Uniform Complexity of Approximating Hypergeometric Functions with Absolute Error

Zilin Du and Chee Yap*
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
251 Mercer Street, New York
NY 10012, USA
zilin@google.com , yap@cs.nyu.edu

June 16, 2009

Abstract

The approximation of the general hypergeometric function $H(\mathbf{a};\mathbf{b};x) = {}_pF_q(\mathbf{a};\mathbf{b};x)$ to any specified absolute error bound is shown to be solvable. In other words, we provide an algorithm that is uniform in the hypergeometric parameters $\mathbf{a} = (a_1,\ldots,a_p), \mathbf{b} = (b_1,\ldots,b_q)$. An explicit bound for the complexity of our algorithm is given when the input numbers are rational. We further address the problem of evaluating H when x is a "blackbox number", i.e., x is represented by a procedure that returns an approximation of x to any specified absolute precision. This generalization allows us to extend our approximability results to most of the familiar transcendental functions of classical analysis that are derived from H. In particular, this solves the so-called Table Maker's Dilemma for such functions. Our algorithm has been implemented in our open-source Core Library.

1 Introduction

Let $\mathbf{a} = (a_1,\ldots,a_p)$ and $\mathbf{b} = (b_1,\ldots,b_q)$ be sequences of rational numbers where $p, q \in \mathbb{N}$ and $p \leq q+1$. The **hypergeometric function** determined by these numerical parameters is

$$F(x) = {}_pF_q(\mathbf{a};\mathbf{b};x) = \sum_{k=0}^{\infty} t_k \tag{1}$$

where $t_k = \frac{(a_1)_k (a_2)_k \cdots (a_p)_k}{(b_1)_k (b_2)_k \cdots (b_q)_k} \frac{x^k}{k!}$ and $(a)_k = (a)(a+1)\cdots(a+k-1)$ is the Pochhammer symbol or rising factorial. Here $\mathbf{a}, \mathbf{b}$ are known as the upper and lower **parameters** and x the **argument** of ${}_pF_q$. Although x is unrestricted for $p < q+1$, it is standard to assume $|x| < 1$ when $p = q+1$. Many well-known functions in analysis are obtained by simple transformations of hypergeometric functions. Typically, we transform $F(x)$ to $E(x) := A(x)F(B(x))$ where $A(x)$ and $B(x)$ are polynomials. This is illustrated in Table 1 (see [23, p. 42ff] for more examples).

We regard the parameters $\mathbf{a}, \mathbf{b}$ as fixed in the function $F(x) = {}_pF_q(\mathbf{a};\mathbf{b};x)$. Let $H(\mathbf{a};\mathbf{b};x) = {}_pF_q(\mathbf{a};\mathbf{b};x)$ denote the **general hypergeometric function** where the parameters as well as p, q can now vary. We want to approximate such functions, on rational input arguments and rational outputs, to prescribed absolute error bounds: this is what we call "absolute approximation". There is a similar concept of "relative approximation"

*This work is supported by NSF Grant CCF-043836.

$E(x) = A(x)F(B(x))$	$F(x)$	$A(x)$	$B(x)$
$\exp(x)$	${}_0F_0(;;x)$	1	1
$\cos(x)$	${}_0F_1(;\frac{1}{2};x)$	1	$-x^2/4$
$\sin(x)$	${}_0F_1(;\frac{3}{2};x)$	x	$-x^2/4$
$\cosh(x)$	${}_0F_1(;\frac{1}{2};x)$	1	$x^2/4$
$\sinh(x)$	${}_0F_1(;\frac{3}{2};x)$	x	$x^2/4$
$\mathrm{erf}(x)$	${}_1F_1(\frac{1}{2};\frac{3}{2};x)$	x	$-x^2$
$(1+x)^{-v}$	${}_1F_0(v;x)$	1	$-x$
$\ln(1+x)$	${}_2F_1(1,1;2;x)$	x	$-x$
$\arcsin(x)$	${}_2F_1(\frac{1}{2},\frac{1}{2};\frac{3}{2};x)$	x	x^2
$\arctan(x)$	${}_2F_1(\frac{1}{2},1;\frac{3}{2};x)$	x	$-x^2$

where relative error is used in place of absolute error. It is conceivable that each hypergeometric function ${}_pF_q(\mathbf{a};\mathbf{b};x)$ is absolutely approximable, but the function H is not absolutely approximable. In other words, the absolute approximation of H requires a *uniform* algorithmic method that is applicable to arbitrary values of the parameters; to our knowledge, this has never been demonstrated before. There are related results such as a uniform procedure to approximate any complex analytic function given by Weihrauch [34, p. 116]. But this result additional input parameters related to the radius of convergence must be explicitly given. We discuss this connection at the end of Section 3. In this paper, we not only provide an algorithm for absolute approximation of H, but also provide a complexity bound. It is interesting to note that the corresponding result for the relative approximability of H is an open problem (we briefly discuss this in the final remarks).

The problem of evaluating hypergeometric functions is a highly classical problem (e.g., [12, 22]). The usual modus operandi here is one that is widely used in numerical analysis: the algorithms are based on fixed-precision arithmetic (e.g., IEEE Standard), and the goal is to design algorithms that try to minimize the round-off errors in the final result. However, it is difficult to give *a priori* guarantees on the final accuracy. It is possible to give *a posteriori* guarantees using interval arithmetic; but such bounds may not be tight. The computational mode of *a posteriori* accuracy guarantees is called **validated** or **certified computation** [30]; following [35], we use the term **guaranteed precision computation** for the stronger notion of *a priori* guarantees, which is used in this paper. In general, one cannot transform a posteriori methods into a priori ones (the obvious method of increasing precision iteratively may fail [35]). This has given rise to the "Table Maker's Dilemma" [21], described as the problem of computing correctly rounded values of (transcendental) functions. This problem was solved in [21] for some elementary functions in the context of double-precision format. Another example is Nardin et al [26, 25], who described an evaluation method for confluent hypergeometric series (on large complex arguments) which they verify to be accurate to at least 9 digits. Their "verification" consists of a battery of 12 tests. The results of our paper will automatically produce guaranteed accuracy in such evaluations. We have implemented the current algorithm for real hypergeometric functions in the Core Library [18]. Some experimental results are reported here.

There are many other applications of guaranteed accuracy computation. Our work were motivated by applications to exact geometric computation [15] and also geometric theorem proving [29]. More generally, see [20] for computer-assisted theorem proving as well as automatic theorem proving. McCullough [10, 11] describes the problem of evaluating commercial statistical packages on standard test data. In McCullough, the answers for test data were computed using 500-digit arithmetic, and the result rounded to 16-digits is empirically deemed to be correctly rounded. Dhiflaoui et al [13] describes an application for certifying large LP solvers.

Closely related to this paper, van der Hoeven [31, 32] presented fast algorithms for evaluating holonomic functions $f(x)$. Such an $f(x)$ satisfies a linear differential equation $\sum_{i=0}^{k} P_i(x)f^{(i)}(x) = 0$ where $P_i(x) \in \mathbb{Z}[x]$. For instance, if $f(x) = {}_2F_1(a,b;c;x)$ then it satisfies the equation

$$x(1-x)f''(x) + (c - x(1+a+b))f'(x) - abf(x) = 0.$$

Van der Hoeven's setting is more general than ours in two ways: first, hypergeometric functions are holonomic and second, he treats complex functions which live on Riemann surfaces. But the complexity results in his general setting are weaker than ours. He shows that $f(x)$ can be approximated to absolute n-bits in time $O(M(n\log^2 n))$ where $M(n)$ is the complexity of multiplying n-bit integers. But this is a "local complexity

bound" in the sense that f is fixed and x restricted to a local neighborhood. In contrast, our complexity bounds are global, and even uniform, results: x is unbounded and our functions f are specified by input parameters $\mathbf{a}, \mathbf{b}$. Below, we clarify this local/global/uniform terminology. Uniform bounds can present nontrivial and subtle difficulties (cf. Section 5). In any case, it remains a challenge to give a uniform complexity bound in van der Hoeven's setting. The connection between [31, 32] and the closely related work of Chudnovsky [9] is described in [33]. Another emphasis of our paper is "guaranteed precision computation" [35]. It is important to note that we never use asymptotic (big-Oh) error bounds; all error bounds are given by inequalities with explicit constants. Thus one can directly implement the algorithms of this paper to achieve guaranteed precision, by exploiting our explicit constants.

The direct predecessor of the present paper is [15], which not only aimed at a uniform approximation algorithm for hypergeometric functions, but also introduced several other issues arising in the efficient implementation of such functions: argument reduction [24], preprocessing of hypergeometric parameters, and use of precomputed constants. For instance, a basic technique to automatically detect the "contiguity relationships" [17, 6] of hypergeometric parameters in order to greatly speed-up evaluation of hypergeometric series. The approximation algorithm of this paper follows the basic strategy in [15], but has three major improvements: (a) The present algorithm is completely general while the original algorithm is only complete when the series (1) satisfies $|t_k| > |t_{k+1}|$ for all $k \geq 0$. (b) In our original algorithm, we truncate the series (1) when k is sufficiently large, based on an implicit criterion on the term t_k. The current algorithm gives an explicit upper bound of $k \leq n_3$ where n_3 is easily computed from the hypergeometric parameters. Because of this, we are able to give explicit complexity (Section 5). For an application of such bounds, see [7]. (c) In order to achieve guaranteed precision under argument reduction, we must bound the sensitivity of the function to variations in its argument. While [15] solves this for the standard elementary functions, the present paper gives a uniform solution for all hypergeometric functions (Section 4).

Computational Model. We assume that our functions such as $F(x)$ or $H(\mathbf{a}; \mathbf{b}; x)$ are real. We also assume that *the parameters* $\mathbf{a}, \mathbf{b}$ *are rational numbers, and* x *is a bigfloat* (defined below). It is important to clarify our computational model for real numbers because this area is currently under debate. This is unlike the situation for computation over a countable domain such as finite strings Σ^* or natural numbers $\mathbb{N}$. In the countable case, the Turing model or equivalent is widely accepted. In the uncountable domain of reals, there are two main competing approaches which we will call the **algebraic approach** (popularized by Smale [2, 28], but it is also the de facto model for theoretical algorithms in computer science) and the **analytic approach** (represented by Weihrauh [34] or Ko [19]). Motivated by current software and research in Exact Geometric Computation (EGC) we propose in [35] a weaker form of computing real function than in the analytic approach. We call this the **approximation approach**. If $\widetilde{a}, a$ are real numbers such that $|\widetilde{a} - a| \leq 2^{-i}$, we say $\widetilde{a}$ is an *i-bit absolute approximation* of a; if $|\widetilde{a} - a| \leq |a|2^{-i}$, we say $\widetilde{a}$ is an *i-bit relative approximation* of a.

