Fast Algorithms for Polynomial Solutions of Linear Differential Equations

Alin Bostan and Bruno Salvy
Projet Algorithmes, Inria Rocquencourt
78153 Le Chesnay (France)
{Alin.Bostan,Bruno.Salvy}@inria.fr

Thomas Cluzeau
LACO, Université de Limoges
123, avenue Albert Thomas, 87060 Limoges (France)
Thomas.Cluzeau@unilim.fr

January 2005

Si l'on se bornait à demander les intégrales entières,
le problème n'offrirait aucune difficulté.[1]
Joseph Liouville, 1833.

Abstract

We investigate polynomial solutions of homogeneous linear differential equations with coefficients that are polynomials with integer coefficients. The problems we consider are the existence of nonzero polynomial solutions, the determination of the dimension of the vector space of polynomial solutions, the computation of a basis of this space. Previous algorithms have a bit complexity that is at least quadratic in an integer N (that can be computed from the equation), even for merely detecting the existence of nonzero polynomial solutions. We give a deterministic algorithm that computes a compact representation of a basis of polynomial solutions in $\mathcal{O}(N \log^3 N)$ bit operations. We also give a probabilistic algorithm that computes the dimension of the space of polynomial solutions in $\mathcal{O}(\sqrt{N} \log^2 N)$ bit operations. In general, the integer N is not bounded polynomially in the bit size of the input differential equation. We isolate a class of equations for which detecting nonzero polynomial solutions can be performed in polynomial complexity. We discuss implementation issues and possible extensions.

Introduction

The computation of polynomial solutions of linear differential equations (LDEs) lies at the heart of several important algorithms. Indeed, known algorithms for finding rational

[1] If we limited ourselves to asking for polynomial solutions, the problem would not offer any difficulty.

solutions of LDEs [25, 28, 2, 1] proceed by computing a common multiple of the polar part of meromorphic solutions and then compute the numerator as a polynomial solution of a derived LDE. Detecting the existence of nonzero rational solutions is thus reduced to detecting the existence of nonzero polynomial solutions. Algorithms for computing Liouvillian solutions [26, 28, 29, 31] or definite integrals [3, 16] construct intermediate LDEs and need to check whether nonzero rational solutions exist. In most cases no rational solution exists and it is therefore important to be able to detect this property efficiently. Another application is the desingularization of linear differential or difference equations that has been shown [17] to boil down to polynomial solutions of an adjoint linear differential equation.

In order to compute polynomial solutions, the algorithms [25, 28, 2, 1, 5] basically compute a basis of power series solutions at a fixed point (0 or infinity) and search for a linear combination of those which is a polynomial. The complexity is related to the largest integer valuation N of Laurent series solutions at infinity, which can be computed from an indicial polynomial (this is developed in Section 1.2). This integer N can be exponentially large compared to the bit size of the input. For instance, the polynomial $(1 + x + x^2)^N$ is a solution of the first order differential equation

$$(1 + x + x^2)y'(x) - (2xN + N)y(x) = 0.$$

The bit-size of the input is of order $\log N$, while the bit-size of the output (in its monomial representation) is of order N. This shows that computing all coefficients of the polynomial solution has at least exponential bit complexity in the size of the input.

Another example is

$$(x - 1)^2(x + 1)y'(x) - ((x - 1)^2N + x + 1)y(x) = 0$$

with solution $(1 + x)^N \exp(x/(1 - x))$. The valuation at infinity is bounded by N, but there are no nonzero polynomial solutions. For this particular example, it is possible to detect the absence of nonzero polynomial solutions by an algorithm whose complexity is only polynomial in the bit-size of the equation [22, Corollary 8.43]. This criterion applies to equations of order 1. We describe in Section 5 a new criterion for some equations of arbitrary order.

In general however, we do not know of similar criteria. Our aim is therefore to decrease the dependency on N of the complexity. By computing all the coefficients of power series solutions with index between 0 and N, previous algorithms have an arithmetic complexity (number of arithmetic operations) which is at least linear in N and a bit complexity which is at least quadratic in N because of the size of these coefficients. Our contribution is to use classical techniques for manipulating recurrences with polynomial coefficients that allow for the computation of the Nth element or a slice of elements starting at index N in bit complexity roughly proportional to N or arithmetic complexity roughly proportional to $\sqrt{N}$. From there, we derive a probabilistic algorithm for the computation of the dimension of polynomial solutions (and in particular the existence of nonzero polynomial solutions) in complexity $\mathcal{O}(\sqrt{N} \log^2 N)$ by performing the computation modulo a suitable prime. We also derive a deterministic algorithm of bit complexity $\mathcal{O}(N \log^3 N)$ to compute this dimension and a representation of polynomial solutions by a recurrence on the coefficients and initial conditions. Since these initial conditions have bit size $\mathcal{O}(N \log N)$, this is close to optimal. This compact representation of polynomials is well suited for applications; for instance, we show that it allows for fast evaluation. If all coefficients of the polynomials are required, then of course we also obtain a quadratic complexity, because of the bit size of the output. However, compared to previous algorithms, our method is faster by a constant factor, since only the coefficients of the polynomials are computed, instead of all those of power series solutions.

This article is structured as follows. In Section 1, we recall the main facts concerning the indicial polynomial and the derived recurrence as well as the basic method for finding polynomial solutions. We also give complexity estimates for all these steps so that comparisons can be performed. Section 2 recalls the binary splitting method for recurrences and applies it to computing the initial conditions of the recurrence giving the coefficients of polynomial solutions. In Section 3, we use the baby-step/giant-step method for the case of modular computations. The algorithms given in Sections 2 and 3 work under the assumption that the origin is an ordinary point of the differential equation. In Section 4, this hypothesis is dropped and we treat the general case. In Section 5, we recall known criteria that make possible an early detection of nonzero polynomial solutions. We also give a new criterion, for a restricted class of equations. It works in polynomial time in the bit size of the equation. Extensive experiments together with implementation issues are described in Section 6. We conclude with comments on possible extensions to nonhomogeneous or parameterized differential equations, to recurrence equations and applications.

Some points of history

The *naive algorithm* for computing polynomial solutions of LDEs is the indeterminate coefficients method: knowing a bound N on the degree of polynomial solutions, take an ansatz $y = \sum_{i=0}^{N} y_i x^i$ where the y_i are unknowns and replacing y in the equation, get a homogeneous linear system for the y_i. Since the coefficients y_i satisfy a linear recurrence $\mathcal{R}$ with fixed order o, the matrix of this linear system has a particular banded form of bandwidth $o+1$. If $o \ll N$, then this system can be solved by Gauss's method in $\mathcal{O}(N)$ arithmetic operations, or $\mathcal{O}(N^2)$ bit operations, see for instance [23, Section 4.3]. This direct method (without complexity considerations) led Liouville [25, page 154] to his comment opening our article.

A variant of this algorithm, called hereafter *the basic algorithm* was suggested (in slightly different variants) in [2, 1, 5]. Roughly speaking, it computes a basis of Laurent series solutions at a fixed point (finite or infinite), then searches for a linear combination P of those series, in which the coefficients of $x^{N+1}, \ldots, x^{N+o}$ are all zero (this suffices to ensure that P is a polynomial solution, since its coefficients satisfy a recurrence of order o).

Regarding only the dependence in N, the complexity of the algorithms in [2, 1, 5] is asymptotically the same as that of the naive algorithm. If a solution of degree N exists, both algorithms are nearly optimal (up to logarithmic factors) for computing the monomial representation of this solution, since the bit size of the latter is of order $\mathcal{O}(N^2)$. In the opposite case, none of them provide a faster way of testing the nonexistence of nontrivial polynomial solutions.

Compact Representation of Polynomial Solutions

Classically, a power-series solution $y(x)$ of a LDE $\mathcal{L}$ with polynomial coefficients is represented by its coefficients y_i in the monomial basis $\{x^i\}$. An alternative data structure for $y(x)$ is the recurrence $\mathfrak{R}$ together with a set of o suitable initial conditions. Polynomial solutions of LDEs naturally inherit this compact encoding, which is well suited for basic operations. For instance, we show in Subsection 2.3 that polynomials represented by recurrences and initial conditions can be evaluated efficiently.

For the moment, we explain the terminology *compact* on an example. Consider the LDE $mxy''(x) - xy'(x) + m^2y(x) = 0$. The recurrence satisfied by the coefficients y_i of a polynomial solution in $\mathbb{Z}[x]$ is $(m^2 - i)y_i + m(i^2 + i)y_{i+1} = 0$, with the initial conditions $y_0 = 0$ and $y_1 = (-1)^{m-1}(m^2)!\, m^{m^2-1}$. From that, it follows easily that the equation admits a one-dimensional $\mathbb{Q}$-vector space of polynomial solutions of degree $N = m^2$; this space is generated by the polynomial

$$y_m(x) = \sum_{i=1}^{N} (-1)^{i+m}\,(N-1)!\binom{N}{i} m^{N-i} x^i.$$

Because of the factorials, to write down all the coefficients of $y_m(x)$, we need a number of bits that is linear is $N^2 \log(N)$. In contrast, the total bit-size of the compact encoding by recurrence and initial conditions is linear in $N \log(N)$.

Complexity Measures and Notations

For our complexity analyzes, the measure we use is the bit (or boolean) complexity. For this purpose, our complexity model is the multi-tape Turing machine, see for instance [27]. We speak of *bit operations* to estimate time complexities in this model. We use the notation $\mathsf{I} : \mathbb{N} \to \mathbb{N}$ to denote the bit complexity of integer multiplication, *i.e.* such that the product of two integers of bit-size d can be computed within $\mathsf{I}(d)$ bit-operations. Alternately, for any $N \in \mathbb{N}$, multiplying two integers bounded by N takes $\mathsf{I}(\log N)$ bit operations. For any prime number p, the bit complexities of the operations $(+, -, \times, \div)$ in the finite field $\mathbb{F}_p := \mathbb{Z}/p\mathbb{Z}$ are in $\mathcal{O}(\mathsf{I}(\log p))$. We denote by $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ a function that represents the arithmetic complexity of polynomial multiplication, *i.e.* such that over any ring R, the product of two polynomials of degree at most d can be computed within $\mathsf{M}(d)$ base ring operations (each ring operation is counted at unit cost). For computations in $\mathbb{F}_p[x]$, the bit complexity is bounded by multiplying the arithmetic cost estimates by the bit complexity of the basic operations in $\mathbb{F}_p$.

