Electronic Notes in Theoretical Computer Science 66 No. 1 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume66.html 12 pages

Hypergeometric Functions in Exact Geometric Computation

Zilin Du [a,1], Maria Eleftheriou [b], José E. Moreira [b], Chee Yap [a,1]

[a] *Department of Computer Science, Courant Institute, New York University, 251 Mercer Street, New York, NY 10012*

[b] *IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598*

Abstract

Most problems in computational geometry are algebraic. A general approach to address nonrobustness in such problems is Exact Geometric Computation (EGC). There are now general libraries that support EGC for the general programmer (e.g., `Core Library`, `LEDA Real`). Many applications require non-algebraic functions as well. In this paper, we describe how to provide non-algebraic functions in the context of other EGC capabilities. We implemented a multiprecision hypergeometric series package which can be used to evaluate common elementary math functions to an arbitrary precision. This can be achieved relatively easily using the `Core Library` which supports a guaranteed precision level of accuracy. We address several issues of efficiency in such a hypergeometric package: automatic error analysis, argument reduction, preprocessing of hypergeometric parameters, and precomputed constants. Some preliminary experimental results are reported.

1 Introduction

There are two basic modes of numerical computation: those that assume fixed precision and those that require arbitrarily high precision. A second dichotomy in numerical computation is based on classifying computations as either algebraic or those that go beyond algebraic. By definition, an **algebraic computation** may only use algebraic functions as primitives (for instance, $+, -, \times, \div, \sqrt{}$); thus **non-algebraic computation** would require some non-algebraic functions such as sin or log. An **algebraic problem** is one that can be solved using algebraic computations [22]. Most problems arising in computational geometry [7,1], and a large part of geometric modeling, are algebraic. One could treat algorithmic issues of non-algebraic computation

[1] Supported by NSF/ITR Grant #CCR-0082056. This paper was presented at CCA 2002, Malaga, Spain (June 12-13, 2002).

using the approach of computable analysis [19]. Instead, we attempt to generalize the stronger techniques available for numerical *algebraic* computation to non-algebraic settings.

This paper addresses numerical computation that is non-algebraic and which requires arbitrarily high precision. More precisely, we want to evaluate hypergeometric functions with **guaranteed precision**. Guaranteed precision for an algebraic setting was first proposed in [20] and embodied in the `Core Library` accuracy API [10]. The problem is that, when we incorporate non-algebraic functions into our system, it is a major open problem whether a similar guarantee can be made.

The concept of guaranteed accuracy is rooted in a general solution to the widespread problem of numerical non-robustness in geometric algorithms [18,21]. This well-known problem plagues many scientific and engineering computations. In principle, such problems can be eliminated for a large class of problems using an approach we call **exact geometric computation** (EGC) [22]. Basically, EGC amounts to computing with guaranteed accuracy. It is important to realize that guaranteed accuracy does not require "exact arithmetic" but arithmetic that has "sufficient accuracy". Techniques such as floating point filters [2,6,12], constructive root bounds [11,5,4,12], and low-degree predicates [13] are among the techniques driven by the EGC mode of computation. There are currently two general numerical libraries that support the EGC mode: `LEDA Real` [3] and the `Core Library` [10].

Many numerical computations require both algebraic and non-algebraic computations. Currently, `Core Library` and `LEDA Real` do not support non-algebraic functions. This paper takes a first step at filling this gap. We describe the design and implementation of a hypergeometric package in the `Core Library`. Most common non-algebraic functions such as $\exp(x), \log(x), \mathrm{erf}(x)$ and the trigonometric functions are hypergeometric. Indeed, all the non-algebraic mathematical functions in a standard library such as `math.h` are hypergeometric. Jeandel [9] describes a recent effort to provide hypergeometric functions in a general multiprecision number package (in this case, *gmp*). While multiprecision arithmetic is a pre-requisite for EGC, it still lacks the critical EGC capability of exact comparison.

