Newton's method and FFT trading

Joris van der Hoeven

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

November 22, 2009

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 for multiplying 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 on the order r of the equation, which is exponential for the original method [vdH02a] and quadratic for a recent improvement [BCO+07]. In this paper, we present a technique to further improve 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.

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 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 $g_0 = 0$. 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

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

[*] This work has been supported by the ANR Gecko and ANR-09-JCJC-0098-01 MaGiX projects. The work has also been supported by the Digiteo 2009-36HD grant and Région Ile-de-France.

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 $\mathcal{C}[[z]][F] = \mathcal{C}[[z]][F_1, \ldots, F_r]$, 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-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 $\mathsf{M}(n) = O(n \log n)$ operations over $\mathcal{C}$, using the fast Fourier transform [CT65]. In general, such roots of unity can be added artificially [SS71, CK91, vdH02a] 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. More details about evaluation-interpolation 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 $\omega < 2.376$ [Str69, Pan84, CW87]. We will denote by $\mathsf{MM}(n, r)$ the cost of multiplying two polynomials of degrees $< n$ with coefficients in $\mathcal{M}_r(\mathcal{C})$. When using evaluation-interpolation in the standard FFT model, 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 [vdH02a]. Recently [BCO+07], this dependence in 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 $O(\mathsf{MM}(n, r))$. 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+07] 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, vdH02a], 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 $f_0, \ldots, f_i$ and $g_0, \ldots, g_i$ are known ($i = 0, \ldots, n - 1$). In [vdH97, vdH02a], we proved that $\mathsf{R}(n) = O(\mathsf{M}(n) \log n)$. In the standard FFT model, this bound was further reduced in [vdH07a] to $\mathsf{R}(n) = O(\mathsf{M}(n)\, \mathrm{e}^{2\sqrt{\log 2 \log \log n}})$. 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 [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 $\mathsf{L}(n, r) = O(\mathsf{MM}(n, r))$ 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 $m = n/r^{\alpha}$. 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 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 $\mathsf{D}(n) \sim 5/3\ \mathsf{M}(n)$, which improves on the best previously known bound $\mathsf{D}(n)\sim 25/12\, \mathsf{M}(n)$ [HZ04]. Notice that $5/3\, \mathsf{M}(n)$ should be read $(5/3)\, \mathsf{M}(n)$ and likewise for other fractions of this kind. Division with remainder of a polynomial of degree $2\ n$ by a polynomial of degree n can be done in time $\mathsf{DR}(n) \sim 2\ \mathsf{M}(n)$; the best previously known bound was $\mathsf{DR}(n) \sim 17/6\ \mathsf{M}(n)$ (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 $\mathsf{E}(n)\sim 7/3\, \mathsf{M}(n)$ instead of $\mathsf{E}(n)\sim 17/6\, \mathsf{M}(n)$ [Ber00].

In two recent papers [Har09a, Har09b] David Harvey has further improved 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 [Har09a] to get better bounds $\mathsf{I}(n) \sim 13/9\ \mathsf{M}(n)$ and $\mathsf{S}(n) \sim 4/3\ \mathsf{M}(n)$ for power series inversion and square roots. In [Har09b], the complexity for exponentiation has been further improved to $\mathsf{E}(n)\sim 13/6\, \mathsf{M}(n)$. 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 to [Ber00] (see also remark 14 below) for a variant and further applications of this technique (called FFT caching by the author). The central idea behind [vdH07a] 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 [vdH02a]. In particular, certified integration of dynamical systems at high precision is a topic which currently interests us [Moo66, Loh88, MB96, Loh01, MB04, vdH07b].

Remark 1. This paper is a thoroughly reworked version of an earlier preprint [vdH06], integrating several helpful remarks by two of the referees and David Harvey.

Operation	Previous bound	This paper	[Har09a, Har09b]
Inversion	$\sim 3/2\,\mathsf{M}(n)$		$\sim 13/9\,\mathsf{M}(n)$
Division	$\sim 25/12\,\mathsf{M}(n)$	$\sim 5/3\,\mathsf{M}(n)$	
Division+Remainder	$\sim 17/6\,\mathsf{M}(n)$	$\sim 2\,\mathsf{M}(n)$	
Square root	$\sim 11/6\,\mathsf{M}(n)$		$\sim 4/3\,\mathsf{M}(n)$
Exponentiation	$\sim 17/6\,\mathsf{M}(n)$	$\sim 7/3\,\mathsf{M}(n)$	$\sim 13/6\,\mathsf{M}(n)$

Table 1. Table with the best asymptotic time complexities of several operations on power series with respect to the cost $\mathsf{M}(n)$ 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.

