

Computing terms of P-finite sequences

Marc Mezzarobba

January 27, 2025

Doctoral funding available

1

Group MAX team, École polytechnique

Starting date Fall 2025 (negotiable)

Topic Differential equations

From computational complexity and differential Galois theory
to low-level implementation details depending on student interests

Advisors J. van der Hoeven + one of { G. Lecerf, M. Mezzarobba,
F. Ollivier, G. Pogudin }

Talk to us ASAP if interested — Tell your friends

1 Introduction

Theorem. Let $f = \sum_{n \geq 0} f_n x^n \in \mathbb{K}[[x]]$.

f is **D-finite**

$\Leftrightarrow$

$(f_n)_{n \in \mathbb{N}}$ is **P-finite** / **P-recursive**

$\Leftrightarrow f$ satisfies a linear ODE
with coefficients in $\mathbb{K}[x]$

$\Leftrightarrow (f_n)$ satisfies a linear recurrence
with coefficients in $\mathbb{K}[n]$

$\Leftrightarrow \dim \operatorname{span}_{\mathbb{K}(x)}(f, f', f'', \dots) < \infty$

Corollary. One can compute the first N terms of a D-finite series in $O(N)$ ops.

"ops" = operations in the base field $\mathbb{K}$

Definition. A sequence $(u_n) \in \mathbb{K}^{\mathbb{N}}$ is called **C-finite** when it satisfies a linear recurrence

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

Theorem. One can compute the N th term of a C-finite sequence

- in $O(s^\theta \log(N))$ ops by binary powering on the companion matrix,
- in $O(M(s) \log(N))$ ops by binary powering modulo charpoly
or by repeated Gräffe transforms.

Remarks.

- Over $\mathbb{Z}$, all three methods take $O(M_{\mathbb{Z}}(N))$ **bit** operations.
- They do not work in the P-finite case.

Definition. A **(generalized) hypergeometric series** is a power series whose coefficient sequence satisfies a **first-order** recurrence relation with polynomial coefficients:

$$f(x) = \sum_{n=0}^{\infty} u_n x^n \quad \text{where} \quad u_{n+1} = \frac{p(n)}{q(n)} u_n, \quad u_0 = 1.$$

For $p, q \in \mathbb{Z}[n]$ and $x \in \mathbb{Q}$, one can compute $\sum_{n=0}^{N-1} u_n x^n$
in $O(M_{\mathbb{Z}}(N \log(N)^2))$ **bit** operations

by splitting $\sum_0^{N-1}$ as $\sum_0^{m-1} + \sum_m^{N-1} = \frac{T(0, m)}{Q(0, m)} + \frac{T(m, N)}{Q(m, N)} u_m$
and computing the numerators & denominators recursively

C-Finite sequences: The direct algorithm over $\mathbb{Z}$

6

$$u_{n+s} + c_{s-1} u_{n+s-1} + \cdots + c_0 u_n = 0$$

$$u_n = \alpha_1^n p_1(n) + \cdots + \alpha_k^n p_k(n)$$

Direct algorithm:

$$\begin{cases} u_s &:= -(c_{s-1} u_{s-1} + \cdots + c_0 u_0) \\ u_{s+1} &:= -(c_{s-1} u_s + \cdots + c_0 u_1) \\ \vdots & \end{cases} \quad |u_n| \leq 2^{Kn}$$

Bit operations:

$$\sum_{n=s}^{N-1} C M_{\mathbb{Z}}(h, K n) \leq C' \frac{N(N-1)}{2} \quad \text{for a fixed rec.}$$
$$= O(N^2)$$

Output size can reach $\Omega(N^2)$ for N terms
 $\Omega(N)$ for one term

Proposition. Let $(u_n)_{n \in \mathbb{N}}$ satisfy a linear recurrence with constant coefficients and unit leading term. Assume $u_0 = 1$.

Given $N \in \mathbb{N}$, one can compute u_N in $O(M_{\mathbb{Z}}(N))$ bit operations.