We suppose that the multiplication time functions M and I are super-additive, *i.e.*, $\mathsf{M}(d_1) + \mathsf{M}(d_2) \leq \mathsf{M}(d_1 + d_2)$ and $\mathsf{I}(d_1) + \mathsf{I}(d_2) \leq \mathsf{I}(d_1 + d_2)$ for all positive integers d_1 and d_2; in particular, the inequalities $2\,\mathsf{M}(d) \leq \mathsf{M}(2d)$ and $2\,\mathsf{I}(d) \leq \mathsf{I}(2d)$ hold for all $d \geq 1$. We also assume that $\mathsf{M}(cd)$ is in $\mathcal{O}(\mathsf{M}(d))$ and that $\mathsf{I}(cd)$ is in $\mathcal{O}(\mathsf{I}(d))$, for all $c > 0$. The basic examples we have in mind are *classical* multiplication, for which $\mathsf{M}(n), \mathsf{I}(n) \in O(n^2)$, Karatsuba's multiplication with $\mathsf{M}(n), \mathsf{I}(n) \in O(n^{1.59})$ and the FFT-based multiplication which have $\mathsf{M}(n), \mathsf{I}(n) \in O(n \log(n) \log(\log(n)))$. Our references for matters related to polynomial and integer arithmetic are the books [21, 27].

The constant ω denotes the matrix multiplication exponent as in [21, Ch. 12], so that two $n \times n$ matrices can be multiplied within $\mathcal{O}(n^\omega)$ arithmetic operations $(2 \leq \omega \leq 3)$.

In what follows, $\log x$ will always denote the logarithm of x in base 2; $\lfloor x \rfloor$ and $\lceil x \rceil$ respectively denote the largest integer less than or equal to x, and the smallest integer larger than or equal to x. We will use the term *bit-size* (or simply *size*) of an integer $a \neq 0$ for $\lambda(a) := \lfloor \log(|a|) \rfloor + 1$. By convention, we assume that $\lambda(0) = 1$. If $p(x)$ is a polynomial with integer coefficients, we let $\lambda(p)$ denote the maximum bit-size of its coefficients. Similarly, if M is an integer matrix, we denote by $\lambda(M)$ the maximum bit-size of its entries.

We denote by $x^{\underline{i}}$ the falling factorial polynomial $x^{\underline{i}} = x(x-1)(x-2)\cdots(x-i+1)$. For a polynomial $f(x)$, the integers $\operatorname{val}(f)$ and $\deg(f)$ stand respectively for the usual x-adic valuation and for the degree in x of f.

All along this text, $\mathcal{L} := \sum_{i=0}^{n} a_i(x)\,\partial^i$ will denote a linear differential operator of order n, where $a_i(x)$ are polynomials in $\mathbb{Z}[x]$, whose gcd is 1 and $a_n \neq 0$. The integers d and ℓ will denote a bound on the degrees of the $a_i(x)$, respectively on the bit-size of their coefficients. In order to give complexity estimates, we restrict ourselves to the case where the a_i have integer coefficients, but the mathematical results presented in Section 1 (the basic algorithm) are valid over any domain.

1 Differential Equations and Recurrence Relations

We now recall well-known facts concerning the calculation of the polynomial solutions of $\mathcal{L}$ (*i.e.*, of a basis of the $\mathbb{Q}$-vector space of $y \in \mathbb{Q}[x]$ satisfying $\mathcal{L}(y) = 0$). We add a (possibly new) complexity analysis. In §1.1 we introduce recurrence and indicial polynomials, in §1.2 we show how to bound the degree of polynomial solutions in terms of the order of $\mathcal{L}$ and of the degree and bit-size of its coefficients. In §1.3, we describe the basic algorithm for polynomial solutions and we estimate its bit-complexity. In §1.4, we reformulate the basic algorithm, so as to pave the way for the algorithmic improvements given in Sections 2 and 3.

1.1 Recurrence

To $\mathcal{L}$ we attach the following nonnegative integers:

$$\alpha := -\min_i(\operatorname{val}(a_i) - i) \quad \text{and} \quad \beta := \max_i(\deg(a_i) - i). \tag{1}$$

The $a_i(x)$ can thus be written $a_i(x) = \sum_{j=-\alpha}^{\beta} a_{i,j}\, x^{i+j}$ for some $a_{i,j} \in \mathbb{Z}$. For $j \in \{-\alpha, \dots, \beta\}$, we introduce the polynomials $u_j(x) := \sum_{i=0}^{n} a_{i,j}\, x^{\underline{i}}$.

Remark that, by definition, $\alpha \leq n$, with equality if and only if $x = 0$ is an ordinary point. Moreover, if $x = 0$ is ordinary, then $u_{-\alpha}(x) = a_{n,-\alpha}\, x^{\underline{n}}$ has roots $0, 1, \dots, n-1$.

If $y = \sum_{i=v}^{N} y_i\, x^i$ is a polynomial solution of $\mathcal{L}$ satisfying $y_v \neq 0$ and $y_N \neq 0$, then a direct calculation shows that

$$\mathcal{L}(y) = \sum_{i=v-\alpha}^{N+\beta} \Big(u_{-\alpha}(i+\alpha)\, y_{i+\alpha} + \cdots + u_{\beta}(i-\beta)\, y_{i-\beta} \Big)\, x^i,$$

with $y_i = 0$ as soon as $i < v$ or $i > N$. Thus, for all $i \in \mathbb{Z}$, the coefficients y_i satisfy the linear recurrence

$$u_{-\alpha}(i+\alpha)\, y_{i+\alpha} + \cdots + u_{\beta}(i-\beta)\, y_{i-\beta} = 0. \tag{2}$$

Definition 1. *The polynomials u_β (resp. $u_{-\alpha}$) are called the indicial polynomial of $\mathcal{L}$ at infinity (resp. at the origin).*

In what follows, we use the notation $o = \alpha + \beta$ for the order of the recurrence (2).

Plugging $i = N + \beta$ (resp. $i = N - \alpha$) in recurrence (2), we see that the possible degrees (resp.valuations) of the polynomial solutions of $\mathcal{L}$ lie among the positive integer roots of the indicial polynomial of $\mathcal{L}$ at infinity (resp. at the origin). More generally, these roots give all the possible valuations at infinity (resp. 0) of power series solutions. This is why a root of the indicial equation at infinity may not correspond to a polynomial solution.

The integer $N = N_\mathcal{L}$ will denote the largest positive integer root of the indicial polynomial u_β; it is a bound on the possible degrees of polynomial solutions of $\mathcal{L}$.

The *soft Big-Oh* notation $\mathcal{O}_{\log}(\ )$ indicates the presence of terms depending polynomially in $\log(\ell)$, $\log(n)$, $\log(d)$ and $\log(\log(N))$.

1.2 Bound on Degree

The following result gathers some useful bounds. We give here exact bounds rather than $O()$ estimates, they will be necessary when we give precise bounds on probabilities in Section 3.2.

Lemma 1. *Let n, m, ℓ and N be positive integers. Then:*

(i) *All the coefficients of the falling factorial polynomial $x^{\underline{n}}$ have sizes bounded by $n\log(n)$.*

(ii) *If $p(x) \in \mathbb{Z}[x]$ has degree n and if its coefficients in the falling factorial basis $x^{\underline{i}}$ have bit size at most ℓ, then the coefficients of $p(x)$ in the monomial basis have sizes bounded by $\ell + 2n\log(n)$.*

(iii) *If $p(x) \in \mathbb{Z}[x]$ has degree n and if $a \in \mathbb{Z}$, the coefficients of the polynomial $p(x + a)$, in particular $p(a)$, have bit-size bounded by $\lambda(p) + n + n\lambda(a)$.*

(iv) *Let M be a $m \times m$ matrix with entries in $\mathbb{Z}[x]$ of degree at most n and with coefficients of size at most ℓ. Then, the entries of the matrix $M(N) \cdots M(1)$ have bit-sizes bounded by $N(\ell + n + \log(m) + n\log(N))$.*

Lemma 2. *Let $\mathcal{L}$ be a LDE of order n. We can compute a bound N on the degree of polynomial solutions of $\mathcal{L}$ in $\mathcal{O}_{\log}(n^2\,(\ell + n))$ bit operations. Moreover the bit-size of N is bounded by $\ell + 2\,n\log(n)$.*

Proof. By Lemma 1, the indicial polynomial u_β of $\mathcal{L}$ at infinity is a polynomial of degree bounded by n whose coefficients have size bounded by $h = \ell + 2\,n\,\log(n)$. From [21, Theorem 15.21], we can compute all the rational roots of u_β in $\mathcal{O}_{\log}(n^2\,h)$ bit operations, so that the first assertion is clear. The second one follows from the fact that the integer roots of $u_\beta(x)$ necessarily divide its trailing coefficient. □

Note that the complexity estimate given in the previous lemma yields more than the degree bound N: it follows from [21, Theorem 15.21], that all the roots of the indicial polynomial of $\mathcal{L}$ at infinity can be computed for the same price.

