Journal of Symbolic Computation 45 (2010) 857–878

Contents lists available at ScienceDirect

Journal of Symbolic Computation

journal homepage: www.elsevier.com/locate/jsc

Newton's method and FFT trading

Joris van der Hoeven [1]

Laboratoire d'Informatique (LIX), UMR 7161 CNRS, École polytechnique, 91128 Palaiseau Cedex, France

ARTICLE INFO

Article history:
Received 1 January 2009
Accepted 10 March 2010
Available online 20 March 2010

Keywords:
Power series
Newton's method
Differential equation
FFT

ABSTRACT

Let $\mathcal{C}[[z]]$ be the ring of power series over an effective ring $\mathcal{C}$. In Brent and Kung (1978), 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 $M(n)$ denotes the complexity for multiplying two polynomials of degree $< n$ over $\mathcal{C}$, then the first n coefficients of the solution can be computed in time $O(M(n))$. However, this complexity does not take into account the dependency on the order r of the equation, which is exponential for the original method (van der Hoeven, 2002) and quadratic for a recent improvement (Bostan et al., 2007). In this paper, we present a technique for further improving the dependency on r, by applying Newton's method up to a lower order, such as n/r, and trading the remaining Newton steps against a lazy or relaxed algorithm in a suitable FFT model. The technique leads to improved asymptotic complexities for several basic operations on formal power series, such as division, exponentiation and the resolution of more general linear and non-linear systems of equations.

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 fact, it will suffice that all natural numbers up to the desired series expansion order can be inverted in $\mathcal{C}$. 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

E-mail address: vdhoeven@lix.polytechnique.fr.
URL: http://www.lix.polytechnique.fr/~vdhoeven.
[1] Fax: +33 1 69 33 41 58.

0747-7171/$ – see front matter © 2010 Published by Elsevier Ltd
doi:10.1016/j.jsc.2010.03.005

$g_0 = 0$. In addition, we require that the coefficient $\Phi(F)_n$ of z^n in $\Phi(F)$ 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

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

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

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

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 evaluation and interpolation. 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 $M(n) = O(n \log n)$ operations over $\mathcal{C}$, using the fast Fourier transform (Cooley and Tukey, 1965). In general, such roots of unity can be added artificially (Schönhage and Strassen, 1971; Cantor and Kaltofen, 1991; van der Hoeven, 2002) and the complexity becomes $M(n) = O(n \log n \log\log n)$. We will respectively refer to these two cases as the *standard* and the *synthetic* FFT models. More details about the evaluation–interpolation approach will be provided in Section 2.

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 $\omega < 2.376$ (Strassen, 1969; Pan, 1984; Coppersmith and Winograd, 1987). We will denote by $\mathrm{MM}(n, r)$ the cost of multiplying two polynomials of degrees $< n$ with coefficients in $\mathcal{M}_r(\mathcal{C})$. When using the evaluation–interpolation approach in the standard FFT model, one has $\mathrm{MM}(n, r) = O(nr^\omega + M(n)r^2)$ and $\mathrm{MM}(n, r) \sim M(n)r^2$ if $r = O(\log n)$.

In Brent and Kung (1978), 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 $O(M(n))$. However, this complexity does not take into account the dependence on the order r, which was shown to be exponential in van der Hoeven (2002). Recently Bostan et al. (2007), this dependence on r has been reduced to a quadratic factor. In particular, the first n coefficients of the solution F to (3) can be computed in time $O(\mathrm{MM}(n, r))$. In fact, the resolution of (2) in the case when F and I are replaced by matrices in $\mathcal{M}_r(\mathcal{C}[[z]])$ and $\mathcal{M}_r(\mathcal{C})$, respectively, can also be done in time $O(\mathrm{MM}(n, r))$. 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 Bostan et al. (2007) 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 van der Hoeven (1997, 2002), the alternative approach of relaxed evaluation was presented in order to solve equations of the form (1). More precisely, let $R(n)$ be the cost of *gradually* computing n terms of the product $h = fg$ 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 $f_0, \ldots, f_i$ and $g_0, \ldots, g_i$ are known ($i = 0, \ldots, n-1$). In van der Hoeven (1997, 2002), we proved that $R(n) = O(M(n) \log n)$. In the standard FFT model, this bound was further reduced in van der Hoeven (2007a) to $R(n) = O(M(n) \mathrm{e}^{2\sqrt{\log 2 \log\log n}})$. We also notice that the additional $O(\log n)$ or $O(\mathrm{e}^{2\sqrt{\log 2 \log\log n}})$ overhead only occurs in FFT models: when using Karatsuba or Toom–Cook multiplication (Karatsuba and Ofman, 1963; Toom, 1963; Cook, 1966), one has $R(n) \sim M(n)$. 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.

Table 1
Table with the best asymptotic time complexities of several operations on power series with respect to the cost $M(n)$ of multiplication. The first column shows the best previously known bounds and the latter two columns the best current bounds, which all involve FFT trading. For simplicity, we assume FFT multiplication. Harvey's bounds for inversion and square roots also assume the standard FFT model.

Operation	Previous bound	This paper	Harvey (2009a,b)
Inversion	$\sim 3/2\, M(n)$		$\sim 13/9\, M(n)$
Division	$\sim 25/12\, M(n)$	$\sim 5/3\, M(n)$	
Division + remainder	$\sim 17/6\, M(n)$	$\sim 2\, M(n)$	
Square root	$\sim 11/6\, M(n)$		$\sim 4/3\, M(n)$
Exponentiation	$\sim 17/6\, M(n)$	$\sim 7/3\, M(n)$	$\sim 13/6\, M(n)$

Let $L(n, r)$ denote the complexity of computing the first n coefficients of the solution to (4). By the foregoing, we have both $L(n, r) = O(\mathrm{MM}(n, r))$ and $L(n, r) = O(M(n)r \log n)$. A natural question is whether we may further reduce this bound to $L(n, r) = O(M(n)r)$ or even $L(n, r) \sim M(n)r$. 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 $m = n/r^{\alpha}$. Then we use Newton iterations for determining successive blocks of m coefficients of F in terms of the previous coefficients of F and AF. The product AF 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 when working on FFT-ed blocks of coefficients.

In fact, FFT trading is already useful in the more elementary case of power series division. In order to enhance the readability of the paper, we will therefore introduce the technique on this example in Section 3. In the FFT model, this leads to an order n division algorithm of time complexity $D(n) \sim 5/3\, M(n)$, which improves on the best previously known bound $D(n) \sim 25/12\, M(n)$ (Hanrot and Zimmermann, 2004). Notice that $5/3\, M(n)$ should be read as $(5/3)M(n)$ and likewise for other fractions of this kind. Division with remainder of a polynomial of degree $2n$ by a polynomial of degree n can be done in time $DR(n) \sim 2\, M(n)$; the best previously known bound was $DR(n) \sim 17/6\, M(n)$ (private communication by Paul Zimmermann).

In Sections 4 and 5, we treat the cases of linear and algebraic differential equations. The main results are summarized in Tables 3 and 4 and analyzed in detail in Section 7. In particular, the exponential of a power series can now be computed at order n in time $E(n) \sim 7/3\, M(n)$ instead of $E(n) \sim 17/6\, M(n)$ (Bernstein, 2000).

In two recent papers Harvey (2009a,b) there has been further improvement of the technique of FFT trading. In the standard FFT model, the FFT coincides up to a constant factor with the inverse of its transpose. This has been exploited in Harvey (2009a) to get better bounds $I(n) \sim 13/9\, M(n)$ and $S(n) \sim 4/3\, M(n)$ for power series inversion and square roots. In Harvey (2009b), the complexity for exponentiation has been further improved to $E(n) \sim 13/6\, M(n)$. In Table 1, we have summarized the new results for elementary operations on power series.

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 the reader to Bernstein (2000) (see also Remark 11 below) for a variant and further applications of this technique (called FFT caching by the author). The central idea behind van der Hoeven (2007a) is also similar. In Section 6, 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 van der Hoeven (2002). In particular, certified integration

of dynamical systems at high precision is a topic which currently interests us (Moore, 1966; Lohner, 1988; Makino and Berz, 1996; Lohner, 2001; Makino and Berz, 2004; van der Hoeven, 2007b).

Remark 1. This paper is a thoroughly reworked version of an earlier preprint (van der Hoeven, 2006), integrating several helpful remarks by two of the referees and David Harvey.

2. Prerequisites

2.1. Evaluation–interpolation schemes

Let $\mathcal{C}$ be a ring in which $n \geqslant 2$ is not a zero-divisor and assume that $\mathcal{C}$ contains a primitive n-th root ω of unity. Given a polynomial $P \in \mathcal{C}[z]$ of degree $< n$, and identifying P with its vector $(P_0, \ldots, P_{n-1})$ of coefficients, its discrete Fourier transform $\mathrm{FFT}_\omega(P)$ is defined by

$$\mathrm{FFT}_\omega(A) = (A(1), A(\omega), A(\omega^2), \ldots, A(\omega^{n-1})) \in \mathcal{C}^n.$$

If n is a power of 2, then the fast Fourier transform (Cooley and Tukey, 1965) allows us to perform the transformation FFT_ω and its inverse

$$\mathrm{FFT}_\omega^{-1} = n^{-1}\,\mathrm{FFT}_{\omega^{-1}}$$

using $F(n) = O(n \log n)$ operations in $\mathcal{C}$. If $P, Q \in \mathcal{C}[z]$ are two polynomials with $\deg(PQ) < n$, then we clearly have $(PQ)(\omega^i) = P(\omega^i)Q(\omega^i)$ for all i, whence

$$\mathrm{FFT}_\omega(PQ) = \mathrm{FFT}_\omega(P)\,\mathrm{FFT}_\omega(Q).$$

Consequently, we may compute PQ using

$$PQ = \mathrm{FFT}_\omega^{-1}(\mathrm{FFT}_\omega(P)\,\mathrm{FFT}_\omega(Q)),$$

where $\mathrm{FFT}_\omega(P)\,\mathrm{FFT}_\omega(Q)$ stands for the componentwise product of the vectors $\mathrm{FFT}_\omega(P)$ and $\mathrm{FFT}_\omega(Q)$. If n is a power of 2, this takes $3F(n) + 2n = O(n \log n)$ operations in $\mathcal{C}$.

