Fast computation of power series solutions of systems of differential equations*

A. Bostan† F. Chyzak† F. Ollivier‡ B. Salvy† É. Schost‡ A. Sedoglavic§

Abstract

We propose algorithms for the computation of the first N terms of a vector (or a full basis) of power series solutions of a linear system of differential equations at an ordinary point, using a number of arithmetic operations that is quasi-linear with respect to N. Similar results are also given in the non-linear case. This extends previous results obtained by Brent and Kung for scalar differential equations of order 1 and 2.

1 Introduction

The efficient computation of power series is a classical problem of symbolic computation. Using fast multiplication algorithms and Newton iteration or other types of divide-and-conquer techniques, quasi-optimal algorithms have been developed for numerous problems, notably by Brent and Kung in [8]. Here quasi-optimal means that if fast Fourier transform is used, the number of arithmetic operations is linear in the number of coefficients to be computed, up to logarithmic factors. The advent of fast computers and good implementations of fast multiplication algorithms makes these algorithms more and more relevant to practical computations.

We are interested in the efficient computation of a large number of coefficients of power series solutions of (systems of) differential equations. The efficiency is measured by the number of arithmetic operations.

This problem arises in combinatorics, where the desired power series are generating functions. Differential equations arise naturally from ordered structures, like m-ary search trees [23] or quadtrees [12]. Another source of differential equations arises from random generation by the recursive method [13]. There, combinatorial specifications are translated into differential-algebraic systems in a large number of unknowns. In these combinatorial contexts, the coefficients of the series are integers whose bit size grows linearly with the index. Then, our algorithms that use few arithmetic operations also perform well in terms of bit complexity and improve upon the previously known algorithms.

Another class of applications comes from numerical analysis, where high order expansions are used to produce Padé approximants with good convergence properties. If the coefficients size does not grow too fast, or if the computations are numerically stable and are performed using floating point arithmetic, then again our algorithms will perform faster than previous ones.

In many cases non-linear systems can be reduced to linear ones (see §5). We thus spend most of our effort on the linear case. The best algorithm known previously, although quasi-optimal in the number of coefficients, is more than exponential in the order of the equation or the dimension of the system [8, 35]. Its complexity is $\mathcal{O}(r^r N \log N)$ for the computation of the first N coefficients of the power series solution of a linear differential equation

$$a_r(t)y^{(r)}(t) + \cdots + a_1(t)y'(t) + a_0(t)y(t) = 0, \tag{1.1}$$

given r initial conditions and the first N coefficients of the power series $a_r, \dots, a_0$.

It turns out to be easier to make a divide-and-conquer approach work if we compute a full basis of solutions of a system instead of only one specified solution. Another obstacle to the use of a Newton iteration is the non-commutativity of the matrices that enter the iterations. We show that this difficulty can be overcome by computing not only the expansion of a basis (in the form of a fundamental matrix of solutions), but also that of the inverse of this matrix, in a simultaneous iteration. This way, we achieve a complexity which is quasi-optimal with respect to the precision and less than cubic in the order (more precise estimates are given below). These results for systems have consequences for single equations that we explore as well. We also offer alternative algorithms in this case that behave better with respect to the order, at the expense of a logarithmic factor in the number of desired terms. Finally, for special classes of coefficients, it is possible to design even faster algorithms. We recall the classical results when the coefficients are polynomials and we give new algorithms for the case

*This work was supported in part by the French National Agency for Research (ANR Gecko).

†ALGO, Inria Rocquencourt, 78153 Le Chesnay, France.

‡LIX, École polytechnique, 91128 Palaiseau, France.

§LIFL, Université Lille I, 59655 Villeneuve d'Ascq, France.

of constant coefficients whose complexity depends on the order of the equation (or dimension of the system) only logarithmically.

We now review more precisely the contents of this article. The coefficients of the series $a_0(t), \dots, a_r(t)$ in (1.1) belong to a field $\mathbb{K}$ (this field can be thought of as being the field of rational numbers $\mathbb{Q}$ or a finite field) and the arithmetic operations in $\mathbb{K}$ are counted at unit cost. Under the hypothesis that $t = 0$ is an ordinary point for Equation (1.1) (i.e., $a_r(0) \neq 0$), we give efficient algorithms taking as input the first N terms of the power series $a_0(t), \dots, a_r(t)$ and answering the following algorithmic questions:

i. find the first N coefficients of the r elements of a basis of power series solutions of (1.1);

ii. given initial conditions $\alpha_0, \dots, \alpha_{r-1}$ in $\mathbb{K}$, find the first N coefficients of the unique solution $y(t)$ in $\mathbb{K}[[t]]$ of Equation (1.1) satisfying

$$y(0) = \alpha_0, \quad y'(0) = \alpha_1, \quad \dots, \quad y^{(r-1)}(0) = \alpha_{r-1}.$$

More generally, we also treat linear first-order systems of differential equations. From the data of initial conditions v in $\mathcal{M}_{r\times r}(\mathbb{K})$ (resp. $\mathcal{M}_{r\times 1}(\mathbb{K})$) and of the first N coefficients of each entry of the matrices A and B in $\mathcal{M}_{r\times r}(\mathbb{K}[[t]])$ (resp. b in $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$), we propose algorithms that compute the first N coefficients:

I. of a fundamental solution Y in $\mathcal{M}_{r\times r}(\mathbb{K}[[t]])$ of $Y' = AY + B$, with $Y(0) = v$, $\det Y(0) \neq 0$;

II. of the unique solution $y(t)$ in $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$ of $y' = Ay + b$, satisfying $y(0) = v$.

Obviously, if an algorithm of algebraic complexity C (i.e., using C arithmetic operations in $\mathbb{K}$) is available for problem **II**, then applying it r times solves problem **I** in time $r\,\mathsf{C}$, while applying it to a companion matrix solves problem **ii** in time C and problem **i** in $r\,\mathsf{C}$. Conversely, an algorithm solving **i** (resp. **I**) also solves **ii** (resp. **II**) within the same complexity, plus that of a linear combination of series. Our reason for distinguishing the four problems **i, ii, I, II** is that in many cases, we are able to give algorithms of better complexity than obtained by these reductions.

The most popular way of solving problems **i**, **ii**, **I**, and **II** is the method of undetermined coefficients. It requires $\mathcal{O}(r^2N^2)$ operations in $\mathbb{K}$ for problem **i** and $\mathcal{O}(rN^2)$ operations in $\mathbb{K}$ for **ii**. Regarding the dependence in N, this is certainly too expensive compared to the size of the output, that is only linear in N in both cases. On the other hand, verifying the correctness of the output for **ii** (resp. **i**) already requires a number of operations in $\mathbb{K}$ that is linear (resp. quadratic) in r: thus there is little hope of improving the dependence in r. Similarly, for problems **I** and **II**, the method of undetermined coefficients requires $\mathcal{O}(N^2)$ multiplications of $r \times r$ scalar matrices (resp. of scalar matrix-vector products in size r), leading to a computational cost that is reasonable with respect to r, but not to N.

