Newton's method and FFT trading*

Joris van der Hoeven

Dépt. de Mathématiques (Bât. 425)
CNRS, Université Paris-Sud
91405 Orsay Cedex
France
Email: joris@texmacs.org

December 9, 2008

Let $\mathcal{C}[[z]]$ be the ring of power series over an effective ring $\mathcal{C}$. In [BK78], it was first shown that differential equations over $\mathcal{C}[[z]]$ may be solved in an asymptotically efficient way using Newton's method. More precisely, if $\mathsf{M}(n)$ denotes the complexity in order two polynomials of degree $< n$ over $\mathcal{C}$, then the first n coefficients of the solution can be computed in time $O(\mathsf{M}(n))$. However, this complexity does not take into account the dependency of on the order r of the equation, which is exponential for the original method [vdH02] and linear for a recent improvement [BCO+06]. In this paper, we present a technique to get rid of this constant factor, by applying Newton's method up to an order like n/r and trading the remaining Newton steps against a lazy or relaxed algorithm in a suitable FFT model.

Keywords: power series, Newton's method, differential equation, FFT
A.M.S. subject classification: 68W25, 37M99, 90C53, 42-04, 68W30, 33F05

1. Introduction

Let $\mathcal{C}[[z]]$ be the ring of power series over an effective ring $\mathcal{C}$. It will be convenient to assume that $\mathcal{C} \supseteq \mathbb{Q}$. In this paper, we are concerned with the efficient resolution of implicit equations over $\mathcal{C}[[z]]$. Such an equation may be presented in fixed-point form as

$$F = \Phi(F), \tag{1}$$

where F is an indeterminate vector in $\mathcal{C}[[z]]^r$ with $r \in \mathbb{N}$. The operator Φ is constructed using classical operations like addition, multiplication, integration or postcomposition with a series $g \in \mathcal{C}[[z]]$ with $g_0 = 0$. In addition, we require that the n-th coefficient of $\Phi(F)_n$ depends only on coefficients F_i with $i < n$, which allows for the recursive determination of all coefficients.

In particular, linear and algebraic differential equations may be put into the form (1). Indeed, given a linear differential system

$$\begin{aligned} F' &= A\,F \\ F_0 &= I \in \mathcal{C}^r \end{aligned} \tag{2}$$

where A is an $r \times r$ matrix with coefficients in $\mathcal{C}[[z]]$, we may take $\Phi(F) = I + \int A\,F$. Similarly, if P is a tuple of r polynomials in $\mathcal{C}[[z]][F] = \mathcal{C}[[z]][F_1, ..., F_r]$, then the initial value problem

$$\begin{aligned} F' &= P(F) \\ F_0 &= I \in \mathcal{C}^r \end{aligned} \tag{3}$$

*. This work was partially supported by the ANR Gecko project.

may be put in the form (1) by taking $\Phi(F) = I + \int P(F)$.

For our complexity results, and unless stated otherwise, we will always assume that polynomials are multiplied using FFT multiplication. If $\mathcal{C}$ contains all 2^p-th roots of unity with $p \in \mathbb{N}$, then it is classical that two polynomials of degrees $< n$ can be multiplied using $\mathsf{M}(n) = O(n \log n)$ operations over $\mathcal{C}$ [CT65]. In general, such roots of unity can be added artificially [SS71, CK91, vdH02] and the complexity becomes $\mathsf{M}(n) = O(n \log n \log\log n)$. We will respectively refer to these two cases as the *standard* and the *synthetic* FFT models. In both models, the cost of one FFT transform at order $2\,n$ is $\sim \mathsf{M}(n)/3$, where we assume that the FFT transform has been replaced by a TFT transform [vdH04, vdH05] in the case when n is not a power of two.

Let $\mathcal{M}_r(\mathcal{C})$ be the set of $r \times r$ matrices over $\mathcal{C}$. It is classical that two matrices in $\mathcal{M}_r(\mathcal{C})$ can be multiplied in time $O(r^\omega)$ with $\omega < 2.376$ [Str69, Pan84, WC87]. We will denote by $\mathsf{MM}(n,\, r)$ the cost of multiplying two polynomials of degrees $< n$ with coefficients in $\mathcal{M}_r(\mathcal{C})$. By FFT-ing matrices of polynomials, one has $\mathsf{MM}(n, r) = O(n\, r^\omega + \mathsf{M}(n)\, r^2)$ and $\mathsf{MM}(n, r) \sim \mathsf{M}(n)\, r^2$ if $r = O(\log n)$.

In [BK78], it was shown that Newton's method may be applied in the power series context for the computation of the first n coefficients of the solution F to (2) or (3) in time $O(\mathsf{M}(n))$. However, this complexity does not take into account the dependence on the order r, which was shown to be exponential in [vdH02]. Recently [BCO+06], this dependence in r has been reduced to a linear factor. In particular, the first n coefficients of the solution F to (3) can be computed in time $O(\mathsf{MM}(n,\, r))$. In fact, the resolution of (2) in the case when F and I are replaced by matrices in $\mathcal{M}_r(\mathcal{C}[[z]])$ resp. $\mathcal{M}_r(\mathcal{C})$ can also be done in time $O(\mathsf{MM}(n, r))$. Taking $I = \mathrm{Id}_r$, this corresponds to the computation of a fundamental system of solutions.

However, the new complexity is not optimal in the case when the matrix A is sparse. This occurs in particular when a linear differential equation

$$f^{(r)} = L_{r-1}\, f^{(r-1)} + \cdots + L_0\, f. \tag{4}$$

is rewritten in matrix form. In this case, the method from [BCO+06] for the asymptotically efficient resolution of the vector version of (4) as a function of n gives rise to an overhead of $O(r)$, due to the fact that we need to compute a full basis of solutions in order to apply the Newton iteration.