Let us compare these three approaches in terms of computing a real function $f : \mathbb{R} \to \mathbb{R}$. How can we input a real number to an algorithm for f? The analytic approach says that a real number x is represented by a **blackbox number** (or oracle) $\underline{x} : \mathbb{N} \to \mathbb{Q}$ where $|\underline{x}(i) - x| \leq 2^{-i}$ for all i. Then x is a **computable real** if it has a blackbox representation $\underline{x}$ that is recursive. Turing machines must be suitably modified to accept inputs and outputs such as $\underline{x}$: Weihrauch [34] introduces TTE machines to this end, but Ko [19] uses oracle Turing machines. Oracle Turing machine are better suited for complexity purposes. The algebraic approach say that a real number may be directly represented as an atomic object in the computational model, and basic operations ($+, -, \times$, etc) can directly operate on such objects. Both approaches lead to well-known difficulties. To these, we may add the criticism of EGC [35, 36]. Central to EGC is the zero problem. In terms of computing the real function $f : \mathbb{R} \to \mathbb{R}$, the **zero problem** amounts to deciding membership in the set $Zero(f) := \{x \in \mathbb{R} : f(x) = 0\}$. Unfortunately, the zero problem is undecidable in the analytic approach, and trivial in the algebraic approach [35, 36]. So neither approaches are appropriate models for the EGC mode of computing.

Instead of presuming to represent all real numbers from the outset, the approximation approach begins from a suitable set $\mathbb{F} \subseteq \mathbb{R}$ of **base reals**. By definition, base reals is any set satisfying three groups of properties: (1) $\mathbb{F}$ is a ring extension of $\mathbb{Z}$, so $\mathbb{Z} \subseteq \mathbb{F} \subseteq \mathbb{R}$. (2) $\mathbb{F}$ is countable and dense in $\mathbb{R}$. (3) $\mathbb{F}$ has a representation in which the ring operations and comparisons are effective. Two standard models for $\mathbb{F}$ are $\mathbb{F} = \mathbb{Q}$ and $\mathbb{F} = \mathbb{D}$. Here, $\mathbb{D}$ is the set $\mathbb{Z}[\frac{1}{2}] = \{m2^n : m, n \in \mathbb{Z}\}$ of **bigfloats** (or **dyadic numbers**).

Finally, we say f is **absolutely approximable** if there is a recursive function $\widetilde{f} : \mathbb{F}^2 \to \mathbb{F}$ such that $\widetilde{f}(x, p)$ is a p-bit absolute approximation of $f(x)$. To indicate that p is a special argument, we may write $\widetilde{f}(x; p)$ or $\widetilde{f}(x)[p]$. All numerical input and output for algorithms are restricted to $\mathbb{F}$. We now define the set $Zero_{\mathbb{F}}(f)$ as $\{x \in \mathbb{F} : f(x) = 0\}$; the **zero problem** amounts to recognizing this set, and it has a well-defined complexity in standard sense.

Our focus on computing approximations $\widetilde{f}(x, p)$ for f conforms to computing practice, and has many advantages: we can build a theory of real approximation and its complexity using standard Turing machines, and directly relate them to standard complexity classes. In contrast, if x remains an arbitrary real number, then the complexity function "$T(x, p)$" (=the number of steps to compute $\widetilde{f}(x, p)$) does not give give rise to a natural complexity theory. If we further assume $\mathbb{F} = \mathbb{Z}[\frac{1}{2}]$, then many important results about the complexity of approximability were obtained three decades ago by Brent (e.g., [3, 4]). In particular, he showed that all the common elementary functions can be approximated efficiently using Newton-like schemes, perhaps combined with AGM-based iterations. Such results form the basis of practical algorithms in this area. We emphasize that Brent's complexity bounds are all of the "local variety", with complexity bounds of the form $M(n) \log^c n$ where $M(n)$ is the complexity of integer multimplication and $c \geq 0$ is some small constant. Warious authors including extended such bounds to more general settings.

Contributions of this Paper. There are three main results:
(I) In Section 3, we show that H is absolutely approximable.
(II) In Section 4, we show that H is absolutely approximable when x is a blackbox number. This generalization is necessary for various applications: (a) argument reduction [24, 15], (b) evaluation of hypergeometric functions at irrational values such as $x = \pi$ or $x = \sqrt{2}$. (c) absolute approximation of standard functions $E(x)$ that require non-trivial transformation of the x parameter.
(III) In Section 5, we bound the complexity of absolute approximation of H. This requires a refinement of the algorithm in Section 3.

For further details, and for any omitted proofs, we refer to Zilin's thesis [14]. In particular, description of our implementation in the Core Library [18] may be found there. A preliminary version of this paper appeared in [16]. Although we follow the basic approach in [14, 16], several detail have been simplified in the present paper.

2 Preliminaries: Representation and Complexity Model

We use a useful notation for error from [36]: a numerical expression of the form "$x \pm h$" where $x, h \in \mathbb{R}$, represents the number $x + \theta h$ for some θ where $|\theta| \leq 1$. Thus θ is an implicit variable, much like the implicit constant[1] in big-Oh notations.

Thus, all appearences of "$\pm$" in this paper should be replaced by a sequence of the form "$+\theta$" for some $|\theta| \leq 1$. For $x, \widetilde{x}, \ell \in \mathbb{R}$, we say $\widetilde{x}$ is an **absolute ℓ-bit approximation** of x if $\widetilde{x} = x \pm 2^{-\ell}$. Similarly, $\widetilde{x}$ is a **relative ℓ-bit approximation** of x if $\widetilde{x} = x(1 \pm 2^{-\ell})$.

For our complexity results, we need to be more explicit about $\mathbb{F}$, its representation and the complexity of basic arithmetic in $\mathbb{F}$. Apropos to the function H, we will assume the set of base reals is $\mathbb{F}_0 := \mathbb{Q}$. However, for internal computation, we will extensively use approximations based on bigfloats $\mathbb{F}_1 := \mathbb{D}$. The most efficient approximate arithmetic over $\mathbb{D}$ goes back to Brent [3]. Brent's results are stated for functions evaluated in some bounded range. We call this "local complexity". However, we will need "global" and "uniform" complexity bounds. Let us clarify these distinction.

Let $f : S \to \mathbb{R}$ be a function where $S \subseteq \mathbb{R}$, and $\widetilde{f} : (S \cap \mathbb{F}) \times \mathbb{F} \to \mathbb{F}$ be an absolute approximation of f.
Local complexity: $T_{f,x}(n)$ denotes the complexity of computing an n-bit absolute approximation of $f(x)$. Here, f and $x \in S$ are fixed. We could slightly generalize this to allow x to range over a bounded range: Brent's results [3] are of this sort.

[1] The analogy with big-Oh notations extends to their usefulness in a long sequence of derivations. See, e.g., the proof of Theorem 12 below. As for big-Oh notations, the equality symbol between two $\pm$-expressions must be treated as "one-way equalities". E.g., we might write $\pi = 3.1 \pm 0.05 = 3 \pm 0.2$ But it would be incorrect to write $\pi = 3 \pm 0.2 = 3.1 \pm 0.05$. To explain this, we view the expression "3.1 ± 0.2" as the set $\{3.1 + \theta 0.2 : |\theta| \leq 1\}$. The one-way equality between two such expressions amounts to asserting a set inclusion relation ($\subseteq$) between the corresponding sets. Thus "$3.1 \pm 0.05 = 3 \pm 0.2$" means "$\{3.1 + \theta 0.05 : |\theta| \leq 1\} \subseteq \{3 + \theta 0.2 : |\theta| \leq 1\}$".

Global complexity: $T_f(m,n)$ denotes the worst-case complexity of computing an n-bit absolute approximation of $f(x)$ where x has an m-bit representation. Here, x can vary over all of S.
Uniform complexity: If G is a set of parametrized real functions, let $T_G(\ell,m,n)$ denote the worst-case complexity of computing an n-bit absolute approximation of $f(x)$, where x has an m-bit representation and $f \in G$ is specified by an ℓ-bit parameter. Thus f and x are both varying.

We discuss representation of bigfloats and rationals. A rational number $a = p/q$ is represented straightforwardly by a pair (p,q) where p,q are integers in binary notation and $q \neq 0$. A bigfloat $x \in \mathbb{D}$ is also represented by a pair e, f of integers in binary notation. The pair e, f represents the value $x = f2^{e-\mathrm{msb}(f)} \in \mathbb{D}$ where $\mathrm{msb}(f) = \lfloor \lg f \rfloor$ ($\lg = \log_2$). We write "$x \sim \langle e, f\rangle$" to indicate that x is represented by the pair (e,f), where e and f are the **exponent** and **fraction** of the representation. Note that $\langle e, f\rangle$ and $\langle e, 2f\rangle$ represents the same bigfloat; but a representation $\langle e, f\rangle$ is **normal** if f is odd, or if $e = f = 0$. Let $\langle f\rangle$ be a shorthand for $f2^{-\mathrm{msb}(f)} \sim \langle 0, f\rangle$. Thus for $f \neq 0$, we have $\langle f\rangle \in [\frac{1}{2}, 1)$.

The **size** of a rational representation (p,q) is defined as $\max\{\lg_1 p, \lg_1 q\}$ where we define $\lg_1 x := \max\{1, \log_2 |x|\}$ (this takes care of the case $x = 0$). Note that we do not take ceiling or floor in the definition of size, so that sizes are not[2] necessarily integer. Similarly, the **size** of a bigfloat representation $\langle e, f\rangle$ is defined as $\max\{\lg_1 e, \lg_1 f\}$. Observe that size definition is relative to representation. So if a bigfloat $a = 2^e f$ could be represented as a rational number by the pair $(f, 2^{e-\mathrm{msb}(f)})$ assuming $e \geq \mathrm{msb}(f)$. Then its rational size is $\max\{e - \mathrm{msb}(f), \lg_1 f\}$. This could be exponentially larger than its bigfloat size, $size(\langle e, f\rangle) = \max\{\lg_1 e, \lg_1 f\}$. Hence, even though bigfloats are special cases of rationals, its representation may be exponentially more "compact" than rational.

In the uniform evaluation of general hypergeometric function $H(\mathbf{a}, \mathbf{b}; x)$, we assume $\mathbf{a}, \mathbf{b}$ have rational representations and x has a bigfloat reprsentation. Complexity bounds are relative to the sizes of these representations.