More generally, let $\mathcal{C}[z]_{;n}$ denote the set of polynomials of degree $< n$. Then an *evaluation–interpolation scheme* at degree $< n$ and $N(n)$ points consists of two computable $\mathcal{C}$-linear mappings

$$\mathcal{C}[z]_{;n} \overset{\mathrm{Eval}}{\longmapsto} \mathcal{C}^{N(n)}$$

$$\mathcal{C}^{N(n)} \overset{\mathrm{Eval}^{-1}}{\longmapsto} \mathcal{C}[z]_{;n}$$

with the property that

$$PQ = \mathrm{Eval}^{-1}(\mathrm{Eval}(P)\,\mathrm{Eval}(Q))$$

for all $P, Q \in \mathcal{C}[z]$ with $\deg(PQ) < n$. We will denote by $E(n)$ the maximum of the time complexities of Eval and Eval^{-1}. Given $P, Q \in \mathcal{C}[z]$ with $\deg(PQ) < n$, we may then compute PQ in time $3E(n) + N(n)$.

An *evaluation–interpolation model* is a recipe which associates an evaluation–interpolation scheme with any degree n. Most fast multiplication schemes in the literature are actually based on evaluation–interpolation models. In the sequel, we will therefore assume that the cost $M(n)$ of multiplying two polynomials of degrees $< n$ is given by

$$M(n) \sim 3E(2n) + N(2n) \tag{5}$$

for a suitable evaluation–interpolation model. Similarly, if scalar $r \times r$ matrices can be multiplied in time r^ω, then we will assume that the cost $\mathrm{MM}(n, r)$ of multiplying two $r \times r$ matrices whose entries are polynomials of degrees $< n$ is given by

$$\mathrm{MM}(n, r) \sim 3E(2n)r^2 + N(2n)r^\omega. \tag{6}$$

Notice also that a matrix–vector product takes a time

$$\mathrm{MV}(n, r) \sim E(2n)(r^2 + 2r) + N(2n)r^2. \tag{7}$$

2.2. Classical FFT models

Let $\mathcal{C}$ again be a ring in which $n = 2^p \in 2^{\mathbb{N}}$ is not a zero-divisor and assume that $\mathcal{C}$ contains a primitive n-th root of unity. Then we have seen that the FFT provides us with an evaluation–interpolation scheme at degree $< n$, with

$$\begin{aligned} E(n) &\asymp n \log n \\ N(n) &= n. \end{aligned}$$

In fact, if $n/2 < m \leqslant n$, then the truncated Fourier transform (van der Hoeven, 2004, 2005) still provides us with an evaluation–interpolation scheme with $E(n) \sim F(n)m/n$ and $N(m) = m$. We will call this evaluation–interpolation model the *standard FFT model*.

If $\mathcal{C}$ does not contain a primitive n-th root of unity, then we may artificially adjoin a suitable root of unity to $\mathcal{C}$ as follows (Schönhage and Strassen, 1971; Cantor and Kaltofen, 1991). We first decompose $p = p_1 + p_2$, $n_1 = 2^{p_1}$, $n_2 = 2^{p_2}$, with $p_1 = \lceil p/2 \rceil$ and $p_2 = \lfloor p/2 \rfloor$. Any polynomial in $\mathcal{C}[z]_{;n}$ corresponds to a unique polynomial in $\mathcal{C}[z]_n = \mathcal{C}[z]/(z^n + 1)$. We will consider the problem of multiplying in the latter ring. Consider the following sequence:

$$\mathcal{C}[z]_n = = \mathcal{C}[z_1]_{n_1}[z]/(z^{n_2} - z_1) \xrightarrow{\iota} \mathcal{C}[z_1]_{n_1}[z_2]_{;2n_2} \xrightarrow{\text{FFT}} (\mathcal{C}[z_1]_{n_1})^{n_2}.$$

The first map is a natural identification when setting $z_1 = z^{n_2}$. The injection ι corresponds to writing elements of $\mathcal{C}[z_1]_{n_1}[z]/(z^{n_2} - z_1)$ as polynomials in z_2 of degrees $< n_2$, and padding with zeros. Since z_1 is a primitive $(2n_1)$-th root of unity and $n_1 = n_2$ or $n_1 = n_2 + 1$, we may finally perform an FFT in z_2 with respect to z_1 or z_1^2. Each of the arrows can be reversed; in the case of ι, we take

$$\iota^{-1}(P_0 + \cdots + P_{2n_2-1} z_2^{2n_2-1}) = (P_0 + z_1 P_{n_2}) + \cdots + (P_{n_2-1} + z_1 P_{2n_2-1}) z^{n_2-1}.$$

In particular, we have $PQ = \iota^{-1}(\text{FFT}^{-1}(\text{FFT}(\iota(P))\,\text{FFT}(\iota(Q))))$ for all $P, Q \in \mathcal{C}[z]_n$. Repeating the construction on $\mathcal{C}[z_1]_{n_1}$, we end up with an evaluation–interpolation model with

$$\begin{aligned} E(n) &= O(n \log n \log\log n) \\ N(n) &= O(n \log n). \end{aligned}$$

We will call this model the *synthetic FFT model*. Using ideas similar to those in Cantor and Kaltofen (1991), the model adapts to the case when 2 is a zero-divisor.

2.3. Classical evaluation–interpolation models

If $\mathcal{C}$ is infinite, then we may also use multipoint evaluation and interpolation in order to construct an evaluation–interpolation scheme at any degree. Using arbitrary points, we obtain (Moenck and Borodin, 1972; Strassen, 1973; Borodin and Moenck, 1974)

$$\begin{aligned} E(n) &= O(M(n) \log n) \\ N(n) &= n. \end{aligned}$$

If it is possible to take points in a geometric progression, then one even has (Bostan and Schost, 2005)

$$\begin{aligned} E(n) &= O(M(n)) \\ N(n) &= n. \end{aligned}$$

Of course, these evaluation–interpolation models are already based on fast multiplication, whence they are not suitable for designing the fast multiplication (5). On the other hand, for large values of r, they may perform better than the synthetic FFT model on matrix and matrix–vector products (6) and (7).

For small values of n, it is sometimes interesting to use simpler, but asymptotically slower evaluation–interpolation models. For instance, we may iterate the construction

$$\mathcal{C}[z]_{;n} \hookrightarrow \mathcal{C}[z^2]_{;\lceil n/2 \rceil}[z]_{;2} \xrightarrow{\text{Kar}} \mathcal{C}[z^2]^3_{;\lceil n/2 \rceil},$$

where

$$\mathrm{Kar}(A_0 + A_1 z) = (A_0, A_0 + A_1, A_1).$$

This yields an evaluation–interpolation model with

$$\begin{aligned} E(n) &= O(n^{\log 3/\log 2}) \\ N(n) &= O(n^{\log 3/\log 2}). \end{aligned}$$

This "*Karatsuba model*" corresponds to even–odd Karatsuba multiplication. In a similar way, one may construct Toom–Cook models.

A lot of the complexity results for polynomials also hold for integers, by considering them as the evaluations of polynomials in 2^b for a suitable word length b. For integer matrix multiplications, several evaluation–interpolation models are of interest. First of all, one may use approximate floating point arithmetic of bit length $2b + O(\log n)$. Secondly, one may fit the b-bit coefficients in $\mathbb{F}_p$ where $\mathbb{F}_p$ has many 2^q-th roots of unity (e.g. $p = 3 \cdot 2^{30} + 1$). These two models are counterparts of the standard FFT model. One may also use the Schönhage–Strassen model (which is the counterpart of the synthetic FFT model). For large matrix sizes, one may finally use Chinese remaindering, which is the natural counterpart of multipoint evaluation.

In practice, operations in $\mathcal{C}$ do not have a constant cost. Nevertheless, when computing with truncations of a power series $f \in \mathcal{C}[[z]]$, it is usually the case that the bit size of f_i is proportional to i (or a power of i). Consequently, the worst cost of an operation in $\mathcal{C}$ is usually bounded by a constant times the average cost of the same operation over the complete computation.

2.4. Middle products

Let $h = fg$ be the product of two power series $f, g \in \mathcal{C}[[z]]$. In order to efficiently compute only a part $h_i, \ldots, h_{i+n-1}$ of h, a useful tool is the so called "middle product" (Hanrot et al., 2004). Let $R, Q \in \mathcal{C}[z]$ be two polynomials with $\deg R < 2n$ and $\deg Q \leqslant n$. Then we define their *middle product* $R \mathbin{\ltimes_n} Q$ (or simply $R \ltimes Q$ if n is clear from the context) by

$$P = R \ltimes Q = \sum_{i<n} \left[\sum_{j=0}^{n} R_{i+j} Q_{n-j} \right] z^i.$$

Notice that this definition generalizes to the case when $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are arbitrary rings with a multiplication $\cdot : \mathcal{A} \times \mathcal{B} \to \mathcal{C}$, and $f \in \mathcal{A}[[z]]$, $g \in \mathcal{B}[[z]]$. In matrix form, we have

$$V_P = \begin{pmatrix} P_0 \\ \vdots \\ P_{n-1} \end{pmatrix} = \begin{pmatrix} Q_n & \cdots & Q_0 & & \\ & \ddots & & \ddots & \\ & & Q_n & \cdots & Q_0 \end{pmatrix} \begin{pmatrix} R_0 \\ \\ \vdots \\ \\ R_{2n-1} \end{pmatrix} = T_Q V_R.$$

This formula is almost the transposed form of a usual product. More precisely, if $R = PQ$ with $\deg P < n$ and $\deg Q \leqslant n$, then we have

$$V_R = \begin{pmatrix} R_0 \\ \\ \vdots \\ \\ R_{2n-1} \end{pmatrix} = \begin{pmatrix} Q_0 & & \\ \vdots & \ddots & \\ Q_n & & Q_0 \\ & \ddots & \vdots \\ & & Q_n \end{pmatrix} \begin{pmatrix} P_0 \\ \vdots \\ P_{n-1} \end{pmatrix} = M_Q V_P.$$

In other words, $T_Q = M_{\mathrm{Rev}(Q)}^{\top}$, where $M^{\top}$ denotes the transpose of a matrix M and $\mathrm{Rev}(Q) = \mathrm{Rev}_n(Q) = Q_0 z^n + \cdots + Q_n$.

