# Computation of the error functions erf & erfc in arbitrary precision with correct rounding

Sylvain Chevillard, Nathalie Revol

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *Computation of the error functions erf & erfc in arbitrary precision with correct rounding*

S. Chevillard — N. Revol

**N° ????**

February 2008

Thème SYM


ISSN 0249-6399 ISRN INRIA/RR--????--FR+ENG

# Computation of the error functions erf & erfc in arbitrary precision with correct rounding

S. Chevillard* †, N. Revol‡ †

Thème SYM — Systèmes symboliques
Projet Arenaire

Rapport de recherche n° ???? — February 2008 — 14 pages

**Abstract:** In this paper, the computation of erf(x) in arbitrary precision is detailed. A feature of our implementation is correct rounding: the returned result is the exact result (as if it were computed with infinite precision) rounded according to the specified rounding mode. The algorithm that computes the correctly rounded value of erf(x) for any argument x is detailed in this paper. In particular, the choice of the approximation formula, the determination of the truncation rank and of the computing precision are presented. When the current truncation rank and computing precision do not suffice to determine the correctly rounded value, they are increased and the computation is done again. The optimal strategy to adapt them is given in this paper. Finally, timings on some experiments are given, and the implementation of the complementary error function erfc is then sketched.

**Key-words:** error function, complementary error function, floating-point arithmetic, arbitrary precision, adaptation of the computing precision, correct rounding

* Sylvain.Chevillard@ens-lyon.fr http://perso.ens-lyon.fr/sylvain.chevillard/
† Université de Lyon, LIP (CNRS-ENSL-INRIA-UCBL), École Normale Supérieure de Lyon, 46 allée d'Italie, 69007 Lyon, France
‡ INRIA, Nathalie.Revol@ens-lyon.fr http://perso.ens-lyon.fr/nathalie.revol/

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

# Évaluation des fonctions d'erreur erf et erfc en précision arbitraire avec arrondi correct

**Résumé :** Dans cet article, nous détaillons comment évaluer la fonction erf en arithmétique flottante à précision arbitraire. Une spécificité de notre implantation est qu'elle retourne le résultat avec arrondi correct : le résultat est égal au résultat exact (le résultat qui serait obtenu si on disposait d'une précision infinie) puis arrondi selon le mode d'arrondi demandé. Nous présentons un algorithme qui évalue, avec arrondi correct, erf x pour tout argument x. En particulier, nous détaillons le choix de la formule d'approximation, la détermination du rang de troncature et de la précision utile pour mener les calculs. Si jamais le rang de troncature et la précision de calcul s'avèrent insuffisants pour permettre d'arrondir correctement le résultat calculé, leurs valeurs sont accrues et le calcul est effectué à nouveau. Nous détaillons également la stratégie optimale pour choisir les nouvelles valeurs de ces paramètres. Nous terminons par des mesures de temps pour quelques expérimentations, puis nous esquissons l'algorithme choisi pour évaluer la fonction complémentaire d'erreur erfc.

**Mots-clés :** fonction d'erreur, fonction complémentaire d, arithmétique à virgule flottante, précision arbitraire, arrondi correct, adaptation de la précision de calcul

# 1 Introduction

The goal of this work is to compute the error functions erf and erfc using arbitrary precision floating-point arithmetic and to deliver the correctly rounded result. Correct rounding means that the returned result is the exact result (as if it were computed with infinite precision) rounded according to the specified rounding mode.

The error functions are widely used in statistics: the probability that a Gaussian random variable $X$ takes values between $-a$ and $a$ is given by $\mathrm{erf}\left((a-m)/(\sqrt{2}\sigma)\right)$, where $m$ is the mean and $\sigma$ is the standard deviation of the distribution for $X$. The error function also appears in the solution of the diffusion equation in specific configurations and in other heat transfer problems. For more information on the error function and the complementary error function, see for instance [13, 18].