In [vdH97, vdH02], the alternative approach of relaxed evaluation was presented in order to solve equations of the form (1). More precisely, let $\mathsf{R}(n)$ be the cost to *gradually* compute n terms of the product $h = f\, g$ of two series $f,\, g \in \mathcal{C}[[z]]$. This means that the terms of f and g are given one by one, and that we require h_i to be returned as soon as $f_0, ..., f_i$ and $g_0, ..., g_i$ are known ($i = 0, ..., n-1$). In [vdH97, vdH02], we proved that $\mathsf{R}(n) = O(\mathsf{M}(n) \log n)$. In the standard FFT model, this bound was further reduced in [vdH03a] to $\mathsf{R}(n) = O(\mathsf{M}(n)\, \mathrm{e}^{2\sqrt{\log\log n}})$. We also notice that the additional $O(\log n)$ or $O(\mathrm{e}^{2\sqrt{\log\log n}})$ overhead only occurs in FFT models: when using Karatsuba or Toom-Cook multiplication [KO63, Too63, Coo66], one has $\mathsf{R}(n) \sim \mathsf{M}(n)$. One particularly nice feature of relaxed evaluation is that the mere relaxed evaluation of $\Phi(F)$ provides us with the solution to (1). Therefore, the complexity of the resolution of systems like (2) or (3) only depends on the sparse size of Φ as an expression, without any additional overhead in r.

Let $\mathsf{L}(n, r)$ denote the complexity of computing the first n coefficients of the solution to (4). By what precedes, we have both $\mathsf{L}(n, r) = O(\mathsf{M}(n)\, r^2)$ and $\mathsf{L}(n, r) = O(\mathsf{M}(n)\, r \log n)$. A natural question is whether we may further reduce this bound to $\mathsf{L}(n, r) = O(\mathsf{M}(n)\, r)$ or even $\mathsf{L}(n, r) \sim \mathsf{M}(n)\, r$. This would be optimal in the sense that the cost of resolution would be the same as the cost of the verification that the result is correct. A similar problem may be stated for the resolution of systems (2) or (3).

In this paper, we present several results in this direction. The idea is as follows. Given $n \in \mathbb{N}$, we first choose a suitable $m < n$, typically of the order $m = n/r^{1+\epsilon}$. Then we use Newton iterations for determining successive blocks of m coefficients of F in terms of the previous coefficients of F and $A\,F$. The product $A\,F$ is computed using a lazy or relaxed method, but on FFT-ed blocks of coefficients. Roughly speaking, we apply Newton's method up to an order m, where the $O(r)$ overhead of the method is not yet perceptible. The remaining Newton steps are then traded against an asymptotically less efficient lazy or relaxed method without the $O(r)$ overhead, but which is actually more efficient for small m when working on FFT-ed blocks of coefficients.

It is well known that FFT multiplication allows for tricks of the above kind, in the case when a given argument is used in several multiplications. In the case of FFT trading, we artificially replace an asymptotically fast method by a slower method on FFT-ed blocks, so as to use this property. We refer to [Ber] (see also remark 6 below) for a variant and further applications of this technique (called FFT caching by the author). The central idea behind [vdH03a] is also similar. In section 4, we outline yet another application to the truncated multiplication of dense polynomials.

The efficient resolution of differential equations in power series admits several interesting applications, which are discussed in more detail in [vdH02]. In particular, certified integration of dynamical systems at high precision is a topic which currently interests us [vdH06a]. More generally, the efficient computation of Taylor models [Moo66, Loh88, MB96, Loh01, MB04] is a potential application.

2. Linear differential equations

Given a power series $f \in \mathcal{C}[[z]]$ and similarly for vectors or matrices of power series (or power series of vectors or matrices), we will use the following notations:

$$\begin{aligned} f_{;i} &= f_0 + \cdots + f_{i-1}\, z^{i-1} \\ f_{i;j} &= f_i + \cdots + f_{j-1}\, z^{j-1}, \end{aligned}$$

where $i, j \in \mathbb{N}$ with $i \leqslant j$.

In order to simplify our exposition, it is convenient to rewrite all differential equations in terms of the operator $\delta = z\partial/\partial z$. Given a matrix $A \in \mathcal{M}_r(\mathcal{C}[[z]])$ with $A_0 = 0$, the equation

$$\delta M = A\, M \tag{5}$$

admits unique solution $M \in \mathcal{M}_r(\mathcal{C}[[z]])$ with $M_0 = \mathrm{Id}_r$. The main idea of [BCO+06] is to provide a Newton iteration for the computation of M. More precisely, with our notations, assume that $M_{;n}$ and $M_{;n}^{-1} = (M^{-1})_{;n}$ are known. Then we have

$$M_{;2n} := [M_{;n} - (M_{;n}\, \delta^{-1}\, M_{;n}^{-1})\, (\delta M_{;n} - A_{;2n}\, M_{;n})]_{;2n}. \tag{6}$$

Indeed, setting

$$\begin{aligned} \mathrm{E} &= A_{;2n}\, M_{;n} - \delta M_{;n} = O(z^n) \\ \Delta &= (M_{;n}\, \delta^{-1}\, M_{;n}^{-1})\, \mathrm{E} = O(z^n), \end{aligned}$$

we have

$$\begin{aligned} (\delta - A)\, \Delta &= (\delta M_{;n})\, (M_{;n})^{-1}\, \Delta + (1 + O(z^n))\, \mathrm{E} - A\, \Delta \\ &= (A\, M_{;n} + O(z^n))\, (M_{;n}^{-1} + O(z^n))\, \Delta + (1 + O(z^n))\, \mathrm{E} - A\, \Delta \\ &= \mathrm{E} + O(z^{2n}), \end{aligned}$$

so that $(\delta - A)(M_{;n} + \Delta) = O(z^{2n})$ and $\Delta = M_{n;2n} + O(z^{2n})$. Computing $M_{;n}$ and $M_{;n}^{-1}$ together using (6) and the usual Newton iteration [Sch33, MC79]

$$M_{;2n}^{-1} = [M_{;n}^{-1} + M_{;n}^{-1}(1 - M_{;n} M_{;n}^{-1})]_{;2n} \tag{7}$$

for inverses yields an algorithm of time complexity $O(\mathsf{MM}(n, r))$. The quantities $\mathrm{E}_{n;2n}$ and $M_{n;2n} = \Delta_{n;2n}$ may be computed efficiently using the middle product algorithm [HQZ04].

