Fast Algorithms for High-Precision Computation of Elementary Functions*

Richard P. Brent

Australian National University
Canberra, ACT 0200, Australia
RNC7@rpbrent.com

12 July 2006

*Presented at RNC7, Nancy, 12 July 2006.

Motivation

In many applications of real-number computation we need to evaluate elementary functions such as $\exp(x)$, $\ln(x)$, $\arctan(x)$ to high precision. Using high-precision values we can sometimes discover identities by *integer relation-finding*:

given nonzero real numbers $x_1, \cdots, x_n$, find integers a_i (not all zero) such that

$$\sum_{i=1}^{n} a_i x_i = 0\,,$$

or show that no such relation exists for $||a|| \le B$, where B is some (large) bound.

If we are successful then we actually find integers a_i such that

$$\left|\sum_{i=1}^{n} a_i x_i\right| < \epsilon\,,$$

where ϵ depends on the precision of the computation. To be confident that this is not just a "near miss", ϵ should be as small as possible.

Special case

If $n = 2$, the problem is to find nonzero integers a_1, a_2 such that

$$a_1 x_1 + a_2 x_2 = 0\,,$$

that is

$$x_1/x_2 = -a_2/a_1\,,$$

so we are trying to show that $x = x_1/x_2$ is rational.

To test if a real number x is rational, we can compute its regular continued fraction

$$x = q_0 + 1/(q_1 + 1/(q_2 + \cdots))$$

and see if it terminates. With approximate computation this will usually be indicated by a very large quotient q_k, that is $1/q_k$ is zero modulo the error of the computation.

Example

Euler's constant $\gamma = 0.57721566\cdots$ can be defined by

$$\gamma = -\int_0^\infty e^{-x}\ln x\,dx = -\int_0^1 \ln\ln\left(\frac{1}{x}\right)\,dx$$

Open problem: Is γ or $\exp(\gamma)$ rational ?

Since the regular continued fraction gives best rational approximations, continued fraction computations can give theorems of the form:

If x is rational, say $x = p/q$, then $|q| > B$ for some (very large) bound B.

To obtain a result like this with given bound B, we need to compute x with absolute error $O(1/B^2)$.

Using this method we know that, if γ or $\exp(\gamma)$ is rational, say p/q, then $|q|$ must be very large. Thus, in some sense these numbers are "unlikely" to be rational.

Why consider $\exp(\pm\gamma)$?

From Merten's theorem,

$$\exp(-\gamma) = \lim_{x\to+\infty} \ln(x) \prod_{p\le x} \frac{p-1}{p} \,,$$

where the product is over primes $p \le x$.

From Grunwald's theorem,

$$\exp(\gamma) = \limsup_{n\to\infty} \frac{\sigma(n)}{n \ln\ln n} \,,$$

where

$$\sigma(n) = \sum_{d|n} d$$

is the *sum-of-divisors* function.

Thus $\exp(\pm\gamma)$ has several “natural” definitions.

Robin's theorem: the Riemann Hypothesis is true iff

$$(\forall n > 5040) \;\; \frac{\sigma(n)}{n \ln\ln n} < \exp(\gamma) \,.$$

Another motivation

If you know efficient algorithms to compute elementary functions to high precision, you might win the next *More Digits friendly Competition*!

Outline

We shall survey some of the well-known (and some not so well-known) techniques for fast evaluation of elementary functions, as well as mentioning some new ideas.

Notation

Let d be the number of binary digits required, so the computation should be accurate with relative (or, if appropriate, absolute) error $O(2^{-d})$. By "high-precision" we mean higher than can be obtained directly using IEEE 754 standard floating-point hardware, typically d several hundred up to millions.

We are interested both in "asymptotically fast" algorithms (the case $d \to +\infty$) and in algorithms that are competitive in some range of d.

Multiplication algorithms

Let $M(d)$ denote the time (measured in word- or bit-operations) required to multiply d-bit numbers with d-bit accuracy (we are generally only interested in the upper half of the $2d$-bit product).

Classically $M(d) = O(d^2)$ and the Schönhage-Strassen algorithm shows that $M(d) = O(d \log d \log \log d)$. However, Schönhage-Strassen is only useful for large d, and there is a significant region $d_1 < d < d_2$ where A. Karatsuba's $O(d^{\lg 3})$ algorithm is best ($\lg 3 = \log_2 3 < 1.59$).

We'll assume that $M(d) \gg d$ and that $M(d)$ satisfies reasonable smoothness conditions.

In the region where Karatsuba's algorithm is best for multiplication, the best algorithms for elementary functions need not be those that are asymptotically the fastest.

Ground rules

Sometimes the best algorithm depends on the ground rules: are certain constants such as π allowed to be precomputed, or does the cost of their computation have to be counted every time in the cost of the elementary function evaluation?

Useful techniques

Techniques for high-precision elementary function evaluation include the following. Often several are used in combination, e.g. argument reduction is used before power series evaluation.

1. Argument reduction
2. Fast power series evaluation
3. The arithmetic-geometric mean (AGM)
4. Newton's method
5. Complex arithmetic
6. Binary splitting

1. Argument reduction

The elementary functions have *addition formulae* such as

$$\exp(x+y) = \exp(x)\exp(y)\,,$$

$$\ln(xy) = \ln(x) + \ln(y)\,,$$

$$\sin(x+y) = \sin(x)\cos(y) + \cos(x)\sin(y)\,,$$

$$\tan(x+y) = \frac{\tan(x)+\tan(y)}{1-\tan(x)\tan(y)}\,.$$

We can use these formulæ to reduce the argument so that power series converge rapidly. Usually we take $x = y$ to get *doubling formulae* such as

$$\exp(2x) = (\exp(x))^2\,,$$

though occasionally *tripling formulae* such as

$$\sin(3x) = 3\sin(x) - 4\sin^3(x)$$

might be useful.

Repeated use of doubling formulæ

If we apply the doubling formula for exp k times, we get

$$\exp(x) = \left(\exp(x/2^k)\right)^{2^k} .$$

Thus, if $|x| = O(1)$, we can reduce the problem of evaluating $\exp(x)$ to that of evaluating $\exp(x/2^k)$, where the argument is now $O(2^{-k})$. The cost is the k squarings that we need to get the final result from $\exp(x/2^k)$.

There is a tradeoff here and k should be chosen to minimise the total time. If the obvious method for power series evaluation is used, then the optimal k is of order $\sqrt{d}$ and the overall time is $O(d^{1/2}M(d))$. (We'll see soon that there are faster ways to evaluate power series, so this is not the best possible result.)

We assumed that $|x| = O(1)$. A more careful analysis would show that the optimal k depends on $|x|$.

Loss of precision

If $x < 0$ we should use

$$\exp(x) = 1/\exp(-x)$$

to avoid cancellation in the power series summation.

