A Note on the Space Complexity of Fast D-Finite Function Evaluation

Marc Mezzarobba

A NOTE ON THE SPACE COMPLEXITY OF FAST D-FINITE FUNCTION EVALUATION

MARC MEZZAROBBA

Abstract. We state and analyze a generalization of the "truncation trick" suggested by Gourdon and Sebah to improve the performance of power series evaluation by binary splitting. It follows from our analysis that the values of D-finite functions (i.e., functions described as solutions of linear differential equations with polynomial coefficients) may be computed with error bounded by 2^{-p} in time $\mathrm{O}(p(\lg p)^{3+o(1)})$ and space $\mathrm{O}(p)$. The standard fast algorithm for this task, due to Chudnovsky and Chudnovsky, achieves the same time complexity bound but requires $\Theta(p \lg p)$ bits of memory.

1. Introduction

Binary splitting is a well-known and widely applicable technique for the fast multiple precision numerical evaluation of rational series. For any series $\sum_n s_n$ with $\limsup_n |s_n|^{1/n} < 1$ whose terms s_n obey a linear recurrence relation with polynomial coefficients, e.g.,

$$\ln 2 = \sum_{n=0}^{\infty} s_n, \qquad s_n = \frac{1}{(n+1)2^{n+1}}, \qquad 2(n+2)s_{n+1} - (n+1)s_n = 0,$$

the binary splitting algorithm allows one to compute the partial sum $\sum_{n=0}^{N-1} s_n$ in $\mathrm{O}(\mathrm{M}(N(\lg N)^2))$ bit operations [5, 3]. Here $\mathrm{M}(n)$ stands for the complexity of multiple precision integer multiplication, and lg denotes the binary logarithm. As $N = \mathrm{O}(p)$ terms of the series are enough to make the approximation error less than 2^{-p}, the complexity of the algorithm is softly linear in the precision p, assuming $\mathrm{M}(n) = \mathrm{O}(n(\lg n)^{\mathrm{O}(1)})$.

Methods based on binary splitting tend to be favored in practice even in cases when asymptotically faster algorithms (typically AGM iterations [2]) would apply. One high-profile example is the computation of billions of digits of classical constants such as π, $\zeta(3)$ or γ. Basically all record computation in recent years were achieved by evaluating suitable series using variants of binary splitting [9, 28].

A drawback of the classical binary splitting algorithm, both from the complexity point of view and in practice, is its comparatively large memory usage. Indeed, the algorithm amounts to the computation of a product tree of matrices derived from the recurrence—see Sect. 3 below for details. The intermediate results are matrices of rational numbers whose bit sizes roughly double from one level to the next. Near the root, their sizes can (and in general do) reach $\Theta(p \lg p)$, even though the output has size $\Theta(p)$.

However, the space complexity can be lowered to $\mathrm{O}(p)$ using a slight variation of the classical algorithm. The basic idea is to truncate the intermediate results to a precision $\mathrm{O}(p)$ when they start taking up more space than the final result.

Of course, these truncations introduce errors. To make the trick into a genuine algorithm, we need to analyze the errors, add a suitable number of "guard digits" at each step and check that the space and time complexities of the resulting process stay within the expected bounds.

The opportunity to improve the practical behavior of binary splitting using truncations has been noticed by authors of implementations on several occasions over the last decade or so. Gourdon and Sebah [10] describe truncation as a "crucial" optimization. Besides the expected drop of memory usage, they report running time improvements by an "appreciable" constant factor. Cheng et al. [4] compare truncation with alternative (less widely applicable but sometimes more efficient) approaches. Most recently, Kreckel [14] explicitly asks how to make sure that the new roundoff errors do not affect the correctness of the result.

Indeed, the above-mentioned error analysis did not appear in the literature until very recently. An article by Yakhontov [26, 27] now provides the required bounds in the case of the generalized hypergeometric series ${}_pF_q$, which covers all examples where the truncation trick had been used before. But the applicability of the method is actually much wider.

The purpose of this note is to present a more general and arguably simpler analysis. Our version is more general in two main respects. First, besides hypergeometric series, it applies to the solutions of linear ordinary differential equations with rational coefficients, also known as *D-finite* (or holonomic) series [21]. D-finite series are exactly those whose coefficients obey a linear recurrence relation with rational coefficients, while hypergeometric series correspond to recurrences of the first order. Second, we take into account the coefficient size of the recurrence that generates the series to be computed. Allowing the size of the coefficients to vary with the target precision p makes it possible to use the modified binary splitting procedure as part of the "bit burst" algorithm [5] to handle evaluations at general real or complex points approximated by rationals of size $\Theta(p)$.

Additionally, our analysis readily adapts to other applications of binary splitting. The simplicity and generality of the proof are direct consequences of viewing the algorithm primarily as the computation of a product tree. See Gosper [8] and Bernstein [1, §12–16] for further comments on this point of view.

The remainder of this note is organized as follows. Section 2 contains some notations and assumptions. In Sect. 3, we recall the standard binary splitting algorithm, which will serve as a subroutine in the linear-space version. Then, in Sect. 4, we state and analyze the "truncated" variant that achieves the linear space complexity for general D-finite functions. Finally, Sect. 5 offers a few comments on other variants of the binary splitting method and possible extensions of the analysis.

