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, yap}@cs.nyu.edu

May 5, 2006

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 $\mathbf{b} = (b_1, \ldots, b_q)$ where $p, q \in \mathbb{N}$ and $p \le q + 1$. 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 ${}_pF_q$. 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 [18, 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" where relative error is used in place of absolute error. It is conceivable that each hypergeometric function

*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	$-x^2$

${}_pF_q(\mathbf{a};\mathbf{b};x)$ 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. 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., [9, 17]). 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** [25]; following [29], 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 [29]). This has given rise to the "Table Maker's Dilemma" [16], described as the problem of computing correctly rounded values of (transcendental) functions. This problem was solved in [16] for some elementary functions in the context of double-precision format. Another example is Nardin et al [21, 20], 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 [13]. 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 [12] and also geometric theorem proving [24]. More generally, see [15] for computer-assisted theorem proving as well as automatic theorem proving. McCullough [7, 8] 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 [10] describes an application for certifying large LP solvers.

Closely related to this paper, van der Hoeven [26, 27] 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 direct predecessor of the present paper is [12], 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 [19], preprocessing of hypergeometric parameters, and use of precomputed constants. The approximation algorithm of this paper follows the basic strategy in [12], 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 [5]. (c) In order to achieve guaranteed precision under argument reduction, we must bound the sensitivity of the function to variations in its argument. While [12] 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. 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 [1, 23], but it is also the de facto model in theoretical algorithms) and the **analytic approach** (represented by Weihrauh [28] or Ko [14]). Motivated by current software and research in Exact Geometric Computation (EGC) we propose in [29] a weaker form of computing real function than in the analytic approach. We call this the **approximation approach**.

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) $\widetilde{x} : \mathbb{N} \to \mathbb{Q}$ where each $\widetilde{x}(i)$ is an ***i*-bit absolute approximation** of x — this simply means $|\widetilde{x}(i) - x| \leq 2^{-i}$. Turing machines may be modified in one of two ways to accept such inputs: Weihrauch [28] introduces TTE machines, but Ko [14] 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 a 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 [29]. 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 [29]. 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**. This set satisfies 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 $Zero(f)$ 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 $\mathbb{F} = \mathbb{D}$, then many important results about the complexity of approximability were obtained three decades ago by Brent (e.g., [2, 3]). 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.

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 [19, 12], (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 [11]. In particular, description of our implementation in the Core Library may be found there.

2 Preliminaries: Representation and Complexity Model

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 [2]. 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 [2] are of this sort.
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 $\in G$ has an ℓ-bit parameter. Here, 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 $\lg_1 x := \max\{1, \log_2 |x|\}$ (to take 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 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 restricted rationals, it may be exponentially more "compact" in representation.

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 [22, 29]. Such pointer machines can simulate a multitape Turing machine in linear time, and Schönhage has shown $M(n) = O(n)$ in this model.

Brent [2] 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. The following results are from [6, Sect. 4 and Appendix] applies 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.

3 The Uniform Evaluation Algorithm

In this section, we present an algorithm for approximating $H(\mathbf{a};\mathbf{b};x)$. 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)). The algorithm proceeds in two stages: in STAGE A, we compute an n_3 such that $|R_n| \le 2^{-\ell-1}$ for all $n > n_3$. In STAGE B, we compute an approximation $\widetilde{S}_n$ such that $|\widetilde{S}_n - S_n| \le 2^{-\ell-1}$. Note that STAGE B is clearly computable – in fact, computing this approximation is trivial to implement in Core Library. It remains to prove that STAGE A is computable. As noted, this general approach is from [12]. Also, [12] gives a more efficient treatment of alternating series; for simplicity, this refinement is omitted in the present paper.