Proof. Write

$$\begin{pmatrix} u_{n+1} \\ u_{n+2} \\ \vdots \\ u_{n+s} \end{pmatrix} = \underbrace{\begin{pmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ -c_0 & -c_1 & \cdots & -c_{s-1} \end{pmatrix}}_{A \in \mathbb{Z}^{s \times s}} \begin{pmatrix} u_n \\ u_{n+1} \\ \vdots \\ u_{n+s-1} \end{pmatrix}.$$

- $\|A^n\| \leq \|A\|^n \leq 2^{Kn}$ for some $K > 0$.
- Cost of binary powering:

$$C \cdot M_{\mathbb{Z}}(K) + \cdots + C \cdot M_{\mathbb{Z}}\left(\frac{N}{4} K\right) + C \cdot M_{\mathbb{Z}}\left(\frac{N}{2} K\right) = O(M_{\mathbb{Z}}(N)).$$

□

Direct computation of $N!$

Algorithm. Repeat $u_n = n \cdot u_{n-1}$ for $n = 1, 2, \dots, N$.

- Arithmetic complexity: $O(N)$ ops Optimal if computing $1!, \dots, N!$
- Bit complexity:

$$\text{size}(n!) = 1 + \lfloor \log_2(n!) \rfloor = n \log_2 n + O(n) \quad (\text{Stirling})$$

Step n is a multiplication of

$$n \log_2 n + O(n) \quad \text{by} \quad \log_2 n + O(1) \quad \text{bits}$$

costing $n M_{\mathbb{Z}}(\log_2 n) + O(n)$ bit operations if done by blocks.

$$\text{Total cost:} \quad \sum_{n=1}^N (n M_{\mathbb{Z}}(\log_2 n) + O(n)) = \frac{N^2}{2} M_{\mathbb{Z}}(\log_2 N) + O(N^2) \text{ bit ops}$$

Quasi-optimal for N terms, unsatisfactory for a single term

Nonsingular recurrences

Definition. We will say that the recurrence relation

$$b_s(n) u_{n+s} + \cdots + b_1(n) u_{n+1} + b_0(n) u_n = 0 \quad (\text{Rec})$$

is **nonsingular** if $b_s(n) \neq 0$ for all $n \in \mathbb{N}$.

Proposition. If (Rec) is nonsingular, then

- its solution space has dimension s ,
- any solution $(u_n)_{n \in \mathbb{N}}$ is determined by $(u_0, \dots, u_{s-1})$.

In other words: there is a basis of solutions of the form

$$\begin{aligned} u^{(0)} &= (1, 0, 0, \dots, 0, *, *, *, \dots) \\ u^{(1)} &= (0, 1, 0, \dots, 0, *, *, *, \dots) \\ &\vdots \\ u^{(s-1)} &= (0, 0, 0, \dots, 1, *, *, *, \dots) \end{aligned}$$

(We will study singular recurrences in the next lecture.)

$$b_s(n) u_{n+s} + \cdots + b_1(n) u_{n+1} + b_0(n) u_n = 0$$

Problems. Given a nonsingular recurrence as above, initial values $u_{0:s}$, and $N \in \mathbb{N}$:

a) Compute $(u_0, \dots, u_{N-1})$

b) Compute u_N

Complexity models: operations in $\mathbb{K}$ ("ops")
 binary operations for $\mathbb{K} = \mathbb{Z}$

Bit sizes: $O(n \log n)$ for a single u_n , $O(N^2 \log N)$ for $u_{0:N}$ (reached)

Direct algorithm: repeat $u_n = -\frac{1}{b_s(n-s)} (b_{s-1}(n-s) u_{n-1} + \cdots + b_0(n-s) u_{n-s})$

Arithmetic cost: $O(N)$ ops

Over $\mathbb{Z}$ with $b_s = 1$: $O(N^2 M_{\mathbb{Z}}(\log N))$ binops

Quasi-optimal (for a fixed rec.) for problem a) $\longrightarrow$ Focus on problem b)

2 Baby steps, giant steps

A baby steps-giant steps algorithm for $N!$

12

[Strassen 1976]

$$N! = \underbrace{1 \cdot 2 \cdots \ell \cdot (\ell + 1) (\ell + 2) \cdots (2\ell) \cdots (\ell^2 - \ell + 1) (\ell^2 - \ell + 2) \cdots \ell^2}_{N^{1/2} \text{ blocks of size } N^{1/2}} \quad \ell = N^{1/2}$$