2. Setting

The performance of the binary splitting algorithm crucially depends on that of integer multiplication. Following common usage, we denote by $\mathrm{M}(n)$ a bound on the time needed to multiply two integers of at most n bits. Currently the best theoretical bound [7] is $\mathrm{M}(n) = \mathrm{O}(n(\lg n)\exp \mathrm{O}(\lg^* n))$, where $\lg^* n = \min\{k \mid \lg^{\circ k} n \leqslant 1\}$. In practice, implementations such as GMP [11] use variants of the Schönhage-Strassen algorithm of complexity $\mathrm{O}(n(\lg n)(\lg\lg n))$. We make the usual assumption [25] that the function $n \mapsto \mathrm{M}(n)/n$ is nondecreasing. It follows that $\mathrm{M}(n) +$

$\mathrm{M}(m) \leqslant \mathrm{M}(n+m)$. We also assume that the *space* complexity of integer multiplication is linear, which is true for the standard algorithms.

Write $\mathrm{K} = \mathrm{Q}(i)$, and define the *bit size* of a number $(x+iy)/w \in \mathrm{K}$ (where $w, x, y \in \mathrm{Z}$) as $\lceil \lg w \rceil + \lceil \lg x \rceil + \lceil \lg y \rceil + 1$. Consider a linear differential equation with coefficients in $\mathrm{K}(z)$. It will prove convenient to clear all denominators (both polynomial and integer) and multiply the equation by a power of z to write it as

$$\left(a_r(z)\left(z\frac{\mathrm{d}}{\mathrm{d}z}\right)^r + \cdots + a_1(z)z\frac{\mathrm{d}}{\mathrm{d}z} + a_0(z)\right)\cdot y(z) = 0, \qquad a_k \in \mathrm{Z}[i][z]. \tag{1}$$

Let $s = \max_k \deg a_k$, and let h_1 denote the maximum bit size of the coefficients of the a_k. Although our complexity estimates depend on r and h_1, we do not consider more general dependencies on the equation. Thus, the a_k are assumed to vary only in ways that can be described in terms of these two parameters. Specifically, we assume that $s = \mathrm{O}(1)$ and that the coefficients of $a_k(z)/a_r(0)$ are all restricted to some bounded domain.

We also assume that 0 is an ordinary (i.e. nonsingular) point of (1). This implies that $a_r(0) \neq 0$ and $s \geqslant r$. The case of *regular singular* points (those for which we still have $a_r(0) \neq 0$ but possibly $s < r$ [13, Chap. 9]) is actually similar [23, 17]; we focus on ordinary points to avoid cumbersome notations.

Let $\rho = \min\{|z| : a_r(z) = 0\} \in (0, \infty]$. Then any formal series solution $y(z) = \sum_{n\geqslant 0} y_n z^n$ of (1) converges on the disk $|z| < \rho$. We select a particular solution (say, by specifying initial values $y(0), \ldots, y^{(r-1)}(0)$ in some fixed, bounded domain), and an evaluation point $\zeta \in \mathrm{K}$ with $|\zeta| < \rho$. Let h_2 denote the bit size of ζ, and let $h = h_1 + h_2$. Again, h_2 is allowed to grow to infinity, but we assume that $|\zeta|$ is bounded away from ρ.

Given $p \geqslant 0$, our goal is to compute a complex number $\omega \in \mathrm{K}$ such that $|\omega - y(\zeta)| \leqslant 2^{-p}$. By a classical argument, which can be reconstructed by substituting a series with indeterminate coefficients into (1), the sequence (y_n) obeys a recurrence relation of the form

$$b_0(n)y_{n+r} + b_1(n)y_{n+r-1} + \cdots + b_s(n)y_{n+r-s} = 0, \qquad b_j \in \mathrm{K}\,[n]\,. \tag{2}$$

Writing $a_k(z) = a_{k,0} + a_{k,1}z + \cdots + a_{k,s}z^s$, the b_j are given explicitly by

$$b_j(n) = \sum_{k=0}^{r} a_{k,j}(n+r-j)^k. \tag{3}$$

Based on the matrix form of the recurrence (2), set

$$B(n) = \begin{pmatrix} \zeta C(n) & 0 \\ R & 1 \end{pmatrix} \in \mathrm{K}(n)^{(s+1)\times(s+1)} \tag{4}$$

where

$$C(n) = \begin{pmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ -\frac{b_s(n)}{b_0(n)} & \cdots & \cdots & -\frac{b_1(n)}{b_0(n)} \end{pmatrix}, \qquad R = \Big(\underbrace{0 \quad \ldots \quad 0}_{s-r \text{ zeroes}} \quad 1 \quad \underbrace{0 \quad \ldots \quad 0}_{r-1 \text{ zeroes}} \Big).$$

Let $P(a,b) = B(b-1)\cdots B(a+1)B(a)$ for all $a \leqslant b$. (In particular, $P(a,a)$ is the identity matrix.)

Algorithm 1. BinSplit(a, b)
1 If $b - a \leqslant$ (some threshold)
  2 Return $\hat{B}(b-1)\cdots\hat{B}(a)$ where $\hat{B}$ is defined by (5)
3 else
  4 Return BinSplit$(\lfloor\frac{a+b}{2}\rfloor, b)\cdot$BinSplit$(a, \lfloor\frac{a+b}{2}\rfloor)$

One may check that $b_0(n) \neq 0$ for $n \geqslant 0$, due to the fact that 0 is an ordinary point of (1). Thus the computation of a partial sum $S_N = \sum_{n=0}^{N-1} y_n\zeta^n$ reduces to that of the matrix product $P(0, N)$. Indeed, we have

$$(y_{n+r-s}\zeta^n, \ldots, y_{n+r-1}\zeta^n, S_n)^{\mathrm{T}} = P(0, n)\,(y_{r-s}, \ldots, y_{r-1}, 0)^{\mathrm{T}}$$

where $y_{r-s} = 0, \ldots, y_{-1} = 0, y_0, \ldots, y_{r-1}$ are easily determined from the initial values of the differential equation.