Lemma 3. *Let $\mathcal{L}$ be a LDE of order n. The polynomials $u_i(x)$ defining recurrence (2) can all be computed using $\mathcal{O}(o\,\mathsf{M}(n)\,\log(n)\,\mathsf{I}(\ell + n\,\log(n)))$ bit operations.*

Proof. By Lemma 1, the bit-size of the coefficients of each $u_i(x)$ is bounded by $B = \lceil \ell + 2n\log(n) \rceil$. Thus, to compute all the polynomials $u_i(x)$, it is sufficient to perform o conversions between the falling factorial and the monomial basis, in degree n, over the finite field $\mathbb{F}_p$, where p is a prime number chosen in the interval $\{2^B, \dots, 2^{B+1}\}$. Each such conversion can be done using $\mathcal{O}(\mathsf{M}(n)\log(n))$ operations in $\mathbb{F}_p$, see [11, Theorem 1] and since one operation in $\mathbb{F}_p$ costs $\mathcal{O}(\mathsf{I}(\log(p)))$ bit operations, the result follows. □

In conclusion, the recurrence (2) and the bound N can be computed within $\mathcal{O}_{\log}(n(n+o)(n+\ell))$ bit operations, provided FFT is used. When $n + o \in \mathcal{O}(o)$ (*e.g.*, if $x = 0$ is an ordinary point, so that $o \geq n$), this is optimal with respect to the bit-size of the recurrence, which is in $\mathcal{O}_{\log}(on(n+\ell))$.

1.3 Basic Algorithm

The *basic algorithm* described in [2, 1, 5] uses the linear recurrence (2) to compute a basis of power series solutions of $\mathcal{L}$ at a point (say at $x = 0$), then search for linear combinations of these power series which yield polynomial solutions.

To simplify the presentation, we suppose in this section that $x = 0$ is an ordinary point for $\mathcal{L}$ (we refer to Section 4 for a discussion on the singular case). Since the number of singularities of $\mathcal{L}$ is at most d, this hypothesis is not restrictive: it comes down to finding an integer z that lies outside the roots of $a_n(x)$, then to performing a shift by z of all the coefficients a_i. Such a point z can be quickly found (*e.g.*, among the integers $0, 1, \dots, d$), since all the integer roots of a_n can be determined in $\mathcal{O}_{\log}(d^2\ell)$ bit operations. Then, the required shifts of the $a_i(x)$ can be performed in $\mathcal{O}_{\log}(n\mathsf{M}(d^2 + d\ell))$ bit operations [20, Theorem 2.4].

Since $x = 0$ is ordinary, Cauchy's theorem guarantees the existence of a basis of power series solutions $y_i(x)$ of $\mathcal{L}$, of valuations $0, 1, \dots, n-1$.

Taking as input the coefficients of $\mathcal{L}$ and an integer N, the *basic algorithm* outputs a basis of the polynomial solutions of $\mathcal{L}$ of degree at most N. The version of this algorithm which is presented in [1] proceeds as follows:

(1) For $0 \leq i \leq n-1$, use the recurrence (2) to compute one by one the coefficients of a power series solution $y_i = x^i + \sum_{j=i+1}^{N+o} y_{i,j}\, x^j + \mathcal{O}(x^{N+o+1})$;

(2) Form the $n \times o$ matrix $M = \Big[y_{ij}\Big]_{0 \leq i \leq n-1}^{N+1 \leq j \leq N+o}$ and compute a basis $\mathfrak{B} \subset \mathbb{Q}^n$ of solutions of $M^t \cdot c = 0$;

(3) For every $c = (c_1, \dots, c_n) \in \mathfrak{B}$, output $P_c = \sum_{i=0}^{n-1} c_i y_i$.

In the variant [2, 5], the recurrence is used backwards starting from indices $N+1, \dots, N+o$ and leading to a linear system on the y_{ij} for small j. We only analyse precisely the algorithm presented above, but its main features are shared by this other variant: number of arithmetic operations linear in N, number of bit operations quadratic in N. Our improvements described in the next two sections (binary splitting, matrix factorials) apply equally well to this variant.

For further reference, let us briefly estimate the bit complexity of the basic algorithm.

A direct analysis shows that each entry of M has bit size of order $\mathcal{O}_{\log}(\ell N + nN\log(N))$, that the entries of c have bit size in $\mathcal{O}_{\log}(o\,\ell N + o\,nN\log(N))$ and that the total bit size of each output polynomial P_c is in $\mathcal{O}_{\log}(o\,\ell N^2 + o\,nN^2\log(N))$.

In step (1), to determine each y_i, one has to multiply integers of bit-size $\mathcal{O}_{\log}(j\ell + jn\log(j))$ and $\mathcal{O}_{\log}(\ell + n\log(j))$, for j from 1 to $N+o$. Thus, the total complexity of this step is

$$n\sum_{j=1}^{N+o} j\,\mathsf{I}\big(\ell + n\log(j)\big) = \mathcal{O}_{\log}\Big(n(N+o)^2\,\mathsf{I}\big(\ell + n\log(N)\big)\Big).$$

In step (2), one has to solve an $o\times n$ linear system with integer coefficients of size $\Gamma = \mathcal{O}_{\log}(\ell N + nN\log(N))$. Doing linear algebra modulo a prime of size $o\Gamma$, this can be performed in $\mathcal{O}_{\log}(o^{\omega}\,\mathsf{I}(o\ell N + onN\log(N)))$ bit operations.

Step (3) consists in computing linear combinations of rational numbers of size $\mathcal{O}_{\log}(o\,\ell N + o\,nN\log(N))$ and polynomials of degree $N+o$ whose coefficients have size $\mathcal{O}_{\log}(\ell N + nN\log(N))$. For each P_c, this can be done in

$$\mathcal{O}_{\log}(o\,n\,(N+o)\,\mathsf{I}(\ell N + nN\log(N))).$$

Supposing that FFT is used and taking into account only the dependence in N, the complexities of steps $(1),(2),(3)$ are $\mathcal{O}\big(N^2\log(N)\big)$, $\mathcal{O}\big(N\log^2(N)\big)$ and $\mathcal{O}\big(N^2\log^2(N)\big)$, respectively, so that the total complexity reads $\mathcal{O}\big(N^2\log^2(N)\big)$. Note that, if a nonzero polynomial solution of degree $\mathcal{O}(N)$ exists, then this complexity is almost optimal with respect to the size of the output, in its monomial representation. However, if nonzero polynomial solutions exist and if they have degrees bounded by a certain $d \ll N$, then the previous algorithm looses its optimal character; in other words, it is not output-sensitive.

Moreover, if only a partial information (such as the existence of nonzero polynomial solutions or their dimension) is required, the previous algorithm cannot provide it with computational effort less than $\mathcal{O}\big(N^2\log(N)\big)$.

1.4 Towards Improving the Basic Algorithm

It is classical that linear recurrences of arbitrary order rewrite as first order matrix recurrences. Indeed, if we let $Y_i := {}^t(y_i, y_{i+1}, \dots, y_{i+o-1})$, the linear recurrence (2) reads

$$Y_{i+1} = \mathcal{R}(i)\,Y_i, \quad \text{for all} \quad i \geq 0, \tag{3}$$

where $\mathcal{R}(x) = \frac{1}{u_{-\alpha}(x+o)} \times \mathcal{C}(x)$ and where $\mathcal{C}(x)$ is the following $o\times o$ companion-type matrix with entries in $\mathbb{Z}[x]$:

$$\mathcal{C}(x) = \begin{bmatrix} 0 & u_{-\alpha}(x+o) & 0 & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & u_{-\alpha}(x+o) \\ -u_{\beta}(x) & -u_{\beta-1}(x+1) & \cdots & u_{-\alpha+1}(x+o-1) \end{bmatrix}.$$

Lemma 4. *The coefficients of the entries of $\mathcal{C}(x)$ have bit sizes bounded by $\ell + n + 3n\log(o) = \mathcal{O}_{\log}(\ell + n)$.*

Proof. Straightforward from Lemma 1. □

Remark that since $x = 0$ is supposed to be an ordinary point, the indicial polynomial $u_{-\alpha}(x)$ at the origin has roots $0, 1, \ldots, n-1$, so that $u_{-\alpha}(x+o) \neq 0$ for all $x \geq -\beta$. This ensures that recurrence (3) can be used for $i \geq -\beta$.

The key point in what follows is that the only coefficients of a power series solution $y(x) = \sum y_i x^i$ which are needed in Step (1) of the basic algorithm can be computed by the equation $Y_{N+1} = \mathcal{R}(N) \cdots \mathcal{R}(k)\, Y_k$, for all $k \geq -\beta$. This formula will be exploited so as to avoid the computation of all the intermediate coefficients $y_0, \ldots, y_{N-1}$.

The following result is a simple consequence of these considerations; it contains the theoretical basis needed in the next sections to improve the complexity of the basic algorithm. For $m, n \geq 1$, we denote by $\mathsf{O}_{m,n}$ the zero matrix of sizes $m \times n$ and by Id_n the $n \times n$ identity matrix.

Proposition 1. *Let $\mathcal{I}$ be the transpose of the $n \times o$ matrix $\left[\mathsf{O}_{n,o-n} \mid \mathsf{Id}_n\right]$. Then, the matrix M^t defined in Step (2) of the basic algorithm equals $\mathcal{R}(N) \cdots \mathcal{R}(-\beta)\mathcal{I}$. Moreover, a vector $c \in \mathbb{Q}^n$ contains the coefficients of $1, x, \ldots, x^{n-1}$ of a polynomial solution of $\mathcal{L}$ if and only if $\mathcal{C}(N) \cdots \mathcal{C}(-\beta)\mathcal{I}c = 0$.*

Proof. We have just seen that $c = \sum_{i \geq 0} c_i x^i$ is a polynomial solution of $\mathcal{L}$ if and only if $\mathcal{R}(N) \cdots \mathcal{R}(-\beta)\mathcal{I}c = 0$. Since the denominator of $\mathcal{R}$ does not cancel at any x in $\{-\beta, \ldots, N\}$ this concludes the proof. □

To put this theoretical result into practice, the basic computational task will be the following

Matrix factorials: given a ring R, an integer $N \geq 0$ and a square matrix $\mathcal{C}$ with coefficients in $R[x]$, compute the product $\mathcal{C}(N) \cdots \mathcal{C}(1)$.