STAGE A. We introduce some notations: if $\mathbf{a} = (a_1,\ldots,a_p)$, we write $s_k^+(\mathbf{a})$ for the sum $\sum_{i:a_i>0} a_i^k$ which range over positive elements in $\mathbf{a}$; similarly, $s_k^-(\mathbf{a}) := \sum_{i:a_i<0} a_i^k$. Let $s_k(\mathbf{a}) := s_k^+(\mathbf{a}) + s_k^-(\mathbf{a}) = \sum_{i=1}^p a_i^k$. 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 $H(\mathbf{a};\mathbf{b};x)$ can be written as $t_{n+1} = t_n f(n) x$ where

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

and $\mathbf{b}' = (b_0,\mathbf{b}) = (b_0,b_1,\ldots,b_q)$ and $b_0 = 1$ is the implicit lower parameter. As a first step towards bounding $R_n = \sum_{k=n}^{\infty} t_k$, we want $|t_k|$ to be decreasing in k. This is ensured by the following:

Lemma 2. *$|x|f(n) < 1$ holds for $n \ge n_0$ where*

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

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) $(\forall k = 1,\ldots,p)[s_k(\mathbf{a}) = s_k(\mathbf{b})]$.
(2) $(\forall k = 1,\ldots,p)[\sigma_k(\mathbf{a}) = \sigma_k(\mathbf{b})]$.
(3) The polynomials $P_{\mathbf{a}}(n)$ and $P_{\mathbf{b}}(n)$ are equal.
(4) $\mathbf{a} = \mathbf{b}$.

We need to determine a value n_1 beyond which $f(n)$ is monotone increasing or decreasing. 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. The following two lemmas gives a bound on 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. Denote by*

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

then $f(n) \searrow n_1$ where

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

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) Denoting by

$$r_1 := \sqrt[k]{\frac{s_k(\mathbf{b}')}{s_k(\mathbf{a})}}, \quad r_2 := \sqrt[k]{\frac{s_k^+(\mathbf{b}') - s_k^-(\mathbf{a})}{s_k^+(\mathbf{a}) - s_k^-(\mathbf{b}')}},$$

we have:
(i) Assume k is even. Then $f(n) \searrow n_1$ provided that $s_k(\mathbf{b}') > s_k(\mathbf{a})$ and

$$n_1 = \max\left\{a_{max}, b_{max}, b_{max}\frac{1}{r_1 - 1}\right\}. \tag{6}$$

Then $f(n) \nearrow n_1$ provided that $s_k(\mathbf{b}') < s_k(\mathbf{a})$ and

$$n_1 = \max\left\{a_{max}, b_{max}, a_{max}\frac{r_1}{r_1 - 1}\right\}. \tag{7}$$

(ii) Assume k is odd. Then $f(n) \nearrow n_1$ provided that $s_k(\mathbf{b}') > s_k(\mathbf{a})$ and

$$n_1 = \max\left\{a_{max}, b_{max}, \frac{b^+_{max} + b^-_{max}r_2}{r_2 - 1}\right\}. \tag{8}$$

Then $f(n) \searrow n_1$ provided that $s_k(\mathbf{b}') < s_k(\mathbf{a})$ and

$$n_1 = \max\left\{a_{max}, b_{max}, \frac{a^+_{max}r_2 + a^-_{max}}{1 - r_2}\right\}. \tag{9}$$

Note that r_2 is only used when k is odd; in this case, r_2 is real. Let us further define

$$n_2 := \max\{n_0, n_1\} \tag{10}$$

for the remainder of this section (this definition of n_2 will be refined in Section 5). Write $t_n = u_n x^n$ where

$$u_n := \frac{(a_1)_n (a_2)_n \cdots (a_p)_n}{(b_1)_n (b_2)_n \cdots (b_q)_n} \frac{1}{n!},$$

LEMMA 6. *Let*

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

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

$$n_3 = \begin{cases} \max\{n_2, \frac{r}{\lg|x|}\} & \text{if } x \le 0 \text{ and } f(n) \nearrow, \\ \max\{n_2, \frac{r+s}{\lg(|x|f(n_2))}\} & \text{if } x \le 0 \text{ and } f(n) \searrow, \\ \max\{n_2, \frac{r+\lg(1-x)}{\lg|x|}\} & \text{if } x \ge 0 \text{ and } f(n) \nearrow, \\ \max\{n_2, \frac{r+s+\lg(1-xf(n_2))}{\lg(|x|f(n_2))}\} & \text{if } x \ge 0 \text{ and } f(n) \searrow. \end{cases} \tag{11}$$