Is EGC possible for non-algebraic computation? The design of the `Core Library` aims to make robust programs easily constructed by any programmer. Towards this end, we define in [20] a natural and simple **numerical accuracy API** with four accuracy levels:

- Level I: Machine Accuracy (i.e., IEEE 754 Standard)
- Level II: Arbitrary Accuracy (e.g., compute to 1000 bits)
- Level III: Guaranteed Accuracy (e.g., guarantee 100 bits)
- Level IV: Mixed Accuracy (i.e., combinations of 3 previous levels)

The goal is to allow a single program to be run in any of these levels, just by calling the library. Level II has no guarantees: computing to 1000 bits do not

guarantee that any fraction of 1000 bits are correct because of error propagation. There is a fundamental gap between Levels II and III that may not be apparent: Level III is more than simply iterating a Level II computation with increasing precision. While we know how to provide Level III capability for all algebraic computation [21], it is an open question whether Level III is possible in the non-algebraic case. We call this the **fundamental problem** of EGC. Let Ω be a set of real or complex functions or constants. In practice, Ω contains $+, -, \times, n \in \mathbb{Z}$ are among its operators. The set of constant expressions over Ω is denoted $E(\Omega)$. The **value** of an expression $e \in E(\Omega)$ is a real (or complex) number, but it may also be undefined. The **Constant Zero Problem** for Ω, denoted CZP(Ω), is to decide for a given $e \in E(\Omega)$, whether the value of e is defined and equal to 0. When Ω contains at least one non-algebraic function such as $\sin x$ or $\log x$, the Constant Zero Problem for Ω is not known to be decidable, and closely related to undecidable ones [14]. Yet, this is precisely what we need for guaranteed accuracy. So the fundamental problem of EGC is really a family of problems, CZP(Ω) for each Ω.

Contributions of This Paper.

- We describe the design and implementation of a multiprecision hypergeometric library. This is easily implemented with Level III accuracy of the `Core Library`.
- This paper introduces the problem of processing of hypergeometric parameters for efficient evaluation of hypergeometric functions.
- We introduce techniques for automatic error analysis of hypergeometric functions.
- We address the problem of argument reduction.
- A natural application of the library is in computing mathematical constants to arbitrary accuracy. This is both an application *of* the hypergeometric package, as well as an application *to* the package. We also define file formats for storing and accessing these constants.

2 Hypergeometric Series

A hypergeometric series $\sum_{k=0}^{\infty} t_k$ is one in which $t_0 = 1$ and the ratio of two consecutive terms is a rational function of the summation index k

$$\frac{t_{k+1}}{t_k} = \frac{P(k)}{Q(k)}x, \tag{1}$$

where $P(k)$ and $Q(k)$ are monic polynomials in k, and x is a constant called the **argument**. By factoring the polynomials $P(k)$ and $Q(k)$ we can write

$$\frac{t_{k+1}}{t_k} = \frac{P(k)}{Q(k)}x = \frac{(a_1+k)(a_2+k)\cdots(a_p+k)}{(b_1+k)(b_2+k)\cdots(b_q+k)(k+1)}x. \tag{2}$$

The rising factorial or Pochhammer symbol $(a)_k$ is given by $(a)_k = a(a+1)(a+2)\cdots(a+k-1)$ for $k \geq 1$ and $(a)_0 = 1$. Using this symbol, the general expression for term t_k becomes

$$t_k = \frac{(a_1)_k (a_2)_k \cdots (a_p)_k}{(b_1)_k (b_2)_k \cdots (b_q)_k} \frac{x^k}{k!}. \tag{3}$$

Thus, a hypergeometric series is completely defined by the sequences $a = (a_1, a_2, \ldots, a_p)$ and $b = (b_1, b_2, \ldots, b_q)$. The b_i's may not be zero or negative integers (otherwise we will have a division by 0). A negative a_i, on the other hand, turns $\sum_{k\geq 0} t_k$ into a finite series and hence a polynomial in x. Note that the conventional factor of $k!$ in the denominator of t_k in (3) amounts to an *implicit* lower parameter of $b_0 = 1$. The hypergeometric series corresponding to these parameters is denoted

$${}_pF_q(a_1, a_2, ..., a_p; b_1, b_1, b_2, ..., b_q; x) = \sum_{k=0}^{\infty} \frac{(a_1)_k (a_2)_k \cdots (a_p)_k}{(b_1)_k (b_2)_k \cdots (b_q)_k} \frac{x^k}{k!}, \tag{4}$$

The above series converges for any complex x when $p \leq q$, and for $|x| < 1$ when $p = q + 1$. The corresponding complex function is a **hypergeometric functions**. In this paper, we are only interested in the case where x as well as the parameters a_i, b_j are real.