Various implementations of the error functions can be found [15], but none returns the correctly rounded result. Most use the standard floating-point single and double precisions defined by the IEEE-754 standard for floating-point arithmetic [10]. Some implementations [6, 7] use a best minimax polynomial approximations of these functions, as determined in [5] for instance, others use a truncated expansion with a fixed number of terms. Even if the quality of the approximation can be quantified (by an upper bound on the error), no implementation guarantees the correct rounding of the result.
Much less implementations are available for arbitrary precision. Maple and Mathematica evaluate the error functions in arbitrary precision. However, they suffer from two problems. Firstly, they do not guarantee the correct rounding of the result. Secondly, it is not even clear what the accuracy of the returned result is: in Maple for instance, the computing precision is (apparently heuristically) chosen and the recurrence is performed using this precision, until the iterates stagnate. The MPFUN package [2] is better specified [3]: if $2^{-n}$ is the required accuracy, the (relative or absolute) error between the computed result and the exact value is $O(2^{-n})$.
The goal of the DLMF project (*Digital Library of Mathematical Functions*, cf. `http://dlmf.nist.gov/`) is to develop a replacement of Abramowitz and Stegun's *Handbook of Mathematical Functions* [1]. Unfortunately, it will not provide an implementation for the evaluation of special functions. The software accompanying the book [8] is not yet available.
The MPFR library (*Multiple Precision Floating-point Reliable library*, [17]) is a library for arbitrary precision floating-point arithmetic with correct rounding. It includes several special functions (Gamma, Zeta). Simultaneous with our work, the error function has been implemented in MPFR as `mpfr_erf`. We will compare the initial implementation and the improved implementation of MPFR with ours in Section 4.

We use the MPFR library for our implementation: we use the arbitrary precision arithmetic and algebraic operations and the $\pi$ constant. We also use intensively a peculiar function in MPFR: `mpfr_can_round`. This function takes as arguments a floating-point approximation $b$ of an unknown value $x$, a bound on the error $|b-x|$ and a destination

precision $p$; it indicates whether rounding $b$ in precision $p$ yields the same result as rounding the exact value $x$. Fig. 1 illustrates this: rounding towards $-\infty$ is assumed, tall vertical bars correspond to floating-point numbers represented with precision $p$, small vertical bars correspond to floating-point numbers represented with the precision of $b$. On the left of Fig. 1, rounding $b$ or $x$ both give $y$; on the right, rounding $b$ yields $z_1$ whereas it is not known whether $x$ should be rounded to $z_1$ or $z_2$.

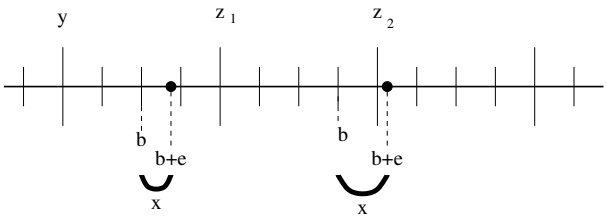


Figure 1: Illustration of `mpfr_can_round`. The unknown $x$ belongs to $[b, b+e]$.

The scheme of an algorithm that returns the correctly rounded evaluation of a function $f$ on $x$ with precision $p$ is thus the following:

1. approximate $f(x)$ with computing precision $q$ (with $q$ larger than $p$) and with error $\leq \varepsilon$;
2. if `mpfr_can_round`$(f(x), \varepsilon, p)$ then return the correctly rounded value of $f(x)$ in precision $p$;
3. otherwise increase $q$, decrease $\varepsilon$ and try again.

The error function erf and the complementary error function erfc are introduced in the next Section. Some formulae and properties are given; they are used either to derive an algorithm for the evaluation of erf and erfc or to simplify some cases. Then our algorithm to evaluate erf and to return the correctly rounded result is presented in Section 3. The choice of the implemented formula is discussed, then the determination of the truncation rank and of the computing precision are explained. When they are not sufficient to enable to round correctly the computed result, they are increased. The strategy which is optimal in term of the time overhead is devised in Section 3.7. Our implementation is compared to Maple and to the `mpfr_erf` function of MPFR on some typical examples, in Section 4: our pre-computations are costly for small precisions, but yield a significant gain in computing time for larger ones. In Section 5, the algorithm for the complementary error function erfc is sketched. Finally, a list of desirable improvements is given in Conclusion.