In the next sections, we will give bit complexity estimates for this operation in the case when $R = \mathbb{Z}$ and respectively $R = \mathbb{F}_p$, for a prime p. We will then apply these results to obtain improved versions of the basic algorithm.

Note that in our applications, we consider slightly more general products $\mathcal{C}(N) \cdots \mathcal{C}(U)$ with U possibly $\neq 1$, but there is no loss in treating only the case $U = 1$.

2 The Binary-Splitting Method

2.1 Matrix factorials over $\mathbb{Z}$

Let $N \geq 0$ and let $\mathcal{C}(x)$ be an $m \times m$ matrix with entries in $\mathbb{Z}[x]$ of degree at most n and whose coefficients have bit-size bounded by ℓ. We wish to compute the matrix with integer entries $\mathcal{A} = \mathcal{C}(N) \cdots \mathcal{C}(1)$.

By Lemma 1, the bit size $\lambda(\mathcal{A})$ is bounded by $N(\ell + n + \log(m) + n\log(N)) = \mathcal{O}\big(N\big(\ell + \log(m) + n\log(N)\big)\big)$.

The simplest particular case $\mathcal{C}(x) = x$ corresponds to the computation of factorials. The naive method (multiply $i+1$ by $i!$ for increasing i) has complexity $\mathcal{O}(N^2\log(N))$, that is, quadratic in the bit size of $N!$. A well-known improvement mentioned in [8, Proposition 1] but possibly known before, is to use recursively the formula $i(i+1)\cdots(i+j) = \big(i(i+1)\cdots\lceil\frac{i+j}{2}\rceil\big)\cdot\big(\lceil\frac{i+j}{2}+1\rceil\cdots(i+j)\big)$.

This technique, commonly called *binary-splitting* (or *divide-and-conquer*) becomes useful in conjunction with fast multiplication. For $N!$ it yields an algorithm of complexity $\mathcal{O}(\mathsf{I}(N\log(N))\log(N))$ bit operations, which is almost linear in the bit-size $N\log(N)$ of the output if FFT is used.

These facts are classical, they were rediscovered many times in various particular cases: base conversion, fast computation of places of π and e [14, 13], computation with linear recurrences [24],[15, Section 6]. The earlier historic trace we could find is [6, Item 178]), which suggests (without proof) that fast evaluation of solutions of LDEs can be done by binary splitting. We refer to [7, Section 12] for a good survey on binary splitting-based algorithms.

We summarize the complexity analysis for integer matrix factorials in the result below (compare [15, Theorem 6.1]).

Lemma 5. *With the notations above, we can compute the matrix factorial $\mathcal{A}$ in $\mathcal{O}\big(m^\omega\,\mathsf{I}(\lambda(\mathcal{A}))\log(N)\big)$ bit operations.*

Remark 1. In [8, 27], even faster methods (by a logarithmic factor) are described for computing factorials. They exploit the decomposition into primes of $N!$. In [15, page 459], it is suggested without proof that this improvement carries also for computing $c(N)\cdots c(1)$, where $c(x)\in\mathbb{Z}[x]$ "using the distribution of prime ideals in the Galois group of $c(x)$". However, these methods do not directly adapt to the matrix case; we leave as an open problem the question whether such an improvement is possible or not for matrix factorials.

2.2 Application to Polynomial Solutions

We now use binary-splitting to give an improved version of the basic algorithm. It computes the polynomial solutions of $\mathcal{L}$ given by their compact representation by a recurrence and initial conditions.

Theorem 1. *Let $\mathcal{L}=\sum_{i=0}^n a_i\,\partial^i$ be a LDE of order n. Let d (resp. ℓ) denote a bound on the degrees of the a_i (resp. on the bit size of their coefficients). Let $N\ge d$. The algorithm* **BinSplitPolySols** *below computes the polynomial solutions of $\mathcal{L}$ of degree at most N in*

$$\mathcal{O}_{\log}(o^\omega\,\mathsf{I}(o\ell N + on\,N\log(N))\,\log(N))$$

bit operations.

Lemma 6. *The entries of $\mathcal{A}_{init}$ have bit sizes bounded by $d(7n\log(o)+n\log(d)+\ell)=\mathcal{O}_{\log}(\ell d+nd)$ and the entries of $\mathcal{A}$ have bit sizes bounded by $N(6n\log(o)+n\log(N)+\ell)+2d(7n\log(o)+n\log(d)+\ell)=\mathcal{O}_{\log}(\ell N+nN\log(N)+dn+d\ell)$.*

Proof. By Lemma 3, the recurrence (2) and the entries of $\mathcal{C}$ can be computed in $\mathcal{O}(o\,\mathsf{M}(n)\,\log(n)\,\mathsf{I}(\ell+n\,\log(n)))=\mathcal{O}_{\log}(o\,\mathsf{M}(n)\,\mathsf{I}(\ell+n))$ bit operations. By Lemma 6 and since $N\ge d$, the coefficients of the entries of $\mathcal{A}$ and $\mathcal{A}_{\text{init}}$ have bit sizes bounded by a certain $\Gamma=\mathcal{O}_{\log}(\ell N+nN\log(N))$. The matrices $\mathcal{A}_{\text{init}}$ and $\mathcal{A}$ are then computed by

```
BinSplitPolySols

Input: $\mathcal{L} \in \mathbb{Z}[x][\partial]$ with $x = 0$ ordinary, $N \geq 1$.
Output: the compact representation of a basis of polynomial solutions of $\mathcal{L}$ of degree $\leq N$.

    1. Compute the recurrence (2) and the matrix $\mathcal{C}$;
    2. Compute the matrix $\mathcal{A}_{\text{init}} = \mathcal{C}(-1) \cdots \mathcal{C}(-\beta)\, \mathcal{I}$;
    3. Compute the matrix $\mathcal{A} = \mathcal{C}(N) \cdots \mathcal{C}(0)\, \mathcal{A}_{\text{init}}$;
    4. Compute a basis $\mathfrak{B} \subset \mathbb{Q}^n$ of solutions of $\mathcal{A}c = 0$;
    5. For every $c = (c_1, \ldots, c_n) \in \mathfrak{B}$, output $\mathcal{A}_{\text{init}} \cdot c$.
```

binary splitting. Using Lemma 5, the complexity of Steps 2 and 3 is in $\mathcal{O}\big(o^\omega\, \mathsf{I}(\Gamma) \log(N)\big)$ bit operations.

In step 4, one has to solve a $o \times n$ linear system with integer coefficients of size Γ. Doing linear algebra modulo a prime of size $o\Gamma$, this can be performed in $\mathcal{O}_{\log}\big(o^\omega\, \mathsf{I}(o\Gamma)\big)$ bit operations. Now, the entries of c have bit size in $\mathcal{O}_{\log}(o\,\Gamma)$ and the entries of $\mathcal{A}_{\text{init}}$ have bit size in $\mathcal{O}_{\log}(\Gamma)$. Thus, in the final step, each product $\mathcal{A}_{\text{init}} \cdot c$ can be done in $\mathcal{O}_{\log}\big(n\, \mathsf{I}(o\,\Gamma)\big)$ bit operations and there are at most $n \leq o$ such products to perform. Putting these estimates together concludes the proof. $\square$

The complexity of our **BinSplitPolySols** algorithm is thus better than that of the basic algorithm as soon as the available integer product is better than quadratic. For instance, using FFT, from Theorem 1, if we neglect the logarithms, we obtain a linear dependence in N instead of the quadratic one.

2.3 Evaluation in the Compact Representation

When nonzero polynomial solutions exist, the output of our algorithm **BinSplitPolySols** from Subsection 2.2 is distinct from that of the basic algorithm (described in Subsection 1.3): instead of the (dense) monomial representation, it returns an encoding of the polynomial solutions by a recurrence and by initial conditions for their coefficients.

We show that this data structure is well suited for fast evaluation. Consider a polynomial P of degree N with integer coefficients of bit size h. Let $a \in \mathbb{Z}$ be an integer of bit size ℓ and consider the problem of evaluating P at a; note that the bit size of the output $P(a)$ is bounded by $\lambda = \mathcal{O}(h + \ell\, N)$. To simplify the exposition, we assume that $h, \ell \ll N$ and that FFT is used for integer multiplication. Horner's algorithm allows to perform the evaluation in $\mathcal{O}(\lambda\, N) = \mathcal{O}(N^2)$ bit operations. A better algorithm, of divide-and-conquer type, was suggested in [19]. It consists in splitting $P(x) = P_1(x) + x^{N/2}\, P_2(x)$, and evaluating recursively the polynomials P_1 and P_2, of half degree. Using FFT, the bit complexity of this algorithm is $\mathcal{O}\left(\lambda \log^2(N)\right) = \mathcal{O}\left(N \log^2(N)\right)$. Neglecting log factors, this complexity is nearly optimal with respect to the bit size of the output. Now suppose that $P = \sum_{i=0}^{N} p_i\, x^i$ is given by a recurrence of order o (with polynomial coefficients) satisfied by the coefficients p_i, together with the first o values $p_0, \dots, p_{o-1}$. Let us assume that the total bit size of the recurrence is negligible compared to N. Now, $P(a)$ equals the Nth term of the sequence $v_k := \sum_{i=0}^{k} p_i \, a^i$, which satisfies a recurrence of order $o+1$ with polynomial coefficients. Therefore, by Lemma 5, $P(a)$ can be computed by binary splitting in $\mathcal{O}(o^\omega \, \mathsf{I}(\lambda) \, \log(N)) = \mathcal{O}(N \log^2(N))$ bit operations, which is again almost linear in the bit size of the output. Similar considerations can be made for evaluation at rationals or algebraic numbers (see also [15, Theorem 5.2]).