This give us:

THEOREM 7. *The general hypergeometric function H is absolutely approximable.*

Proof. Given rational $\mathbf{a}, \mathbf{b}, x, \ell$, we first delete pairs (a_i, b_j) where $a_i = b_j$. This 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 $n_3 = n_3(\ell)$ as given by our lemmas. We then compute an approximation $\widetilde{S_{n_3}}$ which approximates S_{n_3} to $\ell + 1$ absolute bits (e.g., [29]). This value also approximates $H(\mathbf{a}; \mathbf{b}; x)$ to ℓ absolute bits. **Q.E.D.**

It is clear that this approximation result holds even when we view the function H as complex. We may also extend this result to simple functions that are derived from H (cf. Table 1):

COROLLARY 8. *If $A(x), B(x) \in \mathbb{Z}[x]$ then the function of the form $F(x) = A(x)_p F_q(\mathbf{a}; \mathbf{b}; B(x))$ is absolutely approximable.*

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 $\widetilde{x} : \mathbb{N} \to \mathbb{D}$ such that $\widetilde{x}[p]$ is a p-bit absolute approximation to x. In [12], 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 $\widetilde{x}$. 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 [14]. 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 $\widetilde{y}_1, \ldots, \widetilde{y}_q$, 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 $\widetilde{y}_i$ by writing a pair (i, ℓ') on a special tape and entering a special state $q_?$; in the next instant, M enters a special state $q_!$ and the ℓ'-bit approximation $\widetilde{y}_i[\ell']$ appears on a special answer tape.

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 $n \geq 0$, let $F(x) = S_n(x) + R_n(x)$ where $S_n(x) = \sum_{k=0}^{n-1} t_k$ and $R_n(x) = \sum_{k \geq n} t_k$.

Let $[u]$ denote an interval $[\underline{u}, \overline{u}]$ where $\underline{u}, \overline{u}$ are bigfloats. Let $|[u]|$ denote $\max\{|x| : x \in [u]\}$. We shall assume that

$$\begin{cases} \text{If } p = q + 1 \text{ then } |[u]| < 1; \\ \text{If } p < q + 1 \text{ then } |[u]| f(n) < 1. \end{cases} \tag{12}$$

LEMMA 9. *Given $[u]$ satisfying (12), we can compute $\overline{n}_3$ such that $|R_{\overline{n}_3}([u])| < 2^{-\ell-1}$.*

Proof. All the inequalities required in STAGE A can be extended by replacing x by $[u]$, unless $0 \in [u]$. To see this, suppose we want to compute $\overline{n}_0, \overline{n}_1, \overline{n}_3$, in analogy to the n_0, n_1, n_3 given by the above lemmas. (Define $\overline{n}_2 = \max\{\overline{n}_0, \overline{n}_1\}$ as before.) Note that the definition of n_1 does not depend on x, so $\overline{n}_1 = n_1$. But in computing $\overline{n}_0$ we only need an upper bound on $|x|$, and this upper bound must be less than 1 in case of $p = q+1$. So this case is OK. For $\overline{n}_3$, we do not know the sign of x in case $0 \in [u]$. Let the expressions for n_3 in the four cases in Lemma 6 be denoted e_1, e_2, e_3, e_4. If $f(n) \nearrow$, then we choose $\overline{n}_3 = \max\{e_1, e_3\}$, otherwise we choose $\overline{n}_3 = \max\{e_2, e_4\}$. Since we only have the interval $[u]$, we compute e_1, e_2, e_3, e_4 by replacing x in the denominator by $|[u]|$, and replacing x in the numerator by 0. For e_1, e_3, this denominator replacement is valid as long as $|[u]|$ (so that we do not change the sign of the expression). For e_2, e_4, this denominator replacement is valid as long as $|[u]| f(n) < 1$. **Q.E.D.**

LEMMA 10. *Given $[u]$ as in (12), 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). Thus by Lemma 9, we can determine an $\overline{n}_3$ such that truncating the series for $F'(x)$ after the first $\overline{n}_3$ terms incurs an error of at most 1/2. Then compute an approximation $\widetilde{S}$ of the sum of the first $\overline{n}_3$ terms, error at most 1/2. Take $M = \widetilde{S} + 1$. **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 $\widetilde{x}$.