3. Review of the Classical Binary Splitting Algorithm

Since the entries of the matrix $B(n)$ are rational functions of n, the bit size of $P(a, b)$ grows as $\mathrm{O}((b-a)\lg b)$ when $b, (b-a) \to \infty$. This bound is sharp in the sense that it is reached for some (in fact, most) differential equations. Computing $P(a, b)$ as $B(b-1)\cdot[B(b-2)\cdot[\cdots B(a)]]$ then takes time at least quadratic in $b-a$, as can be seen from the combined size of the intermediate results. The term "binary splitting" refers to the technique of reorganizing the product into a *balanced tree* of subproducts, using the relation $P(a, b) = P(m, b)\cdot P(a, m)$ with $m = \lfloor\frac{1}{2}(a+b)\rfloor$, and so on recursively.

A slight complication stems from the fact that removing common divisors between the numerators and denominators of the fractions appearing in the intermediate $P(a, b) \in \mathrm{K}^{r\times r}$ would in general be too expensive. Multiplying the numerators and denominators separately and doing a single final division yields better complexity bounds. Let

$$\hat{B}(n) = b_0(n)\check{\zeta}B(n) \in \mathrm{Z}[i][n]^{(s+1)\times(s+1)}, \quad \zeta = \hat{\zeta}/\check{\zeta} \quad (\hat{\zeta} \in \mathrm{Z}\,[i]\,, \check{\zeta} \in \mathrm{Z}). \tag{5}$$

The entries of $\hat{B}(n)$ are polynomials of degree at most r and bit size $\mathrm{O}(h)$. To compute $P(a, b)$ by binary splitting, we multiply the $\hat{B}(n)$ for $a \leqslant n < b$ using Algorithm 1, and then divide the resulting matrix by its bottom right entry. The general algorithm considered here was first published by Chudnovsky and Chudnovsky [5], with (up to minor details) the analysis summarized in Prop. 1. The idea of binary splitting was known long before [8, 1].

Proposition 1. [5] As $b, N = b-a, h, r \to \infty$ with $r = \mathrm{O}(N)$, Algorithm 1 computes an unreduced fraction equal to $P(a, b)$ in $\mathrm{O}\big(\mathrm{M}\big(N(h+r\lg b)\big)\lg N\big)$ operations, using $\mathrm{O}\big(N(h + r\lg b)\big)$ bits of memory. Assuming $\mathrm{M}(n) = n(\lg n)(\lg\lg n)^{\mathrm{O}(1)}$, both bounds are sharp.

sketch. The bit sizes of the matrices that get multiplied together at any given depth $0 \leqslant \delta < \lceil\lg N\rceil$ in the recursive calls are at most $C2^{-\delta}N(h+d\lg b)$ for some C. Since there are at most 2^δ such products and the multiplication function $\mathrm{M}(\cdot)$ was assumed to be subadditive, the contribution of each level is bounded by $\mathrm{M}(C(b-a)(h + d\lg b))$, whence the total time complexity. See [5, 17] for details. The intermediate results stored or multiplied together at any stage of the computation

are disjoint subproducts of $B(b-1)\cdots B(a)$, and we assumed the space complexity of n-bit integer multiplication to be $\mathrm{O}(n)$, so the space required by the algorithm is linear in the combined size of the $B(n)$. Finally, it is not hard to construct examples of differential equations that reach these bounds. □

Remark 1. The link between our setting and the more common description of the algorithm for hypergeometric series is as follows. In the notation of Haible and Pananikolaou [12] also used in Yakhontov's article, the partial sums of the hypergeometric series are related to its defining parameters a, b, p, q by

$$\begin{pmatrix} \tilde{s}(i+1) \\ S(i) \end{pmatrix} = \begin{pmatrix} \frac{p(i)}{q(i)} & 0 \\ \frac{a(i)}{b(i)}\frac{p(i)}{q(i)} & b(i)q(i) \end{pmatrix} \begin{pmatrix} \tilde{s}(i) \\ S(i-1) \end{pmatrix}, \qquad \tilde{s}(i) = \frac{b(i)}{a(i)} s(i).$$

This equation becomes $\left(\begin{smallmatrix} B(i) \\ T(0,i) \end{smallmatrix}\right) = \left(\begin{smallmatrix} b(i)p(i) & 0 \\ a(i)p(i) & b(i)q(i) \end{smallmatrix}\right)\left(\begin{smallmatrix} B(i-1)P(i-1) \\ T(0,i-1) \end{smallmatrix}\right)$ upon clearing denominators. The standard recursive algorithm for hypergeometric series may be seen an "inlined" computation of the associated product tree. Each recursive step is equivalent to the computation of the matrix product $\left(\begin{smallmatrix} B_rP_r & 0 \\ T_r & B_rQ_r \end{smallmatrix}\right)\left(\begin{smallmatrix} B_lP_l & 0 \\ T_l & B_lQ_l \end{smallmatrix}\right)$.

We return to the evaluation of a D-finite power series within its disk of convergence. From the differential equation (1), suitable initial conditions, the evaluation point ζ and a target precision p, one can *compute* [18] a truncation order N such that $|S_N - y(\zeta)| \leqslant 2^{-p}$ and

$$\begin{cases} N \sim Kp = \big(\lg(|\zeta|/\rho)\big)^{-1} p, & \text{if } \rho < \infty \\ N = \Theta(p/\lg p), & \text{if } \rho = \infty. \end{cases} \tag{6}$$

Combined with these estimates, Proposition 1 implies the following.