By contrast, the algorithms proposed in this article have costs that are linear (up to logarithmic factors) in the complexity $\mathsf{M}(N)$ of polynomial multiplication in degree less than N over $\mathbb{K}$. Using Fast Fourier Transform (FFT) these costs become nearly linear, up to polylogarithmic factors, with respect to N, for all of the four problems above (precise complexity results are stated below). Up to these polylogarithmic terms in N, this estimate is probably not far from the lower algebraic complexity one can expect: indeed, the mere check of the correctness of the output requires, in each case, a computational effort proportional to N.

In Table 1 we gather the complexity estimates corresponding to the best known solutions for each of the problems **i**, **ii**, **I**, and **II** under the hypothesis $N \gg r$ (precise statements are given in Thms. 1 and 2 below). Apart from the general case of power series coefficients, we distinguish two other important cases of special coefficients (constant and polynomial), for which better results can be obtained. In the polynomial coefficients case (third column), these results are well known. In the constant coefficients case, our results improve upon existing algorithms. The last column of Table 1 displays the size, in terms of number of coefficients, of the output solution. This size represents an obvious lower bound complexity; observe that for each problem, the cost of our algorithms are quite close from that lower bound, with respect to both parameters N and r.

We now give a more detailed account of the contributions of this article.

1.1 Newton Iteration.

In the case of first-order equations ($r = 1$), Brent and Kung have shown in [8] (see also [16, 20]) that the problems can be solved with complexity $\mathcal{O}(\mathsf{M}(N))$ by means of a formal Newton iteration. Their algorithm is based on the fact that solving the first-order differential equation $y'(t) = a(t)y(t)$, with $a(t)$ in $\mathbb{K}[[t]]$ is equivalent to computing the *power series exponential* $\exp(\int a(t))$. This equivalence is no longer true in the case of a system $Y' = A(t)Y$ (where $A(t)$ is a power series matrix): for non-commutativity reasons, the matrix exponential $Y(t) = \exp(\int A(t))$ is not a solution of $Y' = A(t)Y$.

Brent and Kung suggested a way to extend their result to higher orders, and van der Hoeven [35] showed that their algorithm has complexity $\mathcal{O}(r^r\,\mathsf{M}(N))$. This

Problem (input, output)	power series coefficients	polynomial coefficients	constant coefficients	output size
i (equation, basis)	$\mathcal{O}(\mathsf{MM}(r,N))$ ⋆	$\mathcal{O}(dr^2N)$	$\mathcal{O}(rN)$ ⋆	rN
ii (equation, one solution)	$\mathcal{O}(r\,\mathsf{M}(N)\log N)$ ⋆	$\mathcal{O}(drN)$	$\mathcal{O}(\mathsf{M}(r)N/r)$ ⋆	N
I (system, basis)	$\mathcal{O}(\mathsf{MM}(r,N))$ ⋆	$\mathcal{O}(dr^\omega N)$	$\mathcal{O}(r\mathsf{M}(r)N)$ ⋆	r^2N
II (system, one solution)	$\mathcal{O}(r^2\,\mathsf{M}(N)\log N)$ ⋆	$\mathcal{O}(dr^2N)$	$\mathcal{O}(\mathsf{M}(r)N)$ ⋆	rN

Table 1: Complexity of solving linear differential equations/systems for $N \gg r$. Entries marked with a '$\star$' correspond to new results.

is good with respect to N, but the exponential dependence in the order r is unacceptable.

Instead, we devise in §2 a specific Newton iteration for $Y' = A(t)Y$. Thus we solve problems **i** and **I** in $\mathcal{O}(\mathsf{MM}(r,N))$, where $\mathsf{MM}(r,N)$ is the number of operations in $\mathbb{K}$ required to multiply $r \times r$ matrices with polynomial entries of degree less than N. For instance, when $\mathbb{K} = \mathbb{Q}$, this is $\mathcal{O}(r^\omega N + r^2\mathsf{M}(N))$, where r^ω can be seen as an abbreviation for $\mathsf{MM}(r,1)$, see §1.5 below.

1.2 Divide-and-conquer. The resolution of problems **i** and **I** by Newton iteration relies on the fact that a whole basis is computed. When dealing with problems **ii** and **II**, we do not know how to preserve this algorithmic structure while simultaneously saving a factor r.

To solve problems **ii** and **II**, we therefore propose in §3 an alternative algorithm, whose complexity is also nearly linear in N. It is not quite as good, being in $\mathcal{O}(\mathsf{M}(N)\log N)$), but its dependence in the order r is better: linear for **i** and quadratic for **ii**. In a different model of computation with power series, based on the so-called *relaxed multiplication*, van der Hoeven briefly outlines another algorithm [35, §4.5.2] solving problem **ii** in $\mathcal{O}(r\,\mathsf{M}(N)\log N)$. To our knowledge, this result cannot be transferred to the usual model of power series multiplication (called zealous in [35]).

We use a divide-and-conquer technique similar to that used in the fast Euclidean algorithm [19, 29, 34]. For instance, to solve problem **ii**, our algorithm divides it into two similar problems of halved size. The key point is that the lowest coefficients of the solution $y(t)$ only depend on the lowest coefficients of the coefficients a_i. Our algorithm first computes the desired solution $y(t)$ at precision only $N/2$, then it recovers the remaining coefficients of $y(t)$ by recursively solving at precision $N/2$ a new differential equation. The main idea of this second algorithm is close to that used for solving first-order difference equations in [14].

We encapsulate our main complexity results in Thm. 1 below. When FFT is used, the functions $\mathsf{M}(N)$ and $\mathsf{MM}(r,N)$ have, up to logarithmic terms, a nearly linear growth in N, see §1.5. Thus, the results in the following theorem are quasi-optimal.

THEOREM 1. *Let N and r be two positive integers and let $\mathbb{K}$ be a field of characteristic zero or at least N. Then we can solve:*

(a) problems **i** *and* **I** *in $\mathcal{O}(\mathsf{MM}(r,N))$ operations in $\mathbb{K}$;*

(b) problem **ii** *in $\mathcal{O}(r\,\mathsf{M}(N)\log N)$ operations in $\mathbb{K}$;*

(c) problem **II** *in $\mathcal{O}\left(r^2\,\mathsf{M}(N)\log N\right)$ operations in $\mathbb{K}$.*

1.3 Special Coefficients. For special classes of coefficients, we give different algorithms of better complexity. We isolate two important classes of equations: that with constant coefficients and that with polynomial coefficients. In the case of constant coefficients, our algorithms are based on the use of the Laplace transform, that allows us to reduce the resolution of differential equations with constant coefficients to manipulations with rational functions. In the case of polynomial coefficients, we exploit the linear recurrence satisfied by the coefficients of solutions. The complexity results are summarized in the following theorem.