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

0. Initialize $s \leftarrow \ell + 1$.
1. Compute x' such that $|x' - x| < 2^{-s}$.
   This is just one call to the black box $\widetilde{x}$.
   Let $[u] = [\underline{u}, \overline{u}]$ where $\underline{u} = x' - 2^{-s}$ and $\overline{u} = x' + 2^{-s}$.
2. While $[u]$ does not satisfy (12),
   keep doubling the precision s in x' until it does.
3. Using Lemma 10, compute M such that $|F'([u])| < M$.
4. [Repeat Step 1]
   Recompute x' such that $|x' - x| < 2^{-s-\max\{0, \lg M\}}$
5. Using Theorem 7, compute and return the value $y' = \widetilde{H}(\mathbf{a}; \mathbf{b}; x')[\ell + 1]$.

THEOREM 11. *The Blackbox Evaluation Algorithm halts and gives a correct output:*

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

Proof. Halting of the algorithm is guaranteed because of the assumption that $|x| < 1$ if $p = q + 1$. Next, we must show that $|y' - F(x)| \leq 2^{-\ell}$. But Step 5 implies $|y' - F(x')| \leq 2^{-\ell-1}$. Hence it suffices to show that

$$|F(x') - F(x)| \leq 2^{-\ell-1}. \tag{13}$$

By Step 4, $x' = x + \delta$ where $|\delta| \leq 2^{-s}/\max\{1, M\} \leq 2^{-\ell-1}/M$. Therefore

$$|F(x') - F(x)| = |F(x + \delta) - F(x)| \leq |\delta \cdot F'(x + \theta\delta)|$$

for some $0 \leq \theta \leq 1$ (intermediate value theorem). According to Step 2,

$$x + \theta\delta \in [u] = [x' - 2^{-s}, x' + 2^{-s}] = [x + \delta - 2^{-s}, x + \delta + 2^{-s}] \subseteq [x - 2^{-\ell}, x + 2^{-\ell}],$$

and thus $|F'(x + \theta\delta)| \leq M$ by Step 3. Hence

$$|F(x') - F(x)| \leq |\delta \cdot F'(x + \theta\delta)| \leq (2^{-\ell-1}/M)(M) = 2^{-\ell-1}.$$

Q.E.D.

The power of the above blackbox algorithm is seen in the following application. Recall that $x \in \mathbb{R}$ is a computable real if it has a recursive blackbox representation $\widetilde{x}$ ([14]).

COROLLARY 12. *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 $\mathbf{a}, \mathbf{b}$.*

The approximable real numbers include all algebraic numbers and most of the common constants of analysis such as π and e. They are also closed under rational operations.

5 Complexity

We bound the complexity of approximating $H(\mathbf{a}; \mathbf{b}; x)$. This is quite involved, and we must further 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 $(\mathbf{a}, \mathbf{b}, x, \ell)$ (all rationals). 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 $m = size(x)$.

Lemma 13. *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 can express N and D by bigfloats $\langle msb(N), N\rangle$ and $\langle msb(D), D\rangle$, then by Lemma 1, we can evaluate $\frac{N}{D}$ to s bits in $O(M(s) + \lg n)$ time. **Q.E.D.**

Corollary 14. *(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 $O(q'(M(s + \lg q') + \lg n))$ or $O(q'(M(s + q') + \lg n))$ 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 first complexity comes from evaluating a balanced binary tree whose leaves are $a_1, \ldots, a_p$. The second comes from evaluating a linear expression. **Q.E.D.**