Algorithm. *Input:* N *Output:* $N!$

1. Let $\ell = \lfloor N^{1/2} \rfloor$

2. Baby steps:

a. Compute $F = (x + 1) (x + 2) \cdots (x + \ell)$ $O(M(\ell) \log \ell)$

3. Giant steps:

a. Compute $P_0 = F(0), P_1 = F(\ell), P_2 = F(2\ell), \dots, P_{\ell-1} = F((\ell - 1)\ell)$
by multipoint evaluation

b. Return $P_0 P_1 \cdots P_{\ell-1} \cdot (\ell^2 + 1) \cdots (N - 1) N$

Idea: if N is composite, $\lfloor \sqrt{N} \rfloor! \wedge N$ is a nontrivial factor

Algorithm. *Input:* N *Output:* a nontrivial factor of N , or 1 if N is prime

1. Let $\ell = \lceil N^{1/4} \rceil$

2. Baby steps:

a. Compute $F = (x+1)(x+2)\cdots(x+\ell) \in (\mathbb{Z}/N\mathbb{Z})[x]$ $O(M(\ell) \log(\ell) M_{\mathbb{Z}}(h))$

3. Giant steps:

a. Compute $P_0 = F(0), \dots, P_{\ell-1} = F((\ell-1)\ell)$ by mulpt ev. $O(M(\ell) \log(\ell) M_{\mathbb{Z}}(h))$

b. Compute $P_0 \wedge N, \dots, P_{\ell-1} \wedge N$ $O(\ell M_{\mathbb{Z}}(h) \log(h))$

$h = 1 + \lfloor \log_2 N \rfloor$

Total $O(M(N^{1/4}) \log(N)^{2+o(1)})$

A TIME-SPACE TRADEOFF FOR LEHMAN'S DETERMINISTIC INTEGER FACTORIZATION METHOD

MARKUS HITTMER

ABSTRACT. Fermat's well-known factorization algorithm is based on finding a representation of natural numbers N as the difference of squares. In 1895, Lawrence generalized this idea and applied it to multiples kN of the original number. A systematic approach to choose suitable values for k has been introduced by Lehman in 1974, which resulted in the first deterministic factorization algorithm considerably faster than trial division. In this paper, we construct a time-space tradeoff for Lawrence's generalization and apply it together with Lehman's result to obtain a deterministic integer factorization algorithm with runtime complexity $O(N^{2/9+o(1)})$. This is the first exponential improvement since the establishment of the $O(N^{1/4+o(1)})$ bound in 1977.

1. INTRODUCTION

We consider the problem of computing the prime factorization of natural numbers N . There is a large variety of probabilistic and heuristic factorization methods achieving subexponential complexity. We refer the reader to the survey [Len00] and to the monographs [Rie94] and [Wag13]. The focus of the present paper is a more theoretical aspect of the integer factorization problem, which concerns *deterministic* algorithms

AN EXPONENT ONE-FIFTH ALGORITHM FOR DETERMINISTIC INTEGER FACTORISATION

DAVID HARVEY

ABSTRACT. Hittmeir recently presented a deterministic algorithm that provably computes the prime factorisation of a positive integer N in $N^{2/9+o(1)}$ bit operations. Prior to this breakthrough, the best known complexity bound for this problem was $N^{1/4+o(1)}$, a result going back to the 1970s. In this paper we push Hittmeir's techniques further, obtaining a rigorous, deterministic factoring algorithm with complexity $N^{1/5+o(1)}$.

1. INTRODUCTION

Let $F(N)$ denote the time required to compute the prime factorisation of an integer $N \geq 2$. By “time” we mean “number of bit operations”, or more precisely, the number of steps performed by a deterministic Turing machine with a fixed, finite number of linear tapes [Pap94]. All integers are assumed to be encoded in the usual binary representation.