# 2 The erf and erfc functions

## 2.1 Definitions

The error function is defined as:

$$\operatorname{erf} : x \mapsto \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt = \frac{1}{\sqrt{\pi}} \int_{-x}^x e^{-t^2} dt,$$

and the complementary error function is defined by

$$\operatorname{erfc} x = 1 - \operatorname{erf} x = \frac{2}{\sqrt{\pi}} \int_x^{+\infty} e^{-t^2} dt.$$

The normalisation factor $2/\sqrt{\pi}$ ensures that erf defines a probability density function.

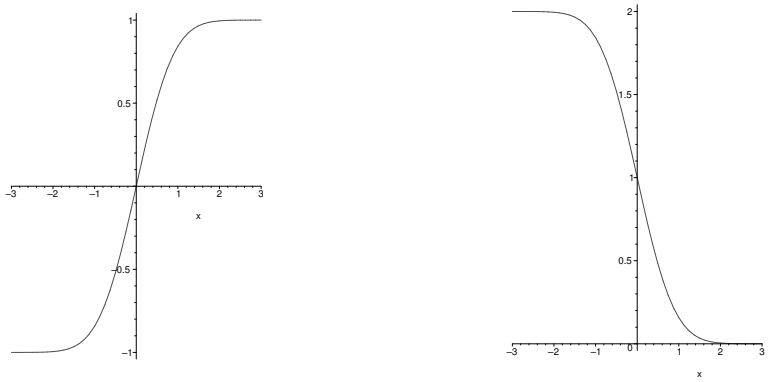


Figure 2: Graph of the erf and erfc functions.

## 2.2 Properties

The erf function is odd: this enables us to consider only nonnegative arguments.

The value of erf in 0 is $\operatorname{erf}(0) = 0$. Its limit when $x$ tends to $+\infty$ is 1.

Unfortunately, no properties similar to $\sin(x + 2\pi) = \sin x$ or $\log(x^2) = 2 \log x$ hold for the error functions. Such properties prove extremely useful for the evaluation of elementary functions. They allow to reduce the domain to a small interval. Such *range reduction* is not possible for the erf and erfc functions.

### 2.3 Approximation formulas

Among the possible formulas that can be used to approximate the erf and erfc functions, the book [1, pp. 297-298] gives the following ones:

- integral representation

$$\operatorname{erf} x = \tfrac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \qquad \operatorname{erfc} x = \tfrac{2}{\sqrt{\pi}} \int_x^{+\infty} e^{-t^2} dt. \tag{1}$$

- series expansion for erf:

$$\operatorname{erf} x = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{+\infty} \frac{(-1)^n x^{2n+1}}{n!(2n+1)} \tag{2}$$

$$= \frac{2}{\sqrt{\pi}} e^{-x^2} \sum_{n=0}^{+\infty} \frac{2^n x^{2n+1}}{1 \times 3 \times \cdots \times (2n+1)} \tag{3}$$

- asymptotic expansion for erfc:

$$\operatorname{erfc} x = \frac{e^{-x^2}}{x\sqrt{\pi}} \left( 1 + \sum_{n=1}^{+\infty} (-1)^n \frac{1 \times 3 \times \cdots \times (2n-1)}{(2x^2)^n} \right) \tag{4}$$

- continued fraction for erfc:

$$\operatorname{erfc} x = \frac{e^{-x^2}}{\sqrt{\pi}} \left( \frac{1}{x+} \; \frac{1/2}{x+} \; \frac{2/2}{x+} \; \frac{3/2}{x+} \; \ldots \right) \tag{5}$$

- enclosure for erfc:

$$\frac{2e^{-x^2}}{\sqrt{\pi}(x + \sqrt{x^2+2})} \leq \operatorname{erfc} x \leq \frac{2e^{-x^2}}{\sqrt{\pi}\left(x + \sqrt{x^2 + \frac{4}{\pi}}\right)} \tag{6}$$

The last inequality will reveal useful to handle separately and rapidly the case of large values of $x$, cf. Section 3.6.

## 3 Algorithm for the erf function

Several approaches could be considered to evaluate $\operatorname{erf} x$ where $x$ is a given floating-point number in arbitrary precision. For instance, a numerical quadrature could be performed [9], using the integral representation (1). This implies a large number of evaluation of the exponential function, which is costly: each evaluation implies the evaluation of a sum of the kind given in the power series (2). Either this sum is evaluated with correct rounding (as in

`mpfr_exp`) and such an accurate evaluation is costly, or it is approximated less accurately but then the corresponding error must be carefully taken into account in the total error bound.

