x) + \cdots$$

and since all h_i have degree smaller than $2d$, the coefficients of h_i are those of degrees $2id, \ldots, 2id + 2d - 2$ in Eq. (2.2).

The polynomials $F(x^{2d}, x)$ and $G(x^{2d}, x)$ have degree smaller than $2d(n-1) + d \leq 2dn$ and thus the product can be performed in $\mathsf{M}(2dn)$ arithmetic operations in $\mathbb{K}$. Since the result has about $4dn$ coefficients, this is quasi-optimal if FFT is used.

Additional Bibliography

Besides those mentioned in the introduction, useful descriptions can be found in the following three books:

Donald E. Knuth. *The Art of Computer Programming*. 3rd edition. Vol. 2: Seminumerical Algorithms. Computer Science and Information Processing. Reading, Mass.: Addison-Wesley Publishing Co., 1997, pp. xiv+762

Peter Henrici. *Applied and computational complex analysis*. Vol. 3. Pure and Applied Mathematics (New York). New York: John Wiley & Sons Inc., 1986, pp. xvi+637. ISBN: 0-471-08703-3

Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, 1974, pp. x+470

Lecture 3

Euclidean Division and Newton Iteration

Summary

A first consequence of fast multiplication is that Euclidean division can also be performed efficiently. This has many applications, to linear recurrences, to multipoint-evaluation and interpolation, themselves at the basis of efficient algorithms by the evaluation-interpolation paradigm. At the root of fast division is Newton's iteration in a symbolic context, whose quadratic convergence leads to fast algorithms for many other operations.

3.1 Euclidean Division

3.1.1 Definition and Naive Algorithm

The only two cases where Euclidean division are used in this course are integers and polynomials. For two integers A, B in $\mathbb{Z}$, Euclidean division produces two integers Q, R in $\mathbb{Z}$ such that $A = BQ + R$ and $|R| < |B|$. Similarly, for two polynomials A, B in $\mathbb{K}[X]$ where $\mathbb{K}$ is a field, Euclidean division of A by B consists in finding two polynomials Q, R in $\mathbb{K}[X]$ such that

$$A = BQ + R, \quad \deg R < \deg B. \tag{3.1}$$

In both cases Q is called the *quotient* and R the *remainder*. We use the notation

$$Q = A \operatorname{quo} B \qquad R = A \operatorname{rem} B$$

and $C \equiv D \bmod B$ when $(C - D) \operatorname{rem} B = 0$.

For polynomials, the naive algorithm consists in recovering the coefficients one by one, starting from those of highest degree. It is given in the following Maple code.

```
NaiveEuclDiv:=proc(A,B,x)
local da,db,Q,R,coB,c;
  Q:=0; R:=A;
  db:=degree(B,x); coB:=coeff(B,x,db);
  do
    dr:=degree(R,x);
    if dr<db then return Q,R fi;
    c:=coeff(R,x,dr)/coB*x^(dr-db);
    Q:=Q+c; R:=expand(R-c*B)
  od
end:
```

It is an exercise to verify that an invariant of this loop is that $A = BQ + R$ at each entry. A variant of the loop is that $\deg R$ decreases. Correctness and termination follow.

Lemma 3.1. *If A, B belong to $\mathbb{K}[x]$ with $\deg A \geq \deg B$, quo and rem use a number of arithmetic operations in $\mathbb{K}$ bounded by $(\deg A - \deg B + 1)(2 \deg B + 1)$.*

Proof. By definition of the degree, the leading coefficient `coB` of B is not 0 so that the division in the computation of c in the loop is always possible. That division costs 1 arithmetic operation. The multiplication by $x^{\deg R - \deg B}$ does not use any arithmetic operation and neither does the addition of c to Q. The computation of $R - cB$ costs at most $2(\deg B + 1)$ arithmetic operations: one for the multiplication and one for the subtraction. However, it is known in advance that the coefficient of degree $\deg R$ in the result is 0 so that only $2 \deg B$ operations are necessary. Adding the cost of the computation of c shows that each iteration of the loop costs at most $2 \deg B + 1$ arithmetic operations.

The loop is repeated until the remainder has degree smaller than $\deg B$, which takes at most $\deg A - \deg B + 1$ iterations. Thus the total number of operations is bounded as announced in the lemma. □

3.1.2 Reduction to Inversion and Multiplication

Dividing Eq. (3.1) by B gives a sum of rational functions

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

If these were rational functions with coefficients in $\mathbb{R}$, then since $\deg R < \deg B$, an asymptotic expansion as $X \to \infty$ would begin with the coefficients of Q. This observation is the starting point for the computation of Euclidean division via fast multiplication.

Changing X into $1/X$ in Eq. (3.1) and multiplying by $X^{\deg A}$ gives an identity between polynomials:

$$\underbrace{X^{\deg A} A(1/X)}_{\mathrm{rev}(A)} = \underbrace{X^{\deg B} B(1/X)}_{\mathrm{rev}(B)} \underbrace{X^{\deg Q} Q(1/X)}_{\mathrm{rev}(Q)} + \underbrace{X^{\deg R} R(1/X)}_{\mathrm{rev}(R)} X^{\deg A - \deg R}, \tag{3.2}$$

where the reverse $\mathrm{rev}(P)$ of a polynomial P is obtained in linear time (and with 0 arithmetic operations) by changing the order of its coefficients. Note that the exponent of X in the last factor is $\deg A - \deg R > \deg A - \deg B = \deg Q$ so that the $\deg Q + 1 = \deg A - \deg B + 1$ coefficients of Q can be determined by working modulo $X^{\deg A - \deg B + 1}$. The only operations that is needed is to invert the polynomial $\mathrm{rev}(B)$ modulo this exponent of X.

3.1.3 Inverse $\bmod X^N$

Given an polynomial $P \in \mathbb{K}[X]$ such that $P(0) \neq 0$ and given $N > 0$ in $\mathbb{N}$, the computation of the inverse of P modulo X^N can be achieved by a divide-and-conquer algorithm thanks to the following.

Lemma 3.2. *If $P \in \mathbb{K}[X]$ is such that $P(0) \neq 0$ and $I \in \mathbb{K}[X]$ is such $IP = 1 + X^N Q$ for some polynomial $Q \in \mathbb{K}[X]$, then there exists $R \in \mathbb{K}[X]$ such that $JP = 1 + X^{2N} R$, with*

$$J = 2I - I^2 P \bmod X^{2N}.$$

Proof. By definition of J,

$$\begin{aligned} JP &= 2IP - I^2 P^2 \bmod X^{2N} \\ &= 2 + 2X^N Q - (1 + 2X^N Q + X^{2N} Q^2) \bmod X^{2N} \\ &= 1 \bmod X^{2N}. \end{aligned}$$

□

This translate into the following Maple code.

```
InvMod:=proc(A,x,N)
local n,Ia;
  if N=1 then return 1/eval(A,x=0) fi;
  n:=ceil(N/2);
  Ia:=thisproc(A,x,n);
  return rem(2*Ia-Ia^2*A,x^N,N)
end:
```

If $C(n)$ is the number of coefficient operations performed by this algorithm, we see that

$$C(n) \leq C(\lceil n/2 \rceil) + 3\mathsf{M}(\lceil n/2 \rceil) + \lambda n.$$

The term $C(\lceil n/2 \rceil)$ comes from the recursive call (`thisproc` in the Maple code); since f is computed modulo $X^{\lceil n/2 \rceil}$, its multiplication by a can be performed in two multiplications of polynomials of that degree, in $\mathsf{M}(\lceil n/2 \rceil)$ each; the result af is $1 + O(X^{\lceil n/2 \rceil})$, so that t itself has valuation at least $\lceil n/2 \rceil$ and the last multiplication of f by t only costs an extra $\mathsf{M}(\lceil n/2 \rceil)$; the addition of f and ft have linear complexity; the division by powers of X does not cost any arithmetic operation.

By the 'master theorem of divide-and-conquer', we deduce the following.

Proposition 3.1 (Sieveking 1972; Kung 1974)**.** *Given $P \in \mathbb{K}[X]$ with $P(0) \neq 0$ and a nonnegative integer N, one can compute $I_P \in \mathbb{K}[X]$ of degree $< N$ such that $I_P \times P = 1 \bmod X^N$ in $O(\mathsf{M}(n))$ arithmetic operations in $\mathbb{A}$.*

3.1.4 Fast Euclidean Division

Using this modular inverse, the algorithm for Euclidean division is straightforward:

```
rev:=proc(P,X)
local i,deg;
  deg:=degree(P,X);
  add(coeff(P,X,deg-i)*X^i,i=0..deg)
end:

EuclDiv:=proc(A,B,X)
local Ib,Q,dq;
  dq:=degree(A,X)-degree(B,X);
  Ib:=InvMod(rev(B,x),X,dq+1);
  Q:=rem(Ib*rev(A),X^(dq+1),X);
  Q:=rev(Q,X);
  return Q,expand(A-B*Q);
end:
```

For the proof of correctness of this algorithm, write $\bar{A}, \bar{B}, \bar{Q}$ for the reverse of the polynomials A, B, Q and I_B for the result of Algorithm `InvMod` on $\bar{B}$ at precision $\deg A - \deg B + 1$. Equation (3.2) then rewrites

$$\bar{A} = \bar{B}\bar{Q} + O(X^{\deg A - \deg B + 1}). \tag{3.3}$$

The definition of Algorithm `InvMod` gives

$$I_B \bar{B} = 1 + O(X^{\deg A - \deg B + 1}).$$

Thus, multiplying Eq. (3.3) by I_B yields

$$I_B \bar{A} = \bar{Q} + O(X^{\deg A - \deg B + 1})$$

and since $\deg Q < \deg A - \deg B + 1$, it follows that

$$\bar{Q} = I_B \bar{A} \operatorname{rem} X^{\deg A - \deg B + 1},$$

which is what is computed by the algorithm.

In terms of complexity, the computation of I_B costs $O(\deg Q)$ operations for $\operatorname{rev}(B) \bmod X^{\deg Q+1}$, followed by $O(\mathsf{M}(\deg Q))$ operations for the modular inverse. Next, the computation of Q uses one multiplication and one reversion, again in complexity $O(\mathsf{M}(\deg Q))$ operations. Finally, the computation of R uses $O(\mathsf{M}(\deg A))$ operations. In summary, we have obtained.

Theorem 3.1. *Euclidean division of two polynomials A, B in $\mathbb{K}[X]$ with $\deg A = O(n)$ and $\deg B = n$ can be computed in $O(\mathsf{M}(n))$ operations in $\mathbb{K}$.*

An important consequence is that for any $P \in \mathbb{K}[X]$, multiplication of two polynomials modulo P can be performed in $O(\mathsf{M}(\deg P))$ operations in $\mathbb{K}$: both polynomials can be taken mod$P$ and thus have degree $< \deg P$, they are multiplied in $(\deg P)$ operations, which gives a result of degree $< 2 \deg P$ and then taking the remainder of the Euclidean division of that polynomial by P gives the result in another $O(\mathsf{M}(\deg P))$ operations.

3.2 Applications of Euclidean Division to Linear Recurrences

Fast Euclidean division leads to fast algorithms operating over solutions of linear recurrences with constant coefficients.

Definition 3.1. *A sequence $(u_n)_{n\in\mathbb{N}}$ of elements of a field $\mathbb{K}$ is called a* linear recurrent sequence *if there exist $(a_0, \ldots, a_{d-1})$ in $\mathbb{K}$ such that it satisfies*

$$u_{n+d} = a_{d-1}u_{n+d-1} + \cdots + a_0 u_n, \quad n \geq 0. \tag{3.4}$$

The polynomial $P = X^d - a_{d-1}X^{d-1} - \cdots - a_0$ is called the characteristic polynomial *of the sequence.*

Example 3.1. These sequences are very common. Besides the classical Fibonacci numbers that satisfy $F_{n+2} = F_{n+1} + F_n$, one can mention

- the number of solutions $(x_1, \ldots, x_r) \in \mathbb{N}^r$ of the equation
$$a_1 x_1 + \cdots + a_r x_r = n;$$
- the number of words of length n in any regular language;
- the entry $(1, 1)$ (for instance) in the matrix A^n, where A is a square matrix;
- the sequence $P(an + b)$ where P is a polynomial, for arbitrary a and b.

Given initial conditions $(u_0, \ldots, u_{d-1})$ all the elements of the sequence are fully determined by Eq. (3.4). Given N, one can thus compute $u_0, \ldots, u_N$ in $O(Nd)$ operations by unrolling the recurrence. We are going to show the following.

Theorem 3.2. *Given $(u_0, \dots, u_{d-1})$, a positive integer N and the coefficients a_i of the linear recurrence (3.4), one can*

1. *compute $u_0, \dots, u_N$ in $O(N\mathsf{M}(d)/d)$ arithmetic operations in $\mathbb{K}$;*
2. *compute u_N in $O(\mathsf{M}(d)\log N)$ arithmetic operations in $\mathbb{K}$.*

Note that if one is using FFT, the first one has complexity linear in N and logarithmic in d; the second one is quasi-linear in d and logarithmic in N.

3.2.1 Companion Matrix

A starting point in the design of efficient algorithms for linear recurrent sequence is to consider the companion matrix of the characteristic polynomial.

Definition 3.2. *If $P = X^d - a_{d-1}X^{d-1} - \cdots - a_0$, the* companion matrix *of P is the matrix*

$$C_P = \begin{pmatrix} 0 & 0 & \dots & 0 & a_0 \\ 1 & 0 & \dots & 0 & a_1 \\ 0 & 1 & \dots & 0 & a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & a_{d-1} \end{pmatrix}.$$

(Note that sometimes in the literature the transpose of this matrix is called the companion matrix.)

Two interpretations of this matrix are useful. First, a simple rewriting of the recurrence (3.4) shows that for any $n \geq 0$,

$$(u_{n+1}, \dots, u_{n+d}) = (u_n, \dots, u_{n+d-1}) \cdot C_P. \tag{3.5}$$

The next interpretation is given by the following.

Lemma 3.3. *The companion matrix C_P of the polynomial P is the matrix of the linear map $Q \mapsto XQ \bmod P$ expressed in the basis $(1, X, \dots, X^{d-1})$.*

Proof. For the first $d-1$ columns, this is a consequence of $X^i X = X^{i+1} \bmod P$. For the last column, it is the fact that

$$X^d = a_0 + \cdots + a_{d-1}X^{d-1} \bmod P.$$ □

Corollary 3.1. *For all nonnegative integers i, k,*

$$u_{i+k} = (u_i, u_{i+1}, \dots, u_{i+d-1}) \cdot V_k,$$

where V_k is the vector of coefficients of $X^k \bmod P$.

Proof. Using Eq. (3.5) k times shows that

$$(u_{i+k}, \dots, u_{i+k+d-1}) = (u_i, \dots, u_{i+d-1}) \cdot C_P^k$$

and therefore

$$u_{i+k} = (u_i, u_{i+1}, \dots, u_{i+d-1}) \cdot V_k,$$

with V_k the first column of C_P^k. The lemma shows that C_P^k is the matrix of multiplication by $X^k \bmod P$ expressed in the basis $(1, X, \dots, X^{d-1})$. Its first column thus has for entries the coefficients of $X^k \bmod P$ in this basis. □

3.2.2 Modular Binary Powering

The corollary above leads to a simple algorithm for the computation of u_N, using

$$X^N \bmod P = \begin{cases} (X^{N/2})^2 \bmod P, & \text{if } N \text{ is even,} \\ (X^{(N-1)/2})^2 X \bmod P, & \text{otherwise.} \end{cases}$$

The complexity of this computation is governed by the inequality

$$C(N) \leq C(\lfloor N/2 \rfloor) + O(\mathsf{M}(d)),$$

whence $C(N) = O(\mathsf{M}(d) \log N)$ arithmetic operations in $\mathbb{K}$. The resulting algorithm, due to Fiduccia (1985), translated in Maple, looks like

```
linrec:=proc(P,x,ini,N)
local i,q;
  q:=binpow(P,x,N);
  add(ini[i+1]*coeff(q,x,i),i=0..degree(P,x)-1)
end:

binpow:=proc(p,x,n)
local q,r;
  if n=1 then x
  else
    q:=binpow(p,x,iquo(n,2,'r'));
    if r=0 then rem(q^2,p,x)
    else rem(q^2*x,p,x)
    fi
  fi
end:
```

3.2.3 Slices of Coefficients

One can compute simultaneously d consecutive coefficients of the linear recurrent sequence at the cost of one multiplication of a fixed polynomial of degree $2d-2$ by a polynomial of degree smaller than d (i.e., in $2\mathsf{M}(d)$ operations) by a simple observation. If $v_0, \ldots, v_{d-1}$ are the coefficients of $X^k \bmod P$, then in the expansion of

$$(u_{2d-2} + \cdots + u_0 X^{2d-2})(v_0 + v_1 X + \cdots + v_{d-1} X^{d-1}),$$

the coefficient of X^{2d-i} for $0 \leq i \leq d-1$ is

$$v_{d-1} u_{i+d-1} + v_{d-2} u_{i+d-2} + \cdots + v_0 u_i = u_{k+i},$$

where the last equality comes from Corollary 3.1. Thus the d elements $(u_k, \ldots, u_{k+d-1})$ can be extracted from this product in $O(\mathsf{M}(d))$ operations. The resulting algorithm to compute the first N terms of the sequence is as follows:

```
linrecfirstterms:=proc(P,x,ini,N)
local d:=degree(P,x),V,u,i,j,k,S,A;
  for i to d do u[i-1]:=ini[i] od;
  for i from 0 to d-2 do
    u[d+i]:=-add(coeff(P,x,j)*u[i+j],j=0..d-1)
  od;
  A:=add(u[i]*x^(2*d-2-i),i=0..2*d-2);
  V:=x^(d-1);
  for k to ceil(N/d) do
    V:=rem(V*x^d,P,x);
    S:=expand(A*V);
    for i to d do u[k*d+d-2+i]:=coeff(S,x,2*d-1-i) od
  od;
  [seq(u[i],i=0..N)]
end:
```

Each iteration of the loop performs $O(\mathsf{M}(d))$ arithmetic operations in $\mathbb{K}$, giving the result in a total of $O(N\mathsf{M}(d)/d)$ operations.

3.2.4 Rational Functions

A fundamental result is that linear recurrent sequence and rational functions are equivalent.

Proposition 3.2. *The sequence (u_n) is a linear recurrent sequence with characteristic polynomial P of degree d if and only if the generating function $U(X) = \sum_{n\geq 0} u_n X^n$ is*

$$U(X) = \frac{N_0(X)}{\overline{P}(X)},$$

where $\overline{P}(X) = X^d P(1/X)$ and $\deg N_0 < d$.

Proof. First, writing

$$\overline{P}(X)U(X) = N_0(X)$$

and extracting the coefficient of X^i for $i \geq d$ shows that the coefficients u_n of U satisfy the linear recurrence (3.4).

Conversely, multiplying the recurrence by X^{n+d} and summing for $n \geq 0$ gives a linear equation for $U(X) = \sum_{n\geq 0} u_n X^n$, leading to the rational function representation. □

Example 3.2. A direct computation gives the generating function of the Fibonacci numbers:

$$\frac{1}{1-x-x^2} = 1 + x + 2x^2 + 3x^3 + 5x^4 + \cdots + F_n x^n + \cdots .$$

It follows that given a rational function P/Q with $\deg Q = d$ and $\deg P < \deg Q$, one can compute the truncated expansion $P/Q + O(X^N)$ in $O(N\mathsf{M}(d)/d)$ operations in $\mathbb{K}$ only by the algorithm above.

Notation 3.1. *If $U(X) = u_0 + u_1 X + \ldots$, we write $[X^n]U(X)$ to denote the nth coefficient of the power series expansion of U.*

3.2.5 Another Algorithm for the Nth Term

In 2021, Bostan and Mori designed new divide-and-conquer algorithm computing the Nth term of a linear recurrent sequence, that does not require the use of Euclidean division. It uses $O(\mathsf{M}(d)\log N)$ arithmetic operations, which is the same as Fiduccia's algorithm from Section 3.2.2, but the new complexity hides a smaller constant factor in the $O()$ estimate. The starting point is Proposition 3.2, showing that it is sufficient to be able to extract an Nth coefficient of the series expansion of a rational fraction $P(t)/Q(t)$ with $\deg P < \deg Q =: d$ (and $Q(0) \neq 0$).

The first step is to multiply numerator and denominator by $Q(-t)$:

$$[t^N]\frac{P(t)}{Q(t)} = [t^N]\frac{P(t)Q(-t)}{Q(t)Q(-t)}.$$

The motivation for doing this is that the denominator becomes an even polynomial:

$$Q(t)Q(-t) = \tilde{Q}(t^2),$$

where $\deg \tilde{Q} = d$. Splitting the numerator into its even and odd parts does not use any arithmetic operation and gives

$$P(t)Q(-t) = P_0(t^2) + tP_1(t^2),$$

with again $\deg P_0 < d$, $\deg P_1 < d$. Thus the quotient itself splits as

$$\frac{P(t)Q(-t)}{Q(t)Q(-t)} = \frac{P_0(t^2)}{\tilde{Q}(t^2)} + t\frac{P_1(t^2)}{\tilde{Q}(t^2)},$$

where the first term on the right-hand side is even, while the other one is odd. It follows that

$$[t^N]\frac{P(t)}{Q(t)} = \begin{cases} [t^N]\frac{P_0(t^2)}{\tilde{Q}(t^2)}, & N \text{ even;} \\ [t^N]\frac{tP_1(t^2)}{\tilde{Q}(t)} = [t^{N-1}]\frac{P_1(t^2)}{\tilde{Q}(t)}, & N \text{ odd.} \end{cases}$$

In summary, this derivation gives the following basis for a divide-and-conquer algorithm:

$$[t^N]\frac{P(t)}{Q(t)} = [t^{\lfloor N/2 \rfloor}]\frac{P_{N \bmod 2}(t)}{\tilde{Q}(t)},$$

where in the right-hand side, the degree of the numerator and denominator obey the same bounds as in the left-hand side, but now N is halved.

The corresponding Maple code is as follows:

```
# Input:
#         P,Q   polynomials in t
#         t     variable
#         N     nonnegative integer
# Output:
#         [t^N](P/Q) the coefficient of t^N in the
#         Taylor expansion of P/Q at t=0
RatFunDAC:=proc(P,Q,t,N)
local k,Qm,Qt,PQm,Pt;
        if N=0 then return eval(P/Q,t=0) fi;
        k:=iquo(N,2,'isodd');
        Qm:=subs(t=-t,Q);
        Qt:=evenpart(Qm*Q,t,k);
        PQm:=Qm*P;
        if isodd=0 then Pt:=evenpart((PQm+subs(t=-t,PQm))/2,t,k)
        else Pt:=evenpart((PQm-subs(t=-t,PQm))/2/t,t,k)
        fi;
        thisproc(Pt,Qt,t,k)
end:

evenpart:=proc(S,t,k) # even part of S mod t^(k+1)
local P,i;
        P:=expand(S);
        add(coeff(P,t,2*i)*t^i,i=0..k)
end:
```

The complexity analysis is straightforward: the number of arithmetic operations with input N satisfies

$$C(N) \leq C(\lfloor N/2 \rfloor) + 2\mathsf{M}(d) = O(\mathsf{M}(d)\log N).$$

A variant of this method also produces $t^N \bmod Q(t)$ in $O(\mathsf{M}(d)\log N)$ arithmetic operations, which can be used as in Section 3.2.3 to compute a slice of coefficients of high index.

3.3 Evaluation and Interpolation

An important algorithmic paradigm in computer algebra is *evaluation-interpolation*: it consists of constructing a polynomial by computing sufficiently many evaluations of it and reconstructing it by interpolation. This is the process that was used in polynomial multiplication by FFT, using roots of unity as the points where evaluation took place. With n such points, FFT led to a complexity in $O(n\log n)$ operations. The main result of this section is that for n arbitrary distinct points, this process can be performed in $O(\mathsf{M}(n)\log n)$ operations.

Given n distinct points $(a_1,\ldots,a_n)$ in a field $\mathbb{K}$ and a polynomial $P \in \mathbb{K}[X]$ of degree smaller than n, *multipoint evaluation* is the computation of $(P(a_1),\ldots,P(a_n))$. The converse operation is *interpolation*.

Denoting by p_i the coefficients of P, Horner's rule evaluates $P(t)$ as

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

Using this method, each evaluation of $P(a_i)$ has complexity $O(n)$, so that the naïve multipoint-evaluation algorithm has quadratic complexity.

Similarly, Lagrange's interpolation gives P from its values $(b_1,\ldots,b_n)$ as

$$P = \sum_i b_i \frac{A_i(X)}{A_i(a_i)}, \quad A_i(X) = \prod_{j\neq i}(X - a_j). \tag{3.6}$$

Thus an algorithm for this computation is

1. Compute $A(X) = \prod_i (X - a_i)$;
2. Deduce $A_1, \ldots, A_n$;
3. Evaluate each A_i at a_i;
4. Compute the final linear combination.

By naïve algorithms, each of these steps has quadratic complexity.

3.3.1 Product Tree

The starting point of a faster algorithm due to Borodin and Moenck (1974) is to construct what is called the *product tree*, displayed in Fig. 3.1.

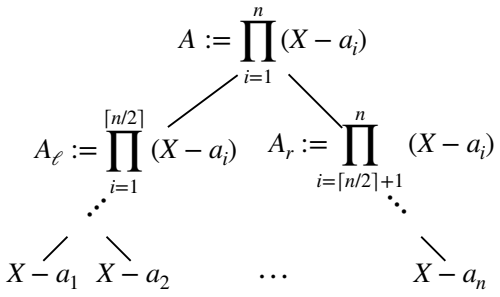


Figure 3.1: Product Tree

It is a binary tree whose leaves store the $X - a_i$ and where each internal node stores the product of the polynomials stored at the roots of its children. The computation of this tree is performed recursively:

```
ProductTree:=proc(L,x)
local k,left,right;
  if nops(L)=1 then [x-op(L)]
  else
    k:=ceil(nops(L)/2);
    left:=thisproc(L[1..k],x);
    right:=thisproc(L[k+1..-1],x);
    [expand(left[1]*right[1]),left,right]
  fi
end:
```

By design, the complexity of this construction obeys

$$C(n) \leq 2C(\lceil n/2 \rceil) + \mathsf{M}(n),$$

whence a complexity $C(n) = O(\mathsf{M}(n) \log n)$.

3.3.2 Fast Multipoint Evaluation

A basic observation is that if

$$P = Q(X) \prod_{i=1}^{n} (X - a_i) + R(X),$$

then the values of P at $a_1, \ldots, a_n$ are the same as those of R at these points. The multipoint evaluation thus relies on the product tree and fast Euclidean division. Let $\mathcal{T}_A$ denote a product tree with A at its root and let $\mathcal{T}_{A_\ell}$ and $\mathcal{T}_{A_r}$ be its left and right subtrees if it is not a leaf. Then the idea is to evaluate $P \bmod A_\ell$ on $\mathcal{T}_{A_\ell}$ and $P \bmod A_r$ on $\mathcal{T}_{A_r}$. The resulting algorithm is

```
EvalPol:=proc(p,T,x)
  if degree(T[1],x)=1 then p
  else thisproc(rem(p,T[2][1],x),T[2],x),
    thisproc(rem(p,T[3][1],x),T[3],x)
  fi
end:
```

Thanks to fast Euclidean division, the complexity is easily seen to satisfy

$$C(n) \leq 2C(\lceil n/2 \rceil) + O(\mathsf{M}(n)),$$

whence $C(n) = O(\mathsf{M}(n) \log n)$.

3.3.3 Interpolation

Dividing the Lagrange interpolation formula (3.6) by $A(X)$ gives a partial fraction decomposition

$$\frac{P(X)}{A(X)} = \sum_{i=1}^{n} \frac{b_i}{A'(a_i)} \frac{1}{X - a_i},$$

where again

$$A(X) = \prod_{i=1}^{n} (X - a_i).$$

The algorithm for fast interpolation is as follows:

1. Compute A by a product tree;
2. Deduce A';
3. Compute $(A'(a_1), \ldots, A'(a_n))$ by multipoint-evaluation;
4. Compute $c_i = b_i / A'(a_i)$ for $i = 1, \ldots, n$;
5. Compute the sum of $c_i/(X - a_i)$ by a divide-and-conquer method;
6. Return its numerator.

Steps 2 and 4 have linear complexity $O(n)$. Step 5 has a complexity $C(n)$ which satisfies $C(n) \leq 2C(\lceil n/2 \rceil) + O(\mathsf{M}(n))$ and thus is in $O(\mathsf{M}(n) \log n)$. This is also the complexity of Steps 1 and 3 by the previous sections. It follows that the total cost is $O(\mathsf{M}(n) \log n)$ arithmetic operations in $\mathbb{K}$.

The Maple version makes more explicit the addition of rational functions.

```
Interpolate:=proc(a,b,x)
local T,Apr,i,n,c,pf;
  n:=nops(a);
  T:=ProductTree(a,x);
  Apr:=[EvalPol(diff(T[1],x),T,x)];
  c:=[seq(b[i]/Apr[i],i=1..n)];
  pf:=AddFrac([seq(c[i]/(x-a[i]),i=1..n)]);
  normal(pf*T[1])
end:

AddFrac:=proc(L)
local k;
  if nops(L)=1 then L[1]
  else
    k:=iquo(nops(L),2);
    normal(thisproc(L[1..k])+thisproc(L[k+1..-1]))
  fi
end:
```

3.3.4 Product of Polynomial Matrices

This is a natural application of the evaluation-interpolation paradigm. Given A, B two matrices in $\mathbb{K}[X]^{d\times d}$ with $\deg A$ and $\deg B$ smaller than n, the aim is to compute the matrix product AB.

Later in the course, we will see the definition of *feasible matrix exponent* as a number ω such that two $d \times d$ matrices with entries in a ring $\mathbb{A}$ can be computed in $O(d^\omega)$ operations in $\mathbb{A}$. In principle ω depends on $\mathbb{A}$, but with the currently known algorithms, it does not.

A direct use of a matrix multiplication algorithm over $\mathbb{K}[X]$ results in a complexity of $O(d^\omega \mathsf{M}(n))$ operations. By contrast, evaluating both matrices at $2n$ points, multiplying the resulting scalar matrices and interpolating the entries of the result leads to a better complexity of

$$O(d^\omega n + d^2 \mathsf{M}(n) \log n).$$

3.4 Newton's Iteration for Division

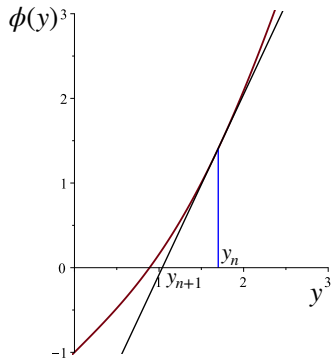


Figure 3.2: Principle of Newton's Iteration

The principle of Newton's iteration is illustrated in Fig. 3.2. In order to solve the equation $\phi(y) = 0$, a good starting point y_0 is chosen and at each step one applies the following recipe:

1. take the tangent to the graph of ϕ at $(y_n, \phi(y_n))$, whose equation is
$$z = \phi(y_n) + \phi'(y_n)(y - y_n);$$
2. take for y_{n+1} the intersection of that tangent with the x-axis.

This leads to the following formula for y_{n+1}, which is central to this lecture
$$y_{n+1} = \mathcal{N}(y_n) := y_n - \frac{\phi(y_n)}{\phi'(y_n)}.$$

Example 3.3. This method was used by Newton himself in his *Treatise of Fluxions* published in 1671 to solve Kepler's equation
$$\phi(y) = 2y - \sin y - 1.$$
With this value of ϕ, the iteration becomes
$$y_{n+1} = y_n - \frac{2y_n - \sin(y_n) - 1}{2 - \cos(y_n)}.$$
Starting with $y_0 = 1.7$, the first few iterates are
$$\begin{aligned} y_0 &= 1.7 \\ y_1 &= 1.0384508857639334498 \\ y_2 &= \mathbf{0.8}942024477681162624 \\ y_3 &= \mathbf{0.88787}359918134588142 \\ y_4 &= \mathbf{0.887862211}60760794737 \\ y_5 &= \mathbf{0.8878622115708660 2403} \end{aligned}$$
where the digits in bold face are those of the solution. The number of correct digits roughly doubles at each iteration. This is typical of *quadratic convergence*, where the error is squared at each step (asymptotically).

3.4.1 Reciprocal and Division

The application of Newton's method to
$$\phi(y) = 1/y - a$$
gives an iteration that does not use division to compute the inverse of a:
$$y_{n+1} = y_n + y_n(1 - ay_n). \tag{3.7}$$
This iteration is illustrated in Fig. 3.3. On the left is the graph of ϕ (here $a = 3$); on the right is the graph of $\mathcal{N}(y) = y + y(1 - ay)$, together with the first iterates.

The iteration (3.7) can easily be turned into a naïve Maple code:

```
Inverse:=proc(a,y0)
local yprev:=0,ynew:=y0;
  while ynew<>yprev do
    yprev,ynew:=ynew,ynew+ynew*(1-a*ynew)
  od;
  ynew
end:
```

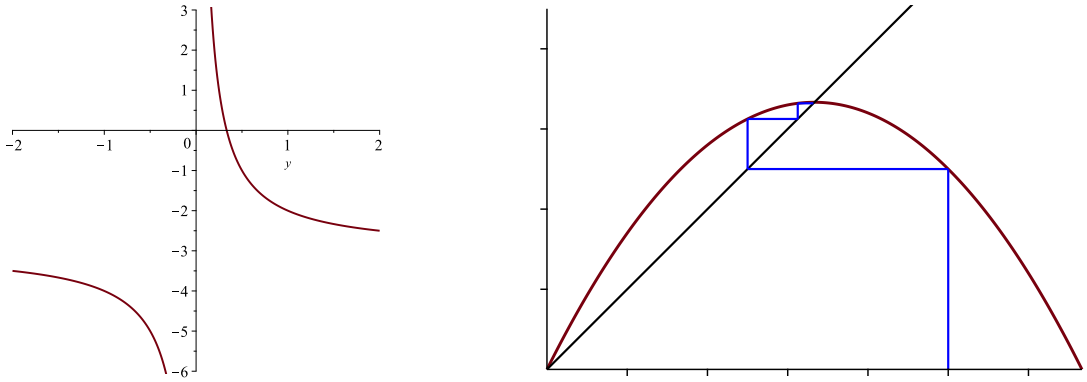


Figure 3.3: Newton iteration for reciprocal

Note however that although this code works, termination of such numerical code is in general difficult to predict due to rounding errors in the floating-point computations.

The iteration (3.7) has quadratic convergence:

$$1 - ay_{n+1} = 1 - ay_n - ay_n(1 - ay_n) = (1 - ay_n)(1 - ay_n) = (1 - ay_n)^2$$

implies

$$\frac{1}{a} - y_{n+1} = a\left(\frac{1}{a} - y_n\right)^2.$$

The distance between y_n and the limit $1/a$ is squared, up to a constant factor, at each step.

An application of this iteration is the computation of Euclidean division for integers: given two positive integers (a, b), the aim is to compute two integers (q, r) such that $a = bq + r$ and $0 \leq r < b$. A simple method is to compute $s := 1/b$ numerically to sufficient precision by Newton's iteration and then compute $q = \lfloor s \times a \rfloor$ and finally deduce $r = a - bq$. A symbolic variant of this idea will give us a fast algorithm for Euclidean division of polynomials in Section 3.1.

3.4.2 Truncated Power Series

We define a *truncated power series* as an expansion of the form

$$A(X) = a_0 + a_1 X + \cdots + a_{n-1}X^{n-1} + O(X^n).$$

This means that the computation is performed modulo the polynomial X^n.

Addition of truncated power series has linear complexity in n; multiplication has complexity $O(\mathsf{M}(n))$, since it is sufficient to multiply two polynomials of degree smaller than n and discard the coefficients of the monomials X^k for $k \geq n$.

The operations that are performed by Newton iteration on real numbers turn out to often work for truncated power series as well and this is the basis of many efficient algorithms for polynomials.

3.4.3 Reciprocal and Division of Truncated Power Series

It is now easy to recognize that the computation of the inverse of a polynomial modX^N in Section 3.1.3, that was using the iteration

$$I \mapsto 2I - I^2 P \bmod X^{2N}$$

is a special case of Newton's iteration for the inverse

$$\mathcal{N}(y) = y + y(1 - ay) \tag{3.8}$$

that is used to solve $1 - ay = 0$, i.e., to compute $1/a$.

Of course, given an algorithm for inversion, we obtain an algorithm for division by multiplying the numerator by the inverse of the denominator and the conclusion is that division can be performed in $O(\mathsf{M}(n))$ operations. It is therefore of the same complexity as multiplication, up to a constant factor.

3.5 Fast Composition of Formal Power Series

In the same way that algorithms for polynomials and integers are very similar, those for polynomials being easier to state and prove, it is fruitful to consider formal power series as an analogue to real numbers. In practice the truncated power series introduced earlier behave very similarly to floating-point numbers. These objects are now introduced formally.

3.5.1 Definition

Definition 3.3. *The set of* formal power series *with coefficients in a ring $\mathbb{A}$ is denoted $\mathbb{A}[[X]]$. Its elements are the sequences of elements of $\mathbb{A}$, i.e., the set $\mathbb{A}^{\mathbb{N}}$. They are written in the form*

$$A(X) = a_0 + a_1 X + a_2 X^2 + \cdots \quad \text{or} \quad A(X) = \sum_{i=0}^{\infty} a_i X^i.$$

This set is given two operations:

Addition:

$$\sum_{i=0}^{\infty} a_i X^i + \sum_{j=0}^{\infty} b_j X^j = \sum_{i=0}^{\infty} (a_i + b_i) X^i.$$

Multiplication:

$$\sum_{i=0}^{\infty} a_i X^i \times \sum_{j=0}^{\infty} b_j X^j = \sum_{k=0}^{\infty} c_k X^k, \quad c_k = \sum_{i+j=k} a_i b_j.$$

The ring $\mathbb{A}$ here is not assumed to be commutative. This will allow us to consider matrices of formal power series as formal power series with matrix coefficients.

The sum symbols ($\sum$) are not meant to suggest any kind of convergence in $\mathbb{A}$. These formal power series are formal objects, a convenient way to manipulate all the elements of a sequence. In the operations like the multiplication above, the coefficient of X^k in the result involves only finitely many of the coefficients of A and B.

By convention, we use small indexed letters (a_i, b_i,...) for the coefficients of the power series whose name is the corresponding capital letter (A, B,...).

Proposition 3.3. *The set $\mathbb{A}[[X]]$ is a ring.*

Proof. The required properties (commutative group for $+$, associativity and distributivity of $\times$ and unit element) are checked as in the case of polynomials. □

3.5.2 Limits

We will define iterations, like Newton's iteration, operating on power series. This implies a notion of convergence, not at the level of coefficients in $\mathbb{A}$, but at the level of power series. Roughly speaking, a sequence of power series will converge when larger and larger numbers of initial coefficients are fixed. One way of defining limits without too much algebraic setup is the following.

Definition 3.4. *The sequence (S_n) of formal power series converges to the formal power series S if there is an increasing sequence of nonnegative integers (r_n) such that $S = S_n + O(X^{r_n})$.*

One can deduce the existence of a limit S from a weak condition on the sequence (S_n).