Instead of doubling the precision at each step, we may also increment the number of known terms with a fixed number of terms m. More precisely, given $n, m > 0$, we have

$$M_{;n+m} := [M_{;n} - (M_{;m}\, \delta^{-1}\, M_{;m}^{-1})(\delta M_{;n} - A_{;n+m}\, M_{;n})]_{;n+m}. \tag{8}$$

This relation is proved in a similar way as (6). The same recurrence may also be applied for computing blocks of coefficients of the unique solution $F \in \mathcal{C}[[z]]^r$ to the vector linear differential equation

$$\delta F = A\, F \tag{9}$$

with initial condition $F_0 = I \in \mathcal{C}^r$:

$$F_{;n+m} := [F_{;n} - (M_{;m}\, \delta^{-1}\, M_{;m}^{-1})(\delta F_{;n} - A_{;n+m}\, F_{;n})]_{;n+m}. \tag{10}$$

Both the right-hand sides of the equations (8) and (10) may be computed efficiently using the middle product.

Assume now that we want to compute $F_{;n}$ and take $m < n$. For simplicity, we will assume that $n = k\, m$ with $k \in \mathbb{N}$ and that $\mathcal{C}$ contains all 2^p-th roots of unity for $p \in \mathbb{N}$. We start by decomposing $F_{;n}$ using

$$\begin{aligned} F_{;n} &= F_{[0]} + \cdots + F_{[k-1]} \\ F_{[i]} &= F_{im;(i+1)m} \end{aligned} \tag{11}$$

and similarly for A. Setting $P = A\, F$, we have

$$P_{[i]} = (A_{[i-1]} + A_{[i]}) \ltimes F_{[0]} + \cdots + (A_{[0]} + A_{[1]}) \ltimes F_{[i-1]} + (A_{[-1]} + A_{[0]}) \ltimes F_{[i]},$$

where $\ltimes$ stands for the middle product (see figure 1). Instead of computing $P_{[i]}$ directly using this formula, we will systematically work with the FFT transforms $F_{[j]}^*$ of $F_{[j]}$ at order $2\, m$ and similarly for $A_{[j-1]} + A_{[j]}$ and $P_{[j]}$, so that

$$P_{[i]}^* = (A_{[i-1]} + A_{[i]})^* \ltimes F_{[0]}^* + \cdots + (A_{[0]} + A_{[1]})^* \ltimes F_{[i-1]}^* + (A_{[-1]} + A_{[0]})^* \ltimes F_{[i]}^*.$$

Recall that we may resort to TFT transforms [vdH04, vdH05] if m is not a power of two. Now assume that $M_{[0]}$, $M_{[0]}^{-1}$ and

$$P_{[i]}^{\mathrm{pre}} = P_{[i]} - (A_{[-1]} + A_{[0]}) \ltimes F_{[i]} = ((A_{[0]} + \cdots + A_{[i]})\,(F_{[0]} + \cdots + F_{[i-1]}))_{[i]}$$

are known. Then the relation (10) yields

$$F_{[i]} = \left[(M_{[0]}\, \delta^{-1}\, M_{[0]}^{-1})\, P_{[i]}^{\mathrm{pre}} \right]_{[i]}. \tag{12}$$

In practice, we compute $F_{[i]}$ via $X = (M_{[0]}^{-1}\, P_{[i]}^{\mathrm{pre}})_{[i]}$, $Y = \delta X$ and $F_{[i]} = (M_{[0]}\, Y)_{[i]}$, using FFT multiplication. Here we notice that the FFT transforms of $M_{[0]}$ and $M_{[0]}^{-1}$ only need to be computed once.

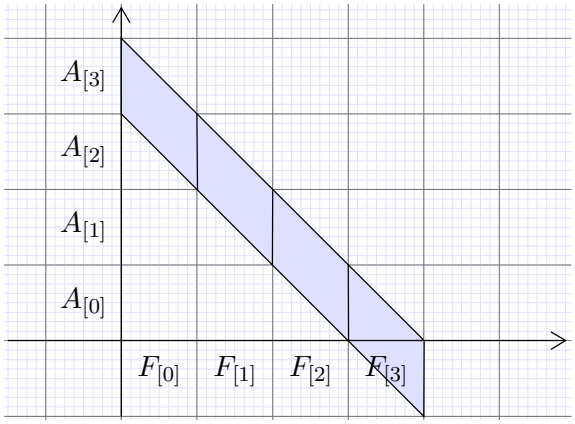

Figure 1. Illustration of the computation of $P_{[3]}$ using middle products.

Putting together what has been said and assuming that $M_{[0]}$, $M_{[0]}^{-1}$ and $F_{[0]}$ are known, we have the following algorithm for the successive computation of $F_{[1]}, ..., F_{[k-1]}$:

for $i = 0, ..., k-1$ **do** $(A_{[i-1]} + A_{[i]})^* := \mathrm{FFT}(A_{[i-1]} + A_{[i]})$
for $i = 1, ..., k-1$ **do**
  $F_{[i-1]}^* := \mathrm{FFT}(F_{[i-1]})$
  $(P_{[i]}^{\mathrm{pre}})^* := (A_{[i-1]} + A_{[i]})^* \ltimes F_{[0]}^* + \cdots + (A_{[0]} + A_{[1]})^* \ltimes F_{[i-1]}^*$
  $P_{[i]}^{\mathrm{pre}} := \mathrm{FFT}^{-1}((P_{[i]}^{\mathrm{pre}})^*)$
  $F_{[i]} := \left[(M_{[0]}\, \delta^{-1}\, M_{[0]}^{-1})\, P_{[i]}^{\mathrm{pre}} \right]_{[i]}$

In this algorithm, the product $P = A\,F$ is evaluated in a lazy manner. Of course, using a straightforward adaptation, one may also use a relaxed algorithm. In particular, the algorithm from [vdH03b] is particularly well-suited in combination with middle products. In the standard FFT model, [vdH03a] is even faster.