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 two, then the fast Fourier transform [CT65] allows us to perform the transformation FFT_ω and its inverse

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

using $\mathsf{F}(n) = O(n \log n)$ 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 two, this takes $3\,\mathsf{F}(n) + 2\,n = O(n \log n)$ 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 $\mathsf{N}(n)$ points consists of two computable $\mathcal{C}$-linear mappings

$$\begin{array}{ccc} \mathcal{C}[z]_{;n} & \overset{\mathrm{Eval}}{\longmapsto} & \mathcal{C}^{\mathsf{N}(n)} \\ \mathcal{C}^{\mathsf{N}(n)} & \overset{\mathrm{Eval}^{-1}}{\longmapsto} & \mathcal{C}[z]_{;n} \end{array}$$

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

An *evaluation-interpolation model* is a recipe which associates an evaluation-interpolation scheme to 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 $\mathsf{M}(n)$ of multiplying two polynomials of degrees $< n$ is given by

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

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 $\mathsf{MM}(n,r)$ of multiplying two $r\times r$ matrices whose entries are polynomials of degrees $<n$ is given by

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

Notice also that a matrix-vector product takes a time

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

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

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

In fact, if $n/2<m\leqslant n$, then the truncated Fourier transform [vdH04, vdH05] still provides us with an evaluation-interpolation scheme with $\mathsf{E}(n)\sim\mathsf{F}(n)\,m/n$ and $\mathsf{N}(m)=m$. 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 [SS71, CK91]. 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 $(2\,n_1)$-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 $P\,Q = \iota^{-1}(\text{FFT}^{-1}(\text{FFT}(\iota(P))\ \text{FFT}(\iota(Q))))$ 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

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

We will call this model the *synthetic FFT model*. Using similar ideas to those in [CK91], 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 [MB72, Str73, BM74]

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

If it is possible to take points in a geometric progression, then one even has [BS05]

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

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

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

This yields an evaluation-interpolation model with

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

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 $2\,b + O(\log n)$. 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 $h = f\,g$ be the product of two power series $f, g \in \mathcal{C}[[z]]$. In order to efficiently compute only a part h_i, ..., h_{i+n-1} of h, a useful tool is the so called "middle product" [HQZ04]. Let $R, Q \in \mathcal{C}[z]$ be two polynomials with $\deg R < 2\,n$ and $\deg Q \leqslant n$. Then we define their *middle product* $R \ltimes_n Q$ (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, $T_Q = M_{\operatorname{Rev}(Q)}^{\top}$, where $M^{\top}$ denotes the transpose of a matrix M and $\operatorname{Rev}(Q)=\operatorname{Rev}_n(Q)=Q_0\,z^n+\cdots+Q_n$.

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 $<2\,n$, restricted to polynomials of degree $<n$ corresponds to an $\mathsf{N}(2\,n)\times n$ matrix E:

$$\operatorname{Eval}(P)=E\,V_P.$$

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

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

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

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

Since this equality holds for all P, it follows that

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

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 $\operatorname{tEval}^{-1}$ of these algorithms in time $\mathsf{E}(2\,n)+O(\mathsf{N}(2\,n))$ [Bor56, Ber]. We may thus compute the middle product using

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

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

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

One may also use the alternative formula [Har09a]

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

which is based on the fact that standard FFT multiplication of R and Q really computes the exact product of $R\,Q$ modulo $z^{2n}-1$. Writing $P=P_0+P_1\,z^n+P_2\,z^{2n}=R\,Q$ 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 notations:

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

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 $\bar{F}_i\in\mathcal{C}[z]$, $\deg\bar{F}_i<m$:

