Linear Recurrences and Linear Differential Equations (1)

These notes are an attempt at adapting part of the material on linear differential equations and recurrences initially planned for lectures 6, 9, and 10 (corresponding more or less to Part III of *Algorithmes efficaces en calcul formel*) in a format more suitable for self-study. Any feedback is welcome.

Many proofs and examples that we would normally discuss in detail during class are presented as exercises, sometimes with a solution or additional hints at the end of the document. These exercises are intended to be easy and I would strongly encourage you to solve them as you read. You don't necessarily need to write down all solutions in full detail, but handwaving alone is not sufficient. You probably should consider contacting me if you are stuck for more than a few minutes on one of the questions.

Paragraphs typeset like this are less central to the course; feel free to skim or skip them.

Introduction

This part of the course deals with computations with linear recurrence relations and linear ordinary differential equations with polynomial coefficients. We will discuss questions such as

- how to efficiently compute solutions given by initial values, or selected terms of such solutions;
- how to carry out "symbolic" operations on functions or sequences using the differential equations or recurrences they satisfy as a data structure (cf. lecture 1);
- how to find all solutions lying in a certain well-defined class, e.g., how to decide if a linear differential equation admits solutions that are rational functions.

This version of the notes focuses on the first point. I may update them with material on the second point in the coming days.

We denote by $\mathbb{K}$ an effective field. The main examples we have in mind are $\mathbb{K}=\mathbb{Q}$ and $\mathbb{K}=\mathbb{F}_p$. (Many of the results could be adapted to the case of a ring.)

We denote by $\mathbb{K}[x]_{<d}$ the polynomials over $\mathbb{K}$ of degree less than d, by $\mathbb{K}^{n\times n}$ the $n\times n$ matrices over $\mathbb{K}$, and by $\mathbb{K}^{\mathbb{N}}$ the sequences indexed by $\mathbb{N}$ of elements of $\mathbb{K}$. Finally, $\mathrm{M}(\cdot)$ is a multiplication function for polynomials (see lecture 1), $\mathrm{M}_{\mathbb{Z}}(\cdot)$ its analogue for integer multiplication (we sometimes drop the subscript), and θ is such that $n\times n$ matrices over $\mathbb{K}$ can be multiplied in $\mathrm{O}(n^\theta)$ operations in $\mathbb{K}$.

1 Linear recurrence relations

1.1 C-finite sequences

Definition 1. A sequence $(u_n)\in\mathbb{K}^{\mathbb{N}}$ is called *C-finite*, or *C-recursive*, when it satisfies a linear recurrence with constant coefficients over $\mathbb{K}$, that is

$$\forall n\in\mathbb{N},\quad u_{n+s}+c_{s-1}\,u_{n+s-1}+\cdots+c_0\,u_n=0 \qquad \text{with } c_i\in\mathbb{K}. \qquad \triangleleft$$

Note that we assume that the leading coefficient[1] is one, but there is no loss of generality in doing so. With this convention, a C-finite sequence is completely determined by the coefficients $c_0,\dots,c_{s-1}$ along with s initial values $u_0,\dots,u_{s-1}$ (and this holds even if $\mathbb{K}$ is replaced by a ring in the definition).

The integer s is the *order* of the recurrence. The polynomial $\chi(t)=t^s+c_{s-1}\,t^{s-1}+\cdots+c_0$ is called the *characteristic polynomial* of the recurrence.

Example. The sequence $(2^n)_{n\in\mathbb{N}}$ is C-finite. The Fibonacci sequence $(F_n)_{n\in\mathbb{N}}$ defined by $F_0=0$, $F_1=1$, and $F_{n+2}=F_n+F_{n+1}$ is C-finite. ◁

Exercise 1. Show that for any polynomial $p\in\mathbb{K}[X]$, the sequence $(p(n))_{n\in\mathbb{N}}$ is C-finite. (Hint: what is the degree of $p(n+1)-p(n)$?) ◁

Remarks.

- You may already know that, in characteristic zero and for large enough n, the general term of a C-finite sequence can be written in closed form as

$$u_n=\sum_{\chi(\alpha)=0} p_\alpha(n)\,\alpha^n$$

  where the summation ranges over the roots of the characteristic polynomial and each $p_\alpha\in\bar{\mathbb{K}}[X]$ is a polynomial of degree $\leqslant m-1$ where m is the multiplicity of the root α. Conversely, any sequence of this form is C-finite.

- *Linear Feedback Shift Registers* (LFSRs) and *Infinite Impulse Response* (IIR) *filters* are essentially the same thing as linear recurrences with constant coefficients (in the latter case, inhomogeneous recurrences, i.e., recurrences with an arbitrary sequence on the rhs). ◁

Proposition 2. A sequence $(u_n)\in\mathbb{K}^{\mathbb{N}}$ is C-finite if and only if its generating series $\sum_{n=0}^{\infty} u_n x^n$ is rational. ◁

Exercise 2.

1. Show that $(u_n)_{n\in\mathbb{N}}$ satisfies $u_{n+s}+c_{s-1}\,u_{n+s-1}+\cdots+c_0\,u_n=0$ if and only if its generating series is of the form (note the order of the coefficients in the denominator)

$$\sum_{n=0}^{\infty} u_n x^n=\frac{p(x)}{1+c_{s-1}\,x+\cdots+c_0\,x^s}$$

   for some $p\in\mathbb{K}[x]_{<s}$. (Extend the sequence by $u_n=0$ for negative indices and multiply by the denominator, or focus on characteristic zero and use the closed form above.)

2. Observe that a sequence that satisfies the recurrence relation only for $n\geqslant n_0$ for some n_0 is still C-finite. Deduce the proposition. ◁

1. One sometimes also requires $c_0\neq 0$; we will not do it here.

1.2 P-finite sequences

Definition 3. A sequence $(u_n) \in \mathbb{K}^{\mathbb{N}}$ is called *P-finite* (or *P-recursive*) when it satisfies a linear recurrence with polynomial coefficients over $\mathbb{K}$, that is, when there are polynomials $b_0, \dots, b_s \in \mathbb{K}[X]$ with $b_s \neq 0$ such that

$$b_s(n)\, u_{n+s} + b_{s-1}(n)\, u_{n+s-1} + \cdots + b_0(n)\, u_n = 0 \tag{REC}$$

for all $n \in \mathbb{N}$. ◁

The recurrence has *order* s, and we call $d = \max_i \deg b_i$ its *degree*.