Lemma 3.4. *If $S_{n+1} - S_n = O(X^{r_n})$ for an increasing sequence of integers (r_n), then the sequence (S_n) converges to a formal power series S.*

Proof. For $i \in \{r_{n-1}, \dots, r_n - 1\}$, define s_i as the coefficient of X^i in S_n. By induction, the condition implies that this is also the coefficient of X^i in all S_k with $k \geq n$. Now define S as the power series $S = s_0 + s_1 X + \cdots$. By definition of (u_i), the formal power series S satisfies $S - S_n = O(X^{r_n})$, which proves the convergence of the sequence (S_n) to S. □

3.5.3 Composition

If F, G are in $\mathbb{A}[[X]]$ with $g_0 = 0$, the composition $F(G(X))$ is defined by

$$F(G(X)) = f_0 + f_1 G + f_2 G^2 + \cdots .$$

This definition has to be interpreted as the limit of the truncated sum. It is meaningful because the summands after $f_k G^k$ are all $O(X^{k+1})$ since $g_0 = 0$.

The composition law gives a simple way to characterize the invertible elements of the ring $\mathbb{A}[[X]]$ for multiplication.

Proposition 3.4. *$A \in \mathbb{A}[[X]]$ is invertible for multiplication if and only if $a_0 \in \mathbb{A}$ is invertible.*

Proof. If A is invertible with inverse B, extracting the coefficient of X^0 in the product $AB = 1$ gives $a_0 b_0 = 1$, showing that a_0 is invertible.

Conversely, assume first that $a_0 = 1$. A direct computation shows that

$$(1 + X)(1 - X + X^2 - X^3 + \dots) = 1.$$

Composing this identity with $A(X) - 1$ shows that A is invertible. Finally if a_0 is invertible, then $B = a_0^{-1} A$ is a power series with $b_0 = 1$, making it invertible. It follows that $A(B^{-1} a_0^{-1}) = a_0 B B^{-1} a_0^{-1} = 1$, showing that A is invertible. □

3.5.4 Fast Composition

Given

$$F = f_0 + f_1 X + \cdots + O(X^{n+1}), \quad G = g_1 X + \cdots + O(X^{n+1}),$$

the computation of the composition

$$F(G(X)) = f_0 + f_1 G + f_2 G^2 + \cdots + O(X^{n+1})$$

by the naive method uses $O(n\mathsf{M}(n))$ operations. A faster method was designed in 1978 by Brent and Kung who reduced the complexity to $O(\sqrt{n \log n}\mathsf{M}(n))$ operations. Then, in 2024, Kinoshita and Li finally found a quasi-optimal algorithm in $O(\mathsf{M}(n) \log n)$ that we now present.

The idea is to apply a divide-and-conquer similar to that of Section 3.2.5 to the expansion of a *bivariate* rational function that has the composition has its coefficient of t^n:

$$\begin{aligned} \frac{\mathrm{rev}(F)}{1 - tG} &= (f_n + f_{n-1} t + \cdots + f_0 t^n)(1 + tG + t^2 G^2 + \cdots) \\ &= f_n + (f_{n-1} + f_n G) t + \cdots + (f_0 + f_1 G + \cdots + f_n G^n) t^n + \cdots . \end{aligned}$$

Given a formal power series

$$A = a_0 + a_1 t + \cdots,$$

the following notation extracts the last d coefficients up to and including that of t^n:

$$\mathcal{F}_{n,d}(A) := a_{n-d+1} + \cdots + a_n t^{d-1}.$$

Basic properties can be checked directly:

$$a_{n-d+1}t^{n-d+1} + \cdots + a_n t^n = t^{n-d+1}\mathcal{F}_{n,d}(A)$$
$$\mathcal{F}_{n,d}(t^k A) = \mathcal{F}_{n-k,d}\, A$$

and for a polynomial Q,

$$\mathcal{F}_{n,d}(QA) = \mathcal{F}_{n,d}(Q\mathcal{F}_{n,d+\deg Q}(A)).$$

Using these relations leads to the divide-and-conquer algorithm:

$$\begin{aligned}
\mathcal{F}_{N,m}\left(\frac{P(t)}{Q(t,x)}\right) \bmod x^n &= \mathcal{F}_{N,m}\left(\frac{Q(t,-x)P(t)}{Q(t,-x)Q(t,x)}\right) \bmod x^n \\
&= \mathcal{F}_{N,m}\left(Q(t,-x)t^{N-m-\deg_t Q+1}\mathcal{F}_{N,m+\deg_t Q}\left(\frac{P(t)}{Q(t,-x)Q(t,x)}\right)\right) \bmod x^n \\
&= \mathcal{F}_{m+\deg_t Q-1,m}\left(Q(t,-x)\,\mathcal{F}_{N,m+\deg_t Q}\left(\frac{P(t)}{\tilde{Q}(t,z)} \bmod z^{\lceil n/2\rceil}\right)\Big|_{z=x^2}\right) \bmod x^n,
\end{aligned}$$

where $\tilde{Q}(t,z)$ is defined by $\tilde{Q}(t,x^2) = Q(t,x)Q(t,-x)$. In the last expression, the truncation order in z has been halved, the degree of P is unchanged and the degree of $\tilde{Q}$ is doubled. It turns out that this is sufficient to obtain a good complexity for the algorithm deduced from this last equality, which is given in Fig. 3.4.

For the complexity analysis, we first note that P and N remain unchanged during a recursive call. We write $C(n, m, \deg_t Q)$ for the complexity, keeping track only of the parameters that change. This complexity satisfies

$$C(n, m, \deg_t Q) \le C(\lceil n/2\rceil, m + \deg_t Q, 2\deg_t Q) + \mathsf{M}(2n\deg_t Q) + \mathsf{M}(2n(m + 2\deg_t Q)).$$

The last two terms are due to the multiplication $Q(t,x)Q(t,-x)$ and the final multiplication by $Q(t,-x)$.

The 'master theorem' of divide-and-conquer does not apply to such a multi-indexed recurrence, but following the same steps as in its proof yields the complexity. For this, we first recall the notation

$$\lceil n/2\rceil_1 = \lceil n/2\rceil, \quad \lceil n/2\rceil_{k+1} = \lceil \lceil n/2\rceil_k/2\rceil. \tag{3.9}$$

We also make use of P the power of 2 that satisfies $n \le P < 2n$.

The composition algorithm invokes the divide-and-conquer routine with input $\operatorname{rev} F(t)/(1 - tG(x))$, so that the complexity of interest is $C(n, 1, 1)$. Using Eq. (3.9) gives

$$\begin{aligned}
C(n,1,1) &\le C(\lceil n/2\rceil_1, 2, 2) + \mathsf{M}(2n) + \mathsf{M}(6n) \\
&\le C(\lceil n/2\rceil_1, 2, 2) + \mathsf{M}(2P) + \mathsf{M}(6P),
\end{aligned}$$

where we used the fact that M is increasing (a consequence of $\mathsf{M}(n+n') \ge \mathsf{M}(n) + \mathsf{M}(n')$). Using Eq. (3.9) again gives

$$\begin{aligned}
C(\lceil n/2\rceil_1, 2, 2) &\le C(\lceil n/2\rceil_2, 4, 4) + \mathsf{M}(4\lceil n/2\rceil_1) + \mathsf{M}(12\lceil n/2\rceil_1) \\
&\le C(\lceil n/2\rceil_2, 4, 4) + \mathsf{M}(2P) + \mathsf{M}(6P),
\end{aligned}$$

where the second line uses the fact that $x \mapsto \lceil x/2\rceil$ is increasing. Injecting the last inequality in that for $C(n,1,1)$ gives

$$C(n,1,1) \le C(\lceil n/2\rceil_2, 4, 4) + 2\mathsf{M}(2P) + 2\mathsf{M}(6P).$$

```
# Input:
#         P,Q          polynomials in K[t],K[t,x]
#         x,t          variables
#         m,N          nonnegative integers
# Output:
#         a_{N-m+1} + ... + a_N t^{m-1} mod x^n
# where P(t)/Q(t,x) = a_0 + a_1 t + ...
CompDAC:=proc(P,Q,x,t,m,n,N)
local k,dQ,Qm,Qt,PQm,Pt,rec,F,i;
        if n=1 then
                F:=series(eval(P/Q,x=0),t,N+1);
                add(coeff(F,t,N-m+1+i)*t^i,i=0..m-1)
        else
                k:=ceil(n/2);
                Qm:=subs(x=-x,Q);
                Qt:=evenpart(Qm*Q,x,k);
                dQ:=degree(Q,t);
                rec:=subs(x=x^2,thisproc(P,Qt,x,t,m+dQ,k,N));
                F:=rem(expand(Qm*rec),x^n,x);
                add(coeff(F,t,dQ+i)*t^i,i=0..m-1);
        fi
end:

# Compute f(g(x)) mod x^m
composition:=proc(f,g,x,m)
local i,t;
        CompDAC(add(coeff(f,x,m-1-i)*t^i,i=0..m-1),1-t*g,x,t,1,m,m-1)
end:
```

Figure 3.4: Divide-and-Conquer Composition of Truncated Formal Power Series

Iterating the same reasoning k times gives

$$C(n,1,1) \le C(\lceil n/2 \rceil_k, 2^k, 2^k) + k\mathsf{M}(2P) + k\mathsf{M}(6P).$$

When $k = \log_2 n$, one reaches the end of the recursion, where the complexity $C(1, 2^k, 2^k)$ is that of a division of truncated power series at precision $N = n$, thus is $O(\mathsf{M}(n))$, giving

$$C(n,1,1) \le O(\mathsf{M}(n)) + \log_2 n(\mathsf{M}(4n) + \mathsf{M}(12n)),$$

where we used again the monotonicity of M. In conclusion, since $\mathsf{M}(mn) \le m^2\mathsf{M}(n)$ for any m, we have

$$C(n,1,1) = O(\mathsf{M}(n)\log n),$$

as was to be proved.

3.5.5 Differentiation and Integration

The definitions of differentiation and integration are made in such a way that they coincide with the familiar ones for entire series.

Definition 3.5. *If $A = a_0 + a_1X + \cdots \in \mathbb{A}[[X]]$ with $\mathbb{A}$ a $\mathbb{Q}$-algebra, its derivative and its integral are defined by*

$$\left(\sum_{i\ge 0} a_i X^i\right)' = \sum_{i\ge 0} (i+1)a_{i+1}X^i, \quad \int \sum_{i\ge 0} a_i X^i = \sum_{i\ge 1} \frac{a_{i-1}}{i} X^i.$$

Recall that a $\mathbb{Q}$-algebra is both a ring and a vector space over $\mathbb{Q}$. Familiar examples are $\mathbb{Q}$ itself, $\mathbb{C}$, the ring of power series $\mathbb{Q}[[Y]]$ for another variable Y, or the ring $\mathcal{M}_n(\mathbb{Q})$ of square matrices with rational entries.

By extraction of the coefficient of X^i on both sides, the following identities are seen to hold:

$$\int A' = A - a_0, \qquad \left(\int A\right)' = A.$$

Lemma 3.5. *If F, G belong to $\mathbb{A}[[X]]$ with $\mathbb{A}$ a $\mathbb{Q}$-algebra and $g_0 = 0$ then*

$$F(G(X))' = F'(G(X))G'(X).$$

Proof. The coefficient of X^i in both sides of the identity is the same as in the composition of the polynomials obtained by truncating F and G after their ith coefficient (i.e., taking all the coefficients of index larger than i as 0). It is therefore sufficient to prove the result for polynomials, for which it is classical. □

Example 3.4. Classical examples of formal power series that behave like their counterpart from analysis are the exponential and logarithm.

Definition 3.6.

$$\exp(X) := \sum_{n\ge 0} \frac{X^n}{n!}, \quad \ln(1+X) := \int (1+X)^{-1}.$$

It is a good exercise to prove the expected identities:

$$(\exp(X))' = \exp(X), \quad \ln(\exp(X)) = X, \quad \exp(\ln(1+X)) = 1 + X.$$

The first one follows by inspection of the coefficient of X^i for arbitrary i.

For the second one, by differentiation

$$(\ln(\exp(X))' = \frac{1}{1+(\exp(X)-1)}(\exp(X))' = \exp(X)^{-1}\exp(X) = 1.$$

Integrating shows that $\ln(\exp(X)) = X - c$, where c is the constant coefficient of $\ln(\exp(X))$, which by the definition of ln, is 0.

For the last one, let $F = \exp(\ln(1+X))$. Then by differentiation

$$(1+X)F' = F.$$

Extracting the coefficient of X^i in this identity gives

$$(i+1)f_{i+1} + if_i = f_i, \quad i \geq 0.$$

With $i = 0$, we get $f_0 = f_1$. Next, $i = 1$ gives $f_2 = 0$ and from there follows that $f_i = 0$ for $i > 2$. The constant coefficient f_0 is obtained as 1 by the definition of exp.

Proposition 3.5 (Taylor expansion)**.** *Let $\mathbb{A}$ be a* commutative $\mathbb{Q}$*-algebra and F, G, H be formal power series in $\mathbb{A}[[X]]$ with $g_0 = h_0 = 0$. Then*

$$F(G+H) = F(G) + F'(G)H + F''(G)\frac{H^2}{2!} + \cdots.$$

Proof. The proof is as the previous one. Extracting coefficients reduces to proving the identity for polynomials, for which it is the classical Taylor formula. □

In this proposition it is important that $\mathbb{A}$ be commutative. Otherwise, even with a simple F like X^2, one gets

$$(G+H)^2 = G^2 + GH + HG + H^2$$

and $GH + HG$ is not necessarily equal to $2GH$.

3.5.6 Other Applications of Division of Power Series

This section is not used in the rest of the course and can be skipped in a first reading.

Logarithm

From

$$\ln(1+U)' = \int \frac{U'}{1+U},$$

we deduce an algorithm for the computation of the logarithm of a power series with constant term 1: $U' + O(X^n)$ is obtained in $O(n)$ operations in $\mathbb{A}$ from U; the division by $1+U$ costs $O(\mathsf{M}(n))$ operations in $\mathbb{A}$ by the previous proposition; the final integration has again complexity $O(n)$. Thus in total, the logarithm is computed in $O(\mathsf{M}(n))$ operations. This idea goes back to Brent in 1976. The code is as follows.

```
Log:=proc(s,x,n)
  series(Int(series(diff(s,x)/s,x,n-1),x),x,n)
end:
```

Exponential

Brent also obtained the same complexity for the computation of the exponential of a power series F. The idea is to use Newton's iteration on the equation

$$\phi(y) = F - \log y,$$

which leads to

$$\mathcal{N}(y) = y + y(F - \log y).$$

The proof of quadratic convergence follows the same lines as before: if $y = e^F(1+S)$, then

$$\log y = F + \log(1+S) = F + S + O(S^2)$$

and

$$\mathcal{N}(y) = e^F(1+S)(1-S+O(S^2)) = e^F(1+O(S^2)).$$

So if $S = O(X^n)$ then $\mathcal{N}(y) - e^F = O(X^{2n})$.

The corresponding Maple code is straightforward

```
Exp:=proc(s,x,n)
local y;
  if n=1 then 1
  else
    y:=convert(Exp(s,x,ceil(n/2)),polynom);
    series(y+y*(s-Log(y,x,n)),x,n)
  fi
end:
```

The complexity analysis gives

$$C(n) \leq C(\lceil n/2 \rceil) + \lambda \mathsf{M}(n) + O(n)$$

(one recursive call, one computation of a logarithm, one multiplication, several additions) and the conclusion $C(n) = O(\mathsf{M}(n))$ follows again from the 'master theorem'.

A combination of the logarithm and the exponential also allows to compute arbitrary powers via $F^\alpha = \exp(\alpha \log F)$ in $O(\mathsf{M}(n))$ when this is well-defined.

3.6 More Newton Iterations

This section is not used in the rest of the course, except in the tutorial that follows this lecture.

3.6.1 Heron's Iteration for Square Roots

Now the function whose zero is sought is

$$\phi(y) = y^2 - a$$

and the iteration becomes

$$y_{n+1} = \frac{1}{2}\left(y_n + \frac{a}{y_n}\right),$$

which was known to Heron of Alexandria (ca 10–70 AD). The corresponding pictures are displayed in Fig. 3.5. and the code becomes

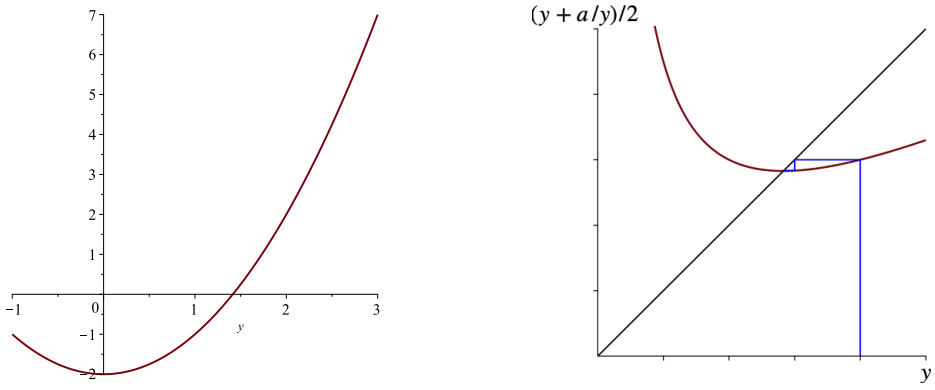


Figure 3.5: Heron's iteration

```
Heron:=proc(a)
local y,s:=a;
  while y<>s do y,s:=s,(s+a/s)/2 od;
  s
end:
```

Example 3.5. The point $y_0 = a$ is always a possible starting point for this iteration. In the case when $a = 2$, the first few iterates are

$$\begin{aligned} y_0 &= 2. \\ y_1 &= \mathbf{1.}50000000000000000000 \\ y_2 &= \mathbf{1.41}66666666666666666667 \\ y_3 &= \mathbf{1.41421}568627450980392 \\ y_4 &= \mathbf{1.4142135623}7468991063 \\ y_5 &= \mathbf{1.41421356237309504880} \end{aligned}$$

The quadratic convergence is easily seen:

$$y_{n+1} - \sqrt{a} = \frac{1}{2}\left(y_n - 2\sqrt{a} + \frac{a}{y_n}\right) = \frac{1}{2y_n}(y_n - \sqrt{a})^2.$$

3.6.2 Square Root

Consider a truncated power series

$$a = 1 + a_1 X + \cdots + O(X^m).$$

Then its square root can be defined by composition with the binomial $(1 + X)^{1/2}$. Recall that Heron's iteration

$$y_{n+1} = \frac{1}{2}\left(y_n + \frac{a}{y_n}\right)$$

satisfies

$$y_{n+1} - \sqrt{a} = \frac{(y_n - \sqrt{a})^2}{2y_n}.$$

This leads again to a divide-and-conquer algorithm for power series

```
Heron:=proc(a,x,n)
local f;
  if n=1 then sqrt(coeff(a,x,0))
  else
    f:=thisproc(a,x,ceil(n/2));
    f:=convert(f,polynom);
    series((f+a/f)/2,x,n)
  fi
end:
```

The complexity obeys $C(m) \leq C(\lceil m/2 \rceil) + O(\mathsf{M}(m))$ and thus $C(m) = O(\mathsf{M}(m))$. A similar method, not presented here, applies to the computation of square roots of integers.

3.6.3 Solving Equations by Newton Iteration: An Example

Kepler's equation reads

$$\phi(y) = y - \frac{1}{2}\sin y - x = 0.$$

It has a solution with $y(0) = 0$ that admits a Taylor expansion $T(x)$ at $x = 0$. Such an expansion can be found by Newton iteration. This iteration reads

$$y_{n+1} = y_n - \frac{y_n - \frac{1}{2}\sin y_n - x}{1 - \frac{1}{2}\cos y_n}.$$

It is a consequence of the general Proposition 3.6 below that

$$T = y_n + O(x^k) \Rightarrow T = y_{n+1} + O(x^{2k}).$$

Thus in order to compute T at precision N, it is only necessary to compute truncations at smaller order of previous iterates. If N is a power of 2, we get the successive iterates

$$\begin{aligned}
y_0 &= 0,\\
y_1 &= 2x + \mathrm{O}\left(x^2\right),\\
y_2 &= 2x - \frac{4}{3}x^3 + \mathrm{O}\left(x^4\right),\\
y_3 &= 2x - \frac{4}{3}x^3 + \frac{44}{15}x^5 - \frac{2696}{315}x^7 + \mathrm{O}\left(x^8\right),\\
y_4 &= 2x - \frac{4}{3}x^3 + \frac{44}{15}x^5 - \frac{2696}{315}x^7 + \frac{81068}{2835}x^9 - \frac{16129352}{155925}x^{11} + \frac{2397755992}{6081075}x^{13} - \frac{90535608368}{58046625}x^{15} + \mathrm{O}\left(x^{16}\right).
\end{aligned}$$

As in the case of the Heron iteration or the computation of inverses, if the target order is N, one does not compute expansions up to a power of 2, but uses a divide-and-conquer iteration.

3.6.4 More General Equations

We now want generalize the previous example to solve an equation

$$\Phi(X, Y) = 0,$$

where Φ is itself a power series in $\mathbb{A}[[X]][[Y]]$. Again, Newton's iteration applies under mild conditions, leading to:

Proposition 3.6. *Let $\Phi \in \mathbb{A}[[X]][[Y]]$ be such that $\Phi(0,0) = 0$ and $\partial\Phi/\partial Y(0,0)$ is invertible in $\mathbb{A}$. Then*

1. *there exists a unique $S \in \mathbb{A}[[X]]$ such that $S(0) = 0$ and $\Phi(X, S(X)) = 0$;*
2. *if $F \in \mathbb{A}[[X]]$ satisfies $S - F = O(X^n)$ for some $n > 0$ then $S - \mathcal{N}(F) = O(X^{2n})$, where*

$$\mathcal{N}(F) = F - \frac{\Phi(X,F)}{\partial\Phi/\partial Y(X,F)}.$$

As a consequence of this proposition, if $\Phi(X,F)$ and $\partial\Phi(X,F)$ can be computed in $O(\mathsf{M}(n))$ operations, then the solution of the equation can also be obtained in $O(\mathsf{M}(n))$ operations. This is in particular the case whenever Φ is a polynomial with respect to Y, i.e., $\Phi \in \mathbb{A}[[X]][Y]$.

Proof of existence. The proof is constructive. Define the sequence S_n by $S_0 = 0$ and $S_{k+1} = \mathcal{N}(S_k)$. By induction, $S_k(0) = 0$ for all k, which makes the iteration well-defined, as only composition with series with constant term 0 are allowed and the $\partial\Phi/\partial Y$ is invertible at (X, S_k) because its constant term $\partial Phi/\partial Y(0,0)$ is.

Next, by definition of S_{k+2}, we see that

$$S_{k+2} - S_{k+1} = O(\Phi(X, S_{k+1}).$$

Finally, by Taylor expansion (Proposition 3.5),

$$\Phi(X, S_{k+1}) = \Phi(X, S_k) + \frac{\partial\Phi}{\partial Y}(X, S_k)(\mathcal{S}_{k+1} - S_k) + O((S_{k+1} - S_k)^2) = O((S_{k+1} - S_k)^2).$$

Thus by induction $S_{k+1} - S_k = O(X^{2^k})$, showing the convergence of S_k. Taking limits shows that the limit S satisfies $S(0) = 0$ and $\Phi(X,S) = 0$. □

Proof of quadratic convergence. We can now perform a Taylor expansion at S:

$$\Phi(X,S) = 0 = \Phi(X,F) + \frac{\partial\Phi}{\partial Y}(X,F)(S-F) + O((S-F)^2).$$

It follows that

$$\begin{aligned} S &= F - \frac{\Phi(X,F)}{\frac{\partial\Phi}{\partial Y}(X,F)} + O((S-F)^2) \\ &= \mathcal{N}(F) + O((S-F)^2), \end{aligned}$$

proving the quadratic convergence. □

Proof of uniqueness. If $\Phi(X,F) = 0$ then the equation above shows that

$$S - F = O((S-F)^2),$$

whose only solution with $\operatorname{val} F > 0$ is 0, by considering the valuations. □

3.6.5 Inverse Composition

Finding G such that $G(F) = F(G) = X$ when F is a power series with $F(0) = 0$ and $F'(0)$ invertible is the special case of the previous result with $\Phi(X,Y) = X - F(Y)$. The complexity of the resulting algorithm is only $O(\mathsf{M}(n)\log n)$ since the composition with F can be obtained by composition of formal power series.

Additional bibliography

The following 3 references mentioned before contain variants of the content of this lecture:

Alin Bostan et al. *Algorithmes Efficaces en Calcul Formel*. Auto-édition, Sept. 2017. ISBN: 979-10-699-0947-2. URL: https://hal.archives-ouvertes.fr/AECF/

Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. 3rd edition. New York: Cambridge University Press, 2013, pp. xiv+785. URL: http://www.cambridge.org/fr/knowledge/isbn/item1170826

Donald E. Knuth. *The Art of Computer Programming*. 3rd edition. Vol. 2: Seminumerical Algorithms. Computer Science and Information Processing. Reading, Mass.: Addison-Wesley Publishing Co., 1997, pp. xiv+762

For more information on the use of Newton's iteration numerically for elementary functions, a good reference is

Jean-Michel Muller. *Elementary Functions: Algorithms and Implementations*. 2nd edition. Birkhäuser Boston, 2006

Lecture 4

Euclid's Algorithm

Summary

Euclid's algorithm computes the gcd. The extended Euclidean algorithm also computes the cofactors. These allow to compute modular inverses. In turn, this is used in an effective version of the Chinese remainder theorem. A truncated version of the extended Euclidean algorithm also solves the rational reconstruction problem, including the important special case of Padé approximants. All these algorithms have an efficient version in quasi-linear complexity.

4.1 Algorithm and Properties

4.1.1 GCDs and Euclidean Functions

Greatest common divisors can be defined in arbitrary commutative rings.

Definition 4.1. *Let A, B be two elements of a commutative ring $\mathbb{A}$. An element $G \in \mathbb{A}$ is a* greatest common divisor *(GCD) of A and B if $G|A$, $G|B$ and for all C,*

$$C|A \text{ and } C|B \Longrightarrow C|G.$$

While this gives a definition, more is needed to obtain an algorithm.

Definition 4.2. *A function $\nu : \mathbb{A} \to \mathbb{N} \cup \{-\infty\}$ is called a* Euclidean function *if for all A, B in $\mathbb{A}$, with $B \neq 0$, there exists Q, R in $\mathbb{A}$ such that*

$$A = QB + R, \quad \nu(R) < \nu(B).$$

We will use the following notation: $Q = A \operatorname{quo} B$, $R = A \operatorname{rem} B$.

The only examples that will be used in this course are

$$\begin{aligned} \mathbb{A} &= \mathbb{Z} \quad \text{and} \quad \nu = |\cdot|; \\ \mathbb{A} &= \mathbb{K}[x] \quad \text{and} \quad \nu = \deg. \end{aligned}$$

4.1.2 Euclid's Algorithm

The classical version of Euclid's algorithm is given in Fig. 4.1[1].

[1] Euclid lived in Alexandria ca 300 B.C.; his algorithm has been called by D. Knuth "the granddaddy of all algorithms". The version for polynomials was known to Stevin in 1585.

Input: A, B
Output: a gcd of A and B
1. $R_0 = A; R_1 := B; i = 1;$
2. While $R_i \neq 0$
   $R_{i+1} = R_{i-1}$ rem R_i;
   $i := i + 1$
3. Return R_{i-1}.

Figure 4.1: Euclid's Algorithm

The proof that this algorithm is correct relies on the following variants and invariants.

Lemma 4.1. *If $C|A$ and $C|B$, then for any i, $C|R_i$. Moreover, for any $i > 1$, $\nu(R_i) < \nu(R_{i-1})$.*

Proof. The proof is by induction. If $C|R_i$ and $C|R_{i-1}$, then from

$$R_{i-1} = QR_i + R_{i+1}$$

it follows that $C|(R_{i-1} - QR_i) = R_{i+1}$. The property on ν comes from the definition of rem. □

Corollary 4.1. *Euclid's algorithm is correct and the number of iterations of the loop is bounded by $\nu(B)$.*

Proof. First, termination of the algorithm follows from the fact that ν decreases at each step when $R_i \neq 0$. This also explains the bound on the number of iterations.

By the previous lemma, any common divisor of R_{i-1} and R_i is also a common divisor of R_i and R_{i+1}. Thus by induction, for all $i > 0$, $\gcd(R_i, R_{i-1}) = \gcd(R_{i+1}, R_i)$. Therefore if k is the final value of i, $\gcd(A, B) = \gcd(R_{k-1}, R_k) = \gcd(R_{k-1}, 0) = R_{k-1}$, proving the correctness of the algorithm. □

4.1.3 Extended Euclidean Algorithm and Bézout's Identity

A small change in the algorithm allows to compute cofactors. The algorithm, already known to Euler in 1748, becomes that of Fig. 4.2.

Input: A, B
Output: a gcd G of A and B
and U, V s.t. $UA + VB = G$
1. $R_0 = A; R_1 = B; i = 1;$
2. $U_0 = 1; V_0 = 0; U_1 = 0; V_1 = 1;$
3. While $R_i \neq 0$
   $Q_i = R_{i-1}$ quo R_i;
   $R_{i+1} = R_{i-1} - Q_i R_i$
   $U_{i+1} = U_{i-1} - Q_i U_i; V_{i+1} = V_{i-1} - Q_i V_i;$
   $i = i + 1$
4. Return $R_{i-1}, U_{i-1}, V_{i-1}$.

Figure 4.2: Extended Euclidean Algorithm

Its properties come from a simple invariant.

Lemma 4.2. *For any $i \geq 0$ during the algorithm,*

$$U_i A + V_i B = R_i.$$

Proof. The identity is satisfied for $i = 0$ and $i = 1$ by definition of U_0, V_0, U_1, V_1. Next, if the identity is satisfied for $i-1$ and i, then

$$\begin{aligned} R_{i+1} &= R_{i-1} - Q_i R_i \\ &= (U_{i-1}A + V_{i-1}B) - Q_i(U_i A + V_i B) \\ &= (U_{i-1} - Q_i U_i)A + (V_{i-1} - Q_i V_i)B \\ &= U_{i+1}A + V_{i+1}B. \end{aligned}$$

□

Corollary 4.2. *If G, U, V is the output of the extended Euclidean algorithm, then $G = \gcd(A, B)$ and*

$$G = UA + VB. \tag{4.1}$$

Equation (4.1) is called *Bézout's identity.*

Proof. The property on the gcd has been proved for Euclid's algorithm. That part of the computation is unaffected by the difference with the extended Euclidean algorithm. The identity then comes from the invariant of the lemma. □

The main application of Bézout's identity is that when $G = 1$, V is the inverse of B modulo A (and U is the inverse of A modulo B). This allows to compute divisions in the ring $\mathbb{A}/(A)$ when they are possible (ie, when the gcd is 1).

Example 4.1. With $A = 1 + X^2$ and $B = a + bX$, Bézout's identity gives

$$b^2(1 + X^2) + (a - bX)(a + bX) = a^2 + b^2.$$

Thus the inverse of $a + bX$ modulo $1 + X^2$ is $(a - bX)/(a^2 + b^2)$. One recognizes the formula for the inverse of complex numbers:

$$\frac{1}{a + ib} = \frac{a - ib}{a^2 + b^2}, \quad (i^2 = -1).$$

Example 4.2. The computation of a modular inverse is the first step of the proof that algebraic power series are differentially finite (and also the first step of the corresponding algorithm), in Lecture 6.

4.1.4 Bounds on Degrees

For the complexity analysis and for the design of efficient algorithms later, it is useful to control the sizes of intermediate quotients and remainders. We focus on the case of polynomials, which is easier.

Proposition 4.1. *When $\mathbb{A} = \mathbb{K}[x]$ and $G \notin \{A, B\}$, the output (G, U, V) of the extended Euclidean algorithm satisfies*

$$\deg(U) + \deg(G) < \deg B, \quad \deg(V) + \deg(G) < \deg A.$$

The proof is by induction from the following invariants.

Lemma 4.3. *When $\mathbb{A} = \mathbb{K}[x]$ and $\deg A \geq \deg B$, the polynomials in the extended Euclidean algorithm satisfy*

$$\deg(U_i) + \deg(R_{i-1}) = \deg B, \quad \deg(V_i) + \deg(R_{i-1}) = \deg A, \quad i \geq 2.$$

Note that for $i = 1$, the first equality in the lemma becomes $\deg 0 + \deg A = \deg B$ and does not always hold.

Proof. For $i \geq 2$, we first prove

$$\deg R_{i-1} > \deg R_i, \tag{4.2}$$
$$\deg Q_i = \deg R_{i-1} - \deg R_i > 0, \tag{4.3}$$
$$\deg U_{i+1} = \deg U_i + \deg Q_i, \tag{4.4}$$
$$\deg V_{i+1} = \deg V_i + \deg Q_i. \tag{4.5}$$

The first one follows from the property of the Euclidean division on the degrees, giving $\deg R_{i+1} < \deg R_i$ for $i \geq 1$ in the loop. The second one then follows again from Euclidean division since $R_i \neq 0$.

Next, we prove two properties by induction: the equalities Eqs. (4.4) and (4.5) and the inequalities $\deg U_i \geq \deg U_{i-1}$ and $\deg V_i \geq \deg V_{i-1}$. For $i = 2$, the values $U_2 = 1, U_1 = 0, V_2 = -Q_1 \neq 0, V_1 = 1$ show that the inequalities hold. The identities follow since $\deg Q_i > 0$ for $i \geq 2$. In turn, they imply $\deg U_{i+1} \geq \deg U_i$ and $\deg V_{i+1} \geq \deg V_i$.

For $i = 2$, the identity

$$A = Q_1 B + R_2$$

gives $U_2 = 1$, $V_2 = -Q_1$. Then $\deg U_2 + \deg R_1 = 0 + \deg B = \deg B$; $\deg V_2 + \deg B = \deg Q_1 + \deg B = \deg Q_1 B = \deg A$, since $\deg R_2 < \deg B \leq \deg A$. Thus the identities in the lemma hold for $i = 2$.

From Eqs. (4.3) to (4.5), an induction gives

$$\deg U_{i+1} + \deg R_i = \deg U_i + \deg Q_i + \deg R_{i-1} - \deg Q_i = \deg B$$
$$\deg V_{i+1} + \deg R_i = \deg V_i + \deg Q_i + \deg R_{i-1} - \deg Q_i = \deg A.$$

□

Proof of Proposition 4.1. Consider first the case when $\deg A \geq \deg B$. Let k be the last value of i in the loop before R_i becomes 0, so that the output of the algorithm is $(G, U, V) = (R_k, U_k, V_k)$. If $k = 1$, then $G = B$, which contradicts the hypothesis. Thus $k \geq 2$ and the lemma applies, giving

$$\deg U + \deg R_{k-1} = \deg B, \quad \deg V + \deg R_{k-1} = \deg A.$$

The proof of the inequalities comes from Eq. (4.2) and $R_k = G$.

If $\deg A < \deg B$, then the first iteration of the loop produces $Q_1 = 0$ and $(R_1, R_2) = (B, A)$, from where the algorithm proceeds as before, whence the result. □

4.1.5 Arithmetic Complexity

The complexity of the extended Euclidean algorithm is quadratic.

Proposition 4.2. *If $\mathbb{A} = \mathbb{K}[x]$, Euclid's algorithm and the extended Euclidean algorithm use $O((\deg A + \deg B)^2)$ arithmetic operations in $\mathbb{K}$.*

Proof. Without loss of generality we assume $\deg A \geq \deg B$: otherwise, the first step of the algorithm obtains 0 as a quotient and $R_2 = R_0$ with no arithmetic operation, which corresponds to swapping A and B.

Euclid's algorithm performs the successive divisions of R_{i-1} by R_i. By Lemma 3.1 in Lecture 3, the complexity of these operations is bounded by

$$\sum_i (\deg R_{i-1} - \deg R_i + 1)(2 \deg R_i + 1).$$

Lemma 4.3 shows that $\deg R_i \le \deg B$ and thus this sum is bounded by

$$(2\deg B + 1)\sum_i (\deg R_{i-1} - \deg R_i + 1).$$

The last sum telescopes and the number of steps is bounded by $\deg B$ (Corollary 4.1), leading to

$$(2\deg B + 1)(\deg A + \deg B) = O((\deg A + \deg B)^2).$$

The extended Euclidean algorithm performs the same operations, plus the computations of $U_{i-1} - Q_i U_i$ and $V_{i-1} - Q_i V_i$. The cost of these operations is at most

$$2\sum_{i\ge 1}(\deg Q_i + 1)(\deg U_i + 1 + \deg V_i + 1) :$$

one multiplication and one subtraction for each coefficient. Since $\deg U_i \le \deg B$ and $\deg V_i \le \deg A$ (by Lemma 4.3), this is bounded by

$$2(\deg A + \deg B + 2)\sum_{i\ge 1}(\deg Q_i + 1).$$

Now, from Eq. (4.3),

$$\sum_{i\ge 1}\deg Q_i = \deg Q_1 + \deg R_1 \le \deg A$$

and the bound on the number of iterations, the total number of operations for the computation of the U_i and V_i is bounded by

$$2(\deg A + \deg B + 2)(\deg A + \deg B) = O((\deg A + \deg B)^2). \qquad \square$$

4.2 Chinese Remainder Theorem

Thanks to the computation of modular inverses (Bézout coefficients), the Chinese remainder theorem[2] becomes an algorithm. We first recall what is meant by this.

4.2.1 Theorem

Theorem 4.1. *Let $I_1, \dots, I_k$ be ideals of the commutative ring $\mathbb{A}$ such that $I_i + I_j = \mathbb{A}$ for all $i \ne j$. Given $r_1, \dots, r_k$ in $\mathbb{A}$, there exists $A \in \mathbb{A}$ such that*

$$A \equiv r_i \bmod I_i, \quad i = 1, \dots, k.$$

Proof. We first show that for each i, there exists $y_i \in \mathbb{A}$ such that $y_i \equiv 1 \bmod I_i$ and $y_i \equiv 0 \bmod I_j$ for $j \ne i$. From this, the result follows by considering

$$A = r_1 y_1 + \cdots + r_k y_k.$$

Consider first the case $i = 1$. Since for all $j \ne 1$, $I_j + I_1 = \mathbb{A}$, there exists $b_j \in I_j$ and $a_j \in I_1$ such that $b_j + a_j = 1$. This implies that

$$\prod_{j\ne 1}(a_j + b_j) = 1.$$

Modulo I_1, the jth term of the product is equal to b_j, and thus

$$y_1 := \prod_{j\ne 1} b_j \equiv 1 \bmod I_1,$$

while y_1 is 0 modulo I_j for $j \ne 1$. The construction of the other y_i is the same. $\square$

[2] Already known to Sun Tsu in the 3rd century, for integers.