THEOREM 2. *Let N and r be two positive integers and let $\mathbb{K}$ be a field of characteristic zero or at least N. Then, for differential equations and systems with constant coefficients, we can solve:*

(a) problem **i** *in $\mathcal{O}(rN)$ operations in $\mathbb{K}$;*

(b) problem **ii** *in $\mathcal{O}(\mathsf{M}(r)(1+N/r))$ operations in $\mathbb{K}$;*

(c) problem **I** *in $\mathcal{O}\left(r^{\omega+1}\log r + r\mathsf{M}(r)N\right)$ operations in $\mathbb{K}$;*

(d) problem **II** *in $\mathcal{O}(r^\omega\log r + \mathsf{M}(r)N)$ operations in $\mathbb{K}$.*

1.4 Non-linear Systems.

As an important consequence of Thm. 1, we improve the known complexity results for the problem of solving *non-linear* systems of differential equations. To do so, we use a classical reduction technique from the non-linear to the linear case, see for instance [28, §25] and [8, §5.2] or [21] in a combinatorial context. For simplicity, we only consider non-linear systems of first order. There is no loss of generality in doing so: more general cases can be reduced to that one by adding new unknowns and possibly differentiating once. The following result generalizes [8, Thm. 5.1]. If $F = (F_1, \dots, F_r)$ is a differentiable function bearing on r variables $y_1, \dots, y_r$, we use the notation $\mathbf{Jac}(F)$ for the Jacobian matrix $(\partial F_i/\partial y_j)_{1 \le i,j \le r}$.

THEOREM 3. *Let $N, r \in \mathbb{N}$, let $\mathbb{K}$ be a field of characteristic 0 or at least N and let φ denote $(\varphi_1, \dots, \varphi_r)$, where the $\varphi_i(t, y)$ are multivariate power series in $\mathbb{K}[[t, y_1, \dots, y_r]]$.*

Let $\mathsf{L} : \mathbb{N} \to \mathbb{N}$ be such that the first n terms of the compositions $\varphi(t, s(t))$ and $\mathbf{Jac}(\varphi)(t, s(t))$ can be computed in $\mathsf{L}(n)$ operations in $\mathbb{K}$, for all $n \in \mathbb{N}$ and for all $s(t)$ in $\mathbb{K}[[t]]^r$.

Suppose in addition that the function $n \mapsto \mathsf{L}(n)/n$ is increasing. Given initial conditions v in $\mathcal{M}_{r\times 1}(\mathbb{K})$, if the differential system

$$y' = \varphi(t, y), \qquad y(0) = v,$$

admits a solution in $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$, then the first N terms of such a solution $y(t)$ can be computed in

$$\mathcal{O}\left(\mathsf{L}(N) + \min(\mathsf{MM}(r, N), r^2\mathsf{M}(N)\log N)\right)$$

operations in $\mathbb{K}$.

Werschulz [36, Thm. 3.2] gave an algorithm solving the same problem using the integral Volterra-type equation technique described in [28, pp. 172–173]. With our notation, his algorithm uses $\mathcal{O}\left(\mathsf{L}(N) + r^2 N\,\mathsf{M}(N)\right)$ operations in $\mathbb{K}$ to compute a solution at precision N. Thus, our algorithm is an improvement for cases where $\mathsf{L}(N)$ is known to be subquadratic with respect to N.

The best known algorithms for power series composition in $r \ge 2$ variables require, at least on "generic" entries, a number $\mathsf{L}(n) = \mathcal{O}(n^{r-1}\mathsf{M}(n))$ of operations in $\mathbb{K}$ to compute the first n coefficients of the composition [7, §3]. This complexity is nearly optimal with respect to the size of a generic input. By contrast, in the univariate case, the best known result [8, Th. 2.2] is $\mathsf{L}(n) = \mathcal{O}(\sqrt{n\log n}\,\mathsf{M}(n))$. For special entries, however, better results can be obtained, already in the univariate case: exponentials, logarithms, powers of univariate power series can be computed [6, §13] in $\mathsf{L}(n) = \mathcal{O}(\mathsf{M}(n))$. As a consequence, if φ is an r-variate sparse polynomial with m monomials of *any* degree, then $\mathsf{L}(n) = \mathcal{O}(mr\,\mathsf{M}(n))$.

Another important class of systems with a subquadratic $\mathsf{L}(N)$ is provided by *rational systems*, where each φ_i is in $\mathbb{K}(y_1, \dots, y_r)$. Supposing that the complexity of evaluation of φ is bounded by L (i.e., for any point z in $\mathbb{K}^r$ at which φ is well defined, the value $\varphi(z)$ can be computed using at most L operations in $\mathbb{K}$), then, the Baur-Strassen theorem [1] implies that the complexity of evaluation of the Jacobian $\mathbf{Jac}(\varphi)$ is bounded by $5L$, and therefore, we can take $\mathsf{L}(n) = \mathsf{M}(n)L$ in the statement of Thm. 3.

1.5 Basic Complexity Notation.

Our algorithms ultimately use, as a basic operation, multiplication of matrices with entries that are polynomials (or truncated power series). Thus, to estimate their complexities in a unified manner, we use a function $\mathsf{MM} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that any two $r \times r$ matrices with polynomial entries in $\mathbb{K}[t]$ of degree less than d can be multiplied using $\mathsf{MM}(r, d)$ operations in $\mathbb{K}$. In particular, $\mathsf{MM}(1, d)$ represents the number of base field operations required to multiply two polynomials of degree less than d, while $\mathsf{MM}(r, 1)$ is the arithmetic cost of scalar $r \times r$ matrix multiplication. For simplicity, we denote $\mathsf{MM}(1, d)$ by $\mathsf{M}(d)$ and we have $\mathsf{MM}(r, 1) = \mathcal{O}(r^\omega)$, where $2 \le \omega \le 3$ is the so-called *exponent of matrix multiplication*, see, e.g., [9] and [15].

Using the algorithms of [30, 10], one can take $\mathsf{M}(d)$ in $\mathcal{O}(d\log d\log\log d)$; over fields supporting FFT, one can take it in $\mathcal{O}(d\log d)$. By [10] we can always choose $\mathsf{MM}(r, d)$ in $\mathcal{O}(r^\omega\,\mathsf{M}(d))$, but better estimates are known in important particular cases. For instance, over fields of characteristic 0 or larger than $2d$, we have $\mathsf{MM}(r, d) = \mathcal{O}(r^\omega d + r^2\,\mathsf{M}(d))$, see [5, Th. 4]. To simplify the complexity analyses of our algorithms, we suppose that the multiplication cost function MM satisfies the following standard growth hypothesis for all integers d_1, d_2 and r:

$$\frac{\mathsf{MM}(r, d_1)}{d_1} \le \frac{\mathsf{MM}(r, d_2)}{d_2} \qquad \text{if } d_1 \le d_2. \tag{1.2}$$

In particular, Equation (1.2) implies the inequalities

$$2\mathsf{M}(2^{\kappa-1}) + 4\mathsf{M}(2^{\kappa-2}) + \ldots + 2^\kappa\mathsf{M}(1) \le \kappa\mathsf{M}(2^\kappa),$$
$$\mathsf{MM}(r, 2^\kappa) + \mathsf{MM}(r, 2^{\kappa-1}) + \ldots + \mathsf{MM}(r, 1) \le 2\mathsf{MM}(r, 2^\kappa).$$

These inequalities are crucial to prove the estimates in Thms. 1 and 3. Note that when the available multiplication algorithm is slower than quasi-linear (e.g., Karatsuba or naive multiplication), then the factor κ in the right-hand side of the first inequality can be replaced by a constant and thus the estimates $\mathsf{M}(N)\log N$ in our complexities become $\mathsf{M}(N)$ in those cases.