We do not assume that the leading coefficient is constant. Therefore, it is no longer true in general that the sequence $(u_n)_{n\in\mathbb{N}}$ is determined by the recurrence (REC) and the first s terms. This leads us to make the following definition.

Definition 4. We will say that the recurrence (REC) is *nonsingular* if $b_s(n) \neq 0$ for all $n \in \mathbb{N}$. ◁

Exercise 3. Which of the following sequences are P-finite over $\mathbb{Q}$?

$$(2^n)_{n\in\mathbb{N}}, \qquad (n!)_{n\in\mathbb{N}}, \qquad (2^{2^n})_{n\in\mathbb{N}}.$$

(solution at the end) ◁

Later it will also be useful to generalize Definition 3 to sequences indexed by the relative integers: a sequence $(u_n)_{n\in\mathbb{Z}}$ is P-finite if it satisfies (REC) for all $n \in \mathbb{Z}$.

Exercise 4. Prove that

- $(u_n)_{n\in\mathbb{N}}$ is P-finite iff there exist $n_0 \in \mathbb{N}$ and a relation of the form (REC) that holds for all $n \geqslant n_0$.
- $(u_n)_{n\in\mathbb{N}}$ is P-finite iff the sequence $(\tilde{u}_n)_{n\in\mathbb{Z}}$ defined by $\tilde{u}_n = u_n$ for $n \geqslant 0$ and $\tilde{u}_n = 0$ for $n < 0$ is P-finite.

Hint: multiply (REC) by a suitable polynomial. ◁

2 Computing terms of linearly recurrent sequences

Our first family of algorithmic problems related to linear recurrences is the following: given a sequence $(u_n)_{n\in\mathbb{N}}$ defined by a recurrence with constant coefficients or by a nonsingular[2] recurrence with polynomial coefficients,

a) how fast can one compute the first N terms $u_0, \dots, u_{N-1}$?

b) how fast can one compute the term u_N alone?

2. The restriction to nonsingular recurrences is not essential. We will discuss some of the technical issues with generalizing the algorithms to the singular case in a different context in the following lectures.

We discuss these questions both in the algebraic complexity model and, for recurrences given by equations with coefficients in $\mathbb{Z}$ (which we view as recurrences over $\mathbb{Q}$ since we assume $\mathbb{K}$ to be a field), in the bit complexity model.

2.1 Computing the first N terms

Clearly, one can compute the first N terms of a P-finite sequence given by a nonsingular recurrence in $O(N)$ arithmetic operations by

$$\begin{cases} u_s &:= -\dfrac{1}{b_s(0)}\left(b_{s-1}(0)\,u_{s-1} + \cdots + b_0(0)\,u_0\right) \\ u_{s+1} &:= -\dfrac{1}{b_s(1)}\left(b_{s-1}(1)\,u_s \quad + \cdots + b_0(1)\,u_1\right) \\ \quad\vdots \end{cases} \tag{1}$$

The method applies in particular to C-finite sequences, and the complexity bound is optimal, both in the C-finite and in the P-finite case.

The picture gets more interesting if we also take into account the order of the recurrence, and, in the P-finite case, the degree of the polynomial coefficients.

Let us focus on the C-finite case. The naïve algorithm presented above uses $\mathrm{O}(s\,N)$ operations in $\mathbb{K}$. The basic idea to do better is that, given a slice $(u_{ks}, \ldots, u_{(k+1)s-1})$ of s consecutive terms, one can compute the next slice $(u_{(k+1)s}, \ldots, u_{(k+2)s-1})$ in only $\mathrm{O}(\mathrm{M}(s))$ operations instead of $\mathrm{O}(s^2)$.

This is best explained in terms of the generating series. By Proposition 2, we can write $\sum_{n\geqslant 0} u_n x^n = p(x)/q(x)$ with $\deg p < s$ and $\deg q \leqslant s$. First consider the case of the first slice. We know $q(x)$ (the recurrence) and $U_0(x) = u_0 + \cdots + u_{s-1}\,x^{s-1}$ (the initial values). Since $p/q = U_0 + \mathrm{O}(x^s)$ and $\deg p < s$, we can recover p in $\mathrm{O}(\mathrm{M}(s))$ operations as $q\,U_0 \operatorname{rem} x^s$, and from there we can compute the first $2\,s$ terms by a power series division in $\mathrm{O}(\mathrm{M}(2\,s)) = \mathrm{O}(\mathrm{M}(s))$ operations. So we now know the next slice $(u_s, \ldots, u_{2s-1})$. But the shifted sequence $(u_{s+n})_{n\in\mathbb{N}}$ satisfies the same recurrence as $(u_n)_{n\in\mathbb{N}}$, so we can iterate the same process: encode the new slice as a polynomial, recover the numerator of the rational generating series of the shifted sequence, divide, and so on.

In conclusion, we have the following complexity result. The same idea leads to a variant of the fast Euclidean division algorithm (cf. lecture 3) for unbalanced degrees that divides a polynomial of degree n by one of degree m in $\mathrm{O}(n\,\mathrm{M}(m)/m)$ operations.

Proposition 5. One can compute the first N terms of a C-finite sequence given by a recurrence of order s over $\mathbb{K}$ in $\mathrm{O}(NM(s)/s)$ operations in $\mathbb{K}$. ◁

In the P-finite case, if the degree of the polynomials b_i is large, it makes sense to use fast multipoint evaluation (lecture 3) for evaluating $b_i(0), b_i(1), \ldots$. The idea just presented in the C-finite case can also be adapted to the P-finite one; this is useful when the order of the recurrence is large compared to the degree of the coefficients.

Let us now turn to the bit complexity of the naïve algorithm (1) in the main case where the question makes sense, that is, for sequences of integers. The first thing we need to do is study the bit size of the intermediate results.

Exercise 5. Let (u_n) be a C-finite sequence given by $u_{n+s} + c_{s-1}\,u_{n+s-1} + \cdots + c_0\,u_n = 0$ with coefficients $c_i \in \mathbb{Z}$ and initial values $(u_0, \ldots, u_{s-1})$ of bounded bit size.