4.2.2 Algorithm when $A = \mathbb{K}[x]$ or $\mathbb{Z}$

When $A = \mathbb{K}[x]$ or $\mathbb{Z}$, the ideals are given by generators p_i (these rings are principal), such that $\gcd(p_i, p_j) = 1$ for $i \neq j$. An algorithm that follows the proof closely is as follows:

1. Compute the product $M = \prod_i p_i$
2. Deduce $m_i = M/p_i$ for all i
3. For all i, compute u_i such that $u_i m_i + v_i p_i = 1$
4. Deduce $c_i = r_i u_i \bmod p_i$
5. Return $\sum c_i m_i$

By design $u_i m_i$ is 1 modulo p_i and 0 modulo p_j for $j \neq i$. It plays the role of y_i in the proof of the theorem. Instead of returning $\sum r_i u_i m_i$, the algorithm first computes c_i which reduces the size of the result. The result is still correct, since the product $c_i m_i$ is 0 modulo p_j for $j \neq i$ and $r_i u_i m_i \bmod p_i$.

In Maple in the case of polynomials, this algorithm becomes

```
# r list of residues
# p list of polynomials
# x variable
crt_naive:=proc(r,p,x)
local k:=nops(r),M,i,m,u,v,c;
  M:=convert(p,`*`);
  for i to k do
    m[i]:=quo(M,p[i],x);
    gcdex(m[i],p[i],x,u[i],v[i]);
    c[i]:=rem(r[i]*u[i],p[i],x)
  od;
  add(c[i]*m[i],i=1..k)
end:
```

4.2.3 Fast Algorithm

Interpolation is the special case of the Chinese remainder theorem where $I_i = (x - a_i)$ for all i. We now generalize the fast algorithm for interpolation of Lecture 3.

Product tree

The first step is the construction of a product tree (Fig. 4.3) whose leaves are the polynomials p_i, that are not restricted by be linear any longer.

The Maple code is almost the same as in the interpolation algorithm

```
ProductTree:=proc(p,x)
local i,left,right;
  if nops(p)=1 then p
  else
    i:=ceil(nops(p)/2);
    left:=thisproc(p[1..i],x);
    right:=thisproc(p[i+1..-1],x);
    [expand(left[1]*right[1]),left,right]
  fi
end:
```

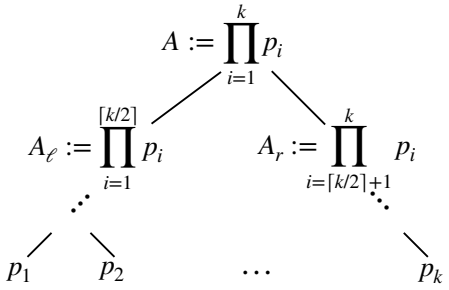


Figure 4.3: Product Tree

Let n denote the sum of the degrees of the p_i. Then the complexity of the recursive computation of this tree obeys the recurrence

$$C(p_1, \ldots, p_k) \leq C(p_1, \ldots, p_{\lceil k/2 \rceil}) + C(p_{\lceil k/2 \rceil + 1}, \ldots, p_k) + \mathsf{M}(n)$$

since besides the construction of both subtrees, only one multiplication is needed, with a result of degree n. By induction, it follows that the complexity is bounded by the sum of $\mathsf{M}(n_i)$ where n_i are the degrees of the polynomials at the nodes of the tree. On each of the $\lceil \log_2 k \rceil$ levels of the tree, this sum is n. In view of the hypothesis on the multiplication function, it follows that the complexity is

$$C(p_1, \ldots, p_k) = O(\mathsf{M}(n) \log k) = O(\mathsf{M}(n) \log n).$$

Fast simultaneous modular reduction

Given $P, p_1, \ldots, p_k$, the next step computes $(P \operatorname{rem} p_1, \ldots, P \operatorname{rem} p_k)$ using the product tree. This is done in exactly the same way as in the fast multipoint evaluation of Lecture 3, and with the same proof.

```
EvalPol:=proc(P,T,x)
  if nops(T)=1 then P
  else thisproc(rem(P,T[2][1],x),T[2],x),
    thisproc(rem(P,T[3][1],x),T[3],x)
  fi
end:

SimultMod:=proc(P,p,x)
  [EvalPol(P,ProductTree(p,x),x)]
end:
```

The complexity inequality is the same as before:

$$C(p_1, \ldots, p_k) \leq C(p_1, \ldots, p_{\lceil k/2 \rceil}) + C(p_{\lceil k/2 \rceil + 1}, \ldots, p_k) + O(\mathsf{M}(n)).$$

The term $O(\mathsf{M}(n))$ comes from both Euclidean divisions, each of complexity bounded by $O(\mathsf{M}(n))$. With the same inequality, the same bound $O(\mathsf{M}(n) \log n)$ follows.

Final linear combination

The last step of the algorithm performs the following computation

$$(c_1, \ldots, c_k) \mapsto c_1 \frac{M}{p_1} + \cdots + c_k \frac{M}{p_k},$$

with $\deg c_i < \deg p_i$. This is, again, achieved by a divide-and-conquer algorithm using the product tree:

```
LinComb:=proc(c,T,x)
local k:=nops(c),r:=ceil(k/2);
  if k=1 then c[1]
  else expand(T[3][1]*thisproc(c[1..r],T[2],x)
    +T[2][1]*thisproc(c[r+1..-1],T[3],x))
  fi
end:
```

Letting A_r and A_ℓ denote the polynomials at the roots of the right and left subtrees, this algorithm implements the identity

$$\sum_{i=1}^{k} c_i \frac{M}{p_i} = A_r \sum_{i \leq \lceil k/2 \rceil} c_i \frac{A_\ell}{p_i} + A_\ell \sum_{i > \lceil k/2 \rceil} c_i \frac{A_r}{p_i}.$$

Again, the complexity satisfies the same inequality, obtained by considering the degrees of the polynomials in each summand. Whence again a complexity in $O(\mathsf{M}(n) \log n)$ for that step.

Simultaneous Inverses

It is also possible to compute all the u_i of the algorithm (the inverses of M/p_i modulo p_i) simultaneously as follows

```
SimultInv:=proc(p,M,x)
local k:=nops(p),L,i,m,u,v;
        L:=SimultMod(M,map(t->t^2,p),x);      # L_i = M mod p_i^2
        for i to k do
                m[i]:=quo(L[i],p[i],x);       # m_i = M/p_i mod p_i
                gcdex(m[i],p[i],x,u[i],v[i]) # u_i = 1/(M/p_i) mod p_i
        od;
        [seq(u[i],i=1..k)]
end:
```

The point of the algorithm is that it is not possible to compute all the $m_i = M/p_i$ in a quasi-linear complexity, since their total size is quadratic. Instead, the algorithm first computes $L_i = M \bmod p_i^2$ by the simultaneous modular reduction above in $O(\mathsf{M}(n) \log n)$ operations. From there, the m_i are obtained from the L_i by an exact division in complexity $O(\mathsf{M}(\deg p_i))$, whence for a total cost of $O(\mathsf{M}(n))$, again by the usual hypothesis on the multiplication function. The modular inverse is then computed and they are all returned. For the complexity of the modular inverse, we use the fast algorithm that will be given below, which leads to a complexity in

$$O(\mathsf{M}(\deg p_i) \log \deg p_i)$$

for each p_i, and therefore again for a total of $O(\mathsf{M}(n) \log n)$.

Conclusion

Finally, the fast algorithm for the Chinese remainder theorem is

```
# r list of residues
# p list of polynomials
# x variable
crt_fast:=proc(r,p,x)
local T,M,u,i,c,k:=nops(r);
  T:=ProductTree(p,x); M:=T[1];
  u:=SimultInv(p,M,x);
  for i to k do
    c[i]:=rem(r[i]*u[i],p[i],x)
  od;
  LinComb([seq(c[i],i=1..k)],T,x)
end:
```

Gathering the complexity analyses of all the steps, we have proved.

Proposition 4.3 (Borodin & Moenck 1974)**.** *Given $p_1, \ldots, p_k$ in $\mathbb{K}[x]$ such that $\gcd(p_i, p_j) = 1$ for $i \neq j$ and given $r_1, \ldots, r_k$ in $\mathbb{K}[x]$ with $\deg r_i < \deg p_i$, one can compute $A \in \mathbb{K}[x]$ such that $A = r_i \bmod p_i$ for all i in $O(\mathsf{M}(n) \log n)$, where $n = \deg p_1 + \cdots + \deg p_k$.*

Similar algorithm and complexity result exist for integers.

4.3 Rational Reconstruction

Another important application of the extended Euclidean algorithm is a solution of the rational reconstruction problem.

4.3.1 Problem and Special Cases

Given A, B in $\mathbb{K}[x]$ with $\deg B < n = \deg A$ and given $k \in \{1, \ldots, n\}$, the problem of rational reconstruction is to find R, V in $\mathbb{K}[x]$ such that

$$\gcd(V, A) = 1,\ \deg R < k,\ \deg V \leq n - k,\ \frac{R}{V} \equiv B \bmod A. \tag{$\mathcal{RR}$}$$

Useful and important approximants are obtained as special cases.

Definition 4.3. *A* Padé approximant *of type (m, ℓ) of a power series $S \in \mathbb{K}[[x]]$ is a rational function R/V with $\deg R \leq m$, $\deg V \leq \ell$, $V(0) \neq 0$ and*

$$\frac{R}{V} = S + O(x^{m+\ell+1}).$$

This is a special case of rational reconstruction with $A = x^n, B = S, k = m + 1, n = m + \ell + 1$.

Definition 4.4. *Given $((u_1, v_1), \ldots, (u_n, v_n))$ in $\mathbb{K}$ and $k \in \{1, \ldots, n\}$, the problem of* Cauchy interpolation *is to find a rational function R/V with $\deg R < k$, $\deg V] \leq n - k$ such that*

$$\frac{R(u_i)}{V(u_i)} = v_i, \quad i = 1, \ldots, n.$$

This is a special case of rational reconstruction with $A = \prod (x - u_i)$ and B the interpolation *polynomial* of the points.

4.3.2 Berlekamp-Massey Algorithm

This algorithm uses a Padé approximant to reconstruct a linear recurrence with constant coefficients from its first terms. By Proposition 3.2 from Lecture 3, if u_n satisfies the linear recurrence

$$u_{n+d} = a_{d-1}u_{n+d-1} + \cdots + a_0 u_n,$$

then the generating series $\sum u_n X^n$ satisfies

$$U(X) = \sum_{n\geq 0} u_n X^n = \frac{N_0(X)}{\overline{P}(X)},$$

with $\deg N_0 < d$ and

$$\overline{P}(X) = 1 - a_{d-1}X - \cdots - a_0 X^d.$$

This fraction can be reconstructed from the first $2d$ coefficients of the sequence. The algorithm is due to Berlekamp and Massey around 1965.

```
# L: [u_0, u_1, ...]
BM:=proc(L,u,n)
local N:=nops(L),m:=iquo(N,2),S,R,i,den,d;
        S:=series(add(L[i+1]*x^i,i=0..N-1)+O(x^N),x,N); # sum u_i x^i
        R:=numapprox[pade](S,x,[N-m-1,m]); # Padé of type (N-floor(N/2)-1, floor(N/2))
        den:=denom(R);
        d:=max(degree(numer(R),x)+1,degree(den,x));
        add(coeff(den,x,d-i)*u(n+i),i=0..d)=0
end:
```

Example 4.3. The simplest example is the sequence of Fibonacci numbers:

```
> L:=[1,1,2,3,5,8]:
> BM(L,u,n);
```

$$u(n) + u(n+1) - u(n+2) = 0$$

The choice of d in the algorithm is due to the fact that the order of the recurrence must be increased if it is not satisfied for the first coefficients: the sequence defined by $u_0 = 1$ and $u_n = F_{n+1}$ for $n \geq 1$ has for generating series

$$\sum_{n\geq 0} u_n X^n = \frac{1 - X^2}{1 - X - X^2}$$

but it does not satisfy the recurrence above.

```
> BM([1,op(L)],u,n);
```

$$u(n+1) + u(n+2) - u(n+3) = 0$$

4.3.3 Existence and Uniqueness of Rational Approximants

Uniqueness is clear: if

$$\frac{R_1}{V_1} = \frac{R_2}{V_2} \bmod A,$$

with $\deg A = n$, $\deg R_i < k$ and $\deg V_i \leq n-k$, then reducing to the same denominator implies that

$$A|(R_1V_2 - R_2V_1),$$

but since that polynomial has degree smaller than n it follows that it must be 0, which implies that $R_1/V_1 = R_2/V_2$.

Unfortunately, the problem of rational reconstruction does not always have a solution.

Example 4.4. With $n = 3, k = 2, A = X^3, B = X^2 + 1$, we are looking for V of degree at most 1, say $V = aX + b$. Then

$$(aX + b)(1 + X^2) = b + ax + bX^2 \bmod X^3.$$

The constraint $\gcd(V, A) = 1$ implies that $b \neq 0$ but then the right-hand side cannot be of degree $< k$.

However, the following simpler problem does admit a solution: given A, B in $\mathbb{K}[x]$ with $\deg B < n = \deg A$ and given $k \in \{1, \ldots, n\}$, find R, V in $\mathbb{K}[x]$ such that

$$\deg R < k, \ \deg V \leq n - k, \ R \equiv BV \bmod A. \qquad (\mathcal{SRR})$$

In other words, the constraint $\gcd(A, V) = 1$ is dropped and then, since V is no longer necessarily invertible modulo A, the equation $R/V \equiv B \bmod A$ is replaced by $R \equiv BV \bmod A$ which is always defined.

This problem admits a solution since, by extracting coefficients of powers of X, it is seen to be equivalent to a linear system of n equations in the $n + 1$ coefficients of R and V.

4.3.4 Computation by Euclid's Algorithm

Theorem 4.2 (McEliece & Shearer 1978)**.** *If $(R_i, U_i, V_i)_i$ is the sequence computed by the extended Euclidean algorithm on (A, B) and j is the smallest index such that $\deg R_j < k$, then*

1. *(R_j, V_j) is a solution of the problem $(\mathcal{SRR})$;*
2. *if the problem $(\mathcal{RR})$ admits a solution (R, V), then $R/V = R_j/V_j$.*

Proof. If

$$U_jA + V_jB = R_j,$$

and $\deg R_j < k \leq \deg R_{j-1}$, then $R_j \equiv V_jB \bmod A$ and by Lemma 4.3,

$$\deg V_j = \deg A - \deg R_{j-1} \leq \deg A - k.$$

This proves the first part of the theorem.

If (R, V) is a solution of the problem $(\mathcal{RR})$, there exists U such that $UA + VB = R$. In matrix form

$$\begin{pmatrix} U & V \\ U_j & V_j \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} R \\ R_j \end{pmatrix}.$$

If that matrix was invertible, then Cramer's rule would imply

$$A = \frac{RV_j - R_jV}{UV_j - U_jV}$$

whose numerator has degree smaller than n, a contradiction. It follows that $UV_j = U_jV$, whence

$$\frac{R}{V} = \frac{U}{V}A + B = \frac{U_j}{V_j}A + B = \frac{R_j}{V_j}.$$

□

4.4 Fast Euclidean Algorithm

In terms of complexity, the main result of this lecture is that all the computations presented so far can be performed in $O(\mathsf{M}(n)\log n)$ operations. An analogous result for integers is available. Here is a more formal statement.

Theorem 4.3. *Given A, B in $\mathbb{K}[x]$ with $n = \deg A > \deg B$, let $(U_i, V_i, R_i)_i$ be the sequence computed by the extended Euclidean algorithm. Then, in $O(\mathsf{M}(n)\log n)$ operations in $\mathbb{K}$, one can compute:*

1. *the sequence $(Q_1, \dots, Q_\ell)$ of quotients;*
2. *$\gcd(A, B)$ and the corresponding Bézout cofactors;*
3. *given $k < n$, (U_j, V_j, R_j) with j minimal such that $\deg R_j < k$.*

The rest of this lecture is devoted to an algorithm leading to a proof this result, under a simplifying assumption.

4.4.1 $1 \Rightarrow 2\&3$

In matrix form, once the quotients are known, the inner loop of the algorithm performs a simple product

$$\begin{pmatrix} R_{i+1} & U_{i+1} & V_{i+1} \\ R_i & U_i & V_i \end{pmatrix} = \underbrace{\begin{pmatrix} -Q_i & 1 \\ 1 & 0 \end{pmatrix}}_{M_i} \begin{pmatrix} R_i & U_i & V_i \\ R_{i-1} & U_{i-1} & V_{i-1} \end{pmatrix}, \quad i \geq 1,$$

$$= M_i \cdots M_1 \begin{pmatrix} B & 0 & 1 \\ A & 1 & 0 \end{pmatrix}. \tag{4.6}$$

It follows that given the polynomials Q_i, the matrix product $M_i \cdots M_1$ can be computed by a product tree in $O(\mathsf{M}(D)\log D)$ operations, where

$$D = \sum_{j \leq i} \deg Q_j \leq \deg A,$$

this last inequality coming from Eq. (4.3). If ℓ is the number of quotients computed by the algorithm, this product gives

$$\begin{pmatrix} 0 & U_{\ell+1} & V_{\ell+1} \\ G & U & V \end{pmatrix},$$

the last row thus contains the gcd and the Bézout cofactors. (Observe that this also provides $\operatorname{lcm}(A, B) = U_{\ell+1}A = -V_{\ell+1}B$.)

Finding j minimal such that $\deg R_j < k$ can also be done from the quotients: by Eq. (4.3),

$$\deg R_j = \deg A - \sum_{i=1}^{j} \deg Q_j.$$

Once the index is found, the matrix product above gives the result in the desired complexity.

4.4.2 Quotients from Truncation

A first idea towards a fast algorithm is that it is not necessary to know all the coefficients of polynomials in order to compute their quotient. We use the following.

Notation 4.1. *If A is in $\mathbb{K}[x]$, we write*

$$A|_d = A + O(X^{\deg A - d - 1}).$$

This is the polynomial obtained from A by keeping only the $d+1$ coefficients of highest degree.

We can now compare the quotient of A, B and that of $A|_d, B|_d$.

Lemma 4.4. *For $\delta = \deg A - \deg B$ and $d \geq \delta$,*

$$A|_d = QB|_d + R', \text{ with } R'|_{d'} = R|_{d'},\ d' = d - (\deg A - \deg R).$$

In words, the quotient is the same and the first d' coefficients of the remainder are the same.

Proof. Recall from Lecture 3 that Euclidean division of A by B can be computed by series expansion from

$$\underbrace{X^{\deg A}A(1/X)}_{\mathrm{rev}(A)} = \underbrace{X^{\deg B}B(1/B)}_{\mathrm{rev}(B)}\underbrace{X^{\deg Q}Q(1/X)}_{\mathrm{rev}(Q)} + \underbrace{X^{\deg R}R(1/X)}_{\mathrm{rev}(R)}X^{\deg A - \deg R},$$

Replacing A and B by $A|_d$ and $B|_d$ amounts to truncating this expansion at order $O(X^{d+1})$. The quotient is recovered since $d + 1 > \delta = \deg Q$. The second term in the right-hand side of the equality above has valuation $\deg A - \deg R$ and thus all the monomials of degree between $\deg A - \deg R$ and d are recovered. This shows that $R|_{d'} = R'|_{d'}$. □

Corollary 4.3. *For $d \geq \deg A + \deg B - 2\deg R_j$, the first j steps of the extended Euclidean algorithm on $(A|_d, B|_d)$ compute $Q_1, \ldots, Q_j$.*

Proof. For $i = 1, \ldots, j$, if $d_i \geq \deg R_{i-1} + \deg R_i - 2\deg R_j$, which is nonnegative for $i \leq j$ since the sequence of degrees decreases, then the lemma shows that one can compute Q_i from $R_{i-1}|_{d_i}$ and $R_i|_{d_i}$ and also obtain $R_{i+1}|_{d_{i+1}}$ with

$$\begin{aligned} d_{i+1} &= d_i - (\deg R_{i-1} - \deg R_{i+1}) \\ &\geq (\deg R_{i-1} + \deg R_i - 2\deg R_j) - (\deg R_{i-1} - \deg R_{i+1}) \\ &= \deg R_i + \deg R_{i+1} - 2\deg R_j. \end{aligned}$$

Starting with d_1 given by the bound in the corollary is therefore sufficient to compute all Q_i, $i = 1, \ldots, j$. □

4.4.3 Divide and Conquer

The basis of the efficient algorithm is to compute the product of the matrices

$$M_i = \begin{pmatrix} -Q_i & 1 \\ 1 & 0 \end{pmatrix}$$

from Eq. (4.6) by a divide-and-conquer approach from the formula

$$M_k \cdots M_1 = \underbrace{(M_k \cdots M_{\lceil k/2 \rceil + 1})}_{\mathcal{M}_2} \cdot \underbrace{(M_{\lceil k/2 \rceil} \cdots M_1)}_{\mathcal{M}_1},$$

and reconstruct the required R_i by

$$\begin{pmatrix} R_{i+1} \\ R_i \end{pmatrix} = M_i \cdots M_1 \begin{pmatrix} B \\ A \end{pmatrix}. \tag{4.7}$$

The outline of the algorithm reflects this splitting:

1. compute $\mathcal{M}_1$ from $A|_d, B|_d$ for an appropriate d;
2. compute $(R_{\lceil k/2 \rceil + 1}, R_{\lceil k/2 \rceil})$ by $\mathcal{M}_1 \cdot \begin{pmatrix} B \\ A \end{pmatrix}$;
3. compute $\mathcal{M}_2$ from $(R_{\lceil k/2 \rceil + 1}|_{d'}, R_{\lceil k/2 \rceil}|_{d'})$ for an appropriate d';
4. return the product $\mathcal{M}_2 \cdot \mathcal{M}_1$.

4.4.4 Half-GCD for a normal degree sequence

The most common situation is the following one, where the algorithm is simpler to state and prove.

Definition 4.5. *The sequence R_i produced by Euclid's algorithm has a* normal degree sequence *when*

$$\deg R_i = n - i, \quad i = 0, \dots, n.$$

In that case, the algorithm outlined above becomes

```
# Input: (A,B) polynomials with deg A > deg B
#        k in {1,...,deg A}
#        x variable
# Output: the quotients (Q_1,...,Q_k)
#         and the product of matrices M_k ··· M_1.
hgcdnormalseq:=proc(A,B,k,x)
local m,d,L1,L2,P1,P2,Q,R;
        if k=1 then
                Q:=quo(A,B,x);
                [Q],Matrix([[-Q,1],[1,0]])
        else
                m:=ceil(k/2);d:=2*m-1;
                L1,P1:=thisproc(cut(A,x,d),cut(B,x,d),m,x);
                R:=map(expand,P1.Vector([B,A]));
                m:=k-m;d:=2*m-1;
                L2,P2:=thisproc(cut(R[2],x,d),cut(R[1],x,d),m,x);
                [op(L2),op(L1)],map(expand,P2.P1)
        fi
end:
```

the procedure `cut` invoked here is simply

```
cut:=proc(p,x,d)
local n:=degree(p,x),i;
  add(coeff(p,x,i)*x^i,i=n-d..n)
end:
```

Note that the only division takes place at the leaves of the recursion, when $k = 1$. When invoked with $k = n$, the algorithm returns the gcd.

We now show the correctness of the algorithm. Since $m = \lceil k/2 \rceil$, Corollary 4.3 shows that $d \geq \deg A + \deg B - 2\deg R_m = n + n - 1 - 2(n-j) = 2m - 1$ is sufficient to compute $Q_1, \dots, Q_m$. From there, the vector (R_{m+1}, R_m) is computed by the matrix product (4.7). Since the sequence is assumed to have normal degree, $\deg R_m = n - m, \deg R_{m+1} = n - m - 1$ and Corollary 4.3 shows that $d \geq \deg R_m + \deg R_{m+1} - 2\deg R_k = n - m + n - m - 1 - 2(n-k) = 2(k-m) - 1$ is sufficient to compute $Q_{m+1}, \dots, Q_k$ and thus the second recursive call is correct too.

Let $C(i,k)$ be the complexity of the algorithm when invoked with (R_{i-1}, R_i) and k. Then the algorithm leads to

$$C(i,k) \leq C(i, \lceil k/2 \rceil) + C(i + \lceil k/2 \rceil, k - \lceil k/2 \rceil) + \lambda \mathsf{M}(3n/2),$$

where the bound $\mathsf{M}(3n/2)$ comes from the fact that A and B have degree $\leq n$ and the product $M_k \cdots M_1$ has entries of degree bounded by $k = \lceil n/2 \rceil$. Since the degrees of the R_i decrease with i, the complexity

$C(i + \lceil k/2 \rceil, k - \lceil k/2 \rceil)$ of the second call is bounded by that of the first one, leading to

$$C(i,k) \leq 2C(i, \lceil k/2 \rceil) + \lambda \mathsf{M}(3n/2),$$

and thus by the 'master theorem',

$$C(1,n) = O(\mathsf{M}(n) \log n).$$

This is due to Moenck (1973).

4.4.5 Half-GCD Algorithm – General Case

In the general case, one cannot depend on the normal degree sequence anymore. Instead of relying on an *a priori* knowledge of the degrees, decisions are made from the degrees of the polynomials computed in the recursive calls. The specification of the algorithm when called with A, B and k is that it returns $Q_1, \ldots, Q_h$ with h such that

$$\deg R_h \geq \deg A - k > \deg R_{h+1}.$$

Proving this invariant leads to the correctness of the algorithm. The proof is technical and omitted here.

```
# Input: (A,B) polynomials with deg A > deg B
#                  k in {1,...,deg A}
#                  x variable
# Output: the quotients (Q_1,...,Q_k)
#         and the product of matrices M_h ··· M_1 .
#          with h s.t. deg R_h ≥ deg A − k > deg R_{h+1} .
hgcd:=proc(A,B,k,x)
local dA:=degree(A,x),dB:=degree(B,x),Q,m,d,L1,L2,P1,P2,R;
        if k<dA-dB then [],Matrix([[1,0],[0,1]])
        elif k<=dA-dB then
                Q:=quo(A,B,x);
                [Q],Matrix([[-Q,1],[1,0]])
        else
                m:=max(ceil(k/2),dA-dB); # prevents infinite loops
                d:=2*m+dB-dA;
                L1,P1:=thisproc(cut(A,x,d),cut(B,x,d),m,x);
                R:=map(expand,P1.Vector([B,A]));
                if R[1]=0 or degree(R[1],x)<dA-k then return L1,P1 fi;
                m:=k-(dA-degree(R[2],x));
                d:=degree(R[1],x)+degree(R[2],x)-2*(dA-k);
                L2,P2:=thisproc(cut(R[2],x,d),cut(R[1],x,d),m,x);
                [op(L1),op(L2)],map(expand,P2.P1);
        fi
end:
```

This algorithm was developed first for integers by Lehmer (1938), analyzed by Knuth and Schönhage in (1971) and extended to polynomials by Brent, Gustavson, Yun (1980).

Additional bibliography

For the algorithmic aspects, the same two books mentioned in the introduction can be used. For a detailed proof of the half-gcd algorithm, a good source is

Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic complexity theory*. Vol. 315. Grundlehren der Mathematischen Wissenschaften. Berlin: Springer-Verlag, 1997, pp. xxiv+618

Lecture 5

Resultant

Summary

Resultants are polynomials that encode the non-triviality of gcds. They have applications in non-linear elimination and in the computation with roots of polynomials when the polynomials are used as a data-structure. Geometrically, they correspond to projection.

In this lecture, $\mathbb{K}$ denotes an arbitrary field, $\mathbb{K}[X]_{<k}$ the set of polynomials of $\mathbb{K}[X]$ whose degree is smaller than k. The polynomials A and B are

$$A = a_0X^n + \cdots + a_n \in \mathbb{K}[X],$$
$$B = b_0X^m + \cdots + b_m \in \mathbb{K}[X].$$

They are assumed to have degrees n and m and that $(m, n) \neq (0, 0)$. The resultant is a polynomial in the coefficients a_i, b_j that vanishes exactly when A and B have common factors of positive degree.

5.1 Definition

5.1.1 First Criterion for the Existence of Common Factors

Lemma 5.1. *The polynomials A and B in $\mathbb{K}[X]$ have a common factor of positive degree if and only if there exist two polynomials U and V in $\mathbb{K}[X]$, not both 0, such that*

$$AU + BV = 0, \qquad \deg V < \deg A, \quad \deg U < \deg B.$$

Proof. Assume $G = \gcd(A, B)$ has positive degree. Then there exist $\hat{A}, \hat{B}$ such that $A = G\hat{A}$, $B = G\hat{B}$ and $A\hat{B} - B\hat{A} = 0$ satisfies the degree constraints.

Conversely, we proceed by contradiction. If A and B do not have a common factor of positive degree, by Bézout's identity (previous lecture), there exists U_1, V_1 such that

$$U_1A + V_1B = 1.$$

Multiplying $AU + BV = 0$ by V_1 gives

$$\begin{aligned} V_1AU + V_1BV &= 0 \\ V_1AU + (1 - U_1A)V &= 0 \\ V &= A(U_1V - V_1U). \end{aligned}$$

The polynomial V is not 0 (otherwise the assumptions imply $A = 0$, which is incompatible with $\gcd(A, B) = 1$). The last identity then implies $\deg V \geq \deg A$, a contradiction. □

5.1.2 Sylvester's Matrix

The previous lemma leads to considering the following matrix, introduced by Sylvester in 1840.

Definition 5.1. *The* Sylvester matrix *of the polynomials A, B, denoted* $\mathrm{Syl}(A, B)$*, is the matrix of the linear map*

$$\begin{aligned} \phi_{A,B} : \mathbb{K}[X]_{<m} \times \mathbb{K}[X]_{<n} &\to \mathbb{K}[X]_{<n+m} \\ (U, V) &\mapsto UA + VB \end{aligned}$$

in the bases $((X^{m-1}, 0), \ldots, (1, 0), (0, X^{n-1}), \ldots, (0, 1))$ *and* $(X^{n+m-1}, \ldots, 1)$.

This matrix has $m + n$ rows and columns and its expression in terms of the coefficients of A and B is

$$\mathrm{Syl}(A, B) := \begin{pmatrix} a_0 & & & b_0 & & \\ a_1 & \ddots & & b_1 & \ddots & \\ \vdots & & a_0 & \vdots & & b_0 \\ a_n & & a_1 & b_m & & b_1 \\ & \ddots & \vdots & & \ddots & \vdots \\ & & a_n & & & b_m \end{pmatrix},$$

with the first m columns involving the a_i; the b_i only appear in the last n.

By Lemma 5.1, Sylvester's matrix is related to the gcd as follows.

Lemma 5.2. $\deg \gcd(A, B) > 0 \Leftrightarrow \det \mathrm{Syl}(A, B) = 0$.

Indeed, the existence of U, V in Lemma 5.1 is equivalent to the kernel of Sylvester's matrix being nontrivial, equivalently, the matrix being singular.

A more precise statement is possible.

Lemma 5.3. $\deg \gcd(A, B) = \dim \ker \phi_{A,B}$.

Proof. Let G be a gcd of A and B, let $\hat{A}, \hat{B}$ be defined by $A = G\hat{A}, B = G\hat{B}$. By definition of the gcd, $\hat{A}$ and $\hat{B}$ are relatively prime. Then if (U, V) is in the kernel of $\phi_{A,B}$, $UA+VB = 0 = G(U\hat{A}+V\hat{B}) = U\hat{A}+V\hat{B} = 0$. Since $\hat{A}, \hat{B}$ are relatively prime, this is equivalent to the existence of a polynomial Q such that $U = Q\hat{B}, V = -Q\hat{A}$. Since $\deg U < m$,, this polynomial Q has degree at most $m - \deg \hat{B} = m - (m - \deg G) = \deg G$. Thus the elements in the kernel of $\phi_{A,B}$ are in one-to-one correspondence with the polynomials in $\mathbb{K}[x]_{<\deg G}$. □

5.1.3 Resultant

Definition 5.2. *The resultant of the polynomials A, B in $\mathbb{K}[X]$ is the determinant of their Sylvester matrix. It is denoted* $\mathrm{Res}(A, B)$.

The resultant was introduced by Bézout in 1764.

Proposition 5.1. *There is a polynomial $R_{n,m} \in \mathbb{Z}[A_0, \ldots, A_n, B_0, \ldots, B_m]$ such that*

$$\mathrm{Res}(A, B) = R_{n,m}(a_0, \ldots, b_n).$$

In other words,

$$\deg \gcd(A, B) > 0 \Leftrightarrow R_{n,m}(a_0, \ldots, b_n) = 0.$$

This polynomial $R_{n,m}$ is thus a 'universal polynomial'. Over any field $\mathbb{K}$, a pair of polynomials of degrees n and m have a nontrivial gcd if and only if their coefficients make $R_{n,m}$ vanish.

Proof. This comes from writing the formula for the determinant as a sum over permutations. □

Example 5.1. If $A = aX + b, B = cX + d$, then

$$\mathrm{Syl}(A, B) = \begin{pmatrix} a & c \\ b & d \end{pmatrix} \to \mathrm{Res}(A, B) = ad - bc.$$

Example 5.2. Let $A = aX^2 + bX + c$ and $B = A' = 2aX + b$. Then

$$\mathrm{Syl}(A, B) = \begin{pmatrix} a & 2a & 0 \\ b & b & 2a \\ c & 0 & b \end{pmatrix} \to \mathrm{Res}(A, B) = -a(b^2 - 4ac).$$

The second factor is the discriminant. This example is generalized below.

Example 5.3. If $A = X - a$ and $B = b_0X^m + \cdots + b_m$, then

$$\mathrm{Syl}(A, B) = \begin{pmatrix} 1 & 0 & \dots & 0 & b_0 \\ -a & 1 & 0 & \vdots & b_1 \\ 0 & -a & \ddots & 0 & b_2 \\ & & \ddots & 1 & \vdots \\ 0 & \dots & 0 & -a & b_m \end{pmatrix} \to \mathrm{Res}(A, B) = B(a).$$

This is obtained by expanding with respect to the first row and induction. A factor $(-1)^m$ comes from b_0 being in the column of index $m + 1$, it is cancelled by another factor $(-1)^m$ coming from the number of $-a$ in the diagonal of the remaining minor.

5.2 Properties of the Resultant

5.2.1 Resultant as a Linear Combination

When A and B are relatively prime, the resultant is a common multiple of the denominators of the Bézout cofactors. More precisely,

Proposition 5.2. *There exist U, V in $\mathbb{K}[X]$ such that*

$$\mathrm{Res}(A, B) = UA + VB.$$

The coefficients of U and V are integer polynomials in the coefficients of A and B.

Proof. By linear combinations of the rows of Sylvester's matrix, it follows that

$$\mathrm{Res}(A,B) = \begin{vmatrix} a_0 & & & b_0 & & \\ a_1 & \ddots & & b_1 & \ddots & \\ \vdots & & a_0 & \vdots & & b_0 \\ a_n & & a_1 & b_m & & b_1 \\ & \ddots & \vdots & & \ddots & \vdots \\ X^m A(X) & & A(X) & X^n B(X) & & B(X) \end{vmatrix}$$

and the conclusion follows by expanding with respect to the last row. □

5.2.2 Homogeneity

Lemma 5.4. *If $\lambda \neq 0$,*

$$\mathrm{Res}(\lambda A, B) = \lambda^m \mathrm{Res}(A,B), \qquad \mathrm{Res}(A, \lambda B) = \lambda^n \mathrm{Res}(A,B).$$

Proof. This is a consequence of the multilinearity of the determinant. □

5.2.3 Quasi-Symmetry

Lemma 5.5.

$$\mathrm{Res}(B,A) = (-1)^{mn} \mathrm{Res}(A,B).$$

Proof. This is the consequence of the fact that the determinant is an alternating form. □

5.2.4 Poisson's Formula

Proposition 5.3. *The linear map*

$$\begin{aligned} \text{Let } \psi_B : \mathbb{K}[X]/(A) &\to \mathbb{K}[X]/(A) \\ V &\mapsto VB \bmod A \end{aligned}$$

has for determinant $a_0^{-m} \mathrm{Res}(A,B)$.

Proof. Introduce the map of Euclidean division:

$$\begin{aligned} \varepsilon_A : \mathbb{K}[X]_{<n+m} &\to \mathbb{K}[X]_{<m} \times \mathbb{K}[X]_{<n} \\ P &\mapsto (Q,R) \text{ s.t. } P = QA + R \end{aligned}$$

and consider the composition $\epsilon_A \circ \phi_{A,B}$. We have

$$\varepsilon_A \circ \phi_{A,B} : (U,V) \xrightarrow{\phi_{A,B}} UA + VB \xrightarrow{\varepsilon_A} (U + (VB \operatorname{quo} A), VB \bmod A).$$

The corresponding matrices are

$$\varepsilon_A : \left(\begin{array}{ccc|c} a_0^{-1} & & 0 & \\ & \ddots & & 0 \\ * & & a_0^{-1} & \\ \hline & * & & \begin{matrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{matrix} \end{array}\right) \qquad \varepsilon_A \circ \phi_{A,B} : \left(\begin{array}{ccc|c} 1 & & 0 & \\ & \ddots & & * \\ 0 & & 1 & \\ \hline & 0 & & M_{\psi_B} \end{array}\right)$$

Taking the determinants on both sides gives the result. □

5.2.5 Multiplicativity

Proposition 5.4. *If B_1, B_2 in $\mathbb{K}[X]$ have positive degree, then*

$$\operatorname{Res}(A, B_1 B_2) = \operatorname{Res}(A, B_1) \operatorname{Res}(A, B_2).$$

Proof. This is a consequence of Poisson's formula, since $\psi_{B_1 B_2} = \psi_{B_2} \circ \psi_{B_1}$. □

5.2.6 Expression in the Roots

This is possibly the most useful result in this lecture.

Theorem 5.1. *If $A = a_0(X - \alpha_1) \cdots (X - \alpha_n)$, $B = b_0(X - \beta_1) \cdots (X - \beta_m)$, then*

$$\operatorname{Res}(A, B) = a_0^m b_0^n \prod_{i=1}^{n} \prod_{j=1}^{m} (\alpha_i - \beta_j) = a_0^m \prod_{i=1}^{n} B(\alpha_i) = (-1)^{mn} b_0^n \prod_{j=1}^{m} A(\beta_j).$$

Proof. The second equality is a consequence of homogeneity, multiplicativity and Example 5.3. The first one follows from the expansion of B and the third one from the first one. □

5.2.7 Discriminants

The discriminant is a polynomial in the coefficients of a polynomial A that detects multiple roots.

Definition 5.3. *The discriminant of $A = a_0 \prod_{i=1}^{n} (X - \alpha_i)$ is*

$$\Delta(A) = a_0^{2n-2} \prod_{i<j} (\alpha_i - \alpha_j)^2 = (-1)^{n(n-1)/2} a_0^{2n-2} \prod_{i \neq j} (\alpha_i - \alpha_j).$$

Proposition 5.5. *$\Delta(A)$ is a polynomial in the coefficients of A and*

$$\operatorname{Res}(A, A') = (-1)^{n(n-1)/2} a_0 \Delta(A).$$

Proof. Again by the theorem,

$$\operatorname{Res}(A, A') = a_0^{n-1} \prod_{j} A'(\alpha_j),$$

and by expansion of the derivative,

$$A'(\alpha_j) = a_0 \prod_{i \neq j} (\alpha_j - \alpha_i).$$

Proposition 5.1 shows that $a_0 \Delta(A)$ is a polynomial in the coefficients of A. But a_0 divides the resultant, since it is a factor of the first row of Sylvester's matrix. □

5.3 Computation of the Resultant

Since the resultant characterizes the polynomials with a nontrivial gcd, it does not come as a surprise that there is a strong relation between the resultant and Euclid's algorithm.