Guard digits

Care has to be taken to use enough guard digits. Since $x/2^k$ is $O(2^{-k})$, we lose at least k bits of precision in summing the power series $1 + x/2^k + \cdots$ then squaring k times. To avoid this, we could evaluate $\mathrm{expm1}(x/2^k)$, where the function expm1 is defined by

$$\mathrm{expm1}(x) = \exp(x) - 1$$

and has a doubling formula

$$\mathrm{expm1}(2x) = \mathrm{expm1}(x)(2 + \mathrm{expm1}(x))$$

that avoids loss of significance when $|x|$ is small.

Loss of precision of the argument

Consider the reduction formula

$$\ln(1+x) = 2\ln(1+y)$$

where

$$1+y = \sqrt{1+x} = 1 + \frac{x}{1+\sqrt{1+x}}\,.$$

It could be better to work with the function log1p defined by

$$\mathrm{log1p}(x) = \ln(1+x)$$

to avoid loss of precision (this time in the argument rather than in the function). Then

$$\mathrm{log1p}(x) = 2\mathrm{log1p}\left(\frac{x}{1+\sqrt{1+x}}\right)$$

avoids loss of significance when $|x|$ is small. However, note that argument reduction for ln or log1p is more expensive than that for exp or expm1, because of the square root.

2. Fast power series evaluation

The elementary functions have power series expansions such as

$$\exp x = \sum_{k\ge 0} \frac{x^k}{k!}\,,$$

$$\ln(1+x) = \sum_{k\ge 0} \frac{(-1)^k x^{k+1}}{k+1}\,,$$

$$\arctan x = \sum_{k\ge 0} \frac{(-1)^k x^{2k+1}}{2k+1}\,,$$

$$\sinh x = \sum_{k\ge 0} \frac{x^{2k+1}}{(2k+1)!}\,.$$

Power series to avoid

In some cases the coefficients in the series are nontrivial to evaluate. For example,

$$\tan x = \sum_{k\geq 1} \frac{T_k}{(2k-1)!} x^{2k-1},$$

where the constants T_k are called *tangent numbers* and can be expressed in terms of Bernoulli numbers. In such cases it is best to avoid direct power series evaluation.

For example, to evaluate $\tan x$ we can use Newton's method on the inverse function (arctan), or we can use $\tan x = \sin x/\cos x$.

Thus, we'll assume for the moment that we have a power series $\sum_{k\geq 0} a_k x^{\alpha k+\beta}$ where a_{k+1}/a_k is a rational function $R(k)$ of k, and hence it is easy to evaluate $a_0, a_1, a_2, \ldots$ sequentially. For example, in the case of $\exp x$,

$$\frac{a_{k+1}}{a_k} = \frac{k!}{(k+1)!} = \frac{1}{k+1}.$$

In general, our assumptions cover hypergeometric functions.

The radius of convergence

If the elementary function is an entire function (e.g. exp, sin) then the power series converges in the whole complex plane. In this case the degree of the denominator of $R(k)$ is greater than that of the numerator.

In other cases (such as $\ln, \arctan$) the function is not entire and the power series only converges in a disk in the complex plane because the function has a singularity on the boundary of this disk. In fact $\ln(x)$ has a singularity at the origin, which is why we consider the power series for $\ln(1+x)$. This power series has radius of convergence 1.

Similarly, the power series for $\arctan(x)$ has radius of convergence 1 because $\arctan(x)$ has a singularity on the unit circle (even though it is uniformly bounded for all real x).

Direct power series evaluation

Using periodicity (in the cases of $\sin, \cos$) and/or argument reduction techniques, we can assume that we want to evaluate a power series $\sum_{k\ge 0} a_k x^k$ where $|x| \le 1/2$ and the radius of convergence of the series is at least 1.

As before, assume that a_{k+1}/a_k is a rational function of k, and hence easy to evaluate.

To sum the series with error $O(2^{-d})$ it is sufficient to take $d + O(1)$ terms, so the time required is

$$O(dM(d))\,.$$

If the function is entire, then the series converges faster and the time is reduced to

$$O\left(\frac{dM(d)}{\log d}\right)\,.$$

However, we can do much better by carrying the argument reduction further!

Power series with argument reduction

By applying argument reduction $k + O(1)$ times, we can ensure that the argument x satisfies

$$|x| < 2^{-k}\,.$$

Then, to obtain d-bit accuracy we only need to sum $O(d/k)$ terms in the power series. The total cost is

$$O\left((k + d/k)M(d)\right)$$

so choosing $k \sim d^{1/2}$ gives cost

$$O\left(d^{1/2}M(d)\right)\,.$$

Note. We are assuming that a step of argument reduction is $O(M(d))$, which is true for the elementary functions.

Examples

For example, this applies to the evaluation of $\exp(x)$ using

$$\exp(x) = (\exp(x/2))^2 \,,$$

to $\mathrm{log1p}(x) = \ln(1+x)$ using

$$\mathrm{log1p}(x) = 2\mathrm{log1p}\left(\frac{x}{1+\sqrt{1+x}}\right) ,$$

and to $\arctan(x)$ using

$$\arctan x = 2\arctan\left(\frac{x}{1+\sqrt{1+x^2}}\right) .$$

Note that in the last two cases each step of the argument reduction requires a square root, but this can be done with cost $O(M(d))$ by Newton's method. Thus in all three cases the overall cost is

$$O\left(d^{1/2}M(d)\right) ,$$

although the implicit constant is smaller for exp than for log1p or arctan.

Using symmetries

A well-known idea is to evaluate $\ln(1+x)$ using the power series

$$\ln\left(\frac{1+y}{1-y}\right) = 2\sum_{k\geq 0}\frac{y^{2k+1}}{2k+1}$$

with y defined by $(1+y)/(1-y) = 1+x$, i.e. $y = x/(2+x)$. This saves half the terms and also reduces the argument, since $y < x/2$ if $x > 0$.

A similar but less well-known idea is to use

$$\sinh(x) = \frac{e^x - e^{-x}}{2} = \sum_{k\geq 0}\frac{x^{2k+1}}{(2k+1)!}$$

to compute $\exp(x)$. Instead of computing $\exp(x)$ directly by the power series method, we first compute $\sinh(x)$ and then use

$$\exp(x) = \sinh(x) + \sqrt{1+\sinh^2(x)}\,.$$

Using symmetries continued

At the cost of a square root, we get rid of half the terms in the power series for $\exp(x)$. This should save a factor of close to 2 for naive power series evaluation, or $\sqrt{2}$ if argument reduction is followed by power series evaluation.