1. Show that there exists a constant $C > 0$ such that the bit size of u_n is bounded by $C\,n$ for every $n > 0$, and that some C-finite sequences reach this bound.

2. Show that the bit complexity of computing $(u_0, \dots, u_{N-1})$ by the naïve method (1) is $O(N^2)$ (use the fact that the integer multiplications at each step of (1) are unbalanced) and that this is quasi-optimal. ◁

Exercise 6. Show that the bit size of $n!$ is $n\log_2 n + O(n)$ (recall Stirling's formula). Deduce that the bit complexity for computing $(1!, 2!, 3!, \dots, (N-1)!)$ by the naïve method is $O(N^2 M_{\mathbb{Z}}(\log_2 N))$ and that this is quasi-optimal. ◁

The complexity bound of Exercise 6 easily generalizes to P-recursive sequences defined by a recurrence $b_s(n)\,u_{n+s} + b_{s-1}(n)\,u_{n+s-1} + \cdots + b_0(n)\,u_n = 0$ with $b_0, \dots, b_{s-1} \in \mathbb{Z}[X]$ and $b_s = 1$: the bit size of the nth term is $O(n\log n)$, and computing the first N terms can be done in $O(N^2 M_{\mathbb{Z}}(\log N))$, which is quasi-optimal in view of the case of the factorial. One can check that the size bounds still hold for the numerators and denominators of $u_n \in \mathbb{Q}$ if now $b_s \in \mathbb{Z}[X]$ is a polynomial with no roots in $\mathbb{N}$; the only thing that changes is that, to get the u_n in lowest terms, we may have to compute at stage n an integer gcd of cost $O(M_{\mathbb{Z}}(n\log(n))\log(n))$. The total cost is still $\tilde{O}(N^2)$, with a slightly worse log factor.

In summary, at least if we focus on the dependency on N, the conclusion so far is the same in all four cases we considered: the naïve algorithm reaches a quasi-optimal complexity for computing the first N terms at once. If, however, we only need the Nth term, then the arithmetic complexity is linear in N even though the output has constant size, and its bit complexity is (at least) quadratic in N in the worst case even though the bit size of the output is at worst quasi-linear.

2.2 Computing the Nth term alone: C-finite case

You may already have encountered the following classical method for computing the Nth term of a C-finite sequence, at least in special cases like that of Fibonacci numbers. The idea is to reduce the order of the recurrence to one by rewriting it in matrix form and then use binary powering.

Algorithm 6. *Input*: $(u_0, \dots, u_{s-1})$, $(c_0, \dots, c_{s-1})$, N *Output*: $(u_N, \dots, u_{N+s-1})$

1. Form the matrix
$$A = \begin{pmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ -c_0 & -c_1 & \cdots & -c_{s-1} \end{pmatrix}.$$
2. Compute $B = A^N$ by binary powering.
3. Return $B \cdot (u_0, \dots, u_{s-1})^{\mathrm{T}}$. ◁

Exercise 7.

1. Show that, for all $n \geqslant 0$, one has $(u_{n+1}, \dots, u_{n+s})^{\mathrm{T}} = A\,(u_n, \dots, u_{n+s-1})^{\mathrm{T}}$ and deduce that Algorithm 6 is correct.

2. Show that it runs in $O(s^\theta \log N)$ operations in $\mathbb{K}$.

3. Assuming that $\mathbb{K} = \mathbb{Q}$ and that $c_0, \dots, c_{s-1}, u_0, \dots, u_{s-1}$ are integers bounded in absolute value by a constant C, show that Algorithm 6 runs in $O(M_{\mathbb{Z}}(N))$ bit operations (where the constant hidden in the $O(\cdot)$ depends on s and C). ◁

Thus, Algorithm 6 has quasi-optimal complexity with respect to N, both in terms of arithmetic operations and in terms of bit operations for integer recurrences.

For recurrences of large order, the complexity with respect to s also leaves a lot to be desired. Here is one way to do better.

Exercise 8. Let $\chi \in \mathbb{K}[X]$ be the characteristic polynomial of the recurrence $u_{n+s} + c_{s-1} u_{n+s-1} + \cdots + c_0 u_n = 0$, and let S denote the shift operator $S : (u_n)_{n\in\mathbb{N}} \mapsto (u_{n+1})_{n\in\mathbb{N}}$. Then $\chi(S)$ is a linear map from $\mathbb{K}^{\mathbb{N}}$ to itself.

1. Show that a sequence $\boldsymbol{u} = (u_n)_{n\in\mathbb{N}}$ satisfies the recurrence iff $\boldsymbol{u} \in \ker \chi(S)$.

2. Let $\boldsymbol{u}$ be a solution. Show that, when $P, Q, R \in \mathbb{K}[X]$ satisfy $P(X) = Q(X)\,\chi(X) + R(X)$, one has $P(S)(\boldsymbol{u}) = R(S)(\boldsymbol{u})$.

3. Deduce an algorithm to compute u_N in $O(M(s) \log N)$ operations in $\mathbb{K}$. (Apply the previous point with $P = X^N$ and use binary powering modulo $\chi(X)$.)

4. Adapt the idea to compute A^N for a matrix $A \in \mathbb{K}^{s\times s}$ and an integer $N \in \mathbb{N}$ in $O(s^{\theta+1} + M(s) \log N)$ operations in $\mathbb{K}$. (Recall that one can compute the characteristic polynomial of A in $O(s^\theta \log s)$ operations.) ◁

See also the slides of Alin's cancelled lecture 4 for another algorithm of similar complexity for the same task, which also forms the basis of the fastest known method for computing the composition of two formal power series.

2.3 Computing the Nth term of a P-finite sequence: bit complexity

2.3.1 *Case of $N!$*

Algorithm 6 does not work in the general P-finite case, since the matrix playing the role of A now depends on n. Let us first look at the simplest nontrivial case where this issue occurs: the computation of $N!$. Recall from Exercise 6 that the bit size of $N!$ is $N \log_2 N + O(N)$, but the cost of computing it iteratively is roughly quadratic in N, and in fact the total size of the intermediate results is already (more than) quadratic.