For a fixed evaluation–interpolation scheme, the product $M_Q V_P$ is computed efficiently using evaluation and interpolation. More precisely, the operator Eval at degree $< 2n$, restricted to polynomials of degree $< n$ corresponds to an $N(2n) \times n$ matrix E:

$$\mathrm{Eval}(P) = EV_P.$$

Let Δ_Q be the diagonal matrix with entries $\mathrm{Eval}(Q)$. Then

$$\mathrm{Eval}(PQ) = \Delta_Q E V_P.$$

Finally, the operator Eval^{-1} at degree $< 2n$ corresponds to a $(2n) \times N(2n)$ matrix $\hat{E}$:

$$PQ = \hat{E} \Delta_Q E V_P.$$

Since this equality holds for all P, it follows that

$$\begin{aligned} M_Q &= \hat{E} \Delta_Q E \\ T_Q &= E^\top \Delta_{\mathrm{Rev}(Q)} \hat{E}^\top. \end{aligned}$$

Assuming that the algorithms Eval and Eval^{-1} for evaluation and interpolation only use $\mathcal{C}$-linear operations, the actions of $E^\top$ and $\hat{E}^\top$ on vectors can be computed by the transpositions tEval and tEval^{-1} of these algorithms in time $E(2n) + O(N(2n))$ (Bordewijk, 1956; Bernstein, 0000). We may thus compute the middle product using

$$R \Join Q = \mathrm{tEval}(\mathrm{tEval}^{-1}(R)\, \mathrm{Eval}(\mathrm{Rev}(Q))) \tag{8}$$

in time $\sim 3E(2n) + O(N(2n))$. In the standard FFT model, the matrix $\hat{E}$ is actually symmetric and E is the upper half of a symmetric matrix. Hence, (8) becomes

$$R \Join Q = \mathrm{FFT}_\omega(\mathrm{FFT}_\omega^{-1}(R)\, \mathrm{FFT}_\omega(\mathrm{Rev}(Q))) \bmod z^n.$$

One may also use the alternative formula (Harvey, 2009a)

$$R \Join Q = \mathrm{FFT}_\omega^{-1}(\mathrm{FFT}_\omega(R)\, \mathrm{FFT}_\omega(Q)) \operatorname{div} z^n,$$

which is based on the fact that standard FFT multiplication of R and Q really computes the exact product of RQ modulo $z^{2n} - 1$. Writing $P = P_0 + P_1 z^n + P_2 z^{2n} = RQ$ with $\deg P_i < n$, we then notice that P_1 coincides with the middle product, whereas $P = (P_0 + P_2) + P_1 z^n$ modulo $z^{2n} - 1$.

3. Division

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) and integers $0 \leqslant i \leqslant j$, we will use the notation

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

By convention, $f_{;0} = 0$ and $f_{i;i} = 0$ for all i.

3.1. Blockwise products

Let $m \in \mathbb{N}^>$ be fixed. Any series $F \in \mathcal{C}[[z]]$ in z may be rewritten blockwise as a power series $\bar{F}$ in $Z = z^m$ with coefficients $\bar{F}_i \in \mathcal{C}[z]$, $\deg \bar{F}_i < m$:

$$\begin{aligned} \bar{F} &= \bar{F}_0 + \bar{F}_1 Z + \bar{F}_2 Z^2 + \cdots \\ \bar{F}_i &= F_{im;(i+1)m}. \end{aligned} \tag{9}$$

Let us now consider the computation of a product $P = AF$, where $A, F \in \mathcal{C}[[z]]$. The coefficients of the blockwise product $\bar{A}\bar{F}$ are polynomials of degrees $< 2m$ instead of $< m$. In order to recover

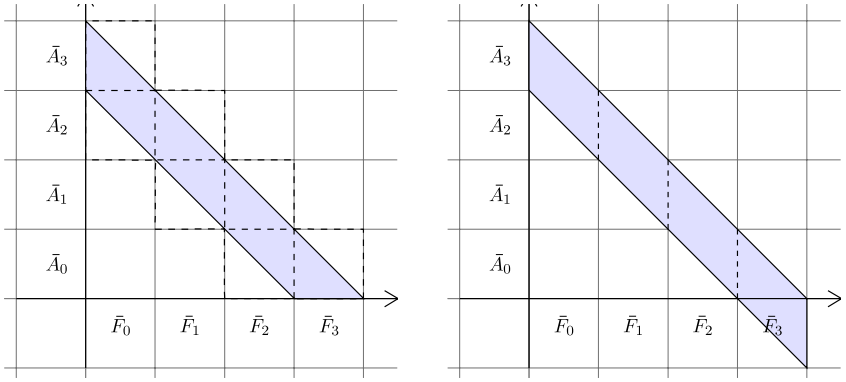

Fig. 1. Two ways to compute the coefficient $\bar{P}_3$, with $P = AF$. At the left-hand side, we use $\bar{P}_3 = (\bar{A}_2\bar{F}_0 + \bar{A}_1\bar{F}_1 + \bar{A}_0\bar{F}_2) \operatorname{div} z^m + (\bar{A}_3\bar{F}_0 + \bar{A}_2\bar{F}_1 + \bar{A}_1\bar{F}_2 + \bar{A}_0\bar{F}_3) \bmod z^m$. At the right-hand side, we use middle products: $\bar{P}_3 = (\bar{A}_2 + \bar{A}_3 z^m) \mathbin{\kappa} \bar{F}_0 + (\bar{A}_1 + \bar{A}_2 z^m) \mathbin{\kappa} \bar{F}_1 + (\bar{A}_0 + \bar{A}_1 z^m) \mathbin{\kappa} \bar{F}_2 + (\bar{A}_{-1} + \bar{A}_0 z^m) \mathbin{\kappa} \bar{F}_3$.

the coefficients of $\bar{P}$, we define the "contraction operator" Con: given a series $\bar{\Phi} \in \mathcal{C}[z][[Z]]$, whose coefficients are polynomials of degrees $< 2m$, we let

$$\operatorname{Con}(\bar{\Phi})_i = \overline{\Phi_{i-1}} \operatorname{div} z^m + \overline{\Phi_i} \bmod z^m.$$

Then we have

$$\bar{P} = \operatorname{Con}(\bar{A}\bar{F}). \tag{10}$$

Alternatively, we may first "extend" the series $\bar{A}$ using

$$\operatorname{Ext}(\bar{A})_i = \bar{A}_{i-1} + \bar{A}_i z^m$$

and then compute $\bar{P}$ using middle products:

$$\bar{P} = \operatorname{Ext}(\bar{A}) \boxtimes \bar{F} = \sum_{i_1+i_2=i} \operatorname{Ext}(\bar{A})_{i_1} \mathbin{\kappa} \bar{F}_{i_2}. \tag{11}$$

These formulas are illustrated in Fig. 1. From now on, and for reasons which are detailed in Remark 2, we will use formula (11) for all product computations.

Assume now that we have fixed an evaluation–interpolation model for polynomials of degrees $< 2m$. Then we may replace the polynomial representations of the blockwise coefficients $\bar{A}_i$ and $\bar{F}_i$ by their transforms

$$\bar{A}_i^* = \operatorname{tEval}^{-1}(\bar{A}_{i-1} + \bar{A}_i z^m) \tag{12}$$

$$\bar{F}_i^* = \operatorname{Eval}(\operatorname{Rev}(\bar{F}_i)), \tag{13}$$

compute convolution products in the transformed model

$$\bar{P}_i^* = \bar{A}_i^* \mathbin{\kappa} \bar{F}_0^* + \cdots + \bar{A}_0^* \mathbin{\kappa} \bar{F}_i^*,$$

and apply (8) in order to recover the coefficients

$$\bar{P}_i = \operatorname{tEval}(\bar{P}_i^*). \tag{14}$$

In particular, assuming $\bar{A}_0^*, \ldots, \bar{A}_i^*$ and $\bar{F}_0^*, \ldots, \bar{F}_i^*$ known, we may compute $\bar{P}_i^*$ using $(i+1)N(2m)$ scalar multiplications and $\bar{P}_i$ using an additional time $E(2m)$.

Remark 2. It turns out that the formula (11) is slightly more convenient and efficient than (10): in the applications below, the coefficients $\bar{F}_i$ will be computed one by one as a function of the previous diagonal sums $\bar{A}_i\bar{F}_0 + \cdots + \bar{A}_1\bar{F}_{i-1}$. In particular, when using (10), the computation of the high part $\bar{A}_0\bar{F}_i \operatorname{div} z^m$ of $\bar{A}_0\bar{F}_i$ will need to be done separately at the next iteration. When using middle products, this computation is naturally integrated into the product $(\bar{A}_i + \bar{A}_{i+1} z^m) \mathbin{\kappa} \bar{F}_0 + \cdots + (\bar{A}_0 + \bar{A}_1 z^m) \mathbin{\kappa} \bar{F}_i$.