The argument reduction

It is more efficient to do argument reduction via the doubling formula for exp than the tripling formula for sinh:

$$\sinh(3x) = \sinh(x)(3 + 4\sinh^2(x))\,,$$

because it takes one multiplication and one squaring (which may be cheaper) to apply the tripling formula, but only two squarings to apply the doubling formula twice (and $3 < 2^2$).

Faster power series evaluation

Once we determine how many terms in the power series are required for the desired accuracy, the problem reduces to evaluating a truncated power series, i.e. a polynomial.

Paterson and Stockmeyer (1973) considered the number of *nonscalar* multiplications required to evaluate a polynomial $P(x) = \sum_{0 \le j < n} a_j x^j$. They showed that $P(x)$ can be evaluated in $O(\sqrt{n})$ nonscalar multiplications (plus $O(n)$ scalar multiplications and $O(n)$ additions, using $O(\sqrt{n})$ storage).

Smith (1989) applied this idea to multiple-precision evaluation of elementary functions. The same idea applies, more generally, to evaluation of hypergeometric functions.

Smith's method

Suppose $n = jk$, and write

$$P(x) = \sum_{\ell=0}^{j-1} x^\ell P_\ell(x^j)\,,$$

where

$$P_\ell(y) = \sum_{m=0}^{k-1} a_{jm+\ell}\, y^m\,.$$

The idea is to precompute the powers

$$x, x^2, x^3, \ldots, x^{j-1}$$

and then

$$y = x^j, y^2, y^3, \ldots, y^{k-1}\,.$$

Now the polynomials $P_\ell(y)$ can be evaluated using only scalar multiplications, since $a_{jm+\ell}\, y^m$ can be computed from $a_{jm+\ell-1} y^m$ using a scalar multiplication by the rational $a_{jm+\ell}/a_{jm+\ell-1}$.

Geometric interpretation

To see the idea geometrically, write $P(x)$ as

$$\begin{array}{rlclclcl}
x^0\ [a_0 & + & a_jy & + & a_{2j}y^2 & + & \cdots] & + \\
x^1\ [a_1 & + & a_{j+1}y & + & a_{2j+1}y^2 & + & \cdots] & + \\
x^2\ [a_2 & + & a_{j+2}y & + & a_{2j+2}y^2 & + & \cdots] & + \\
& \vdots & & \vdots & & \vdots & & \\
x^{j-1}\ [a_{j-1} & + & a_{2j-1}y & + & a_{3j-1}y^2 & + & \cdots] &
\end{array}$$

where $y = x^j$. The terms in square brackets are the polynomials $P_0(y), P_1(y), \ldots, P_{j-1}(y)$.

We traverse the first column of the array, then the second column, then the third, …, finally the j-th column, accumulating sums $S_0, S_1, \ldots S_{j-1}$ (one for each row). At the end of this process $S_\ell = P_\ell(y)$ and we only have to evaluate

$$P(x) = \sum_{\ell=0}^{j-1} x^\ell S_\ell\,.$$

Note. Alternatively, transpose the matrix of coefficients above and interchange the powers of x and y. The complexity is the same.

Complexity of Smith's method

To evaluate a polynomial $P(x)$ of degree $n-1 = jk-1$, Smith's method takes $O(j+k)$ nonscalar multiplications (each costing $O(M(d))$) and $O(jk)$ scalar multiplications. The scalar multiplications involve multiplication and/or division of a multiple-precision number by small integers. Assume that these multiplications and/or divisions take time $c(n)d$. In practice we can safely regard $c(n)$ as constant.

Choosing $j \sim k \sim n^{1/2}$ we get overall time

$$O(n^{1/2}M(d) + nd \cdot c(n))\,.$$

If $n \sim d$ this is not an improvement on the bound $O(d^{1/2}M(d))$ that we obtained already by argument reduction and power series evaluation. However, we can do argument reduction before applying Smith's method. Applying $\sim d^{1/3}$ steps of argument halving, we can take $n \sim d^{2/3}$ and get overall time

$$O(d^{1/3}M(d) + d^{5/3}c(d))\,.$$

The slowly-growing function $c(n)$

The scalar multiplications involve multiplication and/or division of a d-bit multiple-precision number by "small" integers. Here "small" means $O(n)$, i.e. integers with $O(\log n)$ digits. Suppose that these multiplications and/or divisions take time $c(n)d$. There are three cases:

1. The small integers fit in one word. Then $c(n) = O(1)$ is a constant. This is the case that occurs in practice.

2. If the small integers do not fit in one word, they certainly fit in $O(\log n)$ words, so a straightforward implementation gives $c(n) = O(\log n)$.

3. If we split the d-digit numbers into $O(d/\log n)$ blocks each of $O(\log n)$ bits, and apply Schönhage-Strassen multiplication (or division using Newton's method) within each block, we get $c(n) = O(\log\log n \log\log\log n)$.

Complexity of Smith's method cont.

We saw that Smith's method takes time

$$T(d) = O(d^{1/3}M(d) + d^{5/3}c(d)) \,.$$

Which term dominates? There are two cases:

1. $M(d) \gg d^{4/3}c(d)$. Here the first term dominates and $T(d) = O(d^{1/3}M(d))$.
   This case applies if we use classical or Karatsuba multiplication, since $\lg 3 > 4/3$.

2. $M(d) \ll d^{4/3}c(d)$. Here the second term dominates and $T(d) = O(d^{5/3}c(d))$.
   In fact this bound can be improved since our choice $n \sim d^{2/3}$ is no longer optimal.
   With $n \sim \sqrt{M(d)/c(d)}$ we get an improved bound $\Theta(d\sqrt{M(d)c(d)}) \gg d^{3/2}$.

   We can not approach the $O(d^{1+\epsilon})$ that is achievable with AGM-based methods, so we probably should not be using Smith's method (or any method based on power series evaluation) for such large d.

3. The arithmetic-geometric mean

The fastest known methods for very large d are based on the arithmetic-geometric mean (AGM) iteration of Gauss. They take time

$$O(M(d)\log d)\,.$$

The implicit constant here can be quite large, so other methods are better for small d.

Given (a_0, b_0), the AGM iteration is defined by

$$(a_{n+1}, b_{n+1}) = \left(\frac{a_n + b_n}{2}, \sqrt{a_n b_n}\right)\,.$$

For simplicity we'll only consider real, positive starting values (a_0, b_0) for the moment, but the results can be extended to complex starting values (see Borwein & Borwein, *Pi and the AGM*, pp. 15–16) and we'll use that later.

The AGM iteration converges *quadratically* to a limit which we'll denote by $\mathrm{AGM}(a_0, b_0)$.

Why the AGM is useful

The AGM is useful because:

1. It converges quadratically. Eventually the number of correct digits doubles at each iteration, so only $O(\log d)$ iterations are required.