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

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

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

Then we have

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

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:

$$\bar{P}=\operatorname{Ext}(\bar{A})\boxed{\ltimes}\bar{F}=\sum_{i_1+i_2=i}\operatorname{Ext}(\bar{A})_{i_1}\ltimes\bar{F}_{i_2}.\qquad(11)$$

These formulas are illustrated in Figure 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 $<2\,m$. Then we may replace the polynomial representations of the blockwise coefficients $\bar{A}_i$ and $\bar{F}_i$ by their transforms

$$\begin{aligned} \bar{A}_i^* &= \operatorname{tEval}^{-1}(\bar{A}_{i-1}+\bar{A}_i\,z^m) &(12)\\ \bar{F}_i^* &= \operatorname{Eval}(\operatorname{Rev}(\bar{F}_i)), &(13) \end{aligned}$$

compute convolution products in the transformed model

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

and apply (8) in order to recover the coefficients

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

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

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

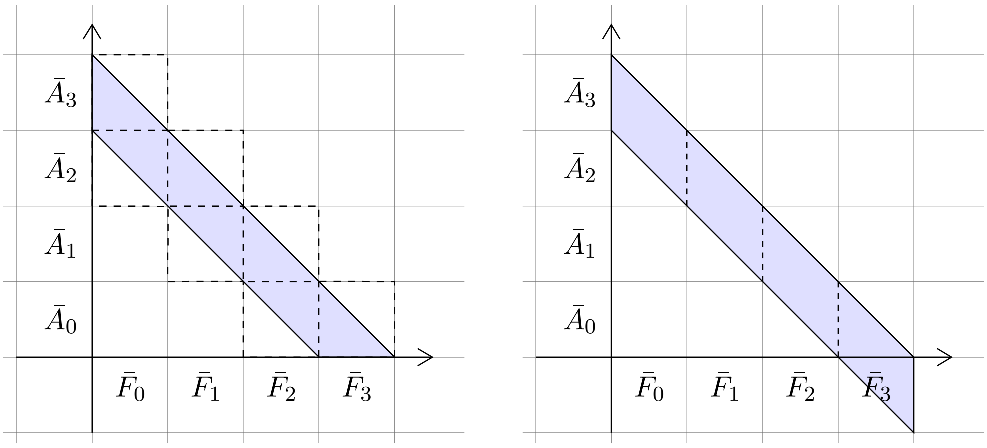

Figure 1. Two ways to compute the coefficient $\bar{P}_3$, with $P = A\,F$. At the left-hand side, we use

$$\bar{P}_3 = (\bar{A}_2\,\bar{F}_0 + \bar{A}_1\,\bar{F}_1 + \bar{A}_0\,\bar{F}_2)\operatorname{div} z^m + (\bar{A}_3\,\bar{F}_0 + \bar{A}_2\,\bar{F}_1 + \bar{A}_1\,\bar{F}_2 + \bar{A}_0\,\bar{F}_3)\bmod z^m.$$

At the right-hand side, we use middle products:

$$\bar{P}_3 = (\bar{A}_2 + \bar{A}_3\,z^m) \ltimes \bar{F}_0 + (\bar{A}_1 + \bar{A}_2\,z^m) \ltimes \bar{F}_1 + (\bar{A}_0 + \bar{A}_1\,z^m) \ltimes \bar{F}_2 + (\bar{A}_{-1} + \bar{A}_0\,z^m) \ltimes \bar{F}_3.$$

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 $n = m\,k$ coefficients of $F = \frac{B}{A}$. Denoting $U = \frac{1}{A}$, we first compute $\bar{U}_0$ using a classical Newton iteration [BK78, Sch00]. 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

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

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 $< 2\,m$ yields the following algorithm:

Algorithm `divide`(B, A, m, k)

INPUT: two truncated power series $A, B \in \mathcal{C}[[z]]$ at order $k\,m > 0$, 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 [vdH02a, vdH07a]. 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 equation (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 $\mathsf{E}(n)/n$ and $\mathsf{N}(n)/n$ are increasing functions in n and $\mathsf{N}(n) = o(\mathsf{E}(n))$. 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 4. *The quotient of two power series in $\mathcal{C}[[z]]$ can be computed at order n in time*

$$\mathsf{D}(n) \lesssim 5 \, \mathsf{E}(2n).$$

Proof. Let us analyze the complexity of `divide`. The precomputation of $\bar{U}_0$ can be done in time $6 \, \mathsf{E}(2m) + 2 \, \mathsf{N}(2m)$ when using a fast algorithm [HQZ04] based on Newton's method. The main loop accounts for

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

- k scalar multiplications in the transformed model of cost $O(k)\,\mathsf{N}(2\,m)$.