3 Baby Steps / Giant Steps

In this section, we devise a probabilistic algorithm which outputs the dimension of the $\mathbb{Q}$-vector space of polynomial solutions of a linear differential operator $\mathcal{L}$ in $\mathbb{Z}[x][\partial]$. Our contribution is to use a classical technique for manipulating linear recurrences with polynomial coefficients that allow for the computation of the Nth element or a slice of elements starting at index N in arithmetic complexity roughly proportional to $\mathsf{M}(\sqrt{N})$. From there, we derive a probabilistic algorithm for the computation of the dimension of polynomial solutions (and in particular the existence of nonzero polynomial solutions) in complexity $\mathcal{O}(\mathsf{M}(\sqrt{N}) \, \mathsf{I}(\log N))$ by performing the computation modulo a suitable prime.

3.1 Matrix factorials in positive characteristic

Chudnovsky and Chudnovsky proposed in [15] an algorithm that determines one term of a linear sequence with polynomial coefficients without computing all intermediate terms. Using the so called *baby steps / giant steps technique*, it requires a number of operations which is roughly linear in $\sqrt{N}$ to compute the Nth term in the sequence. The Chudnovskys rediscovered and generalized Strassen's algorithm [30, Abschnitt 6] (used for deterministic integer factorization) which computes $N! \bmod p$ for a prime $p > N$. This is unsurprising, since the sequence $u_N = N!$ is the prototype for linear sequences with nonconstant coefficients. Strassen's idea is to build the polynomial $M(x) = (x + 1) \cdots (x + \sqrt{N})$, then to evaluate it at the arithmetic progression $0, \sqrt{N}, \dots, N - \sqrt{N}$ using fast multipoint evaluation techniques and to output $N! = M(N - \sqrt{N}) \cdots M(\sqrt{N}) \, M(0)$ (these computations are all done over the field $\mathbb{F}_p$). The Chudnovskys' result proceeds in a similar way for any recurrence, by rewriting it as a first order matrix recurrence $u_{N+1} = \mathcal{C}(N+1) u_N$, where $\mathcal{C}$ is a matrix with rational fraction entries. We recall below the corresponding complexity result (compare [15, Theorem 6.2]).

Lemma 7. *Let F be a finite field of characteristic p and let $\mathcal{C}$ be a $m \times m$ polynomial matrix over F with entries of degree at most d. Let $N > d$. Then, the matrix factorial $\mathcal{A} = \mathcal{C}(N) \cdots \mathcal{C}(1)$ can be computed using*

$$\mathcal{O}\left(m^2 \, \mathsf{M}(\sqrt{dN}) \log(dN) + m^\omega \mathsf{M}(\sqrt{dN})\right)$$

operations in F.

Improvement of the Chudnovskys' Algorithm

The algorithm of Lemma 7 spends an important amount of computation in evaluating a polynomial matrix at points that form an arithmetic progression. Now, this matrix has a very particular form, $M(x) = \mathcal{C}(x+\sqrt{N})\cdots\mathcal{C}(x+1)$, where $\mathcal{C}$ is a given polynomial matrix. This allows us to improve the complexity result in Lemma 7 by a factor of $\log(N)$. Roughly, this speed-up is due to the use of the operation of *extrapolation* recalled below, which can be performed faster than multipoint evaluation.

Lemma 8. [10, Prop. 1] *Let F be a field, let $d \geq 0$ and let P be in $F[x]$ of degree at most d, such that the sequence $P(0), P(1), \dots, P(d)$ is known. Then:*

simple extrapolation: *the values $P(d+1), \dots, P(2d+1)$ can be computed using $2\,\mathsf{M}(d) + \mathcal{O}(d)$ operations in F;*

extrapolation of arbitrary difference: *if $r \in F$, so that the elements $r-d, \dots, r+d$ and $1, 2, \dots, d$ are nonzero in F, then the sequence $P(r), P(r+1), \dots, P(r+d)$ can be computed using $\mathsf{M}(d) + \mathcal{O}(d)$ operations in F.*

Such an evaluation algorithm based on recursive extrapolation has been presented in [9] for the case when all the entries are linear forms. We now give a straightforward extension of [9, Theorem 2]; for completeness, we also give a sketch of the proof. Let us briefly explain the basic algorithmic idea on the example of the usual factorial. As in Strassen's algorithm, we need to evaluate the polynomial $M(x) = (x+1)\cdots(x+n)$ at the arithmetic progression $0, n, \dots, n^2-n$, where $n = \sqrt{N}$. Suppose that we have already evaluated $(x+1)\cdots(x+n/2)$ on half of the points. It then suffices to evaluate this polynomial on the other half, and also, to evaluate $(x+n/2+1)\cdots(x+n)$ on all points. Due to the form of the polynomials, both tasks can be done using only extrapolations, and without constructing the polynomials to be evaluated. Similarly, in the matrix case, the advantage of our algorithm is that, contrary to the Chudnovskys' algorithm, it does not require polynomial matrix multiplications, but only scalar matrix multiplications.

Theorem 2. *Let F be a finite field of characteristic p and let $\mathcal{C}$ be a $m \times m$ polynomial matrix over F with entries of degree at most d. Let $N > d$ and suppose that $p > \sqrt{dN}+1$. Then, the matrix factorial $\mathcal{A} = \mathcal{C}(N)\cdots\mathcal{C}(1)$ can be computed using $\mathcal{O}\left(m^2\,\mathsf{M}(\sqrt{dN}) + m^\omega\sqrt{dN}\right)$ operations in F.*

Proof. For the ease of exposition, let us suppose that N and d are perfect powers of 4, such that $k = \sqrt{dN}$ and $n = \sqrt{N/d}$ are powers of 2. The general case can be treated with no extra difficulty (by writing N in base 4). Let $\mathcal{M}(x)$ be the polynomial matrix $\mathcal{C}(nx+n)\cdots\mathcal{C}(nx+1)$. Since the required scalar matrix $\mathcal{A}$ equals $\mathcal{M}(k-1)\cdots\mathcal{M}(0)$, it suffices to evaluate $\mathcal{M}$ on $0, 1, \dots, k$. For this, we use a recursive algorithm. Suppose that the values of the matrix $\mathcal{M}_0(x) = \mathcal{C}(nx+\frac{n}{2})\cdots\mathcal{C}(nx+1)$ at the points $0, 1, \dots, \frac{k}{2}$ are already known. Let $\mathcal{M}_1(x) = \mathcal{M}_0(x+\frac{1}{2})$, such that $\mathcal{M}(x) = \mathcal{M}_1(x)\mathcal{M}_0(x)$. Since the degree of $\mathcal{M}_0$ is at most $k/2$, the values of $\mathcal{M}_0$ at $0, 1, \dots, k+1$ can be deduced using m^2 simple extrapolations, in complexity $\mathcal{O}(m^2\mathsf{M}(k))$. The values of $\mathcal{M}_1$ at $0, 1, \dots, k+1$ can be deduced by two extrapolations (of difference $1/2$, in degree $k/2$) of the values of $\mathcal{M}_0$ at $0, 1, \dots, k+1$. Since $p > k+1$, the elements $1, 2, \dots, k+1$ and $1/2-k/2, \dots, 1/2+k/2$ are nonzero in F, so these final two extrapolations can also be done in $\mathcal{O}(m^2\mathsf{M}(k))$ operations in F. Finally, the values $\mathcal{M}(i)$ are recovered using $\mathcal{M}(i) = \mathcal{M}_1(i)\mathcal{M}_0(i)$, for $0 \leq i \leq k$. The complexity of the algorithm satisfies the equation $\mathsf{C}(k) \leq \mathsf{C}(\frac{k}{2}) + \mathcal{O}(m^2\mathsf{M}(k) + km^\omega)$, whence $\mathsf{C}(k) = \mathcal{O}(m^2\mathsf{M}(k) + m^\omega\,k)$, as required. $\square$

3.2 Application to Polynomial Solutions

Let $\mathcal{L} \in \mathbb{Z}[x][\partial]$ be a linear differential operator and let N be a bound on the degree of polynomial solutions of $\mathcal{L}$. By choosing a prime number p of order $\mathcal{O}(N \log N)$, the method of the previous section allows us to give a probabilistic algorithm to compute the dimension of the space of polynomial solutions of $\mathcal{L}$. It uses $\mathcal{O}(\mathsf{M}(\sqrt{N}))$ operations in $\mathbb{F}_p$, so its bit complexity is of order $\mathcal{O}(\sqrt{N} \log^2(N))$ if FFT is used.

Before stating the main result of this section, let us recall a fact which will be useful in the following.

Lemma 9. *[21, Theorem 18.9] Let A be nonzero integer of bit-size a. Let $B \in \mathbb{N}$. If p is chosen uniformly at random in the set of prime numbers between B and $2B$, then p does not divide A with probability at least $1 - 2a/B$.*

Corollary 1. *Let $\mathcal{A}$ be an $m \times n$ matrix with integer coefficients of size at most Ω bits. For a prime p, denote by $\mathcal{A}[p]$ the matrix over $\mathbb{F}_p$ whose entries are the reductions modulo p of the entries of $\mathcal{A}$. Let $B \in \mathbb{N}$. Then, for a prime p chosen uniformly at random in the set of prime numbers between B and $2\,B$, the ranks of $\mathcal{A}$ and $\mathcal{A}[p]$ are equal with probability at least $1 - (n \log(n) + 2n\Omega)/B$.*

Proof. Let r denote the rank of $\mathcal{A}$ and let $\mathcal{A}'$ be an $r \times r$ nonsingular minor of $\mathcal{A}$. The condition that p does not divide $\det(\mathcal{A}')$ is sufficient to ensure that $\operatorname{rank}(\mathcal{A}) = \operatorname{rank}(\mathcal{A}[p])$. By Hadamard's bound (see [21, Theorem 16.6]), the bit-size of $\det(\mathcal{A}')$ is bounded $\frac{r}{2}\log(r) + r\,\Omega$ and the conclusion follows from Lemma 9. $\square$

Theorem 3. *Let $\mathcal{L} \in \mathbb{Z}[x][\partial]$ be a linear differential operator of order n, with coefficients of degree at most d and bit-size ℓ. Let $c \geq 0$ and $N \geq n + d$. The algorithm* **ModBsGsPolySols** *below computes the dimension of the $\mathbb{Q}$-vector space of polynomial solutions of $\mathcal{L}$ of degree at most N using*

$$\mathcal{O}_{\log}\Big(\big((n+d)^2\,\mathsf{M}(\sqrt{nN}) + (n+d)^{\omega}\sqrt{nN}\big)\,\mathsf{I}\big(\log(N)\big)\Big)$$

bit operations. The $\mathcal{O}_{\log}(\,)$ notation hides terms depending polynomially in $\log(\ell)$, $\log(n)$, $\log(d)$ and $\log(\log(N))$. The algorithm chooses uniformly at random a prime number in $\{B, \dots, 2B\}$, where $B = \mathcal{O}_{\log}(n^2 \ell N \log^{1+c}(N))$ and outputs the correct result with probability at least $1 - \frac{1}{2\log^c(N)}$.