This can be avoided using a fundamental technique that we already encountered in previous lectures in the case of polynomials: organize the computation as a balanced product tree. More precisely, we split the product as

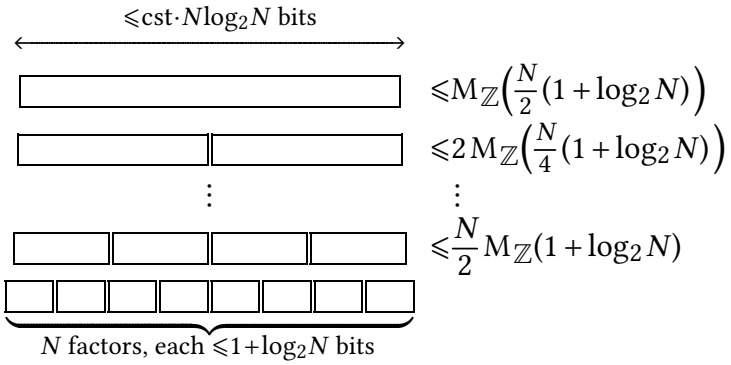
$$N! = \underbrace{(1 \times 2 \times \cdots \times m)}_{P(0,m)} \times \underbrace{((m+1) \times (m+2) \times \cdots \times N)}_{P(m,N)}$$

with $m = \lfloor N/2 \rfloor$ and compute each of the two factors recursively with a similar scheme, that is, using the formula

$$P(\ell, h) = P\left(\left\lfloor \frac{\ell + h}{2} \right\rfloor, h\right) P\left(\ell, \left\lfloor \frac{\ell + h}{2} \right\rfloor\right)$$

until $h \leqslant \ell + 1$.

The complexity analysis (assuming $N = 2^k$ for simplicity) can be summarized in a picture:



Each rectangle represents a recursive call and its size is proportional to the bit size of the result $P(\ell, h)$ of that call. One obtains $P(\ell, h)$ by multiplying two intermediate results of roughly the same size, so the size of $P(\ell, h)$ is roughly double that size. The quantities on the right of the picture are bounds on the total cost of the multiplications performed at each level in the call tree: a single multiplication at the top, two smaller ones just below, and so on.

Exercise 9. Still with $N = 2^k$, argue a bit more carefully that the bit size of the integers being multiplied in a recursive call at depth i (where the depth of the root is $i = 0$) is bounded by $\frac{N}{2^{i+1}}(1 + \log_2 N)$. Deduce that the total cost of the algorithm (neglecting control structures) is at most

$$\frac{1}{2}\mathrm{M}_{\mathbb{Z}}(N(1+\log_2 N))\log_2 N$$

bit operations. *(partial solution at the end)* ◁

Remark. In the special case of $N!$, there are in fact algorithms of complexity $\mathrm{O}(\mathrm{M}_{\mathbb{Z}}(N \log_2 N))$. What makes $N!$ special is that it decomposes into a product of small prime factors raised to large powers, with a regular enough structure that the exponents can be efficiently computed from N. The idea is, roughly speaking, to compute the product of all these small primes using a balanced product tree, but after reorganizing it so that many of the nodes of the tree compute squares (like in the binary powering algorithm) rather than multiplying two subproducts corresponding to two different subtrees. ◁

2.3.2 General case

The same idea applies to general P-finite sequences after reducing to a first-order matrix recurrence and clearing denominators. More precisely, we consider the recurrence (REC) with $b_i \in \mathbb{Z}[X]$. Given $N \in \mathbb{N}$ and $u_0, \ldots, u_{s-1}$, we want to compute $u_N \in \mathbb{Q}$.

We can rewrite the recurrence in matrix form as

$$\begin{bmatrix} u_{n+1} \\ \vdots \\ u_{n+s-1} \\ u_{n+s} \end{bmatrix} = -\frac{1}{b_s} \underbrace{\begin{bmatrix} & -b_s & & \\ & & \ddots & \\ & & & -b_s \\ b_0 & \cdots & \cdots & b_{s-1} \end{bmatrix}}_{B(n) \in \mathbb{Z}[n]_{\leqslant d}^{s \times s}} \underbrace{\begin{bmatrix} u_n \\ \vdots \\ u_{n+s-2} \\ u_{n+s-1} \end{bmatrix}}_{U_n}. \tag{MREC}$$

Then we have

$$U_N = \frac{(-1)^N}{b_s(N-1)\cdots b_s(0)}\, B(N-1)\cdots B(1)\, B(0)\, U_0.$$

Algorithm 7.

1. Compute the "matrix factorial" $B(N-1)\cdots B(1)\,B(0)$ using a product tree.
2. Compute the "polynomial factorial" $b_s(N-1)\cdots b_s(1)\,b_s(0)$ using a product tree.
3. Multiply by U_0, divide. ◁

Observe that for a fixed polynomial p, the bit size of $p(n)$ for $n \in \mathbb{N}$ is $O(\log n)$, and for any matrices $M_1, M_2 \in \mathbb{Z}^{s\times s}$ (s fixed) with entries of bit size $\leqslant h$, the bit size of the entries of $M_1 M_2$ is $\leqslant 2h + O(1)$. This implies that the product of 2^k of the matrices at the leaves of the product tree at step 1 has entries of bit size $\leqslant C\,2^k \log N$ for some constant C depending only on the recurrence. Thus, for a *fixed* recurrence, the analysis done in the case of $N!$ still applies and the cost of step 1 is

$$O(M_{\mathbb{Z}}(N \log N)\log N),$$

where the $O(\cdot)$ hides a dependency on the order and degree of the recurrence and the bit size of its coefficients. A simpler version of the same argument applies to step 2.