Proposition 5.6. *If $A = BQ + R$, with $n \geq m$ and $n > \deg R$, then*

$$\operatorname{Res}(A, B) = (-1)^{mn}\, b_0^{n-\deg R} \operatorname{Res}(B, R).$$

Proof. Starting from the determinant of Sylvester's matrix,

$$\operatorname{Res}(A, B) = \begin{vmatrix} a_0 & & & b_0 & & \\ a_1 & \ddots & & b_1 & \ddots & \\ \vdots & & a_0 & \vdots & & b_0 \\ \vdots & & a_1 & \vdots & & b_1 \\ a_n & & \vdots & b_m & & \vdots \\ & \ddots & \vdots & & \ddots & \vdots \\ & & a_n & & & b_m \end{vmatrix},$$

linear combinations of the columns using the coefficients of Q give

$$\operatorname{Res}(A, B) = \begin{vmatrix} 0 & & & b_0 & & \\ r_0 & \ddots & & b_1 & \ddots & \\ \vdots & \ddots & 0 & \vdots & & b_0 \\ \vdots & & r_0 & \vdots & & b_1 \\ r_d & & \vdots & b_m & & \vdots \\ & \ddots & \vdots & & \ddots & \vdots \\ & & r_d & & & b_m \end{vmatrix}.$$

By mn transposition of columns, this gives

$$\operatorname{Res}(A, B) = (-1)^{mn} \begin{vmatrix} b_0 & & & 0 & & \\ b_1 & \ddots & & r_0 & \ddots & \\ \vdots & & b_0 & \vdots & \ddots & 0 \\ \vdots & & b_1 & \vdots & & r_0 \\ b_m & & \vdots & r_d & & \vdots \\ & \ddots & \vdots & & \ddots & \vdots \\ & & b_m & & & r_d \end{vmatrix}.$$

This last matrix has an upper right block of zeroes of dimension $n - \deg R$. The corresponding upper left block is triangular, with determinant $b_0^{n-\deg R}$. The lower right block is Sylvester's matrix $\operatorname{Syl}(B, R)$. □

A first simple algorithm for the resultant follows from a recursive use of the proposition:

```
euclres:=proc(A,B,x)
local R:=A,S:=B,res:=1,T;
        while degree(S,x)>0 do
                T:=rem(R,S,x);
                res:=res*(-1)^(degree(R,x)*degree(S,x))
                        *lcoeff(S,x)^(degree(R,x)-degree(T,x));
                (R,S):=(S,T)
        od;
        if S=0 then 0           # non-trivial gcd
        else S^degree(R,x)*res  # last res. is a constant
        fi
end:
```

Recall from the previous lecture that it is possible to compute all the quotients in the Euclidean algorithm by the half-gcd algorithm. Using these quotients gives a fast algorithm for the resultant:

```
# deg A ≥ deg B
fastres:=proc(A,B,x)
local d,Q,alpha,ell,i;
        d[0]:=degree(A,x);alpha[0]:=lcoeff(A,x);
        Q:=hgcd(A,B,d[0],x)[1]; ell:=nops(Q);
        for i to ell do d[i]:=d[i-1]-degree(Q[i],x) od; # d_i = deg R_i
        if d[ell]>0 then return 0 fi;                   # α_i = lcoeff(R_i)
        for i to ell do alpha[i]:=alpha[i-1]/lcoeff(Q[i],x) od;
        (-1)^add(d[i-1]*d[i],i=1..ell-1)*alpha[ell]^d[ell-1]*
                mul(alpha[i]^(d[i-1]-d[i+1]),i=1..ell-1)
end:
```

First, the quotients are computed in $O(\mathsf{M}(n)\log n)$ operations. From the quotients, the degrees of the remainders are computed in $O(n)$ operations, and similarly for their leading coefficients. Finally, using the formula of the proposition gives the result in $O(n)$ more operations. We have thus obtained:

Theorem 5.2. *Given A, B in $\mathbb{K}[x]$ with $n = \deg A > \deg B$, then the resultant $\mathrm{Res}(A,B)$ can be computed in $O(\mathsf{M}(n)\log n)$ operations in $\mathbb{K}$.*

5.4 Bivariate Resultant

Up to now, the resultant has been defined for polynomials with coefficients in a field. An extension to a more general setting is as follows.

Definition 5.4. *If A, B belong to the ring $\mathbb{A} = \mathbb{K}[X,Y]$, one defines $\mathrm{Res}_X(A,B) = \det \mathrm{Syl}(A,B)$, with A, B viewed in $\tilde{\mathbb{A}} = \mathbb{K}(Y)[X]$.*

By the proof of Proposition 5.2, there exists U, V in $\mathbb{A}$ such that

$$\mathrm{Res}_X(A,B) = UA + VB \in \mathbb{K}[Y].$$

It follows that if there exists x, y such that $A(x,y) = B(x,y) = 0$, then $R(y) = 0$. We now consider the converse question: if y is such that $R(y) = 0$, does there exists x such that $A(x,y) = B(x,y) = 0$?

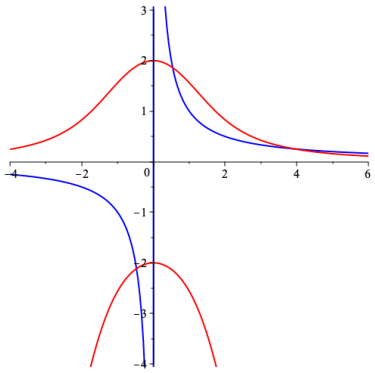


Figure 5.1: Intersection of two Curves

5.4.1 Geometric Interpretation

First, observe that under appropriate conditions, the value of the resultant at a point is the value of the resultant between the polynomials evaluated at that point.

Proposition 5.7 (Specialization). *If A, B are in $\mathbb{K}[X, Y]$ and $R = \mathrm{Res}_X(A, B)$, then for any $y \in \mathbb{K}$,*

$$a_0(y) \neq 0 \text{ and } b_0(y) \neq 0 \Rightarrow \mathrm{Res}(A(X, y), B(X, y)) = R(y).$$

Proof. The conditions imply that $\deg A(X, y) = \deg_X A$ and similarly for B. Thus the Sylvester matrix of $A(X, y), B(X, y)$ is the specialization of that of $A(X, Y), B(X, Y)$ and the conclusion follows from the same specialization property for the determinant. □

We can now conclude on the existence of common zeros.

Theorem 5.3 (Extension). *If $\mathbb{K}$ is algebraically closed, A, B are in $\mathbb{K}[X, Y]$ and there exists $y \in \mathbb{K}$ such that*

$$a_0(y)b_0(y) \neq 0, \quad \mathrm{Res}_X(A, B)(y) = 0,$$

then there exists $x \in \mathbb{K}$ such that

$$A(x, y) = B(x, y) = 0.$$

Note that by exchanging the roles of X, Y one would similarly obtain a polynomial that vanishes at the abscissas of the common solutions.

Proof. By the previous proposition,

$$\mathrm{Res}_X(A, B)(y) = \mathrm{Res}(A(X, y), B(X, y)).$$

If this vanishes, $\gcd(A(X, y), B(X, y))$ has positive degree in X. Since $\mathbb{K}$ is algebraically closed, it has roots x, where $A(x, y) = B(x, y) = 0$. □

Geometrically, this means that the resultant vanishes on the projection of an intersection: its zeroes correspond either to points of the projection on the y-axis or to points where one of the leading coefficients vanishes (corresponding to projective solutions). This reasoning extends to $\mathbb{K}[X, Y_1, \ldots, Y_k]$ for $k \geq 1$.

Example 5.4. With $A = XY - 1$, $B = X^2Y + Y^2 - 4$, the set defined by $A = B = 0$ in $\mathbb{R}^2$ is given by the intersections of the blue and red curve in Fig. 5.1.

The computation of the resultant gives

$$\operatorname{Res}_X(A,B) = \begin{vmatrix} Y & 0 & Y \\ -1 & Y & 0 \\ 0 & -1 & Y^2-4 \end{vmatrix} = Y(Y^3 - 4Y + 1).$$

Its roots correspond either to the vanishing of the common factor Y of the leading coefficients of A, B or to the ordinate of the intersections, i.e., their projection on the y-axis.

Another observation that can be made in this example is that the resultant is not necessarily the smallest degree polynomial that can be obtained by linear combination of A, B with polynomial coefficients. Indeed, one can find that

$$YB - (XY + 1)A = Y^3 - 4Y + 1.$$

(An algorithm for doing so will be given in the Lecture 9.)

5.4.2 Construction of Polynomials with Prescribed Roots

The use of bivariate resultants gives tools to construct polynomials that cancel sums, products, or quotients of roots of polynomials, without knowing these roots.

Proposition 5.8. *If* $A = a_0(X - \alpha_1)\cdots(X - \alpha_n)$, $B = b_0(X - \beta_1)\cdots(X - \beta_m)$, *then*

$$\operatorname{Res}_X(A(T - X), B(X)) = (-1)^{mn} a_0^m b_0^n \prod_{i,j} (T - (\alpha_i + \beta_j))$$

$$\operatorname{Res}_X(A(X), X^m B(T/X)) = a_0^m b_0^n \prod_{i,j} (T - \alpha_i \beta_j)$$

If $\gcd(A, C) = 1$,

$$\operatorname{Res}_X(A(X), C(X)T - B(X)) = a_0^{\max(\deg B - \deg C, 0)} \operatorname{Res}(A, C) \prod_i \left(T - \frac{B(\alpha_i)}{C(\alpha_i)}\right).$$

Proof. This is a corollary to Theorem 5.1.

The polynomial $A(T - X)$ has for roots $T - \alpha_i$. The first formula is then obtained by the first formula of the theorem.

Substitution in the expression for B gives

$$X^m B(T/X) = b_0 \prod (T - \beta_i X).$$

From this and the second formula of the theorem, the second result follows.

Still by the second formula of the theorem, using the fact that $\deg_X(C(X)T - B(X)) = \max(\deg B, \deg C)$, one obtains

$$\operatorname{Res}_X(A(X), C(X)T - B(X)) = a_0^{\max(\deg B, \deg C)} \prod_{i=1}^{n} (C(\alpha_i)T - B(\alpha_i)).$$

Since $\gcd(A, C) = 1$, none of the $C(\alpha_i)$ is 0. They can thus be factored out, giving

$$a_0^m \prod_{i=1}^{n} C(\alpha_i) \prod_{i=1}^{n} \left(T - \frac{B(\alpha_i)}{C(\alpha_i)}\right).$$

The theorem also gives

$$\operatorname{Res}(A, C) = a_0^m \prod_{i=1}^{n} C(\alpha_i),$$

which gives the last formula. □

5.4.3 Degree Bounds

These bounds are used in the design and analysis of an efficient algorithm for the resultant.

Proposition 5.9. *Let A, B be in $\mathbb{K}[X, Y]$.*

1. *If $d_A = \deg_Y A$, $d_B = \deg_Y B$, then $\deg \operatorname{Res}(A, B) \le m d_A + n d_B$;*
2. *if $d_A = \deg A$, $d_B = \deg B$, then $\deg \operatorname{Res}(A, B) \le d_A d_B$.*

Proof. The first statement follows from the expansion of the determinant.

For the second one, considering the degree of the entry (i, j) of the Sylvester matrix gives

$$\deg(\operatorname{Syl}(A, B)_{i,j}) \le \begin{cases} d_A - n + i - j & (j \le m), \\ d_B + i - j & (j > m). \end{cases}$$

It follows that for any permutation σ,

$$\begin{aligned} \sum_{j=1}^{m+n} \deg(\operatorname{Syl}(A, B)_{\sigma(j),j}) &\le \sum_{j=1}^{m} (d_A - n + \sigma(j) - j) + \sum_{j=m+1}^{m+n} (d_B + \sigma(j) - j) \\ &\le m d_A + n d_B - mn + \sum \sigma(j) - \sum j \\ &= d_A d_B - (d_A - n)(d_B - m) \le d_A d_B. \end{aligned}$$

□

5.4.4 Bivariate Resultant

In view of the importance of the resultant in bivariate elimination, we give a last result on the complexity of this operation. Consider two polynomials A, B in $\mathbb{K}[X, Y]$ with $\deg_X B = m \le \deg_X A = n$ and $\deg_Y A$, $\deg_Y B$ of degree at most d. By Proposition 5.9, the degree of $\operatorname{Res}_X(A, B)$ is bounded by $D := 2nd$. A simple evaluation-interpolation algorithm is as follows

1. Evaluate the coefficients of A and B at $D + 1$ points;
2. Evaluate $D + 1$ univariate resultants;
3. Interpolate.

With the complexities of the resultant, of multipoint evaluation and interpolation, this whole algorithm has complexity

$$O\left(n \frac{D}{d} \mathsf{M}(d) \log(d) + D \mathsf{M}(n) \log n + \mathsf{M}(D) \log D\right) = \tilde{O}(n^2 d),$$

where the notation $\tilde{O}$ means that the logarithmic factors are dropped, and this estimate comes from using multiplication based on FFT.

This has been the best complexity for a very long time, until 2018 where G. Villard gave an algorithm in $O(n^{1.58} d)$ operations in $\mathbb{K}$, under some genericity assumptions.

Additional bibliography

A very clear introduction to the resultant can be found in

Serge Lang. *Algebra*. 3rd edition. Vol. 211. Graduate Texts in Mathematics. New York: Springer-Verlag, 2002, pp. xvi+914

Note however that for simplicity, we made the choice of focusing on matrices with entries in a field, which is not the case in Lang's book and in a large part of the literature.

Another nice book for these questions and many more is

D.A. Cox, J.B. Little, and D. O'Shea. *Ideals, varieties, and algorithms*. 4th edition. Springer New York, 2015

Lecture 6

Linear Recurrences and Differential Equations

Summary

Linear recurrences and differential equations with polynomial coefficients provide a data-structure for their solutions. In particular, they enjoy closure properties that lead to the automatic proof of identities.

In this lecture, $\mathbb{K}$ is a field of characteristic 0 (such as $\mathbb{Q}, \mathbb{C}$ or $\mathbb{Q}(t)$). We note $[x^n]A(x)$ the coefficient of x^n in the polynomial or power series $A(x)$.

6.1 Definitions and Examples

6.1.1 Linear Recurrences

Definition 6.1. *A sequence $(a_n)_{n\geq 0}$ of elements of $\mathbb{K}$ is called* polynomially finite *(or P-finite) if it satisfies a linear recurrence of the form*

$$p_d(n)a_{n+d} + \cdots + p_0(n)a_n = 0, \quad n \geq 0,$$

with $p_i \in \mathbb{K}[n]$ and $p_d \neq 0$. The integer d is called the order *of the recurrence.*

Example 6.1. The Fibonacci sequence that satisfies $F_{n+2} = F_{n+1} + F_n$ and more generally any sequence solution of a linear recurrence with constant coefficients is P-finite.

Example 6.2. The Harmonic numbers $H_n = \sum_{k=1}^{n} 1/k$ satisfy

$$H_{n+1} - H_n = \frac{1}{n+1}$$

and therefore

$$(n+1)H_{n+1} - (n+1)H_n = 1$$

and then by subtracting this from its shifted value at $n+1$:

$$(n+2)H_{n+2} - (2n+3)H_{n+1} + (n+1)H_n = 0.$$

More generally, if (a_n) is a rational sequence, $v_n = \sum_{k=1}^{n} a_k$ is P-finite, by the same derivation. More

generally, if (a_n) is P-finite, then $b_n = \sum_{k=1}^{n} a_k$ is P-finite too: it satisfies the recurrence obtained by replacing a_n by $b_n - b_{n-1}$ in Definition 6.1.

Example 6.3. The sequences $n!$, $1/n!$, $\binom{2n}{n}$, $C_n = \frac{1}{n+1}\binom{2n}{n}$ (the Catalan numbers) are P-finite, with recurrences

$$a_{n+1}-(n+1)a_n = 0, \quad (n+1)a_{n+1}-a_n = 0, \quad (n+1)a_{n+1}-2(2n+1)a_n = 0, \quad (n+2)a_{n+1}-2(2n+1)a_n = 0.$$

More generally, any quotient of products of factorials of the form $(an + b)!$ with a a positive integer is P-finite, with a recurrence of order 1.

The definition demands a homogeneous recurrence (the right-hand side of Definition 6.1 is 0), but there is no loss in generality in doing so.

Lemma 6.1. *If $(a_n)_{n\geq 0}$ in $\mathbb{K}^{\mathbb{N}}$ satisfies*

$$p_d(n)a_{n+d} + \cdots + p_0(n)a_n = q(n), \quad n \geq 0, \tag{6.1}$$

with q and p_i in $\mathbb{K}[n]$ and $p_d \neq 0$, then (a_n) is a P-finite sequence.

Proof. If $q(n) = 0$ there is nothing to prove. Otherwise, this is as in the example of Harmonic numbers: shift and subtract. Shifting gives

$$p_d(n+1)a_{n+d+1} + \cdots + p_0(n+1)a_{n+1} = q(n+1), \quad n \geq 0.$$

Multiplying both sides of this recurrence by $q(n)$, both sides of Eq. (6.1) by $q(n+1)$ and subtracting give a homogeneous linear recurrence for (a_n), whose leading coefficient $p_d(n+1)q(n+1)$ is nonzero. □

Sometimes a linear recurrence is not satisfied by the first few values of the sequence. Still, again, the sequence is P-finite.

Lemma 6.2. *If $(a_n) \in \mathbb{K}^{\mathbb{N}}$ satisfies*

$$p_d(n)a_{n+d} + \cdots + p_0(n)a_n = 0, \quad n \geq K \geq 0,$$

with p_i in $\mathbb{K}[n]$ and $p_d \neq 0$, then (a_n) is P-finite.

Proof. Multiply both sides of the recurrence by the nonzero polynomial $n(n-1)\cdots(n-K+1)$. □

The key to proving identities between sequences — that are infinite objects — from recurrences — that are finite objects — is to use the fact that the solution of a recurrence is unique, once sufficiently many initial conditions are known, and provided the leading coefficient does not vanish.

Proposition 6.1. *If (u_n) and (v_n) are two sequences in $\mathbb{K}^{\mathbb{N}}$ that both satisfy*

$$p_d(n)a_{n+d} + \cdots + p_0(n)a_n = 0, \quad n \geq 0,$$

with p_i in $\mathbb{K}[n]$ and $p_d \neq 0$, then

$$\left.\begin{array}{c} a_0 = b_0, \ldots, a_{d-1} = b_{d-1} \\ 0 \notin p_d(\mathbb{N}) \end{array}\right\} \Longrightarrow (a_n) = (b_n).$$

Proof. The proof is by induction on n. The identity is satisfied for $n < d$ and the condition that p_d does not vanish on the elements of $\mathbb{N}$ shows that a_{n+d} is determined by the previous values for all $n \in \mathbb{N}$. □

This proposition will often be used to prove that a sequence is 0, by constructing a linear recurrence it satisfies and checking that all initial conditions are 0.

In terms of computation, recognizing that a sequence is P-finite lets one compute its nth term or its first n terms in low complexity as seen in previous lectures: given the initial conditions, assuming that $0 \notin p_d(\mathbb{N})$ and that the degrees of the coefficients p_i are bounded by m, one can

— compute $u_0, \dots, u_N$ in $O(Nd\mathsf{M}(m)\log m/m)$ arithmetic operations by using multipoint evaluation from Lecture 3 to evaluate the coefficients;

— compute u_N in $O(d^\omega \mathsf{M}(N^{1/2}m^{1/2})\log(Nm))$ arithmetic operations, using the baby steps-giant steps method seen in Lecture 3;

— compute u_N in $O(d^\omega \mathsf{M}_{\mathbb{Z}}(mN\log N + N\log d)\log N)$ *bit* operations when the coefficients p_i belong to $\mathbb{Q}[n]$ by the method of binary splitting seen in Tutorial 1.

Lower complexity estimates are available in the case of linear recurrences with *constant* coefficients. They are left as an exercise.

6.1.2 Linear Differential Equations

Definition 6.2. *A power series $A \in \mathbb{K}[[x]]$ is called* differentially finite *(or D-finite) if it satisfies a linear differential equation of the form*

$$p_r(x)A^{(r)}(x) + \cdots + p_0(x)A(x) = 0, \tag{6.2}$$

with p_i in $\mathbb{K}[x]$ and $p_r \neq 0$.

Example 6.4. Many elementary functions have Taylor series that are D-finite (by extension, we say that these functions are D-finite). For instance, this is the case for

$$\exp(x), \quad \log(1+x), \quad \sin(x), \quad \arcsin(x).$$

Example 6.5. If R is a rational function with $R(0) = 1$, then for any $\alpha \in \mathbb{Q}$, R^α is D-finite, with differential equation

$$Ry' - \alpha R'y = 0.$$

Example 6.6. As a large part of physics is governed by the Laplace operator, that is linear, many of the 'special functions' of mathematical physics are D-finite. This is the case for the Bessel functions, Airy functions, Struve functions, Weber functions,....

Example 6.7. Another source of D-finite power series is provided by combinatorics. In this situation, the power series of interest are *generating functions*, which means that the coefficients are nonnegative integers that count objects of some type. The Catalan numbers above are a special case: they count binary trees with n nodes. Other examples come from lattice path enumeration.

Here is the analogue of Lemma 6.1.

Lemma 6.3. *If the power series $A \in \mathbb{K}[[x]]$ satisfies a linear differential equation of the form*

$$p_r(x)A^{(r)}(x) + \cdots + p_0(x)A(x) = q(x)$$

with $q \in \mathbb{K}(x)$ and $p_r \neq 0$, then A is differentially finite.

Proof. Differentiate both sides of the equation, multiply by q and subtract the equation multiplied by q'. Finally, multiply by a common denominator of all coefficients. □

Useful characterization

The field of fractions of the integral domain $\mathbb{K}[[x]]$ is the field of *Laurent series*.

Definition 6.3. *A* Laurent series *is a formal series of the form*

$$\sum_{n \geq n_0} a_n x^n, \quad n_0 \in \mathbb{Z}.$$

The field of Laurent series with coefficients in $\mathbb{K}$ is denoted $\mathbb{K}((x))$.

Compared to formal power series the difference is that the valuation can be a negative integer.

The following gives a way to prove that a power series is D-finite.

Proposition 6.2. *A formal power series $A \in \mathbb{K}[[x]]$ is D-finite if and only if $A, A', A'', \ldots$ generate a finite-dimensional vector space over $\mathbb{K}(x)$ in $\mathbb{K}((x))$.*

Proof. If A is D-finite, it satisfies a linear differential equation like Eq. (6.2). This implies that $A^{(r)}$ is a linear combination of $A, \ldots, A^{(r-1)}$ with rational function coefficients $-p_i/p_r$. By successive differentiation, it follows by induction that for any $k \geq r$, $A^{(k)}$ belongs to the vector space generated by $A, \ldots, A^{(r-1)}$.

Conversely, if $A, A', A'', \ldots$ generate a vector space of dimension r, then the $r+1$ power series $A, A', \ldots, A^{(r)}$ are linearly dependent over $\mathbb{K}(x)$ and therefore satisfy a linear relation of the form

$$q_r(x)A^{(r)} + \cdots + q_0 A = 0,$$

with q_i rational functions in $\mathbb{K}(x)$, not all 0. Multiplying by a common denominator gives a linear differential equation of the type (6.2). Moreover, the leading coefficient q_r is not 0: if it was, there would exist a differential equation of order smaller than r, and by the previous part of the proof, the dimension of the vector space generated by the derivatives of A would also be smaller than r. □

While this result is useful in proofs, the actual computation of linear differential equations relies on linear algebra in the underlying finite-dimensional vector space.

Notation 6.1. *In this lecture, we denote by V_A the vector space generated by $A, A', \ldots$ and by d_A its dimension:*

$$V_A = \mathrm{Vect}_{\mathbb{K}(x)}(A, A', A'', \ldots), \qquad d_A = \dim V_A.$$

Equivalence

A source both of linear differential equations and of linear recurrences is the following.

Theorem 6.1. *The formal power series $A(x) = \sum_{n \geq 0} a_n x^n$ is differentially finite if and only if the sequence (a_n) is polynomially finite.*

Proof. In both directions, the proof is an algorithm.

Recall the notation $[x^n]A(x)$ for the coefficient of x^n in the power series $A(x)$. By convention, $[x^n]A(x) = 0$ when $n < 0$. By induction on i and j, it follows that

$$[x^n]x^j A^{(i)}(x) = (n-j+1)\cdots(n-j+i)[x^{n+i-j}]A(x).$$

Equation (6.2) can be written in the form

$$\sum_{i,j} c_{i,j} x^j A^{(i)}(x) = 0,$$

where the sum is finite. Extracting the coefficient of x^n in both sides gives

$$\sum_{i,j} c_{i,j}(n-j+1)\cdots(n-j+i)a_{n+i-j} = 0, \quad n \in \mathbb{Z}. \tag{6.3}$$

When n is such that $n+i-j$ is negative, this equation becomes $0 = 0$. For n smaller than $\min_{i,j}(j-i)$, it gives a linear relation between the first coefficients of A. For n larger that this value, it gives a recurrence. Lemma 6.2 then shows that (a_n) is polynomially finite.

Conversely, if (a_n) is the sequence of coefficients of a formal power series $A(x)$, then multiplying the recurrence (6.1) by x^n and summing over n with the help of the relations[1]

$$\sum_{n\geq 0} n^i a_n x^n = \left(x\frac{d}{dx}\right)^i A(x),$$
$$\sum_{n\geq 0} a_{n+j} x^n = \frac{A(x) - a_0 - a_1 x - \cdots - a_{j-1}x^{j-1}}{x^j}$$

leads to a linear differential equation with polynomial coefficients and a rational right-hand side. The conclusion follows from Lemma 6.3. □

Example 6.8. Since $n!$ is a polynomially finite sequence, $\sum_{n\geq 0} n! x^n$ is a differentially finite power series. Convergence is not a relevant issue in these matters.

Example 6.9. By Example 6.5, $(1+x)^\alpha$ is differentially finite. Translating the differential equation

$$(1+x)y' - \alpha y = 0$$

into a recurrence gives

$$(n+1)c_{n+1} + (n-\alpha)c_n = 0.$$

This is a first-order linear recurrence satisfied by

$$c_n = \binom{\alpha}{n} = \frac{\alpha(\alpha-1)\cdots(\alpha-n+1)}{n!},$$

as can be checked by substitution in the recurrence.

Example 6.10. The same idea lets one compute high-order coefficients of the polynomial

$$(1+x)^{2N}(1+x+x^2)^N.$$

[1] The notation $(xd/dx)^i$ means that the operator xd/dx is iterated i times. For instance $(xd/dx)^2 A(x) = x(xA')'$.

Indeed, this polynomial satisfies the linear differential equation

$$\frac{y'}{y} = \frac{2N}{1+x} + N\frac{1+2x}{1+x+x^2}.$$

Reducing to the same denominator gives

$$(1+x+x^2)(1+x)y' - (2N(1+x+x^2) + N(1+2x)(1+x))y = 0.$$

Extracting the coefficient of x^n gives the recurrence

$$(n+3)u_{n+3} - (3N-2n-4)u_{n+2} - (5N-2n-2)u_{n+1} - (4N-n)u_n = 0,$$

with which it is easy to compute the first k coefficients (or just the kth one) for $k = N$ for instance, more efficiently than by expanding the polynomial.

The algorithms following from the proof of the proposition are implemented in the Maple package `gfun` under the names `diffeqtorec` and `rectodiffeq`. They also keep track of the initial conditions if they are given in their input. This simplifies the task of dealing with a computation such as the ones above.

Example 6.11.

```
> f:=(1+x)^alpha;
```

$$f := (1+x)^\alpha$$

```
> diff(y(x),x)=normal(diff(f,x)/f)*y(x);
```

$$\frac{d}{dx}y(x) = \frac{\alpha y(x)}{1+x}$$

```
> gfun:-diffeqtorec({%,y(0)=1},y(x),u(n));
```

$$\{(-\alpha+n)u(n) + (n+1)u(n+1), u(0) = 1\}$$

```
> P:=(1+x)^(2*N)*(1+x+x^2)^N;
```

$$P := (1+x)^{2N}(1+x+x^2)^N$$

```
> rec:=gfun:-diffeqtorec({diff(y(x),x)=normal(diff(P,x)/P)*y(x),y(0)=1},y(x),u(n));
```

$$\{(-4N+n)\,u(n) + (-5N+2n+2)\,u(n+1) + (-3N+2n+4)\,u(n+2) + (n+3)\,u(n+3)\,,$$
$$u(0) = 1, u(1) = 3N, u(2) = \frac{9}{2}N^2 - \frac{1}{2}N\Big\}$$

6.2 Closure Properties

6.2.1 Sum and Product of Differentially Finite Power Series

Theorem 6.2. *If A, B in $\mathbb{K}[[x]]$ are differentially finite, and $\alpha \in \mathbb{K}$, then $\alpha A, A+B$ and AB are differentially finite.*

Proof. 1. Since the differential equations considered here are linear, if A is a solution of such an equation, so is αA.

2. Let V_A and V_B be defined as the vector spaces generated by the derivatives of A and B. Then by induction, for any $k \in \mathbb{N}$,

$$(A+B)^{(k)} = A^{(k)} + B^{(k)} \in V_A + V_B$$

and the dimension of $V_A + V_B$ is at most $d_A + d_B$, showing that all the derivatives of $A+B$ belong to a finite-dimensional vector space and thus that $A+B$ is differentially finite.

3. By Leibniz's rule,

$$\begin{aligned}(AB)^{(k)} &= \sum_{i=0}^{k} \binom{k}{i} A^{(i)} B^{(k-i)} \\ &\in \mathrm{Vect}_{\mathbb{K}(x)}(A^{(i)}B^{(j)} \mid i < d_A, j < d_B)\end{aligned}$$

and that vector space has dimension bounded by $d_A d_B$. □

The algorithm underlying these proofs consists in constructing a matrix whose entries are the coordinates of the successive derivatives of the power series of interest in the generators ($(A^{(i)}, B^{(i)})$ for the sum, $(A^{(i)}B^{(j)})$ for the product) and then looking for the kernel of that matrix when it has more columns than rows. In Maple, this is implemented the the function `poltodiffeq` of the package `gfun`.

Identities are proved using this theorem by constructing a linear differential equation and checking initial conditions.

Example 6.12. Simple identities concerning linear differential equations with constant coefficients can be obtained without even constructing the equations. For instance, here is a simple proof that

$$\sin^2 x + \cos^2 x = 1.$$

The starting point is $y'' + y = 0$ that is satisfied by both sin and cos. Next, if y is a solution of this equation then y^2, yy', y'^2 generate the vector space of derivatives of y^2: all of them have a derivative that is a linear combination of the other ones. Thus $\sin^2$ and $\cos^2$ satisfy the same linear differential equation, that has order at most 3. Since it is linear, their sum $\sin^2 + \cos^2$ is also solution to this equation. Adding -1 gives a power series whose derivatives generate a vector space of dimension at most $3 + 1 = 4$. It follows that $\sin^2 + \cos^2 - 1$ satisfies a linear differential equation of order at most 4. Moreover, all the computations involve only constant coefficients, so that this equation has constant coefficients. In particular the leading coefficient does not vanish at 0 and the Cauchy-Lipschitz theorem (also called Picard-Lindelöf theorem) applies and shows uniqueness of the solution given the first 4 initial conditions. Now, a direct computation shows

$$\sin^2 x + \cos^2 x - 1 = O(x^4),$$

showing that these initial conditions are 0 and concluding the proof of the identity.

Example 6.13. We derive the identity

$$(\arcsin(x))^2 = \sum_{n \geq 0} \frac{n!}{\left(\frac{1}{2}\right) \cdots \left(n + \frac{1}{2}\right)} \frac{x^{2n+2}}{2n+2} \tag{6.4}$$

with intermediate computations in Maple. Let $f = \arcsin x$. From $f' = 1/\sqrt{1-x^2}$ it follows as in Example 6.5 that f is differentially finite:

```
> f:=arcsin(x):
> diff(f,x);
```

$$\frac{1}{\sqrt{1-x^2}}$$

```
> deq:=diff(y(x),x,x)=normal(diff(%,x)/%)*diff(y(x),x);
```

$$\frac{d^2}{dx^2}y(x) = -\frac{x}{x^2-1}\frac{d}{dx}y(x)$$

This equation will be used to rewrite all instances of y'' in terms of y'.

By direct computation using this equation, derivatives of linear combinations $a(x)y^2+b(x)yy'+c(x)y'^2$ are of the same form:

```
> diff(y(x)^2,x);
```

$$2y(x)\left(\frac{d}{dx}(x)\right)$$

```
> subs(deq,diff(y(x)*diff(y(x),x),x));
```

$$\left(\frac{d}{dx}(x)\right)^2 - \frac{xy(x)\frac{d}{dx}(x)}{x^2-1}$$

```
> subs(deq,diff(diff(y(x),x)^2,x));
```

$$-2\frac{x\left(\frac{d}{dx}(x)\right)^2}{x^2-1}$$

This gives a matrix

```
> M:=Matrix([[0,0,0],[2,x/(1-x^2),0],[0,1,2*x/(1-x^2)]]);
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 2 & \frac{x}{-x^2+1} & 0 \\ 0 & 1 & \frac{2x}{-x^2+1} \end{bmatrix}$$

which is such that the derivative of a vector with coordinates $a(x), b(x), c(x)$ in the generators $y(x)^2, y(x)y'(x), y'(x)^2$ is obtained by differentiation and multiplication by M. Thus starting with $y(x)^2$, we get the successive derivatives in this basis:

```
> H[0]:=Vector([1,0,0]):
> for i to 3 do H[i]:=diff(H[i-1],x)+M.H[i-1] od;
```

$$H_1 := \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}$$

$$H_2 := \begin{bmatrix} 0 \\ \frac{2x}{-x^2+1} \\ 2 \end{bmatrix}$$

$$H_3 := \begin{bmatrix} 0 \\ \frac{2}{-x^2+1} + \frac{6x^2}{(-x^2+1)^2} \\ \frac{6x}{-x^2+1} \end{bmatrix}$$

This produces 4 vectors in dimension 3. A linear dependency is obtained by looking for the kernel of the matrix

```
> Matrix([seq(H[i],i=0..3)]);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & \frac{2x}{-x^2+1} & \frac{2}{-x^2+1} + \frac{6x^2}{(-x^2+1)^2} \\ 0 & 0 & 2 & \frac{6x}{-x^2+1} \end{bmatrix}$$

The differential equation is given by the coordinates of a vector in its kernel:

```
> op(LinearAlgebra[NullSpace](%));
```

$$\begin{bmatrix} 0 \\ \frac{1}{x^2-1} \\ \frac{3x}{x^2-1} \\ 1 \end{bmatrix}$$

```
> add(%[i]*diff(y(x),[x$(i-1)]),i=1..4);
```

$$\frac{\frac{d}{dx}y(x)}{x^2-1} + \frac{3x\left(\frac{d^2}{dx^2}y(x)\right)}{x^2-1} + \frac{d^3}{dx^3}y(x)$$

This differential equation satisfied by $\arcsin(x)^2$ can now be translated into a linear recurrence for its Taylor coefficients:

```
> gfun:-diffeqtorec(%,y(x),c(n));
```

$$\left(n^2+2n+1\right)c(n+1) + \left(-n^2-5n-6\right)c(n+3)$$

```
> collect(subs(n=n-1,%),c,factor);
```

$$n^2c(n) - (n+2)(n+1)c(n+2)$$

From this linear recurrence and the initial conditions $c_0 = 1, c_1 = 0$, the formula (6.4) follows.

Corollary 6.1. *If A, B in $\mathbb{K}[[x]]$ are differentially finite, the truncated product $AB + O(x^N)$ can be computed in $O(N)$ operations.*

This is to be contrasted with $O(\mathsf{M}(N))$ for arbitrary power series.

Proof. From the linear differential equations satisfied by A, B, one computes a linear differential equation for the product AB and from there a linear recurrence for its coefficients. Unrolling this recurrence leads to the $O(N)$ complexity. □

6.2.2 Sum and Product of Polynomially Recursive Sequences

The analogue of the previous theorem is the following.

Theorem 6.3. *If (a_n) and (b_n) are polynomially finite sequences in $\mathbb{K}^{\mathbb{N}}$ and $\alpha \in \mathbb{K}$ then $(\alpha a_n), (a_n + b_n)$ and $(a_n b_n)$ are polynomially finite.*

In Maple, this is implemented in the function `poltorec` of the package `gfun`.

Proof. Let (a_n) and (b_n) satisfy the linear recurrences

$$\begin{aligned} p_d(n)a_{n+d} + \cdots + p_0(n)a_n &= 0, \quad n \geq 0, \\ q_\delta(n)b_{n+\delta} + \cdots + q_0(n)b_n &= 0, \quad n \geq 0. \end{aligned}$$

Since p_d and q_δ are nonzero polynomials, there exists $K \in \mathbb{N}$ such that $p_d(n)q_\delta(n) \neq 0$ for $n \geq K$. Then for $k \in \mathbb{N}$,

$$a_{n+k} = r_{d-1}^{[k]}(n)a_{n+d-1} + \cdots + r_0^{[k]}(n)a_n, \quad n \geq K$$
$$b_{n+k} = s_{\delta-1}^{[k]}(n)b_{n+\delta-1} + \cdots + s_0^{[k]}(n)b_n, \quad n \geq K,$$

with $r_i^{[k]}$ and $s_i^{[k]}$ rational functions whose denominators do not vanish in $\mathbb{N}$.

It follows that for $n \geq K$, the shifted sequences $(a_{n+k} + b_{n+k})$ can be rewritten as linear combinations of the shifted sequences (a_{n+i}) and (b_{n+j}) for $i < d$, $j < \delta$ and similarly for the products $(a_{n+k}b_{n+k})$ in terms of $(a_{n+i}b_{n+j})$ for $i < d, j < \delta$. By Lemma 6.2, these sequences are polynomially finite. □

Example 6.14. Cassini's identity on the Fibonacci numbers

$$F_{n+2}F_n - F_{n+1}^2 = (-1)^n, \qquad n \geq 0$$

is proved exactly in the same way as $\sin^2 + \cos^2 = 1$ above, by counting dimensions. Since the recurrences have constant coefficients, there is no need to discuss the largest integer root of the leading coefficient.

As a consequence of the theorem above, another closure property holds for differentially finite power series.