Our complexity bounds will contain terms involving $M(n)$, defined as the complexity of multiplying two n-bit numbers. On a Turing machine, the Schönhage-Strassen bound is $M(n) = O(n \log n \log\log n)$, although it may also be useful to substitute $M(n) = n^2$ in practice. However, the Turing machine model is rather sensitive for some of the tight bounds we aspire to (see next lemma). Hence we prefer to assume that all our complexity results uses the pointer machine model of Schönhage [27, 35]. Such pointer machines can simulate a multitape Turing machine in linear time, and Schönhage has shown $M(n) = O(n)$ in this model.

Brent [3] noted that we can compute $1/x$, $\sqrt{x}$, xy to relative s bits in $O(M(s))$ time, provided $x \neq 0, y$ are *bounded* bigfloats. See [1] for similar results in the integer setting. The following results from [8, Sect. 4 and Appendix] apply to unbounded bigfloats (i.e., are global complexity bounds):

LEMMA 1. *Let x, y be bigfloats with ℓ-bit exponent and m-bit fraction. Then there exist pointer machine algorithms to*
(i) evaluate $x + y$ to relative s bits in $O(s + \ell)$ time.
(ii) evaluate $x \cdot y, 1/x, \sqrt{x}$ to relative s bits in $O(M(s) + \ell)$ time.

Note that the complexity does not depend on m, only on s. Such a result would not be generally possible on the Turing machine model. Also, these results depend on the big-float representation, not the rational representation of numbers.

3 The Uniform Evaluation Algorithm

In this section, we present an algorithm for computing an ℓ-bit absolute approximation of $H(\mathbf{a}; \mathbf{b}; x)$, given $\mathbf{a}, \mathbf{b}, x, \ell$. For any $n \in \mathbb{N}$, write $H(\mathbf{a}; \mathbf{b}; x) = S_n + R_n$ where $S_n = \sum_{k=0}^{n-1} t_k$ and $R_n = \sum_{k=n}^{\infty} t_k$ (see (1)). As in [15], our algorithm proceeds in two stages: in Stage A, we compute an n_3 such that $|R_n| \leq 2^{-\ell-1}$ for all $n > n_3$. In Stage B, we compute an approximation $\widetilde{S}_n$ such that $|\widetilde{S}_n - S_n| \leq 2^{-\ell-1}$. Note that Stage B is clearly computable – in fact, computing this approximation is trivial to implement in Core Library. Note that [15] gives a more efficient treatment of alternating series; we forgo this refinement in the present paper. The rest of this section focuses on Stage A.

[2]The key property for "size" in complexity theory is not that they must be integers, but that there are reasonable bounds on the number of inputs up to any given size. One difficulty in developing a complexity theory for real numbers is to provide a suitable notion of size. The absolute value $|x|$ is unsuitable for measuring the size of x.

STAGE A. We introduce some notations: if $\mathbf{a} = (a_1, \ldots, a_p)$, then $s_k(\mathbf{a}) := \sum_{i=1}^{p} a_i^k$. But we will further split this sum into its positive and negative parts: $s_k(\mathbf{a}) = s_k^+(\mathbf{a}) - s_k^-(\mathbf{a})$ where $s_k^+(\mathbf{a}) := \sum_{i:a_i^k>0} a_i^k$, and $s_k^-(\mathbf{a}) := s_k^+(\mathbf{a}) - s_k(\mathbf{a})$. Note that $s_k^-(\mathbf{a}) \geq 0$, and if k is even, then $s_k^-(\mathbf{a}) = 0$. Relative to $\mathbf{a}$, we define the constants $a^+_{\max}, a^-_{\max}, a_{\max}, \overline{a}$ as follows: $a^+_{\max}$ is the maximum value of the positive a_i's; $a^-_{\max}$ is the maximum of $|a_i|$'s where $a_i < 0$. If there are no negative a_i's, set $a^-_{\max}$ to 0; similarly, there are no positive a_i's, set $a^+_{\max}$ to 0. Finally, set $a_{max} := \max\{a^+_{\max}, a^-_{\max}\}$ and $\overline{a} := \left(\sum_i^p a_i\right)/p$ (average).

Define the polynomial

$$P_{\mathbf{a}}(n) = \prod_{i=1}^{p}(a_i + n). \tag{2}$$

Each term t_{n+1} in our hypergeometric series $H(\mathbf{a}; \mathbf{b}; x)$ (see (1)) can be written as $t_{n+1} = t_n f(n) x$ where

$$f(n) = P_{\mathbf{a}}(n)/P_{\mathbf{b}'}(n), \tag{3}$$

$\mathbf{b}' = (b_0, \mathbf{b}) = (b_0, b_1, \ldots, b_q)$ and $b_0 = 1$ is the implicit lower parameter.

Stage A amounts to computing three numbers:
(0) $n_0 = n_0(x, \mathbf{a}, \mathbf{b})$ such that for $n > n_0$, $|t_n|$ is monotone decreasing.
(1) $n_1 = n_1(\mathbf{a}, \mathbf{b})$ such that for $n > n_1$, $f(n)$ is monotone increasing or decreasing.
(2) n_2 is defined to be $1 + \max n_0, n_1$.
(3) $n_3 = n_3(x, \mathbf{a}, \mathbf{b}, \ell)$ such that for all $n > n_3$, $|R_n| \leq 2^{-\ell-1}$.

We begin with $n_0 = n_0(x, \mathbf{a}, \mathbf{b})$:

LEMMA 2. *Let*

$$n_0 := \begin{cases} \max\{\bar{a} + 2\, b^-_{max}, (2^{q+2}|x|)^{\frac{1}{q+1-p}}\} & \text{if } \ p < q+1, \\ \frac{\bar{a}+b^-_{max}}{|x|^{-(1/p)}-1} + b^-_{max} & \text{if } \ p = q+1. \end{cases} \tag{4}$$

Then for $n > n_0$, we have

$$|xf(n)| \begin{cases} \leq 1/2 & \text{if } \ p < q+1 \\ < 1 & \text{if } \ p = q+1 \end{cases}.$$

Proof. We have

$$\begin{aligned} P(n) &= \prod_{i=1}^{p}(a_i + n) \\ &\leq \left[\frac{(a_1+n) + (a_2+n) + \cdots + (a_p+n)}{p}\right]^p = (\bar{a}+n)^p, \\ Q(n) &= \prod_{j=0}^{q}(b_j + n) \geq (n - b^-_{max})^{q+1}. \end{aligned}$$

Hence

$$|f(n)| = \left|\frac{P(n)}{Q(n)}\right| \leq \frac{(\bar{a}+n)^p}{(n-b^-_{max})^{q+1}}.$$

We consider two cases:

1. $p < q+1$. Thus

$$|xf(n)| \leq \left(1 + \frac{\bar{a}+b^-_{max}}{n-b^-_{max}}\right)^p \frac{|x|}{(n-b^-_{max})^{q+1-p}} \leq \frac{2^{q+1}|x|}{n^{q+1-p}} \leq 1/2$$

provided $n \geq \max\{\bar{a} + 2\, b^-_{max}, (2^{q+2}|x|)^{\frac{1}{q+1-p}}\}$.

2. $p = q+1$. Thus

$$|xf(n)| \leq \left(1 + \frac{\bar{a}+b^-_{max}}{n-b^-_{max}}\right)^p |x| < 1$$

provided $n \geq \frac{\bar{a}+b^-_{max}}{|x|^{-(1/p)}-1} + b^-_{max}$.

Q.E.D.

This lemma implies that $|t_n|$ is monotone decreasing for $n > n_0$. For $p = q + 1$, we are unable to bound $|xf(n)|$ away from 1, and this causes additional complications for our uniform algorithm. So solve this, we next analyze the eventual behavior of $f(n)$. But we first need a lemma characterizing the polynomials $P_{\mathbf{a}}(n)$.

Let $\sigma_k(\mathbf{a})$ be the kth elementary symmetric function on $\mathbf{a}$:

$$\sigma_1(\mathbf{a}) = s_1(\mathbf{a}), \quad \sigma_2(\mathbf{a}) = \sum_{1 \le i < j \le p} a_i a_j,$$

$$\sigma_3(\mathbf{a}) = \sum_{1 \le i < j < k \le p} a_i a_j a_k, \quad \ldots, \quad \sigma_p(\mathbf{a}) = \prod_{i=1}^{p} a_i.$$

The following result is well-known:

LEMMA 3. *Let $\mathbf{a} = (a_1, \ldots, a_p)$ and $\mathbf{b} = (b_1, \ldots, b_p)$ where $a_1 \le \cdots \le a_p$ and $b_1 \le \cdots \le b_p$. The following four statements are equivalent:*
(1) $\mathbf{a} = \mathbf{b}$.
(2) $(\forall k = 1, \ldots, p)[s_k(\mathbf{a}) = s_k(\mathbf{b})]$.
(3) $(\forall k = 1, \ldots, p)[\sigma_k(\mathbf{a}) = \sigma_k(\mathbf{b})]$.
(4) The polynomials $P_{\mathbf{a}}(n)$ and $P_{\mathbf{b}}(n)$ are identical.

The following notations will be useful for describing the eventual monotonic behavior of $f(n)$: write "$f(n) \nearrow n_1$" to mean that for all $n > n_1$, $f(n) < f(n+1)$. Similarly, "$f(n) \searrow n_1$" means that for all $n > n_1$, $f(n) > f(n+1)$. In either case, we can write "$f(n) \updownarrow n_1$". We may also write "$f(n) \nearrow$" if $f(n) \nearrow n_1$ for some n_1. The notation "$f(n) \searrow$" is similar.

We now seek a value $n_1 = n_1(\mathbf{a}, \mathbf{b})$ such that $f(n) \updownarrow n_1$. The next two lemmas defines n_1 for the two cases of $p < q + 1$ and $p = q + 1$.