Elementary Functions. Most common elementary functions [16] are hypergeometric functions. For example, the usual series for $\exp(x)$ is $\sum_{k=0}^{\infty} \frac{x^k}{k!}$ with the initial term $t_0 = 1$. The ratio between two consecutive terms is $t_{k+1}/t_k = x/(k+1)$. From Equation (2), we see that $p = q = 0$ and hence $\exp(x) = {}_0F_0(;;x)$.

Table 1 lists the hypergeometric series representation of some elementary functions. For each function, we list (i) the usual power series representation of that function, (ii) the ratio t_{k+1}/t_k between two consecutive terms of that power series, and (iii) the corresponding hypergeometric series. In some cases, the first term t_0 of the power series is not 1, and has to be factored out.

Table 1
The representation of some elementary functions in terms of hypergeometric series.

Elementary functions	Power series	Ratio t_{k+1}/t_k	Hypergeometric series
$\exp(x)$	$\sum_{k=0}^{\infty} \frac{x^k}{k!}$	$\frac{1}{(k+1)} x$	${}_0F_0(;;x)$
$\mathrm{erf}(x)$	$\frac{2x}{\sqrt{\pi}} \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)k!} x^{2k}$	$-x^2 \frac{(k+\frac{1}{2})}{(k+\frac{3}{2})(k+1)}$	$\left(\frac{2x}{\sqrt{\pi}}\right) {}_1F_1(\frac{1}{2}; \frac{3}{2}; -x^2)$
$\sin(x)$	$x \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k+1)!}$	$(-\frac{1}{4}x^2) \frac{1}{(k+\frac{3}{2})(k+1)}$	$x \cdot {}_0F_1(; \frac{3}{2}; \frac{-x^2}{4})$
$\cos(x)$	$\sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}$	$-\frac{1}{4}x^2 \frac{1}{(k+\frac{1}{2})(k+1)}$	${}_0F_1(; \frac{1}{2}; \frac{-x^2}{4})$
$\arcsin(x)$	$x \sum_{k=0}^{\infty} \frac{(2k)!}{2^{2k}(2k+1)(k!)^2} x^{2k}$	$x^2 \frac{(k+\frac{1}{2})^2}{(k+\frac{3}{2})(k+1)}$	$x \cdot {}_2F_1(\frac{1}{2}, \frac{1}{2}; \frac{3}{2}; x^2)$
$\arctan(x)$	$x \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{2k+1}$	$-x^2 \frac{(k+\frac{1}{2})}{(k+\frac{3}{2})}$	$x \cdot {}_2F_1(\frac{1}{2}, 1; \frac{3}{2}; -x^2)$
$\log(1+x)$	$x \sum_{k=0}^{\infty} \frac{(-x)^k}{k+1}$	$(-x) \frac{(k+1)(k+1)}{(k+2)(k+1)}$	$x \cdot {}_2F_1(1, 1; 2; -x)$

The standard series for $\log(1+x)$ in Table 1 has poor convergence properties. By subtracting the standard series for $\log(1+x)$ from $\log(1-x)$, we

obtain

$$\log \frac{1-x}{1+x} = -2x\left[1 + \frac{x^2}{3} + \frac{x^4}{5} + \cdots\right] = -2x \cdot {}_2F_1(1, \frac{1}{2}; \frac{3}{2}; x^2).$$

Changing variables, we obtain

$$\log y = -2\left(\frac{1-y}{1+y}\right) {}_2F_1\left(1, \frac{1}{2}; \frac{3}{2}; \left(\frac{1-y}{1+y}\right)^2\right),$$

for $0 < y < \infty$. In addition to the functions in Table 1, we also compute $\tan(x)$, $\cot(x)$, and $\arccos(x)$ as $\tan(x) = \frac{\sin(x)}{\cos(x)}$, $\cot(x) = \frac{\cos(x)}{\sin(x)}$, and $\arccos(x) = \arcsin\left(\sqrt{1-x^2}\right)$, $0 \le x \le 1$.

3 Automatic Error Analysis

For this analysis, let $f(n) = P(n)/Q(n)$, $P(n) = \prod_{i=1}^{p}(n + a_i)$, and $Q(n) = \prod_{j=0}^{q}(n+b_j)$. Let $p \le q+1$, since otherwise the series diverge. Moreover, let the hypergeometric parameters $a_1, \ldots, a_p$, $b_1, \ldots, b_q$ be positive. Let $S = \sum_{i=0}^{\infty} t_i$, $S_n = \sum_{i=0}^{n-1} t_i$ and $R_n = S - S_n$.