1.6 Notation for Truncation. It is recurrent in algorithms to split a polynomial into a lower and a higher part. To this end, the following notation proves convenient. Given a polynomial f, the remainder and quotient of its Euclidean division by t^k are respectively denoted $\lceil f \rceil^k$ and $\lfloor f \rfloor_k$. Another occasional operation consists in taking a middle part out of a polynomial. To this end, we let $[f]_k^l$ denote $\left\lfloor \lceil f \rceil^l \right\rfloor_k$. Furthermore, we shall write $f = g \mod t^k$ when two polynomials or series f and g agree up to degree $k-1$ included. To get a nice behaviour of integration with respect to truncation orders, all primitives of series are chosen with zero as their constant coefficient.

2 Newton Iteration for Systems of Linear Differential Equations

Let $Y'(t) = A(t)Y(t) + B(t)$ be a linear differential system, where $A(t)$ and $B(t)$ are $r \times r$ matrices with coefficients in $\mathbb{K}[[t]]$. Given an invertible scalar matrix Y_0, an integer $N \geq 1$, and the expansions of A and B up to precision N, we show in this section how to compute efficiently the power series expansion at precision N of the unique solution of the Cauchy problem

$$Y'(t) = A(t)Y(t) + B(t) \quad \text{and} \quad Y(0) = Y_0.$$

This enables us to answer problems **I** and **i**, the latter being a particular case of the former (through the application to a companion matrix).

2.1 Homogeneous Case. We begin by designing a Newton-type iteration to solve the homogeneous system $Y' = A(t)Y$. The classical Newton iteration to solve an equation $\phi(y) = 0$ is $Y_{\kappa+1} = Y_\kappa - U_\kappa$, where U_κ is a solution of the linearized equation $D\phi|_{Y_\kappa} \cdot U = \phi(Y_\kappa)$ and $D\phi|_{Y_\kappa}$ is the differential of ϕ at Y_κ. We apply this idea to the map $\phi : Y \mapsto Y' - AY$. Since ϕ is linear, it is its own differential and the equation for U becomes

$$U' - AU = Y_\kappa' - AY_\kappa.$$

Taking into account the proper orders of truncation and using Lagrange's method of variation of parameters [18], we are thus led to the iteration

$$\begin{cases} Y_{\kappa+1} &= Y_\kappa - \lceil U_\kappa \rceil^{2^{\kappa+1}}, \\ U_\kappa &= Y_\kappa \int \lceil Y_\kappa^{-1} \rceil^{2^{\kappa+1}} \left(Y_\kappa' - \lceil A \rceil^{2^{\kappa+1}} Y_\kappa \right). \end{cases}$$

Thus we need to compute (approximations of) the solution Y and its inverse simultaneously. Now, a well-known Newton iteration for the inverse Z of Y is

$$(2.3) \qquad Z_{\kappa+1} = \lceil Z_\kappa + Z_\kappa(I_r - YZ_\kappa) \rceil^{2^{\kappa+1}}.$$

Introduced by Schulz [31] in the case of real matrices; its version for matrices of power series is given, e.g., in [24].

```
SolveHomDiffSys($A, N, Y_0$)

Input: $Y_0, A_0, \ldots, A_{N-2}$ in $\mathcal{M}_{r\times r}(\mathbb{K})$, $A = \sum A_i t^i$.
Output: $Y = \sum_{i=0}^{N-1} Y_i t^i$ in $\mathcal{M}_{r\times r}(\mathbb{K})[t]$ such that
$Y' = AY \mod t^{N-1}$, and $Z = Y^{-1} \mod t^{N/2}$.

$Y \leftarrow (I_r + tA_0)Y_0$
$Z \leftarrow Y_0^{-1}$
$m \leftarrow 2$
while $m \leq N/2$ do
    $Z \leftarrow Z + \lceil Z(I_r - YZ) \rceil^m$
    $Y \leftarrow Y - \left\lceil Y \left( \int Z(Y' - \lceil A \rceil^{2m-1} Y) \right) \right\rceil^{2m}$
    $m \leftarrow 2m$
return $Y, Z$
```

Figure 1: Solving the Cauchy problem $Y' = A(t)Y$, $Y(0) = Y_0$ by Newton iteration.

Putting these considerations together, we arrive at the algorithm SolveHomDiffSys in Fig. 1, whose correctness easily follows from Lemma 1 below. Remark that in the scalar case ($r = 1$) algorithm SolveHomDiffSys coincides with the recent algorithm for power series exponential proposed by Hanrot and Zimmermann [17]; see also [2]. In the case $r > 1$, ours is a nontrivial generalization of the latter. Because it takes primitives of series at precision N, algorithm SolveHomDiffSys requires that the elements $2, 3, \ldots, N-1$ be invertible in $\mathbb{K}$. Its complexity C satisfies the recurrence

$$\mathsf{C}(m) = \mathsf{C}(m/2) + \mathcal{O}(\mathsf{MM}(r, m)).$$

This implies, by using the growth hypotheses on MM, that $\mathsf{C}(N) = \mathcal{O}(\mathsf{MM}(r, N))$. This proves the first assertion of Thm. 1.

Lemma 1. *Let m be an even integer. Suppose that $Y_{(0)}$, $Z_{(0)}$ in $\mathcal{M}_{r\times r}(\mathbb{K}[t])$ satisfy*

$$I_r - Y_{(0)}Z_{(0)} = 0 \bmod t^{m/2}, \quad Y_{(0)}' - AY_{(0)} = 0 \bmod t^{m-1},$$

and that they are of degree less than $m/2$ and m, respectively. Define

$$Z := \lceil Z_{(0)} (2I_r - Y_{(0)}Z_{(0)}) \rceil^m \qquad \textit{and}$$

$$Y := \left\lceil Y_{(0)} \left(I_r - \int Z(Y_{(0)}' - AY_{(0)}) \right) \right\rceil^{2m}.$$

```
SolveInhomDiffSys($A, B, N, Y_0$)

Input: $Y_0, A_0, \ldots, A_{N-2}$ in $\mathcal{M}_{r\times r}(\mathbb{K})$, $A = \sum A_i t^i$,
$B_0, \ldots, B_{N-2}$ in $\mathcal{M}_{r\times r}(\mathbb{K})$, $B(t) = \sum B_i t^i$.
Output: $Y_1, \ldots, Y_{N-1}$ in $\mathcal{M}_{r\times r}(\mathbb{K})$ such that
$Y = Y_0 + \sum Y_i t^i$ satisfies $Y' = AY + B \mod t^{N-1}$.

$\widetilde{Y}, \widetilde{Z} \leftarrow$ SolveHomDiffSys($A, N, Y_0$)
$\widetilde{Z} \leftarrow \widetilde{Z} + \lceil \widetilde{Z}(I_r - \widetilde{Y}\widetilde{Z}) \rceil^N$
$Y \leftarrow \lceil \widetilde{Y} \int (\widetilde{Z}B) \rceil^N$
$Y \leftarrow Y + \widetilde{Y}$
return $Y$
```

Figure 2: Solving the Cauchy problem $Y' = AY + B$, $Y(0) = Y_0$, by Newton iteration.