LEMMA 4. *Let $p < q + 1$ and $\{a_1, \ldots, a_p\} \neq \{b_0, b_1, \ldots, b_q\}$ viewed as multi-sets. Then $f(n) \searrow n_1$ where*

$$n_1 := \max\{a^+_{\max}, 2a^-_{\max}, b^-_{\max}, r\} \tag{5}$$

and

$$r := \frac{s_1(\mathbf{b}') + 2s_1^-(\mathbf{a}) - \frac{1}{2}s_1^+(\mathbf{a})}{q + 1 - p},$$

Proof. We have

$$\begin{aligned} f(n) &= \frac{\prod_{i=1}^{p}(a_i + n)}{\prod_{j=0}^{q}(b_j + n)} \\ &= \frac{\prod_{i=1}^{p}(\frac{a_i}{n} + 1)}{\prod_{j=0}^{q}(\frac{b_j}{n} + 1)} \left(\frac{1}{n}\right)^{q+1-p} \\ \log f(n) &= \sum_{i=1}^{p} \log(\frac{a_i}{n} + 1) - \sum_{j=0}^{q} \log(\frac{b_j}{n} + 1) + (q + 1 - p)\log(\frac{1}{n}). \end{aligned}$$

Let $\nu = 1/n$ and define $h(\nu)$ via

$$\begin{aligned} h(\nu) &:= \log f(1/\nu) && (6) \\ &= \sum_{i=1}^{p} \log(a_i\nu + 1) - \sum_{j=0}^{q} \log(b_j\nu + 1) + (q + 1 - p)\log\nu && (7) \\ h'(\nu) &= \sum_{i=1}^{p} \frac{a_i}{a_i\nu + 1} - \sum_{j=0}^{q} \frac{b_j}{b_j\nu + 1} + \frac{q + 1 - p}{\nu}. && (8) \end{aligned}$$

From the bounds,

$$\begin{aligned}
\frac{a_i}{a_i\nu+1} &> \frac{a_i}{2} \quad \text{for} \quad a_i>0, n>a^+_{max},\\
\frac{a_i}{a_i\nu+1} &> 2a_i \quad \text{for} \quad a_i<0, n>2\,a^-_{max},\\
\frac{b_j}{b_j\nu+1} &< b_j \quad \text{for} \quad b_i>0, n>0,\\
\frac{b_j}{b_j\nu+1} &< b_j \quad \text{for} \quad b_i<0, n>b^-_{max},
\end{aligned}$$

we infer

$$h'(\nu) \geq \frac{1}{2}s_1^+(\mathbf{a}) - 2s_1^-(\mathbf{a}) - s_1(\mathbf{b}') + \frac{q+1-p}{\nu},$$

provided $n > \max\{a^+_{\max}, 2a^-_{\max}, b^-_{\max}\}$. Hence, if we choose

$$n_1 = \max\{a^+_{\max}, 2a^-_{\max}, b^-_{\max}, r\},$$

we conclude that $h'(\nu) > 0$. This means as $h(\nu)$ increases with ν. But as $n \to \infty$, ν decreases towards 0 and so $h(\nu)$ is decreasing, i.e., $f(n) \searrow$ as claimed. **Q.E.D.**

LEMMA 5. *Let $p = q+1$ and $\{a_1, \ldots, a_p\} \neq \{b_0, b_1, \ldots, b_q\}$ viewed as multi-sets. Then there exists a smallest index $k = 1, \ldots, p$ such that $s_k(\mathbf{a}) \neq s_k(\mathbf{b}')$.*
(a) We have $f(n) \searrow$ if $(-1)^k(s_k(\mathbf{b}') - s_k(\mathbf{a})) > 0$, and otherwise $f(n) \nearrow$.
(b) We have $f(n) \updownarrow n_1$ where

$$n_1 := \begin{cases} \max\left\{a_{max}, b_{max}, \frac{b^+_{max} + b^-_{max}\, r_0}{r_0 - 1}\right\} & \text{if } \ s_k(\mathbf{b}') > s_k(\mathbf{a}),\\ \max\left\{a_{max}, b_{max}, \frac{a^-_{max} + a^+_{max}\, r_0}{1 - r_0}\right\} & \text{if } \ s_k(\mathbf{b}') < s_k(\mathbf{a}). \end{cases} \tag{9}$$

where

$$r_0 := \sqrt[k]{\frac{s_k^+(\mathbf{b}') + s_k^-(\mathbf{a})}{s_k^+(\mathbf{a}) + s_k^-(\mathbf{b}')}}. \tag{10}$$

Proof. The existence of k comes from lemma 3.
(a) Let $h(\nu) := \log f(1/\nu)$ where $\nu = 1/n$, as in the previous proof. Then

$$h'(\nu) = \sum_{i=1}^{p} \frac{a_i}{a_i\nu+1} - \sum_{j=1}^{q} \frac{b_j}{b_j\nu+1}.$$

In general, for $r \geq 1$, the rth derivative is

$$h^{(r)}(\nu) = (r-1)!(-1)^r \left(\sum_{j=0}^{q} \left(\frac{b_j}{b_j\nu+1} \right)^r - \sum_{i=1}^{p} \left(\frac{a_i}{a_i\nu+1} \right)^r \right).$$

Also, $h^{(0)}(\nu)$ is simply $h(\nu)$. Evaluating at $\nu = 0$, we get

$$\begin{aligned}
h^{(r)}(\nu)\Big|_{\nu=0} &= (r-1)!(-1)^r \left(\sum_{j=0}^{q} b_j^r - \sum_{i=1}^{p} a_i^r \right)\\
&= (r-1)!(-1)^r \left(s_r(\mathbf{b}') - s_r(\mathbf{a}) \right).
\end{aligned}$$

By our choice of index k, we have $h^{(r)}(\nu)\big|_{\nu=0} = 0$ for $r = 1, \ldots, k-1$ and $h^{(k)}(\nu)\big|_{\nu=0} \neq 0$. So for $\varepsilon > 0$ small enough, $h(\varepsilon)$ and $h'(\varepsilon)$ has the sign of $h^{(k)}(\nu)\big|_{\nu=0}$. As in the proof of Lemma 4, $h'(\varepsilon) > 0$ means that $f(n) \searrow$. This proves $f(n) \searrow$ iff $(-1)^k \left(s_k(\mathbf{b}') - s_k(\mathbf{a}) \right) > 0$.

(b) Write

$$\begin{aligned} S_k(\nu) &:= \sum_{j=0}^{q}\left(\frac{b_j}{b_j\nu+1}\right)^k - \sum_{i=1}^{p}\left(\frac{a_i}{a_i\nu+1}\right)^k \\ &= \sum_{j=0}^{q}\left(\frac{b_j^+}{1+b_j^+\nu}\right)^k + \sum_{j=0}^{q}\left(\frac{-b_j^-}{1-b_j^-\nu}\right)^k \\ &\quad - \sum_{i=1}^{p}\left(\frac{a_i^+}{1+a_i^+\nu}\right)^k - \sum_{i=1}^{p}\left(\frac{-a_i^-}{1-a_i^-\nu}\right)^k \end{aligned} \tag{11}$$

where

$$b_j^+ = \begin{cases} b_j & \text{if } b_j > 0 \\ 0 & \text{else.} \end{cases}, \qquad b_j^- = \begin{cases} -b_j & \text{if } b_j < 0 \\ 0 & \text{else.} \end{cases}$$

The definitions of a_i^+, a_i^- is similar. We see that $f(n) \updownarrow n_1$ if $S_k(\nu) = S_k(1/n)$ holds a constant sign for all $n > n_1$. This constant sign is, of course, equal to $sign(h^{(k)}(0)) = (-1)^k(s_k(\mathbf{b}') - s_k(\mathbf{a}))$.

We next assume $n > \max\{a_{max}, b_{max}\}$. This implies

$$0 < \frac{1}{1+b_{max}^+\,\nu} \le \frac{1}{1+b_j^+\nu} \le \frac{1}{1-a_i^-\nu} \le \frac{1}{1-a_{max}^-\,\nu}, \tag{12}$$

$$0 < \frac{1}{1+a_{max}^+} \le \frac{1}{1+a_i^+\nu} \le \frac{1}{1-b_i^-\nu} \le \frac{1}{1-b_{max}^-\,\nu}. \tag{13}$$

We will now consider two cases.

CASE (A): $s_k(\mathbf{b}') > s_k(\mathbf{a})$. In this case,

$$\begin{aligned} s_k^+(\mathbf{b}') - s_k^-(\mathbf{b}') &> s_k^+(\mathbf{a}) - s_k^-(\mathbf{a}) \\ s_k^+(\mathbf{b}') + s_k^-(\mathbf{a}) &> s_k^+(\mathbf{a}) + s_k^-(\mathbf{b}') \\ r_0 &> 1. \end{aligned}$$

Plugging (12) and (13) into (11), we obtain:

$$\begin{aligned} S_k(\nu) &\ge \left(\frac{1}{1+b_{max}^+\,\nu}\right)^k \left(\sum_j (b_j^+)^k - \sum_i (-a_i^-)^k\right) - \left(\frac{1}{1-b_{max}^-\,\nu}\right)^k \left(\sum_i (a_i^+)^k - \sum_j (-b_j^-)^k\right) \\ &= \left(\frac{1}{1+b_{max}^+\,\nu}\right)^k \left(s_k^+(\mathbf{b}') + s_k^-(\mathbf{a})\right) - \left(\frac{1}{1-b_{max}^-\,\nu}\right)^k \left(s_k^+(\mathbf{a}) + s_k^-(\mathbf{b}')\right). \end{aligned} \tag{14}$$

Observe that the above derivation holds for all k; when k is even, the negative terms vanish. It is easy to verify from (14) that $S_k(\nu) > 0$ provided

$$\frac{s_k^+(\mathbf{b}') + s_k^-(\mathbf{a})}{s_k^+(\mathbf{a}) + s_k^-(\mathbf{b}')} > \left(\frac{1-b_{max}^-\,\nu}{1+b_{max}^+\,\nu}\right)^k = \left(\frac{n-b_{max}^-}{n+b_{max}^+}\right)^k$$

i.e.,

$$n > \frac{b_{max}^+ + b_{max}^-\,r_0}{r_0 - 1}.$$

CASE (B): $s_k(\mathbf{b}') < s_k(\mathbf{a})$. As in CASE (A), we see that $r_0 < 1$. Plugging (12) and (13) into (11), we obtain:

$$\begin{aligned} S_k(\nu) &\le \left(\frac{1}{1-a_{max}^-\,\nu}\right)^k \left(\sum_j (b_j^+)^k - \sum_i (-a_i^-)^k\right) - \left(\frac{1}{1+a_{max}^+\,\nu}\right)^k \left(\sum_i (a_i^+)^k - \sum_j (-b_j^-)^k\right) \\ &= \left(\frac{1}{1-a_{max}^-\,\nu}\right)^k \left(s_k^+(\mathbf{b}') + s_k^-(\mathbf{a})\right) - \left(\frac{1}{1+a_{max}^+\,\nu}\right)^k \left(s_k^+(\mathbf{a}) + s_k^-(\mathbf{b}')\right). \end{aligned} \tag{15}$$

Again, we verify from (15) that $S_k(\nu) < 0$ provided

$$n > \frac{a^-_{max} + a^+_{max}\, r_0}{1 - r_0}.$$

Q.E.D.