2. Each iteration takes time $O(M(d))$ because the square root can be computed in time $O(M(d))$ by Newton's method.

3. If we take suitable starting values (a_0, b_0), the result $\mathrm{AGM}(a_0, b_0)$ can be used to compute logarithms (directly) and other elementary functions (less directly), as well as constants such as π.

Elliptic integrals

The theory of the AGM iteration is intimately linked to the theory of elliptic integrals.

The *complete elliptic integral of the first kind* is

$$\begin{aligned} K(k) &= \int_0^{\pi/2} \frac{d\theta}{\sqrt{1-k^2\sin^2\theta}} \\ &= \int_0^1 \frac{dt}{\sqrt{(1-t^2)(1-k^2t^2)}}\,, \end{aligned}$$

and the *complete elliptic integral of the second kind* is

$$\begin{aligned} E(k) &= \int_0^{\pi/2} \sqrt{1-k^2\sin^2\theta}\,d\theta \\ &= \int_0^1 \sqrt{\frac{1-k^2t^2}{1-t^2}}\,dt\,. \end{aligned}$$

$k \in [0,1]$ is called the *modulus* and
$k' = \sqrt{1-k^2}$ is the *complementary modulus.*
It is traditional (though confusing) to write
$K'(k)$ for $K(k')$ and $E'(k)$ for $E(k')$.

The connection with elliptic integrals

Gauss discovered that

$$\frac{1}{\mathrm{AGM}(1,k)} = \frac{2}{\pi} K'(k) \ ,$$

This identity can be used to compute the elliptic integral K rapidly via the AGM iteration. We can also use it to compute logarithms.

From the definition

$$K(k) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1 - k^2 \sin^2 \theta}} \, ,$$

we see that $K(k)$ has a series expansion that converges for $|k| < 1$ (in fact $K(k) = (\pi/2) F(1/2, 1/2; 1; k^2)$ is a hypergeometric function). For small k we have

$$K(k) = \frac{\pi}{2} \left(1 + \frac{k^2}{4} + O(k^4) \right) \ .$$

It can also be shown [B&B, (1.3.10)] that

$$K'(k) = \frac{2}{\pi} \ln \left(\frac{4}{k} \right) K(k) - \frac{k^2}{4} + O(k^4) \ .$$

First AGM algorithm for ln

From these formulæ we easily get

$$\frac{\pi/2}{\mathrm{AGM}(1,k)} = \ln(4/k)(1+O(k^2))\,.$$

Thus, if $x = 4/k$ is large, we have

$$\ln(x) = \frac{\pi/2}{\mathrm{AGM}(1,4/x)}\left(1+O\left(\frac{1}{x^2}\right)\right)\,.$$

If $x \ge 2^{d/2}$, we can compute $\ln(x)$ to precision d using the AGM iteration. It takes about $2\lg(d)$ iterations to converge if $x \in [2^{d/2}, 2^d]$.

Note that we need the constant π, which could be computed by using our formula twice with slightly different arguments x_1 and x_2, then taking differences. More efficient is to use the Brent-Salamin algorithm, which is based on the AGM and the Legendre relation

$$EK' + E'K - KK' = \frac{\pi}{2}\,.$$

Argument expansion

If x is not large enough, we can compute

$$\ln(2^k x) = k \ln 2 + \ln x$$

by the AGM method (assuming the constant $\ln 2$ is known). Alternatively, if $x > 1$, we can square x enough times and compute

$$\ln\left(x^{2^k}\right) = 2^k \ln(x)\,.$$

This method with $x = 2$ gives a way of computing $\ln 2$.

The error term

The $O(k^2)$ error term in the formula

$$\frac{\pi/2}{\mathrm{AGM}(1,k)} = \ln\left(\frac{4}{k}\right)(1+O(k^2))$$

is a nuisance. [B&B, p.11, ex. 4(c)] gives a rigorous bound

$$\left|\frac{\pi/2}{\mathrm{AGM}(1,k)} - \ln\left(\frac{4}{k}\right)\right| \le 4k^2(8-\ln(k))$$

for all $k \in (0,1]$, and the bound can be sharpened to $0.37k^2(2.4-\ln(k))$ if $k \in (0,0.5]$.

The error $O(k^2|\ln k|)$ makes it difficult to accelerate convergence by using a larger value of k (i.e. a smaller value of $x = 4/k$). There is an *exact* formula which is much more elegant and avoids this problem. Its use to compute $\ln x$ was first suggested by Sasaki and Kanada (see [B&B, (7.2.5)], but beware the typo).

Before giving Sasaki & Kanada's formula we need to define some *theta functions* and show how they can be used to parameterise the AGM iteration.

Theta functions

The theta functions that we need are $\theta_2(q)$, $\theta_3(q)$ and $\theta_4(q)$, defined for $|q| < 1$ by

$$
\begin{aligned}
\theta_2(q) &= \sum_{n=-\infty}^{+\infty} q^{(n+1/2)^2} = 2q^{1/4}\sum_{n=0}^{+\infty} q^{n(n+1)}\,, \\
\theta_3(q) &= \sum_{n=-\infty}^{+\infty} q^{n^2} = 1 + 2\sum_{n=1}^{+\infty} q^{n^2}\,, \\
\theta_4(q) &= \theta_3(-q) = 1 + 2\sum_{n=1}^{+\infty} (-1)^n q^{n^2}\,.
\end{aligned}
$$

Note that the defining power series are sparse so it is easy to compute $\theta_2(q)$ and $\theta_3(q)$ for small q. Unfortunately, Smith's method does not help to speed up the computation.

The asymptotically fastest methods to compute theta functions use the AGM. However, we won't follow this trail because it would lead us in circles! (We want to use theta functions to give starting values for the AGM iteration.)

Theta function identities

There are many identities involving theta functions (see [B&B, Ch. 2]). Two that are of interest to us are:

$$\frac{\theta_3^2(q) + \theta_4^2(q)}{2} = \theta_3^2(q^2)$$

and

$$\theta_3(q)\theta_4(q) = \theta_4^2(q^2)$$

which may be written as

$$\sqrt{\theta_3^2(q)\theta_4^2(q)} = \theta_4^2(q^2)$$

to show the connection with the AGM:

$$\begin{aligned} & \mathrm{AGM}(\theta_3^2(q)\,, \theta_4^2(q)) = \cdots \\ = \; & \mathrm{AGM}(\theta_3^2(q^{2^k})\,, \theta_4^2(q^{2^k})) = \cdots = 1 \end{aligned}$$

for any $|q| < 1$.