Exercise 10. What is the complexity of the final step, assuming that the initial values $u_0, \ldots, u_{s-1}$ have constant size? *(solution at the end)* ◁

Theorem 8. If $(u_n) \in \mathbb{Q}^{\mathbb{N}}$ is a P-finite sequence given by a nonsingular recurrence with coefficients in $\mathbb{Z}[X]$, one can compute u_N in $O(M_{\mathbb{Z}}(N \log N)\log N)$ bit operations. ◁

Remark. It is important (even for the complexity analysis, and even more in practice) to keep the denominators separate: if we use the binary splitting method to multiply matrices of (normalized) rational numbers, the cost of computing gcds to reduce the intermediate results dominates. ◁

Example 9. One can compute the first t binary digits of $e = \exp(1)$ in $O(M_{\mathbb{Z}}(t \log_2 t))$ bit operations. Indeed, let

$$e_N = \sum_{k=0}^{N-1} \frac{1}{k!}.$$

One has

$$\begin{aligned} |e_N - e| &\leqslant \frac{1}{N!} \sum_{k=0}^{\infty} \frac{1}{(N+1)\cdots(N+k)} \\ &\leqslant \frac{e}{N!} \\ &\leqslant 2^{-t} \quad \text{for } N = \frac{t + o(t)}{\log_2 t}. \end{aligned}$$

The sequence $u_N = \frac{1}{N!}$ satisfies $(N+1)\,u_{N+1} = u_N$, and $u_N = e_{N+1} - e_N$, so

$$(N+1)\,(e_{N+2} - e_{N+1}) = e_{N+1} - e_N,$$

and one can compute e_N in

$$\mathrm{O}(\mathrm{M}_{\mathbb{Z}}(N\log^2 N)) = \mathrm{O}(\mathrm{M}_{\mathbb{Z}}(t\log t)) \quad \text{bit ops.} \qquad \triangleleft$$

This use of product trees for computing partial sums of series is classically called *binary splitting*.

The same method applied to the series

$$\sum_{n=0}^{\infty} (-1)^n \frac{(6n)!}{(3\,n)!\,n!^3}\frac{(a\,n+b)}{c^{3n}} = \frac{c^{3/2}}{12\,\pi}, \qquad \begin{cases} a = 545140134 \\ b = 13591409 \\ c = 640320, \end{cases}$$

yields a t-bit approximation of π in $\mathrm{O}(\mathrm{M}(t\log^2 t))$ bit operations and is behind all record computations of π in the last few decades. Asymptotically, this is not the fastest known way to compute π (one can do it in $\mathrm{O}(\mathrm{M}(t)\log t)$ bit operations), but it is currently the best in practice.

2.4 Computing the Nth term of a P-finite sequence: arithmetic complexity

Algorithm 7 has quasi-optimal bit complexity with respect to N but still performs a linear number of arithmetic operations and hence is no faster than the naïve iterative method in terms of arithmetic complexity. Can we do better?

2.4.1 Case of $N!$

Again, we start with the case of the factorial, which already contains the key idea. Suppose for a moment $N = \ell^2$, and write

$$N! = \underbrace{\underbrace{1\times 2\times\cdots\times \ell}\times\underbrace{(\ell+1)\times(\ell+2)\times\cdots\times(2\,\ell)}\times\cdots\times\underbrace{(\ell^2-\ell+1)\times\cdots\times\ell^2}}_{\sqrt{N}\text{ blocks of }\sqrt{N}\text{ factors}}.$$

The interesting thing here is that the ℓ blocks all have the same structure: a certain polynomial, here $F(X) = (X+1)\cdots(X+\ell)$, evaluated at a point depending on the block. This means that we can organize the computation as a *baby steps-giant steps* algorithm that first computes F as a polynomial and then uses fast multipoint evaluation to evaluate all ℓ blocks simultaneously.

Algorithm 10. Input: N, output: $N!$.

1. Let $\ell = \lfloor\sqrt{N}\rfloor$.
2. Compute $F(X) = (X+1)\cdots(X+\ell) \in \mathbb{K}[X]$. (*baby steps*)
3. Evaluate F at $0, \ell, 2\,\ell, \ldots, (\ell-1)\,\ell$. (*giant steps*)
4. Multiply: $F(0)\cdot F(\ell)\cdots F((\ell-1)\,\ell)\cdot(\ell^2+1)\cdots N$.
5. Return the product. $\triangleleft$

Exercise 11.

1. Explain how to implement step 2 so that it runs in $\mathrm{O}(\mathrm{M}(\ell)\log\ell)$ operations in $\mathbb{K}$ (use a balanced product tree, that is, split the product in two and recurse).

2. Argue that step 3 can be carried out in $O(M(\ell)\log\ell)$ operations and step 4 in $O(\ell)$ operations.
3. Conclude that one can compute $N!$ viewed as an element of $\mathbb{K}$ in $O(M(\sqrt{N})\log N)$ operations in $\mathbb{K}$. ◁

The main idea behind this algorithm also leads to a simple proof of the following result.

Theorem. There is a *deterministic* integer factoring algorithm that runs in time $\tilde{O}(N^{1/4})$. ◁

The exponent $1/4$ was the best known until 2020. A series of results by Hittmeir and Harvey in 2020 brought it down to $1/5$. On October 29, 2026, Rudich and Rousseau posted a machine-generated proof that integers could be factored in deterministic time $N^{o(1)}$ as a consequence of the refutations of the Exponential Time Hypothesis claimed by OpenAI and Anthropic a few days before.

Proof. To start with, if N is composite, it must have a factor that is also a factor of $\lfloor\sqrt{N}\rfloor!$. One can compute $\lfloor\sqrt{N}\rfloor! \operatorname{rem} N$ in $\tilde{O}(N^{1/4})$ arithmetic operations over $\mathbb{Z}/N\mathbb{Z}$ by the above algorithm[3], and take the gcd of the result with N to get a factor. This does not quite suffice in general because the gcd may be N. However, consider the polynomial $F(X)$ defined as above but with $\ell = \lceil N^{1/4}\rceil$. Then, if N is composite, at least one of the integers $F(0), F(\ell), \ldots, F((\ell-1)\,\ell)$ must have a common factor with N. We compute these values mod N by multipoint evaluation as before, and instead of multiplying them together we take ℓ separate gcds with N. If one of the $\gcd(N, F(k\,\ell) \operatorname{rem} N)$ for $0 \leqslant k < \ell$ is nontrivial, we have found a proper factor, and if one of them is N, we can find a proper factor in another $O(N^{1/4})$ steps by taking gcds with the individual factors or by trial division by the primes in the range. Overall, the method does $\tilde{O}(N^{1/4})$ arithmetic operations in $\mathbb{Z}/N\mathbb{Z}$ and $O(N^{1/4})$ gcds of integers $\leqslant N$, each of which takes $\tilde{O}(\log N)$ *bit* operations. □

2.4.2 *General P-finite sequences*

As before, the idea to generalize the previous algorithm is to rewrite the recurrence as a first-order matrix recurrence (MREC) and compute U_N as

$$U_N = \frac{(-1)^N}{b_s(N-1)\cdots b_s(1)\,b_s(0)}\,B(N-1)\cdots B(1)\,B(0)\,U_0.$$

At a high level, the algorithm is as follows.