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

Lemma 15. *We have $n_0 \le 2^{3q'(n+m)}$. Computing n_1 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. We note that $(2^p|x|)^{1/(q+1-p)} \le 2^p|x| \le 2^{n+m}$ and $\frac{\bar{a}}{1-\sqrt[p]{|x|}} \le 2^{n+mp}$, so $n_0 \le 2^{(n+m)(q'+2)} \le 2^{3q'(n+m)}$. For the case of $p < q + 1$, 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 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 16. *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 14. **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 17. *Let $\alpha = \sqrt[k]{a} - \sqrt[k]{b}$ where $a, b > 0$ are rational numbers of size t. Then $\lg|\alpha| \ge -(2t + 1)k^2$. If a, b are integers then $\lg|\alpha| \ge -(t + k)k$.*

Proof. We use the BFMSS rule [4] to compute $u(\sqrt[k]{a}) \le 2^t$ and $\ell(\sqrt[k]{a}) \le 2^t$, with the same bound when a is replaced by b. Hence $u(\alpha) = 2^{2t+1}$. Since the degree of α is $\le k^2$, we get $|\alpha| \ge u(\alpha)^{-k^2}$ and $\lg|\alpha| \ge -(2t + 1)k^2$. When a, b are integers, we obtain $u(\sqrt[k]{a}) \le 2^{t/k}$ and $\ell(\sqrt[k]{a}) = 1$, giving us the improvement stated. **Q.E.D.**

Corollary 18. *Let $\alpha = s_k(\mathbf{b}') - s_k(\mathbf{a})$, $\beta = \sqrt[k]{s_k(\mathbf{a})} - \sqrt[k]{s_k(\mathbf{b})}$. Then $\lg|\alpha| \ge -2(kn + \lg q')$, $\lg|\beta| \ge -(2q'(kn + \lg q') + 1)k^2$.*

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

LEMMA 19. *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 20. *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 (8) and (9) 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 (10).

LEMMA 21. *If $p = q + 1$, redefine $n_2 := \max\{n_0, n_1, n_2'\}$ where*

$$n_2' := \frac{a^+_{\max} + b^-_{\max}}{\sqrt[q]{1 + 2^{-m}} - 1} + b^-_{\max}. \tag{14}$$

Then,

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

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

LEMMA 22. *$n_2 \leq 2^{q'(7q'^2 n + 3m)}$. The computation of n_2 takes $O(q^3 n + m)$ time.*

LEMMA 23. *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

$$N = 4^m \left(\ell + 2^{4q'(2q'^2 n+m)}\right). \tag{15}$$

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 24. *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 $O(M(r+\lg k)+\lg(2m)), \ldots, x^k = x^{k-1} \times x$ 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 25. *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 26. *The summation S_N takes $O(N^2M(\ell + q'N\lg N + Nm))$ time where N is defined in (15).*

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 27. *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 [29] 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

$$B(a) := {}_2F_1(1-3a, 3a; a; 1/2) = 2^{2-3a}\cos(\pi a). \tag{16}$$

By setting $a = 1/2$, we obtain $B(1/2) = {}_2F_1(-\frac{1}{2}, \frac{3}{2}; \frac{1}{2}; \frac{1}{2}) = 0$. 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

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

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 $b = 4.27916274921598482609951 3e-301$. 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 $B(1/4) = 1.681792830 \cdots 1863989$, 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 [12, 11]). 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] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation.* Springer-Verlag, New York, 1998.

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

[3] 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.

[4] C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. In *Lecture Notes in Computer Science*, pages 254–265. Springer, 2001. to appear, Algorithmica.

[5] 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.*, 2006. Accepted, Special Issue of IJCGA on Geometric Constraints. Guest Editors X.S. Gao and D. Michelucci. Extended abstract in 21st SoCG, 2005.

[6] 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.

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

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

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

[10] M. Dhiflaoui, S. Funke, C. Kwappik, K. Mehlhorn, M. Seel, E. Schmer, R. Schulte, , and D. Weber. Certifying and repairing solutions to large lps, how good are lp-solvers? In *Proc. SODA 2003 (to appear)*, 2003.

[11] 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/`.

[12] 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`.

[13] 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.

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

[15] 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.

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

[17] 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/.

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

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

[20] 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.

[21] 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.

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

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

[24] 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.

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

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

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

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

[29] 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.