Corollary 1. Write $\ell = h + r\lg p$. Under the assumptions of Proposition 1, one can compute $y(\zeta)$ in $\mathrm{O}(\mathrm{M}(\ell p \lg p))$ bit operations, using $\mathrm{O}(\ell p)$ bits of memory. The complexity goes down to $\mathrm{O}(\mathrm{M}(\ell p))$ operations and $\mathrm{O}(\ell p/\lg p)$ bits of memory when $a_r(z)$ is a constant.

This result is the basis of more general evaluation algorithms for D-finite functions [5]. Indeed, binary splitting can be used to compute the required series sums at each step when solving a differential equation of the form (1) by the so-called method of Taylor series [15]. Corollary 1 thus extends to the evaluation of y outside the disk $|z| < \rho$. Chudnovsky and Chudnovsky further showed how to reduce the cost of evaluation from $\Omega(hp) = \Omega(p^2)$ to softly linear in p when $h = \Theta(p)$. This last situation is very natural since it covers the case where the point ζ is itself a $\mathrm{O}(p)$-digits approximation resulting from a previous computation. The method, known as the *bit burst* algorithm, consists in solving the differential equation along a path made of approximations of ζ of exponentially increasing precision. Its time complexity is $\mathrm{O}(\mathrm{M}(p(\lg p)^2))$ [16]. The improvements from the next section apply to all these settings. See also [24] for an overview of more sophisticated applications.

4. "Truncated" Binary Splitting

The superiority of binary splitting over alternatives like summing the series in floating-point arithmetic results from the controlled growth of intermediate results. Indeed, in the product tree computed by Algorithm 1, the exact representations of most subproducts $P(a, b)$ are much more compact than $\Theta(p)$-digits approximations would be. However, as already mentioned, the bit sizes of the $P(a, b)$ also grow

larger than p near the root of the tree. The size of a subproduct appearing at depth δ is roughly $2^{-\delta}N(h + r \lg N)$. Assuming $N = \Theta(p)$, this means that the intermediate results get significantly larger than the output in the top $\Theta(\lg \lg p)$ levels of the tree.

A natural remedy is to use a hybrid of binary splitting and naive summation. More precisely, we split the full product $P(0, N)$ into $\Delta = \Theta(\ln N)$ subproducts of $\mathrm{O}(p)$ bits each, which are computed by binary splitting. The results are accumulated by successive multiplications at precision $\mathrm{O}(p)$.

We make use of the following notations to state and analyze the algorithm. In Equations (7) to (11) below, the coefficients of a general matrix $A \in \mathrm{C}^{k\times k}$ are denoted $a_{p,q} = x_{p,q} + iy_{p,q}$ $(1 \leqslant p, q \leqslant k)$ with $x_{p,q}, y_{p,q} \in \mathrm{R}$. Let $\|\cdot\|$ be a submultiplicative norm on $\mathrm{C}^{k\times k}$, and let $\beta_k > 0$ be such that

$$\|A\| \leqslant \beta_k \mathcal{N}(A), \qquad \mathcal{N}(A) = \max\{|x_{i,j}|, |y_{i,j}|\}_{1\leqslant i,j\leqslant k}. \tag{7}$$

For definiteness, assume for now that $\|\cdot\| = \|\cdot\|_1$ is the matrix norm induced by the vector 1-norm. (We will discuss this choice later.) Then it holds that

$$\mathcal{N}(A) \leqslant \|A\|_1 = \max_{j=1}^{k} \sum_{i=1}^{k} |a_{i,j}| \leqslant \sqrt{2}k\mathcal{N}(A) \tag{8}$$

and

$$\|P(a, b)\| \leqslant \prod_{n=a}^{b-1} \|B(n)\| \leqslant \prod_{n=a}^{b-1} \left(1 + |\zeta| + |\zeta| \max_{k=1}^{s} \left|\frac{b_k(n)}{b_0(n)}\right|\right). \tag{9}$$

Observe that, since 1 is an eigenvalue of $B(n)$ and the norm $\|\cdot\|$ is assumed to be submultiplicative, we have $\|B(n)\| \geqslant 1$ for all n. Besides, it is clear from (3) that $\|B(n)\|$ is bounded.

Given $a \in \mathrm{Q}$ and $\varepsilon < 1$, let

$$\mathrm{Trunc}(a, \varepsilon) = \mathrm{sgn}(a) \lfloor 2^e |a| \rfloor 2^{-e}, \quad e = \lceil \lg \varepsilon^{-1} \rceil. \tag{10}$$

We have $|\mathrm{Trunc}(a, \varepsilon) - a| \leqslant \varepsilon$; the size of $\mathrm{Trunc}(a, \varepsilon)$ is $\mathrm{O}(\lg \varepsilon^{-1})$ for bounded a; and $\mathrm{Trunc}(a, \varepsilon)$ may be computed in $\mathrm{O}(\mathrm{M}(h+e))$ bit operations where h is the bit size of a. We extend the definition to matrices $A \in \mathrm{K}^{k\times k}$ by

$$\mathrm{Trunc}(A, \varepsilon) = \left(\mathrm{Trunc}(x_{p,q}, \beta_k^{-1}\varepsilon) + i\,\mathrm{Trunc}(y_{p,q}, \beta_k^{-1}\varepsilon)\right)_{1\leqslant p,q\leqslant k}, \tag{11}$$

so that again $\|\mathrm{Trunc}(A, \varepsilon) - A\| \leqslant \varepsilon$. Note that we often write expressions of the form $\mathrm{Trunc}(a \star b, \varepsilon)$ for some operator $\star$. Though this does not affect our complexity bounds, it is usually better to compute the approximate value of $a \star b$ directly instead of starting with an exact computation and truncating the result. See Brent and Zimmermann [3] for some relevant algorithms.