3.2. Division

Let $A, B \in \mathcal{C}[[z]]$ be two power series such that A_0 is invertible. Assume that we want to compute the first $n = mk$ coefficients of $F = \frac{B}{A}$. Defining $U = \frac{1}{A}$, we first compute $\bar{U}_0$ using a classical Newton iteration (Brent and Kung, 1978; Schönhage, 2000). Given $0 \leqslant i < k$, assume that $\bar{F}_0, \ldots, \bar{F}_{i-1}$ have been computed, and let

$$\bar{P}_i = (\bar{A}_{i-1} + \bar{A}_i z^m) \ltimes \bar{F}_0 + \cdots + (\bar{A}_0 + \bar{A}_1 z^m) \ltimes \bar{F}_{i-1}.$$

Setting $\bar{A}_{-1} = 0$, we then must have

$$(\overline{AF})_i = \bar{P}_i + (\bar{A}_{-1} + \bar{A}_0 z^m) \ltimes \bar{F}_i = \bar{B}_i.$$

It thus suffices to take

$$\bar{F}_i = \bar{U}_0 (\bar{B}_i - \bar{P}_i) \bmod z^n. \tag{15}$$

Carrying out this iterative method in an evaluation–interpolation model for polynomials of degrees $< 2m$ yields the following algorithm:

Algorithm `divide`(B, A, m, k)
INPUT: two truncated power series $A, B \in \mathcal{C}[[z]]$ at order $km > 0$, such that A_0 is invertible.
OUTPUT: the truncated series $F_{;mk}$, where $F = \frac{B}{A}$.

Compute $\bar{U}_0$ and $\bar{U}_0^* = \mathrm{Eval}(\bar{U}_0)$
  For $i = 0, \ldots, k-1$ do
  $\bar{A}_i^* := \mathrm{tEval}^{-1}(\bar{A}_{i-1} + \bar{A}_i z^m)$
  $\bar{F}_{i-1}^* := \mathrm{Eval}(\mathrm{Rev}(\bar{F}_{i-1}))$
  $\bar{P}_i^* := \bar{A}_i^* \bar{F}_0^* + \cdots + \bar{A}_1^* \bar{F}_{i-1}^*$
  $\bar{P}_i := \mathrm{tEval}(\bar{P}_i^*)$
  $\bar{\Delta}_i^* := \mathrm{Eval}(\bar{B}_i - \bar{P}_i)$
  $\bar{F}_i^* := \bar{U}_i^* \bar{\Delta}_i^*$
  $\bar{F}_i := \mathrm{Eval}^{-1}(\bar{F}_i^*) \bmod z^n$
Return $\bar{F}_0 + \bar{F}_1 z^m + \cdots + \bar{F}_{k-1} z^{(k-1)m}$

Remark 3. In the above algorithm, the coefficients of $\bar{P} = (\mathrm{Ext}(\bar{A}) - \mathrm{Ext}(\bar{A})_0) \boxtimes \bar{F}$ are computed in a naive manner using

$$\bar{P}_i^* := \bar{A}_i^* \bar{F}_0^* + \cdots + \bar{A}_1^* \bar{F}_{i-1}^*.$$

Alternatively, we may rewrite (15) as an implicit equation in the transformed model and use a relaxed algorithm for its resolution (van der Hoeven, 2002, 2007a). For this purpose, we first extend the operators Rev, Eval, etc. blockwise to series S in Z using

$$\begin{aligned} \mathrm{Rev}(S) &= \mathrm{Rev}(S_0) + \mathrm{Rev}(S_1) Z + \cdots \\ \mathrm{Eval}(S) &= \mathrm{Eval}(S_0) + \mathrm{Eval}(S_1) Z + \cdots \\ &\vdots \end{aligned}$$

Then the Eq. (15) may be rewritten as

$$\bar{F}_i = [\bar{U}_0 (\bar{B} - (\mathrm{Ext}(\bar{A}) - \mathrm{Ext}(\bar{A})_0) \boxtimes \bar{F}) \bmod z^n]_i,$$

which leads to the blockwise implicit equation

$$\bar{F} = \bar{F}_0 + \bar{U}_0 (\bar{B} - (\mathrm{Ext}(\bar{A}) - \mathrm{Ext}(\bar{A})_0) \boxtimes \bar{F}) \bmod z^n.$$

In the transformed model, this equation becomes

$$\bar{F} = \bar{F}_0 + \mathrm{Eval}^{-1}\ (\mathrm{Eval}\ (\bar{U}_0)\ \mathrm{Eval}\ (\mathrm{tEval}\ (\mathrm{tEval}^{-1}\ (\mathrm{Ext}(\bar{A}) - \mathrm{Ext}(\bar{A})_0)\ \mathrm{Eval}\ (\mathrm{Rev}(\bar{F}))))) \bmod z^n,$$

and we solve it using a relaxed multiplication algorithm.

3.3. Complexity analysis

From now on, it will be convenient to restrict our attention to evaluation–interpolation models for which $E(n)/n$ and $N(n)/n$ are increasing functions in n and $N(n) = o(E(n))$. Given two functions f and g in n, we will write $f \lesssim g$ if for any $\varepsilon > 0$ we have $f(n) \leqslant (1+\varepsilon)g(n)$ for all sufficiently large n.

Theorem 1. *The quotient of two power series in $\mathcal{C}[[z]]$ can be computed at order n in time*

$$D(n) \lesssim 5E(2n).$$

Proof. Let us analyze the complexity of `divide`. The precomputation of $\bar{U}_0$ can be done in time $6E(2m) + 2N(2m)$ when using a fast algorithm (Hanrot et al., 2004) based on Newton's method. The main loop accounts for:

- Five evaluation–interpolation procedures of cost $5kE(2m)$.
- One naive order k product in the transformed model of cost $O(\frac{1}{2}k(k+1))N(2m)$. In view of Remark 3, the naive product may be replaced by a relaxed product, which leads to a cost $R(k)N(2m)$.
- k scalar multiplications in the transformed model of cost $O(k)N(2m)$.

Adding up these costs, the complete division takes a time

$$D(m,k) \lesssim (5k+7)E(2m) + (R(k)+O(k))N(2m). \tag{16}$$

Choosing k such that $1/k = o(1)$ and $R(k)N(2n/k) = o(kE(2n/k))$, the theorem follows. The choice $k = \lfloor \log\log n \rfloor$ works both in the standard and in the synthetic FFT model. □

Remark 4. In practice, the number k should be chosen not too large, so as to keep $R(k)N(2m)$ reasonably small. According to (16), we need $5k + 7 < 25/4k$ in order to beat the best previously known division algorithm (Hanrot and Zimmermann, 2004), which happens for $k > 28/5$.

Remark 5. For small values of k, the fact that we perform more multiplications in the transformed model is compensated by the fact that the FFTs are computed for smaller sizes. Let us compare in detail a truncated FFT multiplication at order n and a blockwise multiplication as in Section 3.1 at order $n = km$.

For simplicity, we will assume the standard FFT model and naive inner multiplication. The $2n$ inner multiplications in the classical FFT multiplication are replaced by $(k+1)n$ inner multiplications in the blockwise model, accounting for $(k-1)n$ extra multiplications. Every FFT at size $2n$ is replaced by k FFTs of size $2m$, thereby saving approximately $n\log_2 k$ multiplications. For $k = 4$, the blockwise algorithm therefore saves $6n - 3n = 3n$ multiplications. For $k = 8$, we save $9n - 7n = 2n$ multiplications. For $k = 16$, we perform $15n - 12n = 3n$ more multiplications.

Using relaxed Karatsuba multiplication, we only need $81/16\ n$ inner multiplications in the blockwise model for $k = 16$, so we save $(9 - 1/16)n$ multiplications. We also notice that the division algorithm requires five FFTs instead of three for multiplication. For $k = 16$ and naive inner multiplication, this means that we actually "save" $20n - 15n$ multiplications. In any case, the analysis shows that blockwise algorithms should perform well for moderate values of k, at least for the standard FFT model.

Remark 6. For $i = k, \ldots, 2k-2$, the coefficients

$$\bar{\Delta}_i = \bar{B}_i - (\bar{A}_{k-2} + \bar{A}_{k-1}z^m) \times \bar{F}_{i+1-k} - \cdots - (\bar{A}_{i-k} + \bar{A}_{i+1-k}z^m) \times \bar{F}_{k-1}$$

can be computed using $k-1$ additional transforms of cost $(k-1)E(2m)$ and $M(k)N(2m)$ additional inner multiplications. This implies that the quotient and the remainder of a division of a polynomial V of degree $< 2n$ by a polynomial U of degree $n-1$ can be computed in time $\lesssim 6E(2n)$. Indeed, it suffices to take $A(z) = z^{n-1}U(1/z)$, $B(z) = z^{2n-1}V(1/z)$, and apply the above argument.

Remark 7. The division algorithm should also apply for integers and floating point numbers instead of polynomials and truncated power series. Of course, this requires a certain amount of extra work in order to handle carries correctly. We also expect FFT trading to be more efficient for standard FFT models (FFT multiplication over complex doubles or over a field $\mathbb{F}_p$ with a large 2^q-th root of unity) than for the synthetic model (Schönhage–Strassen multiplication).

4. Linear differential equations

In order to simplify our exposition, it is convenient to write all differential equations in terms of the operator $\delta = z\frac{\partial}{\partial z}$. The inverse δ^{-1} of δ is defined by

$$\delta^{-1} f = \sum_{n>0} \frac{f_n}{n} z^n,$$

for all $f \in \mathcal{C}[[z]]$ with $f_0 = 0$.