In the remainder of this section, define

$$n_2 := 1 + \max\{n_0, n_1\}. \tag{16}$$

This definition of n_2 must be refined in Section 5.

Our next goal is to define $n_3 = n_3(x, \mathbf{a}, \mathbf{b}, \ell)$ such that $|R_n| \le 2^{-\ell-1}$ for $n > n_3$. This will be given in the next two lemmas, depending on whether $p < q + 1$ or $p = q + 1$.

LEMMA 6. *If $p < q + 1$ then $|R_n| \le 2^{-\ell-1}$ for all $n > n_3$, where*

$$n_3 := \max\left\{n_2, \ell + 2 + n_2 + \lg|t_{n_2}|\right\}.$$

Proof. For $n > n_2$, we have

$$\begin{array}{rcll} |t_n| & = & |t_{n_2}| \prod_{j=0}^{n-n_2-1} |xf(n_2+j)| & \\ & \le & |t_{n_2}| 2^{-n+n_2} & \text{(Lemma 2)} \end{array}$$

So if we choose

$$n > n_3 := \{n_2, \ell + 2 + n_2 + \lg|t_{n_2}|\}$$

this would ensure $|t_n| \le 2^{-\ell-2}$. Furthermore,

$$\begin{array}{rcll} |R_n| & \le & |t_n| \sum_{i=0}^{\infty} \prod_{j=0}^{i-1} |xf(n+j)| & \\ & \le & |t_n| \sum_{i=0}^{\infty} 2^{-i} & \text{(Lemma 2)} \\ & \le & 2^{-\ell-2} \sum_{i=0}^{\infty} 2^{-i} = 2^{-\ell-1}. & \end{array}$$

Q.E.D.

Write $t_n = u_n x^n$ where

$$u_n := \frac{(a_1)_n (a_2)_n \cdots (a_p)_n}{(b_0)(b_1)_n (b_2)_n \cdots (b_q)_n} = \prod_{i=0}^{n-1} f(i).$$

LEMMA 7. *Let $p = q + 1$ and*

$$r = -\ell - 1 - \lg|u_{n_2}|, \quad s = n_2 \lg f(n_2).$$

Then we have the bound $|R_n| < 2^{-\ell-1}$ for all $n > n_3$ where

$$n_3 = \begin{cases} \max\{n_2, \frac{r + \lg(1-x)}{\lg|x|}\} & \text{if } f(n) \nearrow, \\ \max\{n_2, \frac{r+s+\lg(1-xf(n_2))}{\lg(|x|f(n_2))}\} & \text{if } f(n) \searrow. \end{cases} \tag{17}$$

Proof. Since $p = q + 1$, we have $|x| < 1$ and so $\lg|x| < 0$.
We have

$$\begin{array}{rcl} |R_n| & \le & \displaystyle\sum_{i=0}^{\infty} |t_{n+i}| \\ & \le & \displaystyle |t_n| \sum_{i=0}^{\infty} |x|^i \prod_{j=0}^{i-1} f(n+j)|. \end{array}$$

If $f(n) \nearrow n_2$, then $0 < f(n) < 1$ for $n > n_2$. This is because $\lim f(n) = 1$ as $n \to \infty$. Thus,

$$|R_n| \le |t_n| \sum_{i=0}^{\infty} |x|^i = |t_n| \frac{1}{1-|x|} < |u_{n_2}||x|^n \frac{1}{1-|x|} < 2^{-\ell-1}$$

where the last inequality requires

$$n > n_3 := \frac{-\ell - 1 - \lg|u_{n_2}| + \lg(1-x)}{\lg|x|}.$$

If $f(n) \searrow n_2$, then $f(n) > f(n+1) > 1$ for $n > n_2$. Thus,

$$\begin{aligned} |R_n| &\leq |t_n| \sum_{i=0}^{\infty} |x|^i f(n)^i = |t_n| \frac{1}{1-|x|f(n)} \\ &< |u_{n_2}||x|^n f(n_2)^{n-n_2} \frac{1}{1-|x|f(n_2)} < 2^{-\ell-1}, \end{aligned}$$

where the last inequality requires

$$n > n_3 := \frac{-\ell - 1 - \lg|u_{n_2}| + n_2 \lg f(n_2) + \lg(1-|x|f(n_2))}{\lg(|x|f(n_2))}.$$

Q.E.D.

This give us:

THEOREM 8. *The general hypergeometric function H is absolutely approximable.*

Proof. Given rational $\mathbf{a}, \mathbf{b}, x, \ell$, we successively delete pairs (a_i, b_j) where $a_i = b_j$ $(j > 0)$. Note that b_0 (the implicit lower parameter) is not available for deletion. Eventually, we may reduce p or q to 0; if $p = q = 0$, we are just computing $\exp(x)$. Otherwise, assume $\mathbf{a}, \mathbf{b}$ are distinct as multisets. We then compute an upper bound N_i for the quantity n_i (for $i = 0, 1, 2, 3$) in a straightforward way using standard algorithms. Finally, compute an $(\ell+1)$-bit approximation to S_{N_3} (e.g., using the techniques in [35]). This value approximates $H(\mathbf{a}; \mathbf{b}; x)$ to ℓ absolute bits. **Q.E.D.**

It is clear that Theorem 8 holds under much more general conditions than proved here. For instance, we can view the function H as a complex function, and the parameters $\mathbf{a}, \mathbf{b}$ can be arbitrary algebraic numbers. We may also extend this result to simple functions that are derived from H, such as in Table 1:

COROLLARY 9. *If $A(x), B(x) \in \mathbb{Q}[x]$ then the function of the form $F(x) = A(x)_p F_q(\mathbf{a}; \mathbf{b}; B(x))$ is absolutely approximable.*

REMARK. The proof of theorem 8 yields much quantitative data relating $\mathbf{a}, \mathbf{b}, x$ to convergence; this will be exploited in our complexity analysis in Section 4. Martin Ziegler pointed out to us that Theorem 4.3.11 in Weihrauch [34, p. 116] provided a uniform algorithm for absolute approximations of the more general class of complex analytic functions, $f(z) = \sum_{i \geq 0} a_i z^i$, $a_i \in \mathbb{C}$ with some radius of convergence R. The input to Weihrauch's algorithm is (f, z, r, M, ℓ), where $(z, r, M, \ell) \in \mathbb{C} \times \mathbb{Q} \times \mathbb{Q}$ and f is given by a uniform procedure to generate the coefficients $a_i \in \mathbb{C}$ $(i \geq 0)$ as blackbox numbers, and z is given as a blackbox number. The output is an ℓ-bit approximation of $f(z)$. The auxillary inputs r, M satisfy $R > r > |z|$ and $|a_i| \leq M \cdot r^{-i}$ (for $i \geq 0$). To deduce Theorem 8 from Weihrauch's theorem we can (a) show how to uniformly generate the a_i's from the hypergeometric parameters $\mathbf{a}, \mathbf{b}$, and (b) show how to compute (r, M) from $z, \mathbf{a}, \mathbf{b}$. While step (a) is straightforward, step (b) does not seem any easier than our current approach.

4 Evaluation at a Blackbox Number

This section generalizes the approximability result of Section 3 to the case where x is now an arbitrary real number represented by black-box $\underline{x} : \mathbb{N} \to \mathbb{D}$ such that $\underline{x}[p]$ is a p-bit absolute approximation to x. In [15], such a generalization was given for common elementary functions, exploiting well-known properties of such functions. We now solve this in complete generality.

We assume the usual rational parameters $\mathbf{a}, \mathbf{b}$, the precision $\ell \in \mathbb{Z}$, but $x \in \mathbb{R}$ is represented by $\underline{x}$. Our main result says that $H(\mathbf{a}; \mathbf{b}; x)$ is absolutely approximable. We make this precise with the help of oracle Turing machines [19]. In general, let $f(x_1, \ldots, x_p; y_1, \ldots, y_q)$ be a real function with its arguments separated

into two groups: intuitively, the x_i's represent base reals and y_j's represent black box inputs. We say that f is **absolutely approximable** if there is an oracle Turing machine M taking p base reals inputs $x_1, \ldots, x_p$ and ℓ, and q oracle inputs $\underline{y}_1, \ldots, \underline{y}_q$, such that in finite time, it halts with an absolute ℓ-bit approximation to $f(x_1, \ldots, x_p, y_1, \ldots, y_q)$. M invokes oracle $\underline{y}_i$ by writing a pair (i, ℓ') on a special "query tape" and entering a special query state $q_?$; in the next instant, M enters a special "answer state" $q_!$ and, instantly, some ℓ'-bit approximation $\underline{y}_i[\ell']$ appears on an answer tape for use in subsequent computation.

When $\mathbf{a}, \mathbf{b}$ are understood, we will simply write $F(x)$ and $\widetilde{F}(x)[\ell]$ for $H(\mathbf{a}; \mathbf{b}; x)$ and $\widetilde{H}(\mathbf{a}; \mathbf{b}; x; \ell)$ respectively. For any $n \geq 0$, we use the notation $F(x) = S_n(x) + R_n(x)$ where $S_n(x) := \sum_{k=0}^{n-1} t_k$ and $R_n(x) := \sum_{k \geq n} t_k$. Thus the notation $R_n(x)$ will now explicitly indicate its dependence on x.

We now need bounds that depends on x, but where x is only known to lie within an interval. Let $[\underline{u}, \overline{u}]$ be an interval, also written as $[u]$. It is useful to use the following definitions, defined for any set $S \subseteq \mathbb{R}$, not necessarily an interval. If $c \in \mathbb{R}$, we write $S > c$ if for all $x \in S$, $x > c$. Similarly for $c \leq S$, etc. Also let $|S|$ denote the set $\{|x| : x \in S\}$. If $f : \mathbb{R} \to \mathbb{R}$, then $f(S)$ is the set $\{f(x) : x \in S\}$. E.g., $|[u]| = \max\{|x| : x \in [u]\}$ and $R_n([u]) = \{R_n(x) : x \in [u]\}$.

We shall need a special condition on $[u]$:

$$\text{If } p = q + 1 \quad \text{then} \quad |[u]| < 1. \tag{18}$$

Note that we do not require $|[u]| > 0$ (cf. [14, 16]), a condition that is more difficult to ensure.