The complete binary splitting algorithm with truncations is stated as Algorithm 2. Its key properties are summarized in the following propositions.

Proposition 2. The output $\tilde{P} = \mathrm{TruncBinSplit}(p)$ of Algorithm 2 is such that $\|\tilde{P} - P(0, N)\| \leqslant 2^{-p}$.

Proof. Set $P^{(q)} = P(0, \lfloor \frac{q}{\Delta} N \rfloor)$ and $Q^{(q)} = P(\lfloor \frac{q}{\Delta} N \rfloor, \lfloor \frac{q+1}{\Delta} N \rfloor)$. Then, for $0 \leqslant q \leqslant \Delta$, it holds that

$$\|\tilde{P}^{(q)} - P^{(q)}\| \leqslant \frac{q}{\Delta} \frac{\varepsilon}{M^{\Delta-q}}. \tag{12}$$

Algorithm 2. TruncBinSplit(p)
The notation $X^{(q)}$, $q = 0, 1, \ldots$ refers to a single memory location X at different points q of the computation.

1 Set $\varepsilon = 2^{-p}$
2 Compute N such that $|S_N - y(\zeta)| \leqslant \varepsilon$ [22, 18]
3 Set $\Delta = \lceil \frac{N}{p}(h + r \lg N) \rceil$, where h and r are given following Eq. (1)
4 Compute M such that $\max_{q=0}^{\Delta-1} \|P(\lfloor \frac{q}{\Delta} N \rfloor, \lfloor \frac{q+1}{\Delta} N \rfloor)\| + \varepsilon \leqslant M \leqslant C^{N/\Delta}$, where C does not depend on p, h, r [say, by approximating the right-hand side of (9) from above with $\mathrm{O}(\lg p)$ bits of precision]
5 Initialize $\tilde{P}^{(0)} := \mathrm{id} \in \mathrm{K}^{(s+1)\times(s+1)}$
6 For $q = 0, 1, \ldots, \Delta - 1$
7 $\quad \hat{Q} = (\hat{Q}_{i,j}) := \mathrm{BinSplit}(\lfloor \frac{q}{\Delta} N \rfloor, \lfloor \frac{q+1}{\Delta} N \rfloor)$ (Algorithm 1)
8 $\quad \tilde{Q}^{(q)} := \mathrm{Trunc}(\hat{Q}_{s+1,s+1}^{-1} \cdot \hat{Q}, \frac{1}{2\Delta} M^{-\Delta+1} \varepsilon)$
9 $\quad \tilde{P}^{(q+1)} := \mathrm{Trunc}(\tilde{Q}^{(q)} \cdot \tilde{P}^{(q)}, \frac{1}{2\Delta} M^{-\Delta+q+1} \varepsilon)$
10 Return $\tilde{P}^{(\Delta)}$

Indeed, this is true for $q = 0$. After Step 8 of each loop iteration, we have the bound $\|\tilde{Q}^{(q)} - Q^{(q)}\| \leqslant \frac{1}{2\Delta} M^{-\Delta+1} \varepsilon \leqslant \varepsilon$ since $\|B(n)\| \geqslant 1$ for all n. Using (12) and the inequality $\|\tilde{Q}^{(q)}\| \leqslant M$ from Step 3, it follows that

$$\begin{aligned} \|\tilde{Q}^{(q)} \tilde{P}^{(q)} - Q^{(q)} P^{(q)}\| &\leqslant \|\tilde{Q}^{(q)} - Q^{(q)}\| \|P^{(q)}\| + \|\tilde{Q}^{(q)}\| \|\tilde{P}^{(q)} - P^{(q)}\| \\ &\leqslant \frac{2q+1}{2\Delta} \frac{\varepsilon}{M^{\Delta-q-1}}. \end{aligned}$$

After taking into account the truncation error from Step 9, we obtain

$$\|\tilde{P}^{(q+1)} - P^{(q+1)}\| = \|\tilde{P}^{(q+1)} - Q^{(q)} P^{(q)}\| \leqslant \frac{q+1}{\Delta} \frac{\varepsilon}{M^{\Delta-q-1}}$$

which concludes the induction. □

Proposition 3. Not counting the cost of Step 2, Algorithm 2 runs in time

$$\begin{cases} \mathrm{O}\big(\mathrm{M}(p)(h + r \lg p) \lg p\big), & \text{if } \rho < \infty, \\ \mathrm{O}\big(\mathrm{M}(p)(h + r \lg p)\big), & \text{if } \rho = \infty, \end{cases} \tag{13}$$

as $p, h, r \to \infty$ with $r = \mathrm{O}(\lg p)$ and $h = \mathrm{O}(p)$. In both cases, it uses $\mathrm{O}(p)$ bits of memory (where the hidden constant is independent of h and r, under the same growth assumptions).

We neglect the cost of finding N to avoid a lengthy discussion of the complexity of the corresponding bound computation algorithms. It could actually be checked to be polynomial in r and $\lg p$.

Proof. Computing the bound M using Equation (9) as suggested is more than enough to ensure that $\lg M = \mathrm{O}(N/\Delta)$. It requires $\mathrm{O}(N)$ arithmetic operations on $\mathrm{O}(\lg p)$-bit numbers, that is, $o(N(\lg N)^2)$ bit operations.