Definition 6.4. *If $A = \sum a_n x^n$ and $B = \sum b_n x^n$ are two power series, then their* Hadamard product *is*

$$A \odot B = \sum_{n \geq 0} a_n b_n x^n.$$

Corollary 6.2. *If A and B are two differentially finite power series, then so is their Hadamard product $A \odot B$.*

Proof. Since A and B are differentially finite, their sequences of coefficients (a_n) and (b_n) are polynomially finite. By the previous theorem, so is their product $(a_n b_n)$ and thus the power series $\sum a_n b_n x^n$ is differentially finite. □

Example 6.15. Mehler's identity for the Hermite polynomials,

$$\sum_{n=0}^{\infty} H_n(x)H_n(y)\frac{u^n}{n!} = \frac{\exp\left(\frac{4u(xy-u(x^2+y^2))}{1-4u^2}\right)}{\sqrt{1-4u^2}},$$

was proved in Lecture 1 using the linear recurrence for the Hermite polynomials and closure properties of sequence. Alternatively, one can start from the generating function

$$H_x(z) := \sum_{n \geq 0} H_n(x)\frac{z^n}{n!} = \exp(z(2x - z))$$

and use closure by Hadamard product, writing the left-hand side of Mehler's identity as

$$H_x(z) \odot H_y(z) \odot \sum_{n \geq 0} n! z^n.$$

The resulting computation is as follows:

```
> f1:=exp(z*(2*x-z)):
> deq1:=gfun:-holexprtodiffeq(f1,f(z));
```

$$deq1 := \{(-2x+2z)\,f(z) + \frac{d}{dz}f(z)\,, f(0)=1\}$$

```
> deq2:=subs(x=y,deq1):
> deq3:=gfun:-hadamardproduct(deq1,deq2,f(z));
```

$$deq3 := \left\{(-16xy+16z)\,f(z) + \left(8x^2+8y^2-12\right)\left(\frac{d}{dz}f(z)\right) + (-4xy-8z)\left(\frac{d^2}{dz^2}f(z)\right) \right.$$
$$\left. +3\frac{d^3}{dz^3}f(z) + z\left(\frac{d^4}{dz^4}f(z)\right), f(0)=1, D(f)(0)=4xy, D^{(2)}(f)(0)=8x^2y^2-4x^2-4y^2+2\right\}$$

```
> deq4:=gfun:-hadamardproduct(deq3,
  gfun:-rectodiffeq({u(n+1)-(n+1)*u(n),u(0)=1},u(n),f(z)),f(z));
```

$$\{(-16xy\,z^2+8x^2z+8y^2z+16z^3-4xy-4z)\,f(z) + (16z^4-8z^2+1)\left(\frac{d}{dz}f(z)\right), f(0)=1\}$$

This is a linear differential equation of order 1, whose solution reduces to integrating a rational function, whence the result

```
> dsolve(deq4,f(z)) assuming z>0,z<1/2;
```

$$f(z) = \frac{\mathrm{e}^{\frac{-4xyz+x^2+y^2}{(2z+1)(2z-1)}}\sqrt{\frac{1}{(2z+1)(-2z+1)}}}{\mathrm{e}^{-x^2-y^2}}$$

6.3 Algebraic Power Series

Besides rational power series (whose coefficients satisfy linear recurrences with *constant* coefficients, see Lecture 3) and their rational powers (Example 6.5), a large class of differentially finite power series is provided by algebraic power series.

Definition 6.5. *A power series $A \in \mathbb{K}[[x]]$ is* algebraic *if there is a nonzero polynomial $P \in \mathbb{K}[x,y]$ such that $P(x, A(x)) = 0$.*

Example 6.16. Many generating functions coming from combinatorics are algebraic. This is the case for the generating series of the Catalan numbers

$$\frac{1-\sqrt{1-4x}}{2x} = \sum_{n\geq 0}\frac{1}{n+1}\binom{2n}{n},$$

that count binary trees with n nodes; the Schröder numbers, with generating series

$$\frac{1-x-\sqrt{1-6x+x^2}}{2x} = 1+2x+6x^2+22x^3+90x^4+\cdots$$

and many other ones. A large class is provided by the number of words of length n in an unambiguous context-free language.

The main result is that such series are differentially finite, with an explicit bound on the order of the differential equation.

Theorem 6.4 (Abel 1827; Cockle 1860)**.** *If $A \in \mathbb{K}[[x]]$ satisfies a polynomial of degree $\leq d$, then it satisfies a linear differential equation of order $\leq d$.*

Proof. Let $P(x, y)$ be a nonzero polynomial of minimal degree in y such that $P(x, A) = 0$. By Bézout's theorem (see Lecture 4), there exist two polynomials U, V in $\mathbb{K}(x)[y]$ such that

$$UP + V\frac{\partial P}{\partial y} = 1$$

(P having minimal degree and $\mathbb{K}$ having characteristic 0, P is relatively prime with its derivative with respect to y.) Differentiating the equation $P(x, A) = 0$ with respect to x gives

$$\frac{\partial P}{\partial y}(x, A)A' + \frac{\partial P}{\partial x}(x, A) = 0.$$

Multiplying by $V(x, A)$ gives

$$A' = -V(x, A)\frac{\partial P}{\partial x}(x, A).$$

Taking the remainder of the Euclidean division of the right-hand side by $P(x, A)$ shows that

$$A' \in \mathrm{Vect}_{\mathbb{K}(x)}(1, A, \ldots, A^{d-1}).$$

By induction, it follows that for all $k \geq 0$,

$$A^{(k)} \in \mathrm{Vect}_{\mathbb{K}(x)}(1, A, \ldots, A^{d-1}).$$

This shows that A is differentially finite. □

Corollary 6.3. *If A is an algebraic power series, then the truncated power series $A(x) + O(x^N)$ can be computed in $O(N)$ arithmetic operations in $\mathbb{K}$.*

This complexity is to be contrasted with the complexity $O(\mathsf{M}(N))$ that would be obtained by Newton's iteration.

Proposition 6.3. *If $F \in \mathbb{K}[[x]]$ is differentially finite and $A \in x\mathbb{K}[[x]]$ is algebraic, then the composition $F(A)$ is differentially finite.*

Proof. From the previous proof, it follows that

$$A^{(k)} \in \mathrm{Vect}_{\mathbb{K}(x)}(1, A, \ldots, A^{d-1})$$

for all $k \in \mathbb{N}$, where d is the degree of a nonzero polynomial cancelling A. By the chain rule, it follows by induction that

$$\frac{d^k}{dx^k}(F(A(x)) \in \mathrm{Vect}_{\mathbb{K}(x)}(F^{(i)}(A)A^j \mid 0 \leq i < d_F, 0 \leq j < d),$$

which concludes the proof. □

Example 6.17. It follows without any computation that many power series are differentially finite, like

$$\sin\left(\frac{x}{1-x}\right),\ \exp\left(\frac{1-\sqrt{1-4x}}{2}\right),\ \operatorname{arctanh}\left(x\sqrt[3]{\frac{1+x}{1-x}}\right),\ldots.$$

6.4 Solutions of Linear Recurrences

The closure properties presented before allow to compute with solutions of linear recurrences using the equations as a data-structure. Still, there are cases when it is useful to detect solutions of a special type as a sub-step in other algorithms. We now present algorithms that decide the existence of solutions of given types: either they find one or they prove that none exists.

6.4.1 Space of Solutions

Again, we consider the recurrence

$$p_d(n)a_{n+d} + \cdots + p_0(n)a_n = 0, \quad n \geq 0, \tag{6.5}$$

with polynomial coefficients $p_i \in \mathbb{K}[n]$ and $p_d \neq 0$. We first consider the set of all sequences that are solutions of this recurrence. This is slightly different from the more familiar setting of linear recurrences with constant coefficients, as the dimension of the space of solutions can be larger than the order of the recurrence.

Proposition 6.4. *The set V of solutions of Eq. (6.5) in $\mathbb{K}^{\mathbb{N}}$ is a $\mathbb{K}$-vector space and its dimension satisfies*

$$d \leq \dim V \leq d + \operatorname{card} H,$$

where $H = \{k \in \mathbb{N} \mid p_d(k) = 0\}$.

In the rest of this section, we make the assumption that this set H can be computed (this is not a difficulty for the fields $\mathbb{K}$ that we encounter in this course).

Proof. When $n \notin H$, a_{n+d} is determined by the recurrence and thus for n larger than $K := \max(\{0, \ldots, d-1\} \cup H)$, the sequence is determined by its previous values. The values $a_0, \ldots, a_{K+d-1}$ are given by a linear system of K equations in $K + d$ unknowns. The rank of the corresponding matrix is at most K and it is at least $K - \operatorname{card} H$ as it has a band shape with nonzero entries at the right-most end of the rows corresponding to indices not in H. □

6.4.2 Solutions with finite support

The support of a sequence $(u_n) \in \mathbb{K}^{\mathbb{N}}$ is the set of indices where $u_n \neq 0$. A sequence has finite support when only finitely many of its values are nonzero. For instance, the recurrence

$$2(n-2)(n-4)u_{n+2} - (7n-24)u_{n+1} + 6(n-3)u_n = 0$$

has for solution $(u_0, u_1, u_2, u_3, \ldots) = (572, 309, 180, 108, 0, \ldots)$ where all $u_k = 0$ for $k \geq 4$.

If a solution of Eq. (6.5) has finite support with maximal element k, then $u_k \neq 0$ while $u_{k+1} = u_{k+2} = \cdots = 0$ and by considering Eq. (6.5) at $n = k$, it follows that necessarily, $p_0(k) = 0$. Conversely, if k is a nonnegative integer such that $p_0(k) = 0$, then one can consider a sequence defined by $u_n = 0$ for $n > k$, $u_k = 1$ and unrolling the recurrence to compute $u_{k-1}, u_{k-2}, \ldots, u_0$. As in the situation of Section 6.4.1, this leads to considering a rectangular matrix whose kernel has dimension bounded by the number of nonnegative integers k such that $p_0(k) = 0$. Thus we have.

Proposition 6.5. *The recurrence Eq. (6.5) admits solutions with finite support if and only if the set* $J := \{k \in \mathbb{N} \mid p_0(k) = 0\}$ *is nonempty. In that case, the vector space of solutions with finite support has dimension between 1 and* $\operatorname{card} J$.

6.4.3 Polynomial solutions

A sequence (u_n) is called *polynomial* when there exists a polynomial $p \in \mathbb{K}[n]$ such that $u_n = p(n)$ for all $n \geq 0$. For instance, the recurrence

$$(n+1)(2n-41)u_n - (4n^2 - 72n + 104)u_{n+1} + (2n+5)(n-19)u_{n+2} = 0$$

has the polynomial solution

$$\begin{aligned} u_n =& 1024n^{10} - 102400n^9 + 4363008n^8 - 103280640n^7 + 1484617344n^6 - 13296944640n^5 \\ &+ 73171198112n^4 - 234070964480n^3 + 384903129612n^2 - 240900096240n + 7202019825. \end{aligned}$$

A simple way of finding such a solution is to start by finding an upper bound on its possible degree and then using an undeterminate coefficients method, which leads to solving a linear system in the coefficients of the unknown polynomial.

In order to bound the possible degrees of polynomial solutions, one can use the fact that the *difference operator*

$$\Delta : (u_n) \mapsto (u_{n+1} - u_n)$$

has the property that it maps polynomials of degree d to polynomials of degree $d-1$. Moreover, it is easily seen by induction that

$$(u_{n+k}) = \sum_{i=0}^{k} \Delta^i(u_n).$$

This allows to rewrite the recurrence Eq. (6.5) in the form

$$q_d(n)\Delta^d(a_n) + \cdots + q_0(n)(a_n) = 0. \tag{6.6}$$

If a_n were a polynomial of degree δ, then the terms of this equations have degrees $\delta - d + \deg q_d, \delta - (d-1) + \deg q_{d-1}, \ldots, \delta + \deg q_0$. Let then $b = \max_k(\deg q_k - k)$ and $E = \{k \mid \deg q_k - k = b\}$ be the set of indices where this maximal value is reached. The coefficient of degree $\delta + b$ in Eq. (6.6) is, up to a nonzero constant factor,

$$I(\delta) = \sum_{k \in E} \operatorname{lc}(q_k)\delta(\delta-1)\cdots(\delta-k+1),$$

where lc denotes the *leading coefficient* of a polynomial. As a consequence, a solution of degree δ can only exist if $I(\delta) = 0$. If the polynomial $I(\delta)$ does not have nonnegative integer roots, then there does not exist any polynomial solution to Eq. (6.5) and otherwise, the largest nonnegative integer roots of I thus gives the desired bound on the possible degrees of polynomial solutions.

In the example above, the recurrence rewrites

$$(2n+5)(n-19)\Delta^2 u_n + (6n - 294)\Delta u_n - 240u_n = 0.$$

From there, it follows that $b = 0$, $E = \{0, 1, 2\}$ and

$$I(\delta) = 2\delta(\delta-1) + 6\delta - 240 = 2(\delta+12)(\delta-10).$$

It follows that the degree of polynomial solutions is bounded by 10 and then substituting $u_n = c_0 + c_1 n + \cdots + c_{10}n^{10}$ into the recurrence and extracting coefficients of n gives a linear system in the c_i having the coefficients as a solution.

6.4.4 Rational solutions

There exists a more recent and less straightforward algorithm for the problem of finding whether the recurrence admits a solution $u_n = r(n)$ with $r \in \mathbb{K}(n)$ a rational function. For instance,

$$\begin{aligned}(n+2)\,(n-8)\,(n-3)\left(n^2-14n+50\right)u_{n+2}\\ -(n+1)\,(n-9)\,(n-4)\left(3n^2-39n+130\right)u_{n+1}\\ 2n\,(n-10)\,(n-5)\left(n^2-12n+37\right)u_n = 0\end{aligned}$$

has for solution $u_n = 1/(n(n-10))$ and the question is to detect it.

The strategy of the algorithm, due to Abramov (in 1995), is to find a multiple $Q(n)$ of all possible denominators of solutions of the recurrence. Given such a denominator, injecting $a_n = b_n/Q(n)$ into the recurrence and normalizing, one is left with a linear recurrence in b_n of which a *polynomial* solution sought. Such a solution exists if and only if the original equation had a rational solution.

The key to finding multiples of the denominator is the following.

Lemma 6.4. *If $a_n = P(n)/Q(n)$ is a solution of Eq. (6.5) with P, Q two polynomials in $\mathbb{K}[n]$ and $\gcd(P,Q) = 1$, then*

$$Q(n) \mid \gcd\left(p_0(n)\cdots p_0(n+H), p_d(n-d)\cdots p_d(n-d-H)\right),$$

where $H := \max\{k \in \mathbb{N} \mid \gcd(p_0(n+k), p_d(n-d)) \neq 1\}$.

Proof. Assume that a solution $a_n = P(n)/Q(n)$ as in the lemma exists and let $M(n) := \operatorname{lcm}(Q(n+1), \ldots, Q(n+d))$. Substituting $a_n = P(n)/Q(n)$ and taking the numerator gives

$$M(n)\left(p_d(n)\frac{P(n+d)}{Q(n+d)} + \cdots + p_1(n)\frac{P(n+1)}{Q(n+1)}\right) + p_0(n)\frac{P(n)M(n)}{Q(n)} = 0.$$

The first term is a polynomial while $\gcd(P,Q) = 1$. This implies that

$$Q(n) \mid p_0(n)M(n) = p_0(n)\operatorname{lcm}(Q(n+1), \ldots, Q(n+d)).$$

From this divisibility, it follows that $Q(n+1) \mid p_0(n+1)M(n+1)$ and therefore

$$Q(n) \mid p_0(n)\operatorname{lcm}(p_0(n+1)\operatorname{lcm}(Q(n+2), \ldots, Q(n+d+1)), Q(n+2), \ldots, Q(n+d)).$$

It follows that

$$Q(n) \mid p_0(n)p_0(n+1)\operatorname{lcm}(Q(n+2), \ldots, Q(n+d+1))$$

and by induction for all $j \geq 1$,

$$Q(n) \mid p_0(n)\cdots p_0(n+j)\operatorname{lcm}(Q(n+j+1), \ldots, Q(n+j+d)).$$

Similarly, on gets

$$Q(n) \mid p_d(n-d)\cdots p_d(n-d-j)\operatorname{lcm}(Q(n-d-j-1), \ldots, Q(n-d-j-d)).$$

As Q is a polynomial, there exists K such that $\gcd(Q(n), Q(n+j)) = 1$ for $j \geq K$. Thus, letting j increase, taking the gcd and using the definition of H gives the conclusion of the lemma. □

Abramov's algorithm for rational solutions follows from this idea:

```
# Input:
#         the extremal coefficients p_0, p_d of the recurrence
#         its order;
#         the variable
# Output:
#         a multiple of the denominators of rational solutions
abramov:=proc(p0,pd,d,n)
local R,X,H,Q,A,B,i,h,g;
        R:=resultant(subs(n=n+X,p0),subs(n=n-d,pd),n);
        H:=roots(R,X);
        H:=sort(select(type,map2(op,1,H),nonnegint));
        if H=[] then return 1 fi;
        Q:=1; A:=p0; B:=subs(n=n-d,pd);
        for i to nops(H) do
                h:=H[-i];
                g:=gcd(subs(n=n+h,A),B);
                Q:=Q*mul(subs(n=n-i,g),i=0..h);
                A:=normal(A/subs(n=n-h,g));
                B:=normal(B/g)
        end do;
        return Q
end:
```

The proof of its correctness that follows is due to Chen et al. (2008). It relies on two lemmas.

Lemma 6.5. *The maximal value of the list H computed by the algorithm is is H from the previous lemma.*

Proof. The resultant of $p_0(n+X)$ and $p_d(n-d)$ with respect to n vanishes when those two polynomials have a root in common. It follows that the maximal integer root of this resultant is the maximal $k \in \mathbb{N}$ such that $\gcd(p_0(n+k), p_d(n-d)) \neq 1$. □

Lemma 6.6. *The output of the algorithm is*

$$\gcd(p_0(n)\cdots p_0(n+H), p_d(n-d)\cdots p_d(n-d-H)).$$

Proof. Write A_i, B_i, Q_i, g_i for the values of A, B, Q, g at the entry of the loop and h_i for the value of h during the loop. By induction, we prove simultaneously two properties: $\gcd(A_i(n+k), B_i(n)) = 1$ for $k > h_i$ and

$$Q_i(n)\gcd(A_i(n)\cdots A_i(n+h_i), B_i(n)\cdots B_i(n-h_i))$$

is invariant. Initially, by the previous lemma, for $k > H$, $\gcd(A(n+k), B_(n)) = 1$ and the gcd is what we want to compute.

Next, we have

$$\begin{aligned}
&\gcd(A_i(n)\cdots A_i(n+h_i), B_i(n)\cdots B_i(n-h_i))\\
&\quad= \gcd\Big(\underbrace{\frac{A_i(n)}{g_i(n-h_i)}}_{A_{i+1}(n)}\cdots\frac{A_i(n+h_i)}{g_i(n)}, \underbrace{\frac{B_i(n)}{g_i(n)}}_{B_{i+1}(n)}\cdots\frac{B_i(n-h_i)}{g_i(n-h_i)}\Big)g_i(n)\cdots g_i(n-h_i)\\
&\quad= \gcd(A_{i+1}(n)\cdots A_{i+1}(n+h_{i+1}), B_{i+1}(n)\cdots B_{i+1}(n-h_{i+1}))Q_{i+1}(n)/Q_i(n).
\end{aligned}$$

The property on $\gcd(A_{i+1}(n+k), B_{i+1}(n))$ comes from the fact that all common factors of $A_i(n+k)$ and $B_i(n)$ with $k \geq h_i$ have been removed by the loop. □

6.5 Solutions of Linear Differential Equations

We now start from a linear differential equation

$$q_r(x)y^{(r)}(x) + \cdots + q_0(x)y(x) = 0, \tag{6.7}$$

with $q_i(x) \in \mathbb{K}[x]$ and $q_r \neq 0$ and we study the existence and computation of solutions of various types. It turns out that this study also gives simple criteria to detect that a power series such as $\tan x$ is *not* a solution of a linear differential equation.

6.5.1 Polynomial solutions

A simple algorithm to recover polynomial solutions of linear differential equations comes from the observation that their sequences of coefficients have finite support. The algorithm then consists in translating the differential equation into a recurrence, finding its solutions with finite support and returning the polynomials with those sequences as coefficients. Conversely, if there are not solutions with finite support to the recurrence, there are no polynomial solutions to the differential equation.

6.5.2 Power series solutions

If the differential equation Eq. (6.7) has a series solution $\sum_{n\geq 0} y_n x^n$, one can compute a linear recurrence that the sequence (y_n) satisfies. Assume that this recurrence has the form Eq. (6.5) (note that the translation may also give linear constraints on $y_0, \ldots, y_{d-1}$). An important question when unrolling the recurrence is whether its leading term vanishes at nonnegative integers, as these correspond to indices where the sequence cannot necessarily be extended.

Definition 6.6. *The* indicial polynomial *of Eq. (6.7) at 0 is the polynomial* $\mathrm{Ind}_0(n) := p_d(n-d)$.

An important observation is that this polynomial is known when $q_r(0) \neq 0$.

Lemma 6.7. *If $q_r(0) \neq 0$, the indicial polynomial at 0 is*

$$\mathrm{Ind}_0(n) = q_r(0)n(n-1)\cdots(n-r+1).$$

Proof. The recurrence is given explicitly in Eq. (6.3). When $q_r(0) \neq 0$, the maximal value of $i-j$ is obtained with $j = 0$ and $i = r$, where $c_{i,j} = c_{r,0} = q_r(0)$ and the result follows. □

The indicial polynomial gives very precise information on the power series solutions.

Definition 6.7. *The* valuation $\operatorname{val} A$ *of a power series is the smallest index of its nonzero coefficients (and by convention* $\operatorname{val} 0 = \infty$*).*

Proposition 6.6. *If $A \in \mathbb{K}[[x]]$ is a power series solution of Eq. (6.7), then its valuation* $\operatorname{val} A$ *is a zero of the indicial polynomial* $\mathrm{Ind}_0(n)$.

Proof. This follows from evaluating the recurrence Eq. (6.5) satisfied by the coefficients of A at $n = \operatorname{val} A - d$. □

Corollary 6.4. *If $q_r(0) \neq 0$, then Eq. (6.7) has a basis of r power series solutions. They can be taken with valuations $0, \dots, r-1$.*

Proof. This is a consequence of the fact that in this situation, the set H from Proposition 6.4 is empty, together with the previous result. □

6.5.3 Generalized power series

One can extend slightly the class of solutions one is looking for. For instance, the differential equation

$$4x^2y'' + 4xy' + (4x^2 - 1)y = 0$$

has for solution

$$\sqrt{x}\left(1 - \frac{1}{6}x^2 + \frac{1}{120}x^4 - \frac{1}{5040}x^6 + \cdots\right).$$

The following definition captures such solutions.

Definition 6.8. *A series of the form $x^\alpha F(x)$ with F a power series in $\mathbb{K}[[x]]$, $F(0) \neq 0$ and $\alpha \in \overline{\mathbb{K}}$ is called a* generalized power series.

Again, the indicial polynomial lets one detect such solutions.

Proposition 6.7. *If $x^\alpha F(x)$ is a generalized power series solution of Eq. (6.7), then $\mathrm{Ind}_0(\alpha) = 0$. If moreover, $\alpha \notin \{0, \dots, r-1\}$, then $q_r(0) \neq 0$.*

Proof. The idea is to inject $x^\alpha F(x)$ into the equation, multiply by $1/x^\alpha$ and extract the coefficient of x^n. This gives the same recurrence as in Eq. (6.3), but with $n - \alpha$ in place of n. □

6.5.4 Change of points

Instead of considering power series in powers of the variable x, one can consider power series in powers of $x - c$ for some $c \in \overline{\mathbb{K}}$, or generalized power series of the form

$$Y(x - c) = (x - c)^\alpha \sum_{k \geq 0} y_k (x - c)^k. \tag{6.8}$$

Such a generalized power series is a solution of Eq. (6.7) if and only if the power series $Y(x)$ is solution of

$$q_r(x + c)y^{(r)}(x) + \cdots + q_0(x + c)y(x) = 0. \tag{6.9}$$

Definition 6.9. *The indicial polynomial of Eq. (6.7) at c is the indicial polynomial of Eq. (6.9) at 0.*

The same reasoning as before gives a way to find those points where a nontrivial generalized power series solution exists.

Corollary 6.5. *If $Y(x - c)$ is a solution of Eq. (6.7) such that $y_0 \neq 0$ and $\alpha \notin \{0, \dots, r-1\}$, then $q_r(c) = 0$.*

This corollary shows that functions like $\tan x$, that have infinitely many points where they have an expansion of the form Eq. (6.8) with $\alpha = -1$ cannot be differentially finite, as all those points should be roots of the nonzero polynomial q_r, whose number of roots is finite.

6.5.5 Rational solutions

By the previous discussion, the denominator of a rational solution can only vanish at roots c of the leading coefficient q_r. The multiplicity of c as a root of the denominator has to be the opposite of a negative integer root of the indicial polynomial at c. Taking the minimal such integer root at each root of q_r (and 0 when no negative integer root exists) and multiplying over all c such that $q_r(c) = 0$ gives a multiple Q of the possible denominators of the rational solutions of the differential equation. Then, changing the unknown function as $y(x) = P(x)/Q(x)$ gives a linear differential equation in P whose *polynomial* solutions have to be found.

Additional bibliography

The material discussed in this lecture is not usually present in computer algebra books, except

Alin Bostan et al. *Algorithmes Efficaces en Calcul Formel*. Auto-édition, Sept. 2017. ISBN: 979-10-699-0947-2. URL: https://hal.archives-ouvertes.fr/AECF/

In English, part of it can be found in the very accessible

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete mathematics*. 2nd edition. Reading, MA: Addison-Wesley Publishing Company, 1994, pp. xiv+657

and, from a more mathematical point of view, in

Richard P. Stanley. *Enumerative combinatorics*. Vol. 2. Cambridge University Press, 1999, pp. xii+581

Lecture 7

Fast Linear Algebra

Summary

Given an $n \times n$ matrix A with entries in a field $\mathbb{K}$, common questions covered in this lecture are the computation of its determinant, its inverse, or the solution of the linear system $Ay = b$ given a vector b. The model of complexity is in terms of number of arithmetic operations in $\mathbb{K}$. While the questions are of an *exact* nature, it is sometimes possible to take advantage of *approximation*, provided one pays attention to the quality of the result. This is discussed in the first section. The second one recalls the classical Gaussian elimination method. Next, it is shown that many operations can be computed in $O(n^\omega)$ operations, where $\omega < 3$ is the complexity of matrix multiplication. Finally, the equivalence of these problems from the complexity point of view is explored.

7.1 Approximations

Even for exact operations, fast algorithms may rely on approximations in intermediate steps.

7.1.1 Approximation by Floating-Point Numbers

The advantage of using floating-point approximation is that operations become very fast. For instance, a random 8×8 matrix with integers bounded by 10^9 in absolute value has a determinant of order $\approx 10^{70}$, which is correctly and efficiently approximated by a numerical method. However, one can exhibit an innocent-looking 8×8 matrix with integers of that size, for which the numerical computation of the determinant with precision 20, 40 and 60 returns $\approx 10^{35}, \approx 10^{16}, \approx 96$, while the exact result computed in exact arithmetic is found to be exactly 96. On that same example, an inverse computed numerically with precision 20 or 40, when multiplied by the original matrix, is not close to the identity. (Such matrices are well-known in numerical analysis and called *ill-conditioned*.)

Thus the question is to determine how trustful a numerical result is when one wants to extract exact information out of it. For instance, given two matrices M, A it may be possible to certify that A is close to M^{-1} thanks to the following.

Proposition 7.1. *If $\|AM - I\|_\infty < 1$ then A and M are invertible.*

Proof. The spectral radius $\rho(M)$ (the maximum of the absolute values of the eigenvalues) satisfies

$$\rho(M) \leq \|M\|_\infty.$$

(Proof: consider an eigenvector u for an eigenvalue λ of maximum modulus then $\|Mu\| = |\lambda| \cdot \|u\| = \rho(M)\|u\| \leq \|M\|_\infty \|u\|$.)

Thus the hypothesis implies that $\rho(AM-I) < 1$. As the eigenvalues of $AM-I$ are those of AM minus 1, this implies that 0 cannot be an eigenvalue of AM, therefore its determinant is not 0, and neither is that of M or A. So in that situation we can certify that a matrix is invertible from an approximate inverse. □

Now, when we compute this bound numerically, it is necessary to make sure that a numerical value smaller than 1 corresponds to an actual value smaller than 1. In this instance of certifying inverses, this can be achieved by changing the default rounding mode, so that each arithmetic operation in the product AM is computed with a rounding mode towards infinity (in Maple this is done with the environment variable `Rounding`). Then, increasing the precision if necessary, one can certify invertibility numerically.

7.1.2 Approximation by Modular Evaluation

For matrices with rational functions in $\mathbb{Q}(x_1, \ldots, x_k)$ as entries, another type of approximation is achieved by evaluating the variables at random integers modulo a prime number. In particular if the determinant computed that way is not 0, then the exact determinant is not 0. More generally, this method allows to obtain a lower bound on the rank, which is (probabilistically) equal to the actual rank.

7.1.3 Approximation by Series Expansion

In order to solve $AY = b$ where A and b have polynomial entries in $\mathbb{K}[x]$, an idea is to solve as power series and reconstruct rational functions from there by Padé approximants (see Lecture 4). This time, one can certify the result by multiplying the matrix by the resulting vector of rational functions. If the solution does not match, one can increase the precision of the series expansions.

If the matrix is an $n \times n$ matrix of polynomials of degree $< d$ and d is also a bound on the degree of b, then by Cramer's rule, all the entries of the solution are rational functions with numerator and denominator of degree $< nd$, so in order to reconstruct the result, it is known in advance that a precision $2nd$ in the series expansions is sufficient.

In this context, this approach (with a lifting method which is beyond the scope of this course) gives a complexity that is the fastest known.

7.1.4 Approximation for Small Relations (Kannan-Lenstra-Lovász)

This is used to compute factors of polynomials in $\mathbb{Q}[x]$, with numerical computations in intermediate steps. From a numerical approximation λ of a root of the polynomial, the aim is to find a 'small' vector of coefficients so that $p_0 + p_1\lambda + \cdots + p_n\lambda^n \approx 0$. The technique is to consider the matrix

$$A = \begin{pmatrix} 1 & 0 & \ldots & 0 & [C] \\ 0 & 1 & & 0 & [C\lambda] \\ & & \ddots & 0 & \vdots \\ 0 & & & 1 & [C\lambda^n] \end{pmatrix}$$

for a large integer C, where $[\cdot]$ denotes rounding to the nearest integer. Then the algorithm LLL computes small vectors that can be obtained by linear combination of the rows with integer coefficients. The coefficients of the factor of p having λ as its root appear as coefficients of the smallest vector in the basis. They can be certified by division.

7.2 Gaussian Elimination

There are basically two methods for exact linear algebra: Gaussian elimination that we consider in this lecture and Krylov methods.

Definition 7.1. *A matrix in $\mathbb{K}^{n\times m}$ is in* row echelon form *if*

- *the zero rows come after the nonzero ones;*
- *the first nonzero entry in each row is on the right of that of the previous row.*

The process of *Gaussian elimination* brings a matrix A to row echelon form by a sequence of two *elementary* operations: linear combination of a row with a multiple of a row above it and permutation of two rows. Both these operations are linear and can be represented as multiplication of the original matrix on the left by either a simple lower triangular matrix for the first one and a simple permutation matrix for the second one. Because these matrices can be grouped according to their type, one obtains that

$$EQA = U,$$

where the lower triangular matrix E is a product of linear combinations of rows, while the matrix Q is a permutation matrix. Reversing the operations gives the following.

Theorem 7.1. *Using $O(n^3)$ arithmetic operations in $\mathbb{K}$, a matrix $A \in \mathbb{K}^{n\times n}$ can be written*

$$A = PLU,$$

with P a permutation, L a non-singular lower triangular matrix and U in row echelon form.

This form has many applications:

- inverse: if A is invertible, then U is triangular and invertible as well and the inverse is simply $U^{-1}L^{-1}P^{-1}$;
- rank: the number of nonzero rows in U is the rank of A;
- determinant: by computing the product of the diagonal elements of L and U, and the signature of P;
- linear system solving: by solving the triangular U starting from the bottom row.

7.3 Gaussian Elimination is not Optimal

This is the title of a landmark article by Strassen in 1969. The starting point is that matrix multiplication can be achieved in a better complexity that $O(n^3)$ and then many operations can be performed in the same complexity as matrix multiplication. This is parallel to the situation of polynomials where $\mathsf{M}(n)$ becomes the yardstick for fast algorithms.

7.3.1 Fast Matrix Multiplication

Definition 7.2. *A* feasible matrix exponent θ *is a real number such that two matrices in $\mathbb{K}^{n\times n}$ can be multiplied with $O(n^\theta)$ operations in $\mathbb{K}$. The* matrix multiplication exponent *is $\omega = \inf \theta$.*

Strassen proved that $\omega < 3$ by exhibiting a divide-and-conquer algorithm whose complexity is $O(n^{\log_2 7})$ operations and $\log_2 7 \simeq 2.807$. Since then, decreasing the bound on ω has been an active area of research with new progress every year. The last result (in 2025) gives $\omega \leq 2.371339$. The most recent algorithms are not intended to be practical, but aim at reducing the gap with 2 until a definite answer is known.

7.3.2 Reductions

Reducing problems to matrix multiplication is important both theoretically (to obtain algorithms of sub-cubic complexity) and practically. Indeed, matrix multiplication is implemented very efficiently at low level and thus reducing to it can bring huge practical savings. This is true not only in a computer algebra setting, but also in a numerical setting, where numerical libraries rely on carefully handcrafted and very efficient matrix multiplication.

7.3.3 Matrix Inversion

Let $M \in \mathbb{K}^{n\times n}$ and assume for simplicity that n is a power of 2. Then M can be split into 4 blocks

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}.$$

It is easy to check the key identity

$$\begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix} \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix} = \begin{bmatrix} A & 0 \\ 0 & D - CA^{-1}B \end{bmatrix}.$$

Writing $S = D - CA^{-1}B$ (that is called the *Schur complement* of the block D), one deduces the inverse

$$M^{-1} = \begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix} \begin{bmatrix} A^{-1} & 0 \\ 0 & S^{-1} \end{bmatrix} \begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix}.$$

Thus, inversion can be performed recursively on two blocks of half the size. The complexity for doing so satisfies

$$C(n) \leq 2C(n/2) + 6(n/2)^\omega + O(n^2),$$

from which the "Master theorem of divide-and-conquer" allows to deduce the following.

Theorem 7.2. *Matrix inversion can be achieved in $3n^\omega + O(n^2)$ arithmetic operations if all matrices encountered are invertible.*

The next step is to ensure that the blocks are invertible. The algorithm is probabilistic of the Last Vegas type: it either returns the correct result or fails.

7.3.4 Algebraic Preconditioning

The idea is to multiply the original matrix by a random structured matrix in such a way that

- multiplication can be performed fast;
- the result can be recovered fast.

Permutation matrices would be sufficient thanks to the following.

Lemma 7.1. *If A is invertible, there exists a permutation P such that all principal minors of PA are nonzero.*

Proof. This follows from Gaussian elimination: there exists a permutation P such that $PA = LU$ with L lower triangular and U upper triangular. The principal minors in PA are then obtained as the product of the corresponding minors of L and U that are both non-singular since A is. □

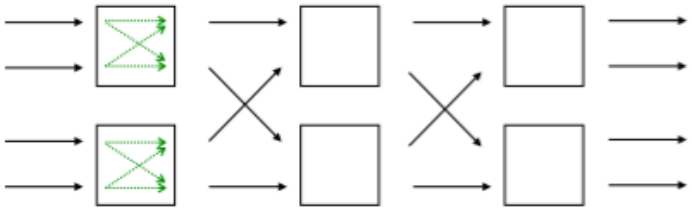
An argument is needed to show that the principal minors of the Schur complement are nonzero as well.

Beneš Networks

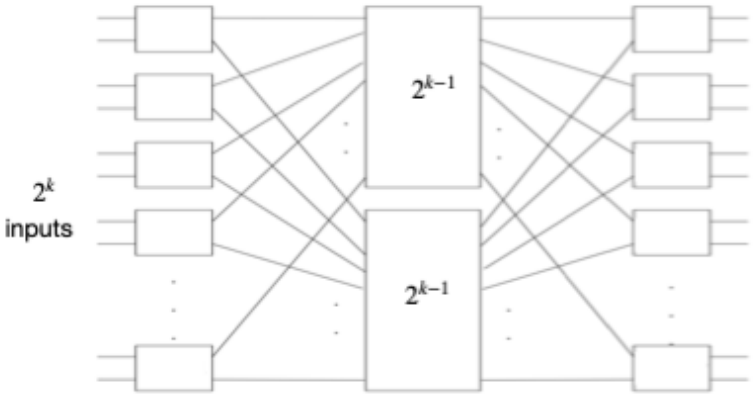
The permutation is constructed inductively from a basic block that can be one of

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

i.e., either the identity matrix or a transposition. To construct all permutations of $\{1, \ldots, 4\}$, one combines such matrices by multiplication. This can be depicted in the form of a network as



where each of the squares represent one of the 2×2 matrices above. Following the arrows from left to right makes it clear that any permutation can be obtained that way. More generally, a *Beneš network* over 2^k inputs is obtained recursively by this construction:



In terms of matrices, the left and right column in this picture are each obtained by one $2^k \times 2^k$ matrix, product of 2^{k-1} instances of a transposition or the identity. For $n = 2^k$, this network has $2k - 1$ layers and can realize any permutation. Computing the product is performed one layer at a time, each for $O(n^2)$ operations, so the whole multiplication costs $O(n^2 \log n)$ operations.

Probabilistic Analysis by the Schwartz-Zippel Lemma

One could draw a permutation at random, but it is unclear what the probability of success (nonzero principle minors) would be. Instead, it is possible to resort to a design technique for probabilistic algorithms in computer algebra based on concentrating the 'bad' situations into the zero-set of a polynomial.

The matrix

$$\mathcal{E}(a, b, c, d) = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

is the identity matrix at $(a, b, c, d) = (1, 0, 0, 1)$ and the transposition matrix at $(0, 1, 1, 0)$. It has a known inverse provided its determinant, a polynomial in a, b, c, d is not zero. The matrix associated to a Beneš network can therefore be interpreted as a matrix with polynomial entries, evaluated at a point (different permutations arising from different points). Its inverse is also easy to compute. There are 4 variables for each 2×2 block, thus a total of $c_k = 4 \times 2^k + 2c_{k-1} = 4(2k+1)2^k = O(n \log n)$ variables. The degree of the entries in the matrix is bounded by the number of layers, that is $2k - 1 = 2 \log n - 1$.

The principal minors in the product PA for a permutation P constructed this way are therefore obtained by evaluating a nonzero polynomial of degree at most $2\log n - 1$ in these variables. The probability of obtaining a 0 minor can be controlled by the following classical tool in the probabilistic analysis of algorithms in computer algebra.

Lemma 7.2 (Schwartz-Zippel). *Given $\Delta(x_1, \ldots, x_\ell) \neq 0$ a polynomial of degree d and $\alpha_1, \ldots, \alpha_\ell$ chosen uniformly and independently from a finite subset S of $\mathbb{K}$, one has*

$$\text{Prob}\left(\Delta(\alpha_1, \ldots, \alpha_\ell) \neq 0\right) \geq 1 - \frac{d}{\text{card}\, S}.$$

Note that the bound does not depend on the number of variables.