Algorithm 11.

1. Compute the "matrix factorial" $\Pi = B(N-1)\cdots B(1)\,B(0)$ as in Algorithm 10 (see below).
2. Same for the "polynomial factorial" $\delta = b_s(N-1)\cdots b_s(0)$.
3. Return $(-1)^N \Pi \cdot [u_0, \ldots, u_{s-1}]^{\mathrm{T}} / \delta$. ◁

There are some minor complications compared with the case of $N!$. Firstly,

$$F(X) = B(X+\ell-1)\cdots B(X+1)\,B(X)$$

3. $\mathbb{Z}/N\mathbb{Z}$ is not a field, but if we ever have to divide by a noninvertible element, we have a factor of N for free.

is an $s \times s$ matrix of polynomials *of degree* $\leqslant \ell \cdot d$. To apply multipoint evaluation, we want to evaluate these polynomials at $m \approx \ell \cdot d$ points. In other words, we want to have $m \approx \ell \cdot d$ blocks of ℓ factors, and so we take $\ell \approx \sqrt{N/d}$ instead of $\ell \approx \sqrt{N}$.

Secondly, computing $B(X+1), B(X+2), \ldots$ from $B(X)$ is no longer trivial.

Exercise 12. Adapt Algorithm 10 to perform step 1 of Algorithm 11 in

$$\mathrm{O}\big(s^\theta \mathrm{M}\big(\sqrt{Nd}\big)\log(Nd)\big)$$

operations in $\mathbb{K}$. You may assume without proof that, given a polynomial $P \in \mathbb{K}[X]$ of degree $<d$ and a point $a \in \mathbb{K}$, one can compute the *Taylor shift* $P(X+a)$ in $\mathrm{O}(\mathrm{M}(d)\log d)$ operations.

(solution at the end) ◁

To be complete, we still need to prove the fact we took for granted in Exercise 12. Here is a simple method that fits within the required complexity bound. One can actually do better, at least in characteristic zero.

Algorithm 12. (Assuming w.l.o.g. that $d = 2^k$.)

1. Compute $(x+a), (x+a)^2, \ldots, (x+a)^{2^{k-1}}$	$O(M(d))$
2. Compute $P(x+a)$ using the following recursive subroutine.	$O(M(d)\log(d))$
Given $Q(x)$ of degree less than δ:	$C(\delta) = 2\,C(\delta/2) + O(M(\delta))$
a. Write $Q(x) = Q_0(x) + x^{\delta/2} Q_1(x)$ with $\deg Q_0, \deg Q_1 < \delta/2$	$O(1)$
b. Recursively compute $Q_0(x+a)$, $Q_1(x+a)$	$2\,C(\delta/2)$
c. Deduce $Q(x+a)$	$O(M(\delta))$ ◁

In conclusion, we have the following result, which is still far from the obvious lower bound, but already a nice improvement over the naïve method. An important drawback in practice of both this algorithm and Algorithm 7 above, though, is that they force us to work with a complete basis of solutions even if we are only interested in a single one.

Theorem 13. One can compute the Nth term of a P-finite sequence given by a nonsingular recurrence of order s and degree d over $\mathbb{K}$ in $\mathrm{O}\big(s^\theta \mathrm{M}\big(\sqrt{Nd}\big)\log(Nd)\big)$ operations in $\mathbb{K}$. ◁

2.5 Summary

		C-finite	P-finite
first N terms	arith	$\mathrm{O}(N)$	$\mathrm{O}(N)$
	bit	$\mathrm{O}(N^2)$	$\tilde{\mathrm{O}}(N^2)$
Nth term	arith	$\mathrm{O}(\log N)$	$\tilde{\mathrm{O}}(\sqrt{N})$
	bit	$\mathrm{O}(\mathrm{M}(N))$	$\tilde{\mathrm{O}}(N)$

3 Linear differential equations with polynomial coefficients

From now on, we assume that $\mathbb{K}$ has characteristic zero.

3.1 D-finite functions

Definition 14. A "function" f is called *differentially finite* (also differentia*b*ly finite, D-finite, holonomic) over $\mathbb{K}$ when it satisfies a linear homogeneous differential equation

$$a_r(x) f^{(r)}(x) + \cdots + a_1(x) f'(x) + a_0(x) f(x) = 0 \tag{DEQ}$$

with coefficients $a_0, \ldots, a_r \in \mathbb{K}[x]$ and with $a_r \neq 0$. ◁

Here by a "function" we mean for instance a formal power series $f \in \mathbb{K}[[x]]$, a formal Laurent series[4] $f \in \mathbb{K}((x))$, a sufficiently differentiable function $f\colon \mathbb{R} \to \mathbb{R}$ or $f\colon \mathbb{C} \to \mathbb{C}$, or more generally anything for which the equation makes sense (technically, an element of a differential module over $\mathbb{K}[x]$).

Exercise 13. Check that the following "functions" are D-finite:

$$\exp(x), \qquad x^2 + 5\,x^3 + x^{12}, \quad \arctan(x), \qquad \sum_{k=0}^{\infty} k!\, x^k.$$

(The first three are D-finite both as formal power series and as analytic functions. The final one is a power series with zero radius of convergence, so it does not define an analytic function, but is still formally D-finite.) ◁

We will later see theorems that imply that more complicated expressions like

$$\frac{\sin(x) + \exp(x)^2}{\sqrt[5]{x^7 + 1}}$$

define D-finite functions. We will also see that, in contrast, $\tan(x)$ or $\sum_{k=0}^{\infty} 2^{k!}\, x^k$ are not D-finite.

The solutions of (DEQ) form a $\mathbb{K}$-vector space.

In the case of "normal" functions, you likely have encountered one version or another of the classical existence theorem saying that this space has dimension equal to the order r of the equation and that solutions are characterized by initial values at any point where the leading coefficient a_r does not vanish. Here we quote without proof the version for complex analytic functions.