Determining the best minimax polynomial approximation is not considered when the precision is variable. When the precision is fixed (single or double IEEE-754 precision), then this polynomial has to be determined only once, but when the precision varies this polynomial has to be recomputed for each precision and this computation is costly. Furthermore, the determination of the approximating polynomial involves the evaluation of the approximated function.

Another solution is to use a Taylor expansion, such as those given in equations (2) or (3). An advantage of using equation (2) is that it involves only rational calculations, whereas equation (3) requires one evaluation of the exponential function, which is costly. A second advantage of equation (2) is that it is an alternate series and that the remainder of a truncated alternate series is extremely easy to bound: the first neglected term directly provides a bound. On the contrary, the truncation error is not obvious to derive for equation (3). Furthermore, this first neglected term tends rapidly to 0 as $n$ tends to $\infty$, at least for small values of $x$. However, this term of order $n$ can be large when $x$ is large, at least for relatively small values of $n$. This means that to get a small enough denominator that compensates for the large numerator, it may thus be necessary to truncate after a large number $N$ of terms. Another main drawback of using equation (2) is that summing an alternate series usually implies having cancellation. On the opposite, the other power series (3) should exhibit no problem of numerical stability, since only positive terms are added. We nevertheless decided upon using equation (2) and we will show how to circumvent this difficulty in Section 3.3.

For the time being, we focus on the direct evaluation of erf and we will discuss the use of erfc and of formulae (4) and (5) in Section 5.

Several problems must then be solved, in order to derive our algorithm:

- at which number $N$ of terms should we truncate?
- what is an upper bound on the remainder, *i.e.* on the error due to the truncation?
- in which order should the sum be computed in order to minimise the effect of the roundoff errors?
- what is an upper bound on these roundoff errors?
- what computing precision should be used?
- how can the computed result be correctly rounded?

In the following, these questions will receive an answer.

### 3.1 Truncation order and truncation error

Let us denote by $a_n(x) = \left((-1)^n x^{2n+1}\right) / \left(n!(2n+1)\right)$ and by $S_N(x)$ the truncated sum $\sum_{n=0}^{N} a_n(x)$. The term $|a_n(x)|$ is either decreasing with $n$ if $x$ is small enough, or it begins by increasing, starting from a value larger than 1 and then it decreases. The proof of this result can be found in [4]. Since the series is an alternating series with terms decreasing to 0 for $n$ large enough, if $N$ is such that $|a_{N+1}(x)| < 1$, it is well known that the truncation error satisfies

$$\left|\operatorname{erf} x - \frac{2}{\sqrt{\pi}} S_N(x)\right| \leq \frac{2}{\sqrt{\pi}} |a_{N+1}(x)|.$$

It is also known that the sign of $\operatorname{erf} x - 2/\sqrt{\pi} S_N(x)$ alternates: $2/\sqrt{\pi} S_N(x) \geq \operatorname{erf} x$ for even $N$ and $2/\sqrt{\pi} S_N(x) \leq \operatorname{erf} x$ for odd $N$.

Firstly, we compute $2/\sqrt{\pi}(x - x^3/3)$ which is a rough approximation from below of $\operatorname{erf} x$: we use this value to get an estimation $e$ (from below) of the exponent of the floating-point representation of $\operatorname{erf} x$.

If the target precision for the floating-point approximation of $\operatorname{erf} x$ is $p$ bits, we truncate the expansion series for $\operatorname{erf} x$ at an order $N$ that satisfies both $|a_{N+1}(x)| < 1$ and $2/\sqrt{\pi}|a_{N+1}(x)| \leq 2^{e-p-8}$: when $|a_{N+1}(x)| < 1$, then the series becomes an alternate series with decreasing terms in absolute value. The 8 extra bits are for safety, they are expected to absorb the roundoff errors implied by the floating-point summation and to enable to round correctly the computed result.

To determine such an order $N$, we simply compute the consecutive $|a_k(x)|$, $k = 0, 1, \cdots$ until $|a_k(x)| < \min(1, \sqrt{\pi}/2 \cdot 2^{e-p-8})$, using a small computing precision. This small precision is simply the IEEE-754 double precision, to spare computing time.