Adding up these costs, the complete division takes a time

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

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

Remark 5. In practice, the number k should be chosen not too large, so as to keep $\mathsf{R}(k)\,\mathsf{N}(2\,m)$ reasonably small. According to (16), we need $5\,k+7<25/4\,k$ in order to beat the best previously known division algorithm [HZ04], which happens for $k>28/5$.

Remark 6. 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 $n=k\,m$.

For simplicity, we will assume the standard FFT model and naive inner multiplication. The $2\,n$ 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 $2\,n$ is replaced by k FFTs of size $2\,m$, thereby saving approximately $n\log_2 k$ multiplications. For $k=4$, the blockwise algorithm therefore saves $6\,n-3\,n=3\,n$ multiplications. For $k=8$, we save $9\,n-7\,n=2\,n$ multiplications. For $k=16$, we perform $15\,n-12\,n=3\,n$ 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” $20\,n-15\,n$ 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 7. For $i=k,\ldots,2\,k-2$, the coefficients

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

can be computed using $k-1$ additional transforms of cost $(k-1)\,\mathsf{E}(2\,m)$ and $\mathsf{M}(k)\,\mathsf{N}(2\,m)$ additional inner multiplications. This implies that the quotient and the remainder of a division of a polynomial V of degree $<2\,n$ by a polynomial U of degree $n-1$ can be computed in time $\lesssim 6\,\mathsf{E}(2\,n)$. 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 8. 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

$$\delta M = A\, M \tag{17}$$

admits a unique solution $M \in \mathcal{M}_r(\mathcal{C}[[z]])$ with $M_0 = \mathrm{Id}_r$. The main idea of [BCO+07] 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

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

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 (18) and the usual Newton iteration [Sch33, MC79]

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

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

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 as (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

$$\delta F = A\, F \tag{21}$$

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 equations (20) and (22) may be computed efficiently using the middle product algorithm.

4.2. Blockwise resolution

The block notations $\bar{A}$ and $\bar{F}$ from section 3.1 naturally generalize to series of matrices and series of vectors. The derivation δ operates in a blockwise fashion:

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

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):

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

Assume that we have fixed an evaluation-interpolation scheme at degree $<2\,m$. 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 evaluation-interpolation:

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

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 9. In the above algorithm, the coefficients of $\bar{P}$ are computed in a naive manner. In a similar way as in Remark 3, we may use a relaxed algorithm instead. More precisely, the equation (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 $\mathsf{L}(n, r, s)$ the time complexity in order to compute the truncated solution $F_{;n}$ to (21) at order n.

Theorem 10. *Consider the differential equation* (21)*, where A has s non-zero entries. Assume that $r^{\omega-1} = o(n)$ and $\mathsf{R}(r^{\omega-1})/r^{\omega-1} = o(\mathsf{E}(2\,n)/\mathsf{N}(2\,n))$. Then there exists an algorithm which computes the truncated solution $F_{;n}$ to* (21) *at order n in time*

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

Proof. In our algorithm, let ϕ_n be a function which increases towards infinity, such that $\phi_n\,r^{\omega-1} = o(n)$ and $\mathsf{R}(\phi_n\,r^{\omega-1})/(\phi_n\,r^{\omega-1}) = o(\mathsf{E}(2\,n)/\mathsf{N}(2\,n))$. We take $k = \lfloor \phi_n\,r^{\omega-1} \rfloor$ and $m = \lceil n/k \rceil$, so that $k\,m - n \leqslant k = o(n)$. 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

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

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

   $$O(\mathsf{E}(2\,m)\,r^2) = o(\mathsf{E}(2\,n)\,r).$$

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

   $$\lesssim \mathsf{E}(2\,m)\,k\,(s + 2\,r) \leqslant \mathsf{E}(2\,n)\,(s + 2\,r).$$