Then Y and Z satisfy the equations

(2.4) $\;I_r - YZ = 0 \bmod t^m,\; Y' - AY = 0 \bmod t^{2m-1}.$

Proof. By definition, $I_r - YZ$ is equal, modulo t^m, to

$$(I_r - Y_{(0)}Z_{(0)})^2 - (Y - Y_{(0)})Z_{(0)}(2I_r - Y_{(0)}Z_{(0)}).$$

Since by hypothesis $I_r - Y_{(0)}Z_{(0)}$ and $Y - Y_{(0)}$ are zero modulo $t^{m/2}$, the right-hand side is zero modulo t^m and this establishes the first formula in Equation (2.4). Similarly, write $Q = \int Z(Y'_{(0)} - AY_{(0)})$ and observe $Q = 0 \mod t^m$ to get the following equality modulo t^{2m-1}:

$$Y' - AY = (I - YZ)(Y'_{(0)} - AY_{(0)}) - (Y'_{(0)} - AY_{(0)})Q.$$

Now, $Y'_{(0)} - AY_{(0)}$ is 0 modulo t^{m-1}, while Q and $I_r - YZ$ are zero modulo t^m and therefore the right-hand side of the last equation is zero modulo t^{2m-1}, proving the last part of the lemma. □

2.2 General Case. We want to solve the system $Y' = AY + B$, where B is an $r \times r$ matrix with coefficients in $\mathbb{K}[[t]]$. Suppose that we have already computed the solution $\widetilde{Y}$ of the associate homogeneous system $\widetilde{Y}' = A\widetilde{Y}$, together with its inverse $\widetilde{Z}$. Then, by the method of variation of parameters, $Y_{(1)} = \widetilde{Y} \int \widetilde{Z}B$ is a particular solution of the inhomogeneous problem. Thus, the general solution has the form $Y = Y_{(1)} + \widetilde{Y}$.

Now, to compute the particular solution $Y_{(1)}$ at precision N, we need to know both $\widetilde{Y}$ and $\widetilde{Z}$ at the same precision N. To do this, we first apply the algorithm for the homogeneous case and iterate (2.3) once. The resulting algorithm is given in Fig. 2.

3 Divide-and-conquer Algorithm

We now give our second algorithm, that allows us to solve problems **ii** and **II** and to finish the proof of Thm. 1. First, we briefly sketch the main idea in the particular case of a homogeneous differential equation $\mathcal{L}y = 0$, where $\mathcal{L}$ is a linear differential operator in $\delta = t\frac{d}{dt}$ with coefficients in $\mathbb{K}[[t]]$. (The introduction of δ is only for pedagogical reasons.) The starting remark is that if a power series y is written as $y_0 + t^m y_1$, then $\mathcal{L}(\delta)y = \mathcal{L}(\delta)y_0 + t^m\mathcal{L}(\delta + m)y_1$. Thus, to compute a solution y of $\mathcal{L}(\delta)y = 0 \mod t^{2m}$, it suffices to determine the lower part of y as a solution of $\mathcal{L}(\delta)y_0 = 0 \mod t^m$, and then to compute the higher part y_1, as a solution of the inhomogeneous equation $\mathcal{L}(\delta + m)y_1 = -R \mod t^m$, where the rest R is computed so that $\mathcal{L}(\delta)y_0 = t^m R \mod t^{2m}$.

Our algorithm DivideAndConquer makes a recursive use of this idea. Since, during the recursions, we are naturally led to treat inhomogeneous equations of a slightly more general form than that of **II** we introduce the notation $\mathcal{E}(s, p, m)$ for the vector equation

$$ty' + (pI_r - tA)y = s \mod t^m.$$

The algorithm DivideAndConquer is described in Fig. 3. Choosing $p = 0$ and $s(t) = tb(t)$ we retrieve the equation of problem **II**. Our algorithm Solve to solve problem **II** is thus a specialization of DivideAndConquer, defined by making Solve(A, b, N, v) simply call DivideAndConquer($tA, tb, 0, N, v$). Its correctness relies on the following immediate lemma.

Lemma 2. *Let A in $\mathcal{M}_{r\times r}(\mathbb{K}[[t]])$, s in $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$, and let p, d in $\mathbb{N}$. Decompose $\lceil s \rceil^m$ into a sum $s_0 + t^d s_1$. Suppose that y_0 in $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$ satisfies the equation $\mathcal{E}(s_0, p, d)$, set R to be*

$$\lceil (ty'_0 + (pI_r - tA)y_0 - s_0)/t^d \rceil^{m-d},$$

and let y_1 in $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$ be a solution of the equation $\mathcal{E}(s_1 - R, p + d, m - d)$. Then the sum $y := y_0 + t^d y_1$ is a solution of the equation $\mathcal{E}(s, p, m)$.

The only divisions performed along algorithm Solve are by $1, \ldots, N-1$. As a consequence of this remark and of the previous lemma, we deduce the complexity estimates in the proposition below; for a general matrix A, this proves point (c) in Thm. 1, while the particular case when A is companion proves point (b).

Proposition 1. *Given the first m terms of the entries of $A \in \mathcal{M}_{r\times r}(\mathbb{K}[[t]])$ and of $s \in \mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$, given v in $\mathcal{M}_{r\times 1}(\mathbb{K})$, algorithm DivideAndConquer(A, s, p, m, v) computes the unique solution of the linear differential*

```
DivideAndConquer($A, s, p, m, v$)

Input: $A_0, \dots, A_{m-1}$ in $\mathcal{M}_{r\times r}(\mathbb{K})$, $A = \sum A_i t^i$,
$s_0, \dots, s_{m-1}, v$ in $\mathcal{M}_{r\times 1}(\mathbb{K})$, $s = \sum s_i t^i$, $p$ in $\mathbb{K}$.
Output: $y = \sum_{i=0}^{N-1} y_i t^i$ in $\mathcal{M}_{r\times 1}(\mathbb{K})[t]$ such that
$ty' + (pI_r - tA)y = s \mod t^m$, $y(0) = v$.

If $m = 1$ then
  if $p = 0$ then return $v$
  else return $p^{-1}s(0)$
else
  $d \leftarrow \lfloor m/2 \rfloor$
  $s \leftarrow \lceil s \rceil^d$
  $y_0 \leftarrow$ DivideAndConquer($A, s, p, d, v$)
  $R \leftarrow [s - ty_0' - (pI_r - tA)y_0]_d^m$
  $y_1 \leftarrow$ DivideAndConquer($A, R, p + d, m - d, v$)
  return $y_0 + t^d y_1$
```

Figure 3: Solving $ty' + (pI_r - tA)y = s \mod t^m$, $y(0) = v$, by divide-and-conquer.

system $ty' + (pI_r - tA)y = s \mod t^m$, $y(0) = v$, *using* $\mathcal{O}(r^2\,\mathsf{M}(m)\log m)$ *operations in* $\mathbb{K}$. *If A is a companion matrix, the cost reduces to* $\mathsf{C}(m) = \mathcal{O}(r\,\mathsf{M}(m)\log m)$.