ModBsGsPolySols

Input: $\mathcal{L} \in \mathbb{Z}[x][\partial]$ with $x = 0$ ordinary, $N \geq 1$.
Output: the dimension of the $\mathbb{Q}$-vector space of the polynomial solutions of $\mathcal{L}$ of degree at most N.

```
1. Choose $p$ prime in $\{B, \dots, 2B\}$ ($B$ defd below);
2. Compute the reduction $\mathcal{C}[p]$ of $\mathcal{C}$ modulo $p$;
3. Compute $\mathcal{A}[p] = \mathcal{C}[p](N) \cdots \mathcal{C}[p](-\beta)\,\mathcal{I}[p]$;
4. Return the integer $n - \operatorname{rank}(\mathcal{A}[p])$.
```

Proof. Let us first study the correctness of the algorithm. Let, as before, $o = \alpha + \beta$ denote the order of the recurrence (2). Since $x = 0$ is ordinary, we have $n \le o \le n + d$ and the indicial polynomial $u_{-\alpha}(x)$ has the form $ax(x-1)\cdots(x-n+1)$, where a has size bounded by ℓ. On the other hand, since $p > d + n + N$, none of the elements $-\beta + o - n + 1, \dots, o + N$ is zero in $\mathbb{F}_p$. This ensures that $u_{-\alpha}(i+o)[p]$ is invertible in $\mathbb{F}_p$, for all $-\beta \le i \le N$, provided that p does not divide a. Therefore, if p is not a divisor of a, the recurrence (3) can be used for $-\beta \le i \le N$ and the conclusion of Proposition 1 still holds for $\mathcal{L}[p]$ (over $\mathbb{F}_p$). In other words, the algorithm returns the dimension of the $\mathbb{F}_p$-vector space of the solutions in $\mathbb{F}_p[x]$ of $\mathcal{L}[p]$.

Let us call p a *good prime* if p does not divide a and if, simultaneously, the matrices $\mathcal{A}$ and $\mathcal{A}[p]$ have the same rank. In short, we have just proved that if the algorithm chooses a good prime p, then the dimension of polynomial solutions over $\mathbb{Q}$ and over $\mathbb{F}_p$ coincide, and thus, the algorithm **ModBsGsPolySols** returns the correct output. Let us now estimate the probability of choosing a good prime.

Using Lemma 6, the entries of $\mathcal{A}$ have sizes upper bounded by $\Gamma = N(6n\log(o) + n\log(N) + \ell) + 2d(7n\log(o) + n\log(d) + \ell)$, which is, by the assumption $N \ge d$, in $\mathcal{O}_{\log}(\ell N + nN\log(N))$. Let B be the integer $B = \lceil 2\log^c(N)\,(\ell + n\log(n) + 2n\Gamma)\rceil$, so that $B = \mathcal{O}_{\log}\big(n^2\ell N\log^{1+c}(N)\big)$.

Let us suppose that the prime p is chosen uniformly at random in the set of prime numbers between B and $2B$. Then, using Corollary 1, it is easy to infer that p is a good prime with probability at least $1 - \frac{1}{2\log^c(N)}$.

Let us finally prove the complexity estimate. Using the algorithm from Lemma 3, Step 2 can be done using $\mathcal{O}((d+n)\,\mathsf{M}(n)\log(n))$ operations in $\mathbb{F}_p$. Step 4 can be done using $\mathcal{O}(n^\omega)$ operations in $\mathbb{F}_p$. Since $N \ge d + n$ and $p > N + 1$, Theorem 2 can be used to perform Step 3 and this concludes the complexity analysis, since every operation in $\mathbb{F}_p$ costs $\mathcal{O}(\mathsf{I}(\log(p))) = \mathcal{O}_{\log}\big(\mathsf{I}(\log(N))\big)$ bit operations. □

Remark 2. The algorithm **ModBsGsPolySols** can be easily modified, so as to return also the degrees of all the polynomial solutions, within the same complexity bound $\mathcal{O}_{\log}(\mathsf{M}(\sqrt{N})\mathsf{I}(\log(N)))$ and with the same probability.

Remark 3. Combining our two algorithms leads to an algorithm for computing polynomial solutions which is *output-sensitive*. To see that, suppose that the indicial polynomial of $\mathcal{L}$ has positive integer roots $N_1 < \cdots < N_k = N$ and that the polynomial solutions of $\mathcal{L}$ have degrees $d_1 < \cdots < d_r = d$. Using our **ModBsGsPolySols** algorithm, we compute the degrees d_i in bit complexity roughly linear in $\sqrt{N}$; then, using our algorithm **BinSplitPolySols**, we return a compact representation of solutions in bit complexity roughly linear in d. If $d \ll N$, this strategy has its benefits; for instance, if $d \approx \sqrt{N}$, we compute the solutions in bit complexity roughly linear in $\sqrt{N}$ instead of N^2 by the basic algorithm.

Remark 4. To our knowledge, the only existing modular test for computing the dimension of the space of nonzero polynomial solutions was given in [18]. It consists in choosing a "small prime"p and in returning the dimension of the $\mathbb{F}_p(x^p)$-vector space of solutions in $\mathbb{F}_p[x]$ of $\mathcal{L}[p]$. There are at least two drawbacks of this method. First, it provides only partial information (an upper bound on the dimension) and second, there exist families of examples on which it fails modulo any prime p. In contrast, our algorithm **ModBsGsPolySols** gives the dimension and the exact degrees of the polynomial solutions and it returns a wrong output with (predictable) arbitrarily small probability.

4 General Case

So far, we have concentrated on the case when the origin is an ordinary point of the differential equation. As already mentioned, one can always shift the equation to recover this situation. However, the algorithm we have described at an ordinary point can also be executed at a singular point, with minor changes that we are now going to describe. Moreover, it turns out that it can be beneficial to execute the algorithm or parts of it at a singular point. This is discussed below.

We first recall the properties of regular points that we have used: *(i)* the indicial polynomial at 0 is $ax(x-1)\cdots(x-n+1)$; *(ii)* there are n linearly independent power series solutions; *(iii)* the order o of the recurrence satisfies $o \geq n$. In general, none of these properties holds at a singular point. This has an impact on the basic algorithm as well as its improved versions using binary splitting or the baby step/giant step technique. The computation of a bound on the degree of solutions is unaffected, as well as the computation of the recurrence. The main difference is in the computation of the initial conditions and in the representation of these initial conditions in the output.

4.1 Valuations

The degree of the indicial polynomial $u_{-\alpha}$ behaves like that of u_β and the computations of its integer roots can be performed with the complexity described in Section 1.2. These integer roots contain the possible valuations of power series (and in particular polynomial) solutions.

4.2 Initial Conditions and Representation of Solutions

We now assume that there are integer roots $0 \leq i_1 < \cdots < i_k$ of the indicial polynomial at 0 that are smaller than the upper bound N on the degree of polynomial solutions. The compact representation of solutions is composed of the recurrence (2) and a set of vectors $(a_1, \ldots, a_k)$ such that a basis of polynomial solutions is defined by the corresponding equalities $y_{i_j} = a_j$. In the ordinary case, $k = n$ and $i_j = j - 1$ for $j = 1, \ldots, n$ so that this representation is the same as before.

The number k is an upper bound on the dimension of power series solutions of valuation smaller than N. We now describe the incremental computation of matrices that play the same rôle as matrix $\mathcal{A}_{\text{init}}$ of Algorithm **BinSplitPolySols**. A new phenomenon is that the vanishing of the indicial polynomial $u_{-\alpha}$ at the i_j's in (2) induces linear constraints on the values of $y_{i_1}, \ldots, y_{i_{j-1}}$.

The algorithm starts with a matrix $\mathcal{A}_1$ which is the transpose of the $1 \times o$ matrix $[0_{o-1} 1]$. We denote by $\mathcal{J}$ a Jordan block matrix of size $o \times o$ with 0 on the diagonal. Then for j from 2 to k, the algorithm proceeds as follows

1. Compute $\mathcal{A} := \mathcal{C}(i_j - o - 1) \cdots \mathcal{C}(i_{j-1} - o + 1)\mathcal{A}_{j-1}$;

2. Compute a matrix of maximal rank with linearly independent columns $\mathcal{F}_j$ such that $\mathcal{C}(i_j - o)\mathcal{A}\mathcal{F}_j = 0$;

3. Compute the product $\mathcal{A} := \mathcal{J}\mathcal{A}_{j-1}\mathcal{F}_j$;

4. Extend $\mathcal{A}$ with a column $[0_{o-1}1]$ and assign this to $\mathcal{A}_j$.

At the end of the jth iteration, the columns of the matrix $\mathcal{A}_j$ give the coefficients of indices $i_j - o + 1, \dots, i_j$ of elements of a basis of solutions of the differential equation in the space of polynomials modulo x^{i_j+1}. In Step 1, whenever $i_j = i_{j-1} + 1$, the product of the $\mathcal{C}$'s has to be interpreted as the identity matrix. In this step, the solutions defined by $\mathcal{A}_{j-1}$ are extended up to index $i_j - 1$. In Step 2, the columns of $\mathcal{F}_j$ form a basis of the kernel of $\mathcal{C}(i_j - o)\mathcal{A}_{j-1}$, this is where the linear constraints imposed by the vanishing of $u_{-\alpha}(i_j)$ are taken into account. The next two steps obtain the new basis up to index i_j. Note that at the end of all iterations, the number of columns of $\mathcal{A}_k$ is exactly the dimension $s \le k$ of the space of power series solutions of the equation.