### 3.2 Upper bound on the roundoff error of the summands

Using Higham's notation [11, Chapter 3] and intermediate lemmas [4], we obtained the following bound.

**Theorem**

If $5\lfloor N/2 \rfloor u < 1$, with $u = \operatorname{ulp}(1) = 1^+ - 1$ where $1^+$ is the smallest floating-point number strictly larger than 1 (using the current computing precision), then the absolute error between $2/\sqrt{\pi} S_N(x)$ and its computed value $2/\sqrt{\pi}\widehat{S_N(x)}$ satisfies

$$\left|\frac{2}{\sqrt{\pi}} S_N(x) - \frac{2}{\sqrt{\pi}}\widehat{S_N(x)}\right| \leq \frac{2}{\sqrt{\pi}} \frac{e^{x^2} - 1}{x} \gamma_{5\frac{N}{2}} \tag{7}$$

where $\gamma_k = (ku)/(1 - ku)$.

### 3.3 Summation order

In [11, Chapter 4], N. Higham devotes a complete chapter to the summation problem. The question is: in which order should $k$ summands $s_1, \ldots, s_k$ be added in floating-point arithmetic to minimise the roundoff error? When all summands are nonnegative, the recommended order is the increasing order. In our case, summands are of alternating signs. To reduce our problem to the sum of nonnegative terms, we group our summands by pairs of two consecutive terms: if $N$ is odd,

$$\begin{aligned} S_N(x) &= \sum_{n=0}^{N} a_n(x) = \sum_{k=0}^{\lfloor N/2 \rfloor - 1} \left(a_{2k}(x) + a_{2k+1}(x)\right) \\ &= x \sum_{k=0}^{\lfloor N/2 \rfloor - 1} \frac{x^{4k}}{(2k)!} \left(\frac{1}{4k+1} - \frac{x^2}{(2k+1)\times(4k+1)}\right). \end{aligned} \tag{8}$$

Depending on the value of $x$, the first terms may not be nonnegative, but the last terms are positive and are decreasing.

Our summation algorithm proceeds with $k$ decreasing from $\lfloor N/2 \rfloor - 1$ down to 0. It uses Horner's scheme.

### 3.4 Choice of the computing precision

In order to get a result with $p$ correct bits, one has to perform intermediate computations with $p' \geq p$ bits of precision.

We choose to have a roundoff error of the same order $\varepsilon$ as the truncation error, *i.e.* $\varepsilon = 2^{e-p-8}$ where $p$ is the target precision. Replacing $\varepsilon$ by this value and $u$ by $2^{1-p'}$ in (7) enables us to deduce the required precision $p'$ for the intermediate computations.

### 3.5 Rounding the computed result

We have derived an a priori bound on the error between the computed value $\hat{y}$ and the exact value $y = \operatorname{erf} x$: this bound is $2\varepsilon$.

It is possible to refine this a priori bound by estimating the error, during the computations. The roundoff error given in (7) is overestimated: in particular (7) does not take into account the order of the summation. We adapted the computation of running error bound explained in [11, Chapter 5] for Horner's scheme to our problem, cf. [4].

We can then invoke the `mpfr_can_round` function with the computed value $\hat{y}$ that approximates $y = \operatorname{erf} x$, this new error bound and the target precision $p$ to determine if rounding $\hat{y}$ in the required direction and in precision $p$ is equivalent to rounding $y = \operatorname{erf} x$. If the answer is negative, we restart this process with a larger truncation order $N' > N$ and a larger computing precision $p'' > p'$. The optimal choice of $N'$ and $p''$ is detailed in Section

3.7.

One remaining question is whether this process will always terminate. It is more generally known as the Table Maker's Dilemma [14]. To our knowledge, there exists no theorem stating that our process will eventually stop, *i.e.* stating that there is no floating-point value $x$ and no precision $t$ such that $\operatorname{erf} x$ is exactly a floating-point value using this precision $t$.