THEOREM 1. *Consider the differential equation* (9)*, where A has s non-zero entries. Assume the standard FFT model. Then there exists an algorithm which computes the truncated solution $F_{;n}$ to* (9) *at order n in time $\sim \mathsf{M}(n)\,(s + 4/3\,r)$, provided that $r = o(\log n)$. In particular, $\mathsf{L}(n, r) \sim 7/3\,\mathsf{M}(n)\,r$.*

PROOF. In our algorithm, we take $k = \lfloor r^{1+\epsilon_n} \rfloor$, where ϵ_n grows very slowly to infinity (such as $\epsilon_n = \log\log\log n$), so that $\log m \sim \log n$. Let us carefully examine the cost of our algorithm for this choice of k:

1. The computation of $F_{[0]}$, $M_{[0]}$ and $M_{[0]}^{-1}$ requires a time $O(\mathsf{M}(m)\,r^2) = o(\mathsf{M}(n)\,r)$.
2. The computation of the FFT-transforms $A_{[i]}^* := \mathrm{FFT}(A_{[i]})$, $F_{[i]}^* := \mathrm{FFT}(F_{[i]})$ and the inverse transforms $P_{[i]} := \mathrm{FFT}^{-1}(P_{[i]}^*)$ has the same complexity $\sim \mathsf{M}(m)\,k\,s \sim \mathsf{M}(n)\,s$ as the computation of the final matrix product $A\,F$ at order n.
3. The computation of $O(k^2)$ middle products $(A_{[i-1]} + A_{[i]})^* \ltimes F_{[j]}^*$ in the FFT-model requires a time $O(k^2\,m\,s) = O(k\,n\,s)$. Using a relaxed multiplication algorithm, the cost further reduces to $O(\mathsf{R}(k)\,m\,s) = O(n\,\epsilon_n\,s\log^2 r) = o(\mathsf{M}(n)\,s)$.
4. The computation of the $F_{[i]}$ using the Newton iteration (12) involves
   a. Matrix FFT-transforms of $M_{[0]}$ and $M_{[0]}^{-1}$, of cost $O(\mathsf{M}(m)\,r^2) = o(\mathsf{M}(n)\,r)$.

b. $4\,(k-1)$ vectorial FFT-transforms of cost $\sim 4/3\,\mathsf{M}(m)\,k\,r \sim 4/3\,\mathsf{M}(n)\,r$.

c. $O(k)$ matrix vector multiplications in the FFT-model, of cost $O(k\,m\,r^2) = O(n\,r^2) = o(\mathsf{M}(n)\,r)$.

Adding up the different contributions completes the proof. Notice that the computation of $\lceil n/k \rceil\, k - n < k$ more terms has negligible cost, in the case when n is not a multiple of k. □

REMARK 2. In the synthetic FFT model, the recursive FFT-transforms $A^*_{[i]}$, $F^*_{[i]}$ and $M^*_{[i]}$ require an additional $O(\log\, n)$ space overhead, when using the polynomial adaptation [CK91, vdH02] of Schönhage-Strassen multiplication. Consequently, the cost in point 3 now becomes

$$O(\mathsf{R}(k)\,m\,s\log n) = O(n\,\epsilon_n\,s\log^2 r\log\log r\log n) = O\left(\mathsf{M}(n)\,s\,\frac{\epsilon_n \log^2 r \log\log r}{\log\log n}\right).$$

Provided that $\log^2 r \log\log r = o(\log\log n)$, we obtain the same complexity as in the theorem, by choosing ϵ_n sufficiently slow.

REMARK 3. With minor changes, the algorithm can be adapted in order to compute the unique solution of the matrix $\delta M = A\,M$ with $M_0 = \mathrm{Id}_r$ (which corresponds to a fundamental system of solutions to (9)). In that case, the complexity becomes $\sim \mathsf{M}(n)\,(r^2 + 4/3\,r^2) \sim 7/3\,\mathsf{M}(n)\,r^2$.

REMARK 4. It is instructive to compare our complexity bounds with the old complexity bounds if we do not use FFT trading. In that case, let $\mathsf{T}(n, r)$ denote the complexity of computing both $M_{;n}$ and $M_{;n}^{-1}$. One has

$$\mathsf{T}(2\,n, r) = \mathsf{T}(n, r) + 5\,\mathsf{M}(n)\,r^2 + O(n\,r^\omega),$$

since the product $A_{;n}\,F_{;n}$ and the formulas (6) and (7) give rise to $1 + 2 + 2 = 5$ matrix multiplications. This yields $\mathsf{T}(n,\, r) \sim 5\ \mathsf{M}(n)\ r^2$ from which we may subtract $\mathsf{M}(n)\ r^2$ in the case when M_n^{-1} is not needed. The bound may be further improved to $\mathsf{T}(n, r) \sim 9/2\,\mathsf{M}(n)\,r^2$ using [Sch00]. Similarly, the old bound for the resolution of (9) is $\sim \mathsf{M}(n)\,(17/6\,r^2 + s/2 + 2/3\,r)$, or $\sim \mathsf{M}(n)\,(31/12\,r^2 + s/2 + 2/3\,r)$ when using [Sch00].