Proof. Let $s = \text{card}\, S$. By induction on the number of variables we show that the polynomial has at most $ds^{\ell-1}$ in S^ℓ. For 1 variable, this is clear, as the polynomial has at most d zeros. For ℓ variables, write the polynomial as

$$\Delta(x_1, \ldots, x_\ell) = q_k(x_1, \ldots, x_{\ell-1})x_\ell^k + \cdots + q_0(x_1, \ldots, x_{\ell-1}),$$

with $\deg q_i \leq d - i$ and $q_k \neq 0$. By the induction hypothesis, the leading coefficient has at most $(d-k)s^{\ell-2}$ zeros in $S^{\ell-1}$, that correspond to at most $(d-k)s^{\ell-1}$ zeros of Δ since x_ℓ takes at most s values. When the leading coefficient does not vanish, then there are at most k values of x_ℓ where Δ vanishes. Thus, in total, we have at most

$$(d-k)s^{\ell-1} + ks^{\ell-1} = ds^{\ell-1}$$

zeros, as was to be proved. □

Thus if $\mathbb{K}$ is a field with more than $d = 2\log n - 1$ elements one can take each of the $O(n \log n)$ variables uniformly at random in a sufficiently large set, multiply the input matrix A by the resulting Beneš network, perform Strassen's inversion and multiply back by the inverse of the network. This succeeds with a nonzero probability that can be made arbitrarily close to 1 by repeating the process until success is encountered. If the field $\mathbb{K}$ is not large enough, a common technique is to construct an algebraic extension, which leads to an extra factor of order $d = O(\log n)$ for each of the arithmetic operations.

7.4 Program Transformations and Complexity

In the previous section it is shown that the inversion of a nonsingular matrix in $\mathbb{K}^{n\times n}$ can be achieved in $O(n^\omega)$ operations, i.e., for a constant number of matrix multiplications. It turns out that in the same complexity, it is possible to compute: the determinant; the solution of a linear system $Ax = b$ for $b \in \mathbb{K}^n$ (or detect that no such solution exists); the rank; an echelon form; the row rank profile (minimal indices of linearly independent rows); a basis of the kernel; the characteristic polynomial.

This section shows that in many cases, one cannot expect a better complexity: not only do these operations cost as much as matrix multiplication, but also improving them would give a faster algorithm for matrix multiplication. More precisely, we now show that

1. the inverse of a matrix can be computed in $O(C_{\text{det}})$ where C_{det} is the complexity of the determinant (in a certain complexity model);
2. the product of matrices can be computed in $O(C_{\text{inv}})$, where C_{inv} is the complexity of matrix inversion (paying attention to the complexity model too).

The previous section shows that $C_{\text{inv}} = O(n^\omega)$ and a variant of the algorithm giving this result also implies that $C_{\text{det}} = O(n^\omega)$. The results presented here then imply

$$n^\omega = O(C_{\text{inv}}) = O(C_{\text{det}}) = O(n^\omega),$$

so that all these operations have the same complexity, up to constant factors.

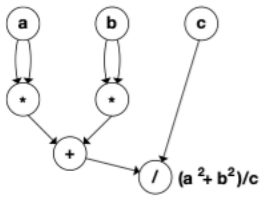


Figure 7.1: Circuit evaluating $(a^2 + b^2)/c$

7.4.1 Automatic Differentiation of Straight-Line Programs

Definition 7.3. A straight-line program *(SLP) over a field* $\mathbb{K}$ *is a sequence of elementary instructions of the form* $v_i = a_i \in \mathbb{K}$ *or* $v_i = \circ(a_i, b_i)$ *with* $\circ \in \{+, -, \times, \div\}$ *and* a_i *and* b_i *either a constant in* $\mathbb{K}$ *or a* v_j *with* $j < i$*; the input is given as the first* v_i*. The number of instructions is the* length *of the program. (It is understood that* a/b *raises an error if* b *is not invertible.)*

Example 7.1. The following SLP computes $(x^4 - 1)/(x - 1)$:

$$t_0 := x; t_1 := t_0^2; t_2 := t_1^2; t_3 := t_0 - 1; t_4 := t_2 - 1; t_5 := t_4/t_3.$$

(An analogous decomposition is performed in Maple by the command `codegen[optimize]` that is used for code generation.)

A SLP can also be viewed as an arithmetic circuit in the form of a directed acyclic multigraph, as shown in Fig. 7.1. Such a program or graph computes a rational function of its input variables. The remarkable result is that, whatever the number of input variables, the gradient (the list of partial derivatives) with respect to those input variables can be computed in roughly the same cost as the original program.

Theorem 7.3 (Baur-Strassen 1983)**.** *Given a SLP of length* L *for* $f \in K(x_1, \ldots, x_n)$*, there exists a SLP of length* $O(L)$ *for the* n *first-order partial derivatives of* f*.*

Example 7.2. Even with $n = 1$, this result is not trivial. With $p(x) = (x - 1) \times \cdots \times (x - n)$, the naive program for the computation of the derivative from the formula

$$p' = (x - 2) \times \cdots \times (x - n) + \cdots + (x - 1) \times \cdots \times (x - n - 1)$$

has complexity $O(n^2)$. From a straight-line program computing p, like

$$t_0 := x; t_1 := t_0 - 1; t_2 := t_0 - 2; \ldots; t_n := t_0 - n;$$
$$t_{n+1} = t_1 \times t_2; t_{n+2} = t_{n+1} \times t_3; \ldots; t_{2n-1} = t_{2n-2} \times t_n,$$

the computation can proceed backwards: since $p(x) = q(x) \times (x - n)$, then $p'(x) = q'(x) \times (x - n) + q(x)$. Thus the derivative is obtained by the program

$$t_0 := x; t_1 := t_0 - 1; t_2 := t_0 - 2; \ldots; t_n := t_0 - n;$$
$$t_{n+1} = t_1 \times t_2; t'_{n+1} = t_1 + t_2;$$
$$\ldots$$
$$t_{2n-1} = t_{2n-2} \times t_n; t'_{2n-1} = t'_{2n-2} \times t_n + t_{2n-2}.$$

The length of the program for p is $2n$; the program for p' has length only $3n-1$.

Proof of Theorem 7.3. Let $x_1, \ldots, x_n$ be the input variables and $g_1, \ldots, g_\ell$ be the variables introduced at each step so that $g_\ell = f(x_1, \ldots, x_n)$. Write $\Delta_0 = f$ and to each of the g_i, associate a function $\Delta_i(x_1, \ldots, x_n, g_1, \ldots, g_i)$ expressing f in terms of all the variables up to g_i, so that

$$f(x_1, \ldots, x_n) = \Delta_0(x_1, \ldots, x_n) = \Delta_1(x_1, \ldots, x_n, g_1) = \cdots = \Delta_\ell(x_1, \ldots, x_n, g_1, \ldots, g_\ell) = g_\ell.$$

The partial derivatives of Δ_i will be obtained in terms of those of Δ_{i+1} by the chain rule in a constant number of operations, proving the theorem. The starting point of the induction is that

$$\frac{\partial \Delta_\ell}{\partial g_i} = \begin{cases} 1 \text{ if } i = \ell, \\ 0 \text{ otherwise.} \end{cases}$$

Next, we consider the case when $\circ \in \{+, -, \times, \div\}$. These are all similar, we treat $\div$ in detail. If $g_{i+1} = g_j/g_k$ with j, k less than $i+1$, then

$$\Delta_i(x_1, \ldots, x_n, g_1, \ldots, g_i) = \Delta_{i+1}(x_1, \ldots, x_n, g_1, \ldots, g_i, g_j/g_k),$$

so that

$$\frac{\partial \Delta_i}{\partial g_m} = \begin{cases} \frac{\partial \Delta_{i+1}}{\partial g_m}, & \text{if } m \notin \{j, k\}, \\ \frac{\partial \Delta_{i+1}}{\partial g_j} + \frac{1}{g_k} \frac{\partial \Delta_{i+1}}{\partial g_{i+1}}(x_1, \ldots, x_n, g_1, \ldots, g_i, g_j/g_k), & \text{if } m = j, \\ \frac{\partial \Delta_{i+1}}{\partial g_k} - \frac{g_j}{g_k^2} \frac{\partial \Delta_{i+1}}{\partial g_{i+1}}(x_1, \ldots, x_n, g_1, \ldots, g_i, g_j/g_k), & \text{if } m = k. \end{cases}$$

Thus, from the partial derivatives of Δ_{i+1}, only 5 operations are required to compute all the partial derivatives of Δ_i. (The other operations $\circ \in \{+, -, \times\}$ need even fewer operations.) □

In Maple, this operation is implemented in the `codegen[GRADIENT]` function. This is in particular very useful to find zeros of a function given by a program by means of Newton iteration.

7.4.2 From Determinant to Inversion by Differentiation

The *Laplace expansion* along the jth column of the determinant of a matrix $A = (a_{i,j})$ is

$$\det A = \sum_{i=1}^{n} a_{ij} C_{ij}, \tag{7.1}$$

where C_{ij} is the determinant of the submatrix obtained by removing the ith row and the jth column of A. The C_{ij} are the entries of the *cofactor matrix* of A, that satisfies

$$AC^{\mathsf{T}} = (\det A) I_n.$$

This implies that, up to a constant factor $\det A$, the entries of the comatrix give those of A^{-1}. Differentiating Eq. (7.1) shows that these entries are exactly the partial derivatives of $\det A$ with respect to the entries of A. In follows from the previous section that from any SLP computing the determinant of A in L operations, one can deduce a SLP for A^{-1} in at most $5L$ operations. In other words, *computing the inverse has the same complexity as computing the determinant.*

7.4.3 From Inversion to Matrix Multiplication

Winograd observed the following reduction

$$\begin{bmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{bmatrix}^{-1} = \begin{bmatrix} I & -A & A \cdot B \\ 0 & I & -B \\ 0 & 0 & I \end{bmatrix},$$

whose verification is straightforward. It follows that any algorithm that is able to compute the inverse of a $3n \times 3n$ matrix is also able to compute the product of $n \times n$ matrices in the same complexity.

In summary, we have showed that inversion and determinant are equivalent to matrix multiplication from the complexity point of view. Other operations are also known to be in the same complexity class: rank, basis of the kernel, LU decomposition and computation of the characteristic polynomial (this last result was proved in 2021). At this point, only one fundamental problem resists this reduction: solving systems of linear equation for which it is not known whether a method faster than $O(n^\omega)$ could exist or not.

7.5 Transposition

Recall from Lecture 4 that multipoint evaluation of a polynomial of degree n in $\mathbb{K}[x]$ can be achieved in $O(\mathsf{M}(n)\log n)$ arithmetic operations in $\mathbb{K}$ and that interpolation can be achieved in the same complexity. We now prove the following.

Theorem 7.4 (Bostan-Schost 2004)**.** *A SLP of length L for interpolation at $a_1, \ldots, a_n$ with distinct a_i can be transformed into a program for multipoint evaluation at these points of length $O(L) + O(\mathsf{M}(n))$.*

In particular, for special sets of points a_i for which one can interpolate in only $O(\mathsf{M}(n))$, one gets multipoint evaluation at the same cost.

This section is devoted to a proof of this result. Recall that evaluation of a polynomial P of degree $< n$ at $a_1, \ldots, a_n$ corresponds to multiplication of the vector of coefficients of P by the Vandermonde matrix associated with the a_i,

$$V = \begin{bmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^{n-1} \\ 1 & a_2 & a_2^2 & \cdots & a_2^{n-1} \\ 1 & a_3 & a_3^2 & \cdots & a_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_n & a_n^2 & \cdots & a_n^{n-1} \end{bmatrix}.$$

Interpolation is the multiplication of the vector of values by V^{-1}. It is the operation that is given by the SLP in the theorem. Instead of computing the product by V directly, the algorithm starts from the product $H = V^{\mathsf{T}} \cdot V$. This is

$$H = \begin{bmatrix} \sum_k 1 & \sum_k a_k & \sum_k a_k^2 & \cdots & \sum_k a_k^{n-1} \\ \sum_k a_k & \sum_k a_k^2 & \sum_k a_k^3 & \cdots & \sum_k a_k^n \\ \sum_k a_k^2 & \sum_k a_k^3 & \sum_k a_k^4 & \cdots & \sum_k a_k^{n+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum_k a_k^{n-1} & \sum_k a_k^n & \sum_k a_k^{n+1} & \cdots & \sum_k a_k^{2n-2} \end{bmatrix}.$$

This matrix is a *Hankel matrix*: its entries are constant along ascending diagonals. Here, its entries are the Newton sums (or power sums) of the a_i,

$$h_i = \sum_{k=1}^{n} a^i.$$

This implies that the entries can be computed in $O(\mathsf{M}(n))$ operations from their generating function: if $\hat{A} = \prod(1 - xa_i)$, then

$$\frac{\hat{A}'}{\hat{A}} = -\sum \frac{a_i}{1 - xa_i} = \sum_{i \geq 1} \left(\sum_k a_k^i \right) x^i.$$

(The complexity follows by Newton's iteration, see Lecture 3.) The consequence of the structure of the matrix H is that it can be multiplied by a vector in only $2\mathsf{M}(n)$ operations: if

$$H \begin{bmatrix} p_{n-1} \\ \vdots \\ p_0 \end{bmatrix} = \begin{bmatrix} q_0 \\ \vdots \\ q_{n-1} \end{bmatrix},$$

then it is a direct observation that the entries are related by

$$(h_0 + h_1 x + \cdots + h_{2n-2}x^{2n-2})(p_0 + xp_1 + \cdots + x^{n-1}p_{n-1}) = \cdots + x^{n-1}(q_0 + q_1 x + \cdots + q_{n-1}x^{n-1}) + \cdots .$$

This shows that the coefficients $q_0, \ldots, q_{n-1}$ are recovered from the terms of degree x^{n-1} to x^{2n-2} in this polynomial of degree $3n-3$. (This operation where only the terms of degree $n, \ldots, 2n$ are needed in the product of a polynomial of degree $2n$ by a polynomial of degree n is called the *middle product*.) The algorithm computing multipoint evaluation of $P(x)$ at $a_1, \ldots, a_n$ follows:

1. compute $\overline{P} = x^{n-1}P(x)$ (no arithmetic operation);
2. compute the product $h(x)P(x)$ and extract its middle part in $O(\mathsf{M}(n))$ operations;
3. multiply the resulting vector by the transpose of V^{-1}.

For this last operation, we are given as input a program that computes the product by V^{-1}. The proof of the theorem is then a consequence of the following.

Proposition 7.2 (Transposition principle)**.** *Given a SLP program of length L that computes $A \cdot u$, one deduces a program for $A^{\mathsf{T}} \cdot v$ in $O(L)$ operations.*

Proof. From the program $u \mapsto A \cdot u$, one easily deduces a program for $(u, v) \mapsto v^{\mathsf{T}} \cdot A \cdot u$, which computes

$$f(u_1, \ldots, u_n) = \sum_{i,j} v_i a_{ij} u_j.$$

The partial derivatives are

$$\frac{\partial f}{\partial u_j} = \sum_i v_i a_{ij} = (v^{\mathsf{T}} \cdot A)_j = (A^{\mathsf{T}} \cdot v)_j,$$

giving the product by A^{T}. □

This proof gives the announced result in $O(L)$, but a tighter bound $L - n + m$ is possible for $A \in \mathbb{K}^{n \times m}$. We leave it as an exercise to show that as a consequence, the middle-product can be obtained in only $\mathsf{M}(n) + O(n)$ operations.

Lecture 8

Polynomial and Integer Matrices

Summary

In the previous lecture, the algorithms deal with matrices with entries in a field and their complexity estimates measure the number of arithmetic operations that are used.

For integer or polynomial matrices, these estimates do not reflect the time complexity, as the intermediate computations and the results may turn out to be much larger than the input.

This lecture shows how one can control the growth in the computation, using new tools designed specifically for integer or polynomial matrices.

Example 8.1. Gaussian elimination on the integer matrix

$$A = \begin{bmatrix} 60 & -50 & 22 & 19 \\ 84 & -29 & 39 & 26 \\ 85 & -96 & 5 & -36 \\ -45 & -20 & -71 & -79 \end{bmatrix}$$

produces a sequence of more and more triangular matrices of larger and larger rational numbers

$$\begin{bmatrix} 60 & -50 & 22 & 19 \\ 0 & 41 & \frac{41}{5} & -\frac{3}{5} \\ 0 & -\frac{151}{6} & -\frac{157}{6} & -\frac{755}{12} \\ 0 & -\frac{115}{2} & -\frac{109}{2} & -\frac{259}{4} \end{bmatrix}, \quad \begin{bmatrix} 60 & -50 & 22 & 19 \\ 0 & 41 & \frac{41}{5} & -\frac{3}{5} \\ 0 & 0 & -\frac{317}{15} & -\frac{155681}{2460} \\ 0 & 0 & -43 & -\frac{10757}{164} \end{bmatrix}, \quad \begin{bmatrix} 60 & -50 & 22 & 19 \\ 0 & 41 & \frac{41}{5} & -\frac{3}{5} \\ 0 & 0 & -\frac{317}{15} & -\frac{155681}{2460} \\ 0 & 0 & 0 & \frac{1642157}{25994} \end{bmatrix}.$$

Denoting by L_i the rows of the matrices, the transformation from A to the last matrix is a sequence of operations of the type $L_i := L_i - \alpha L_j$ for some $j < i$. This leaves the determinant invariant and in the end we obtain in particular the integer $\det A = -3284314$ as a product of the rational diagonal elements of the last matrix. The sequence above was obtained by taking for pivots the diagonal elements. One could also start each stage from the bottom row and add a linear combination with a random row above it that has the same number of initial 0. The final matrix is the same, but empirically at least, the intermediate rational numbers that occur in the computation are much larger, so much so that the bit complexity does not remain polynomial in the dimension.

It is thus important to keep control over all sizes that play a role in the computations. The situation here is similar to what happens for multiplication: algorithms for matrices of polynomials resemble their counterpart for matrices of integers, but are usually easier to state and analyse.

Notation 8.1. *We write $\mathbb{K}[x]_{\leq d}^{n \times m}$ for the set of $n \times m$ matrices with polynomial entries whose degree is at most d.*

8.1 Determinant Bounds

For a matrix $A \in \mathbb{K}[x]_{\le d}^{n\times n}$, using the Leibniz formula for its determinant in terms of permutations shows that $\deg \det A \le nd$. Over subrings of $\mathbb{R}$, one can use the following.

Proposition 8.1 (Hadamard's inequality on the determinant)**.** *For $A \in \mathbb{R}^{n\times n}$,*

$$|\det A| \le \|A_1\|_2 \cdots \|A_n\|_2,$$

where the A_i are the columns of A.

The geometric meaning is that the determinant is a measure of the volume marked out by the column vectors, which is bounded by the case when they are orthogonal. A consequence is that if the entries of A are all bounded in absolute value by β, one obtains

$$|\det A| \le n^{n/2}\beta^n.$$

Proof. A 1-line proof is by Gram-Schmidt orthogonalization; a slightly longer but more elementary one can be found on Wikipedia. □

8.2 Gauss-Bareiss Elimination

Denote by $a_{ij}^{[k]}$ the (i,j) entry of the matrix obtained from A after pivoting with the entry (k,k), i.e., after adding to row i the product of row k with $-a_{ik}^{[k-1]}/a_{kk}^{[k-1]}$. This means that

$$a_{ij}^{[k]} = a_{ij}^{[k-1]} - \frac{a_{ik}^{[k-1]}}{a_{kk}^{[k-1]}} a_{kj}^{[k-1]}. \tag{8.1}$$

A remarkable result is that these entries can be related to minors of the original matrix. If $A = (a_{ij})$, u and v are integers or integer intervals, denote by A_{uv} the submatrix obtained by extracting from A the rows in u and the columns in v. Then, define

$$a_{ij}^{(k)} = \begin{vmatrix} A_{1..k,1..k} & A_{1..k,j} \\ A_{i,1..k} & a_{ij} \end{vmatrix}$$

and note that $\det A_{1..k,1..k} = a_{kk}^{(k-1)}$.

Theorem 8.1 (Bareiss 1968)**.** *For all i,j,k where $a_{ij}^{[k]}$ is defined, one has*

$$a_{ij}^{[k]} = \frac{a_{ij}^{(k)}}{a_{kk}^{(k-1)}}.$$

Proof. A consequence of an identity on determinants due to Sylvester (1851) is that the minors $a_{ij}^{(k)}$ are related by

$$a_{i,j}^{(k)} a_{k-1,k-1}^{(k-2)} = a_{kk}^{(k-1)} a_{ij}^{(k-1)} - a_{ik}^{(k-1)} a_{kj}^{(k-1)}. \tag{8.2}$$

The proof follows by induction. For $k = 0$, $a_{ij}^{(k)} = a_{ij}$ and the principal minor of order 0 is 1. Next, injecting the induction hypothesis in Eq. (8.1) gives

$$a_{ij}^{[k]} = \frac{a_{ij}^{(k-1)}}{a_{k-1,k-1}^{(k-2)}} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} \frac{a_{kj}^{(k-1)}}{a_{k-1,k-1}^{(k-2)}}.$$

Multiplying both sides by $a_{kk}^{(k-1)}$ and using Eq. (8.2) concludes the proof. □

Two consequences follow: the elements on the diagonal of the final triangular matrix are quotients of successive principal minors of A; the size of all the entries in the intermediate matrices are controlled by Hadamard's bound which makes the complexity of the algorithm on integer matrices polynomial.

This reasoning assumes that the pivot at position (k, k) is always nonzero. The result persists otherwise, by performing an appropriate permutation of the rows of the matrix.

8.3 Polynomial Matrices by Gaussian Elimination

8.3.1 Determinant

Since the determinant has degree $\leq nd$ it can be computed by evaluation-interpolation in $O(n^4 d)$ operations using cubic Gaussian elimination at each point, which reduces to $O(n^{\omega+1}d)$ if one uses fast determinant at each point. By contrast, one can multiply two matrices in $\mathbb{K}[x]_{\leq d}^{n\times n}$ in only $\tilde{O}(n^\omega d)$ operations (e.g., by evaluation-interpolation, or if the ring is too small, by viewing the matrices of polynomials as polynomials with matrix coefficients and using an algorithm due to Cantor and Kaltofen that performs the product of polynomials of degree $\leq d$ over any algebra in $\tilde{O}(d)$ operations in the algebra).

8.3.2 Linear System Solving

Consider the linear system $A(x)Y(x) = b(x)$ with $\deg A, b \leq d$. By Cramer's formula, the entries of the solution is a rational function with numerator and denominator of degree upper bounded by $2nd$. This gives an algorithm based on Padé approximants to solve the system:

1. Compute the truncated power series $\hat{Y}(x)$ such that $A(x)\hat{Y}(x) = b(x) \bmod x^{2nd+1}$ (e.g., by Newton's iteration);
2. Reconstruct $Y(x)$ from $\hat{Y}(x)$ by rational reconstruction.

The second step costs $\tilde{O}(n^2 d)$, its complexity is dominated by that of the first step, in $\tilde{O}(n^{\omega+1}d)$ operations.

For this algorithm to succeed, the series expansion must exists, which is the case when $\det A(0) = 0$. If not, one picks another point $\alpha \in \mathbb{K}$ where $A(\alpha)$ is nonsingular and solves the system $A(x+\alpha)Y(x+\alpha) = b(x+\alpha)$. Shifting the matrix A and the vector b can be achieved by evaluation-interpolation in a complexity that is negligible compared to the resolution.

Thus in both cases the cost obtained by these algorithms is of order the product of the algebraic cost n^ω by the output size nd. Faster algorithms have appeared since the 1990s that decrease this complexity and the one of other problems on polynomial matrices to only $\tilde{O}(n^\omega d)$. This is the object of the next sections.

8.4 Minimal Approximant Bases

The key tool that is as the basis of fast algorithms is the following.

Definition 8.1. *Given a matrix of power series $H \in \mathbb{K}[[x]]^{n\times m}$ with $m \geq n$, an* approximant basis *of H at order σ is a matrix $B \in \mathbb{K}[x]^{m\times m}$ such that $HB = O(x^\sigma)$ and all vectors v such that $Hv = O(x^\sigma)$ are linear combinations with polynomial coefficients of the columns of B. The basis B is called* minimal *when the degree of its columns is minimal.*

Example 8.2. The following formula shows the product HB in a case where $n = 1, m = 2, \sigma = 6$:

$$\begin{bmatrix} 5 + 3x + 2x^2 + x^3 + x^4 & -1 \end{bmatrix} \begin{bmatrix} x^2 - x - 1 & 2x^4 - 3x^3 \\ -8x - 5 & x^4 - 15x^3 \end{bmatrix} = \begin{bmatrix} x^6 & x^6 - x^7 + 2x^8 \end{bmatrix}$$

The first column of B gives the denominator and numerator of a Padé approximant of order $(1, 2)$ of $H_{1,1}$.

More generally, for a square matrix A, a minimal approximant basis of $[A \ -I]$ splits into two blocks D, N such that $AD - N = O(x^\sigma)$, showing how these bases give a matrix variant of Padé approximants.

8.4.1 Kernel Basis

Intuitively, if σ becomes large enough, the columns of a minimal approximant basis should contain a basis of the kernel of the matrix A. This can be made more precise thanks to the following result, that we give without proof.

Theorem 8.2. *Generically, a matrix $A \in \mathbb{K}[x]^{m \times 2m}$ of degree d has a kernel basis over $\mathbb{K}[x]$ of degree at most d.*

Here, 'generically' means that there exists a nonzero polynomial P in the entries a_{ij} of A such that whenever $P(a_{ij}) \neq 0$, the result holds.

Corollary 8.1. *Generically, a minimal approximant basis of $A \in \mathbb{K}[x]^{m \times 2m}_{\leq d}$ at precision $\sigma = 2d + 1$ contains m columns of degree $\leq d$ that form a basis of the kernel of A.*

Proof. By definition of the approximant basis, if $A(x)u(x) = 0$, then $u(x)$ is a $\mathbb{K}[x]$-linear combination of the columns of B. Generically, the kernel of A has dimension m and by the previous theorem, it has a basis of degree bounded by d. Therefore by minimality, B has at least m linearly independent columns of degree bounded by d. It does not contain more such columns since generically the dimension of the kernel is m. □

Note also that as a consequence, if $\deg v \leq d$ and $Av = O(x^{2d+1})$ then $Av = 0$.

8.4.2 Computation of an Approximant Basis

Incremental Computation

A first algorithm proceeds incrementally. We present it in the simpler case when H has dimension $1 \times m$. (This special case, which generalizes Padé approximants, is interesting in its own right as it corresponds to Hermite-Padé approximants — the tool underlying the guessing routines of `gfun` used in the tutorials.)

At step $i < \sigma$, one has a matrix B_i such that $HB_i = O(x^i)$. Initially $B_0 = I_m$. At step i, one can write $HB_i = R + O(x^{i+1})$ with $R \in \mathbb{K}^{1 \times m}$. If $R = 0$ then one can take $B_{i+1} = B_i$. Otherwise, take as pivot the column C of B_i which is such that: the corresponding column of R is not zero; among those, its degree is minimal; among those, the index of its highest degree polynomial is minimal. Then, by linear combination with a constant coefficient of each of the other columns of B_i with C, the corresponding entry of R can be set to 0 and finally one replaces C by xC to cancel the remaining entry of R. Proceeding this way, increasing the degree of only one column, one gets from $O(x^i)$ to $O(x^{i+1})$. As one always picks the column of minimal degree, these increments spread out in the matrix and one ends up, in general, with a matrix of degree roughly σ/m. We state without proof that this does produce a minimal approximant basis.

Divide-and-Conquer Algorithm

We sketch an algorithm that takes advantage of fast polynomial multiplication in the same way as in the half-gcd algorithm of Lecture 4. The basic idea is that if one has computed an approximant basis $B^{(1)}$ at order $\sigma/2$, then by multiplication one can compute a new matrix H' s.t.

$$HB^{(1)} = x^{\sigma/2}H' \bmod x^{\sigma}.$$

From there, one computes an approximant basis $B^{(2)}$ of H' at precision $\sigma/2$, so that

$$HB^{(1)}B^{(2)} = x^{\sigma/2}H'B^{(2)} \bmod x^{\sigma} = O(x^{\sigma})$$

and thus $B = B^{(1)}B^{(2)}$ satisfies the first condition $HB = O(x^{\sigma})$ of an approximant basis.

If this was always an approximant basis, one would obtain a complexity $C(\sigma) = 2C(\sigma/2) + \tilde{O}(m^{\omega}\sigma) = \tilde{O}(m^{\omega}\sigma)$ operations. Actually, a variant can be designed in the same way that the half-gcd algorithm needs adjustment of the degrees in the intermediate steps. The complexity is then as in the simplified situation above. We state without proof the result of this line of work.

Theorem 8.3. *Given $A \in \mathbb{K}[[x]]^{m\times 2m}$, an approximant basis of order $O(\sigma)$ can be computed in $\tilde{O}(m^{\omega}\sigma)$ arithmetic operations.*

Corollary 8.2. *Given $A \in \mathbb{K}[x]^{n\times n}_{\leq d}$, a kernel basis of A can be computed in $\tilde{O}(n^{\omega}d)$ arithmetic operations.*

Proof. At least in generic situations, this is a consequence of the previous theorem and of Corollary 8.1. □

8.5 Fast Matrix Inverse

We consider a matrix $M \in \mathbb{K}[x]^{n\times n}_{\leq d}$. Recall that Strassen's algorithm for the inverse of a matrix proceeds recursively on blocks of half the size by computing the transformation

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \mapsto \begin{bmatrix} A & 0 \\ 0 & D - CA^{-1}B \end{bmatrix}.$$

When applied with integer or polynomial matrices, what happens is that the size of the entries increases too much along the diagonal. The idea of the new algorithm (Jeannerod, Villard 2005) is to use kernel bases of the top and lower halves of the matrix to reach the same situation. More precisely, let $\overline{U} \in \mathbb{K}[x]^{n/2\times n}$ be a (left) kernel basis of the second half of the columns, i.e.,

$$\overline{U}\begin{bmatrix} B \\ D \end{bmatrix} = 0.$$

(Compared to the previous convention, this is obtained as the transpose of the kernel basis of the transpose of the block formed by the last $n/2$ columns of A.) Similarly, let $\underline{U} \in \mathbb{K}[x]^{n/2\times n}$ be a left kernel basis of the first half of the columns. Stacking these two matrices one obtains

$$\begin{bmatrix} \overline{U} \\ \underline{U} \end{bmatrix}\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} A' & 0 \\ 0 & D' \end{bmatrix}.$$

Compared to Strassen's algorithm what happens, generically, is that the degree of the matrix $U = [\overline{U}, \underline{U}]^{\mathsf{T}}$ is bounded by d and the degrees of A' and D' are therefore bounded by $2d$. Computing again the left kernels

of the first and second halves of the columns of A' and D' and stacking the blocks properly gives a new matrix U' of degree bounded by $2d$ such that

$$U'UM = \begin{bmatrix} * & 0 & 0 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{bmatrix},$$

where each entry denotes a block of size $(n/4) \times (n/4)$, with entries of degree bounded by $4d$. Iterating this process $\log_2 n$ steps, we get a polynomial matrix $V = \cdots U''U'U$ and a diagonal matrix D such that $VM = D$ and therefore

$$M^{-1} = D^{-1}V.$$

The matrix V has degree bounded by $(d + 2d + 4d + \cdots + 2^{\log_2 n} d) \leq 2nd$ and similarly for the matrix D. Efficiency is achieved by making use of the block structure of the matrices. At step $i = 1, \ldots, \log_2 n$, the computation performs:

1. 2^i computations of approximant bases of dimension $n/2^i$ and degree $2^{i-1}d$;
2. the update of the diagonal blocks with 2^i multiplications of blocks of degree $2^{i-1}d$;
3. the update of the product on the left with 2^{i-1} block-rows of dimension $n/2^i$ and degree $2^{i-1}d$.

In total, the complexity for computing $U, U', U'', \ldots$ and D is bounded by a constant times

$$\sum_{i=1}^{\log_2 n} \left(\frac{n}{2^i}\right)^{\omega} 2^i d = O(n^{\omega} d).$$

This completes the (sketch of the) proof of the following.

Theorem 8.4. *The inverse of an $n \times n$ polynomial matrix of degree d can be computed in essentially optimal time $\tilde{O}(n^3 d)$.*

Note that more work is needed to extend the idea presented here that works generically to an algorithm that works in all cases. This has been completed by Zhou, Labahn and Storjohann in 2015.

Example 8.3. A surprising application is to the computation of $A, A^2, \ldots, A^n$ for a constant matrix A in only $\tilde{O}(n^3)$ operations (which is optimal). The starting point is the expansion

$$(I - xA)^{-1} = I + xA + x^2A^2 + \cdots.$$

Thus one computes the inverse of $I - xA$ in $\tilde{O}(n^3)$ operations by the previous theorem and then the series expansion at order n of each of its rational entries. By Newton's iteration (Lecture 3) this uses $\tilde{O}(n)$ operations for each entry, whence a total complexity of $\tilde{O}(n^3)$ for that step too.

Lecture 9

Hypergeometric Summation

Summary

Hypergeometric sequences occur in a large number of formulas from combinatorics, probability or special function theory. Their indefinite sums, when hypergeometric, can be computed by Gosper's algorithm. For definite sums, the starting point of all the modern developments is Zeilberger's algorithm. This algorithm produces a linear recurrence with polynomial coefficients. The hypergeometric solutions of such recurrences can be found by Petkovšek's algorithm.

In this lecture, $\mathbb{K}$ denotes an arbitrary field *of characteristic 0.*

Typical sums that can be computed by the algorithms presented here are

$$\sum_{k=0}^{n} \frac{4^k}{\binom{2k}{k}} = \frac{(2n+1)4^{n+1}}{3\binom{2(n+1)}{n+1}} + \frac{1}{3} \tag{9.1}$$

$$\sum_{k=1}^{n} \frac{(-1)^{k+1}\,(4k+1)\,(2k)!}{4^k\,(2k-1)\,k!\,(k+1)!} = \frac{2\,(-1)^{n+1}\,(n+2)\,(2n+2)!}{4^{n+1}\,(2n+1)\,(n+2)!\,(n+1)!} + 1 \tag{9.2}$$

$$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n} \tag{9.3}$$

$$\sum_{k=0}^{n} \binom{2n-2k}{n-k}\binom{2k}{k} = 4^n. \tag{9.4}$$

For another similar sequence,

$$u_n = \sum_{k=0}^{n} \binom{n}{k}^2 \binom{n+k}{k}^2,$$

that plays an important role in Apéry's proof that $\zeta(3)$ is irrational, Zeilberger's algorithm finds

$$(n+1)^3 u_{n+1} - (34n^3 + 51n^2 + 27n + 5)u_n + n^3 u_{n-1} = 0$$

and Petkovšek's algorithm proves that (u_n) cannot be made much simpler, in the technical sense that it is not hypergeometric (Definition 9.2 below).

9.1 Gosper's Algorithm for Indefinite Summation

9.1.1 Indefinite Sums

Definition 9.1. *An* indefinite sum *of a sequence* $(f_k) \in \mathbb{K}^{\mathbb{N}}$ *is a sequence* $(F_k) \in \mathbb{K}^{\mathbb{N}}$ *such that*

$$F_{k+1} - F_k = f_k, \qquad k \in \mathbb{N}.$$

Indefinite sums are discrete analogues of primitives: given such a sum, one can compute sums between endpoints, as

$$\sum_{k=m}^{p} f_k = F_{p+1} - F_m.$$

In the case of integrals, there are functions without any nice primitive, but whose integral over some domain can be expressed simply. The discrete analogue of this phenomenon will be the object of Section 9.2.

9.1.2 Hypergeometric Sequences

The basic algorithms for symbolic summation focus on a special class of sequences, that are polynomially finite (Lecture 6) or order 1.

Definition 9.2. *The sequence* $(u_k) \in \mathbb{K}^{\mathbb{N}}$ *is* hypergeometric *if there exist two non-zero polynomials* P *and* Q *in* $\mathbb{K}[k]$ *such that*

$$Q(k)u_{k+1} = P(k)u_k, \qquad k \in \mathbb{N}.$$

Example 9.1. The name comes from the special case of geometric sequences (a^k), obtained with $Q = 1, P = a$. Other examples are the sequences $(k!)$, $(\binom{3k}{k})_k, (\binom{x}{k})_k, (\binom{k}{m})_k$ when $m \in \mathbb{N}$, since

$$(k+1)! = (k+1)k!, \quad \binom{3(k+1)}{k+1} = \frac{3(3k+1)(3k+2)}{2(2k+1)(k+1)}\binom{3k}{k}, \tag{9.5}$$

$$\binom{x}{k+1} = \frac{x-k}{k+1}\binom{x}{k}, \quad \binom{k+1}{m} = \frac{k+1}{k+1-m}\binom{k}{m}. \tag{9.6}$$

In the case of complex sequences, since polynomials can be fully factored, a complete characterisation of hypergeometric sequences is possible.

Lemma 9.1. *The sequence* $(u_k) \in \mathbb{C}^{\mathbb{N}}$ *is hypergeometric if and only if there exists* $N \in \mathbb{N}$, C, A, $a_1, \dots, a_p$, $b_1, \dots, b_q$ *in* $\mathbb{C}$ *such that*

$$u_{N+k} = C\,A^k\,\frac{\prod_{i=1}^{p}(a_i)(a_i+1)\cdots(a_i+k-1)}{\prod_{j=1}^{q}(b_j)(b_j+1)\cdots(b_j+k-1)}, \quad k \in \mathbb{N}.$$

Proof. First, if u_k is as given in the lemma, then

$$\frac{u_{N+k+1}}{u_{N+k}} = A\,\frac{\prod_{i=1}^{p}(a_i+k)}{\prod_{j=1}^{q}(b_j+k)}.$$

Since both numerator and denominator of the right-hand side are polynomials, this sequence is indeed hypergeometric.

Conversely, for two polynomials P and Q in $\mathbb{C}[k]$, the fraction P/Q can be written as in the right-hand side above, with $-a_i$ the roots of P, $-b_j$ the roots of Q and A the quotient of their leading coefficients. Let

$-N$ be a strict lower bound for the negative integer roots of Q and let $v_k = u_{N+k}$. Then

$$\frac{v_{k+1}}{v_k} = \frac{P(N+k)}{Q(N+k)}, \qquad k \in \mathbb{N},$$

since the denominator does not vanish in $\mathbb{N}$. Taking $\tilde{a}_i = a_i + N$, $\tilde{b}_j = b_j + N$ gives the result with $\tilde{a}_i$ and $\tilde{b}_j$ in place of a_i and b_j. □

9.1.3 Indefinite Hypergeometric Summation

The problem of indefinite hypergeometric summation is to decide whether a given hypergeometric sequence admits a hypergeometric indefinite sum, and if so, to compute this indefinite sum.

More formally, the input is formed of two polynomials P, Q in $\mathbb{K}[k]$ that define a sequence (u_k) as in Definition 9.2. The output is either a hypergeometric sequence (v_k) such that $v_{k+1} - v_k = u_k$, or a statement that no such sequence exists.

Example 9.2. For the sequence $u_k = 4^k / \binom{2k}{k}$ of Eq. (9.1), the input would be

$$P = 2k + 2, \quad Q = 2k + 1.$$

The output will be

$$v_k = \frac{2k-1}{3} u_k.$$

9.1.4 Reduction to Rational Sequences

If the desired indefinite hypergeometric sequence (v_k) exists, then there exists a rational function $S(k)$ such that $v_{k+1} = S(k)v_k$ for k larger than the largest integer root of the denominator of S, therefore

$$u_k = v_{k+1} - v_k = (S(k) - 1)v_k$$

implies that v_k is actually the product of u_k by a *rational* unknown function $T(k) = 1/(S(k) - 1)$.