Apart from scaling, the AGM iteration is parameterised by $(\theta_3^2(q^{2^k}), \theta_4^2(q^{2^k}))$ for $k = 0, 1, 2, \ldots$

The scaling factor

Since $\mathrm{AGM}(\theta_3^2(q)\,, \theta_4^2(q)) = 1$, scaling gives

$$\mathrm{AGM}(1, k') = \frac{1}{\theta_3^2(q)}$$

if

$$k' = \frac{\theta_4^2(q)}{\theta_3^2(q)}\,.$$

Equivalently, since $\theta_2^4 + \theta_4^4 = \theta_3^4$ (Jacobi),

$$k = \frac{\theta_2^2(q)}{\theta_3^2(q)}\,.$$

However, we know that

$$\frac{1}{\mathrm{AGM}(1, k')} = \frac{2}{\pi} K(k)\,,$$

so

$$K(k) = \frac{\pi}{2}\theta_3^2(q)\,.$$

Thus, the theta functions are closely related to elliptic integrals. In the theory q is usually called the *nome* associated with the modulus k.

From q to k and k to q

We saw that

$$k = \frac{\theta_2^2(q)}{\theta_3^2(q)}\,,$$

which gives k in terms of q. There is also a nice inverse formula which gives q in terms of k:

$$q = \exp(-\pi K'(k)/K(k))\,,$$

or equivalently

$$\ln\left(\frac{1}{q}\right) = \frac{\pi K'(k)}{K(k)}\,.$$

For a proof see [B&B, §2.3].

Sasaki and Kanada's formula

Putting all these pieces together gives Sasaki and Kanada's elegant formula:

$$\ln\left(\frac{1}{q}\right) = \frac{\pi}{\mathrm{AGM}(\theta_2^2(q)\,,\theta_3^2(q))}\,.$$

Second AGM algorithm for ln

Suppose $x \gg 1$. Let $q = 1/x$, compute $\theta_2(q^4)$ and $\theta_3(q^4)$ from their defining series, then compute $\mathrm{AGM}(\theta_2^2(q^4)\,,\theta_3^2(q^4))$. Sasaki and Kanada's formula (with q replaced by q^4 to avoid the $q^{1/4}$ term in the definition of $\theta_2(q)$) gives

$$\ln(x) = \frac{\pi/4}{\mathrm{AGM}(\theta_2^2(q^4)\,,\theta_3^2(q^4))}\,.$$

There is a tradeoff between increasing x (by squaring or multiplication by a power of 2) and taking longer to compute $\theta_2(q^4)$ and $\theta_3(q^4)$ from their series. In practice it seems good to increase x so that $q = 1/x$ is small enough that $O(q^{36})$ terms are negligible. Then we can use

$$\theta_2(q^4) = 2\left(q + q^9 + q^{25} + O(q^{49})\right)\,,$$

$$\theta_3(q^4) = 1 + 2\left(q^4 + q^{16} + O(q^{36})\right)\,.$$

We need $x \geq 2^{d/36}$ which is much better than the requirement $x \geq 2^{d/2}$ for the first AGM algorithm. We save about four AGM iterations at the cost of a few multiplications.

Implementation notes

Since

$$\mathrm{AGM}(\theta_2^2\,,\theta_3^2) = \frac{\mathrm{AGM}(2\theta_2\theta_3\,,\theta_2^2+\theta_3^2)}{2}\,,$$

we can avoid the first square root in the AGM iteration. Also, it only takes two nonscalar multiplications to compute $2\theta_2\theta_3$ and $\theta_2^2+\theta_3^2$.

Constants

d-bit square roots take time $\sim 4.25M(d)$ so one AGM iteration takes time $\sim 5.25M(d)$.

The AGM algorithms require $2\lg(d)+O(1)$ AGM iterations. The total time to compute $\ln(x)$ by the AGM is $\sim 10.5\lg(d)M(d)$.

Paul Zimmermann notes that the constant is smaller if d-bit multiplication produces a $2d$-bit result $(4.25M(d) \to 2.25M^*(d))$.

Dan Bernstein notes that the algorithms can be speeded up if redundant FFTs in the square root computations are eliminated (assuming that multiplication uses the FFT).

Drawbacks of the AGM

1. The AGM iteration is *not* self-correcting, so we have to work with full precision (plus any necessary guard digits) throughout. In contrast, when using Newton's method or evaluating power series, many of the computations can be performed with reduced precision.

2. The AGM with real arguments gives $\ln(x)$ directly. To obtain $\exp(x)$ we need to apply Newton's method. To evaluate trigonometric functions such as $\sin(x)$, $\cos(x)$, $\arctan(x)$ we need to work with complex arguments, which increases the constant hidden in the "O" time bound. Alternatively, we can use Landen transformations for incomplete elliptic integrals, but this gives even larger constants.

3. Because it converges so fast, it is difficult to speed up the AGM. At best we can save $O(1)$ iterations.

4. Newton's method

If we have an algorithm for computing a function $F(y)$, then Newton's method generally allows us to compute the inverse function $G(x)$.

For example, we can compute $F(y) = \ln y$ in time $O(M(d) \log d)$ by one of the AGM algorithms. Using Newton's method, we can compute $G(x) = \exp x$, also in time $O(M(d) \log d)$.

Newton's method increases the constant factor since we have to evaluate F with precision $d, d/2, d/4, \ldots$ (not in this order). Assuming that

$$\sum_{0 \le k \le \lfloor \lg d \rfloor} M(d/2^k) \sim \sum_{0 \le k \le \lfloor \lg d \rfloor} 2^{-k} M(d) \sim 2M(d)\,,$$

the constant is multiplied by two. However, this increase in the constant can be avoided by using higher-order methods.

The Newton iteration

If x is regarded as fixed, then Newton's method applied to solve

$$F(y) - x = 0$$

for y gives the iteration

$$y_{j+1} = y_j - \frac{F(y_j) - x}{F'(y_j)},$$

and under certain conditions this converges quadratically to $y = G(x)$.

If $F(y)$ is an elementary function, then it is easy to calculate the derivative $F'(y)$. For example, if $F(y) = \ln(y)$, then $F'(y) = 1/y$, and the Newton iteration is

$$y_{j+1} = y_j - y_j(\ln(y_j) - x).$$

We can write this as

$$y_{j+1} = y_j(1 + \delta_j)$$

where

$$\delta_j = x - \ln(y_j).$$

Higher order methods

It is easy to get a higher-order method using the addition formula for exp. We have

$$\exp(x) = \exp(\ln(y_j) + \delta_j) = y_j \sum_{k=0}^{\infty} \frac{\delta_j^k}{k!}\,.$$

To get a method of order $r > 1$, truncate the series after r terms, giving an iteration

$$y_{j+1} = y_j \sum_{k=0}^{r-1} \frac{\delta_j^k}{k!}\,.$$

Newton's method is just the case $r = 2$.