4.1. Newton iteration

Given a matrix $A \in \mathcal{M}_r(\mathcal{C}[[z]])$ with $A_0 = 0$, the equation

$$\delta M = AM \tag{17}$$

admits a unique solution $M \in \mathcal{M}_r(\mathcal{C}[[z]])$ with $M_0 = \mathrm{Id}_r$. The main idea of Bostan et al. (2007) is to provide a Newton iteration for the computation of M. More precisely, assume that $M_{;n}$ and $M_{;n}^{-1} = (M^{-1})_{;n}$ are known. Then we have

$$M_{;2n} := [M_{;n} - (M_{;n}\delta^{-1}M_{;n}^{-1})(\delta M_{;n} - A_{;2n}M_{;n})]_{;2n}. \tag{18}$$

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

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

and so $(\delta - A)(M_{;n} + \Delta) = O(z^{2n})$ and $\Delta = M_{n;2n} + O(z^{2n})$. Computing $M_{;n}$ and $M_{;n}^{-1}$ together using (18) and the usual Newton iteration (Schulz, 1933; Moenck and Carter, 1979)

$$M_{;2n}^{-1} = [M_{;n}^{-1} + M_{;n}^{-1}(\mathrm{Id}_r - M_{;n}M_{;n}^{-1})]_{;2n} \tag{19}$$

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

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 $n \geqslant m > 0$, we have

$$M_{;n+m} := [M_{;n} - (M_{;m}\delta^{-1}M_{;m}^{-1})(\delta M_{;n} - A_{;n+m}M_{;n})]_{;n+m}. \tag{20}$$

This relation is proved in a similar way to (18). 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

$$\delta F = AF \tag{21}$$

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

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

or

$$F_{n;n+m} := [(M_{;m}\delta^{-1}M_{;m}^{-1})((A_{;n+m}F_{;n})_{n;n+m}z^n)]_{n;n+m}. \tag{22}$$

Both the right-hand sides of the Eqs. (20) and (22) may be computed efficiently using the middle product algorithm.

4.2. Blockwise resolution

The block notation $\bar{A}$, $\bar{F}$ from Section 3.1 naturally generalizes to series of matrices and series of vectors. The derivation δ operates in a blockwise fashion:

$$\bar{\delta}(\bar{F}_i Z^i) = (\bar{\delta}_i \bar{F}_i) Z^i = (im\bar{F}_{i,0} + \cdots + (im + m - 1)\bar{F}_{i,m-1} z^{m-1}) Z^i.$$

We define the blockwise operator $\bar{\Lambda}$, with

$$\bar{\Lambda}\bar{P} = \bar{\Lambda}\bar{P}_0 + \bar{\Lambda}(\bar{P}_1 Z) + \cdots,$$

by

$$\bar{\Lambda}(\bar{P}_i Z^i) = (\bar{\Lambda}_i \bar{P}_i) Z^i = [\overline{(\bar{M}_0 \delta^{-1} \bar{M}_0^{-1})(\bar{P}_i Z^i)}]_i Z^i.$$

In practice, we may compute $\bar{\Lambda}_i \bar{P}_i$ by

$$\begin{aligned} X &= \bar{M}_0^{-1} \bar{P}_i \bmod z^m \\ Y &= \bar{\delta}_i^{-1} X \\ \bar{\Lambda}_i \bar{P}_i &= \bar{M}_0 Y \bmod z^m. \end{aligned}$$

Now (22) yields a formula for the blockwise resolution of (17):

$$\begin{aligned} \bar{F}_i &= \bar{\Lambda}_i [\overline{A(F - \bar{F}_i z^{im})}]_i. \\ &= \bar{\Lambda}_i [(\bar{A}_{i-1} + \bar{A}_i z^m) \rtimes \bar{F}_0 + \cdots + (\bar{A}_0 + \bar{A}_1 z^m) \rtimes \bar{F}_{i-1}]. \end{aligned} \tag{23}$$

Assume that we have fixed an evaluation–interpolation scheme at degree $< 2m$. Replacing the blockwise coefficients $\bar{A}_i$ and $\bar{F}_i$ by their transforms (12), (13) and applying (23), we may compute $\bar{\Lambda}_i \bar{P}_i$ by an evaluation–interpolation procedure:

$$\begin{aligned} X &= \mathrm{Eval}^{-1}(\mathrm{Eval}(\bar{M}_0^{-1})\, \mathrm{Eval}(\bar{P}_i)) \bmod z^m \\ Y &= \bar{\delta}_i^{-1} X \\ \bar{\Lambda}_i \bar{P}_i &= \mathrm{Eval}^{-1}(\mathrm{Eval}(\bar{M}_0)\, \mathrm{Eval}(Y)) \bmod z^m. \end{aligned}$$

Of course, $\mathrm{Eval}(\bar{M}_0)$ and $\mathrm{Eval}(\bar{M}_0^{-1})$ only need to be computed once. This leads to the following algorithm for the computation of $\bar{F}_0, \ldots, \bar{F}_{k-1}$:

Algorithm `lin_solve`(A, I, m, k)
INPUT: a linear initial value problem (21) and orders m and k
OUTPUT: the truncated series $F_{;mk}$

Compute $\bar{M}_0$, $\bar{M}_0^{-1}$ and $\bar{F}_0$ as in Section 4.1
For $i = 1, \ldots, k-1$ do
  $\bar{A}_i^* := \mathrm{tEval}^{-1}(\bar{A}_{i-1} + \bar{A}_i z^m)$
  $\bar{F}_{i-1}^* := \mathrm{Eval}(\mathrm{Rev}(\bar{F}_{i-1}))$
  $\bar{P}_i^* := \bar{A}_i^* \bar{F}_0^* + \cdots + \bar{A}_1^* \bar{F}_{i-1}^*$
  $\bar{P}_i := \mathrm{tEval}(\bar{P}_i^*)$
  $\bar{F}_i := \bar{\Lambda}_i \bar{P}_i$
Return $\bar{F}_0 + \bar{F}_1 z^m + \cdots + \bar{F}_{k-1} z^{(k-1)m}$

Remark 8. In the above algorithm, the coefficients of $\bar{P}$ are computed in a naive manner. In a similar way to in Remark 3, we may use a relaxed algorithm instead. More precisely, the Eq. (23) may be rewritten as

$$\bar{F}_i = [\bar{\Lambda}((\mathrm{Ext}(\bar{A}) - \mathrm{Ext}(\bar{A})_0) \boxtimes \bar{F})]_i,$$

which leads to the blockwise implicit equation

$$\bar{F} = \bar{F}_0 + \bar{\Lambda}((\mathrm{Ext}(\bar{A}) - \mathrm{Ext}(\bar{A})_0) \boxtimes \bar{F}).$$

In the transformed model, this equation becomes

$$\bar{F} = \bar{F}_0 + \bar{\Lambda}\,\mathrm{tEval}(\mathrm{tEval}^{-1}(\mathrm{Ext}(\bar{A}) - \mathrm{Ext}(\bar{A})_0)\,\mathrm{Eval}(\mathrm{Rev}(\bar{F}))).$$

We understand that $\bar{\Lambda}$ is computed blockwise in the transformed model.

4.3. Complexity analysis

Assuming that A has s non-zero entries, we denote by $L(n, r, s)$ the time complexity for computing the truncated solution $F_{;n}$ to (21) at order n.

Theorem 2. *Consider the differential equation (21), where A has s non-zero entries. Assume that* $r^{\omega-1} = o(n)$ *and* $R(r^{\omega-1})/r^{\omega-1} = o(E(2n)/N(2n))$. *Then there exists an algorithm which computes the truncated solution* $F_{;n}$ *to (21) at order n in time*

$$L(n, r, s) \lesssim E(2n)(s + 6r) + O(N(2n)r^2). \tag{24}$$

Proof. In our algorithm, let ϕ_n be a function which increases towards infinity, such that $\phi_n r^{\omega-1} = o(n)$ and $R(\phi_n r^{\omega-1})/(\phi_n r^{\omega-1}) = o(E(2n)/N(2n))$. We take $k = \lfloor \phi_n r^{\omega-1} \rfloor$ and $m = \lceil n/k \rceil$, and so $km - n \leqslant k = o(n)$. Let us carefully examine the cost of our algorithm for these choices of k and m:

(1) By the choice of k, the precomputation of $\bar{F}_0$, $\bar{M}_0$ and $\bar{M}_0^{-1}$ requires a time

$$O(\mathrm{MM}(m, r)) = O(E(2m)r^2 + N(2m)r^\omega) = O\left(E(2n)\frac{r^\omega}{k}\right) = o(E(2n)r).$$

Similarly, the precomputation of $\mathrm{Eval}(\bar{M}_0)$ and $\mathrm{Eval}(\bar{M}_0^{-1})$ can be done in time

$$O(E(2m)r^2) = o(E(2n)r).$$

(2) The computation of the transforms $\bar{A}_i^*$, $\bar{F}_i^*$ and the inverse transforms $\bar{P}_i$ can be done in time

$$\lesssim E(2m)k(s + 2r) \leqslant E(2n)(s + 2r).$$

(3) The computation of $O(k^2)$ products $\bar{A}_i^* \bar{F}_j^*$ in the transformed model requires a time

$$O(k^2 N(2m)s) = O(N(2n)ks).$$

Using a relaxed multiplication algorithm, this cost further reduces to

$$O(R(k)N(2m)s) = O\left(N(2n)\frac{R(k)}{k}s\right).$$

(4) The computation of $\bar{F}_i = \bar{\Lambda}_i \bar{P}_i$ involves $4(k-1)$ vectorial evaluations and interpolations, of cost

$$\lesssim 4E(2m)kr \leqslant 4E(2n)r$$

and $O(k)$ matrix–vector multiplications in the transformed model, of cost

$$O(kN(2m)r^2) = O(N(2n)r^2).$$

Adding up the different contributions, we obtain the bound

$$L(n, r, s) \lesssim E(2n)(s + 6r) + N(2n)O\left(\frac{R(k)}{k}s + r^2\right).$$

By construction, $N(2n)R(k)/k = o(E(2n))$, and the result follows. □

Corollary 1. *In the standard FFT model, and under the assumption $r = o(\log n)$, we have*

$$L(n, r, s) \lesssim (s/3 + 2r)M(n)$$
$$L(n, r) \lesssim 7/3r\, M(n).$$

The same bounds are obtained in the synthetic FFT model if $r = o(\log \log n)$.