REMARK 5. In point 3 of the proof, the computation of the middle products using a naive lazy algorithm requires a time $O(k\ n\ s) = O(r^{\epsilon_n}\ \mathsf{M}(n)\ s)$. In practice, we may actually take $\epsilon_n = 0$, in which case there is no particular penalty when using a naive algorithm instead of a relaxed one. In fact, for larger values of r, it is rather the hypothesis $r = O(\log n)$ which is easily violated. In that case, one may take $1 < k < r$ instead of $k \geqslant r$ and still gain a constant factor between 1 and r.

REMARK 6. The results of this section apply in particular to the computation of the exponential $f = \mathrm{e}^g$ of a power series g. In that case, theorem 1 provides a way to compute $f_{;n}$ in time $\sim 7/3\,\mathsf{M}(n)$, which yields an improvement over [Ber, HZ]. Notice that FFT trading is a variant of Newton caching in [Ber], but not exactly the same: in our case, we use an "order k" Newton iteration, whereas Bernstein uses classical Newton iterations on block-decomposed series. In the case of power series division f/g at order n or division with remainder of a polynomial of degree $< 2\,n$ by a polynomial of degree $< n$, the present technique also allows for improvements w.r.t. [Ber, HZ]. In both cases, the new complexity is $\sim 5/3\,\mathsf{M}(n)$. In addition, we notice that the technique of FFT trading allows for a "smooth junction" between the Karatsuba (or Toom-Cook) model and the FFT model.

3. ALGEBRAIC DIFFERENTIAL EQUATIONS

Assuming that one is able to solve the linearized version of an implicit equation (1), it is classical that Newton's method can again be used to solve the equation itself [BK78, vdH02, BCO+06]. Before we show how to do this for algebraic systems of differential equations, let us first give a few definitions for polynomial expressions.

Given a vector $F = \mathcal{C}[[z]]^r$ of series variables, we will represent polynomials in $\mathcal{C}[[z]][F] \cong \mathcal{C}[F][[z]] = \mathcal{C}[[z]][F_1, ..., F_r]$ by dags (directed acyclic graphs), whose leaves are either series in $\mathcal{C}[[z]]$ or variables F_i, and whose inner nodes are additions, subtractions or multiplications. An example of such a dag is shown in figure 2. We will denote by s_1 and s_2 the number of nodes which occur as an operand resp. result of a multiplication. We call $s = (s_1 + s_2)/3$ the *multiplicative size* s of the dag and the total number t of nodes the *total size* of the dag. Using the FFT, one may compute $P(F)_{;n}$ in terms of $F_{;n}$ in time $\sim \mathsf{M}(n)\, s + n\, a$.

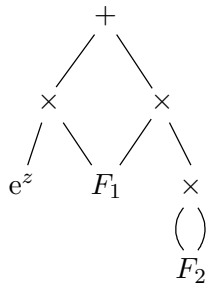

Figure 2. Example of a polynomial expression in $\mathcal{C}[[z]][F_1, F_2]$, represented by a dag. In this particular example, the multiplicative size of the polynomial is $s = 7/3$ (since $s_1 = 4$ and $s_2 = 3$)and its total size 7. Notice in particular that squares only count for $2/3$ in the multiplicative size.

Now assume that we are given an r-dimensional system

$$\delta F = P(F), \tag{13}$$

with initial condition $F_0 = I \in \mathcal{C}^r$, and where $P(F)$ is a tuple of r polynomials in $z\, \mathcal{C}[[z]][F_1, ..., F_r]^r \cong \mathcal{C}[F_1, ..., F_r][[z]]^r$. Given the unique solution F to this initial value problem, consider the Jacobian matrix

$$J = \frac{\partial P}{\partial F}(F) = \begin{pmatrix} \frac{\partial P_1}{\partial F_1} & \cdots & \frac{\partial P_1}{\partial F_r} \\ \vdots & & \vdots \\ \frac{\partial P_r}{\partial F_1} & \cdots & \frac{\partial P_r}{\partial F_r} \end{pmatrix}(F).$$

Assuming that $F_{;m}$ is known, we may compute $J_{;m}$ in time $O(\mathsf{M}(m))$ using automatic differentiation. As usual, this complexity hides an (r, s, t)-dependent overhead, which is bounded by $O(r\,(s + t\, n/\mathsf{M}(n)))$. For $n \geqslant m$, we have

$$\begin{aligned} P(F_{;n} + F_{n;n+m}) &= P(F_{;n}) + J_{;m}\, F_{n;n+m} + O(z^{n+m}) \\ \delta F_{;n+m} &= P(F_{;n})_{;n+m} + J_{;m}\, F_{n;n+m}, \end{aligned}$$

so that

$$F_{n;n+m} = [(\delta - J_{;m})^{-1}\, (P(F_{;n})_{n;n+m})]_{n;n+m}. \tag{14}$$

Let us again adopt the notation (11). Having determined $F_{[i-1]}$ and $Q(F)_{[i-1]}$ for each subexpression $Q(F)$ of $P(F)$ up to a given order i, the computation of $F_{[i]}$ and all $Q(F)_{[i]}$ can be done in three steps:

1. The computation of all $Q(\tilde{F})_{[i]}$, using lazy or relaxed evaluation, where $\tilde{F} = F_{[0]} + \cdots + F_{[i-1]}$.

2. The determination of $F_{[i]} = \Big[(\delta - J_{[0]})^{-1} P(\tilde{F})_{[i]} \Big]_{[i]}$ using (14).

3. The computation of all $Q(F)_{[i]}$.

We notice that $Q(F)_{[i]}$ and $Q(\tilde{F})_{[i]}$ are almost identical, since

$$Q(F)_{[i]} - Q(\tilde{F})_{[i]} = \left[\frac{\partial Q}{\partial F} (F)\, F_{[i]} \right]_{[i]}.$$

If $Q = UV$ is a product, then $Q(\tilde{F})_{[i]}$ can be determined from $U(\tilde{F})_{[i]}$ and $V(\tilde{F})_{[i]}$ using a suitable middle product with "omitted extremes" (see figure 3). Step 3 consists of an adjustment, which puts these extremes back in the sum. Of course, the computations of products $Q = UV$ can be done in a relaxed fashion.