Our goal is to compute a value $\widetilde{S}$ such that $|\widetilde{S} - S| \le \varepsilon$, where $\varepsilon \ge 0$ is a given (absolute) error bound. If we evaluate the series to n terms only, we obtain an approximate value $\widetilde{S}_n$, and so $|S_n - \widetilde{S}_n|$ is the **evaluation error**. The term $|R_n|$ is the **truncation error**. A third source of error occurst when the argument x is only an approximated of some true value x^*: $|x - x^*| \le \varepsilon'$, where ε' is the **argument error bound**. Argument reduction introduces such errors, which will be discussed in the next section.

Using the `Core Library`, it turns out that the approximation error $|S_n - \widetilde{S}_n|$ is a non-issue because we simply rely on the `Core Library` facility to compute an expression to any desired error bounds. We can simply set the approximation error to $\varepsilon/2$. It remains to bound the truncation error by $\varepsilon/2$. Our goal is to determine the n such that $|R_n| \le \varepsilon/2$. There are two basic cases, depending on whether the argument x is negative or not.

(A) The Alternating Case. Let $t_n t_{n+1} \le 0$ for all $n \in \mathbb{N}$ (this happens iff $x \le 0$). In this case, we have a well-known fact: *if* $|t_i| \ge |t_{i+1}|$ *for all* $i \ge n$ *then* $|R_n| \le |t_n|$ *and* $R_n t_n \ge 0$. To apply this result to hypergeometric series, let $\overline{a} = (a_1 + \cdots + a_p)/p$, $\overline{b} = (1 + b_1 + \cdots + b_q)/(q+1)$, and $a^* = \max_{i=1,\ldots,p} a_i$.

Lemma 3.1 *Let* $R_n = \sum_{i \ge n} t_i$ *be alternating. Then* $|R_n| \le |t_n|$ *in the following two situations:*
(i) Case $p = q + 1$*:* $\overline{b} > \overline{a}$ *and* $n \ge \frac{2^p (a^*)^2}{(\overline{b} - \overline{a})p}$.
(ii) Case $p \le q$*:* $n \ge \max\{2, 2^p a^*\}$.

(B) The Geometric Case. When the series is not alternating, we use the following observation:

Lemma 3.2 *There is a monotone decreasing function $g(n)$ such that $f(n) \leq g(n)$ and $\lim_{n\to\infty} g(n) = \lim_{n\to\infty} f(n)$.*

Proof. Observe that for any $a \leq 0, b \leq 0$, $(n+a)/(n+b) \leq 1$ if $a \leq b$, $(n+a)/(n+b) < (m+a)/(m+b) < a/b$ if $a > b, n > m$. Reorder the a_i's and b_j's such that for some $0 \leq r \leq p$, we have that $a_i > b_{i-1}$ for $i = 1, \ldots, r$, and $a_i \leq b_{i-1}$ for $i = r+1, \ldots, p$. Then

$$\begin{aligned} f(n) &= \frac{\prod_{i=1}^{p}(n+a_i)}{\prod_{j=0}^{q}(n+b_i)} = \prod_{i=1}^{p} \frac{n+a_i}{n+b_{i-1}} \prod_{j=p}^{q} \frac{1}{n+b_i} \\ &\leq \prod_{i=1}^{r} \frac{n+a_i}{n+b_{i-1}} \prod_{j=p}^{q} \frac{1}{n+b_i} = g(n). \end{aligned}$$

Thus $g(n)$ is monotone decreasing. Note that $\lim_{n\to\infty} g(n) = \lim_{n\to\infty} f(n)$ (this limit is 0 or 1, depending on whether $p < q+1$ or not). **Q.E.D.**

Lemma 3.3 *With $g(n)$ given in Lemma 3.2, if $xg(n) < 1$ then $|R_n| \leq \frac{t_n}{1-xg(n)}$.*

Proof. Note that for $i \geq 0$,

$$t_{n+i} = t_n x^i \prod_{j=0}^{i-1} f(n+j) \leq t_n x^i \prod_{j=0}^{i-1} g(n+j) \leq t_n x^i g(n)^i.$$

Summing,

$$R_n = \sum_{i\geq 0} t_{n+i} \leq t_n \sum_{i\geq 0} x^i g(n)^i = t_n/(1-xg(n)).$$

Q.E.D.