This algorithm also applies in the nonsingular case. Step 1 never changes $\mathcal{A}_j$; in Step 2, $\mathcal{F}_j$ is always the identity matrix and the net result of all these steps is the computation that is performed in Step 2 of **BinSplitPolySols**.

4.3 Final Steps

The next part of the algorithm is similar to **BinSplitPolySols**:

1. Compute the matrix $\mathcal{A} = \mathcal{C}(N) \cdots \mathcal{C}(i_k - o + 1)\mathcal{A}_k$;

2. Compute a matrix of maximal rank with linearly independent columns $\mathcal{B}$ such that $\mathcal{A}\mathcal{B} = 0$.

At this stage, we have the dimension of the space of polynomial solutions as the number of columns of $\mathcal{B}$. The values of the coefficients of indices $i_k - o + 1, \dots, i_k$ of a basis of this space are the entries of $\mathcal{A}_k\mathcal{B}$. The next step is to recover the coefficients of indices $i_1, \dots, i_{k-1}$ of these elements. These are obtained with no matrix inversion by the following incremental process for j decreasing from k to 2:

1. Compute the matrix product $\mathcal{B} := \mathcal{B}\mathcal{F}_j$;

2. Output the last line of $\mathcal{A}_{j-1}\mathcal{B}$.

Again, in the ordinary case, this algorithm also applies, and since the $\mathcal{F}_j$ are all equal to identity, we recover Step 5 of **BinSplitPolySols**.

5 Criteria

An obvious necessary condition for the existence of nonzero polynomial solutions can be derived from the discussion in the previous section: for all singular points, the indicial polynomial must have integer roots that are smaller than the bound on the degrees provided by the indicial polynomial at infinity. This gives a clearly effective criterion [4].

Another criterion can be deduced from the general algorithm above.

Proposition 2. *If the matrix $\mathcal{A}_k$ computed by the algorithm has rank at least o, then the linear differential equation has a nonzero polynomial solution.*

The rank of the matrix $\mathcal{A}_k$ is the dimension of the space of power series solutions of the differential equation. In the special case when the point is ordinary, this rank is n. The difference $o - n$ is exactly β in this case and the hypothesis of Proposition 2 becomes $\beta = 0$. In this case, we obtain the following corollary that gives a polynomial time criterion.

Corollary 2. *Let $\mathcal{L} = \sum_{i=0}^{n} a_i\, \partial^i \in \mathbb{Z}[x][\partial]$ be a linear differential operator such that $\deg(a_i) \leq i$, for $i = 0, \dots, n$. Then $\mathcal{L}$ has a nonzero polynomial solution if and only if the indicial polynomial at infinity u_β has a positive integer root. Moreover, if ℓ denotes a bound on the bit-size of the coefficients of the $a_i(x)$, then we can detect if $\mathcal{L}$ has a nonzero polynomial solution using $\mathcal{O}_{\log}(n^2\,(\ell + n))$ bit operations.*

Before giving the proof, let us point out a reformulation with a more analytic flavour of the hypothesis in Corollary 2: $x = \infty$ is a regular singular point and $a_0 \in \mathbb{Z}$.

Proof. We can suppose that $x = 0$ is an ordinary point, since a translation preserves the degrees of the coefficients $a_i(x)$ and it does not affect the dimension of the space of polynomial solutions of $\mathcal{L}$. Now, the existence of polynomial solutions is a consequence of the previous proposition. The complexity estimate follows from Lemma 2. $\square$

For singular points, the order o can be smaller than the order of the differential equation. The rank of the matrix $\mathcal{A}_k$ is always at least 1 because of the extra column that is added at the end. Thus another special case of this proposition is [12, Theorem 2] on recurrences of order 1.

We do not obtain a generalization of the polynomial complexity estimates in Corollary 2, since the indices i_j are not polynomially bounded in the size of the differential equation. However, in a case when the bound N on the degrees is huge and none of the other criteria has worked, it might be a good idea to spend some time on the singularities where the order o is small and the largest i_k not too large and compute the corresponding matrix. Also, the recurrences at irregular singular points have smaller order. This can be helpful even if the criterion does not apply.

Finally, we recall the useful criterion given by [22, Corollary 8.43]. For differential equations of order 1, the matter is reduced to checking whether a logarithmic derivative is that of a polynomial. This is done by partial fraction decomposition.

6 Experiments

We have implemented our algorithms **BinSplitPolySols** and **ModBsGsPolySols** in the computer algebra systems Maple v. 9.5 and respectively Magma v. 2.11-2. Our choice is motivated by the fact that both Magma and Maple provide implementations of fast integer and polynomial arithmetic (using Karatsuba and FFT multiplications). For instance, Maple now uses the GNU Multi Precision Arithmetic Library (GMP). This is important, since in our experiments over $\mathbb{Z}$, the computations require sometimes up to millions of bits. Similarly, Magma employs asymptotically fast algorithms for performing arithmetic with univariate polynomials over $\mathbb{F}_p$. These include Karatsuba (for degrees greater then 64) and FFT-based methods (for degrees greater than 128) for multiplication of polynomials. Again, this is crucial, since in our modular baby-step / giant-step algorithm, the theoretical gains are valid only in conjunction with fast polynomial multiplication.

We have implemented the basic algorithm in Maple. Since the performances of our implementation are very similar to those of Maple's function **PolynomialSolutions** from the **LinearFunctionalSystems** package, we made the choice to report only timings obtained with this function for the basic algorithm. We note that Maple provides another implementation of (a variant of) the basic algorithm for finding polynomial solutions, namely the function **polysols** from **DEtools** package, but **LFS** clearly outperforms **DEtools** on all the set of examples we considered. This is why we have chosen to display only the timings of **LFS** for comparisons.

The equations used in these tables are as follows: Example 1 is $\sqrt{N}xy''-xy'+Ny=0$; Example 2 is $(1-x^2)y''-2xy'+N(N+1)y=0$, where N is a perfect square. In both cases, the dimension of polynomial solutions is 1 and any nonzero solution has degree N. In the first example, the recurrence is of first order, in the second one it has order $o=2$, but only two terms of the recurrence are nonzero. Example 3 is $(x^2+2x+1)y'-(Nx+N-1)y=0$ and Example 4 is $2x^3y''+((3-2N)x^2+x)y'-(Nx+1)y=0$. The second one is taken from [12]. The first one has no nonzero polynomial solutions, but its indicial equation at infinity has N as root; the recurrence has order $o=2$. The second one has a 1-dimensional space of polynomial solutions and the recurrence has order 1. Finally, in Example 5 we consider a family of LDEs indexed by N of order $n=3$; the recurrence (2) has order $o=7$, the indicial equation is $(x-d)(x-N)=0$, but there is no solution in of degree N.

All the tests have been performed on the computers of the MEDICIS resource center[2], using a 2 GB, 2200+ AMD 64 Athlon processor. The timings given in the tables are in seconds.

The timings shown in these tables prove that the theoretical complexity estimations can be observed in practice:
– The cost of **LFS** is multiplied by more than 16 when the degree N is multiplied by 4. This is in agreement with the fact that the basic algorithm has complexity (at least) quadratic in N. Moreover, the memory requirements are also roughly proportional to N^2, and this naturally becomes prohibitive (the mention > 4Gb means that the execution was stopped after 4Gb of memory were exhausted.)
– The cost of **BinSplit** is multiplied by slightly more than 5 when the degree N is multiplied by 4. This accurately reflects the behaviour of the GMP's integer multiplication.
– The cost of **BsGs** is multiplied by slightly more than 2 when the degree N is multiplied by 4. Again, this is in line with the complexity estimates and shows that the polynomial multiplication we are using is quite good.
– When the recurrence has 2 terms (as in Examples 1 and 2) the algorithm **BinSplit** essentially computes scalar factorials; moreover, there is no linear algebra step. In the opposite case ($o>1$), **BinSplit** multiplies (pairs of) matrices of small size, but containing potentially huge integer entries. Moreover, the practical cost of the linear algebra step

[2]http://www.medicis.polytechnique.fr

	Example 1			Example 2		
N	L.F.S.	BinSplit	BsGs	L.F.S.	BinSplit	BsGs
2^4	.38e-1	.13e-1	.1e-1	.13e-1	.9e-1	.0
2^6	.18e-1	.15e-1	.0e-1	0.03	.11e-1	.0
2^8	.58e-1	.15e-1	.0e-1	0.05	.25e-1	.0
2^{10}	.21	.17e-1	.1e-1	0.21	.3e-1	.1e-1
2^{12}	1.39	.45e-1	.1e-1	0.92	.14	.0
2^{14}	26.87	.59e-1	.4e-1	6.48	.55	.2e-1
2^{16}	$>$ 4Gb	.29	.8e-1	174.19	2.75	.5e-1
2^{18}		1.47	.1	$>$ 4Gb	14.31	.11
2^{20}		7.63	.45		74.52	.25
2^{22}		42.07	1.08		398.6	.59
2^{24}		215.62	2.57		$>$ 1h	1.35
2^{26}			6.13			7.34
2^{28}			14.69			17.46
2^{30}			37.57			42.07
2^{32}			96.21			102.65

becomes far from negligible. An important improvement (not implemented yet) is to use Strassen's algorithm for integer matrices and extensions like Waksman's algorithm; the gain should be already visible on 2×2 matrices.
– The timings in Example 5 clearly shows the advantage of using the baby step/giant step method to first compute the actual largest degrees before computing the solutions. This way, we get an algorithm that is output-sensitive. Without this information, even though the polynomial solutions have moderate degrees (up to $d = 81$), **LFS** spends a lot of time in (uselessly) unravelling recurrences up to order $N = d^2$. (The entries * are estimated timings.)