In this paper we prove the following result:

Theorem 1.1. *There is an integer factorisation algorithm achieving*

$$F(N) = O(N^{1/5} \log^{16/5} N).$$

Generalization to P-recursive sequences

16

[Chudnovsky & Chudnovsky 1987]

Write the recurrence in matrix form, pull out the denominator:

$$\begin{pmatrix} u_{n+1} \\ \vdots \\ u_{n+s-1} \\ u_{n+s} \end{pmatrix} = \frac{1}{b_s(n)} \underbrace{\begin{pmatrix} & b_s(n) & & \\ & & \ddots & \\ & & & b_s(n) \\ -b_0(n) & -b_1(n) & \cdots & -b_{s-1}(n) \end{pmatrix}}_{B(n)} \underbrace{\begin{pmatrix} u_n \\ \vdots \\ u_{n+s-2} \\ u_{n+s-1} \end{pmatrix}}_{U_n}$$

Then

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

$B(n)$ = matrix of polynomials of degree $< d$

Algorithm. *Input:* $B \in \mathbb{K}[n]^{s \times s}$ of $\deg < d$, $N \in \mathbb{N}$ *Output:* $B(N-1) \cdots B(1) B(0)$

1. Write $N = \ell m$ with $\ell = (N/d)^{1/2}$ and $m = (N/d)^{1/2}$ (assumed exact for simplicity)

2. Baby steps:

a. Compute $B(X+1), \dots, B(X+\ell-1)$ $O(\ell M(d) \log(d) s^2)$

b. Compute $F(X) = B(X+\ell-1) \cdots B(X+1) B(X)$ $O(M(\ell d) \log(\ell) s^\theta)$

3. Giant steps:

a. Compute $F(0), F(\ell), \dots, F((m-1)\ell)$ simultaneously $O(M(m) \log(m) s^\theta)$

b. Deduce and return the product $F((m-1)\ell) \cdots F(\ell) F(0)$ $O(m s^\theta)$

$\deg F(X) < \ell d$ $\ell d \leq m$

Total $O\left(M(m) \log(m) s^\theta\right)$

naïvely step 2a takes $O(\ell d^2 s^2)$ ops

Exercise 1. Design an algorithm to compute $B(x+a)$ from $B(x)$ in $O(M(d) \log d)$ ops.

Nth term of a P-recursive sequence

Algorithm. *Notation as before.*

1. Compute $B(N-1) \cdots B(1) B(0)$ by the previous algorithm $O(M(m) \log(m) s^\theta)$
2. Compute $b_s(N-1) \cdots b_s(1) b_s(0)$ by the previous algorithm $O(M(m) \log(m))$
3. Divide, return $O(s^2)$

Theorem. Let $(u^{(0)}, \dots, u^{(s-1)})$ be the basis of solutions s.t. $u_i^{(j)} = \delta_{i,j}$ of a nonsingular recurrence of order s and degree $< d$. One can compute the matrix $(u_{N+i}^{(j)})_{i,j} \in \mathbb{K}^{s \times s}$ in

$$O\left(M(\sqrt{N}d) \log(Nd) s^\theta\right) \text{ ops.}$$

Corollary. One can compute the N th term of a P-recursive sequence given by a nonsingular recurrence in $O(M(\sqrt{N}) \log N)$ ops.

3 Binary splitting

Computing $N!$ in quasi-linear time

Algorithm. Use a product tree. That is, split the product as

$$N! = \underbrace{1 \cdot 2 \cdots m}_{P(0,m)} \cdot \underbrace{(m+1) \cdots N}_{P(m,N)}, \quad m = \lfloor N/2 \rfloor,$$

and recurse.

Using $\text{size}(P(\ell, h)) \leq 1 + (h - \ell) \log_2 N$, the cost $C(\ell, h)$ of computing $P(\ell, h)$ satisfies

$$C(\ell, h) \leq C(\ell, m) + C(m, h) + M_{\mathbb{Z}}(1 + \lceil (h - \ell)/2 \rceil \log_2 N) \quad m = \lfloor (\ell + h)/2 \rfloor.$$

The total cost of the multiplications at any given recursion depth is

$$\begin{aligned} &\leq \sum_i M_{\mathbb{Z}}(1 + \lceil H_i/2 \rceil \log_2 N) \quad \text{where} \quad \sum_i H_i \leq N \\ &\leq M_{\mathbb{Z}}\left(\frac{N}{2} \log_2 N + O(N)\right). \end{aligned}$$

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

Nth term of a P-recursive sequence

21

[Chudnovsky & Chudnovsky 1987]

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

Same idea as before:

write $U_n = (u_n, \dots, u_{n+s-1})$ and $U_N = \frac{1}{b_s(N-1) \cdots b_s(1) b_s(0)} B(N-1) \cdots B(1) B(0) U_0$

Algorithm.