Proof. The correctness of the algorithm follows from the previous lemma. The cost $\mathsf{C}(m)$ of the algorithm satisfies the recurrence

$$\mathsf{C}(m) = \mathsf{C}(\lfloor m/2 \rfloor) + \mathsf{C}(\lceil m/2 \rceil) + r^2\,\mathsf{M}(m) + \mathcal{O}(rm),$$

where the term $r^2\,\mathsf{M}(m)$ comes from the application of A to y_0 used to compute the rest R. From this recurrence, it is easy to infer that $\mathsf{C}(m) = \mathcal{O}(r^2\,\mathsf{M}(m)\log m)$. Finally, when A is a companion matrix, the vector R can be computed in time $O(r\,\mathsf{M}(m))$. This implies that in this case $\mathsf{C}(m) = \mathcal{O}(r\,\mathsf{M}(m)\log m)$. $\square$

4 Faster Algorithms for Special Coefficients

4.1 Constant Coefficients. For the particular case of constant coefficients, various algorithms have been proposed to solve problems **i**, **ii**, **I**, and **II**, see for instance [27, 25, 26, 22] and the references therein. Again, the most naive algorithm is based on the method of undetermined coefficients. On the other hand, most books on differential equations recommend to simplify the calculations using the Jordan form of matrices. The main drawback of that approach is that computations are done over the algebraic closure of the base field $\mathbb{K}$. We propose new algorithms of better complexity, that only need to perform operations in the base field $\mathbb{K}$.

We concentrate first on problems **ii, II** (computing a single solution of a first-order equation/system). Let A be an $r \times r$ constant matrix and let v be a vector of initial conditions. Given $N \geq 1$, we want to compute the first N coefficients of the series expansion of a solution y in $\mathcal{M}_{r\times 1}(\mathbb{K}[[t]])$ of $y' = Ay$, with $y(0) = v$.

Our algorithm uses $\mathcal{O}(r^\omega \log r + N\mathsf{M}(r))$ operations in $\mathbb{K}$ for a general constant matrix A (problem **II**) and only $\mathcal{O}(N\mathsf{M}(r)/r)$ operations in $\mathbb{K}$ in the case where A is a companion matrix (problem **ii**).

The idea to compute the truncated solution $y_N = \sum_{i=0}^{N-1} A^i v t^i/i!$ is to first compute its Laplace transform $z_N = \sum_{i=0}^{N-1} A^i v t^i$: indeed, one can switch from y_N to z_N using only $\mathcal{O}(Nr)$ operations in $\mathbb{K}$. Now, the vector z_N is the truncation at precision N of $z = \sum_{i\geq 0} A^i v t^i = (I - tA)^{-1}v$. By Cramer's rule, z is a vector of rational functions $z_i(t)$ with numerator and denominator of degree at most r. The idea is to first compute z as a rational function, and then to deduce its expansion modulo t^N. The first part of the algorithm does not depend on N and thus it can be seen as a precomputation. For instance, one can use [33, Corollary 12], to compute the rational form of z in complexity $\mathcal{O}(r^\omega \log r)$. In the second step of the algorithm, we have to expand r rational functions of degree at most r at precision N. Each such expansion can be performed using $\mathcal{O}(N\mathsf{M}(r)/r)$ operations in $\mathbb{K}$, see, e.g., the proof of [4, Prop. 1]. The total cost of the algorithm is thus $\mathcal{O}(r^\omega \log r + N\mathsf{M}(r))$. We give below a simplified variant with same complexity, avoiding the use of the algorithm in [33] for the precomputation step and relying instead on a technique that is classical for minimal polynomials computations [9].

```
1. Compute the vectors $v, Av, A^2v, A^3v, \dots, A^{2r}v$
   in $\mathcal{O}(r^\omega \log r)$, as follows:
   for $\kappa$ from 1 to $1 + \log r$ do
     (a) compute $A^{2^\kappa}$
     (b) compute $A^{2^\kappa} \times [v \,|\, Av \,|\, \cdots \,|\, A^{2^\kappa - 1}v]$, thus
         getting $[A^{2^\kappa}v \,|\, A^{2^\kappa+1}v \,|\, \cdots \,|\, A^{2^{\kappa+1}-1}v]$
2. For each $j = 1, \dots, r$:
     (a) recover the rational fraction whose series
         expansion is $\sum (A^i v)_j t^i$ by Padé approxi-
         mation in $\mathcal{O}(\mathsf{M}(r)\log r)$ operations
     (b) compute its expansion up to precision $t^N$
         in $\mathcal{O}(N\,\mathsf{M}(r)/r)$ operations
     (c) recover the expansion of $y$ from that of $z$,
         using $\mathcal{O}(N)$ operations.
```

A quick analysis yields the announced total cost of

$\mathcal{O}(r^\omega \log r + N\mathsf{M}(r))$ operations for problem **II**.

We now turn to the estimation of the cost for problems **i** and **I** (bases of solutions). For **i**, an obvious solution is to iterate r times our solution for **ii**. A better solution is based on the use of the Laplace transform and the remark that the first N terms of a whole basis of an order r recurrence with constant coefficients can be computed in $O(rN)$ operations [11, Prop. 4.2].

For **I**, an obvious solution is again to iterate r times our solution for problem **II**. This leads to the cost stated in point (c) of Thm. 1. Note that the exponent $\omega+1$ in the cost of the precomputation can be reduced to ω by a different approach, based on the precomputation of the Frobenius form of A [32]; for space limitation, we cannot give here the details of this alternative algorithm.

4.2 Polynomial Coefficients.

If the coefficients in one of the problems **i, ii, I**, and **II** are polynomials in $\mathbb{K}[t]$ of degree at most d, using the linear recurrence of order d satisfied by the coefficients of the solution seemingly yields the lowest possible complexity. Consider for instance problem **II**. Plugging $A = \sum_{i=0}^d t^i A_i$, $b = \sum_{i=0}^d t^i b_i$, and $y = \sum_{i\geq 0}^d t^i y_i$ in the equation $y' = Ay + b$, we arrive at the following recurrence

$$(d+k+1)y_{k+d+1} = A_d y_k + \cdots + A_0 y_{k+d} + b_{k+d},$$

that is valid for all $k \geq -d$. Thus, to compute $y_0, \ldots, y_N$, we need to perform Nd matrix-vector products; this is done using $\mathcal{O}(dNr^2)$ operations in $\mathbb{K}$. A similar analysis implies the other complexity estimates in the third column of Table 1.