7 Extensions

We now list a few direct extensions of this work, where the same algorithmic ideas can be applied, possibly after some preprocessing.

7.1 Inhomogeneous LDEs

The differential equation $\mathcal{L}(y) = Q(x)$ where Q is a polynomial gives rise to the same recurrence as (2), except that the right-hand side is nonzero for $i \leq \deg Q$. This leads to new affine constraints for the coefficients of power series of index up to $\deg Q + \alpha$. The rest of the computation is governed by (2) and can be dealt with by the methods we have described.

7.2 Parameterized Case

The problem is that of finding k-tuples $(\lambda_1, \ldots, \lambda_k)$ such that the differential equation $L(y) = \lambda_1 Q_1(x) + \cdots + \lambda_k Q_k(x)$ has polynomial solutions. Here, the Q_i's are given polyno-

	Example 3			Example 4		
N	L.F.S.	BinSplit	BsGs	L.F.S.	BinSplit	BsGs
2^2	.30e-1	.10e-1	.0	.22e-1	.4e-2	.0
2^4	.12e-1	.12e-1	.0	.22e-1	.2e-2	.0
2^6	.30e-1	.12e-1	.0	.216	.1e-2	.1e-1
2^8	.50e-1	.72e-1	.0	.7e-1	.2e-2	.1e-1
2^{10}	.2	.87e-1	.0	.54	.6e-2	.1e-1
2^{12}	1.10	.271	.1e-1	3.74	.3e-1	.3e-1
2^{14}	16.27	1.11	.2e-1	48.80	.9e-1	.6e-1
2^{16}	$>$ 4Gb	5.06	.5e-1	$>$ 4Gb	.54	.19
2^{18}		24.08	.12		3.15	.45
2^{20}		115.39	.27		16.72	1.09
2^{22}		416.81	.68		93.64	2.57
2^{24}		2199.9	1.57		506.81	6.17
2^{26}		$>$ 1h	3.77		$>$ 1h	14.84
2^{28}			9.25			35.91
2^{30}			22.85			88.14

mials. This problem is important in differential versions of Zeilberger's algorithm [3, 16]. As above, after $\alpha + \max_i \deg Q_i$ steps, the rest of the computation is as before and results in a system that is linear in the λ_i's.

7.3 Linear Recurrence Equations

It is not true in general that the coefficients of polynomial solutions of a linear recurrence obey a linear recurrence, if the polynomials are expressed in the monomial basis. However, it has been observed in [1] that the coefficients in a binomial basis do obey such a recurrence. Then, all the algorithmic techniques we have described here also apply. This gives fast algorithms for polynomial solutions of linear recurrences with polynomial coefficients having integer coefficients. As in the differential case, inhomogeneous and parameterized inhomogeneous equations can also be handled. This has an impact on various algorithms of computer algebra (Gosper, Zeilberger, Chyzak). We plan to come back to this application in future work.

References

[1] S. A. Abramov, M. Bronstein, and M. Petkovšek. On polynomial solutions of linear operator equations. In A. H. M. Levelt, editor, *Symbolic and Algebraic Computation*, pages 290–296, New York, 1995. ACM Press. Proc. of ISSAC'95, Montreal, Canada.

[2] S. A. Abramov and K. Yu. Kvashenko. Fast algorithms to search for the rational solutions of linear differential equations with polynomial coefficients. In Stephen M. Watt, editor, *Symbolic and Algebraic Computation*, pages 267–270, New York, 1991. ACM Press. Proc. of ISSAC'91, Bonn, Germany.

Example 5					
		LFS.	BinSplit		BsGs
d	N	Total	System	Total	Total
9	81	0.14	.3e-1	.4e-1	0.03
16	256	0.63	.3e-1	.5e-1	0.05
25	625	4.90	.4e-1	.7e-1	0.08
36	1296	30.14	.6e-1	.7e-1	0.13
49	2401	192	.7e-1	.8e-1	0.17
64	4096	746.94	.7e-1	.9e-1	0.23
81	6561	4098.17	.8e-1	.10	0.3
100	10000	> 6 h *	.97e-1	.12	0.38
121	14641	> 1 day *	.114	.137	0.52
144	20736	> 1 week *	.143	.179	0.7
169	28561	> 1 month *	.169	.203	0.9

[3] G. Almkvist and D. Zeilberger. The method of differentiating under the integral sign. *Journal of Symbolic Computation*, 10:571–591, 1990.

[4] M. A. Barkatou. A fast algorithm to compute the rational solutions of systems of linear differential equations. Technical Report Research Report 973-M, IMAG Grenoble, 1997.

[5] M. A. Barkatou. On rational solutions of systems of linear differential equations. *J. Symbolic Comput.*, 28(4-5):547–567, 1999.

[6] M. Beeler, R. W. Gosper, and R. Schroeppel. *HAKMEM.* Artificial Intelligence Memo No. 239. Massachusetts Institute of Technology, 1972. http://www.inwap.com/pdp10/hbaker/hakmem/hakmem.html.

[7] D. J. Bernstein. Fast multiplication and its applications. To appear in Buhler-Stevenhagen *Algorithmic number theory* book.

[8] P. B. Borwein. On the complexity of calculating factorials. *J. Algorithms*, 6(3):376–380, 1985.

[9] A. Bostan, P. Gaudry, and É. Schost. Linear recurrences with polynomial coefficients and application to integer factorization and Cartier-Manin operator, May 2004. 29 pages. Submitted.

[10] A. Bostan, P. Gaudry, and É. Schost. Linear recurrences with polynomial coefficients and computation of the Cartier-Manin operator on hyperelliptic curves. In *International Conference on Finite Fields and Applications (Toulouse, 2003)*, volume 2948 of *Lecture Notes in Computer Science*, pages 40–58. Springer–Verlag, 2004.

[11] A. Bostan and É. Schost. Polynomial evaluation and interpolation on special sets of points. *Journal of Complexity*, 2005. Festschrift for Arnold Schönhage, to appear.

[12] D. Boucher. About the polynomial solutions of homogeneous linear differential equations depending on parameters. In Samuel S. Dooley, editor, *Symbolic and Algebraic*

Computation, pages 261–268, New York, 1999. ACM Press. Proc. ISSAC'99, Vancouver, Canada.

[13] R. P. Brent. Fast multiple-precision evaluation of elementary functions. *J. Assoc. Comput. Mach.*, 23(2):242–251, 1976.

[14] R. P. Brent. Multiple-precision zero-finding methods and the complexity of elementary function evaluation. In *Analytic computational complexity (Proc. Sympos., Carnegie-Mellon Univ., Pittsburgh, Pa., 1975)*, pages 151–176. Academic Press, New York, 1976.

[15] D. V. Chudnovsky and G. V. Chudnovsky. Approximations and complex multiplication according to Ramanujan. In *Ramanujan revisited (Urbana-Champaign, Ill., 1987)*, pages 375–472. Academic Press, Boston, MA, 1988.

[16] F. Chyzak. An extension of Zeilberger's fast algorithm to general holonomic functions. *Discrete Mathematics*, 217(1-3):115–134, 2000.

[17] F. Chyzak, Ph. Dumas, H. Le, J. Martins, M. Mishna, and B. Salvy. Taming apparent singularities via Ore closure. Preprint, July 2004.

[18] Th. Cluzeau. *Algorithmique modulaire des équations différentielles linéaires.* PhD thesis, Université de Limoges, 2004.

[19] G. Estrin. Organization of computer systems—the fixed plus variable structure computer. In *AFIPS conference proceedings*, volume 17, pages 33–40, 1960. Proc. of the Western Joint Computer Conference.

[20] J. von zur Gathen and J. Gerhard. Fast algorithms for Taylor shifts and certain difference equations. In Wolfgang W. Küchlin, editor, *Symbolic and Algebraic Computation*, pages 40–47, New York, 1997. ACM Press. Proc. ISSAC'97, Maui, Hawaii, United States.

[21] J. von zur Gathen and J. Gerhard. *Modern computer algebra.* Cambridge University Press, New York, 1999.

[22] J. Gerhard. In *Modular algorithms in symbolic summation and symbolic integration*, number 3218 in Lecture Notes in Computer Science. Springer, 2004. PhD thesis, Uni-Gesamthochschule Paderborn, 2001.

[23] G. H. Golub and Charles F. Van Loan. *Matrix computations.* Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[24] P. M. Kogge and H. S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Computers*, C-22:786–793, 1973.

[25] J. Liouville. Second mémoire sur la détermination des intégrales dont la valeur est algébrique. *Journal de l'École polytechnique*, (Cahier 14):149–193, 1833.

[26] F. M. Marotte. *Les équations différentielles linéaires et la théorie des groupes.* PhD thesis, Faculté des Sciences de Paris, 1898.

[27] A. Schönhage, A. F. W. Grotefeld, and E. Vetter. *Fast algorithms.* Bibliographisches Institut, Mannheim, 1994. A multitape Turing machine implementation.

[28] M. F Singer. Liouvillian solutions of n-th order homogeneous linear differential equations. *American Journal of Mathematics*, 103(4):661–682, 1981.

[29] M. F. Singer and F. Ulmer. Linear differential equations and products of linear forms. *Journal of Pure and Applied Algebra*, 117/118:549–563, 1997.

[30] V. Strassen. Einige Resultate über Berechnungskomplexität. *Jahresbericht d. Deutschen Mathem.-Vereinigung*, 78(1):1–8, 1976.

[31] M. van Hoeij, J.-F. Ragot, F. Ulmer, and J.-A. Weil. Liouvillian solutions of linear differential equations of order three and higher. *Journal of Symbolic Computation*, 28(4-5):589–609, 1999.