## 3.6 How to avoid costly computations when $x$ is large

When $x$ is large, $\operatorname{erf} x$ is very close to 1. The enclosure (6) for $\operatorname{erfc} x$ can be applied to produce an enclosure for $\operatorname{erf} x = 1 - \operatorname{erfc} x$. We compute the left hand side and the right hand side of (6): if this enclosure is tight enough, one of these two values may be used as an approximation of $\operatorname{erf} x$. If the enclosure is not tight enough or if it is not possible to get the correct rounding of $\operatorname{erf} x$ from these values, then the computation based on the expansion is launched. In practice, the enclosure is considered as "tight enough" if the difference between the two bounds is less or equal to $u$.

## 3.7 Optimal adaptation of the computing precision

Kreinovich and Rump [12] established an optimal strategy to automatically adapt the computing precision to the computational needs. One of their assumptions is that the computation can detect that the current computing precision does not suffice. This is the case of our procedure, since `mpfr_can_round` reports a failure precisely when the accuracy of the computed result does not suffice to permit to round it correctly. The second assumption in [12] is that the computation is restarted from scratch when the computing precision is increased. This also applies to our computation, since we did not try to develop sophisticated procedures to correct the previous result. Indeed, we clear all our previous intermediate results and restart the summation: each term of the sum is re-computed with the new, higher precision, starting from a larger index and summing down the indices.

The strategy to adapt the computing precision thus consists in choosing a new precision (and, in our case, a new truncation index) such that the time of the computation using the new precision is twice the time of the previous computation. In [12], it is proven that this strategy yields the smallest overhead factor, which is 4, compared to the computational time when the best computing precision were known in advance and used.

In our case, we have to decide how to modify the truncation order and the computing precision, in order to multiply the computational time by 2. First, we have to establish the computing time of our method: it is proportional to $N$, the number of terms of the series (since a Horner scheme is used, cf. formula (8), a constant number of arithmetic operations is performed for the summation of each term), and to $M(t)$ the time of a multiplication or division of two numbers of $t$ bits. In MPFR, this time is quadratic, in theory the best known

Table 1: Comparison of Maple, MPFR and our implementation for some typical examples and various precisions.

| $x$ | $p$ | Maple 6 | Maple 10 | mpfr_erf 2.1.1 | mpfr_erf 2.3.0 | our erf |
|---|---|---|---|---|---|---|
| 0.25 | 100 | - | - | 0.02 ms | 0.03 ms | 0.06 ms |
| 0.25 | 1 000 | 20 ms | - | 0.75 ms | 0.85 ms | 0.43 ms |
| 0.25 | 10 000 | 580 ms | - | 102 ms | 72 ms | 50 ms |
| 0.25 | 100 000 | | 1.1ms | | 44 600 ms | 11 480 ms |
| $\pi$ | 100 | - | 0.013 ms | 0.73 ms | 0.010 ms | 0.015 ms |
| $\pi$ | 1 000 | 60 ms | 0.017 ms | 1.9 ms | 2.3 ms | 1.1 ms |
| $\pi$ | 10 000 | 7 320 ms | 0.228 ms | 204 ms | 270 ms | 104 ms |
| $\pi$ | 100 000 | 1 692 000 ms | 270 ms | 340 700 ms | 45 800 ms | 19 900 ms |
| 100 | 1 000 | - | 0.01 ms | 0.002 ms | - | - |
| 100 | 10 000 | - | 0.01 ms | 0.007 ms | - | - |
| 100 | 100 000 | | 1 110 000 ms | | 38 300 ms | 158 000 ms |
| 100 | 14 427 | 20 ms | - | 191 200 ms | 4 200 ms | - |
| 100 | 14 449 | 20 ms | - | 191 200 ms | 4 200 ms | 10 300m s |

complexity is $\mathcal{O}(t \log t \log\log t)$. Second, let us recall that our method shares the total error into two equal parts, one half for the truncation error and one half for the roundoff error. This additional constraint enables us to decide how to increase $N$ and $t$. With a quadratic multiplication/division time, the computing precision $t$ has to be changed to $t' = 2^{2/3}t$ and the truncation order becomes $N' = 2^{1/3}N$. This is what has been used for the experiments presented below. With a superlinear complexity for $M(t)$, one would get $t' = \sqrt{2}t$ and $N' = \sqrt{2}N$.