By Proposition 1, each of the Δ calls to BinSplit requires

$$\mathrm{O}\big(\mathrm{M}(\tfrac{N}{\Delta}(h + r \lg N)) \lg N\big) = \mathrm{O}\big(\mathrm{M}(p) \lg p\big)$$

bit operations. The resulting matrices $Q^{(p)}$ all have size $\mathrm{O}(p)$, hence the divisions from Step 8 can be done in $\mathrm{O}(\mathrm{M}(p))$ operations using Newton's method [25, Chap. 9].

		Time	Space (classical)	Space (trunc.)
$\rho < \infty$	BinSplit	$\mathrm{O}(\mathrm{M}(p(h + r\lg p)\lg p))$	$\mathrm{O}(p(h + r\lg p))$	$\mathrm{O}(p)$
	BitBurst	$\mathrm{O}(\mathrm{M}(p(\lg p)^2))$	$\mathrm{O}(p\lg p)$	$\mathrm{O}(p)$
$\rho = \infty$	BinSplit	$\mathrm{O}(\mathrm{M}(p(h + r\lg p)))$	$\mathrm{O}(p(r + h/\lg p))$	$\mathrm{O}(p)$
	BitBurst	$\mathrm{O}(\mathrm{M}(p(\lg p)^2))$	$\mathrm{O}(p)$	$\mathrm{O}(p)$

TABLE 1. Complexity of some D-finite function evaluation algorithms based on binary splitting. The rows labeled "BinSplit" summarize the cost of computing a single sum by binary splitting, with or without truncations. Those labeled "BitBurst" refer to the computation of $y(\zeta)$ by the "bit burst" method, using either of Algorithm 1 and Algorithm 2 at each step. All entries are asymptotic bounds as $p, h \to \infty$ with $h = \mathrm{O}(p)$. In the "BinSplit" case, we also let r tend to infinity under the assumption that $r = \mathrm{O}(\lg p)$. The whole point of the "bit burst" method is to get rid the dependency on h.

The truncations in Steps 8 and 9 ensure that the bit sizes of $\tilde{P}$ and $\tilde{Q}$ are always at most

$$\lg \varepsilon^{-1} + \Delta \lg M + \lg \Delta + \mathrm{O}(1) = \mathrm{O}(p). \tag{14}$$

It follows that the matrix multiplications from Step 9 take $\mathrm{O}(\mathrm{M}(p))$ operations each. Summing up, each iteration of the loop from Step 6 can be performed in $\mathrm{O}(\mathrm{M}(p)\lg p)$ operations, for a total of $\mathrm{O}(\Delta \mathrm{M}(p)\lg p)$. Equation (13) follows upon setting $N = \mathrm{O}(p)$ or $N = \mathrm{O}(p/\lg p)$ according to (6).

The required memory comprises space for the current values of $\tilde{P}^{(q)}$ and $Q^{(q)}$, any temporary storage used by the operations from Steps 7 to 9, and an additional $\mathrm{O}(\lg p)$ bits to manipulate auxiliary variables such as M and q. We have seen that $\tilde{P}^{(q)}$ and $Q^{(q)}$ have bit size $\mathrm{O}(p)$. Besides, our assumption that fast integer multiplication could be performed in linear space implies the same property for division by Newton's method. Thus, Steps 8 and 9 use $\mathrm{O}(p)$ bits of auxiliary storage. Finally, again by Proposition 1, the calls to Algorithm 1 use $\mathrm{O}((N/\Delta)(h + r\lg p)) = \mathrm{O}(p)$ bits of memory. □

Plugging Algorithm 2 into the numerical evaluation algorithms mentioned at the end of Sect. 3 yields corresponding improvements for the evaluation of D-finite functions at more general points. Table 1 summarizes the complexity bounds we obtain. The omitted proofs are direct adaptations of those that apply without truncations [5, 22, 17]. There would be much to say on the hidden constant factors. The main result may be stated more precisely as follows.

Theorem 1. Let $U \subset \mathrm{C}$ be a simply connected domain such that $0 \in U$ and $a_r(z) \neq 0$ for all $z \in U$. Fix $\ell_0, \dots, \ell_{r-1} \in \mathrm{C}$ and $\zeta \in U$. Assume that 0 is an ordinary point of (1), and let y be the unique solution of (1) defined on U and such that $y^{(k)}(0) = \ell_k$, $0 \leqslant k < r$. Then, the value $y(\zeta)$ may be computed with error bounded by 2^{-p} in time $\mathrm{O}(\mathrm{M}(p)(\lg p)^2)$ and space $\mathrm{O}(p)$, not counting the resources needed to approximate the ℓ_k or ζ to precision $\mathrm{O}(p)$ or to find suitable truncation orders for the Taylor series involved.

Finally, some comments are in order regarding the "working precision", that is, the size p' of the entries of $\tilde{P}$ and $\tilde{Q}$ in Algorithm 2. Equation (14) suggests a number of "guard digits" $p' - p = \Theta(p)$. Moreover, if the bound M is computed using (9), the hidden constant depends on the choice of $\|\cdot\|$.

Let $B_\infty = \lim_{n\to\infty} B(n)$. For the norm $\|\cdot\|_{\text{opt}}$ given by Lemma 1 below, we have

$$\lg \|P(a,b)\|_{\text{opt}} \leqslant \sum_{n=a}^{b-1} \lg\bigl(\|B_\infty\|_{\text{opt}} + \mathrm{O}(n^{-1})\bigr) = \mathrm{O}\bigl(\lg(b-a)\bigr),$$

and hence $\lg \|P(a,b)\| = \mathrm{O}(\lg(b-a))$ for any norm $\|\cdot\|$.