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

   $$O(k^2\,\mathsf{N}(2\,m)\,s) = O(\mathsf{N}(2\,n)\,k\,s).$$

   Using a relaxed multiplication algorithm, this cost further reduces to

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

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

   $$\lesssim 4\,\mathsf{E}(2\,m)\,k\,r \leqslant 4\,\mathsf{E}(2\,n)\,r$$

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

   $$O(k\,\mathsf{N}(2\,m)\,r^2) = O(\mathsf{N}(2\,n)\,r^2).$$

Adding up the different contributions, we obtain the bound

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

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

Corollary 11. *In the standard FFT model, and under the assumption $r = o(\log n)$, we have*

$$\begin{aligned} \mathsf{L}(n, r, s) &\lesssim (s/3 + 2\,r)\,\mathsf{M}(n) \\ \mathsf{L}(n, r) &\lesssim 7/3\,r\,\mathsf{M}(n). \end{aligned}$$

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

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

PROOF. In the standard FFT model, we have $\mathsf{E}(2\,n) \sim \mathsf{M}(n)/3$, $\mathsf{E}(2\,n)/\mathsf{N}(2\,n) = O(\log n)$ and $\mathsf{R}(r^{\omega-1})/r^{\omega-1} = O(\mathrm{e}^{2\sqrt{\log 2 \log \log r}} \log r)$. 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 $\mathsf{E}(2\,n) \sim \mathsf{M}(n)/3$, $\mathsf{E}(2\,n)/\mathsf{N}(2\,n) = O(\log \log n)$ and $\mathsf{R}(r^{\omega-1})/r^{\omega-1} = O((\log r)^2 \log \log r)$. 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 12. In practice, one should choose ϕ_n just sufficiently large such 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 13. The use of middle products was needed in order to achieve the factor $s/3 + 2\,r$ in Corollary 11. As explained in Remark 2, using a more straightforward multiplication algorithm seems to require one additional transform. This leads to the factor $s/3 + 7\,r/3$.

REMARK 14. Corollary 11 applies in particular to the exponentiation $f = \mathrm{e}^g$ of a power series g. We obtain an algorithm of time complexity $\mathsf{L}(n,1) \lesssim 7/3\, \mathsf{M}(n)$, which yields an improvement over [Ber00, HZ04]. Notice that FFT trading is a variant of Newton caching in [Ber00], 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 15. With minor changes, the algorithm can be adapted in order to compute the unique solution of the matrix differential equation $\delta M = M F$ with $M_0 = \mathrm{Id}_r$. The unique solution M corresponds to a fundamental system of solutions to (21). A similar complexity analysis to the one in the proof of Theorem 10 yields the bound

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

Under the additional hypotheses of the corollary, we thus get

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

REMARK 16. In the standard FFT model, the conditions of Theorem 10 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

$$\mathsf{L}(n,r) = n\, r\, O(\log n + r).$$

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

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

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

$$\mathsf{L}(n,r,r^2) = O(n\, r^2 \log n),$$

whereas the relaxed approach yields the bound

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

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

Remark 17. In the case when $\mathcal{C}$ does not admit many 2^p-th roots of unity, we have the choice between the synthetic FFT model and multipoint evaluation-interpolation. In the synthetic FFT model, the almost optimal bounds from Corollary 11 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 multipoint evaluation-interpolation with $\mathsf{E}(2\,n)\sim c\,\mathsf{M}(n)$ and $\mathsf{N}(n)=n$ for some constant $c>1$. In a similar way as in Corollary 11, one obtains the bound

$$\mathsf{L}(n,r,s)\lesssim c\,\mathsf{M}(n)\,(s/3+2\,r)$$

under the assumption $r=o(\log n\log\log n)$, since $\mathsf{E}(2\,n)/\mathsf{N}(2\,n)=O(\log n\log\log n)$. 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 $\mathsf{R}(n)=O(\mathsf{M}(n)\log n)$ in this context. Therefore, we potentially gain a factor $O(\log n)$ compared to the relaxed approach.