# 4 Experimental results

We compare the computing times of Maple, MPFR `mpfr_erf` and our implementation of erf on some typical values of $x$ (small, medium and large) and various precisions. Results are given in Table 1. In Table 1, the symbol $-$ means that the displayed computing time was 0, ie. meaningless. Our experiments were conducted on a Intel Xeon, 2.66GHz, 512KB cache, with Maple 6, Maple 10, gcc 4.1.2, GMP 4.2.1, MPFR 2.1.1 and MPFR 2.3.0. The improvement between the two versions of MPFR resides in the insertion of preliminary tests. These tests check whether an approximation based on very few terms of the expansion, such as three, suffices to get the correctly rounded result. They also help in choosing between the Taylor expansion, for an argument $x$ close to 0, and the asymptotic expansion for large arguments $x$.

For small precisions, our implementation is the costliest one. This is explained by the relatively large amount of time spent to determine the truncation order $N$. The difference in times between the earlier version of MPFR and the more recent one also lies in the time to perform the various tests mentioned above.

For larger precisions, as long as $x$ remains small, our implementation is the most efficient one: indeed, determining a small value for $N$ yields a small amount of computation during the summation phase. Since the precision is large, computing and summing extra terms is costly.

For large values of $x$ and relatively small precisions, $\operatorname{erf} x$ is so close to 1 that the answer can be given almost instantaneously. Our implementation is slower than the other ones because its computations may be somewhat too involved. However, one can see that for $x = 100$ and for precisions between 14 427 and 14 448, our implementation still concludes that computing the expansion series is useless, whereas MPFR 2.1.1 or 2.3.0 launches this costly computation. However, MPFR 2.3.0 is faster than MPFR 2.1.1 and our implementation: it uses an asymptotic expansion, which requires less computation. Maple is always faster: it computes $\operatorname{erfc} x$ very quickly to conclude.

# 5 Algorithm for the erfc function

Since $\operatorname{erfc} x = 1 - \operatorname{erf} x$, one could compute erfc from erf. However, for large values of $x$, $\operatorname{erf} x \simeq 1$: its binary representation is $0.1\cdots1\alpha_1\alpha_2\alpha_3\cdots$ with a large number $\beta$ of leading 1s. The binary representation of $\operatorname{erfc} x$ is $0.\widetilde{\alpha_1}\widetilde{\alpha_2}\widetilde{\alpha_3}\cdots\times 2^{-\beta}$, where the $\widetilde{\alpha_i}$s depend directly from the $\alpha_j$s: this means that the information relevant for $\operatorname{erfc} x$ can be obtained from $\operatorname{erf} x$ only if $\operatorname{erf} x$ is computed with a precision large enough to determine the leading 1s and then the other bits $\alpha_1\alpha_2\alpha_3\cdots$. It is thus cheaper to compute $\operatorname{erfc} x$ directly. As this is shown by the computational times of Maple or MPFR 2.3.0, it is also less costly to compute first $\operatorname{erfc} x$ for large $x$ and to deduce $\operatorname{erf} x$ from it.

Again, no range reduction applies to reduce the domain for the evaluation of erfc. As for erf, various formulae at hand were expansions, either Taylor or asymptotic, and continued fraction. Because the truncation error for the expansion series was more difficult to bound from above than in the case of erf, we chose the continued fraction approach for erfc. Our approach is similar to the approach for the erf function, cf. [4]:

- determination of the truncation order and of the truncation error: this required an adaptation of results known for general continued fractions to our case.
- determination of the roundoff error.

- choice of the method to evaluate the convergents of the continued fraction. A first method consists in computing separately (using two recurrence formulas) the numerators and the denominators of the convergents. A second method consists in working with the simplified rational fraction. With the first method, divisions are avoided but the values of the numerators and denominators can become large.
- determination of the computing precision: we followed the same approach as for erf.
- correct rounding of the result: again, we used `mpfr_can_round` to determine whether the computed result can be rounded, and we restart the computation with an increased computing precision if needed.

# 6 Conclusion and improvements

This work is a first step towards the evaluation of the error functions in arbitrary precisions with correct rounding. Some improvements could yield a better efficiency.