4. The ring of *formal Laurent series* in the variable x over a ring R, denoted $R((x))$, is the set of formal objects of the form $\sum_{n=N_0}^{\infty} a_n x^n$ with $a_n \in R$ for some $N_0 \in \mathbb{Z}$, equipped with the operations + and × implied by the notation. Observe that $\mathbb{K}((x))$ is a field (and is the fraction field of $\mathbb{K}[[x]]$) when $\mathbb{K}$ is a field. Elements of $\mathbb{K}(x)$ can be expanded in Laurent series at any point of $\mathbb{K}$, giving several ways of embedding $\mathbb{K}(x)$ into $\mathbb{K}((x))$. By default, we identify rational functions with their Laurent series expansions at zero.

Also note that formal Laurent series are not the usual Laurent series from complex analysis: the latter allows for infinitely many terms of negative index, so that multiplication is only properly defined for convergent series.

Theorem 15. Suppose $a_r(0) \neq 0$ in the equation

$$a_r(x) f^{(r)}(x) + \cdots + a_1(x) f'(x) + a_0(x) f(x) = 0, \qquad a_0, \ldots, a_r \in \mathbb{C}[x]. \tag{2}$$

Then there exists a neighborhood $U \subseteq \mathbb{C}$ of 0 such that, for any $(v_0, \ldots, v_{r-1}) \in \mathbb{C}^r$, the differential equation (2) has a unique analytic solution with

$$f(0) = v_0, \quad f'(0) = v_1, \quad \ldots, \quad f^{(r-1)}(0) = v_{r-1}. \qquad \triangleleft$$

It is not too hard to see that a function that is analytic at zero is D-finite if and only if its series expansion is D-finite, so this also gives a roundabout proof that, when $\mathbb{K} = \mathbb{C}$ and $a_r(0) \neq 0$, the equation (DEQ) admits r linearly independent series solutions.

Exercise 14. Show that a D-finite meromorphic function has finitely many poles. Deduce that $\tan(x)$ is not D-finite. $\triangleleft$

Unlike what we saw before in the case of recurrences, here, for the most common types of "functions" at least[5], we can consider coefficients in $\mathbb{K}(x)$ instead of $\mathbb{K}[x]$ and freely multiply or divide the whole equation by nonzero polynomials without changing the solution space (if the "functions" are elements of $\mathbb{K}[[x]]$, we implicitly view them as Laurent series to be able to divide by polynomials that vanish at 0). When the space of "functions" is a vector space over $\mathbb{K}(x)$, like $\mathbb{K}((x))$ or the space of complex meromorphic functions, a differential equation of the form (DEQ) is precisely a $\mathbb{K}(x)$-linear relation between $f, f', \ldots, f^{(r)}$, leading to the following reformulation of the definition.

Proposition 16. Let $\mathscr{F}$ be a vector space over $\mathbb{K}(x)$ (equipped with a derivation compatible with that of $\mathbb{K}(x)$). A "function" $f \in \mathscr{F}$ is differentially finite if and only if the subspace

$$\operatorname{span}_{\mathbb{K}(x)}(f, f', f'', \ldots)$$

generated by f and all its derivatives is finite-dimensional (over $\mathbb{K}(x)$). $\triangleleft$

3.2 Differential equations and recurrences

Theorem 17. A power series is D-finite if and only if its coefficient sequence is P-finite. $\triangleleft$

In other words, D-finite functions are to P-recursive sequences what rational functions are to C-finite sequences[6].

Exercise 15. Show that $\sum_{k=0}^{\infty} 2^{k!} x^k$ is not D-finite. (Recall Exercise 3.) $\triangleleft$

Exercise 16. The goal of this exercise is to prove Theorem 17.

5. This is no longer true if the solutions we are interested in are distributions rather than normal functions. On the other hand, on the recurrence side, it becomes legitimate to multiply or divide a recurrence by a polynomial in n if we look for solutions as analytic functions or as germs of sequences as $n \to \infty$ rather than sequences.

6. In fact, allowing for a polynomial right-hand side in Definition 14 does not change the class of D-finite functions, and Theorem 17 easily generalizes to inhomogeneous equations. The result on C-finite sequences and rational functions then becomes the special case of differential equations of order zero.

It turns out to be convenient to prove a slightly more general version: a formal double-sided power series $\sum_{n\in\mathbb{Z}} c_n x^n$ is D-finite if and only if its coefficient sequence $(c_n)_{n\in\mathbb{Z}}$ is P-finite. We denote by $\mathbb{K}[[x^{\pm 1}]]$ the set (it is not a ring!) of double-sided power series over $\mathbb{K}$.

1. Convince yourself that the generalized statement makes sense and that it implies the theorem. (Recall Exercise 4.)
2. Consider the operators $X\colon \mathbb{K}[[x^{\pm 1}]] \to \mathbb{K}[[x^{\pm 1}]]$ defined by
$$(X(f))(x) = xf(x)$$
and $S\colon \mathbb{K}^{\mathbb{Z}} \to \mathbb{K}^{\mathbb{Z}}$ defined by
$$S((u_n)_{n\in\mathbb{Z}}) = (u_{n+1})_{n\in\mathbb{Z}}.$$
Show that S is invertible and $X(\sum_{n\in\mathbb{Z}} c_n x^n) = \sum_{n\in\mathbb{Z}} (S^{-1}(c))_n x^n$.
3. Now consider $D\colon \mathbb{K}[[x^{\pm 1}]] \to \mathbb{K}[[x^{\pm 1}]]$ and $N\colon \mathbb{K}^{\mathbb{Z}} \to \mathbb{K}^{\mathbb{Z}}$ defined by
$$D(f) = f', \qquad N((u_n)_{n\in\mathbb{Z}}) = (n\,u_n)_{n\in\mathbb{Z}}.$$
Show that $D(\sum_{n\in\mathbb{Z}} c_n x^n) = \sum_{n\in\mathbb{Z}} (((N+1)\circ S)(c))_n x^n$.