5 Non-linear Systems of Differential Equations

Let $\varphi(t,y) = (\varphi_1(t,y), \ldots, \varphi_r(t,y))$, where each φ_i is a power series in $\mathbb{K}[[t, y_1, \ldots, y_r]]$. We consider the first-order non-linear system in y

$$(\mathcal{N}) \qquad \begin{cases} y_1'(t) = \varphi_1(t, y_1(t), \ldots, y_r(t)), \\ \qquad \vdots \\ y_r'(t) = \varphi_r(t, y_1(t), \ldots, y_r(t)). \end{cases}$$

To solve $(\mathcal{N})$, we use the classical technique of *linearization.* The idea is to attach, to an approximate solution y_0 of $(\mathcal{N})$, a linear system in the new unknown z,

$$(\mathcal{T}, y_0) \qquad z' = \mathbf{Jac}(\varphi)(y_0)z - y_0' + \varphi(y_0),$$

whose solutions serve to obtain a better approximation of an exact solution of $(\mathcal{N})$. Indeed, let us denote by $(\mathcal{N}_m), (\mathcal{T}_m)$ the systems $(\mathcal{N}), (\mathcal{T})$ where all the equalities are taken modulo t^m. Taylor's formula states that the expansion $\varphi(y+z) - \varphi(y) - \mathbf{Jac}(\varphi)(y)z$ is equal to 0 modulo z^2. If y is a solution of $(\mathcal{N}_m)$ and if z is a solution of $(\mathcal{T}_{2m}, y)$, then $y + z$ is a solution of $(\mathcal{N}_{2m})$. This justifies the correctness of Algorithm SolveNonLinearSys in Fig. 4.

To analyze the complexity of this algorithm, it suffices to remark that for each integer κ between 1 and $\lfloor \log N \rfloor$, one has to compute one solution of a linear inhomogeneous first-order system at precision 2^κ and to evaluate φ and its Jacobian on a series at the same precision. This concludes the proof of Thm. 3.

```
SolveNonLinearSys($\phi, v$)

Input: $N$ in $\mathbb{N}$, $\varphi(t,y)$ in $\mathbb{K}[[t, y_1, \ldots, y_r]]^r$, $v$ in $\mathbb{K}^r$
Output: first $N$ terms of a $y(t)$ in $\mathbb{K}[[t]]$ such that $y(t)' = \varphi(t, y(t)) \mod t^N$ and $y(0) = v$.

$m \leftarrow 1$
$y \leftarrow v$
while $m \leq N/2$ do
    $A \leftarrow \lceil \mathbf{Jac}(\varphi)(y) \rceil^{2m}$
    $b \leftarrow \lceil \varphi(y) - y' \rceil^{2m}$
    $z \leftarrow$ Solve$(A, b, 2m, 0)$
    $y \leftarrow y + z$
    $m \leftarrow 2m$
return $y$
```

Figure 4: Solving the non-linear $y' = \varphi(t,y)$, $y(0) = v$.

6 Implementation and Timings

We implemented our algorithms SolveDiffHomSys and Solve in Magma [3][1]. We used Magma's built-in polynomial arithmetic (using successively naive, Karatsuba, and FFT multiplication algorithms), as well as Magma's scalar matrix multiplication (of cubic complexity in the range of our interest). We give three tables of timings. First, we compare in Table 2 the performances of our algorithm SolveDiffHomSys with that of the naive quadratic algorithm, for computing a basis of (truncated power series) solutions of a homogeneous system. The order of the system varies from 2 to 16, while the precision required for the solution varies from 256 to 4096; the base field is $\mathbb{Z}/p\mathbb{Z}$, where p is a 32-bit prime.

Then we display in Figs. 5 and 6 the timings obtained respectively with algorithm SolveDiffHomSys and with the algorithm for polynomial matrix multiplication

[1] All the computations have been done on an Athlon processor at 2.2 GHz with 2 GB of memory of the MEDICIS ressource center http://medicis.polytechnique.fr.

$N \diagdown r$	2	4	8	16
256	0.02 vs. 2.09	0.08 vs. 6.11	0.44 vs. 28.16	2.859 vs. 168.96
512	0.04 vs. 8.12	0.17 vs. 25.35	0.989 vs. 113.65	6.41 vs. 688.52
1024	0.08 vs. 32.18	0.39 vs. 104.26	2.30 vs. 484.16	15 vs. 2795.71
2048	0.18 vs. 128.48	0.94 vs. 424.65	5.54 vs. 2025.68	36.62 vs. > 3 hours *
4096	0.42 vs. 503.6	2.26 vs. 1686.42	13.69 vs. 8348.03	92.11 vs. > 1/2 day *

Table 2: Computation of a basis of a linear homogeneous system with r equations, at precision N: comparison of timings (in seconds) between algorithm SolveDiffHomSys and the naive algorithm. Entries marked with a '$\star$' are estimated timings.

PolyMatMul that was used as a primitive of SolveDiffHomSys. The similar shapes of the two surfaces indicate that the complexity prediction of point (a) in Thm. 1 is well respected in our implementation: SolveDiffHomSys uses a constant number (between 4 and 5) of polynomial multiplications; note that the abrupt jumps at powers of 2 reflect the performance of Magma's FFT implementation of polynomial arithmetic.

In Fig. 7 we give the timings for the computation of one solution of a linear differential equation of order 2, 4, and 8, respectively, using our algorithm Solve in §3. Again, the shape of the three curves experimentally confirms the nearly linear behaviour established in point (b) of Thm. 1, both in the precision N and in the order r of the complexity of algorithm Solve. Finally, Fig. 8 displays the three curves from Fig. 7 together with the timings curve for the naive quadratic algorithm computing one solution of a linear differential equation of order 2. The conclusion is that our algorithm Solve becomes very early superior to the quadratic one.

We also implemented our algorithms of §4.1 for constant coefficients. For reasons of space limitation, we only provide a few experimental results for problem **II**. Over the same finite field, we computed: a solution of a linear system with $r = 8$ at precision $N \approx 10^6$ in 24.53s; one at doubled precision in doubled time 49.05s; one for doubled order $r = 16$ in doubled time 49.79s.

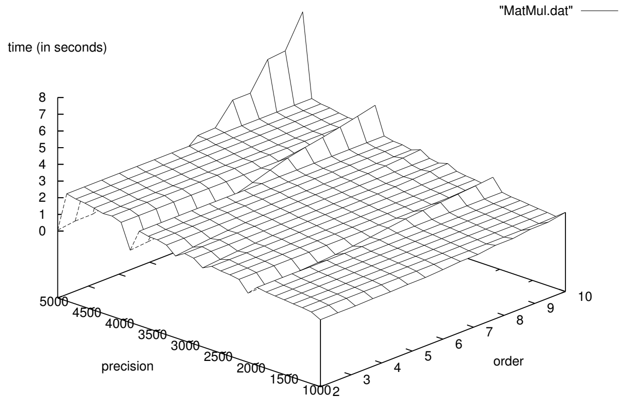

Figure 5: Timings of algorithm PolyMatMul.