Remark 18. One may wonder whether the technique of FFT trading is useful in asymptotically less efficient models such as the Karatsuba model. Recall however that $\mathsf{R}(n)=O(\mathsf{M}(n))$ in any model with $\mathsf{M}(n)\asymp n^\alpha$ for $\alpha>1$. The Karatsuba model is even essentially relaxed, in the sense that $\mathsf{R}(n)=\mathsf{M}(n)$. 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 $\mathsf{E}(n)\sim\mathsf{N}(n)$ in such models, so the second term in (24) can never be neglected with respect to the first term.

Remark 19. 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 $\mathsf{T}(n,r)$ 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 (18) and (19) give rise to $1+2+2=5$ matrix multiplications. This yields

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

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

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

Using a trick from [Sch00], one may even prove that

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

which yields

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

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 [BK78, vdH02a, BCO+07]. 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 will 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 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* of the dag and the total number t of nodes the *total size* of the dag. Using evaluation-interpolation, one may compute $P(F)_{;n}$ in terms of $F_{;n}$ in time $\lesssim 3\, s\, \mathsf{E}(2\, n) + t\, \mathsf{N}(2\, n)$.

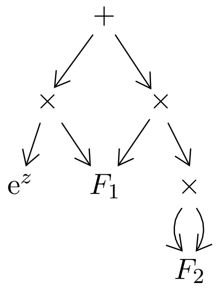

Figure 2. Example of a polynomial expression in $\mathcal{C}[[z]][F_1, F_2]$, represented by a dag. In this particular example, the multiplicative size of the polynomial is $s = 7/3$ (since $s_1 = 4$ and $s_2 = 3$) and its total size 7.

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 $\lesssim (2\, r + 1)\, s\, \mathsf{M}(m) + O(r\, t\, m)$ using the standard differentiation rules. For $n \geqslant m$, we have

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

so that

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

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

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

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

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

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 $\bar{S}_i Z^i$ by

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

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

Theorem 20. *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 $\mathsf{R}(r^{\omega-1})/r^{\omega-1} = o(\mathsf{E}(2n)/\mathsf{N}(2n))$. Then there exists an algorithm which computes $F_{;n}$ in time*

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

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

Corollary 21. *In the standard FFT model, and assuming that $r = o(\log n)$, we have*

$$\mathsf{A}(n, r, s, t) \;\lesssim\; \mathsf{M}(n)\,(2s + 4r/3) + O(n\,t).$$

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

Remark 22. In the case when most multiplications in $P(F)$ only depend linearly on F, it is possible to adapt a similar technique as in the previous section and perform these multiplications using the middle product. This allows for a reduction of the factor $2\,s$ to something between s and $2\,s$.

Remark 23. When solving (26) using Newton's method [BCO+07] with the optimization from [Sch00], one obtains the bound

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

However, the factor $2\ r\ s$ is quite pessimistic. For instance, if the expressions $P_1(F)$, ..., $P_r(F)$ do not share any common subexpressions, then we may use automatic differentiation [BS83] 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 $s_2' = 3\,s_2$, whence $s' \leqslant 4\,s$ and

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

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 to remove this problem is to use the TFT (truncated Fourier transform) as detailed in [vdH04, vdH05]. Another way to smooth the complexity is to cut P and Q in 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 $2\,m$ for evaluation-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 $i < 2\,k-1$ and return $\bar{R}_0 + \cdots + \bar{R}_{2k-2}\,z^{(2k-2)m}$.

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

$$4\,k\,\mathsf{E}(2\,m) + 2\,k^2\,m \sim 4/3\,C\,n \log n + 2\,k\,n$$

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

$$3\,k\,\mathsf{E}(2\,m) + k^2\,m \sim C\,n \log n + k\,n.$$

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

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

$$C\,n \log n + 2\,n$$

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 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 even larger dimensions, one may use [LS03] or [vdH02a, Section 6.3.5]. The idea is again to cut P in blocks

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

where $k = \lceil n/m \rceil$ is small (and m preferably a power of two). Each block is then transformed using 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

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

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

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

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

$$\begin{aligned} M_{k,d} &\sim \frac{1}{(d-1)!}\, k^{d-1} \\ N_{k,d} &\sim \frac{1}{(2\,d)!}\, k^{2d} \end{aligned}$$

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

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

i.e. if

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

In Table 2, we have shown the values of $N_{k,d}/(d\, k^d)$ 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 $k \leqslant d$.

d k	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

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

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 [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 even/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 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 $\mathsf{R}(n) = O(\mathsf{M}(n)\, \mathrm{e}^{2\sqrt{\log 2 \log \log n}})$ in the standard FFT model and $\mathsf{R}(n) = O(\mathsf{M}(n) \log n)$ 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 interesting for quite large values of n. As pointed out in remark 18, FFT trading looses 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 will be available in the Mathemagix system [vdH+02b]. Notice also that Newton iterations are better suited to parallel computing than is relaxed evaluation.