1. Compute $B(N-1) \cdots B(1) B(0)$ by **binary splitting**
2. Compute $b_s(N-1) \cdots b_s(1) b_s(0)$ by **binary splitting**
3. Divide

Theorem. One can compute the N th term of a sequence $(u_n) \in \mathbb{Q}^{\mathbb{N}}$ given by a nonsingular recurrence with coefficients in $\mathbb{Z}[n]$ in bit operations.

Problem. Compute the coefficient of x^{2N} in

$$(1+x)^{2N} (1+x+x^2)^N.$$

Let $f(x) = (1+x)^{2N} (1+x+x^2)^N$. One has

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

Convert ODE to recurrence, use binary splitting.

The case of hypergeometric sums

Goal. Compute $\Sigma_N = \sum_{n=0}^{N-1} u_n$ where $u_{n+1} = \frac{p(n)}{q(n)} u_n$, $u_0 = 1$

Last week's version.

$$\text{Write } \sum_{n=\ell}^{h-1} u_n = \sum_{n=\ell}^{h-1} \frac{p(n-1) \cdots p(\ell)}{q(n-1) \cdots q(\ell)} u_\ell = \frac{T(\ell, h)}{Q(\ell, h)} u_\ell \quad \text{where } Q(\ell, h) = q(h-1) \cdots q(\ell)$$

$$u_h = \frac{P(\ell, h)}{Q(\ell, h)} u_\ell \quad P(\ell, h) = p(h-1) \cdots p(\ell)$$

$$\text{Then } \sum_{n=\ell}^{h-1} u_n = \sum_{n=\ell}^{m-1} u_n + \sum_{n=m}^{h-1} u_n \quad \text{gives} \quad \frac{T(\ell, h)}{Q(\ell, h)} u_\ell = \frac{T(\ell, m)}{Q(\ell, m)} u_\ell + \frac{T(m, h)}{Q(m, h)} \frac{P(\ell, m)}{Q(\ell, m)} u_\ell$$

$$T(\ell, h) = Q(m, h) T(\ell, m) + P(\ell, m) T(m, h).$$

Matrix version.

$$\begin{pmatrix} u_{n+1} \\ \Sigma_{n+1} \end{pmatrix} = \frac{1}{q(n)} \underbrace{\begin{pmatrix} p(n) & 0 \\ q(n) & q(n) \end{pmatrix}}_{B(n)} \begin{pmatrix} u_n \\ \Sigma_n \end{pmatrix}, \quad B(h-1) \cdots B(\ell) = \begin{pmatrix} P(\ell, h) & 0 \\ T(\ell, h) & Q(\ell, h) \end{pmatrix}$$

An exercise for next time

Exercise. Give an algorithm to convert an n -bit number from base 2 to base 10 in $O(M_{\mathbb{Z}}(n) \log n)$ bit operations, where $M_{\mathbb{Z}}(n)$ is a bound on the cost of n -bit integer multiplication.

4 Partial sums of D-finite series

Application to sums of D-finite series

26

Let $\Sigma_n = \sum_{k=0}^{n-1} u_k \xi^k$ for some fixed $\xi \in \mathbb{R}$.

If $(u_n)_{n \in \mathbb{N}}$ satisfies a rec. with poly. coeffs, then (Σ_n) too. (why?)