With these notations, the differential equation (DEQ) can be rewritten as

$$[a_r(X)\,D^r + \cdots + a_1(X)\,D + a_0(X)](f) = 0 \tag{3}$$

where operators are multiplied by composition. Similarly, the recurrence

$$\forall n\in\mathbb{Z}, \quad b_s(n)\,u_{n+s} + \cdots + b_1(n)\,u_{n+1} + b_0(n)\,u_n = 0$$

becomes

$$[b_s(N)\,S^s + \cdots + b_1(N)\,S + b_0(N)]((u_n)_{n\in\mathbb{Z}}) = 0.$$

4. Prove that $f(x) = \sum_{n=-\infty}^{\infty} c_n x^n$ satisfies (3) if and only if
$$\underbrace{[a_r(S^{-1})\,((N+1)\,S)^r + \cdots + a_1(S^{-1})\,((N+1)\,S) + a_0(S^{-1})]}_{R}((c_n)_{n\in\mathbb{Z}}) = 0.$$
5. Prove that $SN = (N+1)\,S$ and $S^{-1}N = (N-1)\,S^{-1}$. Deduce that
$$R = S^{-\delta}\,[b_s(N)\,S^s + \cdots + b_1(N)\,S + b_0(N)]$$
for some $\delta\in\mathbb{N}$, $s\in\mathbb{N}$, and $b_0,\ldots,b_s\in\mathbb{K}[T]$ with $b_s\neq 0$. Conclude that the coefficient sequence of a D-finite function is P-finite.
6. Convince yourself that the converse can be proved by a very similar reasoning. ◁

The proof gives a conversion algorithm: to compute the recurrence associated to a differential equation, it suffices to replace D by $(N+1)\,S$ and X by S^{-1} in the operator form (3), then move the powers of S to the right of powers of N using the commutation rules $SN = (N+1)\,S$ and $S^{-1}N = (N-1)\,S^{-1}$.

Note that a differential equation of order $\leqslant r$ and degree $\leqslant d$ becomes a recurrence of order $\leqslant d+r$ and degree $\leqslant r$.

Exercise 17. This may seem counterintuitive: didn't we just say that differential equations correspond to recurrences, and aren't solutions of both defined by initial values in number equal to their order? Where is the trick? *(solution at the end)* ◁

Theorem 17 also means that the results of the previous sections apply to the computation of coefficients of D-finite series.

Corollary 18. One can compute the first N terms of any fixed D-finite power series over $\mathbb{K}$ in $O(N)$ operations in $\mathbb{K}$. ◁

There is a subtlety here: we haven't specified exactly how to encode "a D-finite series" on the computer, since we need not only a differential equation but also a convenient set of initial values. In fact, when the leading coefficient of the differential equation vanishes at 0, the first r terms of a series solution may not be enough to pinpoint it. As a very related issue, we need to explain what to do if the leading coefficient of the associated recurrence has an integer root… But what we can do in any case is give initial terms to get past the largest integer root of the leading coefficient of the recurrence as part of the input together with the differential equation. For a fixed D-finite series, this is $O(1)$ terms anyway. The rest is (maybe) a story for one of the next lectures!

Exercise 18. Explain how to compute the coefficient of x^{2N} in

$$f(x) = (1+x)^{2N}(1+x+x^2)^N$$

for $N = 10^6$. Even better, compute it! *(solution at the end)* ◁

Hints and solutions

Exercise 3. Any C-finite sequence is P-finite, including $(2^n)_{n\in\mathbb{N}}$. The factorials form a P-finite sequence since $(n+1)! = (n+1)\,n!$ for all $n \in \mathbb{N}$ (this is a nonsingular recurrence of order 1). The sequence $(2^{2^n})_{n\in\mathbb{N}}$ grows too fast: any P-finite sequence is bounded by $C\,n!^\kappa$ for some constants C and κ. ◁

Exercise 9. The sum over all recursive calls is

$$\begin{aligned}\sum_{i=0}^{\log_2(N)-1} 2^i \mathrm{M}_{\mathbb{Z}}\Big(\frac{N}{2^{i+1}}(1+\log_2 N)\Big) &\leqslant \frac{1}{2}\sum_{i=0}^{\log_2(N)-1} \mathrm{M}_{\mathbb{Z}}(N(1+\log_2 N)) \\ &= \frac{1}{2}\mathrm{M}_{\mathbb{Z}}(N(1+\log_2 N))\log_2 N\end{aligned}$$

where the inequality comes from the superlinearity assumptions on $\mathrm{M}_{\mathbb{Z}}(\cdot)$. ◁

Exercise 10. The last step hides a gcd of integers of size $\Theta(N\log N)$ in the worst case, so its complexity is $O(\mathrm{M}(N\log N)\log N)$ as well, but (with the best currently known algorithms) no better. ◁

Exercise 12.

1. Set $\ell = \lfloor \sqrt{N/d} \rfloor$ and $m = \lfloor N/\ell \rfloor$.
2. Compute $B(X+1), B(X+2), \dots, B(X+\ell-1)$. $O(s^2\, \ell\, M(d) \log d)$
3. Compute $F(X) = B(X+\ell-1) \cdots B(X+1)\, B(X)$. $O(s^\theta M(\ell\, d) \log \ell)$
4. Compute $F(0), F(\ell), \dots, F((m-1)\,\ell)$. $O(s^2 M(m) \log m)$
5. Deduce $B(N-1) \cdots B(m\,\ell) \cdot F((m-1)\,\ell) \cdots F(0)$. $O(s^\theta m)$

Total $O\big(s^\theta M\big(\sqrt{Nd}\big) \log(Nd)\big)$ since $m \approx \ell\, d \approx \sqrt{Nd}$. ◁

Exercise 17. The recurrence obtained by direct conversion of the differential equation must hold at all integers including negative ones. Its restriction to nonnegative indices may have additional solutions $(c_n)_{n\in\mathbb{N}}$ that do not extend to solutions $(c_n)_{n\in\mathbb{Z}}$ with $c_n = 0$ for large negative n and thus do not correspond to solutions of the differential equations. Additionally, nothing says that the recurrence is nonsingular. Solutions over $\mathbb{N}$ of *nonsingular* recurrences of order s with polynomial coefficients are characterized by their first s terms, but solutions of singular recurrences not necessarily. ◁

Exercise 18. One has

$$\frac{f'(x)}{f(x)} = 2N\frac{1}{1+x} + N\frac{2x+1}{1+x+x^2}.$$

Convert this differential equation into a recurrence and use Section 2.3. ◁