Remark. Of course, $7/3r\, M(n)$ should be read $(7/3)r\, M(n)$ and likewise below.

Proof. In the standard FFT model, we have $E(2n) \sim M(n)/3$, $E(2n)/N(2n) = O(\log n)$ and $R(r^{\omega-1})/r^{\omega-1} = O(\mathrm{e}^{2\sqrt{\log 2 \log\log r}} \log r)$. If $r = o(\log n)$, we may therefore apply the theorem and the second term in (24) becomes negligible with respect to the first one.

In the synthetic FFT model, we have $E(2n) \sim M(n)/3$, $E(2n)/N(2n) = O(\log \log n)$ and $R(r^{\omega-1})/r^{\omega-1} = O((\log r)^2 \log \log r)$. If $r = o(\log \log n)$, we may again apply the theorem and the second term in (24) becomes negligible. □

4.4. Further observations

Remark 9. In practice, one should choose ϕ_n just sufficiently large that the precomputation has a small cost with respect to the remainder of the computation. This is already the case for ϕ_n close to 1.

Remark 10. The use of middle products was needed in order to achieve the factor $s/3 + 2r$ in Corollary 1. As explained in Remark 2, using a more straightforward multiplication algorithm seems to require one additional transform. This leads to the factor $s/3 + 7r/3$.

Remark 11. Corollary 1 applies in particular to the exponentiation $f = \mathrm{e}^g$ of a power series g. We obtain an algorithm of time complexity $L(n, 1) \lesssim 7/3\, M(n)$, which yields an improvement over Bernstein (2000) and Hanrot and Zimmermann (2004). Notice that FFT trading is a variant of Newton caching in Bernstein (2000), 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.

Remark 12. With minor changes, the algorithm can be adapted in order to compute the unique solution of the matrix differential equation $\delta M = MF$ with $M_0 = \mathrm{Id}_r$. The unique solution M corresponds to a fundamental system of solutions to (21). A complexity analysis similar to the one in the proof of Theorem 2 yields the bound

$$\mathrm{LM}(n, r) \lesssim 7r^2 E(2n) + O(r^\omega N(2n)).$$

Under the additional hypotheses of the corollary, we thus get

$$\mathrm{LM}(n, r) \lesssim 7/3r^2\, M(n).$$

Remark 13. In the standard FFT model, the conditions of Theorem 2 reduce to

$$\log r = o\left(\frac{\log n}{\mathrm{e}^{2\sqrt{\log 2 \log\log n}}}\right). \tag{25}$$

If $s = r$, then we obtain

$$L(n, r) = nrO(\log n + r).$$

This complexity should be compared to the bound provided by a relaxed approach

$$L(n, r) = O(R(n)r) = O(nr \log n \mathrm{e}^{2\sqrt{\log 2 \log\log n}}).$$

If $r = o(\log n)$, our new approach gains a factor $O(\mathrm{e}^{2\sqrt{\log 2 \log\log n}})$. On the other hand, the relaxed approach becomes more efficient for moderate orders $\log n \mathrm{e}^{2\sqrt{\log 2 \log\log n}} = O(r)$.

In the case when $s = r^2$, the theorem yields

$$L(n, r, r^2) = O(nr^2 \log n),$$

whereas the relaxed approach yields the bound

$$L(n, r, r^2) = O(R(n)r^2) = O(nr^2 \log n e^{2\sqrt{\log 2 \log\log n}}).$$

We thus gain a factor $O(e^{2\sqrt{\log 2 \log\log n}})$ under the sole assumption (25).

Remark 14. In the case where $\mathcal{C}$ does not admit many 2^p-th roots of unity, we have the choice between the synthetic FFT model and the multipoint evaluation–interpolation approach. In the synthetic FFT model, the almost optimal bounds from Corollary 1 are reached under the rather harsh assumption $r = o(\log\log n)$. This makes the method interesting only for particularly low orders $r \leqslant 3$ (maybe $r \leqslant 5$ for really huge values of n).

If $\mathcal{C}$ admits an infinity of points in geometric progression, then we may also use the multipoint evaluation–interpolation approach with $E(2n) \sim cM(n)$ and $N(n) = n$ for some constant $c > 1$. In a similar way to in Corollary 1, one obtains the bound

$$L(n, r, s) \lesssim cM(n)(s/3 + 2r)$$

under the assumption $r = o(\log n \log\log n)$, since $E(2n)/N(2n) = O(\log n \log\log n)$. If $s = r^2$, then the assumption may even be replaced by $(\log r)^2 \log\log r = o(\log n \log\log n)$. Recall that $R(n) = O(M(n) \log n)$ in this context. Therefore, we potentially gain a factor $O(\log n)$ compared to the relaxed approach.

Remark 15. One may wonder whether the technique of FFT trading is useful in asymptotically less efficient models such as the Karatsuba model. Recall however that $R(n) = O(M(n))$ in any model with $M(n) \asymp n^\alpha$ for $\alpha > 1$. The Karatsuba model is even essentially relaxed, in the sense that $R(n) = M(n)$. Therefore, the use of Newton's method at best allows for the gain of a constant factor. Moreover, FFT trading also does not help, since $E(n) \sim N(n)$ in such models, so the second term in (24) can never be neglected with respect to the first term.

Remark 16. It is instructive to compare our complexity bounds with the complexity bounds if we only use Newton's method and neither FFT trading nor relaxed computations. In that case, let $T(n, r)$ denote the complexity of computing both $M_{;n}$ and $M_{;n}^{-1}$. One has

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

since the product $A_{;n}F_{;n}$ and the formulas (18) and (19) give rise to $1+2+2 = 5$ matrix multiplications. This yields

$$T(n, r) \lesssim 5\, M(n)r^2.$$

Notice that we may subtract the cost $M(n)r^2$ if the final M_n^{-1} is not needed. It follows that

$$L(n, r, s) \lesssim M(n)(17/6r^2 + s/2 + 2/3r).$$

Using a trick from Schönhage (2000), one may even prove that

$$T(n, r) \lesssim 9/2\, M(n)r^2,$$

which yields

$$L(n, r, s) \lesssim M(n)(31/12r^2 + s/2 + 2/3r).$$

5. 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 (Brent and Kung, 1978; van der Hoeven, 2002; Bostan et al., 2007). 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 $F \in \mathcal{C}[[z]]^r$ of series variables, we represent polynomials in $\mathcal{C}[[z]][F] \cong \mathcal{C}[F][[z]] = \mathcal{C}[[z]][F_1, \ldots, F_r]$ by dags (directed acyclic graphs), whose leaves are either series in $\mathcal{C}[[z]]$ or

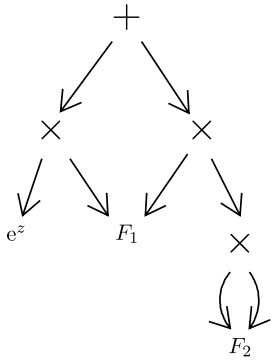

Fig. 2. Example of a polynomial expression in $\mathcal{C}[[z]][F_1, F_2]$, represented by a dag. In this particular example, the multiplicative size of the polynomial is $s = 7/3$ (since $s_1 = 4$ and $s_2 = 3$) and its total size 7.

variables F_i, and whose inner nodes are additions, subtractions or multiplications. An example of such a dag is shown in Fig. 2. We will denote by s_1 and s_2 the number of nodes which occur as an operand and a result of a multiplication, respectively. We call $s = (s_1 + s_2)/3$ the *multiplicative size* of the dag and the total number t of nodes the *total size* of the dag. Using an evaluation–interpolation procedure, one may compute $P(F)_{;n}$ in terms of $F_{;n}$ in time $\lesssim 3sE(2n) + tN(2n)$.

Now assume that we are given an r-dimensional system

$$\delta F = P(F), \tag{26}$$

with initial condition $F_0 = I \in \mathcal{C}^r$, and where $P(F)$ is a tuple of r polynomials in $z\mathcal{C}[[z]][F_1, \ldots, F_r]^r \cong \mathcal{C}[F_1, \ldots, F_r][[z]]^r$. 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 $\lesssim (2r+1)sM(m)+O(rtm)$ using the standard differentiation rules. For $n \geqslant m$, we have

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

and so

$$F_{n;n+m} = [(\delta - J_{;m})^{-1}(P(F_{;n})_{n;n+m}z^n)]_{n;n+m}. \tag{27}$$

Let us again adopt the notation (9). We will compute $\bar{F}$ and $\overline{Q(F)}$ for any subexpression $Q(F)$ of $P(F)$ in a relaxed manner. Each series $\overline{Q(F)}$ will actually be broken up into its head $\overline{Q(F)}_0$ and its tail $\overline{Q(F)}_* = \overline{Q(F)} - \overline{Q(F)}_0$, so that sums and products are really computed using

$$\begin{aligned} \overline{U(F)} + \overline{V(F)} &= (\overline{U(F)}_0 + \overline{V(F)}_0) + (\overline{U(F)}_* + \overline{V(F)}_*) \\ \overline{U(F)V(F)} &= (\overline{U(F)}_0\overline{V(F)}_0) + (\overline{U(F)}_0\overline{V(F)}_* + \overline{U(F)}_*\overline{V(F)}_0 + \overline{U(F)}_*\overline{V(F)}_*). \end{aligned}$$

Assume that $\bar{F}_j$ and $\overline{Q(F)}_j$ have been evaluated for all $j < i$ and notice that

$$F_{;n} = \bar{F}_0 + \cdots + \bar{F}_{i-1}z^{(i-1)m}.$$

The advantage of our modified way to compute the $\overline{Q(F)}$ is that it also allows us to efficiently evaluate $\overline{Q(F_{;n})}_i$. Indeed, since $\overline{Q(F_{;n})} - \overline{Q(F)} = O(Z^i)$, we have

$$\begin{aligned}\overline{(U+V)(F_{;n})}_i &= \overline{U(F_{;n})}_i + \overline{V(F_{;n})}_i\\ \overline{(UV)(F_{;n})}_i &= (\overline{U(F_{;n})}_i \overline{V(F)}_0 + \overline{U(F)}_0 \overline{V(F_{;n})}_i + (\overline{U(F)}_* \overline{V(F)}_*)_i) \bmod z^m\\ &\quad + (\overline{U(F)V(F)})_{i-1} \operatorname{div} z^m.\end{aligned}$$

We may finally compute $\bar{F}_i$ using

$$\bar{F}_i = \bar{\Lambda}_i \overline{P(F_{;n})}_i, \tag{28}$$

where $\bar{\Lambda}$ is the blockwise operator which acts on $\bar{S}_i Z^i$ via

$$\bar{\Lambda}_i \bar{S}_i = [\overline{(\delta - \bar{J}_0)^{-1} (\bar{S}_i Z^i)}]_i.$$

Let us now analyze the time complexity $A(n, r, s, t)$ of the computation of $F_{;n}$.