LEMMA 10. *Let $\ell \in \mathbb{Z}$ and $[u] = [\underline{u}, \overline{u}]$ where $\underline{u}, \overline{u}$ are bigfloats. If $[u]$ satisfies (18), we can compute an integer N_3 such that $|R_{N_3}([u])| < 2^{-\ell-1}$.*

Proof. The idea is that the inequalities in Stage A that involve x will now be strengthened so that they remain true for any choice of $x \in [u]$. This defines the quantities $\overline{n}_0, \overline{n}_1, \overline{n}_2, \overline{n}_3$, in analogy to n_0, n_1, n_2, n_3. Then we must show how to compute upper bounds N_i on $\overline{n}_i$ $(i = 0, \ldots, 3)$. The general strategy is to proceed as in (the proof of) Theorem 8, using $\max|[u]|$ in place of x. In the following, we note some additional detail:

(a) The general strategy works for computing N_1. But in fact, n_1 (see (5) and (9)) does not depend on x, and so $\overline{n}_1 = n_1$.

(b) To compute N_0, there are two cases (see (4)). For $p < q+1$, the general strategy works. For $p = q+1$, we only need to further ensure $\max|[u]| < 1$; this is guaranteed by our assumption (18).

(c) To compute N_3, when $p < q + 1$, Lemma 6 shows that the general strategy works. Here, we need an upper on $|t_n| = |x^n u_n|$ (with $n = n_2$). This can be obtained as $(\max|[u]|)^n |u_n|$. When $p = q+1$, (17) shows that we need (i) an upper bound on $|\lg(1 - [u]f(n_2))|$ and (ii) a non-zero lower bound on $|\lg([u]f(n_2))|$. Both (i) and (ii) only require $|[u]| < 1$, which is guaranteed by (18). **Q.E.D.**

LEMMA 11. *Given $[u]$ as in (18), we can compute an $M \in \mathbb{F}_1$ such that $|F'([u])| \leq M$ where $F'(x)$ denotes differentiation with respect to x.*

Proof. Note that $F'(x)$ is just another hypergeometric function multiplied by a rational number since

$$F'(x) = \frac{P_{\mathbf{a}}(0)}{P_{\mathbf{b}}(0)} H(\mathbf{a}+1; \mathbf{b}+1; x)$$

where $P_{\mathbf{a}}(n)$ is given by (2) and $\mathbf{a}+1 = (a_1+1, a_2+1, \ldots, a_p+1)$ (similarly for $\mathbf{b}+1$). Thus by Lemma 10, we can determine an N_3 such that truncating the series for $F'(x)$ after the first N_3 terms incurs an error of at most $1/2$. Then compute an approximation $\widetilde{S}$ of the sum of the first N_3 terms, with absolute error at most $1/2$. We may choose $M = \widetilde{S} + 1$. **Q.E.D.**

The following algorithm will absolutely approximate H relative to a blackbox number x:

BLACKBOX EVALUATION ALGORITHM

Input: Rational $\mathbf{a}, \mathbf{b}, \ell$ and $x \in \mathbb{R}$ represented by a blackbox $\underline{x}$.

Output: A bigfloat $\widetilde{y}$ such that $\widetilde{y} = H(\mathbf{a}; \mathbf{b}; x) \pm 2^{-\ell}$.

0. Initialize $s \leftarrow \ell + 1$.
1. Compute x' such that $x' = x \pm 2^{-s}$.
   This is just one call to the black box $\underline{x}$.
   Let $[u] = [\underline{u}, \overline{u}]$ where $\underline{u} = x' - 2^{-s}$ and $\overline{u} = x' + 2^{-s}$.
2. [Loop] While $[u]$ does not satisfy (18),
   keep doubling the precision s in x' until it does.
3. Compute M such that $|F'([u])| \leq M$ (cf. Lemma 11).

If $M < 1$, set $M = 1$.

4. [Repeat Step 1]
   Once more, recmpute $x' = x \pm (2^{-s}/M)$
5. Using Theorem 8, compute and return the value $y' = H(\mathbf{a}; \mathbf{b}; x') \pm 2^{-\ell-1}$.

THEOREM 12. *The Blackbox Evaluation Algorithm halts and gives a correct output:*

$$y' = H(\mathbf{a}; \mathbf{b}; x) \pm 2^{-\ell}.$$

Proof. We need to prove partial correctness (i.e., any output is correct) and halting. Halting is easy: the algorithm has one loop in Step 2, seeking to produce an interval $[u]$ satisfying (18). This loop must halt since the input is assumed to satisfy $|x| < 1$ if $p = q + 1$.

Write $F(x)$ for $H(\mathbf{a}; \mathbf{b}; x)$. Note that Step 5 of the algorithm returns $y' = F(x') \pm 2^{-\ell-1}$. So partial correctness amounts to the claim that y' is equal to $F(x) \pm 2^{-\ell}$. This claim follows if we show

$$F(x') = F(x) \pm 2^{-\ell-1}.$$

By Step 4, $x' = x \pm \delta$ where $\delta = 2^{-s}/M$ where $s \geq \ell + 1$ and $M \geq 1$. The Mean Value Theorem says

$$F(x \pm \delta) = F(x) \pm \delta \cdot F'(x \pm \delta). \tag{19}$$

According to Step 2,

$$x \pm \delta = x \pm 2^{-s}/M = x \pm 2^{-s} = x \pm 2^{-\ell-1}.$$

Therefore, by Step 3, $|F'(x \pm \delta)| \leq M$. Plugging into (19),

$$F(x') = F(x \pm \delta) = F(x) \pm \delta \cdot M = F(x) \pm 2^{-\ell-1}$$

as desired. **Q.E.D.**

The power of the above blackbox algorithm is seen in the following application.

COROLLARY 13. *Fixing x, define $G(\mathbf{a}; \mathbf{b}) := H(\mathbf{a}; \mathbf{b}; x)$. If x is any computable real number, then $G(\mathbf{a}; \mathbf{b})$ is absolutely approximable for all rational $\mathbf{a}, \mathbf{b}$.*

Computable real numbers (i.e., those with a recursive blackbox) include all algebraic numbers and most of the common constants of analysis such as π and e.

5 Complexity

We bound the complexity of approximating $H(\mathbf{a}; \mathbf{b}; x)$ to absolute ℓ-bits. This is quite involved but mostly routine book-keeping. But one interesting detail is that we need to sharpen the algorithm of Section 3 before an explicit bound is possible.

Throughout this section, we fix the the input to $\widetilde{H}$ with the usual parameters $(\mathbf{a}, \mathbf{b}, x, \ell)$ where $\mathbf{a}, \mathbf{b}, x$ are rational, ℓ an integer. We say the **size** of this input is the quadruple

$$(q', n, m, \ell)$$

where $q' = q+1$, $n = \max_{i,j}\{size(a_i), size(b_j)\}$ and $m = size(x)$. Our main result is a complexity bound as a function of the size (q', n, m, ℓ). We remark that it is also of interest to bound the complexity under the assumption that x is a bigfloat, since this allows more compact representation of x.

We will write N_i for the computed upper bound on n_i $(i = 0, 1, 2, 3)$. Although our main result concerns approximation with absolute error bounds, we will need the following useful lemma on relative error approximation:

LEMMA 14. *Let $a = \frac{N}{D}$ be a rational number and $n = size(a) = \max\{\lg|N|, \lg|D|\}$. There exist algorithms to approximate a to relative s bits of precision in $O(M(s) + \lg n)$ time.*

Proof. We represent N and D as bigfloats $\langle msb(N), N\rangle$ and $\langle msb(D), D\rangle$, then by Lemma 1, we can evaluate $\frac{N}{D}$ to relative s bits in $O(M(s) + \lg n)$ time. **Q.E.D.**

COROLLARY 15. *(i) Let a, b be rational numbers with at most n bits in their numerators and denominators. Then we can evaluate $a + b$ to relative s bits in $O(M(s) + \lg n)$ time. (ii) We can compute $s_1(\mathbf{a})$ to relative s bits in $O(q'(M(s + \lg q') + \lg n))$ time.*

Proof. (i) Evaluating $a + b$ to relative s bits can be done by first approximating a and b to relative $s + 2$ bits, truncating them to $s + 2$ bits, and then adding those values up to relative s bits. The first step takes $O(M(s) + \lg n)$ time and the addition takes $O(s + \lg n)$ time, so the total running time is $O(M(s) + \lg n)$. (ii) The complexity comes from evaluating a balanced binary tree whose leaves are $a_1, \ldots, a_p$. **Q.E.D.**

The upper bound N_0 (for n_0) in our algorithm and the complexity of computing N_0 is shown below:

LEMMA 16. *We have $N_0 \le 2^{3q'(n+m)}$. Computing N_0 takes $O(M(q) + m)$ time for the case $p < q + 1$ and $O(M(n) + \lg m + \lg q')$ time for the case of $p = q + 1$.*

Proof. Recall from (4) that

$$n_0 := \begin{cases} \max\{\bar{a} + 2\, b^-_{max}, (2^{q+2}|x|)^{\frac{1}{q+1-p}}\} & \text{if } p < q+1, \\ \frac{\bar{a}+b^-_{max}}{|x|^{-(1/p)}-1} + b^-_{max} & \text{if } p = q+1. \end{cases}$$

Consider $p < q + 1$. If $|x| \ge 1$ then $(2^{q+2}|x|)^{1/(q+1-p)} \le 2^{q+2}|x| \le 2^{q+2+m}$. The bound $(2^{q+2}|x|)^{1/q+1-p} \le 2^{q+2+m}$ still holds even if $|x| < 1$. It is enough to compute $\sqrt[q+1-p]{|x|}$ to relative q' bits, which takes $O(M(q) + m)$ time. For the case of $p = q + 1$, we have $\frac{\bar{a}}{1-\sqrt[p]{|x|}} \le 2^{n+mp}$. So $N_0 \le 2^{(n+m)(q'+2)} \le 2^{3q'(n+m)}$. We compute $\frac{1}{1-\sqrt[p]{|x|}}$ to relative n bits, which takes $O(M(n) + \lg m + \lg q')$ time. **Q.E.D.**

In the case of $p < q + 1$, the complexity of computing N_1 is easy and shown as follows:

LEMMA 17. *If $p < q + 1$, then $n_1 \le 3q'2^n$. Moreover, we can compute it in time $O(q'M(n + \lg q'))$.*

Proof. To bound n_1, we note that

$$|s_1(\mathbf{b}')|, |s_1^+(\mathbf{a})|, |s_1^-(\mathbf{a})|$$

are each bounded by $q'2^n$. Hence $|r| \le 3q'2^n$. We can compute

$$r' = \frac{\lceil s_1(\mathbf{b}')\rceil - \frac{1}{2}\lfloor s_1^+(\mathbf{a})\rfloor - 2\lfloor s_1^-(\mathbf{a})\rfloor}{q+1-p}$$

and

$$n_0 = \max\{\lceil a^+_{max}\rceil, 2\lceil a^-_{max}\rceil, \lceil b^-_{max}\rceil, r'\}.$$