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	$\sim 15/4\, \mathsf{M}(n)\, r^2$	$\sim \mathsf{M}(n)\, (31/12\, r^2 + s/2 + 2/3\, r)$
New	$\sim 7/3\, \mathsf{M}(n)\, r^2$	$\sim \mathsf{M}(n)\, (s/3 + 2\, r)$

Table 3. 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 $r = o(\log \log n)$ 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	$\sim \mathsf{M}(n)\, (2\, s\, r + 2\, s + 13/6\, r^2 + 4/3\, r) + O(t\, r\, n)$
New	$\sim \mathsf{M}(n)\, (2\, s + 4/3\, r) + O(t\, n)$

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 $\sim \mathsf{M}(n)\, (2\, s + 4/3\, r) + O(t\, n \log n)$, under the assumption $r = o(\log \log n)$.

One 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 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

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

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

Bibliography

[BCO+07] A. Bostan, F. Chyzak, F. Ollivier, B. Salvy, É. Schost, and A. Sedoglavic. Fast computation of power series solutions of systems of differential equation, 2007. New Orleans, January 2007.

[Ber] D. Bernstein. The transposition principle. `http://cr.yp.to/transposition.html`.

[Ber00] D. Bernstein. Removing redundancy in high precision Newton iteration. Available from `http://cr.yp.to/fastnewton.html`, 2000.

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

[BM74] A. Borodin and R.T. Moenck. Fast modular transforms. *Journal of Computer and System Sciences*, 8:366–386, 1974.

[Bor56] J. L. Bordewijk. Inter-reciprocity applied to electrical networks. *Applied Scientific Research B: Electrophysics, Acoustics, Optics, Mathematical Methods*, 6:1–74, 1956.

[BS83] Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983.

[BS05] A. Bostan and É. Schost. Polynomial evaluation and interpolation on special sets of points. *Journal of Complexity*, 21(4):420–446, August 2005. Festschrift for the 70th Birthday of Arnold Schönhage.

[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.

[CW87] D. Coppersmith and S. Winograd. 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.

[Har09a] David Harvey. Faster algorithms for the square root and reciprocal of power series, 2009. `http://arxiv.org/abs/0910.1926`.

[Har09b] David Harvey. Faster exponentials of power series, 2009. `http://arxiv.org/abs/0911.3110`.

[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.

[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`.

[HZ04] G. Hanrot and P. Zimmermann. Newton iteration revisited. `http://www.loria.fr/ zimmerma/papers/fastnewton.ps.gz`, 2004.

[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 on enclosure methods*, pages 201–217, Wien, New York, 2001. Springer.

[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.

[MB72] R.T. Moenck and A. Borodin. Fast modular transforms via division. In *Thirteenth annual IEEE symposium on switching and automata theory*, pages 90–96, Univ. Maryland, College Park, Md., 1972.

[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*, 7:281–292, 1971.

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

[Str73] V. Strassen. Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. *Numer. Math.*, 20:238–251, 1973.

[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.

[vdH02a] J. van der Hoeven. Relax, but don't be too lazy. *JSC*, 34:479–542, 2002.

[vdH+02b] J. van der Hoeven et al. Mathemagix, 2002. `http://www.mathemagix.org`.

[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.

[vdH06] J. van der Hoeven. Newton's method and FFT trading. Technical Report 2006-17, Univ. Paris-Sud, 2006. `http://www.texmacs.org/joris/fnewton/fnewton-abs.html`.

[vdH07a] J. van der Hoeven. New algorithms for relaxed multiplication. *JSC*, 42(8):792–802, 2007.

[vdH07b] J. van der Hoeven. On effective analytic continuation. *MCS*, 1(1):111–175, 2007.