Under the same assumptions as before, the r-th order method multiplies the constant by a factor

$$1 + \frac{1}{r} + \frac{1}{r^2} + \cdots = \frac{r}{r-1}\,.$$

This is true for any fixed $r > 1$. If we allow r to increase with d, we need to take the overhead of an iteration into account: essentially it is r extra multiplications.

Getting the best constant

If $\ln(y)$ can be evaluated in time

$$\sim c\ln(d)M(d)\,,$$

the r-th order method evaluates $\exp(x)$ in time

$$\sim \frac{r}{r-1}(r + c\ln(d))M(d)\,.$$

Taking $r \sim \sqrt{c\ln d}$ gives $\exp(x)$ in time

$$\sim c\ln(d)M(d)\left(1 + O\left(\frac{1}{\sqrt{\ln d}}\right)\right)\,.$$

Since $1/\sqrt{\ln d} \to 0$ (slowly!) as $d \to \infty$, we see that $\exp(x)$ can be evaluated in time

$$\sim c\ln(d)M(d)\,,$$

asymptotically the same time as for $\ln(y)$. In practice, though, the overhead of computing an inverse function is significant.

Other inverse functions

Other pairs of elementary functions may be handled in much the same way as the pair $(\exp, \ln)$, since they all satisfy simple addition formulæ.

For example, replacing exp by tan and ln by arctan, we define

$$\delta_j = x - \arctan(y_j)\,,$$

and use

$$\tan(x) = \tan(\arctan(y_j) + \delta_j) = \frac{y_j + \tan(\delta_j)}{1 - y_j \tan(\delta_j)}\,.$$

The first r terms in the Taylor series for $\tan(\delta)$ can be found in $O(r^2)$ operations by inversion of the Taylor series for arctan (this bound can be improved to $O(r \log r)$ but $O(r^2)$ is sufficient).

If we take $r \sim (\ln d)^{1/3}$, we can compute tan in essentially the same time as arctan, since the multiplicative factor

$$1 + O\left(\frac{1}{\ln d}\right)^{1/3} \to 1 \;\text{ as }\; d \to \infty\,.$$

5. Complex arithmetic

In some cases the asymptotically fastest algorithms require the use of complex arithmetic to produce a real result. It would be nice to avoid this because complex arithmetic is significantly slower than real arithmetic.

Examples where we seem to need complex arithmetic to get the asymptotically fastest algorithms are:

1. $\arctan(x)$, $\arcsin(x)$, $\arccos(x)$ via the AGM using

$$\arctan(x) = \Im(\ln(1 + ix))\,.$$

2. $\tan(x)$, $\sin(x)$, $\cos(x)$ using Newton's method and the above, or

$$\cos(x) + i\sin(x) = \exp(ix)\,,$$

where the complex exponential is computed by Newton's method from the complex ln.

The complex AGM

The theory that we outlined for the AGM iteration and AGM algorithms for $\log(z)$ can be extended without problems to complex $z \notin (-\infty, 0]$, provided we always choose the square root with positive real part.

The constants

A complex multiplication takes three real multiplications (using Karatsuba's trick), and a complex squaring takes two real multiplications. Taking this into account, we get the following asymptotic upper bounds.

Operation	real	complex
squaring	$M(d)$	$2M(d)$
multiplication	$M(d)$	$3M(d)$
square root	$4.25M(d)$	$10.25M(d)$
AGM iteration	$5.25M(d)$	$13.25M(d)$
ln via AGM	$10.5\lg(d)M(d)$	$26.5\lg(d)M(d)$

Exercise: Improve the constants.

6. Binary splitting

Since the asymptotically fastest algorithms for arctan, sin, cos etc have a large constant hidden in their time bound $O(M(d)\lg d)$, it is interesting to look for other algorithms that may be competitive for a large range of precisions even if not asymptotically optimal. One such algorithm (or class of algorithms) is based on *binary splitting* or the closely related *FEE method*.

The time complexity of these algorithms is usually

$$O((\log d)^\alpha M(d))$$

for some constant $\alpha > 1$ depending on how fast the relevant power series converges, and perhaps also on the multiplication algorithm (classical/Karatsuba or Schönhage-Strassen).

I won't discuss the rather controversial history of these algorithms except to note that the idea is quite old, e.g. over thirty years ago I gave a binary splitting algorithm for computing $\exp(x)$ with $\alpha = 2$.

The idea

Suppose we want to compute $\arctan(x)$ for rational $x = p/q$, where p and q are small integers and $|x| \le 1/2$. The Taylor series gives

$$\arctan\left(\frac{p}{q}\right) \approx \sum_{0 \le k \le d/2} \frac{(-1)^k p^{2k+1}}{(2k+1)q^{2k+1}}\,.$$

The finite sum, if computed exactly, gives a rational approximation P/Q to $\arctan(p/q)$, and

$$\log|Q| = O(d \log d)\,.$$

(Note: the series for exp converges faster, so in this case $\log|Q| = O(d)$.)

The finite sum can be computed by "divide and conquer": sum the first half to get P_1/Q_1 say, and the second half to get P_2/Q_2, then

$$\frac{P}{Q} = \frac{P_1}{Q_1} + \frac{P_2}{Q_2} = \frac{P_1Q_2 + P_2Q_1}{Q_1Q_2}\,.$$

The rationals P_1/Q_1 and P_2/Q_2 are computed by a recursive application of the same method, hence the term "binary splitting".

Complexity

The overall time complexity is

$$O\left(\sum_k M((d/2^k)\log(d/2^k))\right) = O((\log d)^\alpha M(d)),$$

where $\alpha = 2$ for Schönhage-Strassen multiplication; in general $\alpha \le 2$.

We can save a little by working to precision d rather than $d \log d$ at the top levels; for classical or Karatsuba multiplication this reduces α to 1, but we still have $\alpha = 2$ for Schönhage-Strassen multiplication.

In practice the multiplication algorithm would not be fixed but would depend on the size of the integers being multiplied. The complexity depends on the algorithm that is used at the top levels.

Repeated application of the idea

If $x \in (0, 0.5]$ and we want to compute $\arctan(x)$, we can approximate x by a rational p/q and compute $\arctan(p/q)$ as a first approximation to $\arctan(x)$. Now

$$\tan(\arctan(x) - \arctan(p/q)) = \frac{x - p/q}{1 + px/q}\,,$$

so

$$\arctan(x) = \arctan(p/q) + \arctan(\delta)$$

where

$$\delta = \frac{x - p/q}{1 + px/q} = \frac{qx - p}{q + px}\,.$$

We can apply the same idea to approximate $\arctan(\delta)$, until eventually we get a sufficiently accurate approximation to $\arctan(x)$. Note that $|\delta| \le |x - p/q|$, so it is easy to ensure that the process converges.