Since $s_1(\mathbf{b}'), s_1^+(\mathbf{a}), s_1^-(\mathbf{a})$ has at most $(n + \lg q')$ bits, we can evaluate them to relative $(n + \lg q')$ bits to get $\lceil s_1(\mathbf{b}')\rceil, \lfloor s_1^+(\mathbf{a})\rfloor, \lfloor s_1^-(\mathbf{a})\rfloor$ in time $O(q'M(n + \lg q'))$ by Corollary 15. **Q.E.D.**

However, for the case of $p = q + 1$, we need first analyze the complexity of finding the smallest k, which is shown in the following three lemmas:

LEMMA 18. *Let $\alpha = \sqrt[k]{a} - \sqrt[k]{b}$ where $a, b > 0$ are rational numbers of size t. Then $\lg|\alpha| \geq -(2t+1)k^2$. If a, b are integers then $\lg|\alpha| \geq -(t+k)k$.*

Proof. We use the BFMSS rule [5] to compute $u(\sqrt[k]{a}) \leq 2^t$ and $\ell(\sqrt[k]{a}) \leq 2^t$, with the same bound when a is replaced by b. Hence $u(\alpha) = 2^{2t+1}$. Since the degree of α is $\leq k^2$, we get $|\alpha| \geq u(\alpha)^{-k^2}$ and $\lg|\alpha| \geq -(2t+1)k^2$. When a, b are integers, we obtain $u(\sqrt[k]{a}) \leq 2^{t/k}$ and $\ell(\sqrt[k]{a}) = 1$, giving us the improvement stated. **Q.E.D.**

COROLLARY 19. *Let $\alpha = s_k(\mathbf{b}') - s_k(\mathbf{a})$, $\beta = \sqrt[k]{s_k(\mathbf{a})} - \sqrt[k]{s_k(\mathbf{b})}$. Then $\lg|\alpha| \geq -2(kn + \lg q')$, $\lg|\beta| \geq -(2q'(kn + \lg q') + 1)k^2$.*

Proof. We note that $s_k(\mathbf{a})$ and $s_k(\mathbf{b})$ have size at most $(kn + \lg q')$. **Q.E.D.**

LEMMA 20. *Deciding the sign of $s_k(\mathbf{b}') - s_k(\mathbf{a})$ takes $O(kn + \lg q')$ time and deciding the smallest k such that $s_k(\mathbf{b}') \neq s_k(\mathbf{a})$ takes $O(q'^2 n + q' \lg q')$ time.*

Proof. We can evaluate $s_k(\mathbf{b}') - s_k(\mathbf{a})$ to absolute $2(kn + \lg q')$ bits to get the sign in time $O(kn + \lg q')$. In at most q' steps, we can find the smallest index k, which takes $O(q'^2 n + q' \lg q')$ time. **Q.E.D.**

Now we have the complexity of computing n_1:

LEMMA 21. *If $p = q + 1$, then $n_1 \leq 2^{4q'^3 n}$. Moreover, we can compute it in time $O(M(q^3 n))$.*

Proof. To see the upper bound for n_1, we note from the definition in (**??**) and (**??**) that the numerator is at most $2^{2n + \lg q' + 1}$ and the denominator is at least $2^{-q'^2(2(nq' + \lg q') + 1)}$, hence $n_1 \leq 2^{4q'^3 n}$. Then we can compute n_1 to $4q'^3 n$ relative bits which will take $O(M(q'^3 n))$ time. **Q.E.D.**

If we follow the scheme of the previous section, it remains to bound n_3. However, no finite bound is possible under that scheme. To see this, note that for $n > n_2$, although we know that $|xf(n)| < 1$, it can be arbitrarily close to 1, which implies n_3 can be arbitrarily large. Moreover, this difficulty only arises when $p = q + 1$. Hence, when $p = q + 1$, we shall modify the definition in (16).

LEMMA 22. *If $p = q + 1$, redefine $n_2 := \max\{n_0, n_1, n_2'\}$ where*

$$n_2' := \frac{a_{\max}^+ + b_{\max}^-}{\sqrt[q]{1 + 2^{-m}} - 1} + b_{\max}^-. \tag{20}$$

Then,

$$1 - |xf(n_2)| \geq 2^{-2m}.$$

and hence $-\lg(|x| f(n_2)) \leq 2m$.

LEMMA 23. *$n_2 \leq 2^{q'(7q'^2 n + 3m)}$. The computation of n_2 takes $O(q^3 n + m)$ time.*

LEMMA 24. *We have the bound*

$$\begin{aligned} n_3 &\leq 4^m \left(\ell + 1 + 2q'^2(7q'^2 n + 3m) 2^{q'(7q'^2 n + 3m)}\right) \\ &\leq 4^m \left(\ell + 2^{4q'(2q'^2 n + m)}\right) \end{aligned}$$

The computation of n_3 takes time

$$O(M(\lg(\ell) + q'^3 n + qm)).$$

Let us define

$$N = 4^m \left(\ell + 2^{4q'(2q'^2 n + m)}\right). \tag{21}$$

Now we analyze the complexity of computing the approximation $\widetilde{S}_N$ such that $|\widetilde{S}_N - S_N| \leq 2^{-\ell - 1}$ where $S_N = \sum_{k=0}^{N} t_k$. It is sufficient that for each k, $0 \leq k \leq N$, we compute an approximation $\widetilde{t}_k$ such that

$$|\widetilde{t}_k - t_k| \leq 2^{-\ell - 1 - \lg N}.$$

Noting that $t_k = u_k x^k$, $\lg|u_k| \leq 2kq' \lg(2^n + k)$ and $\lg|x^k| \leq km$, it is sufficient to compute u_k and x_k to relative $r + 2$ bits where

$$r = (\ell + 1 + \lg N + 2kq' \lg(2^n + k) + km).$$

LEMMA 25. *We can compute x^k to $r+2$ relative precision in time $O(kM(r))$.*

Proof. To compute x^k to relative $r+2$ bits, we can use two steps:

1. compute x to relative $(r+2+\lg(k+1))$ bits and truncate,
2. multiply them (in linear order or binary tree order).

Note that x is a rational number $\frac{N}{D}$ with bit length m, so the first step can be done in time $O(M(r+\lg k)+\lg m)$. For the second step, if we do the multiplication in linear order, $x^2 = x \times x$ can be done in time $O(M(r+\lg k)+\lg m)$, $x^3 = x^2 \times x$ can be done in time $O(M(r+\lg k)+\lg(2m))$, ..., $x^k = x^{k-1} \times x$ can be done in time $O(M(r+\lg k)+\lg((k-1)m))$, so the total running time is $O(kM(r+\lg k)+k\lg(km)) = O(kM(r))$. **Q.E.D.**

LEMMA 26. *We can compute u_k to $r+2$ relative precision in time $O((k+q')M(r))$.*

Proof. Computing u_k to relative $r+2$ bits can be done the following steps:

1. compute $(a_i)_k, (b_j)_k$ to relative $(r+4+\lg q')$ bits and truncate,
2. compute $P(k) = (a_1)_k(a_2)_k \cdots (a_p)_k$ to relative $(r+4)$ bits and truncate,
3. compute $Q(k) = (b_1)_k(b_2)_k \cdots (b_q)_k(1)_k$ to relative $(r+4)$ bits and truncate,
4. compute $\frac{P(k)}{Q(k)}$ to relative $r+2$ bits.

The first step can be done in time $O(kM(r+\lg q+\lg k)+k\lg(kn)) = O(kM(r))$. The second and third step can be done in time $O(q'M(r+\lg q')+kq'\lg(2^n+k)) = O(q'M(r))$, and the final step can be done in time $O(M(r)+kq'\lg(2^n+k)) = O(M(r))$. Therefore, computing u_k takes $O((k+q')M(r))$ time. **Q.E.D.**

COROLLARY 27. *The summation S_N takes $O(N^2M(\ell+q'N\lg N+Nm))$ time where N is defined in (21).*

Proof. Note that $r = O(\ell+q'N\lg N+Nm)$, so computing t_k takes at most $O(NM(\ell+q'N\lg N+Nm))$ time, hence the total running time for computing S_N is $O(N^2M(\ell+q'N\lg N+Nm))$. **Q.E.D.**

The upshot of these calculations yields:

THEOREM 28. *The general hypergeometric function H can be approximated in time that is*

$$O(N^2M(\ell+q'N\lg N+Nm))$$

where

$$N = 4^m\left(\ell + 2^{4q'(2q'^2n+m)}\right)$$

and $M(n)$ is defined as the complexity of multiplying two n-bit numbers. Thus, the uniform complexity of hypergeometric functions are polynomial in ℓ and single exponential in n, m, q'.

6 Final Remarks

Finally, we briefly discuss the main open problem arising from this work, and also mention our implementation.

The open problem is whether H is relatively approximable. Yap [35] has shown that a real function f is relatively approximable iff it is absolutely approximable and its zero problem $Zero(f)$ is decidable. Since we have shown H to be absolutely approximable, it suffices to show the decidability of the zero problem, $Zero(H)$:

Given rational $\mathbf{a}, \mathbf{b}, x$, is $H(\mathbf{a};\mathbf{b};x) = 0$?

For instance, Beukers has shown that

$$B(a) := {}_2F_1(1-3a, 3a; a; 1/2) = 2^{2-3a}\cos(\pi a). \tag{22}$$

By setting $a = 1/2$, we obtain $B(1/2) =_2 F_1(-\frac{1}{2}, \frac{3}{2}; \frac{1}{2}; \frac{1}{2}) = 0$. So the issue is to detect identities of this sort automatically. We consider two related problems:

(RD): Given rational $\mathbf{a}, \mathbf{b}, x$, is $H(\mathbf{a}; \mathbf{b}; x)$ rational?
(RE): Given rational $\mathbf{a}, \mathbf{b}, x, r$, is $H(\mathbf{a}; \mathbf{b}; x) = r$?