Lemma 1. There exists a matrix norm $\|\cdot\|_{\text{opt}}$ such that $\|B_\infty\|_{\text{opt}} = 1$.

Proof. We mimic the classical proof of Householder's theorem [20, Sect. 4.2]. By (3), the limit $C_\infty = \lim_{n\to\infty} C(n)$ is the companion matrix of the polynomial $z^s a_r(1/z)$. The eigenvalues of ζC_∞ are strictly smaller than 1 in absolute value since $|\zeta| < \rho$. Let Γ be such that $\Gamma^{-1} C_\infty \Gamma$ is in (lower) Jordan normal form. Let $\lambda > 0$, and set $\Pi = \operatorname{diag}(\Gamma, 1) \cdot \operatorname{diag}(1, \lambda, \ldots, \lambda^s)$. Then $\Pi^{-1} B_\infty \Pi$ is lower triangular, with off-diagonal entries tending to zero as $\lambda \to 0$. Hence we have $\|\Pi^{-1} B_\infty \Pi\|_1 = 1$ for λ small enough. We choose such a λ (e.g., $\lambda = \frac{1-|\zeta|/\rho}{2\max(1,|\zeta|)}$) and set $\|A\|_{\text{opt}} = \|\Pi^{-1} A \Pi\|_1$. □

One way to eliminate the overestimation in the algorithm is to compute approximations of the matrices $P(\lfloor \frac{q}{\Delta} N \rfloor, \lfloor \frac{q+1}{\Delta} N \rfloor)$ with $\mathrm{O}(\lg p)$ digits of precision before doing the computation at full precision. One then uses the norms of these approximate products instead of those of the individual $B(n)$ to determine M. We can also explicitly construct an approximation $\tilde{\Pi}$ of the matrix Π from the proof of Lemma 1 precise enough that $\|\tilde{\Pi}^{-1} B_\infty \tilde{\Pi}\|_1 = 1$, and use the corresponding norm instead of $\|\cdot\|_1$ in (9). (Compare [22, Algorithm B].) Other options include computing symbolic bounds on the coefficients of $P(a,b)$ as a function of a and b [18] or finding an explicit integer n_0 such that $n \geqslant n_0 \Rightarrow \|B(n)\|_{\text{opt}} = 1$ based on the symbolic expression of n. Which variant to use in practice depends on the features of the implementation platform.

In any case, replacing the $\mathrm{O}(\cdot)$ in the space complexity bound by an explicit constant would also require more specific assumptions on the memory representation of the objects we work with, as well as finer control on the space complexity of integer multiplication and division (see, e.g., Roche [19]).

5. Final Remarks

What we lose and what we retain. The price we pay for the reduced memory usage is the ability to easily extend the computation to higher precision. Indeed, the classical algorithm computes the exact value of the matrix $P(0,N)$, from which we can deduce $P(0,N')$ for any $N' > N$ in time roughly proportional to $N' - N$. This is no longer true with the linear-space variant. In some "lucky" cases where $P(0,N)$ can be represented exactly in linear space, it is possible to get the memory usage down to $\mathrm{O}(N)$ while preserving restartability: see Cheng et al. [4] and the references therein. Additionally, the resulting running time is reportedly lower than using truncations, probably owing to the fact that the size of the subproducts in the $\lg(N/\Delta)$ lower levels of the tree is reduced as well. Unfortunately, the applicability of the technique is limited to very special cases.

Two other traditional selling points of the binary splitting method are its easy parallelization and good memory locality. Nothing is lost in this respect, except that the memory bound grows to $\Theta(t \cdot p)$ when using $t = o(\lg N)$ parallel tasks in the approximate part of the computation.

Generalizations. The idea of binary splitting "with truncations" and the outline of its analysis adapt to various settings not covered here. For instance, we may consider systems of linear differential equations instead of scalar equations [5]. Product trees of matrices over number fields $\mathrm{K}' = \mathrm{Q}(\alpha)$ other than $\mathrm{Q}(i)$ or over rings of truncated power series $\mathrm{K}'[[\varepsilon]] / \langle \varepsilon^k \rangle$ are also useful, respectively, to evaluate limits of D-finite functions at regular singular points of their defining equations, and to make the analytic continuation process more efficient for equations of large order [22, 17]. It is not essential either that the coefficients of the recurrence relation satisfied by the y_n are rational functions of n: all we really ask is that they have suitable growth properties and can be computed fast.

Implementation. We are working on an implementation of the algorithm from Sect. 4 in an experimental branch of the software package NumGfun [16]. The current state of the code is available from

`http://marc.mezzarobba.net/supplementary-material/trunc-CASC2012/`.

A comparison (updated periodically) with the implementation of binary splitting without truncations used in previous releases of NumGfun is also included.

Acknowledgments. I would like to thank Nicolas Brisebarre and Bruno Salvy for encouraging me to write this note and offering useful comments, and Anne Vaugon for proofreading parts of it.

References

[1] D. J. Bernstein. Fast multiplication and its applications. In J. Buhler and P. Stevenhagen, editors, *Algorithmic Number Theory*, pages 325–384. Cambridge University Press, 2008. URL: `http://www.msri.org/communications/books/Book44/`.

[2] J. M. Borwein and P. B. Borwein. *Pi and the AGM*. Wiley, 1987.