Since (u_k) is hypergeometric, there also exists a rational function $R(k)$ such that $u_{k+1} = R(k)u_k$ for k sufficiently large. Injecting $v_k = T(k)u_k$ and this formula into $v_{k+1} - v_k = u_k$ gives

$$T(k+1)R(k) - T(k) = 1, \tag{$\mathcal{E}$}$$

an equation where R is known and T is the unknown.

Example 9.3. With $R = (2k + 2)/(2k + 1)$ from the previous example, this equation has for rational solution $T(k) = (2k - 1)/3$.

Example 9.4. With $R(k) = -(4k + 5)(2k - 1)/(2(4k + 1)(k + 2))$, the desired solution would be

$$T(k) = -2\frac{k+1}{4k+1}.$$

9.1.5 Gosper Normal Form

The resolution of Eq. ($\mathcal{E}$) is made difficult by the fact that roots of the numerator and denominator of R could differ by an integer. This difficulty is circumvented by a normal form of rational functions introduced by Gosper in 1978.

Lemma 9.2. *Any rational function $R(k) \in \mathbb{K}(k)$ can be written in the form*

$$R(k) = \frac{A(k)}{B(k)} \frac{C(k+1)}{C(k)},$$

with A, B, C polynomials in $\mathbb{K}[k]$ and the property that for all $m \in \mathbb{N}$, $\gcd(A(k), B(k+m)) = 1$.

Proof. The proof is effective. It is given by the following algorithm

```
# Gosper form of P/Q
GosperForm:=proc(P,Q,k)
local A,B,C,R,H,h,i,j,G;
        A:=P; B:=Q; C:=1;
        # 1. Integer differences between roots of P,Q
        R:=resultant(A,subs(k=k+d,B),k);
        H:=select(type,map2(op,1,roots(R,d)),nonnegint);
        # 2. Construct C and update A,B
        for h in sort(H) do
                G:=gcd(subs(k=k+h,B),A);
                A:=quo(A,G,k); B:=quo(B,subs(k=k-h,G),k);
                C:=C*mul(subs(k=k-j,G),j=1..h)
        end do;
        A,B,C
end:
```

First, the algorithm finds all the nonnegative integers $h \in \mathbb{N}$ such that $\gcd(P(k), Q(k+h)) \neq 1$, by computing the resultant between those polynomials and finding its integer roots. The idea is that for such a root h, writing $G(k)$ for this gcd and $P = G\tilde{P}$, $Q = G(k-h)\tilde{Q}$, one has

$$\frac{P(k)}{Q(k)} = \frac{\tilde{P}(k)}{\tilde{Q}(k)} \frac{G(k) \cdots G(k-h+1)}{G(k-1) \cdots G(k-h)}.$$

These roots are then sorted by increasing order and the polynomial C is used to accumulate fractions as above.

Writing $h_1 < h_2 < \cdots < h_m$ the elements of H and G_i, A_i, C_i the polynomials constructed with $h = h_i$, the correctness of the algorithm follows from two properties that are maintained at the end of each iteration of the loop:

1. $\dfrac{A_i(k)}{B_i(k)} \dfrac{C_i(k+1)}{C_i(k)} = \dfrac{P(k)}{Q(k)}$;
2. $\gcd(A_i(k), B_i(k+\ell)) \neq 1, \ell \in \mathbb{N} \Rightarrow \ell \geq h_{i+1}$ (with the convention $h_{m+1} = \infty$.)

They both follow from a simple induction. □

The following extra property will be used later.

Exercise 9.1. At the end of the algorithm, $\gcd(A(k), C(k)) = 1$, $\gcd(B(k), C(k+1)) = 1$.

9.1.6 Reduction to a Polynomial Unknown

Armed with Gosper's Normal Form, we can now prove the main result of this section.

Theorem 9.1 (Gosper, 1978). *If $T(k) \in \mathbb{K}(k)$ is a rational solution of the recurrence Eq. ($\mathcal{E}$) and A, B, C are given by Lemma 9.2, then*

$$T(k) = \frac{B(k-1)}{C(k)} X(k),$$

with $X(k)$ a polynomial *solution of the recurrence*

$$A(k)X(k+1) - B(k-1)X(k) = C(k). \tag{$\mathcal{E}'$}$$

Proof. Injecting the given form of $T(k)$ into Eq. ($\mathcal{E}$) and reducing common denominators gives Eq. ($\mathcal{E}'$), but at that stage $X(k)$ is only known to be rational.

Assume now that $X(k) = P(k)/Q(k)$ with $\gcd(P, Q) = 1$. We are going to show that $Q = 1$. Injecting this expression for X in Eq. ($\mathcal{E}'$) and multiplying by $Q(k)Q(k+1)$ gives

$$A(k)P(k+1)Q(k) - B(k-1)P(k)Q(k+1) = C(k)Q(k)Q(k+1).$$

Now one observes that $Q(k)$ dividing two of the three terms, it has to divide the third one. As it is relatively prime with P, it follows that

$$Q(k) \mid B(k-1)Q(k+1).$$

Therefore $Q(k+1) | B(k)Q(k+2)$ and so on, so that for any $K \in \mathbb{N} \setminus \{0\}$,

$$Q(k) \mid B(k+K-2) \cdots B(k-1)\, Q(k+K).$$

Similarly, $Q(k+1)$ divides two of the terms, leading to

$$Q(k) \mid A(k-1) \cdots A(k-K)Q(k-K).$$

As Q is a polynomial, there exists K such that $\gcd(Q(k), Q(k+K)) = \gcd(Q(k), Q(k-K)) = 1$. Then

$$Q(k) \mid \gcd(B(k+K-2) \cdots B(k-1), A(k-1) \cdots A(k-K)) = 1,$$

where the last equality comes from the properties demanded by the Gosper normal form. □

9.1.7 Polynomial Solution

The problem of indefinite hypergeometric summation has thus been reduced to finding polynomial solutions of an equation of the form

$$A(k)X(k+1) - B(k-1)X(k) = C(k). \tag{$\mathcal{E}'$}$$

A simple strategy to solve such equations is to first find a bound on the possible degrees of its polynomial solutions, then use undeterminate coefficients, which results in a linear system for those coefficients by extracting the coefficients of powers of k in the equation.

In order to find a bound, assume that as $k \to \infty$,

$$X(k) \sim \gamma k^D, \;\; A(k) \sim \lambda k^d, \;\; A(k) - B(k-1) \sim \mu k^\delta,$$

where all quantities are known from the input, except for the degree D, and also the linear factor γ that is irrelevant. Rewriting the recurrence as

$$A(k)(X(k+1) - X(k)) + (A(k) - B(k-1))X(k) = C(k)$$

and exploiting the fact that $X(k+1) - X(k) \sim \gamma D k^{D-1}$, one obtains the following distinction of cases

$$\begin{cases} \text{If } d - 1 \neq \delta, & D \leq \begin{cases} \deg C - \max(\delta, d-1), & \text{if } \deg C \neq \delta, \\ 0 & \text{if } \deg C = \delta, \end{cases} \\ \text{otherwise,} & D \leq \max(\deg C - \delta, -\mu/\lambda), \end{cases}$$

where the bound $-\mu/\lambda$ only appears if it is an integer.

9.1.8 Gosper's Algorithm

The algorithm is obtained by putting together the results of the discussion so far.

```
# Indefinite sum of u(k)
# Input: P,Q s.t. u(k+1)/u(k) = P/Q
# Output: t s.t. t(k)u(k) is an indefinite sum of u(k)
#         or FAIL if none exists
Gosper:=proc(P,Q,k)
local v,a,b,c,eq,x,pol;
        a,b,c:=GosperForm(P,Q,k);
        eq:=a*x(k+1)-subs(k=k-1,b)*x(k)=c;
        pol:=LREtools[polysols](eq,x(k),{});
        if pol=NULL then FAIL
        else pol*subs(k=k-1,b)/c fi
end:
```

Example 9.5. If

$$u_k = \frac{\prod_{j=1}^{k-1} j^2}{\prod_{j=1}^{k}(j^2+1)},$$

one obtains

$$\frac{u_{k+1}}{u_k} = \frac{k^2}{(k+1)^2+1}$$

which is already in Gosper normal form. The equation to be solved becomes

$$k^2 X(k+1) - (k^2+1)X(k) = 1.$$

With the notation of our discussion on degree bounds, we have $d = 2, \delta = 0, \deg C = 0$. It follows that a bound on the degree of solutions is $D \leq 0$. Setting $X(k) = \lambda$ and extracting the coefficients of $1, k, k^2$ gives a system of one equation:

$$-\lambda = 1.$$

An indefinite hypergeometric sum has thus been found:

$$\sum_k u_k = -(k^2+1)u_k.$$

Since the denominator of u_k does not vanish on $\mathbb{N}$, one deduces for instance that

$$\sum_{k=1}^{n} \frac{\prod_{j=1}^{k-1} j^2}{\prod_{j=1}^{k}(j^2+1)} = 2u_1 - (n^2+1)u_{n+1} = 1 - \prod_{j=1}^{n} \frac{j^2}{j^2+1}.$$

As Gosper's algorithm is a decision algorithm, it also allows to determine when sequences do not admit a hypergeometric indefinite sum.

Example 9.6. With $u_k = k!$, the equation to be solved becomes

$$(k+1)X(k+1) - X(k) = 1$$

and since the degree of the first term is always one more than the degree of X, this equation cannot have a polynomial solution, proving that the sequence $(k!)$ does not have an hypergeometric indefinite sum.

Similarly, with $u_k = \binom{n}{k}$, the equation becomes

$$(n-k)X(k+1) - kX(k) = 1,$$

the quantities for the degree bounds are $d = 1, \delta = 0, \deg C = 0, -\mu/\lambda = n \notin \mathbb{N}$, leading to the bound $D \leq 0$ and the system

$$-2\lambda = 0, \lambda n - 1 = 0,$$

that does not have any solution.

9.2 Zeilberger's Algorithm for Definite Summation

As shown by the last example, the sequence $(\binom{n}{k})$ with n fixed does not admit an indefinite hypergeometric sum. However, it has a very simple *definite* sum

$$\sum_{k=0}^{n} \binom{n}{k} = (1+1)^n = 2^n.$$

Other similar examples of sequences that do not have an indefinite hypergeometric sum but a nice definite one are

$$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n}, \qquad \sum_{k=0}^{n} \binom{2n-2k}{n-k}\binom{2k}{k} = 4^n$$

and a larger one, due to Dixon around 1900,

$$\sum_{k\in\mathbb{Z}} (-1)^k \binom{a+b}{a+k}\binom{b+c}{b+k}\binom{c+a}{c+k} = \frac{(a+b+c)!}{a!b!c!}, \quad a,b,c \in \mathbb{N},$$

where the sum indicated over $\mathbb{Z}$ reduces to $\max(-a,-b,-c) \leq k \leq \min(a,b,c)$, as at least one of the binomial coefficients is 0 outside of that range.

9.2.1 Definitions

We first extend the definition of hypergeometric sequences to a bivariate setting.

Definition 9.3. *The sequence* $(f_{n,k}) \in \mathbb{K}^{\mathbb{N}\times\mathbb{N}}$ *is* hypergeometric *if there are four non-zero polynomials in* $\mathbb{K}[n,k]$ *such that*

$$B(n,k)f_{n,k+1} = A(n,k)f_{n,k}, \quad D(n,k)f_{n+1,k} = C(n,k)f_{n,k}.$$

Example 9.7. The binomial coefficient sequence $\binom{n}{k}$ is hypergeometric: it satisfies

$$\binom{n+1}{k} = \frac{n+1}{n+1-k}\binom{n}{k}, \quad \binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}.$$

The problem of *definite hypergeometric summation* is, given two rational functions as above that define a hypergeometric sequence $f_{n,k}$, to find a linear recurrence with polynomial coefficients for the sequence

$$F_n = \sum_{k\in\mathbb{Z}} f_{n,k}.$$

Actually, a more useful result is obtained by Zeilberger's algorithm described below.

9.2.2 Creative Telescoping

If one knows Pascal's triangle

$$\binom{n+1}{k+1} = \binom{n}{k+1} + \binom{n}{k},$$

then subtracting this identity in $\sum \binom{n+1}{k}$ leads to a telescoping:

$$F_{n+1} = \sum_k \binom{n+1}{k} = \sum_k \left(\binom{n+1}{k} - \binom{n+1}{k+1} + \binom{n}{k+1} - \binom{n}{k} + 2\binom{n}{k} \right) = 2F_n.$$

Writing $f_{n,k}$ for $\binom{n}{k}$, this process has produced the identity

$$f_{n+1,k} - 2f_{n,k} = (f_{n,k+1} - f_{n+1,k+1}) - (f_{n,k} - f_{n+1,k}),$$

from which the result $F_{n+1} = 2F_n$ is obtained by summing over k. The method of *creative telescoping* generalizes this example.

Definition 9.4. *A* telescoping identity *for a bivariate hypergeometric sequence $f_{n,k}$ is an equation of the form*

$$t_p(n) f_{n+p,k} + \cdots + t_0(n) f_{n,k} = g_{n,k+1} - g_{n,k}$$

with rational t_i in $\mathbb{K}(n)$, not all 0. The left-hand side is called a telescoper *and the right-hand side a* certificate.

The reason for the name certificate is that given a certificate $g_{n,k}$, one can check the identity obtained by summing over k, which is not straightforward without it.

9.2.3 Zeilberger's Algorithm

The equation of Definition 9.4 expresses $g_{n,k}$ as an *indefinite* sum of the left-hand side. The principle of Zeilberger's algorithm is to use Gosper's algorithm with unknown t_i in order to find $g_{n,k}$. Let

$$U_k = t_p(n) f_{n+p,k} + \cdots + t_0(n) f_{n,k},$$

where we focus on k, but U_k also depends on n, and on the t_i. Then

$$\frac{U_{k+1}}{U_k} = \frac{t_p(n)\frac{f_{n+p,k+1}}{f_{n,k+1}} + \cdots + t_0(n)\frac{f_{n,k+1}}{f_{n,k+1}}}{t_p(n)\frac{f_{n+p,k}}{f_{n,k}} + \cdots + t_0(n)\frac{f_{n,k}}{f_{n,k}}} \frac{f_{n,k+1}}{f_{n,k}} =: \frac{P(k+1,t)/q(k+1)}{P(k,t)/q(k)} \frac{f_{n,k+1}}{f_{n,k}},$$

where $P(k,t)$ is a linear combination of the t_i with coefficients in $\mathbb{K}(n)[k]$ and $q \in \mathbb{K}(n)[k]$ exist since $f_{n,k}$ is hypergeometric.

Next, consider the Gosper normal form of $q(k) f_{n,k+1}/(q(k+1) f_{n,k})$ *in the field* $\mathbb{K}'(k)$, with $\mathbb{K}' = \mathbb{K}(n)$,

$$\frac{q(k) f_{n,k+1}}{q(k+1) f_{n,k}} = \frac{A(k)}{B(k)} \frac{C(k+1)}{C(k)}.$$

It follows that

$$\frac{U_{k+1}}{U_k} = \frac{A(k)}{B(k)} \frac{P(k+1,t) C(k+1)}{P(k,t) C(k)}.$$

This leads to the equation

$$(\mathcal{E}') \qquad A(k)X(k+1) - B(k-1)X(k) = C(k)P(k,t),$$

where the t_i appear only in the right-hand side. The question is to find values of t_i such that this equation has a *polynomial* solution $X(k)$ in $\mathbb{K}(n)[k]$.

Input: $f_{n+1,k}/f_{n,k}$, $f_{n,k+1}/f_{n,k}$ in $\mathbb{K}(n,k)$; $m \in \mathbb{N}$
Output: $(t_i(n))_{0\le i\le p}$ in $\mathbb{K}(n)$ ($p \le m$) and $R(n,k) \in \mathbb{K}(n,k)$ s.t.
$t_p(n)f_{n+p,k} + \cdots + t_0(n)f_{n,k} = R(n,k+1)f_{n,k+1} - R(n,k)f_{n,k}$, or $\nexists$.

1. For $p = 0,\dots,m$
   a) Set $t_p\dfrac{f_{n+p,k+1}}{f_{n,k+1}} + \cdots + t_0 = \dfrac{P(k,t)}{q(k)}$, $\quad P \in \mathbb{K}(n)[k,t], q \in \mathbb{K}(n)[k]$
   b) Compute the Gosper form
   $$\frac{q(k)}{q(k+1)}\frac{f_{n,k+1}}{f_{n,k}} = \frac{A(k)C(k+1)}{B(k)C(k)}, \quad A,B,C \in \mathbb{K}(n)[k]$$
   c) In the equation $A(k)X(k+1) - B(k-1)X(k) = C(k)P(k)$,
      i. Find a bound D on the degree of $X \in \mathbb{K}(n)[k]$
      ii. Try and solve the linear system in the coefficients of X and $t_0, \dots, t_p$
      iii. If this has a nonzero solution, return $(t_0, \dots, t_p)$ (telescoper)
      and $B(k-1)X(k)/(P(k)C(k))$ (certificate)
2. Return $\nexists$

Figure 9.1: Zeilberger's Algorithm

Going back to the discussion on degree bounds in Section 9.1.7, one observes that these bounds depend only on A, B and $\deg C + \deg P$, none of which depends on the unknown t_i. Thus one can compute a degree bound as before. And then, extracting coefficients of powers of k gives a system that is linear in the coefficients of X *and also in the* t_i. This is Zeilberger's algorithm, presented in Fig. 9.1. The algorithm loops over increasing orders for the left-hand side. In general, there is no guarantee that any telescoping identity will be found, which is why it also takes a bound m on the orders to be tried. There is an important class of *proper hypergeometric* sequence for which it is proved to terminate, but we do not discuss it here.

9.2.4 Example

We show the steps of the algorithm on the sum

$$S_n = \sum_{k=0}^{n} \frac{(n+a+b+c+k)!}{(n-k)!(a-k)!(b+k)!(c+k)!k!}.$$

If

$$f_{n,k} = \frac{(n+a+b+c+k)!}{(n-k)!(a-k)!(b+k)!(c+k)!k!},$$

the input of the algorithm is formed of the two rational functions

$$\frac{f_{n+1,k}}{f_{n,k}} = \frac{n+a+b+c+k+1}{n-k}, \quad \frac{f_{n,k+1}}{f_{n,k}} = \frac{(n+a+b+c+k+1)(n-k)(a-k)}{(b+k+1)(c+k+1)(k+1)} \tag{9.7}$$

in $\mathbb{Q}(a,b,c,n,k)$.

Having no *a priori* bound on the order of a telescoper for $f_{n,k}$, we start by looking for a telescoper of order $p = 1$. With $U_k = t_1 f_{n+1,k} + t_0 f_{n,k}$, we find

$$\frac{U_{k+1}}{U_k} = \frac{\frac{P(k+1,t)}{n-k}}{\frac{P(k,t)}{n+1-k}}\frac{f_{n,k+1}}{f_{n,k}},$$

where

$$P(k,t) = (n+k+1+a+b+c)t_1(n) + (n+1-k)t_0(n).$$

Next, we observe that

$$\frac{(n+a+b+c+k+1)(n+1-k)(a-k)}{(b+k+1)(c+k+1)(k+1)}$$

is already in Gosper normal form, since no root of the numerator differs from a root of the denominator by an integer.

Thus the linear recurrence for the polynomial X is

$$(n+1-k)(a-k)(a+b+c+n+k+1)X(k+1) - k(b+k)(c+k)X(k) = P(k,t).$$

With the notation of the discussion on degree bounds in Section 5.4.3, we have

$$d = 3, \quad \delta = 1, \quad \deg C = 1,$$

from where the bound $D \leq 0$ follows. Setting $X(k) = \lambda$, the coefficients of k^3 and k^2 vanish and extracting the coefficients of k^0, k^1 gives the system

$$\begin{aligned}\{&-a\,(-n-1)\,(n+1+a+b+c)\,\lambda - t_0\,(-n-1) - (-n-1-a-b-c)\,t_1,\\ &\qquad - (a\,(-n-1) + (a+n+1)\,(n+1+a+b+c))\,\lambda - bc\lambda - t_0 + t_1\}\,,\end{aligned}$$

linear in λ, t_0, t_1. It has a nonzero solution

$$\begin{aligned} t_0 &= -\,(a+b+n+1)\,(a+c+n+1)\,(a+b+c+n+1)\,,\\ t_1 &= (n+1)\,(b+n+1)\,(c+n+1)\,,\\ \lambda &= -(a+b+c+2n+2).\end{aligned}$$

This means that with these values,

$$t_1(n) f_{n+1,k} + t_0(n) f_{n,k} = \Delta_k \left(\frac{B(k-1)x(k)}{P(k)}\, U_k \right),$$

where Δ_k is the difference operator: $\Delta_k(v_k) = (v_{k+1} - v_k)$.

One checks that the denominator does not vanish for $k \in \{-1, 0, \ldots, n+1\}$. Thus, summing this identity for k from -1 to n gives

$$\frac{\lambda B(n)}{P(n+1)} U_{n+1} - \lambda \frac{B(-2)}{P(-1)} U_{-1}.$$

Normalizing Eq. (9.7) and evaluating at $k = n$ shows both $f_{n,n} = 0$ and $f_{n,n+1} = 0$. Therefore $U_{n+1} = t_1(n) f_{n+1,n+1} + t_0(n) f_{n,n+1} = 0$. Similarly, $U_{-1} = 0$ is obtained by evaluating the second equation of Eq. (9.7) at $k = -1$. This shows that S_n satisfies the linear recurrence of order 1

$$(n+1)\,(b+n+1)\,(c+n+1)\,S_{n+1} - (a+b+n+1)\,(a+c+n+1)\,(a+b+c+n+1)\,S_n = 0.$$

In view of the initial condition

$$S_0 = \frac{(a+b+c)!}{a!b!c!},$$

we have found

$$S_n = \sum_{k=0}^{n} \frac{(n+a+b+c+k)!}{(n-k)!(a-k)!(b+k)!(c+k)!k!} = \frac{(a+b+n)!(a+c+n)!(a+b+c+n)!}{n!a!(a+b)!(a+c)!(b+n)!(c+n)!}.$$

9.3 Petkovšek's Algorithm

The last algorithm of this lecture aims at finding hypergeometric solutions of linear recurrences. One source of such recurrences is Zeilberger's algorithm from the previous section. For instance, with

$$D_n = \sum_{k=0}^{n} (-1)^k \binom{n}{k} \binom{3k}{n},$$

Zeilberger's algorithm finds the linear recurrence

$$2(2n+3)D_{n+2} + 3(5n+7)D_{n+1} + 9(n+1)D_n = 0.$$

Petkovšek's algorithm starts from there and finds that D_n is indeed hypergeometric (see below).

Similarly, for the sequence

$$A_n = \sum_{k=0}^{n} \binom{n}{k}^2 \binom{n+k}{k}^2$$

from Apéry's proof, Zeilberger's algorithm returns the recurrence

$$(n+1)^3 A_{n+1} - (34n^3 + 51n^2 + 27n + 5)A_n + n^3 A_{n-1} = 0,$$

from which Petkovšek's algorithm shows that A_n is *not* hypergeometric.

9.3.1 Petkovšek's Algorithm

The given recurrence is

$$p_r(n)u_{n+r} + \cdots + p_0(n)u_n = 0$$

and u_n is assumed to be hypergeometric. The analysis is simpler if one takes a stronger form of the Gosper normal form of u_{n+1}/u_n:

$$\frac{u_{n+1}}{u_n} = Z\frac{A(n)}{B(n)}\frac{C(n+1)}{C(n)},$$

where it is assumed that A, B, C satisfy the same property as earlier, and moreover are all monic. This is easy to achieve for any rational function P/Q: take Z to be the quotient of the leading coefficients of P and Q, then run the algorithm on $P/\operatorname{lc}(P)$ and $Q/\operatorname{lc}(Q)$ (lc denotes the leading coefficient) paying attention to take monic gcds.

Injecting this expression in the recurrence and multiplying by common factors gives

$$\begin{aligned} &Z^r p_r(n)A(n+r-1)\cdots A(n)C(n+r)+ \\ &\quad Z^{r-1}p_{r-1}(n)B(n+r-1)A(n+r-2)\cdots A(n)C(n+r-1) \\ &\quad\quad +\cdots+ \\ &\quad\quad Zp_1(n)B(n+r-1)\cdots B(n+1)A(n)C(n+1)+ \\ &\quad\quad\quad p_0(n)B(n+r-1)\cdots B(n)C(n) = 0. \end{aligned} \tag{9.8}$$

If A and B are known, this equation first gives Z by extracting the leading coefficient of the recurrence. Next, C has to be a nonzero polynomial solution of the recurrence, which is searched for as before, by bounding the degree and then using undeterminate coefficients.

The properties of the Gosper form (including Exercise 9.1) prove divisibilities of A and B:

$$A(n) \mid p_0(n), \quad B(n+r-1) \mid p_r(n).$$

Thus A and B are factors of the extremal coefficients. The resulting algorithm is given in Fig. 9.2.

9.3.2 Examples

Example 9.8. With the recurrence

$$2(2n+3)D_{n+2} + 3(5n+7)D_{n+1} + 9(n+1)D_n = 0, \tag{9.9}$$

the monic factors of the extremal coefficients are

$$\{1, n+3/2\}, \quad \{1, n+1\}.$$

Input: $p_r(n)u_{n+r} + \cdots + p_0(n)u_n = 0$ with $p_i \in \mathbb{K}[n]$
Output: A hypergeometric solution if one exists; 0 otherwise
1. For all monic factors $a(n)$ of $p_0(n)$, $b(n)$ of $p_r(n-r+1)$,
   (a) Set $P_i = p_i(n)a(n)\cdots a(n+i-1)b(n+i)\cdots b(n+r-1)$ for $i = 0,\ldots,r$;
   (b) Let $M := \max \deg P_i$; $\alpha_i = [n^M]P_i$; $Q = \sum_i \alpha_i Z^i$;
   (c) For all $z \in \mathbb{K}$ s.t. $Q(z) = 0$,
       If $z^r P_r(n)c(n+r) + \cdots + P_0(n)c(n) = 0$
       has a nonzero polynomial solution $c(n) \in \mathbb{K}[n]$,
       return $u_{n+1}/u_n = za(n)c(n+1)/(b(n)c(n))$.
2. Return 0

Figure 9.2: Petkovšek's Algorithm

The algorithm loops over the 4 choices. The first one is, say, $A = B = 1$. Then Eq. (9.8) becomes

$$2(2n+3)Z^2C(n+2) + 3(5n+7)ZC(n+1) + 9(n+1)C(n) = 0.$$

Extracting the coefficient of n in each coefficient gives

$$4Z^2 + 15Z + 9 = (Z+3)(4Z+3).$$

The next loop is over the roots $z = -3, z = -3/4$ of this polynomial. Taking the first choice ($z = -3$) gives the recurrence

$$18(2n+3)c(n+2) - 9(5n+7)c(n+1) + 9(n+1)c(n) = 0,$$

which can be rewritten in the form

$$18(2n+3)(c(n+2) - 2c(n+1) + c(n)) + 9(3n+5)(c(n+1) - c(n)) = 0,$$

where the first term has degree $\deg c - 1$ when $\deg c \geq 2$ and the second one $\deg c$ when $\deg c \geq 1$. It follows that the only possibility for a polynomial solution is $\deg c = 0$. Indeed, $c(n) = 1$ is clearly a solution, which shows that $(-3)^n$ is a solution of Eq. (9.9). Since the initial conditions $D_0 = 1$ and $D_1 = -3$ match, this proves the identity

$$D_n = \sum_{k=0}^{n} (-1)^k \binom{n}{k}\binom{3k}{n} = (-3)^n.$$

Example 9.9. The case of Apéry's sequence is more complicated. The recurrence

$$(n+1)^3 A_{n+1} - (34n^3 + 51n^2 + 27n + 5)A_n + n^3 A_{n-1} = 0$$

leads to considering all the possible monic factors among

$$\{1, n+1, (n+1)^2, (n+1)^3\}, \{1, n, n^2, n^3\},$$

which gives 16 choices. For each of them, one constructs Eq. (9.8) and for each of the solutions of the equation for Z, one searches for a polynomial solution. It is better to have a computer algebra system do it automatically. In Maple, this is achieved by the command

```
> LREtools[hypergeomsols](rec,A(n),{});
```

0

showing that this recurrence does not admit any hypergeometric solution and therefore that the sequence (A_n) is not hypergeometric.

Additional Bibliography

The most elementary introduction is by the authors of these algorithms themselves

Marko Petkovšek, Herbert S. Wilf, and Doron Zeilberger. $A = B$. Wellesley, MA: A. K. Peters, 1996, pp. xii+212

Another clear introduction to Zeilberger's algorithm is in the recent editions of

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete mathematics*. 2nd edition. Reading, MA: Addison-Wesley Publishing Company, 1994, pp. xiv+657

Lecture 10

Gröbner Bases

Summary

Gröbner bases are bases of polynomial ideals that play a fundamental role in the manipulation of polynomial systems. They allow to test ideal membership, to solve elimination problems and to test membership in the radical of a polynomial ideal. This lets one answer geometric questions, thanks to Hilbert's Nullstellensatz.

In this lecture, $\mathbb{A} = \mathbb{K}[x_1, \ldots, x_n]$, with $\mathbb{K}$ an arbitrary field.

Polynomial systems can be used to encode a variety of problems, from geometry (where circles and lines are encoded by polynomial equations), to robotics (the possible movements of each joint of a robot being translations or rotations are also encoded by polynomials), or graph coloring (colors are values, each vertex is given a variable). For all these problems, Gröbner bases can help answer questions related to the encoded situation.

10.1 Questions about Polynomial Systems

10.1.1 Existence of Solutions

A first question is the existence and the number of solutions. In some cases, like the system

$$\{x + y = 0,\ x + y + 1 = 0\},$$

the answer is that there is no solution, whatever the situation. In other cases, like

$$x^2 + y^2 + 1 = 0,$$

the answer depends on the field $\mathbb{K}$ where solutions are considered. This equation does not have any solution in $\mathbb{R}^2$ but it does have solutions in $\mathbb{C}^2$. This latter situation is the one to which Gröbner bases are more suited. There are also ways to answer this type of problems over the real numbers for systems using Gröbner bases in intermediate computations, but they will not be discussed here.

Another situation where the existence of solutions is important is in SAT-solving. For instance, a boolean formula like

$$(x_1 \vee \overline{x}_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x}_4) \wedge \cdots$$

translates into the polynomial system

$$\{x_1(1 - x_1) = 0,\ x_2(1 - x_2) = 0,\ x_3(1 - x_3) = 0, \ldots, x_1(1 - x_3)x_4 = 0,\ x_2x_3(1 - x_4) = 0, \ldots\}$$

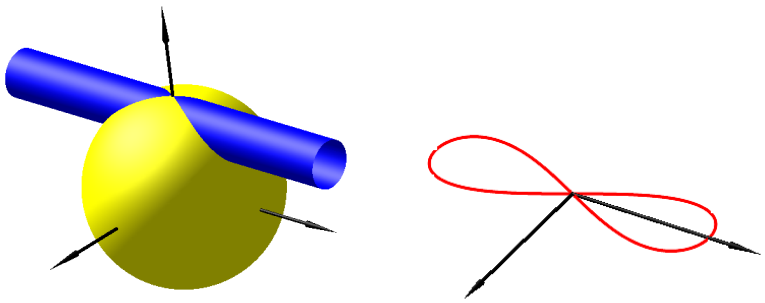


Figure 10.1: Intersection of a sphere and a cylinder and its projection

Since SAT-solving is known to be an NP-complete problem, this shows that answering this type of questions for polynomial systems is not going to be easy computationally. Indeed, computational complexity turns out to be a major obstacle in the efficient solution of questions related to polynomial systems and a major source of frustration for users of Gröbner bases.

10.1.2 Elimination

There are many applications of elimination. As mentioned in Lecture 5, elimination corresponds to projection. For instance, the projection of the intersection of a sphere and a cylinder displayed in Fig. 10.1 has an equation given by eliminating z between the equations

$$\begin{cases} x^2+y^2+z^2=1, \\ y^2+(z-3/4)^2=1/16 \end{cases}$$

which gives

$$4x^4-3x^2+9y^2=0.$$

Another application is *implicitation*, where parameters are eliminated from a parameterization in order to obtain an implicit equation for a curve or a surface. The most basic example is to go from the parameterization of a circle by the tangent of the half-angle:

$$\begin{cases} x=\frac{1-t^2}{1+t^2}, \\ y=\frac{2t}{1+t^2} \end{cases}$$

to the equation $x^2+y^2-1=0$ by eliminating t.

Again, Gröbner bases can perform those computations.

10.1.3 Solutions

"Solving" a polynomial system can take many forms. Usually, one aims at finding a representation of the set of solutions on which further computations can be performed easily. For instance, the system

$$\begin{cases} x^2+y+z=1, \\ x+y^2+z=1, \\ x+y+z^2=1, \end{cases} \tag{10.1}$$

has for solutions the intersections of the three surfaces displayed in Fig. 10.2. The system itself is obviously a

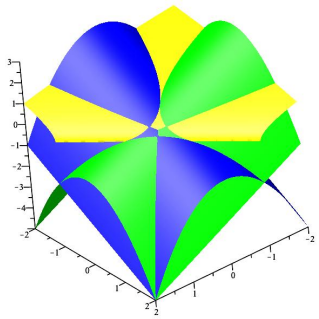


Figure 10.2: Solutions of Eq. (10.1)

representation of its solutions. Other convenient representations can be computed with the help of Gröbner bases, such as a *triangular system*:

$$\begin{cases} x + y + z^2 = 1, \\ y^2 - y - z^2 + z = 0, \\ z^2(2y + z^2 - 1) = 0, \\ z^2(z-1)^2(z^2 + 2z - 1) = 0, \end{cases}$$

where the last equation has only one variable z and for each of its solutions, the two previous ones let one find corresponding values for y and from there the first one gives x. Moreover, this representation keeps track of the multiplicity of the solutions (a notion which we do not define precisely here).

Another useful representation is called a *rational parameterization*:

$$\begin{cases} p(u) & = (u-1)(u-2)(u-4)(u^2+14u-49) = 0, \\ x & = \frac{-u^4+36u^3-251u^2+618u-504}{p'(u)}, \\ y & = \frac{-u^4+37u^3-241u^2+513u-308}{p'(u)}, \\ z & = \frac{-u^4+39u^3-215u^2+387u-210}{p'(u)}. \end{cases}$$

It consists of a square-free polynomial in one variable $p(u)$ (square-free means that it has only simple roots) and a set of rational functions that give a solution point for each root of p. The division by $p'(u)$ might seem unnecessary, since p' can be inverted modulo p, but keeping it leads to parameterizations with smaller integer coefficients (not proved here either). Another interest of this representation is that there exist algorithms computing it more efficiently than going through a Gröbner basis computation, at least in theory.

10.1.4 Ideals

Given a polynomial system, its solutions are also solutions of all linear combinations of its equations with polynomial coefficients. These form an ideal (the definition of an ideal is recalled in Appendix A.)

Definition 10.1. *The* ideal generated by $(f_1, \ldots, f_s) \in \mathbb{A}^s$ *is*

$$\langle f_1, \ldots, f_s \rangle = \left\{ \sum_{i=1}^{s} U_i f_i \mid U_i \in \mathbb{A} \right\}.$$

That it is an ideal is readily checked.

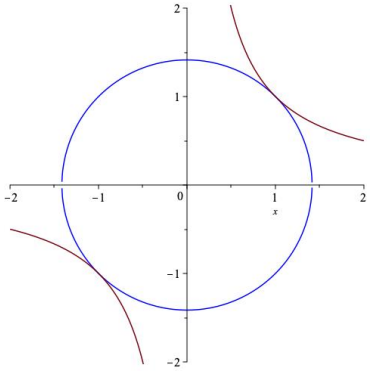


Figure 10.3: Intersection of a circle and a hyperbola

Example 10.1. In one variable ($n = 1, \mathbb{A} = \mathbb{K}[x]$), ideals of $\mathbb{K}[x]$ are principal (generated by one element) and

$$\langle f_1, \ldots, f_s \rangle = \langle \gcd(f_1, \ldots, f_s) \rangle.$$

10.1.5 Ideal Membership

An ideal can be given by generators in many ways. Thus the intersection of a circle and a hyperbola in Fig. 10.3 can be described by the ideal

$$\mathcal{I} = \langle x^2 + y^2 - 2, xy - 1 \rangle.$$

In the vocabulary of the next section, a Gröbner basis of $\mathcal{I}$ (for the graded reverse lexicographic order) is found to be

$$\mathcal{I} = \langle x^2 + y^2 - 2, xy - 1, x - 2y + y^3 \rangle.$$

It contains the original two polynomials, plus another one that can be obtained as $y(x^2 + y^2 - 2) - x(xy - 1)$ and has the advantage of having x as its only monomial in the variable x. Another basis, for the lexicographic order, is the triangular system

$$\mathcal{I} = \langle y^4 - 2y^2 + 1, x - 2y + y^3 \rangle.$$

That this is the same ideal is not completely obvious at first sight. The first polynomial, in y only, can be found as $y(x - 2y + y^3) - (xy - 1)$, showing the inclusion of this last ideal in the previous one. For the other direction, one can first observe that

$$xy - 1 = y(x - 2y + y^3) - (y^4 - 2y^2 + 1)$$

and next, that

$$x^2 + y^2 - 2 = x(x - 2y + y^3) - (y^2 - 2)(xy - 1).$$

These computations are special cases of the *ideal membership problem*. Given a polynomial f and the generators $f_1, \ldots, f_s$ of an ideal, the question of ideal membership is to test whether $f \in \langle f_1, \ldots, f_s \rangle$.

For instance, in the previous example, where the solutions all satisfy $y^2 - 1 = 0$, it is natural to wonder whether $y^2 - 1 \in \mathcal{I}$. Again, Gröbner bases can be used to answer this question.

An important special case is to test whether $1 \in \langle f_1, \ldots, f_s \rangle$, i.e., detect that the equations are inconsistent.

10.2 Gröbner Bases

Gröbner bases can be viewed as a common generalization of Euclidean division and of Gaussian elimination. In Euclidean division, one takes two polynomials, compares their highest degree terms, multiplies by an appropriate monomial that allows to perform a subtraction that reduces the degree of the higher degree polynomial. One difficulty in several variables is that there can be several "highest degree terms" and choices have to be made. This is the purpose of monomial orders.

10.2.1 Monomial Orders

Recall that $\mathbb{A} = \mathbb{K}[x_1, \ldots, x_n]$. We use the notation

$$x^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n}, \quad (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n.$$

Definition 10.2. *A* monomial *is an element of* $\mathbb{A}$ *of the form* x^α, $\alpha \in \mathbb{N}^n$. *A* term *is an element of* $\mathbb{A}$ *of the form* λx^α *with* $\lambda \neq 0 \in \mathbb{K}$ *and* $\alpha \in \mathbb{N}^n$.