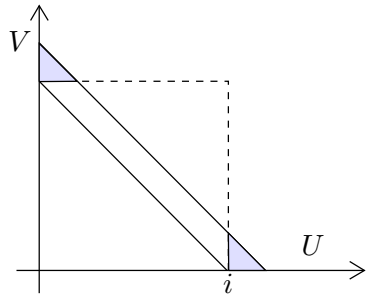

Figure 3. Illustration of the product $Q(F)_{[i]} = (UV)(F)_{[i]}$. The part inside the square corresponds to $Q(\tilde{F})_{[i]}$ and the two small triangles to the difference $Q(F)_{[i]} - Q(\tilde{F})_{[i]}$.

THEOREM 7. *Consider an r-dimensional system* (13)*, where P is a polynomial, given by a dag of multiplicative size s and total size t. Assume the standard FFT model. Then there exist an algorithm which computes $F_{;n}$ in time $\sim \mathsf{M}(n)\,(s + 4/3\,r) + O(n\,t)$, provided that $r = o(\log n)$.*

PROOF. When working systematically with the FFT-ed values of the $Q(F)_{[i]}$, steps 1 and 3 give rise to a cost $\mathsf{M}(n)\,s$ for the FFT transforms and a cost $O(n\,t)$ for the scalar multiplications and additions. In a similar way as in the proof of theorem 1, the computation of the $F_{[i]}$ gives rise to a cost $\sim 4/3\,\mathsf{M}(n)\,r$. Again, the cost of the computation of the initial $F_{[0]}$ and $J_{[0]}$ is negligible. □

REMARK 8. The bound becomes $\sim \mathsf{M}(n)\,(s + 4/3\,r) + O(t\,n \log n)$ in the synthetic FFT model and under the assumption $\log^2 r \log\log r = o(\log\log n)$. This bound is derived in a similar way as in remark 2.

REMARK 9. A detailed comparison between the new and old [BCO+06] complexities is difficult, because the size parameter s is not entirely adequate for expressing the old complexity. In the worst case, the old complexity is $\sim \mathsf{M}(n)\,(s\,r + 8/3\,r^2 + 4/3\,r) + O(t\,r\,n)$, which further improves to $\sim \mathsf{M}(n)\,(s\,r + 13/6\,r^2 + 4/3\,r) + O(t\,r\,n)$ using [Sch00]. However, the factor $s\,r$ is quite pessimistic, since it occurs only when most of the subexpressions $Q(F)$ of $P(F)$ depend on most of the variables $F_1, \ldots, F_r$. If the multiplicative subexpressions $Q(F)$ depend on an average number of μ variables F_j, then the process of automatic differentiation can be optimized so as to replace s by μ in the bound (roughly speaking).

4. TRUNCATED MULTIPLICATION

It is well-known that discrete FFT transforms are most efficient on blocks of size 2^p with $p \in \mathbb{N}$. In particular, without taking particular care, one may lose a factor 2 when computing the product of two polynomials P and Q of degree $n-1$ with $n \notin 2^{\mathbb{N}}$. One strategy to remove this problem is to use the TFT (truncated Fourier transform) as detailed in [vdH04] with some corrections and further improvements in [vdH05].

An alternative approach is to cut P and Q into $k = \lceil n/m \rceil$ parts of size $m = 2^p$, where k grows slowly to infinity with n. Let us denote

$$\begin{aligned} P_{i;j} &= P_i + \cdots + P_{j-1} z^{j-i-1} \\ P_{[i]} &= P_{im;(i+1)m} \end{aligned}$$

Attention to the minor change with respect to the notations from section 2. Now we multiply P and Q by

1. FFT-ing the blocks $P_{[i]}$ and $Q_{[i]}$ at size $2\,m$.
2. Naively multiplying the resulting FFT-ed polynomials in $U = z^m$:
$$\begin{aligned} P^* &= P^*_{[0]} + \cdots + P^*_{[k-1]} U^{k-1} \\ Q^* &= Q^*_{[0]} + \cdots + Q^*_{[k-1]} U^{k-1} \end{aligned}$$
3. Transforming the result back.

This approach has a cost $\sim C\,k\,m \log m + 2\,k^2\,m \sim C\,n \log n + 2\,k\,n$ which behaves more "smoothly" as a function of n.

In this particular case, it turns out that the TFT transform is always better, because the additional linear factor $2\,k\,n$ is reduced to $2\,n$. However, in the multivariate setting, the TFT also has its drawbacks. More precisely, consider two multivariate polynomials $P, Q \in \mathcal{C}[z_1, ..., z_d]$ whose supports have a "dense flavour". Typically, we may assume the supports to be convex subsets of $\mathbb{N}^d$. In addition one may consider truncated products, where we are only interested in certain monomials of the product. In order to apply the TFT, one typically has to require in addition that the supports of P and Q are initial segments of $\mathbb{N}^d$. Even then, the overhead for certain types of supports may increase if d gets large.