Better formulation:

$$\begin{pmatrix} u_{n+1} \xi^{n+1} \\ \vdots \\ u_{n+s} \xi^{n+1} \\ \Sigma_{n+1} \end{pmatrix} = \frac{1}{b_s(n)} \begin{pmatrix} \begin{pmatrix} B(n) \xi \\ b_s(n) \end{pmatrix} & \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \\ b_s(n) & 0 \cdots 0 \end{pmatrix} \begin{pmatrix} u_n \xi^n \\ \vdots \\ u_{n+s-1} \xi^n \\ \Sigma_n \end{pmatrix}$$

$$\begin{pmatrix} u_{n+1} \xi^{n+1} \\ \vdots \\ u_{n+s} \xi^{n+1} \\ \Sigma_{n+1} \end{pmatrix} = \frac{1}{b_s(n)} \begin{pmatrix} \begin{pmatrix} B(n) \xi \\ b_s(n) \end{pmatrix} & \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \\ b_s(n) & 0 \cdots 0 \end{pmatrix} \begin{pmatrix} u_n \xi^n \\ \vdots \\ u_{n+s-1} \xi^n \\ \Sigma_n \end{pmatrix}$$

Working with p -bit approximations and ignoring rounding errors:

$$\Sigma_N \text{ to } p\text{-bit precision in } O(M(\sqrt{N}) \log(N) M_{\mathbb{Z}}(p)) \text{ ops}$$

Target accuracy 2^{-t} typically requires $N = O(t)$ (geometric convergence)

If rounding errors negligible, working precision $p = t + O(1)$

$\rightsquigarrow$ evaluation of D-finite series to precision t in $\tilde{O}(t^{3/2})$ ops

Binary splitting for D-finite series

Again: $\sum_{k=0}^{n-1} u_k \xi^k$ satisfies a recurrence

(Note that ξ enters into the recurrence!)

The previous result on binary splitting yields:

Corollary. One can evaluate the N th partial sum of a **fixed** D-finite series at a **fixed** point $\xi \in \mathbb{Q}$ in $O(M(N \log^2 N))$ bit operations.

Typical case: $N = O(t)$

t = target bit accuracy

Application: High-precision evaluation of classical constants ²⁹

- $e = \exp(1)$ with error $\leq 2^{-t}$ in $O(M(t \log t))$ bit operations

$$e = \sum_{k=0}^{n-1} \frac{1}{k!} + \underbrace{\frac{1}{n!} \sum_{k=0}^{\infty} \frac{1}{(n+1) \cdots (n+k)}}_{\leq e/n!}, \quad \frac{e}{n!} \leq 2^{-t} \text{ for } n = \frac{t + o(t)}{\log_2 t}$$

Cost of the binary splitting method: $O\left(M\left(\frac{t}{\log_2 t} \log\left(\frac{t}{\log_2 t}\right)^2\right)\right) = O(M(t \log t))$.

- $\ln(2)$ in $O(M(t \log(t)^2))$ bit operations: $\ln(2) = -\ln(1 + \xi)$ with $\xi = -\frac{1}{2}$

Radius of convergence = 1 $\Rightarrow$ general term = $O(2^{-k}) \Rightarrow$ need $O(t)$ terms.

$$\bullet \frac{1}{\pi} = \frac{12}{c^{3/2}} \sum_{n=0}^{\infty} (-1)^n \frac{(6n)!}{(3n)! n!^3} \frac{(a n + b)}{c^{3n}}, \quad \begin{cases} a = 545140134 \\ b = 13591409 \\ c = 640320 \end{cases} \quad [\text{Chudnovsky}^2 \text{ 1987}]$$

1 hypergeometric series, 1 square root, 1 division

Used in record computations — although another algo. yields t digits of π in only $O(M(t) \log t)$ bit ops
[Salamin 1976, Brent 1978]

Dependency on the evaluation point

$$\begin{pmatrix} u_{n+1} \xi^{n+1} \\ \vdots \\ u_{n+s} \xi^{n+1} \\ \Sigma_{n+1} \end{pmatrix} = \frac{1}{b_s(n)} \begin{pmatrix} \begin{pmatrix} B(n) \xi \\ b_s(n) \end{pmatrix} & \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \\ b_s(n) & 0 \cdots 0 \end{pmatrix} \begin{pmatrix} u_n \xi^n \\ \vdots \\ u_{n+s-1} \xi^n \\ \Sigma_n \end{pmatrix}$$

If ξ is of bit size h , then (for a fixed differential equation):

- the matrices, taken at $n \leq N$, have bit size $O(h + \log N)$,
- the cost of computing the product tree for N terms becomes

$$O\left(M\left(\overbrace{N(h + \log N)}^{\text{size of each row}}\right) \underbrace{\log N}_{\text{depth}}\right).$$

size of each leaf

If N and h are both $\Theta(t)$, the cost becomes quadratic in t !