A even simpler choice for $g(n)$ is $g(n) = n^{p-q-1}$. In practice, there is a tradeoff between choosing $g(n)$ as small as possible and the computational effort to compute $g(n)$.

Implementation. The above bounds are applied directly to the elementary functions in Table 1. Here is an algorithm to dynamically evaluate any hypergeometric series:
(1) First we check if $p > q+1$ or ($p = q+1$ and $|x| \geq 1$). If so, we report a "divergent series" error.
(2) Else, we check if $x < 0$. If so, we can apply case (A) for alternating series.
(3) Else, we compute $g(n)$ and apply case (B) for the geometric case.

In general, to determine the n such that $|R_n| \leq \varepsilon/2$, we need to upper bound the value of t_n. A simple method is to accumulate the terms in S_n while at the same time check the next term t_n. On the face of it, computing t_n requires an iteration, as the number of factors in t_n grows with n. But in many situations, the number of factors in t_n is independent of n because of

cancellation (e.g., in $\log(1+x)$.) In this case, we can determine n in constant time by a direct evaluation. This yields a much faster implementation. This situation is exploited under parameter processing (see below).

4 Argument Reduction

An issue in the efficient evaluation of hypergeometric functions is the well-known problem of argument reduction. Each hypergeometric series is generally valid within a bounded range, and the problem is to reduce a general argument to this range. Even when an argument is in the valid range, argument reduction can still be applied to achieve faster convergence. As noted in [16, p.145–147], argument reduction in trigonometric functions are prone to catastrophic errors.

Whenever we perform argument reductions, an error is introduced into the modified arguments. We need to bound the effects of this error. For instance, argument reduction for the trigonometric functions uses the fact that they have period 2π. By exploiting other properties, the arguments can be reduced to a range of size $\pi/2$. If r is the reduced argument corresponding to an original argument of x, we have

$$r = x - \frac{\pi}{2}\left\lfloor \frac{2}{\pi} x \right\rfloor.$$

But we can only compute an approximation $\widetilde{r}$ to r. Using a sufficiently accurate π, we can bound $|r - \widetilde{r}|$ by any desired error bound ε'. The choice of ε' for the standard functions is deduced from the following lemma.

Lemma 4.1 *For $\varepsilon > 0$, we have the following bounds:*

$$\begin{aligned}
|\sin(x+\varepsilon) - \sin x| &\le \varepsilon \\
|\cos(x+\varepsilon) - \cos x| &\le \varepsilon \\
|\tan(x+\varepsilon) - \tan x| &\le 4\varepsilon, && 0 \le x \le \pi/4, \varepsilon < \pi/12 \\
|\cot(x+\varepsilon) - \cot x| &\le 2\varepsilon, && \pi/4 \le x \le \pi/2, \varepsilon \le \pi/4 \\
|\arcsin(x+\varepsilon) - \arcsin x| &\le 2\varepsilon, && |x| < 0.5, \varepsilon \le 1/4 \\
|\arccos(x+\varepsilon) - \arccos x| &\le 2\varepsilon, && |x| < 0.5, \varepsilon \le 1/4 \\
|\arctan(x+\varepsilon) - \arctan x| &\le \varepsilon, && |x| < 1 \\
|\log(x+\varepsilon) - \log x| &\le \varepsilon/x, && x > 0 \\
|\exp(x+\varepsilon) - \exp x| &\le 2\varepsilon \exp(x), && \varepsilon \le \log(2)
\end{aligned}$$

The proof use the remainder form of the Taylor expansion, $f(x+h) = f(x) + hf'(x+\theta), \quad 0 \le \theta \le h$.

Natural Log function. If $x > 2$, we let $x = 2^k r$ where $k \in \mathbb{N}$ and $1 < r \le 2$. Then $\log(x) = k\log(2) + \log(r)$. Here are the steps to approximate this expression:

1. First compute $k = \lfloor \log_2 x \rfloor$.

2. Compute $\widetilde{\log(2)}$ as an approximation of $\log(2)$ to absolute error $\leq \varepsilon/(2k)$.
3. Compute $\widetilde{r}$ such that $|r - \widetilde{r}| \leq \varepsilon/4$ where $r = x2^{-k}$.
4. Compute $\widetilde{\log(\widetilde{r})}$ as an approximation of $\log(\widetilde{r})$ to absolute error $\leq \varepsilon/4$.
5. Return $z = k\widetilde{\log(2)} + \widetilde{\log(\widetilde{r})}$.