Complexity of repeated application

If we use a sequence of about $\lg d$ rationals $p_0/q_0, p_1/q_1, \ldots$, where

$$q_i = 2^{2^i}\,,$$

then the computation of each $\arctan(p_i/q_i)$ takes time $O((\log d)^\alpha M(d))$ and the overall time to compute $\arctan(x)$ is

$$O((\log d)^{\alpha+1} M(d))\,.$$

The exponent $\alpha + 1$ is 2 or 3. Although this is not asymptotically as fast as AGM-based algorithms, the implicit constants for binary splitting are small and the idea is useful for quite large d (at least 10^6 decimal places).

Comparison of arctan algorithms

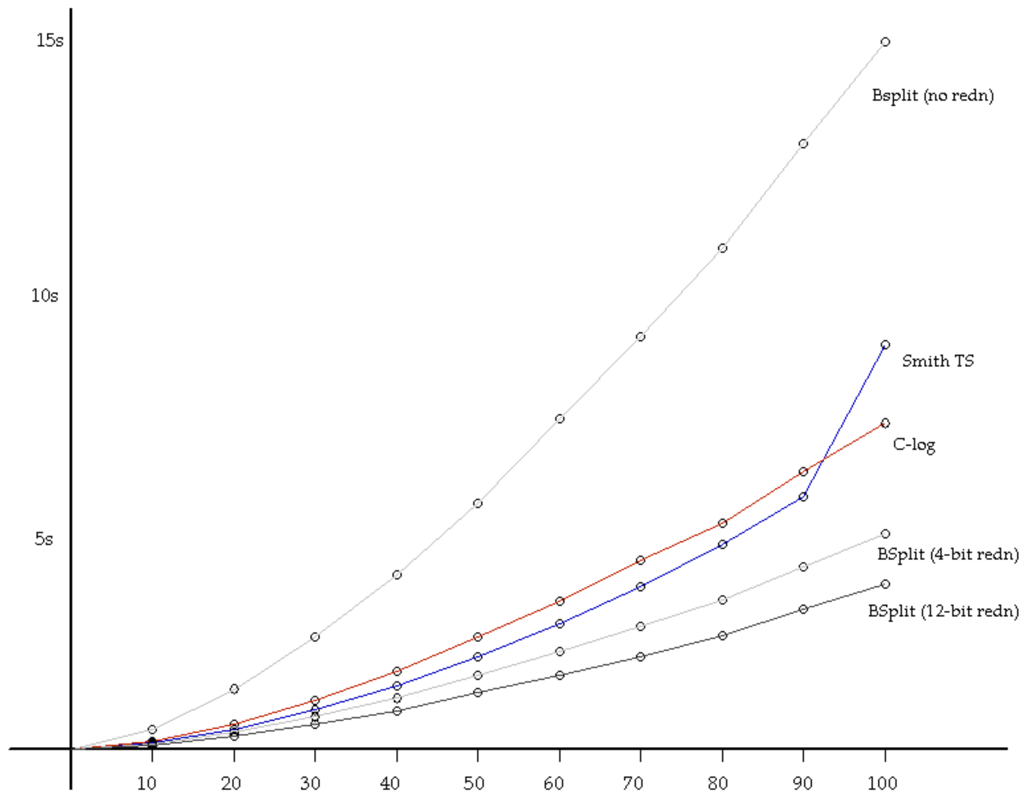

Figure 2. Arctan(x) computation times - 10,000 to 100,000 digits precision

Linear scale to 100,000D (decimal places).

Times are for computing arctan(3/7) (given in binary not rational form) on a 3GHz Pentium 4 using GMP. The multiplication algorithm(s) are chosen by GMP.

Note the crossover for Smith and complex AGM at about 95,000D.

Arctan algorithms to 1,000,000D

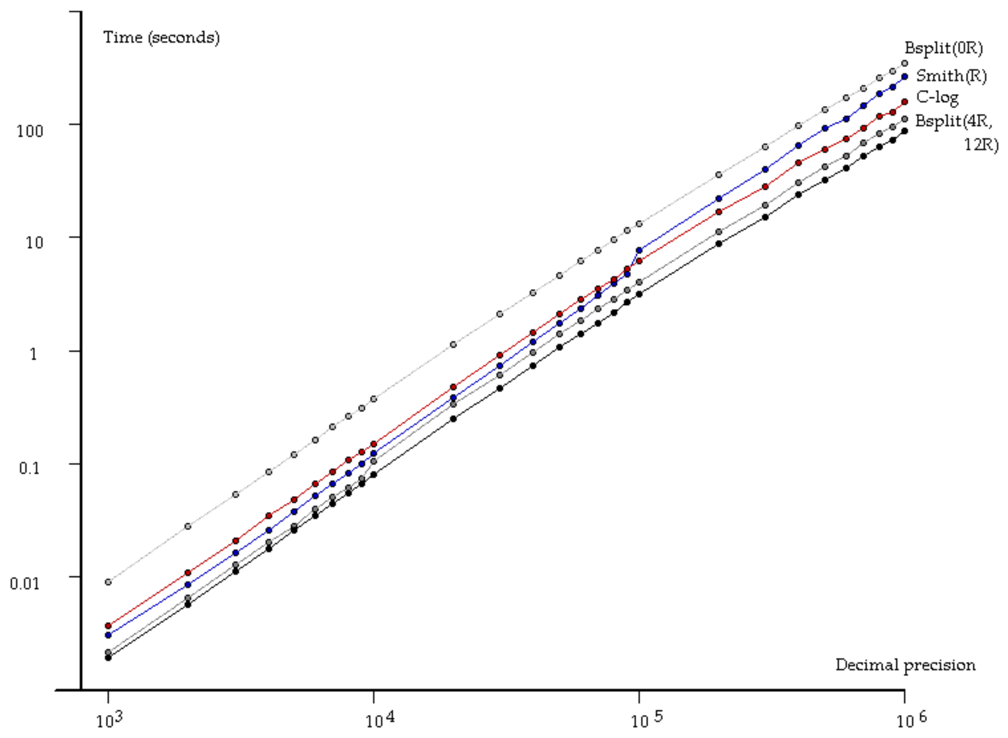

Figure 4. Arctan(x) computation times, log-log plot (all precisions)

Log-log scale to 1,000,000D.

The crossover is the one that we saw on the previous slide (Smith and complex AGM at about 95,000D).

Postscript: The version (4.1.4) of GMP used does not implement sqrt optimally (there is an extra factor of $\log(d)$) which may explain the lack of a crossover for complex AGM and binary splitting.