Definition 10.3. *A* monomial order *is a total order on the monomials, that is compatible with multiplication* ($x^\alpha \prec x^\beta \Rightarrow x^\alpha x^\gamma \prec x^\beta x^\gamma$) *and such that every nonempty set of monomials has a smallest element.*

The following consequence of the definition plays an important role later.

Lemma 10.1. $1 \preceq x^\alpha$, $\alpha \in \mathbb{N}^n$.

Proof. By contradiction. If $1 \succ x^\alpha$, then by compatibility with multiplication, the set $\{x^{k\alpha} \mid k \in \mathbb{N}\}$ does not have a smallest element. □

Example 10.2. In one variable, since $1 \prec x$, compatibility with multiplication implies that

$$m \geq k \Leftrightarrow x^m \succeq x^k.$$

The case when $n > 1$ is much more diverse.

10.2.2 Examples of Monomial Orders

Definition 10.4. *In the* lexicographic order, $x^\alpha \preceq x^\beta$ *when* $\alpha = \beta$ *or the first nonzero entry of* $(\alpha - \beta) \in \mathbb{Z}^n$ *is positive.*

In Maple, this order is denoted 'plex' for *pure lexicographic order*.

Other orders can be defined with the help of matrices that indicate how ties are broken. Important matrices in practice are

$$M_{\text{grlex}} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & & & \\ & \ddots & & \\ & & 1 & \end{pmatrix}, \quad M_{\text{grevlex}} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ & & & -1 \\ & & \iddots & \\ & -1 & & \end{pmatrix}, \quad M_{\text{elim}}^{(i)} = \begin{pmatrix} \overbrace{1 \cdots 1}^{i} & 0 \cdots 0 \\ 0 \cdots 0 & 1 \cdots 1 \\ & & -1 \\ & \iddots & \\ -1 & & \end{pmatrix}.$$

Definition 10.5. *For $M \in \{\mathrm{Id}, M_{\mathrm{grlex}}, M_{\mathrm{grevlex}}, M^{(i)}_{\mathrm{elim}}\}$, one defines a monomial order by $x^\alpha \preceq x^\beta$ when $\alpha = \beta$ or the first nonzero entry of $M \cdot (\alpha - \beta)$ is positive. The corresponding monomial orders are called:*

- lexicographic order *if* $M = \mathrm{Id}$*;*
- graded lexicographic order *if* $M = M_{\mathrm{grevlex}}$*;*
- graded reverse lexicographic order *if* $M = M_{\mathrm{grevlex}}$ *('tdeg' in Maple, for* total degree order*);*
- ith elimination order *if* $M = M^{(i)}_{\mathrm{elim}}$.

It is readily checked that the first definition is equivalent to the one above.

The intuition for practical use of these monomial orders is that computations with the graded reverse lexicographic order (tdeg) tend to be faster, computations with the lexicographic order (plex) tend to be slower but give more direct information, the elimination orders are in between.

10.2.3 Gröbner Bases

Only a few more definitions stand between us and the definition of Gröbner bases.

Definition 10.6. *Given a monomial order $\preceq$ and a polynomial $P \neq 0$, one calls* leading monomial *of P (denoted $\mathrm{LM}(P)$) the largest monomial for $\preceq$; one calls* leading term *(denoted $\mathrm{LT}(P)$) the corresponding term and* leading coefficient *($\mathrm{LC}(P)$) the corresponding coefficient.*

Definition 10.7. *The* stairs *of an ideal $\mathcal{I} \subset \mathbb{A}$ is the set $\{\mathrm{LM}(P) \mid P \in \mathcal{I} \setminus \{0\}\}$.*

The stairs are visualized by displaying the exponents, and typically look as in Fig. 10.4. By compatibility of the monomial order with multiplication, for each point p of this picture, the region $p + \mathbb{N}^n$ belongs to it too.

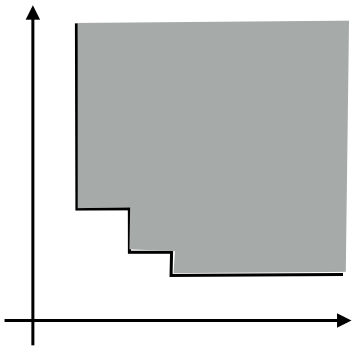


Figure 10.4: Stairs of an ideal

And finally,

Definition 10.8. *A finite subset G of an ideal $\mathcal{I} \subset \mathbb{A}$ is a* Gröbner basis *of $\mathcal{I}$ if $\langle \mathrm{LM}(G) \rangle = \langle \mathrm{LM}(\mathcal{I}) \rangle$.*

The proof that Gröbner bases exist for all ideals of $\mathbb{A}$ is postponed to the next section.

In the example of the picture of the stairs above, the basis would contain one polynomial for each of the three corners of they gray area.

An easy result at this stage is a characterization of trivial ideals.

Corollary 10.1. *The ideal $\mathcal{I}$ is trivial ($\mathcal{I} = \mathbb{A}$) if and only if $1 \in G$, where G is a Gröbner basis of $\mathcal{I}$ for any monomial order.*

Proof. Note that 1 is the only polynomial with leading monomial 1, up to nonzero constant factors. It follows that if $1 \in \mathcal{I}$, then $1 \in G$. Conversely, if $1 \in G$, there is a polynomial with leading monomial 1 in $\mathcal{I}$, and this implies $1 \in \mathcal{I}$. □

10.2.4 Examples

Intersection Circle-Hyperbola

In Section 10.1.5, we gave the Gröbner bases for the ideal $\mathcal{I} = \{x^2 + y^2 - 2, xy - 1\}$, for the graded reverse lexicographic order and for the lexicographic order. Underlining their leading monomials, they are

$$\{\underline{x^2} + y^2 - 2,\ \underline{xy} - 1,\ x - 2y + \underline{y^3}\}, \qquad \{\underline{y^4} - 2y^2 + 1,\ \underline{x} - 2y + y^3\}. \tag{10.2}$$

Note in particular that the number of elements in the basis depends on the monomial order. The corresponding stairs are given in Fig. 10.5.

Figure 10.5: Stairs of the ideal of Eq. (10.2)

The second one is typical of a favorable situation (called 'shape lemma'), where one polynomial involves only one variable; the other ones each involve that variable and another one with respect to which it is linear. Thus, the solution set is completely parameterized by the solutions of the univariate polynomial.

Univariate Situation

If $n = 1$ (univariate case), then recall from Example 10.1 that any ideal is generated by the gcd g of its elements. Then $\langle \mathrm{LM}(g) \rangle = \langle x^{\deg g} \rangle$ and this is clearly equal to $\langle \mathrm{LM}(\mathcal{I}) \rangle$, since all elements of $\mathcal{I}$ have degree $\geq \deg g$. Thus, $\{g\}$ is a Gröbner basis of the ideal.

Linear Systems

If the degree of all polynomials is 1, then we are dealing with a linear system of equations. Assume that A is a matrix in row echelon form (the first nonzero entry of each row is to the right of the first nonzero entry of the row above it), as depicted here:

$$\begin{pmatrix} 0 & \blacksquare & \star & \star & \star & \star & \star \\ 0 & 0 & 0 & \blacksquare & \star & \star & \star \\ 0 & 0 & 0 & 0 & \blacksquare & \star & \star \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Such matrices are the result of Gaussian elimination. Then the system

$$G = \left\{ \sum_{j=1}^{n} a_{ij} x_j \mid i = 1, \dots, m \right\}$$

is a Gröbner basis of the ideal $\langle G \rangle$ for the lexicographic order. This is easier to prove with the tools introduced in the next section.

10.2.5 Division

Monomial orders have let us define the leading monomial of a polynomial. This is the first step in the Euclidean division in $\mathbb{K}[x]$, that we now generalize to the multivariate setting.

Proposition 10.1. *If $G = \{g_1, \dots, g_m\} \subset \mathbb{A}$ and $F \in \mathbb{A}$, there exists $(B, r) \in \mathbb{A}^2$ such that*

- *$F = B + r$;*
- *$B \in \langle G \rangle$;*
- *$r = 0$ or no $\mathrm{LM}(g_i)$ divides a monomial of r.*

If G is the Gröbner basis of an ideal $\mathcal{I}$, then (B, r) is unique.

The polynomial r is denoted $\overline{F}^G$.

Before proving this proposition, we discuss a few consequences of this result.

Corollary 10.2. *Ideal membership in $\mathcal{I}$ can be tested with a Gröbner basis G of $\mathcal{I}$: $F \in \mathcal{I} \Leftrightarrow \overline{F}^G = 0$.*

Proof. If $\overline{F}^G = 0$ then F is equal to the polynomial B of the proposition, which is in $\mathcal{I}$ since $G \subset \mathcal{I}$. Conversely, if $F \in \mathcal{I}$, then $r = \overline{F}^G = F - B \in \mathcal{I}$. Then by definition of a Gröbner basis, either $r = 0$ or $\mathrm{LM}(r) \in \langle \mathrm{LM}(G) \rangle$. As the second case does not occur by the proposition, it follows that $r = 0$. □

Corollary 10.3. *An ideal $\mathcal{I}$ is generated by a Gröbner basis of it: $\mathcal{I} = \langle G \rangle$.*

Proof. It is a consequence of the previous corollary: if $F \in \mathcal{I}$, then $\overline{F}^G = 0$ and $F \in \langle G \rangle$. The other inclusion comes from $G \subset \mathcal{I}$. □

Proof of Proposition 10.1. Uniqueness is easy: if $F = B_1 + r_1 = B_2 + r_2$ with the properties of the proposition, as no $\mathrm{LM}(g_i)$ divides the monomials of r_1 and r_2 the same happens for the monomials of $r_2 - r_1 = B_1 - B_2 \in \mathcal{I}$, which implies that it is 0.

The existence is constructive and given by the algorithm of Fig. 10.6. This algorithm is very similar to Euclidean division, except that one divides by a set of polynomials rather than only one. The correctness of the algorithm comes from three properties:

- invariant: at the end of each iteration of the loop,

  $$F - f = a_1 g_1 + \cdots + a_m g_m + r,$$

  as can be checked by induction;
- invariant: at the end of each iteration of the loop, no $\mathrm{LT}(g_i)$ divides a monomial of r, this comes from the way monomials are added to r;

Input: $F, G = \{g_1, \ldots, g_m\}$
Output: r and $(a_1, \ldots, a_m)$ s.t. $F = a_1 g_1 + \cdots + a_m g_m + r$
  $r = a_1 = \cdots = a_m = 0; f = F;$
  While $f \neq 0$ do
    $S := \{i \text{ s.t. } \mathrm{LT}(g_i) | \mathrm{LT}(f)\}$
    If $S = \emptyset$ then
      $r = r + \mathrm{LT}(f); f = f - \mathrm{LT}(f)$
    else
      $i = \min S; a_i = a_i + \mathrm{LT}(f)/\mathrm{LT}(g_i);$
      $f = f - (\mathrm{LT}(f)/\mathrm{LT}(g_i))g_i$
  Return $(r, a_1, \ldots, a_m)$

Figure 10.6: Division Algorithm

– variant: each iteration of the loop decreases $\mathrm{LM}(f)$. This follows from the fact that f is always subtracted a polynomial whose leading term is $\mathrm{LT}(f)$. Since the set of $\{\mathrm{LT}(f)\}$ constructed during the algorithm has a smallest element, by the properties of monomial orders, it follows that the loop can only be performed a finite number of times. Then at the end the set S is empty and $f = 0$. □

10.2.6 Elimination

Gröbner bases solve elimination problems.

Theorem 10.1. *Let $G = \{g_1, \ldots, g_m\}$ be a Gröbner basis of $\mathcal{I} \subset A$ for the* lexicographic *order. Then $G \cap \mathbb{K}[x_i, \ldots, x_n]$ is a Gröbner basis of the ideal $\mathcal{I} \cap \mathbb{K}[x_i, \ldots, x_n]$ for the lexicographic order.*

Proof. We use the notation

$$\mathbb{A}_i = \mathbb{K}[x_i, \ldots, x_n], \quad G_i = G \cap \mathbb{A}_i, \quad \mathcal{I}_i = \mathcal{I} \cap \mathbb{A}_i.$$

Since $G_i \subset \mathcal{I}_i$, it follows that $\langle \mathrm{LM}(G_i) \rangle \subset \langle \mathrm{LM}(\mathcal{I}_i) \rangle$ in $\mathbb{A}_i$.

Conversely, if $F \in \mathcal{I}_i \setminus \{0\} \subset \mathcal{I}$, then there exists $g_j \in G$ such that $\mathrm{LM}(g_j) \mid \mathrm{LM}(F)$. By definition of the lexicographic order, this implies $\mathrm{LM}(g_j) \in \mathbb{A}_i$ and thus also $g_j \in \mathbb{A}_i$ and thus $g_j \in G_i$. We have proved $\mathrm{LT}(F) \in \langle \mathrm{LT}(G_i) \rangle$. □

It is an exercise to show that the same result holds for the ith elimination order, except that now G_i is a basis for the graded reverse lexicographic order.

Example 10.3. Coming back to the ideal $\mathcal{I}$ encoding the intersection of three surfaces in Section 10.1.3, it follows that the entirety of $\mathcal{I} \cap \mathbb{K}[y, z]$ is generated by

$$\{y^2 - y - z^2 + z, \quad z^2(2y + z^2 - 1), \quad z^2(z-1)^2(z^2 + 2z - 1)\},$$

while the intersection of $\mathcal{I}$ with $\mathbb{K}[z]$ is generated by the last polynomial.

10.3 Buchberger's Algorithm

We now describe Buchberger's algorithm, which computes a Gröbner basis for an ideal of $\mathbb{A}$ and a monomial order, and thereby proves that such a basis always exists.

10.3.1 S-polynomials

Definition 10.9. *The* S-polynomial *of f and g in $\mathbb{A}$ is the polynomial*

$$S(f,g) = \operatorname{lcm}(\operatorname{LM}(f), \operatorname{LM}(g)) \left(\frac{f}{\operatorname{LT}(f)} - \frac{g}{\operatorname{LT}(g)} \right) \in \langle f, g \rangle.$$

Note that this is indeed a polynomial, as the lcm in the numerator is a multiple of both leading terms in the denominators. The idea behind this construction is summarized in the following.

Lemma 10.2. *The leading monomial of the S-polynomial of f and g is smaller than the lcm of their leading monomials:*

$$\operatorname{LM}(S(f,g)) \prec \operatorname{lcm}(\operatorname{LM}(f), \operatorname{LM}(g)).$$

Proof. Both leading terms of f and g are multiplied by monomials in such a way that the leading terms of both summands in $S(f,g)$ are $\operatorname{lcm}(\operatorname{LM}(f), \operatorname{LM}(G))$, with opposite signs. It follows that the sum has smaller leading monomial. □

The following characterization of Gröbner bases in terms of S-polynomials is the basis of Buchberger's algorithm.

Proposition 10.2. *The set of polynomials $G = \{g_1, \dots, g_m\} \subset \mathbb{A}$ is a Gröbner basis of $\langle G \rangle$ if and only if*

$$\overline{S(g_i, g_j)}^{G} = 0, \qquad 1 \leq i < j \leq m.$$

Exercise 10.1. Use this characterization to prove the Gröbner basis for linear systems from Section 10.2.4.

Proof. Since $S(g_i, g_j) \in \langle g_i, g_j \rangle \subset \langle G \rangle$, it follows from Proposition 10.1 that it reduces to 0 by the division algorithm.

For the converse inclusion, starting from $F \in \langle G \rangle$, the aim is to show that there is $g_i \in G$ such that $\operatorname{LM}(g_i) \mid \operatorname{LM}(F)$. Since $F \in \langle G \rangle$, it decomposes as

$$F = h_1 g_1 + \cdots + h_m g_m. \tag{10.3}$$

There is no uniqueness of these decompositions and among all of them, we choose one such that

$$\delta = \max_i \operatorname{LM}(h_i g_i) \text{ is minimal},$$

which is possible, since the set of $\operatorname{LM}(h_i g_i)$ among all possible decompositions has a smallest element by the monomial order.

Of course, if $\delta = \operatorname{LM}(F)$, the result is proved. Otherwise, $\delta \succ \operatorname{LM}(F)$ and the idea is to use the S-polynomials $S(g_i, g_j)$ to obtain a decomposition of F with a smaller δ, a contradiction. Up to renumbering the g_i, we may assume that

$$\operatorname{LM}(h_i g_i) = \delta \text{ for } i \leq k, \quad \operatorname{LT}(h_i g_i) \prec \delta \text{ for } i > k.$$

Thus F rewrites as

$$F = \sum_{i=1}^{k} \operatorname{LT}(h_i) g_i + \sum_{i=1}^{k} (h_i - \operatorname{LT}(h_i)) g_i + \sum_{i>k} h_i g_i, \tag{10.4}$$

where the last two sums have leading monomial smaller than δ. Furthermore, one can rewrite

$$\operatorname{LT}(h_i) g_i = \operatorname{LT}(h_i) \operatorname{LT}(g_i) \frac{g_i}{\operatorname{LT}(g_i)} = \operatorname{LC}(h_i g_i) \delta \frac{g_i}{\operatorname{LT}(g_i)}.$$

For any $i \leq k$, since $\delta = \mathrm{LM}(h_i g_i) = \mathrm{LM}(h_k g_k)$, it follows that

$$\gamma_{i,k} = \mathrm{lcm}(\mathrm{LM}(g_i), \mathrm{LM}(g_k)) \mid \delta.$$

In view of the definition of S-polynomials, we get an identity among polynomials

$$\delta \frac{g_k}{\mathrm{LT}(g_k)} + \frac{\delta}{\gamma_{i,k}} S(g_i, g_k) = \delta \frac{g_k}{\mathrm{LT}(g_k)} + \frac{\delta}{\gamma_{i,k}} \gamma_{i,k} \left(\frac{g_i}{\mathrm{LT}(g_i)} - \frac{g_k}{\mathrm{LT}(g_k)} \right) = \delta \frac{g_i}{\mathrm{LT}(g_i)}.$$

This can be used to rewrite all g_i in terms of g_k, plus smaller order terms. Injecting in Eq. (10.4) gives

$$\sum_{i=1}^{k} \mathrm{LT}(h_i) g_i = \left(\sum_{i=1}^{k} \mathrm{LC}(h_i g_i) \right) \delta \frac{g_k}{\mathrm{LT}(g_k)} + \sum_{i=1}^{k} \mathrm{LC}(h_i g_i) \frac{\delta}{\gamma_{i,k}} S(g_i, g_k).$$

If the sum in the first term on the right-hand side is not 0, then the leading monomial of the right-hand side is δ, which is impossible since $\delta \succ \mathrm{LM}(F)$. Thus the first sum 0. In the second sum, all terms have leading term smaller than δ and, since $\overline{S(g_i, g_j)}^{G} = 0$, they decompose as linear combinations of the g_i. Putting together this identity and the right-hand side of Eq. (10.4) gives a decomposition of F like Eq. (10.3), but with a smaller leading monomial, a contradiction. □

10.3.2 Buchberger's Algorithm

The algorithm, given in Fig. 10.7, is a direct consequence of the previous proposition. It was discovered by Buchberger in 1965.

```
Input: f_1, ..., f_m in 𝔸; a monomial order ≤
Output: a Gröbner basis of ⟨f_1, ..., f_m⟩ for ≤
G = {f_1, ..., f_m}
S = {S(f_i, f_j) | i < j}
While S ≠ ∅
    Pick p ∈ S
    S := S\{p}
    g := p̄^G
    If g ≠ 0
        S := S ∪ {S(g, h) | h ∈ G}
        G := G ∪ {g}
Return G
```

Figure 10.7: Buchberger's algorithm

A clear invariant is that $\langle G \rangle = \langle f_1, \ldots, f_m \rangle$ at each iteration of the loop. That the algorithm is correct if it terminates is a consequence of Proposition 10.2. At each iteration, we also have the variant that either the number of elements of S decreases, or the ideal $\langle \mathrm{LT}(G) \rangle$ increases.

Proposition 10.3 (Dickson's Lemma)**.** *For any $A \subset \mathbb{N}^n$, the ideal $\mathcal{I} = \langle x^\alpha \mid \alpha \in A \rangle$ admits a finite basis $\{x^{\alpha(1)}, \ldots, x^{\alpha(s)}\}$.*

Proof of termination of Buchberger's algorithm. Consider the sequence of ideals $\langle \mathrm{LT}(G) \rangle$ constructed during the execution of the algorithm. Note that it is strictly increasing every time a new g is added to G. The union of these ideals is an ideal as in Dickson's lemma and therefore admits a finite basis. Each of the generators must belong to one of the ideals and therefore the sequence becomes stationary after a finite number of steps. □

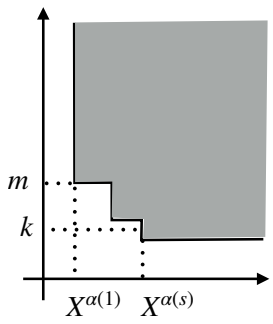


Figure 10.8: Illustration of the proof of Dickson's lemma

Proof of Dickson's lemma. The proof is by induction on the dimension n. For $n = 1$, the generator is $x^{\min(A)}$.

For $n > 1$, the proof proceeds by projection on the case with $n - 1$ variables. It is illustrated in a simple situation in Fig. 10.8. Let $X = x_1, \ldots, x_{n-1}$ and $Y = x_n$ and consider the ideal

$$\mathcal{J} = \{X^\alpha \mid \exists m, X^\alpha Y^m \in \mathcal{I}\}.$$

By the induction hypothesis, it admits a basis $\{X^{\alpha(1)}, \ldots, X^{\alpha(s)}\}$. For each $i = 1, \ldots, s$, let

$$m_i := \min\{m \in \mathbb{N} \mid X^{\alpha(i)} Y^m \in \mathcal{I}\}, \qquad m = \max m_i.$$

Then, for $k = 0, \ldots, m$, let

$$\mathcal{J}_k = \{X^\alpha \mid X^\alpha Y^k \in \mathcal{I}\}.$$

Again by the induction hypothesis, each of these ideals has a finite basis B_k. It follows that

$$\{MY^k \mid M \in B_k, 0 \leq k \leq m\}$$

is a finite basis of $\mathcal{I}$. □

Another consequence of Dickson's lemma is the following.

Theorem 10.2 (Hilbert's Basis Theorem)**.** *Every ideal $\mathcal{I} \subset \mathbb{K}[x_1, \ldots, x_n]$ has a finite generating set.*

Proof. By Dickson's lemma, the ideal $\langle \mathrm{LM}(\mathcal{I}) \rangle$ has a finite basis $\{\mathrm{LM}(g_1), \ldots, \mathrm{LM}(g_m)\}$. Then $G = \{g_1, \ldots, g_m\}$ is a Gröbner basis of $\langle G \rangle$, since

$$\langle \mathrm{LM}(\langle G \rangle) \rangle \supset \langle \mathrm{LM}(G) \rangle = \langle \mathrm{LM}(\mathcal{I}) \rangle \supset \langle \mathrm{LM}(\langle G \rangle) \rangle.$$

For any $F \in \mathcal{I}$, the division algorithm can thus be applied to F with respect to G and gives

$$F = h_1 g_1 + \cdots + h_m g_m + r.$$

In this decomposition, $r \in \mathcal{I}$ by subtraction. But since none of its monomials is divisible by the $\mathrm{LM}(g_i)$ that generate $\langle \mathrm{LM}(\mathcal{I}) \rangle$, it has to be 0, which proves that G generates $\mathcal{I}$ and is thus a Gröbner basis of it. □

10.4 Radicals and Nullstellensatz

Ideal membership is not a complete answer to the characterization of solutions of polynomial systems. In the example from Section 10.1.5 of the intersection of a circle and a hyperbola, one can test that $y^2 - 1$ does not belong to the ideal $\mathcal{I} = \langle x^2 + y^2 - 2,\ xy - 1 \rangle$, using Gröbner bases. The corresponding Maple commands are:

```
> F:=[x^2+y^2-2,x*y-1]:
> G:=Groebner[Basis](F,tdeg(x,y));
```

$$G := [xy - 1, x^2 + y^2 - 2, y^3 + x - 2y]$$

```
> Groebner[NormalForm](y^2-1,G,tdeg(x,y));
```

$$y^2 - 1$$

This shows that $\overline{y^2-1}^G \neq 0$ and thus that $y^2 - 1$ is not in the ideal, although it vanishes on all solutions of the system. The right object to describe this situation is the radical of the ideal.

10.4.1 Radicals

Definition 10.10. *The* radical *of an ideal $\mathcal{I}$ of a ring $\mathbb{A}$ is the ideal*

$$\sqrt{\mathcal{I}} = \{f \in \mathbb{A} \mid \exists m \in \mathbb{N}, f^m \in \mathcal{I}\}.$$

Testing whether a polynomial belongs to the radical of a polynomial ideal can be performed using Gröbner bases thanks to the following important result.

Proposition 10.4 (Rabinowitsch's Trick)**.**

$$f \in \sqrt{\langle f_1, \ldots, f_m\rangle} \Leftrightarrow \langle f_1, \ldots, f_m, 1 - tf\rangle = \langle 1\rangle = \mathbb{K}[x_1, \ldots, x_n, t].$$

Using this proposition, the previous computation can be concluded:

```
> Groebner[Basis]([op(F),1-t*(y^2-1)],tdeg(x,y,t));
```

$$[1]$$

This shows that $y^2 - 1 \in \sqrt{\mathcal{I}}$ and therefore vanishes on all the solutions of the system.

Proof. If $f^p \in \langle f_1, \ldots, f_m\rangle$, then writing

$$1 = (1 - tf)(1 + \cdots + t^{p-1}f^{p-1}) - t^p f^p$$

shows that $1 \in \langle f_1, \ldots, f_m, 1 - tf\rangle$.

Conversely, if 1 belongs to this ideal, then it can be written

$$1 = g_1(x, t)f_1 + \cdots + g_m(x, t)f_m + g(x, t)(1 - tf)$$

for polynomials $g_i(x, t)$ and $g(x, t)$ in $\mathbb{K}[x_1, \ldots, x_n, t]$. Evaluating this identity at $t = 1/f$ and multiplying by the common denominator (a power of f) gives a decomposition of that power of f as a linear combination of $f_1, \ldots, f_m$, showing that f is in the radical of the ideal they generate. □

10.4.2 Hilbert's Nullstellensatz

The last result of this lecture makes an explicit relation between algebra and geometry, when the field is algebraically closed.

Theorem 10.3 (Hilbert's Nullstellensatz)**.** *If $\mathbb{K}$ is algebraically closed and $f, f_1, \ldots, f_m$ belong to $\mathbb{A} = \mathbb{K}[x_1, \ldots, x_n]$ then f belongs to the ideal $\sqrt{\langle f_1, \ldots, f_m\rangle}$ if and only if f vanishes on the common solutions of $(f_1, \ldots, f_m)$ in $\mathbb{K}^n$.*

Proof. The direct implication is clear: if there exists p such that $f^p \in \langle f_1, \ldots, f_m \rangle$, then f^p vanishes at the common zeros of $f_1, \ldots, f_m$ and therefore so does f.

The converse implication requires more work.

Lemma 10.3 (Noether Normalization Lemma)**.** *Let $f \in \mathbb{K}[x_1, \ldots, x_n]$ with $n \geq 2$ have degree $d > 0$. If $\mathbb{K}$ is infinite, there exists $(\lambda_1, \ldots, \lambda_{n-1}) \in \mathbb{K}^{n-1}$ such that the coefficient of x_n^d of*

$$f(x_1 + \lambda_1 x_n, \ldots, x_{n-1} + \lambda_{n-1} x_n, x_n)$$

is not zero.

This lemma is illustrated in Fig. 10.9. The polynomial $xy - 1$ does not have the property: there is value of x, $x = 0$, where its degree in y decreases. The change to $(x + y/5)y - 1$ corrects this.

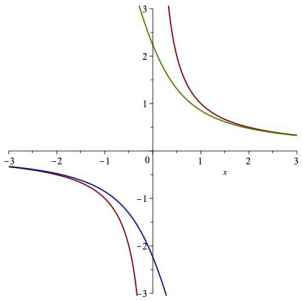


Figure 10.9: Noether Normalization

Proof. Denote by a_α the coefficient of x^α in f for all $\alpha \in \mathbb{N}^n$. Since the degree of f is d, the homogeneous polynomial

$$g(x_1, \ldots, x_n) = \sum_{\alpha_1 + \cdots + \alpha_n = d} a_\alpha x^\alpha$$

is not 0. The coefficient in the lemma is precisely $g(\lambda_1, \ldots, \lambda_{n-1}, 1)$. We now construct a point $(\lambda_1, \ldots, \lambda_{n-1}, 1)$ where g does not vanish, by induction on n. If $n = 2$, g is a univariate nonzero polynomial. Therefore it has a finite number of zeros and since $\mathbb{K}$ is infinite, one can find a λ_1 which is not one of them. Otherwise, viewing g as a polynomial in λ_{n-1}, we first find by induction a point $(\lambda_1, \ldots, \lambda_{n-2})$ where the leading coefficient does not vanish. At this point, we are left with a nonzero polynomial in x_{n-1} for which again, a suitable λ_{n-1} can be found. □

Another ingredient in the proof is the weak Nullstellensatz, that asserts that the polynomials of an ideal that does not contain 1 have common zeros when the field is algebraically closed (and therefore infinite).

Lemma 10.4 (Weak Nullstellensatz)**.** *If $\mathcal{I}$ is a strict ideal of $\mathbb{A} = \mathbb{K}[x_1, \ldots, x_n]$ and $\mathbb{K}$ is algebraically closed, there exists $a \in \mathbb{K}^n$ such that for all $f \in \mathcal{I}$, $f(a) = 0$.*

A consequence is that if $f_1, \ldots, f_s$ are polynomials in $\mathbb{A}$ that do not have any common solution in the algebraically closed $\mathbb{K}$, then there exist polynomials $g_1, \ldots, g_s$ such that

$$1 = g_1 f_1 + \cdots + g_s f_s.$$

Indeed, the lemma asserts that the ideal $\mathcal{I}$ generated by $(f_1, \ldots, f_s)$ is not strict and therefore contains 1.

Proof. The proof is by induction on n. The case when $n = 1$ follows from the algebraic closure of $\mathbb{K}$.

Since $\mathcal{I}$ is a strict ideal of $\mathcal{A}$, there exists $g \neq 0$ in $\mathcal{I}$ of degree at least 1. By the Noether normalization lemma, up to changing coordinates, one can assume that

$$g = x_n^d + g_{d-1}x_n^{d-1} + \cdots + g_0, \quad g_i \in \mathbb{K}[x_1, \ldots, x_{n-1}].$$

(Indeed, for any $(\lambda_1, \ldots, \lambda_{n-1}) \in \mathbb{K}^{n-1}$, the ideal $\mathcal{K} = \{f(x_1 + \lambda_1 x_n, \ldots, x_{n-1} + \lambda_{n-1}x_n, x_n) \mid f \in \mathcal{I}\}$ is a strict ideal since it is the case for $\mathcal{I}$.)

By induction, there exists $a' \in \mathbb{K}^{n-1}$ a common zero of the polynomials in the ideal $\mathcal{I}' = \mathcal{I} \cap \mathbb{K}[x_1, \ldots, x_{n-1}]$. Consider now the ideal

$$\mathcal{J} = \{f(a', x_n) \mid f \in \mathcal{I}\} \subset \mathbb{K}[x_n].$$

We want to show that $\mathcal{J}$ is a strict ideal of $\mathbb{K}[x_n]$, since then the case $n = 1$ of the lemma gives a_n that concludes the proof. If $\mathcal{J}$ is not strict, there exists $f \in \mathcal{I}$,

$$f = f_e x_n^e + \cdots + f_0, \quad f_i \in \mathbb{K}[x_1, \ldots, x_{n-1}],$$

such that $f(a', x_n) = 1$. This means that $f_0(a') = 1$ and $f_i(a') = 0$ for $i > 0$. The resultant $\mathrm{Res}_{x_n}(f, g)$ belongs to $\mathcal{I}'$ (see Lecture 5). However, looking at its expression

$$\mathrm{Res}_{X_n}(f, g) := \begin{vmatrix} f_0 & & & g_0 & & \\ f_1 & \ddots & & g_1 & \ddots & \\ \vdots & & f_0 & \vdots & & g_0 \\ f_e & & f_1 & 1 & & g_1 \\ & \ddots & \vdots & & \ddots & \vdots \\ & & f_e & & & 1 \end{vmatrix}$$

shows that it is 1 at a' rather than 0, a contradiction. □

We can now conclude the proof of the Nullstellensatz itself.

If f vanishes on the common solutions of $(f_1, \ldots, f_m)$ in $\mathbb{K}^n$, then $(f_1, \ldots, f_m, 1 - tf)$ do not have common zeros. By the weak Nullstellensatz, this implies that the ideal $\langle f_1, \ldots, f_m, 1 - tf\rangle$ is not strict. Then by Rabinowitsch's trick, f belongs to the radical of $\langle f_1, \ldots, f_m\rangle$, as was to be proved. □

Additional bibliography

A wonderful and very readable account of the questions related to Gröbner bases and their applications is the classical

D.A. Cox, J.B. Little, and D. O'Shea. *Ideals, varieties, and algorithms.* 4th edition. Springer New York, 2015

Appendix A

Basic Algebraic Structures

A *group* is a set G with a law of composition such that: the law is associative; G has an identity element e for this law; any element of G has an inverse for this law.

The *order* of an element $a \in G$ is the smallest positive integer n such that $a^n = e$, if it exists (denoting the composition law by a product). If such a n does not exist, the order is infinite.

A *ring* is a set $\mathbb{A}$ with two laws of compositions, called addition and multiplication and: $\mathbb{A}$ is a commutative group for addition; multiplication is associative; $\mathbb{A}$ has an identity element for multiplication, noted 1; multiplication is distributive over addition. The identity element for addition is noted 0.

The *characteristic* of the ring is the order of 1 for the additive law.

A *unit* of $\mathbb{A}$ is an element that has a left and right inverse for multiplication. The units of the ring form a group, sometimes denoted $\mathbb{A}^*$.

A *zero divisor* $x \in \mathbb{A}$ is a nonzero element such that there exists $y \neq 0$ with $xy = 0$. Examples are: 2 in $\mathbb{Z}/6\mathbb{Z}$; the matrix $\left(\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right)$, whose square is 0.

A ring is called *commutative* when its multiplication is.

An *integral domain* is a commutative ring where $1 \neq 0$ and that has no zero divisor. Examples are $\mathbb{Z}$ and $\mathbb{Q}[x_1, \ldots, x_n]$.

A *field* is a commutative ring where $1 \neq 0$ and every nonzero element is invertible.

The *field of fraction* of an integral domain $\mathbb{A}$ is the smallest field that contains $\mathbb{A}$. Its elements are denoted $b^{-1}a$ or a/b, with a and b in $\mathbb{A}$. Examples are $\mathbb{Q}$, field of fractions of $\mathbb{Z}$, and $\mathbb{K}(x_1, \ldots, x_n)$, field of fractions of $\mathbb{K}[x_1, \ldots, x_n]$ for $\mathbb{K}$ a field.

An *ideal* $\mathcal{I}$ in a commutative ring $\mathbb{A}$ is a subset of $\mathbb{A}$ that is an additive subgroup of $\mathbb{A}$, closed by multiplication by elements of $\mathbb{A}$.

An ideal is *principal* if it is of the form $\mathbb{A}a$ for some $a \in \mathbb{A}$, called a *generator* of the ideal. More generally, given elements $a_1, \ldots, a_n$ of $\mathbb{A}$, the set $\{u_1a_1 + \cdots + u_na_n \mid (u_1, \ldots, u_n) \in \mathbb{A}^n\}$ is an ideal and the a_i are its *generators*. This ideal is often denoted $(a_1, \ldots, a_n)$.

A commutative ring is *principal* when all of its ideals are principal. This is the case of $\mathbb{Z}$ and $\mathbb{K}[X]$ with a field $\mathbb{K}$. It is not the case of $\mathbb{K}[X, Y]$: the ideal generated by (X, Y) cannot be generated by a single element.

An ideal $\mathcal{I}$ of a commutative ring $\mathbb{A}$ is *prime* if $\mathcal{I} \neq \mathbb{A}$ and whenever a, b in $\mathbb{A}$ have their product in $\mathcal{I}$, then one of a, b belongs to $\mathcal{I}$. Examples are: the ideal $p\mathbb{Z}$ of $\mathbb{Z}$ with p a prime number; the ideal $(P(X))$ of $\mathbb{Q}[X]$ with P an irreducible polynomial.

An ideal different from the ring is *maximal* if is contained in only two ideals: itself and the ring. Maximal ideals are prime. If the ring is principal, then the converse holds.

The *quotient ring* of a ring $\mathbb{A}$ by an ideal $\mathcal{I}$ is the set of equivalence classes for the relation $a \sim b$ when $a - b \in \mathcal{I}$. This set forms a commutative ring denoted $\mathbb{A}/\mathcal{I}$. A quotient by a prime ideal is an integral domain. A quotient by a maximal ideal is a field. Examples are $\mathbb{Z}/k\mathbb{Z}$ for $k \in \mathbb{Z} \setminus \{0\}$ or $\mathbb{R}[x]/(x^2+1)$, which is isomorphic to $\mathbb{C}$.

A field $\mathbb{K}$ is *algebraically closed* if every non-constant polynomial in $\mathbb{K}[X]$ has a root in $\mathbb{K}$. The basic example is $\mathbb{C}$. A finite field cannot be algebraically closed.

A *vector space* V over a field $\mathbb{K}$ is a commutative group, usually written additively, together with an external law of multiplication by elements of $\mathbb{K}$ on V such that for all a, b in $\mathbb{K}$ and all x, y in V: $(a+b)x = ax + bx$, $a(x+y) = ax + ay$, $(ab)x = a(bx)$, $1x = x$.

An *algebra* $\mathbb{A}$ over a field $\mathbb{K}$ (or a $\mathbb{K}$-algebra) is a ring that is also a vector space over $\mathbb{K}$ with the same addition and the scalar multiplication satisfies $a \cdot (xy) = (a \cdot x)y = x(a \cdot y)$. Examples are : $n \times n$ matrices of elements of $\mathbb{K}$; polynomials in $\mathbb{K}[x_1, \ldots, x_n]$.

If $\mathcal{I}$ is an ideal of a $\mathbb{K}$-algebra $\mathbb{A}$, then the quotient $\mathbb{A}/\mathcal{I}$ is also a $\mathbb{K}$-algebra. For example, for any nonzero $P \in \mathbb{K}[X]$, the quotient $\mathbb{K}[X]/(P)$ is an algebra (it has finite dimension, that is the degree of P).

Additional bibliography

Wikipedia is usually reliable for these notions. Much more detailed information can be found in the classical

Serge Lang. *Algebra*. 3rd edition. Vol. 211. Graduate Texts in Mathematics. New York: Springer-Verlag, 2002, pp. xvi+914

An amazingly compact book that is a pleasure to read and leads its readers to very advanced material is

Igor R. Shafarevich. *Basic notions of algebra*. Vol. 11. Encyclopaedia of Mathematical Sciences. Translated from the 1986 Russian original by Miles Reid, Reprint of the 1997 English translation [MR1634541], Algebra, I. Berlin: Springer-Verlag, 2005, pp. ii+ 258. ISBN: 978-3-540-25177-4; 3-540-25177-4