A first improvement lies in a joint use of erf and erfc, depending on small or large arguments. It is preferable to evaluate $\operatorname{erf} x$ when $x$ is small and possibly to compute $\operatorname{erfc} x$ as $1 - \operatorname{erf} x$, since the relevant information lies in $\operatorname{erf} x$. Conversely, it is preferable to evaluate $\operatorname{erfc} x$ when $x$ is large. The computing precision, and consequently the computing time, would be reduced.

The problem of infinite loop of the algorithm should also be handled. Any theoretical result on the Table Maker's Dilemma for the error functions is welcome. If it were known that this process halts, even with a huge (theoretical) computing precision, then this problem would vanish.

Finally, overflows and underflows occurring during intermediate computations may happen: they should be handled with more care than in our current implementation.

Solving these problems involves a theoretical but also experimental study: thresholds between "small" and "large" precisions must be determined, as well as thresholds between "small" and "'large" arguments.

# References

[1] M. Abramowitz M. and I. A. Stegun eds., *Handbook of Mathematical Functions*, Dover, New York, 1965, 1972, Originally published by National Bureau of Standards in 1964.

[2] R.P. Brent, *A Fortran Multiple-Precision Arithmetic Package*, ACM Transactions on Mathematical Software, vol 4, 1978, pp 57-70.

[3] R.P. Brent, *Unrestricted algorithms for elementary and special functions*, IFIP (Information Processing) 80, North-Holland, Amsterdam, 1980, pp 613-619.

[4] S. Chevillard, *Calcul de la fonction erf en précision arbitraire*, Master's thesis, ENS Lyon, 2003. `http://enslyon.free.fr/rapports/info/Sylvain_Chevillard_1.ps.gz`

[5] W.J. Cody, *Rational Chebyshev Approximations for the Error Function*, Mathematics of Computation, vol 23, 1969, pp 631–637.

[6] W.J. Cody, *Performance evaluation of programs for the error and complementary error functions*, ACM Transactions on Mathematical Software, vol 16, 1990, pp 29–37.

[7] W.J. Cody, *Algorithm 715: SPECFUN - A Portable FORTRAN Package of Special Function Routines and Test Drivers*, ACM Transactions on Mathematical Software, vol 19, no 1, 1993, pp 22–32.

[8] A. Cuyt, V. Petersen, B. Verdonk, H. Waadeland, W.B. Jones and C. Bonan-Hamada, *Handbook of continued fractions for special functions*, Kluwer Academic Publishers, Dordrecht, in preparation.

[9] A. Gil, J. Segura and N. Temme, *Computing special functions by using quadrature rules*, Numerical Algorithms, vol 33, 2003, pp 265–275.

[10] American National Standards Institute and Institute of Electrical and Electronic Engineers, *IEEE Standard for Binary Floating-Point Arithmetic*, *ANSI/IEEE Standard, Std 754-1985,* New York, 1985.

[11] N. Higham, *Accuracy and stability of numerical algorithms (2nd edition)*, SIAM, 2002.

[12] V. Kreinovich and S. Rump, *Optimal adaptation of the computing precision*, Reliable Computing, vol 12, no 5, 2006, pp 365–369.

[13] N. Lebedev, *Special Functions and their Applications*, Prentice Hall, 1965.

[14] V. Lefèvre, J.-M. Muller and A. Tisserand, *The Table Maker's Dilemma*, IEEE Transactions on Computers, vol 47, no 11, 1998, pp 1235–1243.

[15] D. Lozier and F. Olver, *Numerical Evaluation of Special Functions*, in W. Gautschi ed., *Mathematics of Computation 1943-1993: A Half-Century of Computational Mathematics*, Proc. of Symposia in Applied Mathematics 48, AMS, 1994, pp 79–125. Updated version available on: `http://math.nist.gov/nesf`.

[16] D. Lozier, *A Proposed Software Test Service for Special Functions*, National Institute of Standards and Technology, Report NISTIR 5916, 1996, `http://math.nist.gov/~DLozier/publications/nistir5916.ps`

[17] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier and P. Zimmermann, *MPFR: A Multiple-Precision Binary Floating-Point Library With Correct Rounding*, ACM Transactions on Mathematical Software, vol 33, no 2, 2006, article 13. `http://www.mpfr.org/`

[18] N. Temme, *Special Functions*, John Wiley and Sons, 1996.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399