Arctan algorithms: detail

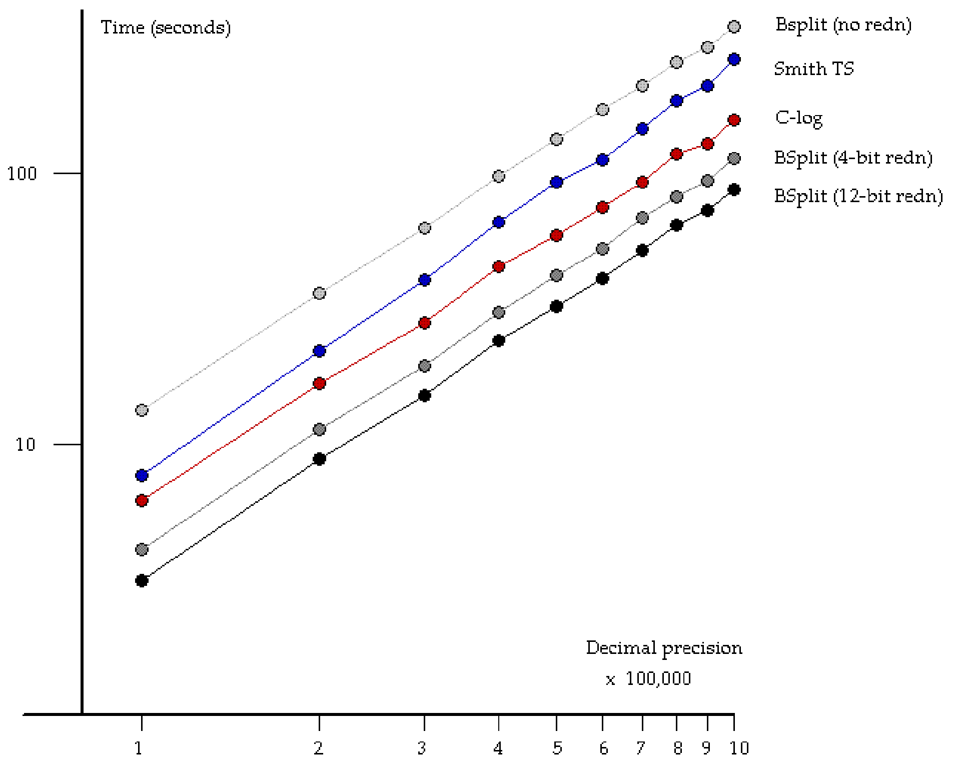

Figure 5. Arctan(x) computation times, log-log plot (detail)

Log-log scale for 100,000D to 1,000,000D.

Binary splitting with 12-bit reduction (12 steps of argument reduction) is well ahead of complex AGM.

Binary splitting with 4-bit reduction is also ahead of complex AGM, though not by so much.

There is no sign of a crossover (see postscript on previous slide).

Generalisations

The idea of binary splitting can be generalised. For example, the Chudnovskys gave a “bit-burst” algorithm which applies to fast evaluation of solutions of linear differential equations. However, that is the topic of another talk!

Acknowledgements

This talk is based on a chapter of a book that Paul Zimmermann and I are writing. Many thanks to Paul for help in debugging my slides.

Thanks also to Jim White (ANU) for many interesting discussions, and for providing the timing comparisons for arctan algorithms.

References

[1] D. H. Bailey, High-precision floating-point arithmetic in scientific computation, *Computing in Science and Engineering*, May–June 2005, 54–61; also report LBNL–57487. Available from `http://crd.lbl.gov/~dhbailey/dhbpapers/`.

[2] D. J. Bernstein, Removing redundancy in high-precision Newton iteration, draft, 2004. `http://cr.yp.to/papers.html#fastnewton`.

[3] J. M. Borwein and P. B. Borwein, *Pi and the AGM*, Monographies et Études de la Société Mathématique du Canada, John Wiley & Sons, Toronto, 1987.

[4] R. P. Brent, Multiple-precision zero-finding methods and the complexity of elementary function evaluation, in *Analytic Computational Complexity* (edited by J. F. Traub), Academic Press, New York, 1975, 151–176. Available from `http://wwwmaths.anu.edu.au/~brent/pub/pub028.html`.

[5] R. P. Brent, The complexity of multiple-precision arithmetic, in *The Complexity of Computational Problem Solving* (edited by R. S. Anderssen and R. P. Brent), University of Queensland Press, Brisbane, 1976, 126–165. Postscript added 1999. Available from `http://wwwmaths.anu.edu.au/~brent/pub/pub032.html`.

[6] R. P. Brent, Fast multiple-precision evaluation of elementary functions, *J. ACM* **23** (1976), 242–251. `http://wwwmaths.anu.edu.au/~brent/pub/pub034.html`.

[7] R. P. Brent and H. T. Kung, Fast algorithms for manipulating formal power series, *J. ACM* **25** (1978), 581–595. `http://wwwmaths.anu.edu.au/~brent/pub/pub045.html`.

[8] R. P. Brent and P. Zimmermann, *Modern Computer Arithmetic*, book in preparation, 2006.

[9] D. V. Chudnovsky and G. V. Chudnovsky, Computer algebra in the service of mathematical physics and number theory, in *Computers in Mathematics* (edited by D. V. Chudnovsky and R. D. Jenks), Lecture Notes in Pure and Applied Mathematics, Vol. 125, Marcel Dekker, New York, 1990, 109–232.

[10] X. Gourdon and P. Sebah, Numbers, constants and computation: binary splitting method, `http://numbers.computation.free.fr/Constants/Algorithms/splitting.html`.

[11] A. Karatsuba and Y. Ofman, Multiplication of multidigit numbers on automata (in Russian), *Doklady Akad. Nauk SSSR* **145** (1962), 293–294. English translation in *Sov. Phys. Dokl.* **7** (1963), 595–596.

[12] E. A. Karatsuba, Fast evaluation of transcendental functions (in Russian), *Probl. Peredachi Inf.* **27**, 4 (1991), 87–110. English translation in *Problems of Information Transmission* **27** (1991), 339–360. See also `http://www.ccas.ru/personal/karatsuba/faqen.htm`.

[13] M. S. Paterson and L. J. Stockmeyer, On the number of nonscalar multiplications necessary to evaluate polynomials, *SIAM J. Computing* **2** (1973), 60–66.

[14] G. Robin, Grandes valeurs de la fonction somme des diviseurs et hypothèse de Riemann, *J. Math. Pures Appl.* **63** (1984), 187–213.

[15] T. Sasaki and Y. Kanada, Practically fast multiple-precision evaluation of $\log(x)$, *J. Inf. Process.* **5** (1982), 247–250. See also [3, §7.2].

[16] A. Schönhage and V. Strassen, Schnelle Multiplikation Grosser Zahlen, *Computing* **7** (1971), 281–292.

[17] D. M. Smith, Efficient multiple-precision evaluation of elementary functions, *Math. Comp.* **52** (1989), 131–134.