Clearly, $Zero(H)$ can be reduced to the special case of (RE) in which the parameters $\mathbf{a}, \mathbf{b}$ are positive, using the transformation $F = {}_pF_q(\mathbf{a}; \mathbf{b}; x) = S_n + R_n$ where $R_n = \sum_{k\geq n} t_k$, then

$$R_n = t_{np+1}F_{q+1}(\mathbf{a}+n, 1; \mathbf{b}+n, 1+n; x)$$

For instance, Beuker's example above can be transformed in this way to

$${}_2F_1(-\frac{1}{2}, \frac{3}{2}; \frac{1}{2}; \frac{1}{2}) = 1 + t_1 \cdot {}_3F_2(\frac{1}{2}, \frac{5}{2}, 1; \frac{3}{2}, 2; \frac{1}{2})$$

where $t_1 = -3/4$. Hence,

$${}_3F_2(\frac{1}{2}, \frac{5}{2}, 1; \frac{3}{2}, 2; \frac{1}{2}) = 3/4.$$

Implementation. We implemented the uniform evaluation algorithm of Section 3 in Core Library; all outputs are confirmed with Maple. We use this algorithm to approximate instances of Beuker's series: e.g., approximating $B(1/2) =_2 F_1(-1/2, 3/2; 1/2; 1/2)$ to 300 digits (996 bits) yields $b = 4.279162749215984826099513e-301$. It may be verified that $-\lg|b| = 998$. Again, for $B(1/4) = {}_2F_1(1/4, 3/4; 1/4; 1/2) = 2^{2-3/4}\cos(\pi/4) = 2^{3/4}$, we obtain $n_0 = 1, n_3 = 995$ and $B(1/4) = 1.681792830\cdots1863989$, which agrees with $2^{3/4}$ for 300 digits. We actually implemented two versions of the evaluation algorithm: the **non-progressive** version computes n_0, n_1, n_3 directly, using the lemmas above. It turns out that $n_1 = 1$ in all our examples, so we focus on n_0 and n_3 only. In the **progressive** version, we evaluate the partial sum $S_n = \sum_{k=0}^{n-1} t_k$ for increasing n. For each n, we check to see if the conditions that define n_0 and n_3 hold: for n_0, this means $|x|f(n) < 1$ for $n \geq n_0$. For n_3, this means $R_n = \sum_{k\geq n} t_k \leq 2^{-\ell-1}$ for $n \geq n_3$; we use the fact that for $n \geq n_1$, if $f(n) \nearrow$ then $|R_n| \leq \frac{|t_n|}{1-|x|}$; else if $f(n) \searrow$ then $|R_n| \leq \frac{|t_n|}{1-|x|f(n)}$ (see [15, 14]). The trade-off is that we spend more time per term but we need fewer terms in the summation; the timing below shows that the progressive version is faster. The following table gives some timing (in seconds) for approximating various elementary functions $f(x)$ to 300 absolute bits, starting at $x = x_0$ to $x = x_1$, by increments of δ_x. The corresponding timing in Maple is shown. The range of values for n_1 and n_3 (non-progressive) and $\widetilde{n}_1, \widetilde{n}_3$ (progressive) are indicated. Our implementation is straightforward, and thus there is room for optimization. Although slower than Maple, our performance seems respectable because it is likely that Maple uses well-known asymptotically fast algorithms for such elementary functions. All timings are on a Sun Blade 1000 2x750 MHz UltraSPARC III, with 2 GB memory.

$f(x)$	x_0	x_1	δ_x	Prog. Time	Non-Prog.	Maple Time	n_1	$\widetilde{n}_1$	n_3	$\widetilde{n}_3$
$\exp(x)$	0.01	3.14	0.07	0.4	1.1	0.17	1	1	$60-64$	$8-15$
$\sin(x)$	0.01	3.14	0.07	3.5	1:34.8	0.24	$1-2$	$1-2$	994	$43-106$
$\arctan(x)$	0.01	3.14	0.07	8.7	29.5	0.34	$1-150$	1	995	$65-487$
$\arcsin(x)$	0.01	1.00	0.07	24.6	1:24.1	0.18	$1-2$	1	995	$75-444$

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Massachusetts, 1974.

[2] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, New York, 1998.

[3] R. P. Brent. Fast multiple-precision evaluation of elementary functions. *J. of the ACM*, 23:242–251, 1976.

[4] R. P. Brent. Multiple-precision zero-finding methods and the complexity of elementary function evaluation. In J. F. Traub, editor, *Proc. Symp. on Analytic Computational Complexity*, pages 151–176. Academic Press, 1976.

[5] C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. In *9th ESA*, volume 2161 of *Lecture Notes in Computer Science*, pages 254–265. Springer, 2001. To appear, Algorithmica.

[6] V. V. Bytev, M. Y. Kalmykov, and B. A. Kniehl. Differential reduction of generalized hypergeometric functions in application ot Feynman diagrams: One-variable case. *arXiv:0904.0214v2[hep-th]*, Apr. 2009.

[7] E.-C. Chang, S. W. Choi, D. Kwon, H. Park, and C. Yap. Shortest paths for disc obstacles is computable. *Int'l. J. Comput. Geometry and Appl.*, 16(5-6):567–590, 2006. Special Issue of IJCGA on Geometric Constraints. (Eds. X.S. Gao and D. Michelucci).

[8] J. Choi, J. Sellen, and C. Yap. Approximate Euclidean shortest paths in 3-space. *Int'l. J. Comput. Geometry and Appl.*, 7(4):271–295, 1997. Also: 10th ACM Symp. on Comp. Geom. (1994)pp.41–48.

[9] D. V. Chudnovsky and G. V. Chudnovsky. Computer algebra in the service of mathematical physics and number theory. In D. V. Chudnovsky and R. D. Jenks, editors, *Computers in Mathematics*, pages 109–232. Marcel Dekker, Inc, New York and Basel, 1990.

[10] B. M. Cullough. Assessing the reliability of statistical software: Part I. *The American Statistician*, 52:358–366, 1998.

[11] B. M. Cullough. Assessing the reliability of statistical software: Part II. *The American Statistician*, 53:149–159, 1999.

[12] B. P. Demidovich and I. A. Maron. *Computational Mathematics*. MIR Publishers, Moscow, 1976. Translated from Russian by G.Yankovsky.

[13] M. Dhiflaoui, S. Funke, C. Kwappik, K. Mehlhorn, M. Seel, E. Schömer, R. Schulte, , and D. Weber. Certifying and repairing solutions to large LPs, how good are LP-solvers? In *SODA*, pages 255–56, 2003.

[14] Z. Du. *Guaranteed Precision for Transcendental and Algebraic Computation made Easy*. Ph.D. thesis, New York University, Department of Computer Science, Courant Institute, May 2006. Download from `http://cs.nyu.edu/exact/doc/`.

[15] Z. Du, M. Eleftheriou, J. Moreira, and C. Yap. Hypergeometric functions in exact geometric computation. In V.Brattka, M.Schoeder, and K.Weihrauch, editors, *Proc. 5th Workshop on Computability and Complexity in Analysis*, pages 55–66, 2002. Malaga, Spain, July 12-13, 2002. In Electronic Notes in Theoretical Computer Science, 66:1 (2002), `http://www.elsevier.nl/locate/entcs/volume66.html`.

[16] Z. Du and C. Yap. Uniform complexity of approximating hypergeometric functions with absolute error. In S. Pae and H. Park, editors, *Proc. 7th Asian Symp. on Computer Math. (ASCM 2005)*, pages 246–249, 2006. Korea Institute for Advanced Study, Seoul, Dec 8–10, 2005.

[17] E.D.Rainville. ??? *Bull. Amer. Math. Soc.*, 51:714, 1945.

[18] V. Karamcheti, C. Li, I. Pechtchanski, and C. Yap. A Core library for robust numerical and geometric computation. In *15th ACM Symp. Computational Geometry*, pages 351–359, 1999.

[19] K.-I. Ko. *Complexity Theory of Real Functions*. Progress in Theoretical Computer Science. Birkhäuser, Boston, 1991.

[20] H. Koch, A. Schenkel, and P. Wittwer. Computer-assisted proofs in analysis and programming in logic: A case study. *SIAM Review*, 38(4):565–604, 1996.

[21] V. Lefèvre, J.-M. Muller, and A. Tisserand. Towards correctly rounded transcendentals. *IEEE Trans. Computers*, 47(11):1235–1243, 1998.

[22] D. Lozier and F. Olver. Numerical evaluation of special functions. In W. Gautschi, editor, *Mathematics of Computation 1943–1993: A Half-Century of Computational Mathematics*, volume 48, pages 79–125. American Math. Soc., Providence, Rhode Island, 1994 and 2000. Proceedings of Symposia in Applied Mathematics. Updated version (2000) available from http://math.nist.gov/nesf/.

[23] Y. L. Luke. *Algorithms for the Computation of Mathematical Functions*. Academic Press, 1977.

[24] J.-M. Muller. *Elementary Functions: Algorithms and Implementation*. Birkhäuser, Boston, 1997.

[25] M. Nardin, W. Perger, and A. Bhalla. Algorithm 707, CONHYP: A numerical evaluator of the confluent hypergeometric function for complex arguments of large magnitudes. *ACM Trans. Math. Softw.*, 18(3):345–349, 1992.

[26] M. Nardin, W. Perger, and A. Bhalla. Numerical evaluation of the confluent hypergeometric function for complex arguments of large magnitudes. *J. Computational and Applied Math.*, 39:193–200, 1992.

[27] A. Schönhage. Storage modification machines. *SIAM J. Computing*, 9:490–508, 1980.

[28] S. Smale. Some remarks on the foundations of numerical analysis. *SIAM Review*, 32:211–220, 1990.

[29] D. Tulone, C. Yap, and C. Li. Randomized zero testing of radical expressions and elementary geometry theorem proving. In J. Richter-Gebert and D. Wang, editors, *Proc. 3rd Int'l. Workshop on Automated Deduction in Geometry (ADG 2000)*, volume 2061 of *Lecture Notes in Artificial Intelligence*, pages 58–82. Springer, 2001. Zurich, Switzerland.

[30] C. Ullrich, editor. *Computer Arithmetic and Self-validating Numerical Methods*. Academic Press, Boston, 1990.

[31] J. van der Hoeven. Fast evaluation of holonomic functions. *Theor. Comput. Sci.*, 210(1):199–215, 1999.

[32] J. van der Hoeven. Fast evaluation of holonomic functions near and in regular singularities. *J. Symb. Comput.*, 31(6):717–743, 2001.

[33] J. van der Hoeven. Efficient accelero-summation of holonomic functions, 2005. Preprint.

[34] K. Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.

[35] C. K. Yap. On guaranteed accuracy computation. In F. Chen and D. Wang, editors, *Geometric Computation*, chapter 12, pages 322–373. World Scientific Publishing Co., Singapore, 2004.

[36] C. K. Yap. Theory of real computation according to EGC. In P. Hertling, C. Hoffmann, W. Luther, and N.Revol, editors, *Reliable Implementation of Real Number Algorithms: Theory and Practice*, number 5045 in Lecture Notes in Computer Science, pages 193–237. Springer, 2008.