5 The “bit-burst” method

Fast high-precision evaluation of the exponential function 32

[Brent 1976]

Goal: for a real number $\frac{1}{2} \leq \xi < 1$, compute $\exp(\xi)$ with error $\leq 2^{-t}$ in $\tilde{O}(t)$ bit ops.

We assume that a sufficiently accurate approximation of ξ is given ($t + O(1)$ bits suffice)

Write

$$\xi = 0.\underbrace{\xi_1}_{m_0}\underbrace{\xi_2\xi_3}_{m_1}\underbrace{\xi_4\xi_5\xi_6\xi_7}_{m_2}\underbrace{\xi_8\xi_9\xi_{10}\xi_{11}\xi_{12}\xi_{13}\xi_{14}\xi_{15}}_{m_{K-1}}\xi_{16}\xi_{17}\dots$$
$$= m_0 + m_1 + m_2 + \dots + m_{K-1}$$

where $\begin{cases} m_k \leq 2^{-2^k+1} \\ m_k \text{ fits on } 2^k \text{ bits} \end{cases}$

Then $\exp(\xi) = \exp(m_0) \exp(m_1) \dots \exp(m_{K-1})$ and $K = O(\log t)$

Algorithm. Evaluate each m_k by binary splitting, then multiply.

The final multiplications cost $O(M(t) \log t)$ in total.

Remark: can reduce to $\xi \in [1/2, 1)$ using $\exp(2x) = \exp(x)^2$.

$$\xi = m_0 + m_1 + m_2 + \cdots + m_{K-1} \quad \text{where } \begin{cases} m_k \leq 2^{-2^k+1} \\ m_k \text{ fits on } 2^k \text{ bits} \end{cases}$$

Computation of a single $\exp(m_k)$:

- Because $m_k \leq 2^{-2^k+1}$, only $N = O(2^{-k}t)$ terms of the series are needed
- Cost of binary splitting:

$$\begin{aligned} O\left(\underbrace{M\left(\underbrace{N}_{\text{size of each leaf}} \underbrace{(h + \log N)}_{\text{size of each row}}\right)}_{\text{depth}} \log N\right) &= O\left(\underbrace{M\left(2^{-k}t \underbrace{(2^k + \log t)}_{\text{size of each leaf}}\right)}_{\text{depth}} \log t\right) \\ &= O(M(t \log t + 2^{-k}t \log^2 t)) \end{aligned}$$

$$\text{Total: } \sum_{k=0}^{K-1} C M(t \log t + 2^{-k}t \log^2 t) \leq C \cdot M\left(\sum_{k=0}^{K-1} (t \log t + 2^{-k}t \log^2 t)\right) = O(M(t \log(t)^2))$$

Fix a differential operator L ; assume that 0 is an ordinary point.

Consider a basis $y_1, \dots, y_r$ of analytic solutions.

- Suppose that the series expansion of y_k converges on $\{|\xi| < \rho\}$.

Binary splitting $\rightsquigarrow y(\xi)$ for $|\xi| \leq \frac{1}{2}\rho$ of bit size $O(1)$ in $\tilde{O}(t)$ bit ops.

- Derivatives have the same radius of convergence, are still D-finite.

$\rightsquigarrow (y(\xi), y'(\xi), \dots, y^{(r-1)}(\xi))$ in $\tilde{O}(t)$ ops

- We can do that for $y_1, \dots, y_r \rightsquigarrow \begin{pmatrix} y_1(\xi) & \cdots & y_r(\xi) \\ \vdots & & \vdots \\ y_1^{(r-1)}(\xi) & \cdots & y_r^{(r-1)}(\xi) \end{pmatrix}$ in $\tilde{O}(t)$ ops