[3] R. P. Brent and P. Zimmermann. *Modern Computer Arithmetic*. Cambridge University Press, 2010. URL: `http://www.loria.fr/~zimmerma/mca/mca-cup-0.5.7.pdf`.

[4] H. Cheng, G. Hanrot, E. Thomé, E. Zima, and P. Zimmermann. Time- and space-efficient evaluation of some hypergeometric constants. In D. Wang, editor, *ISSAC '07*, pages 85–91. ACM, 2007. URL: `http://www.cs.uleth.ca/~cheng/papers/issac2007.pdf`.

[5] D. V. Chudnovsky and G. V. Chudnovsky. Computer algebra in the service of mathematical physics and number theory. In Chudnovsky and Jenks [6], pages 109–232.

[6] D. V. Chudnovsky and R. D. Jenks, editors. *Computers in Mathematics*, volume 125 of *Lecture Notes in Pure and Applied Mathematics*, Stanford University, 1986. Dekker, 1990.

[7] M. Fürer. Faster integer multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009. URL: `http://www.cse.psu.edu/~furer/Papers/mult.pdf`.

[8] W. Gosper. Strip mining in the abandoned orefields of nineteenth century mathematics. In Chudnovsky and Jenks [6], pages 261–284.

[9] X. Gourdon and P. Sebah. Constants and records of computation. Online. Updated August 12, 2010. URL: `http://numbers.computation.free.fr/Constants/constants.html`.

[10] X. Gourdon and P. Sebah. Binary splitting method, 2001. URL: `http://numbers.computation.free.fr/Constants/Algorithms/splitting.ps`.

[11] T. Granlund et al. *GNU Multiple Precision Arithmetic Library*. URL: `http://gmplib.org/`.

[12] B. Haible and T. Papanikolaou. Fast multiprecision evaluation of series of rational numbers, 1997. URL: `http://www.informatik.tu-darmstadt.de/TI/Mitarbeiter/papanik/ps/TI-97-7.ps.gz`.

[13] E. Hille. *Ordinary differential equations in the complex domain*. Wiley, 1976. Dover reprint, 1997.

[14] R. B. Kreckel. decimal(γ) =∼ "0.57721566[0-9]{1001262760}39288477", 2008. URL: http://www.ginac.de/~kreckel/news.html#EulerConstantOneBillionDigits.
[15] J. H. Mathews. Bibliography for Taylor series method for D.E.'s, 2003. URL: http://math.fullerton.edu/mathews/n2003/taylorde/TaylorDEBib/Links/TaylorDEBib_lnk_3.html.
[16] M. Mezzarobba. NumGfun: a package for numerical and analytic computation with D-finite functions. In W. Koepf, editor, *ISSAC '10*, pages 139–146. ACM, 2010. URL: http://arxiv.org/abs/1002.3077, doi:10.1145/1837934.1837965.
[17] M. Mezzarobba. *Autour de l'évaluation numérique des fonctions D-finies*. Thèse de doctorat, École polytechnique, Nov. 2011. URL: http://tel.archives-ouvertes.fr/pastel-00663017/.
[18] M. Mezzarobba and B. Salvy. Effective bounds for P-recursive sequences. *Journal of Symbolic Computation*, 45(10):1075–1096, 2010. URL: http://arxiv.org/abs/0904.2452, doi:10.1016/j.jsc.2010.06.024.
[19] D. S. Roche. *Efficient Computation with Sparse and Dense Polynomials*. PhD thesis, University of Waterloo, 2011. URL: http://uwspace.uwaterloo.ca/handle/10012/5869.
[20] D. Serre. *Matrices*, volume 216 of *Graduate Texts in Mathematics*. Springer, 2002.
[21] R. P. Stanley. Differentiably finite power series. *European Journal of Combinatorics*, 1(2):175–188, 1980.
[22] J. van der Hoeven. Fast evaluation of holonomic functions. *Theoretical Computer Science*, 210(1):199–216, 1999. URL: http://www.texmacs.org/joris/hol/hol-abs.html.
[23] J. van der Hoeven. Fast evaluation of holonomic functions near and in regular singularities. *Journal of Symbolic Computation*, 31(6):717–743, 2001. URL: http://www.texmacs.org/joris/singhol/singhol-abs.html.
[24] J. van der Hoeven. *Transséries et analyse complexe effective*. Habilitation à diriger des recherches, Université Paris-Sud, Orsay, France, 2007. URL: http://www.texmacs.org/joris/hab/hab-abs.html.
[25] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2nd edition, 2003.
[26] S. V. Yakhontov. Calculation of hypergeometric series with quasi-linear time and linear space complexity. *Vestnik Samarskogo Gosudarstvennogo Tekhnicheskogo Universiteta. Seriya: Fiziko-Matematicheskie Nauki*, 24:149–156, 2011.
[27] S. V. Yakhontov. A simple algorithm for the evaluation of the hypergeometric series using quasi-linear time and linear space. Preprint 1106.2301v1, arXiv, June 2011. English version of [26]. URL: http://arxiv.org/abs/1106.2301.
[28] A. J. Yee. Mathematical constants – billions of digits. Online. Updated March 7, 2011. URL: http://www.numberworld.org/digits/.

Inria, AriC, LIP (UMR 5668 CNRS-ENS Lyon-Inria-UCBL), ENS de Lyon, Lyon, France
E-mail address: marc@mezzarobba.net

The present version is identical except for minor corrections and formatting differences to the article published in V.P. Gerdt et al. (Eds.), CASC 2012, LNCS 7442, pp. 212–223, 2012. The original version is available at www.springerlink.com.