Using lemma 4.1 (among other things), we can prove correctness of this procedure. That is, $|z - \log(x)| \leq \varepsilon$. Moreover, each of the steps is easily computed using the `Core Library`. Step 2 requires an approximation to the constant $\log(2)$, which we precompute (see Section 6).

Exponential function. Let $k = \lfloor x/\log(2) \rfloor$ and $r = x - k\log(2)$. Then $\exp(x) = 2^k exp(r)$. Note that $1 \leq r < 2$.

1. First, we compute k (requires a suitable approximation to $\log 2$).
2. Compute $\widetilde{r}$ as an approximation to $r = x - k\log 2$, to absolute error $\varepsilon 2^{-k-2}e^{-2}$.
3. Compute $\widetilde{\exp(\widetilde{r})}$ as an approximation to $\exp(\widetilde{r})$ to absolute error $\varepsilon 2^{-k-1}$.
4. Return $z = 2^k\widetilde{\exp(\widetilde{r})}$.

Trigonometric functions. To compute $\arcsin(x)$ when $0.5 < x \leq 1$, use

$$\arcsin(x) = \frac{\pi}{2} - 2\arcsin\left(\sqrt{\frac{1-x}{2}}\right)$$

From Lemma 4.1, we see that it is sufficient to compute π to absolute error bound of $\varepsilon/2$ and compute $\sqrt{(1-x)/2}$ to absolute error bound of $\varepsilon/8$. A similar reduction applies for $\arccos(x)$. For $\arctan(x)$ when $|x| > 1$, we use

$$\arctan(x) = \frac{\pi}{2} - \arctan\left(\frac{1}{x}\right).$$

Again, we need to compute $1/x$ to absolute error bound of $\varepsilon/2$. The cases for $\sin, \cos, \tan, \cot$ are even simpler.

5 Hypergeometric Parameter Pre-processing

Preprocessing is achieved in our implementation by introducing a "registration facility" for hypergeometric functions before they are called for the first time. Preprocessing includes computing bounding functions for the truncation error (e.g., the function $g(n)$ in Section 3). But we now discuss another important aspect of preprocessing.

Extra hypergeometric parameters are sometimes artificially introduced in order to achieve the standard form of these series. For instance, one of the upper parameters in $x \cdot {}_2F_1(1, 1; 2; -x)$ $(= \log(1 + x))$ amounts to cancelling the implicit lower parameter of $b_0 = 1$. This leads to a factor $k!/k!$ in the kth term t_k. While mathematically harmless, this has major performance impact in the `Core Library` evaluation mechanism. The example of $\log(1 + x)$ also illustrates another improvement possible: the upper parameter of 1 with a

lower parameter of 2 amounts to the factor $1/(k+1)$ in the ratio t_{k+1}/t_k. Again, it is important not to evaluate this factor as $(1)_k/(2)_k$. More generally, whenever an upper and a lower parameter differs by an integer, cancellations occur and one can gain improvements in efficiency by recognizing this.

We outline a general algorithm for processing the hypergeometric parameters. Let $a_1, a_2, , \ldots, a_p$ and $b_0, b_1, , \ldots, b_q$ be parameters of ${}_pF_q$. Note that we have added $b_0 = 1$ to the standard list of lower parameters.

(1) We first sort the a's and then the b's. Let $a_1 \leq a_2 \leq \cdots \leq a_p$ and $b_0 \leq b_1 \leq \cdots \leq b_q$ be the sorted result.

(2) By a merge-like algorithm we eliminate common terms from both lists. Note that we still maintain the separate lists.

(3) Next we form the maximum number of (a_i, b_j) of an upper and a lower parameter where $a_i - b_i$ is an integer. Let us call two real numbers x, y **equivalent** if $x - y \in \mathbb{Z}$. Let (A_i, B_i) $(i = 1, \ldots, r)$ be the set of such equivalent pairs; these are called ab-pairs since A_i is an upper parameter and B_i a lower parameter. Their corresponding values A_i, B_i are deleted from the original parameter lists. It is easy to see that the maximum number r of ab-pairs is unique. However, the set of these pairs are not unique. To ensure the most efficient code, we need to match A_i to B_i so as to minimize the sum $\sum_{i=1}^{r} |A_i - B_i|$. This "matching problem" is solved below.