Theorem 3. *Consider an r-dimensional system (26), where P is a polynomial, given by a dag of multiplicative size s and total size t. Assume that $r^{\omega-1} = o(n)$ and*

$$R(r^{\omega-1})/r^{\omega-1} = o(E(2n)/N(2n)).$$

Then there exists an algorithm which computes $F_{;n}$ in time

$$A(n, r, s, t) \lesssim E(2n)(6s + 4r) + O(nt).$$

Proof. In order to perform all multiplications in the transformed model, we have to compute both $\overline{U(F)}_i^* = \mathrm{Eval}(\overline{U(F)}_i)$ and $\overline{U(F_{;n})}_i^* = \mathrm{Eval}(\overline{U(F_{;n})}_i)$ for each argument $U(F)$ of a multiplicative subexpression of $P(F)$ and $\overline{Q(F)}_i = \mathrm{Eval}^{-1}(\overline{Q(F)}_i^*)$ and $\overline{Q(F_{;n})}_i = \mathrm{Eval}^{-1}(\overline{Q(F_{;n})}_i^*)$ for each multiplicative subexpression $Q(F)$ of $P(F)$. This amounts to a total of $6ks$ evaluation–interpolation procedures of size $2m$, of cost $\lesssim 6sE(2n)$. The computations of the $\bar{F}_i$ using (28) induce an additional cost $4rE(2n)$. The relaxed multiplications in the transformed model correspond to a cost $N(2m)O(sR(k)) = o(E(2n)s)$. The additions are done in the untransformed model, in time $O(nt)$. The precomputation of $\bar{J}_0, \bar{J}_0^{-1}$ and its transforms have a negligible cost $O(rsM(m) + rtm + r^2E(2m) + r^\omega N(2m))$. □

Corollary 2. *In the standard FFT model, and assuming that $r = o(\log n)$, we have*

$$A(n, r, s, t) \lesssim M(n)(2s + 4r/3) + O(nt).$$

The same bound holds in the synthetic FFT model, assuming that $r = o(\log\log n)$.

Remark 17. In the case when most multiplications in $P(F)$ only depend linearly on F, it is possible to adopt a technique similar to that used in the previous section and perform these multiplications using the middle product. This allows for a reduction of the factor $2s$ to something between s and $2s$.

Remark 18. When solving (26) using Newton's method (Bostan et al., 2007) with the optimization from Schönhage (2000), one obtains the bound

$$A(n, r, s, t) \lesssim (2rs + 2s + 13/6r^2 + 4/3r)M(n) + O(rtn).$$

However, the factor $2rs$ is quite pessimistic. For instance, if the expressions $P_1(F), \ldots, P_r(F)$ do not share any common subexpressions, then we may use automatic differentiation Baur and Strassen (1983) for the computation of J. The multiplicative size $s' = s'_1 + s'_2$ for this circuit is given by $s'_1 = s_1 + s_2$ and $s'_2 = 3s_2$, whence $s' \leqslant 4s$ and

$$A(n, r, s, t) \lesssim (6s + 13/6r^2 + 4/3r)M(n) + O(rtn).$$

6. Truncated multiplication

Assume the standard FFT model. It is well-known that discrete FFTs 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 degrees $< n$ with $n \notin 2^{\mathbb{N}}$. One strategy for removing this problem is to use the TFT (truncated Fourier transform) as detailed in van der Hoeven (2004, 2005). Another way to smooth the complexity is to cut P and Q into smaller blocks, and trade superfluous and asymptotically expensive FFTs against asymptotically less expensive multiplications in the FFT model.

More precisely, we cut P and Q into $k = \lceil n/m \rceil$ parts of size $m = 2^p$, where $k = o(\log n)$ grows slowly to infinity with n. With the notation (9), and using FFTs at size $2m$ for evaluation and interpolation, we compute PQ as follows:

(1) We first transform $\bar{P}_i^* = \mathrm{Eval}(\bar{P}_i)$ and $\bar{Q}_i^* = \mathrm{Eval}(\bar{Q}_i)$ for $i < k$.
(2) We compute the naive product $\bar{R}^* = \bar{P}^* \bar{Q}^*$ of the polynomials $\bar{P}$ and $\bar{Q}$ in Z.
(3) We compute $\bar{R}_i = \mathrm{Eval}^{-1}(\bar{R}_i^*)$ for $i < 2k-1$ and return $\bar{R}_0 + \cdots + \bar{R}_{2k-2} z^{(2k-2)m}$.

Let C be the constant such that $M(m) \sim 3E(2m) \sim Cm \log m$ for $m \in 2^{\mathbb{N}}$. Then the above algorithm requires

$$4kE(2m) + 2k^2 m \sim 4/3Cn \log n + 2kn$$

operations in $\mathcal{C}$. If we only need the truncated product $(PQ)_{;n}$, then we may save k inverse transforms and half of the inner multiplications, so the complexity reduces to

$$3kE(2m) + k^2 m \sim Cn \log n + kn.$$

Both complexities depend smoothly on n and admit no major jumps at powers of 2.

In this particular case, it turns out that the TFT transform is always better, because both the full and the truncated product can be computed using only

$$Cn \log n + 2n$$

operations in $\mathcal{C}$. However, in the multivariate setting, the TFT also has its pitfalls. More precisely, consider two multivariate polynomials $P, Q \in \mathcal{C}[z_1, \ldots, z_d]$ 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 support types 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 even larger dimensions, one may use Lecerf and Schost (2003) or van der Hoeven (2002, Section 6.3.5). The idea is again to cut P into blocks

$$P = \sum_{i=(i_1,\ldots,i_d)} \bar{P}_i Z^i \qquad (Z^i = Z_1^{i_1} \cdots Z_d^{i_d})$$
$$\bar{P}_i = \sum_{j<(m,\ldots,m)} P_{mi+j} z^j \qquad (z^j = z_1^{j_1} \cdots z_d^{j_d})$$

where $k = \lceil n/m \rceil$ is small (and m preferably a power of 2). Each block is then transformed using an FFT (or a suitable TFT, since the supports of the blocks are still initial segments when restricted to the block). We next compute the truncated product of the transformed polynomials $\sum \bar{P}_i^* Z^i$ and $\sum \bar{Q}_i^* Z^i$ in a naive way and transform back.

Let us analyze the complexity of this algorithm. The number $M_{k,d}$ of monomials of total degree k is given by

$$M_{k,d} = \binom{k+d-1}{d-1}.$$

In particular, $M_d(z) = \sum_{k=0}^{\infty} M_{k,d} z^k = (1-z)^{-d}$ and $M_{0,d} = 1$. In order to compute the monomials in $\bar{P}\bar{Q}$ of total degree k, we need

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

since $H_d(z) = \sum_k H_{k,d} z^k = M_d(z)^2 = M_{2d}(z)$. In total, we thus need

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

multiplications of TFT-ed blocks, since $N_d(z) = \sum_k N_{k,d} z^k = \frac{z}{1-z} H_d(z) = z M_{2d+1}(z)$. For large k, we have

$$M_{k,d} \sim \frac{1}{(d-1)!} k^{d-1}$$
$$N_{k,d} \sim \frac{1}{(2d)!} k^{2d}.$$

We may therefore hope for some gain with respect to plain FFT multiplication whenever

$$N_{k,d} m^d \sim \frac{N_{k,d}}{k^d} n^d \sim \frac{k^d}{(2d)!} n^d < d n^d \log n \sim M(n^d),$$

i.e. if

$$\log n > \frac{N_{k,d}}{d k^d} \sim \frac{k^d}{d(2d)!}.$$

In Table 2, we have shown the values of $N_{k,d}/(dk^d)$ for small values of k and d. It is clear from the table that FFT trading can be used quite systematically in order to improve the performance. For larger dimensions, the gain becomes particularly important. This should not come as a surprise, because naive multiplication is more efficient than FFT multiplication for $k \leqslant d$.