- By multiplying these matrices for steps corresponding to a decomposition

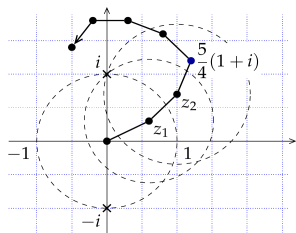
$$\xi = 0. \underbrace{\xi_1 \xi_2 \xi_3 \xi_4 \xi_5 \xi_6 \xi_7}_{\text{group 1}} \underbrace{\xi_8 \xi_9 \xi_{10} \xi_{11} \xi_{12} \xi_{13} \xi_{14} \xi_{15} \xi_{16} \xi_{17} \dots}_{\text{group 2}}$$

we can evaluate the solutions at complex points of bit size t in $\tilde{O}(t)$ ops.

Fast high-precision evaluation of D-finite functions (sketch) 35

[Chudnovsky & Chudnovsky 1987, van der Hoeven 1999, ...]

- By multiplying matrices
$$\begin{pmatrix} y_1(\xi) & \cdots & y_r(\xi) \\ \vdots & & \vdots \\ y_1^{(r-1)}(\xi) & \cdots & y_r^{(r-1)}(\xi) \end{pmatrix},$$
 we can also evaluate the (analytic continuation of) the solutions outside the disk $|\xi| < \rho$.



- For fixed ξ , computing $y(\xi)$ with an error $\leq 2^{-t}$ requires $O(t)$ digits of ξ .
- All necessary error bounds can be computed automatically.

Pseudo-theorem: “one can evaluate a **fixed** D-finite function at a **fixed** point $\in \mathbb{C}$ with an error $\leq 2^{-t}$ in $\tilde{O}(t)$ bit operations”.

(Can be stated rigorously with more care.)

6 Rectangular splitting

Rectangular splitting for polynomials

[Paterson & Stockmeyer 1973]

Goal: evaluate $p(\xi) = a_{d-1} \xi^{d-1} + \dots + a_0$ with “small” a_i at a “large” (p -bit) $\xi \in \mathbb{R}$

$$\begin{aligned}
 p(x) = & \quad (\quad a_0 + \quad a_1 x + \dots + a_{\ell-1} x^{\ell-1}) \quad x^0 \quad (d = m \ell) \\
 & + (\quad a_{\ell} + \quad a_{\ell+1} x + \dots + a_{2\ell-1} x^{\ell-1}) \quad x^{\ell} \\
 & \quad \vdots \\
 & + (a_{(m-1)\ell} + a_{(m-1)\ell+1} x + \dots + a_{m\ell-1} x^{\ell-1}) x^{(m-1)\ell}
 \end{aligned}$$

Algorithm.

1. (Baby steps) Compute $\xi^2, \dots, \xi^{\ell}$ $O(\ell)$ costly ops
2. Evaluate the inner polynomials $O(\ell m)$ cheap ops
3. (Giant steps) Compute $\xi^{2\ell}, \dots, \xi^{(m-1)\ell}$ $O(m)$ costly ops
4. Evaluate the outer polynomial $O(m)$ costly ops

Same idea for evaluating $p \in \mathbb{K}[x]$ on a polynomial / matrix / ...

Rectangular splitting for hypergeometric series

38

$$f(x) = a_0 + a_0 a_1 x + a_0 a_1 a_2 x^2 + \cdots \quad a_n = p(n) / q(n)$$

$$\begin{aligned}
 & + a_0 \cdots a_{\ell-1} \left(a_0 (1 + a_1 (x + a_2 (x^2 + \cdots + a_{\ell-1} x^{\ell-1}))) x^0 \right. \\
 & + a_\ell (1 + a_{\ell+1} (x + a_{\ell+2} (x^2 + \cdots + a_{2\ell-1} x^{\ell-1}))) x^\ell \\
 & + a_\ell \cdots a_{2\ell-1} \left(\begin{array}{c} \vdots \\ \vdots \end{array} \right. \\
 & \left. + a_{(m-1)\ell} \cdots a_{m\ell-1} \left(a_{(m-1)\ell} (1 + \cdots (\cdots (\cdots + a_{m\ell-1} x^{\ell-1}))) x^{(m-1)\ell} \right) \cdots \right)
 \end{aligned}$$