(4) We compute the successive terms t_n as follows: Let s_n be the term that is computed from the remaining upper and lower parameter list as follows:

$$s_n = s_{n-1} \times f_n$$

where $f_n = (a_1+n)(a_2+n)\cdots(a_p+n)/(b_0+n)\cdots(b_q+n)$. We then initialize t_n to s_n. Then for each pair (A, B) where $B - A = k \geq 1$, we update

$$t_n = t_n * \frac{A(A+1)\cdots(A+k-1)}{(A+n)\cdots(A+k+n-1)}$$

If $A - B \geq 1$, there is an analogous factor. There is a special type of pairs that can be further exploited: when A, B are multiples of halves (this can be generalized too). In case $A = \alpha/2$ and $B - A = k \geq 1$, then (A, B) contributes the following factor to t_n:

$$\frac{\alpha(\alpha+2)\cdots(\alpha+2(k-1))}{(\alpha+2n)\cdots(\alpha+2(k+n-1))}$$

In the `Core Library`, this formulation will again lead to expressions of smaller depth, and more efficient evaluation. The following table shows the speedup when we exploit parameter reduction (in the standard series for $\log(1+x)$).

Number of digits	100	200	300	400	500
No preprocessing (secs.)	1.	5.01	6.99	8.89	10.46
Parameter preprocessing (secs.)	0.22	0.52	0.88	1.35	1.83
Speedup:	4.5	9.6	7.9	6.8	5.7

Minimum Matching Problem. Above, we had to solve the following problem[2]. Given two sorted lists of real numbers, $(A_1 < A_2 < \cdots < A_m)$ and $(B_1 < B_2 < \cdots < B_n)$, where $m \leq n$, we want to compute a set of m pairs $(A_1, B_{\alpha(1)}), \ldots, (A_m, B_{\alpha(m)})$ such that the sum $S = \sum_{i=1}^{m} |A_i - B_{\alpha(i)}|$ is minimized. Two pairs $(A_i, B_{\alpha(i)})$ and $(A_j, B_{\alpha(j)})$ are said to **cross** if $i < j$ and $\alpha(i) > \alpha(j)$. It is easy to see that if we "uncross" such a pair, we obtain a solution whose sum S is not more than the original. Hence we consider only non-crossing pairs. Consider the subproblems $P(i, j)$ comprising the input lists $(A_1, \ldots, A_i)$ and $(B_1, \ldots, B_j)$. Let $S(i, j)$ be the minimum value for subproblem $P(i, j)$. When $i = j$, the solution is unique in the obvious way. Otherwise, for $i < j$, $S(i, j) = \min\{S(i, j-1), S(i-1, j-1) + |A_i - B_j|\}$. Then, using standard dynamic programming, we can solve this problem in time $O(mn)$.

6 Mathematical Constants: Evaluation, File Formats and Access

The hypergeometric evaluation algorithm requires arbitrarily precise constants. When doing argument reduction for trigonometric functions, we need π. For argument reduction for $\exp(x)$ and for $\log(1 + x)$ we need $\log 2$. For the error function $\mathrm{erf}(x)$, we need $1/\sqrt{\pi}$. We could compute these constants on the fly, but performance is improved by precomputing these constants, storing them in files, and accessing them as needed. The following table compares the time to compute π to a certain number of bits (using Machin's formula) versus the time it takes to read the same value from a text file.

Bits	100	1000	3000	5000	7000	9000	10000	20000
On the fly (secs.)	0.04	0.50	2.49	5.88	10.66	17.19	21.38	107.61
Precomputed (secs.)	0.01	0.01	0.01	0.01	0.02	0.02	0.03	0.11
Speedup	4	50	249	588	533	859	713	978

We now describe facilities to compute, to store and to read constants in file formats. A fundamental decision was to use text files rather than binary files, as the former is human readable. The format supports both integer, floating point and rational number representations. Next, the base of the numbers

[2] This might appear to be a known problem, but we have not found a reference.

can be binary, hexadecimal or decimal. The advantage of binary/hexadecimal is that conversion into the internal format of the `Core Library` takes linear time. The formal specification is distributed with `Core Library` version 1.3 or higher [3].