One particularly interesting case for complexity studies is the computation of the truncated product of two dense polynomials P and Q with total degree $< n$. This is typically encountered in the integration of dynamical systems using Taylor models. Although the TFT is a powerful tool for small dimensions ($d \leqslant 4$), FFT trading might be an interesting complement for moderate dimensions ($5 \leqslant d \leqslant 8$). For really huge dimensions, one may use [LS03] or [vdH02, Section 6.3.5]. The idea is again to cut P in blocks

$$\begin{aligned} P &= \sum_{i=(i_1,...,i_d)} P_{[i]} U^i \qquad (U^i = U_1^{i_1} \cdots U_d^{i_d}) \\ P_{[i]} &= \sum_{j<(m,...,m)} P_{mi+j} z^j \qquad (z^j = z_1^{j_1} \cdots z_d^{j_d}) \end{aligned}$$

where $k = \lceil n/m \rceil$ is small (and m preferably a power of two). Each block is then transformed using a suitable TFT transform (notice that the supports of the blocks are still initial segments when restricted to the block). We next compute the truncated product of the TFT-ed polynomials $\sum P^*_{[i]} U^i$ and $\sum Q^*_{[i]} U^i$ in a naive way and TFT back. The naive multiplication step involves

$$N_{k,d} = \binom{d+k-1}{k-1}\binom{d}{0} + \binom{d+k-2}{k-2}\binom{d+1}{1} + \cdots + \binom{d}{0}\binom{d+k-1}{k-1}$$

multiplications of TFT-ed blocks. We may therefore hope for some gain whenever $N_{k,d}\ (n/k)^d$ is small with respect to $\mathsf{M}(n^d) \sim d\ \mathsf{M}(n)$. We always gain for $k = 2$ and usually also for $k = 3$, in which case $N_{k,d} \sim 2\ d^2$. Even for $k = 4$, when $N_{k,d} \sim 4/3\ d^3$, it is quite possible that one may gain in practice.

The main advantage of the above method over other techniques, such as the TFT, is that the shape of the support is preserved during the reduction $\sum P_i\, z^i \to \sum P_{[i]}\, U^i$ (as well as for the "destination support"). However, the TFT also allows for some additional tricks [vdH05, Section 9] and it is not yet clear to us which approach is best in practice. Of course, the above technique becomes even more useful in the case of more general truncated multiplications for dense supports with shapes which do not allow for TFT multiplication.

For small values of n, we notice that the pair/odd version of Karatsuba multiplication presents the same advantage of geometry preservation (see [HZ02] for the one-dimensional case). In fact, fast multiplication using FFT trading is quite analogous to this method, which generalizes for Toom-Cook multiplication. In the context of numerical computations, the property of geometry preservation is reflected by increased numerical stability.

To finish, we would like to draw the attention of the reader to another advantage of FFT trading: for really huge values of n, it allows for the reduction of the memory consumption. For instance, assume that we want to multiply two truncated power series P and Q at order n. With the above notations, one may first compute $P^*_{[0]}, \ldots, P^*_{[k-1]}$. For $i = 0, \ldots, k-1$, we next compute $Q^*_{[i]}$, $R^*_{[i]} = P^*_{[0]}\, Q^*_{[i]} + \cdots + P^*_{[k-1]}\, Q^*_{[i]}$ and $\mathrm{FFT}^{-1}(R^*_{[i]})$. The idea is now that $P^*_{[0]}$, ..., $P^*_{[i-1]}$, $R^*_{[0]}$, ..., $R^*_{[i-1]}$ are no longer used at stage i, so we may remove them from memory.

5. CONCLUSION

We have summarized the main results of this paper in tables 1 and 2. We recall that $\mathsf{R}(n) = O(\mathsf{M}(n)\, \mathrm{e}^{2\sqrt{\log\log n}})$ in the standard FFT model and $\mathsf{R}(n) = O(\mathsf{M}(n) \log n)$ otherwise. In practice, we expect that the factor $O(\mathrm{e}^{2\sqrt{\log\log n}})$ behaves very much like a constant, which equals 1 at the point where we enter the FFT model. Consequently, we expect the new algorithms to become only interesting for quite large values of n. We plan to come back to this issue as soon as implementations of all algorithms will be available in the MMXLIB library [vdH06b]. On the other hand, Newton iterations are better suited to parallel computing than relaxed evaluation. An interesting remaining problem is to reduce the cost of computing a fundamental system of solutions to (4). This would be possible if one can speed up the joint computation of the FFT transforms of $f, \delta f, \ldots, \delta^{(r-1)} f$.

Resolution of an r-dimensional system of linear differential equations		
Algorithm	Fundamental system	One solution
Relaxed	$\sim \mathsf{R}(n)\, r^2$	$\sim \mathsf{R}(n)\, s$
Newton	$\sim 4\, \mathsf{M}(n)\, r^2$	$\sim \mathsf{M}(n)\, (17/6\, r^2 + s/2 + 2/3\, r)$
New	$\sim 7/3\, \mathsf{M}(n)\, r^2$	$\sim \mathsf{M}(n)\, (s + 4/3\, r)$

Table 1. Complexities for the resolution of an r-dimensional system $\delta F = A\, F$ of linear differential equations up to n terms. We either compute a fundamental system of solutions or a single solution with a prescribed initial condition. The parameter s stands for the number of non-zero coefficients of the matrix A (we always have $s \leqslant r^2$). We assume that $r = o(\log n)$ in the standard FFT model and $\log^2 r \log\log r = o(\log\log n)$ in the synthetic FFT model.

Resolution of an r-dimensional system of algebraic differential equations	
Algorithm	Complexity
Relaxed	$\sim \mathsf{R}(n)\, s + O(n\, t)$
Newton	$\sim \mathsf{M}(n)\, (s\, r + 8/3\, r^2 + 4/3\, r) + O(t\, r\, n)$
New	$\sim \mathsf{M}(n)\, (s + 4/3\, r) + O(t\, n)$

Table 2. Complexities for the resolution of an r-dimensional system $\delta F = P(F)$ up to n terms, where P is a polynomial of multiplicative size s and total size t. For the bottom line, we assume the standard FFT model and we require that $r = o(\log n)$. In the synthetic FFT model, the bound becomes $\sim \mathsf{M}(n)\, (s + 4/3\, r) + O(t\, n \log n)$, under the assumption $\log^2 r \log\log r = o(\log\log n)$.