Table 2
Numerical values of $N_{k,d}/(dk^d)$ for small d and k.

k	d							
	1	2	3	4	5	6	7	8
1	1.0000	0.50000	0.33333	0.25000	0.20000	0.16667	0.14286	0.12500
2	1.5000	0.62500	0.29167	0.14063	0.068750	0.033854	0.016741	0.0083008
3	2.0000	0.83333	0.34568	0.13889	0.054321	0.020805	0.0078385	0.0029150
4	2.5000	1.0938	0.43750	0.16113	0.055859	0.018514	0.0059291	0.0018482
5	3.0000	1.4000	0.56000	0.19800	0.064064	0.019413	0.0055954	0.0015504
6	3.5000	1.7500	0.71296	0.24826	0.077238	0.022105	0.0059340	0.0015144
7	4.0000	2.1429	0.89796	0.31268	0.095294	0.026299	0.0067236	0.0016179
8	4.5000	2.5781	1.1172	0.39276	0.11870	0.032036	0.0079209	0.0018266
9	5.0000	3.0556	1.3731	0.49040	0.14821	0.039506	0.0095509	0.0021357
10	5.5000	3.5750	1.6683	0.60775	0.18476	0.048988	0.011674	0.0025537
11	6.0000	4.1364	2.0055	0.74718	0.22944	0.060836	0.014378	0.0030975
12	6.5000	4.7396	2.3873	0.91124	0.28350	0.075468	0.017771	0.0037902

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 $\sum P_i z^i \to \sum \bar{P}_i Z^i$ (as well as for the "destination support"). However, the TFT also allows for some additional tricks (van der Hoeven, 2005, 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 even/odd version of Karatsuba multiplication presents the same advantage of geometry preservation (see Hanrot and Zimmermann, 2002 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 leads to a reduction in memory usage. Indeed, when computing the coefficients of a product sequentially $\bar{R} = \bar{P}\bar{Q}$, we only need to store the transform $\bar{R}_i^*$ of one coefficient in the result at a time.

7. Conclusion

We have summarized the main results of this paper in Tables 3 and 4. We recall that $R(n) = O(M(n)\mathrm{e}^{2\sqrt{\log 2 \log\log n}})$ in the standard FFT model and $R(n) = O(M(n)\log n)$ otherwise. Consequently, the new approach allows at best for a gain $O(\mathrm{e}^{2\sqrt{\log 2 \log\log n}})$ in the standard FFT model and $O(\log n)$ in the synthetic FFT model. In practice, the factor $O(\mathrm{e}^{2\sqrt{\log 2 \log\log n}})$ behaves very much like a constant, so the new algorithms become interesting only for quite large values of n. As pointed out in Remark 15, FFT trading loses its interest in asymptotically slower evaluation–interpolation models, such as the Karatsuba model. We plan to come back to practical complexity issues as soon as implementations of all algorithms are available in the Mathemagix system (van der Hoevenet al., 2002). Notice also that Newton iterations are better suited to parallel computing than relaxed evaluation is.

Table 3
Complexities for the resolution of an r-dimensional system $\delta F = AF$ 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 $r = o(\log\log n)$ in the synthetic FFT model.

Resolution of an r-dimensional system of linear differential equations		
Algorithm	Fundamental system	One solution
Relaxed	$\sim R(n)r^2$	$\sim R(n)s$
Newton	$\sim 15/4\, M(n)r^2$	$\sim M(n)(31/12r^2 + s/2 + 2/3r)$
New	$\sim 7/3\, M(n)r^2$	$\sim M(n)(s/3 + 2r)$

One interesting remaining problem is to reduce the cost of computing a fundamental system of solutions to (4). This would be possible if one could speed up the joint computation of the FFTs of $f, \delta f, \ldots, \delta^{(r-1)}f$.

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

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

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

$$\delta F = P(F, F(qz)),$$

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 an $O(\mathrm{e}^{2\sqrt{\log 2 \log\log n}})$ overhead in the standard FFT model.

Table 4
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 $\sim M(n)(2s + 4/3r) + O(tn \log n)$, under the assumption that $r = o(\log \log n)$.

Resolution of an r-dimensional system of algebraic differential equations	
Algorithm	Complexity
Relaxed	$\sim R(n)s + O(nt)$
Newton	$\sim M(n)(2sr + 2s + 13/6r^2 + 4/3r) + O(trn)$
New	$\sim M(n)(2s + 4/3r) + O(tn)$

References

Baur, W., Strassen, V., 1983. The complexity of partial derivatives. Theoretical Computer Science 22, 317–330.
Bernstein, D., The transposition principle. http://cr.yp.to/transposition.html.
Bernstein, D., 2000. Removing redundancy in high precision Newton iteration. Available from http://cr.yp.to/fastnewton.html.
Bordewijk, J.L., 1956. Inter-reciprocity applied to electrical networks. Applied Scientific Research B: Electrophysics, Acoustics, Optics, Mathematical Methods 6, 1–74.
Borodin, A., Moenck, R., 1974. Fast modular transforms. Journal of Computer and System Sciences 8, 366–386.
Bostan, A., Chyzak, F., Ollivier, F., Salvy, B., Schost, E., Sedoglavic, A., 2007. Fast computation of power series solutions of systems of differential equations. In: Proceedings of the 18th ACM–SIAM Symposium on Discrete Algorithms. New Orleans, Louisiana, USA, pp. 1012–1021.
Bostan, A., Schost, É., 2005. Polynomial evaluation and interpolation on special sets of points. Journal of Complexity 21 (4), 420–446, festschrift for the 70th Birthday of Arnold Schönhage.
Brent, R., Kung, H., 1978. Fast algorithms for manipulating formal power series. Journal of the ACM 25, 581–595.
Cantor, D., Kaltofen, E., 1991. On fast multiplication of polynomials over arbitrary algebras. Acta Informatica 28, 693–701.
Cook, S., 1966. On the minimum computation time of functions. Ph.D. Thesis, Harvard University.
Cooley, J., Tukey, J., 1965. An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation 19, 297–301.
Coppersmith, D., Winograd, S., 1987. Matrix multiplication via arithmetic progressions. In: Proc. of the 19th Annual Symposium on Theory of Computing. New York City, pp. 1–6.
Hanrot, G., Quercia, M., Zimmermann, P., 2004. The middle product algorithm I. Speeding up the division and square root of power series. Applicable Algebra in Engineering, Communication and Computing 14 (6), 415–438.
Hanrot, G., Zimmermann, P., 2002. A long note on Mulders' short product. Research Report 4654, INRIA. Available from http://www.loria.fr/~hanrot/Papers/mulders.ps.
Hanrot, G., Zimmermann, P., 2004. Newton iteration revisited. http://www.loria.fr/~zimmerma/papers/fastnewton.ps.gz.
Harvey, D., 2009a. Faster algorithms for the square root and reciprocal of power series. http://arxiv.org/abs/0910.1926.
Harvey, D., 2009b. Faster exponentials of power series. http://arxiv.org/abs/0911.3110.
Karatsuba, A., Ofman, J., 1963. Multiplication of multidigit numbers on automata. Soviet Physics Doklady 7, 595–596.
Lecerf, G., Schost, E., 2003. Fast multivariate power series multiplication in characteristic zero. SADIO Electronic Journal on Informatics and Operations Research 5 (1), 1–10.
Lohner, R., 1988. Einschließung der Lösung gewöhnlicher Anfangs- und randwertaufgaben und anwendugen. Ph.D. Thesis, Universität Karlsruhe.
Lohner, R., 2001. On the ubiquity of the wrapping effect in the computation of error bounds. In: Kulisch, U., Lohner, R., Facius, A. (Eds.), Perspectives on enclosure methods. Springer, Vienna, New York, pp. 201–217.
Makino, K., Berz, M., 1996. Remainder differential algebras and their applications. In: Berz, M., Bischof, C., Corliss, G., Griewank, A. (Eds.), Computational Differentiation: Techniques, Applications and Tools. SIAM, Philadelphia, pp. 63–74.
Makino, K., Berz, M., 2004. Suppression of the wrapping effect by Taylor model-based validated integrators. Tech. Rep. MSU Report MSUHEP 40910, Michigan State University.
Moenck, R., Borodin, A., 1972. Fast modular transforms via division. In: Thirteenth Annual IEEE Symposium on Switching and Automata Theory. Univ. Maryland, College Park, MD, pp. 90–96.
Moenck, R., Carter, J., 1979. Approximate algorithms to derive exact solutions to systems of linear equations. In: Symbolic and Algebraic Computation (EUROSAM'79, Marseille). In: LNCS, vol. 72. Springer, Berlin, pp. 65–73.
Moore, R., 1966. Interval Analysis. Prentice Hall, Englewood Cliffs, NJ.
Pan, V., 1984. How to multiply matrices faster. In: Lect. Notes in Math., vol. 179. Springer.
Schönhage, A., 2000. Variations on computing reciprocals of power series. Information Processing Letters 74, 41–46.
Schönhage, A., Strassen, V., 1971. Schnelle Multiplikation grosser Zahlen. Computing 7, 281–292.
Schulz, G., 1933. Iterative Berechnung der reziproken Matrix. Zeitschrift für Angewandte Mathematik und Mechanik 13, 57–59.
Strassen, V., 1969. Gaussian elimination is not optimal. Numerische Mathematik 13, 352–356.
Strassen, V., 1973. Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. Numerische Mathematik 20, 238–251.
Toom, A., 1963. The complexity of a scheme of functional elements realizing the multiplication of integers. Soviet Mathematics 4 (2), 714–716.
van der Hoeven, J., 1997. Lazy multiplication of formal power series. In: Küchlin, W.W. (Ed.), Proc. ISSAC'97. Maui, Hawaii, pp. 17–20.
van der Hoeven, J., 2002. Relax, but don't be too lazy. Journal of Symbolic Computation 34, 479–542.

van der Hoeven, J., 2004. The truncated Fourier transform and applications. In: Gutierrez, J. (Ed.), Proc. ISSAC 2004, July 4–7. Univ. of Cantabria, Santander, Spain, pp. 290–296.
van der Hoeven, J., 2005. Notes on the Truncated Fourier Transform. Tech. Rep. 2005-5. Université Paris-Sud, Orsay, France.
van der Hoeven, J., 2006. Newton's method and FFT trading. Tech. Rep. 2006-17. Univ. Paris-Sud, http://www.texmacs.org/joris/fnewton/fnewton-abs.html.
van der Hoeven, J., 2007a. New algorithms for relaxed multiplication. Journal of Symbolic Computation 42 (8), 792–802.
van der Hoeven, J., 2007b. On effective analytic continuation. Mathematics in Computer Science 1 (1), 111–175.
van der Hoeven, J., et al. 2002. Mathemagix. http://www.mathemagix.org.