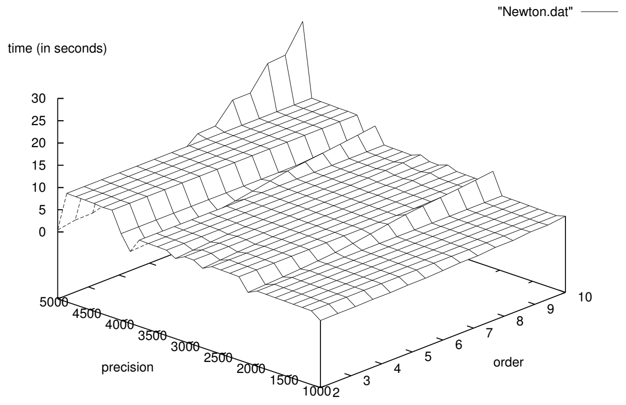

Figure 6: Timings of algorithm SolveDiffHomSys.

References

[1] W. Baur and V. Strassen. The complexity of partial derivatives. *Theoret. Comput. Sci.*, 22:317–330, 1983.

[2] D. J. Bernstein. Removing redundancy in high-precision Newton iteration, 2000. http://cr.yp.to/.

[3] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997. See also http://magma.maths.usyd.edu.au/.

[4] A. Bostan, P. Flajolet, B. Salvy, and É. Schost. Fast computation of special resultants. *Journal of Symbolic Computation*, 41(1):1–29, January 2006.

[5] A. Bostan and É. Schost. Polynomial evaluation and interpolation on special sets of points. *Journal of Complexity*, 21(4):420–446, August 2005.

[6] R. P. Brent. Multiple-precision zero-finding methods and the complexity of elementary function evaluation.

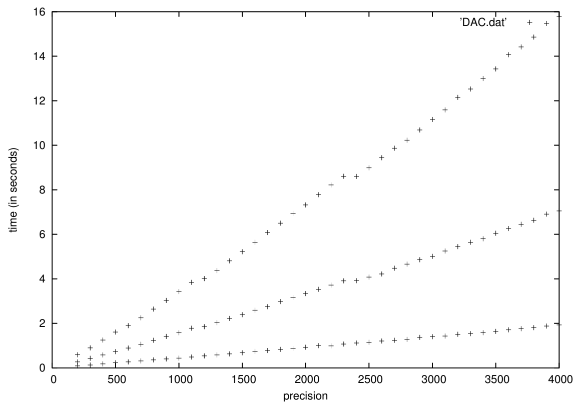

Figure 7: Timings of algorithm Solve for equations of orders 2, 4, and 8.

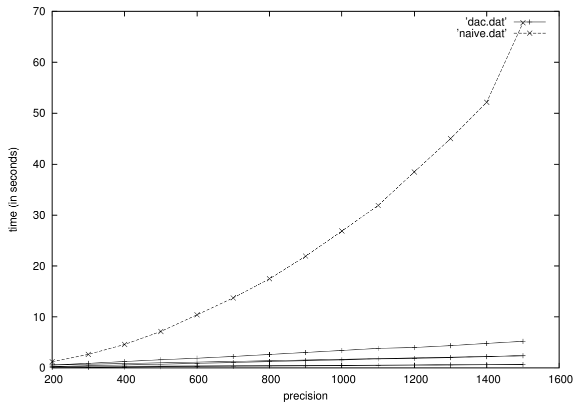

Figure 8: Same, compared to the naive algorithm for a second-order equation.

In *Analytic computational complexity*, pages 151–176. Academic Press, New York, 1976.

[7] R. P. Brent and H. T. Kung. Fast algorithms for composition and reversion of multivariate power series. In *Proc. Conf. Theoretical Computer Science, University of Waterloo, Canada, Aug. 1977*, pages 149–158, 1977.

[8] R. P. Brent and H. T. Kung. Fast algorithms for manipulating formal power series. *J. ACM*, 25(4):581–595, 1978.

[9] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren Math. Wiss.* Springer–Verlag, 1997.

[10] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Inform.*, 28(7):693–701, 1991.

[11] Fiduccia, C. M. An efficient formula for linear recurrences *SIAM J. Comput.*, 14(1):106–112, 1985.

[12] P. Flajolet, G. Labelle, L. Laforest, and B. Salvy. Hypergeometrics and the cost structure of quadtrees. *Random Structures & Algorithms*, 7(1):117–144, 1995.

[13] P. Flajolet, P. Zimmermann, and B. Van Cutsem. A Calculus for the Random Generation of Labelled Combinatorial Structures. *TCS*, 132(1-2):1–35, 1994.

[14] J. von zur Gathen and J. Gerhard. Fast algorithms for Taylor shifts and certain difference equations. In *ISSAC'97*, pages 40–47. ACM Press, 1997.

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

[16] K. O. Geddes. Convergence behavior of the Newton iteration for first order differential equations. In *EUROSAM'79*, pages 189–199. Springer-Verlag, 1979.

[17] G. Hanrot and P. Zimmermann. Newton iteration revisited, 2002. http://www.loria.fr/~zimmerma/.

[18] E. L. Ince. *Ordinary Differential Equations*. New York: Dover Publications, 1956.

[19] D. E. Knuth. The analysis of algorithms. In *Actes du Congrès International des Mathématiciens (Nice, 1970), Tome 3*, pages 269–274. Paris, 1971.

[20] H. T. Kung and J. F. Traub. All algebraic functions can be computed fast. *J. ACM*, 25(2):245–260, 1978.

[21] Gilbert Labelle. On combinatorial differential equations. *J. Math. Anal. Appl.*, 113(2):344–381, 1986.

[22] U. Luther and K. Rost. Matrix exponentials and inversion of confluent Vandermonde matrices. *Electron. Trans. Numer. Anal.*, 18:91–100, 2004.

[23] H. Mahmoud and B. Pittel. On the most probable shape of a search tree grown from a random permutation. *SIAM J. Alg. Discr. Meth.*, 5(1):69–81, 1984.

[24] R. T. Moenck and J. H. Carter. Approximate algorithms to derive exact solutions to systems of linear equations. In *EUROSAM'79*, pages 65–73. 1979.

[25] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Rev.*, 20(4):801–836, 1978.

[26] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Rev.*, 45(1):3–49, 2003.

[27] W. O. Pennell. A New Method for Determining a Series Solution of Linear Differential Equations with Constant or Variable Coefficients. *Amer. Math. Monthly*, 33(6):293–307, 1926.

[28] L. B. Rall. *Computational solution of nonlinear operator equations*. J. Wiley & Sons Inc., New York, 1969.

[29] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Inform.*, 1:139–144, 1971.

[30] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.

[31] G. Schulz. Iterative Berechnung der reziproken Matrix. *Z. Angew. Math. Mech.*, 13:57–59, 1933.

[32] A. Storjohann. Deterministic computation of the Frobenius form. In *FOCS'01*, pages 368–377.

[33] A. Storjohann. High-order lifting. In *ISSAC'02*, pages 246–254. ACM, 2002.

[34] V. Strassen. The computational complexity of continued fractions. *SIAM J. Comput.*, 12:1–27, 1983.

[35] J. van der Hoeven. Relax, but don't be too lazy. *J. Symbolic Comput.*, 34(6):479–542, 2002.

[36] A. G. Werschulz. Computational complexity of one-step methods for systems of differential equations. *Math. Comp.*, 34(149):155–174, 1980.