A final interesting question is to which extent Newton's method can be generalized. Clearly, it is not hard to consider more general equations of the kind

$$\delta F = P(F, F(z^2), ..., F(z^p)),$$

since the series $F(z^2), ..., F(z^p)$ merely act as perturbations. However, it seems harder (but maybe not impossible) to deal with equations of the type

$$\delta F = P(F, F(q\, z)),$$

since it is not clear *a priori* how to generalize the concept of a fundamental system of solutions and its use in the Newton iteration. In the case of partial differential equations with initial conditions on a hyperplane, the fundamental system of solutions generally has infinite dimension, so essentially new ideas would be needed here. Nevertheless, the strategy of relaxed evaluation applies in all these cases, with the usual $O(\log n)$ overhead in general and $O(\mathrm{e}^{2\sqrt{\log\log n}})$ overhead in the synthetic FFT model.

BIBLIOGRAPHY

[BCO+06] A. Bostan, F. Chyzak, F. Ollivier, B. Salvy, É. Schost, and A. Sedoglavic. Fast computation of power series solutions of systems of differential equation. preprint, april 2006. submitted, 13 pages.

[Ber] D. Bernstein. Removing redundancy in high precision Newton iteration. Available from `http://cr.yp.to/fastnewton.html`.

[BK78] R.P. Brent and H.T. Kung. Fast algorithms for manipulating formal power series. *Journal of the ACM*, 25:581–595, 1978.

[CK91] D.G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28:693–701, 1991.

[Coo66] S.A. Cook. *On the minimum computation time of functions*. PhD thesis, Harvard University, 1966.

[CT65] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Computat.*, 19:297–301, 1965.

[HQZ04] Guillaume Hanrot, Michel Quercia, and Paul Zimmermann. The middle product algorithm I. speeding up the division and square root of power series. *AAECC*, 14(6):415–438, 2004.

[HZ] G. Hanrot and P. Zimmermann. Newton iteration revisited. `http://www.loria.fr/ zimmerma/papers/fastnewton.ps.gz`.

[HZ02] Guillaume Hanrot and Paul Zimmermann. A long note on Mulders' short product. Research Report 4654, INRIA, December 2002. Available from `http://www.loria.fr/ hanrot/Papers/mulders.ps`.

[KO63] A. Karatsuba and J. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7:595–596, 1963.

[Loh88] R. Lohner. *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendugen*. PhD thesis, Universität Karlsruhe, 1988.

[Loh01] R. Lohner. On the ubiquity of the wrapping effect in the computation of error bounds. In U. Kulisch, R. Lohner, and A. Facius, editors, *Perspectives of enclosure methods*, pages 201–217. Springer, 2001.

[LS03] G. Lecerf and É. Schost. Fast multivariate power series multiplication in characteristic zero. *SADIO Electronic Journal on Informatics and Operations Research*, 5(1):1–10, September 2003.

[MB96] K. Makino and M. Berz. Remainder differential algebras and their applications. In M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors, *Computational differentiation: techniques, applications and tools*, pages 63–74, SIAM, Philadelphia, 1996.

[MB04] K. Makino and M. Berz. Suppression of the wrapping effect by taylor model-based validated integrators. Technical Report MSU Report MSUHEP 40910, Michigan State University, 2004.

[MC79] R.T. Moenck and J.H. Carter. Approximate algorithms to derive exact solutions to systems of linear equations. In *Symbolic and algebraic computation (EUROSAM '79, Marseille)*, volume 72 of *LNCS*, pages 65–73, Berlin, 1979. Springer.

[Moo66] R.E. Moore. *Interval analysis*. Prentice Hall, Englewood Cliffs, N.J., 1966.

[Pan84] V. Pan. *How to multiply matrices faster*, volume 179 of *Lect. Notes in Math.* Springer, 1984.

[Sch33] G. Schulz. Iterative Berechnung der reziproken Matrix. *Z. Angew. Math. Mech.*, 13:57–59, 1933.

[Sch00] A. Schönhage. Variations on computing reciprocals of power series. *Inform. Process. Lett.*, 74:41–46, 2000.

[SS71] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing 7*, 7:281–292, 1971.

[Str69] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:352–356, 1969.

[Too63] A.L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Mathematics*, 4(2):714–716, 1963.

[vdH97] J. van der Hoeven. Lazy multiplication of formal power series. In W. W. Küchlin, editor, *Proc. ISSAC '97*, pages 17–20, Maui, Hawaii, July 1997.

[vdH02] J. van der Hoeven. Relax, but don't be too lazy. *JSC*, 34:479–542, 2002.

[vdH03a] J. van der Hoeven. New algorithms for relaxed multiplication. Technical Report 2003-44, Université Paris-Sud, Orsay, France, 2003.

[vdH03b] J. van der Hoeven. Relaxed multiplication using the middle product. In Manuel Bronstein, editor, *Proc. ISSAC '03*, pages 143–147, Philadelphia, USA, August 2003.

[vdH04] J. van der Hoeven. The truncated Fourier transform and applications. In J. Gutierrez, editor, *Proc. ISSAC 2004*, pages 290–296, Univ. of Cantabria, Santander, Spain, July 4–7 2004.

[vdH05] J. van der Hoeven. Notes on the Truncated Fourier Transform. Technical Report 2005-5, Université Paris-Sud, Orsay, France, 2005.

[vdH06a] J. van der Hoeven. On effective analytic continuation. Technical Report 2006-15, Univ. Paris-Sud, 2006.

[vdH06b] J. van der Hoeven et al. Mmxlib: the standard library for Mathemagix, 2002–2006. `http://www.mathemagix.org/mml.html`.

[WC87] Winograd and Coppersmith. Matrix multiplication via arithmetic progressions. In *Proc. of the* 19^{th} *Annual Symposium on Theory of Computing*, pages 1–6, New York City, may 25–27 1987.