7 Conclusions

This paper describes an effort to incorporate non-algebraic functions into a framework that supports guaranteed precision. The implementation of such a package is naturally achieved using the basic capabilities of the `Core Library`. There are several issues of efficiency and automation which we have addressed: automatic error bound computation, hypergeometric parameter processing, argument reduction, and finally constant precomputation, storage and retrieval. The full paper will describe implementation details such as a facility for registering hypergeometric functions. Topics for future work include acceleration of series. The hypergeometric package is distributed with the `Core Library`, at our website `http://cs.nyu.edu/exact/`.

References

[1] J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. Cambridge University Press, 1997. Translated by Hervé Brönnimann.

[2] H. Brönnimann, C. Burnikel, and S. Pion. Interval arithmetic yields efficient dynamic filters for Computation Geometry. *ACM Symp. on Computational Geometry*, 14:165–174, 1998.

[3] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. Exact geometric computation made easy. In *Proc. 15th ACM Symp. Comp. Geom.*, pages 341–450, 1999.

[4] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. A strong and easily computable separation bound for arithmetic expressi ons involving radicals. *Algorithmica*, 27:87–99, 2000.

[5] C. Burnikel, S. Funke, K. Mehlhorn, and S. Schirra. A separation bound for real algebraic expressions, 2001. preprint (submitted to journal).

[6] C. Burnikel, S. Funke, and M. Seel. Exact geometric comptuation using cascading. *Int'l. J. Computational Geometry and Applications*, 11(3):245–266, 2001. Special Issue.

[7] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[8] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schoenherr. The CGAL kernel: a basis for geometric computation. In M. C. Lin and D. Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, pages 191–202, Berlin, 1996. Springer. Lecture Notes in Computer Science No. 1148; Proc. 1st ACM Workshop on Applied Computational Geometry (WACG), Federated Computing Research Conference 1996, Philadelphia, USA.

[3] Look under the directory `progs/fileIO`.

[9] E. Jeandel. Évaluation rapide de fonctions hypergéométriques. Rapport Technique 242, INRIA, 2000. 17 pages.

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

[11] C. Li and C. Yap. A new constructive root bound for algebraic expressions. In *Proc. 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 496–505. ACM and SIAM, Jan. 2001.

[12] C. Li and C. Yap. Recent progress in exact geometric computation. In S. Basu and L. Gonzalez-Vega, editors, *Proc. DIMACS Workshop on Algorithmic and Quantitative Aspects of Real Algebraic Geometry in Mathematics and Computer Science, March 12 - 16, 2001.*, DIMACS Books Series. American Math. Society, 2001. Submitted. Paper download `http://cs.nyu.edu/exact/doc/`.

[13] G. Liotta, F. Preparata, and R. Tamassia. Robust proximity queries: an illustration of degree-driven algorithm design. *SIAM J. Computing*, 2001. to apear.

[14] Y. V. Matiyasevich. *Hilbert's Tenth Problem*. The MIT Press, Cambridge, Massachusetts, 1994.

[15] K. Mehlhorn and S. Näher. LEDA: a platform for combinatorial and geometric computing. *CACM*, 38:96–102, 1995.

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

[17] M. Overmars. Designing the computational geometry algorithms library CGAL. In M. C. Lin and D. Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, pages 53–58, Berlin, 1996. Springer. Lecture Notes in Computer Science No. 1148; Proc. 1st ACM Workshop on Applied Computational Geometry (WACG).

[18] S. Schirra. Precision and robustness in geometric computations. In M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer, editors, *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes Comp. Science*, pages 255–287. Springer, 1997. Chapter 9.

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

[20] C. Yap. A new number core for robust numerical and geometric libraries. In *3rd CGC Workshop on Geometric Computing*, 1998. Invited Talk. Brown University, Oct 11–12, 1998. For abstracts, see `http://www.cs.brown.edu/cgc/cgc98/home.html`.

[21] C. K. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 35, pages 653–668. CRC Press LLC, Boca Raton, FL, 1997.

[22] C. K. Yap. Towards exact geometric computation. *Computational Geometry: Theory and Applications*, 7:3–23, 1997. Invited talk, Proceed. 5th Canadian Conference on Comp. Geometry, Waterloo, Aug 5–9, 1993.